AlphaPy Documentation

Release 2.5.0

Mark Conway, Robert D. Scott II

Aug 29, 2020

INTRODUCTION

CHAPTER

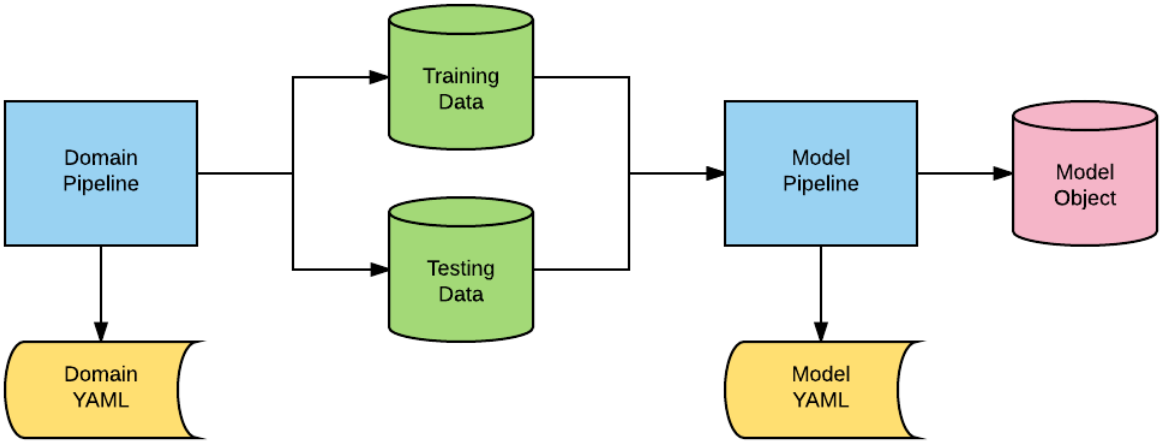
ONE

INTRODUCTION

AlphaPy is a machine learning framework for both speculators and data scientists. It is written in Python with the `scikit-learn` and `pandas` libraries, as well as many other helpful libraries for feature engineering and visualization. Here are just some of the things you can do with AlphaPy:

- Run machine learning models using `scikit-learn` and `xgboost`.
- Create models for analyzing the markets with *MarketFlow*.
- Predict sporting events with *SportFlow*.
- Develop trading systems and analyze portfolios using *MarketFlow* and Quantopian's `pyfolio`.

The `alphapy` package is the base platform. The *domain* pipelines MarketFlow (`mflow`) and SportFlow (`sflow`) run on top of `alphapy`. As shown in the diagram below, we separate the domain pipeline from the model pipeline. The main job of a domain pipeline is to transform the raw application data into canonical form, i.e., a training set and a testing set. The model pipeline is flexible enough to handle any project and evolved over many Kaggle competitions.



Let's review all of the components in the diagram:

`Domain Pipeline:` This is the Python code that creates the standard training and testing data. For example, you may be combining different data frames or collecting time series data from an external feed. These data are transformed for input into the model pipeline.

`Domain YAML:` AlphaPy uses configuration files written in YAML to give the data scientist maximum flexibility. Typically, you will have a standard YAML template for each domain or application.

`Training Data:` The training data is an external file that is read as a pandas dataframe. For classification, one of the columns will represent the target or dependent variable.

Testing Data: The testing data is an external file that is read as a pandas dataframe. For classification, the labels may or may not be included.

Model Pipeline: This Python code is generic for running all classification or regression models. The pipeline begins with data and ends with a model object for new predictions.

Model YAML: The configuration file has specific sections for running the model pipeline. Every aspect of creating a model is controlled through this file.

Model Object: All models are saved to disk. You can load and run your trained model on new data in scoring mode.

1.1 Core Functionality

AlphaPy has been developed primarily for supervised learning tasks. You can generate models for any classification or regression problem.

- Binary Classification: classify elements into one of two groups
- Multiclass Classification: classify elements into multiple categories
- Regression: predict real values based on derived coefficients

Classification Algorithms:

- AdaBoost
- Extra Trees
- Gradient Boosting
- K-Nearest Neighbors
- Logistic Regression
- Support Vector Machine (including Linear)
- Naive Bayes (including Multinomial)
- Radial Basis Functions
- Random Forests
- XGBoost Binary and Multiclass

Regression Algorithms:

- Extra Trees
- Gradient Boosting
- K-Nearest Neighbor
- Linear Regression
- Random Forests
- XGBoost

1.2 External Packages

AlphaPy relies on a number of key packages in both its model and domain pipelines. Although most packages are included in the Anaconda Python platform, most of the following packages are not, so please refer to the Web or Github site for further information.

- categorical-encoding: https://github.com/scikit-learn-contrib/categorical-encoding
- imbalanced-learn: https://github.com/scikit-learn-contrib/imbalanced-learn
- pyfolio: https://github.com/quantopian/pyfolio
- XGBoost: https://github.com/dmlc/xgboost

CHAPTER

TWO

QUICK START

Install the `alphapy` package:

```
pip install -U alphapy
```

Note: Please refer to *Installation* for further details.

From your command line application, clone the repository:

```
git clone https://github.com/ScottFreeLLC/AlphaPy
```

Change your directory to the examples location:

```
cd AlphaPy/alphapy/examples
```

Go to any of the examples below:

- *Kaggle Tutorial*
- *Market Prediction Tutorial*
- *NCAA Basketball Tutorial*
- *Trading System Tutorial*

Note: Note that you can work entirely within a Jupyter notebook, or solely from the command line. Generally, we like to run the pipelines first and then perform our analysis within a notebook.

CHAPTER

THREE

INSTALLATION

You should already have pip, Python, and XGBoost (see below) installed on your system. Run the following command to install AlphaPy:

```
pip install -U alphapy
```

3.1 XGBoost

For Macintosh and Window users, XGBoost will *not* install automatically with `pip`. For instructions to install XGBoost on your specific platform, go to http://xgboost.readthedocs.io/en/latest/build.html.

3.2 Anaconda Python

Note: If you already have the Anaconda Python distribution, then you can create a virtual environment for AlphaPy with *conda* with the following recipe.

```
conda create -n alphapy python=3.5
source activate alphapy
conda install -c conda-forge bokeh
conda install -c conda-forge ipython
conda install -c conda-forge matplotlib
conda install -c conda-forge numpy
conda install -c conda-forge pandas
conda install -c conda-forge pyyaml
conda install -c conda-forge scikit-learn
conda install -c conda-forge scipy
conda install -c conda-forge seaborn
conda install -c conda-forge xgboost
pip install pandas_datareader
pip install imbalanced-learn
pip install category_encoders
pip install pyfolio
```

CHAPTER

FOUR

COMMAND LINE

The AlphaPy Command Line Interface (CLI) was designed to be as simple as possible. First, change the directory to your project location, where you have already followed the *Project Structure* specifications:

```
cd path/to/project
```

Run this command to train a model:

```
alphapy
```

Usage:

```
alphapy [--train | --predict]
```

The AlphaPy CLI has the following options:

--train Train a new model and make predictions [Default]

--predict Make predictions from a saved model

The domain pipelines have additional options for time series:

```
mflow [--train | --predict] [--tdate yyyy-mm-dd] [--pdate yyyy-mm-dd]
sflow [--train | --predict] [--tdate yyyy-mm-dd] [--pdate yyyy-mm-dd]
```

--train Train a new model and make predictions (Default)

--predict Make predictions from a saved model

--tdate The training date in format YYYY-MM-DD (Default: Earliest Date in the Data)

--pdate The prediction date in format YYYY-MM-DD (Default: Today's Date)

CHAPTER

FIVE

SUPPORT

The official channel for support is to open an issue on Github.

http://github.com/ScottFreeLLC/AlphaPy/issues

Follow us on Twitter:

https://twitter.com/scottfreellc?lang=en

5.1 Donations

If you like the software, please click on the *Donate* button below:

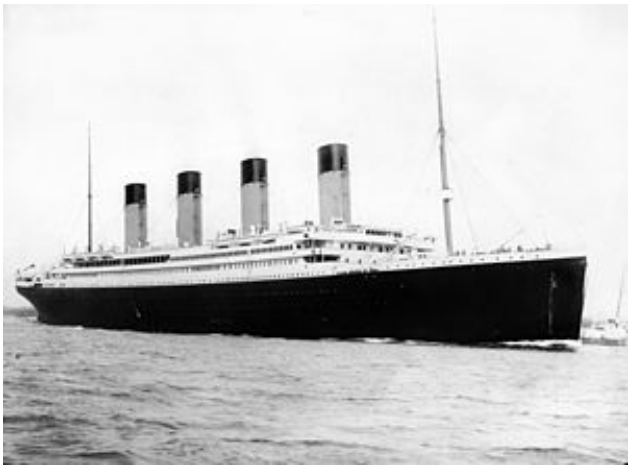
Note: Thank you, we appreciate your generosity. Enjoy the software!

CHAPTER

SIX

KAGGLE TUTORIAL

AlphaPy Running Time: Approximately 2 minutes



The most popular introductory project on Kaggle is Titanic, in which you apply machine learning to predict which passengers were most likely to survive the sinking of the famous ship. In this tutorial, we will run AlphaPy to train a model, generate predictions, and create a submission file so you can see where you land on the Kaggle leaderboard.

Note: AlphaPy is a good starter for most Kaggle competitions. We also use it for other competitions such as the crowd-sourced hedge fund Numerai.

Step 1: From the `examples` directory, change your directory:

```
cd Kaggle
```

Before running AlphaPy, let's briefly review the `model.yml` file in the `config` directory. We will submit the actual predictions (1 vs. 0) instead of the probabilities, so `submit_probas` is set to `False`. All features will be included except for the `PassengerId`. The target variable is `Survived`, the label we are trying to accurately predict.

We'll compare random forests and XGBoost, run recursive feature elimination and a grid search, and select the best model. Note that a blended model of all the algorithms is a candidate for best model. The details of each algorithm are located in the `algos.yml` file.

Listing 1: **model.yml**

```
project:
    directory           : .
    file_extension      : csv
```

(continues on next page)

(continued from previous page)

```
    submission_file   : 'gender_submission'
    submit_probas     : False

data:
    drop              : ['PassengerId']
    features          : '*'
    sampling          :
        option        : False
        method        : under_random
        ratio         : 0.5
    sentinel          : -1
    separator         : ','
    shuffle           : False
    split             : 0.4
    target            : Survived
    target_value      : 1

model:
    algorithms        : ['RF', 'XGB']
    balance_classes   : True
    calibration       :
        option        : False
        type          : sigmoid
    cv_folds          : 3
    estimators        : 51
    feature_selection :
        option        : False
        percentage    : 50
        uni_grid      : [5, 10, 15, 20, 25]
        score_func    : f_classif
    grid_search       :
        option        : True
        iterations    : 50
        random        : True
        subsample     : False
        sampling_pct  : 0.2
    pvalue_level      : 0.01
    rfe               :
        option        : True
        step          : 3
    scoring_function  : roc_auc
    type              : classification

features:
    clustering        :
        option        : True
        increment     : 3
        maximum       : 30
        minimum       : 3
    counts            :
        option        : True
    encoding          :
        rounding      : 2
        type          : factorize
    factors           : []
    interactions      :
        option        : True
```

(continues on next page)

(continued from previous page)

```
        poly_degree    : 5
        sampling_pct   : 10
    isomap             :
        option         : False
        components     : 2
        neighbors      : 5
    logtransform       :
        option         : False
    numpy              :
        option         : True
    pca                :
        option         : False
        increment      : 1
        maximum        : 10
        minimum        : 2
        whiten         : False
    scaling            :
        option         : True
        type           : standard
    scipy              :
        option         : False
    text               :
        ngrams         : 3
        vectorize      : False
    tsne               :
        option         : False
        components     : 2
        learning_rate  : 1000.0
        perplexity     : 30.0
    variance           :
        option         : True
        threshold      : 0.1

pipeline:
    number_jobs        : -1
    seed               : 42
    verbosity          : 0

plots:
    calibration        : True
    confusion_matrix   : True
    importances        : True
    learning_curve     : True
    roc_curve          : True

xgboost:
    stopping_rounds    : 20
```

Step 2: Now, we are ready to run AlphaPy. Enter the following command:

```
alphapy
```

As `alphapy` runs, you will see the progress of the workflow, and the logging output is saved in `alphapy.log`. When the workflow completes, your project structure will look like this, with a different datestamp:

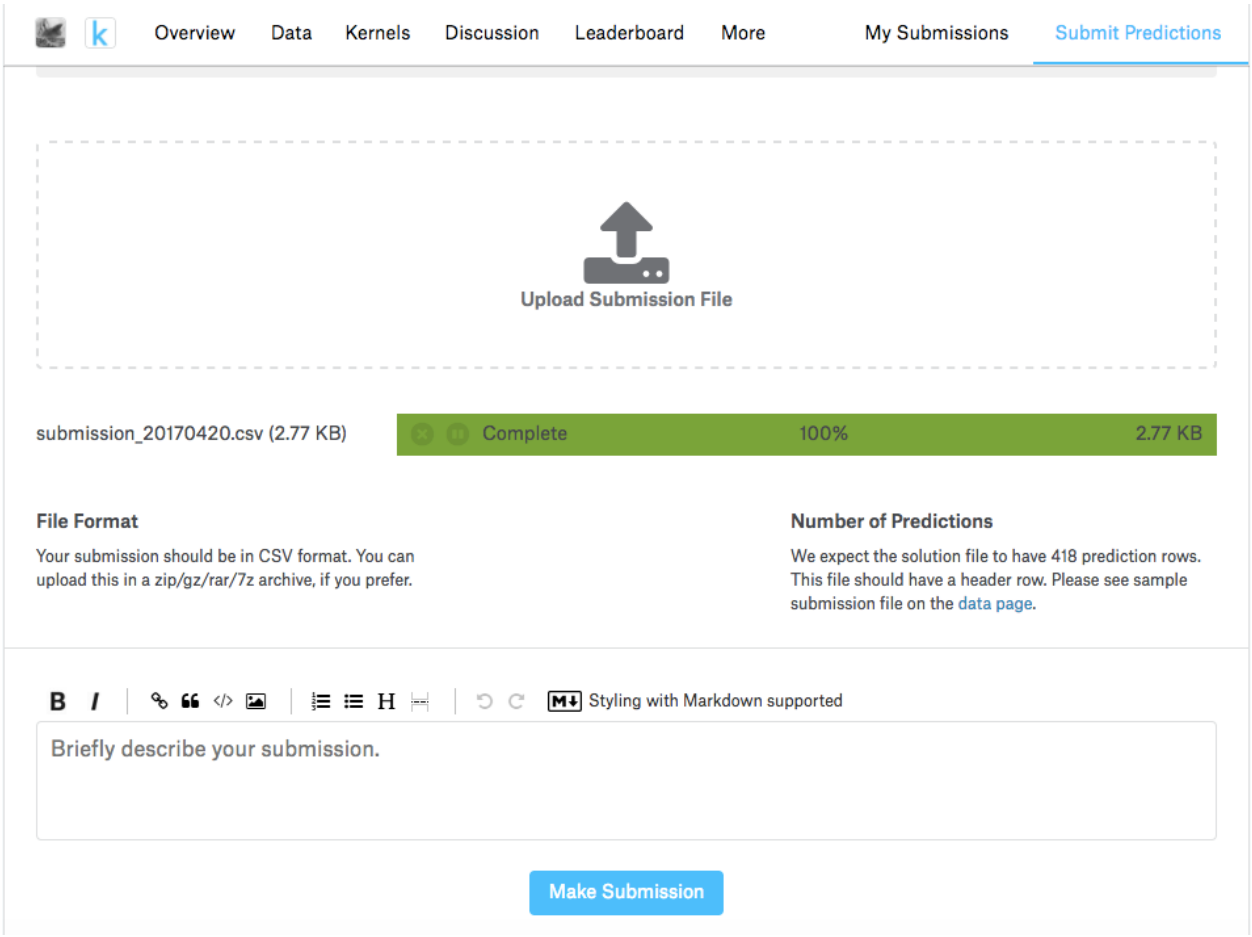
```
Kaggle
```

(continues on next page)

(continued from previous page)

```
├── alphapy.log
├── config
    ├── algos.yml
    ├── model.yml
└── data
└── input
    ├── test.csv
    ├── train.csv
└── model
    ├── feature_map_20170420.pkl
    ├── model_20170420.pkl
└── output
    ├── predictions_20170420.csv
    ├── probabilities_20170420.csv
    ├── rankings_20170420.csv
    ├── submission_20170420.csv
└── plots
    ├── calibration_train.png
    ├── confusion_train_RF.png
    ├── confusion_train_XGB.png
    ├── feature_importance_train_RF.png
    ├── feature_importance_train_XGB.png
    ├── learning_curve_train_RF.png
    ├── learning_curve_train_XGB.png
    ├── roc_curve_train.png
```

Step 3: To see how your model ranks on the Kaggle leaderboard, upload the submission file from the `output` directory to the Web site https://www.kaggle.com/c/titanic/submit.

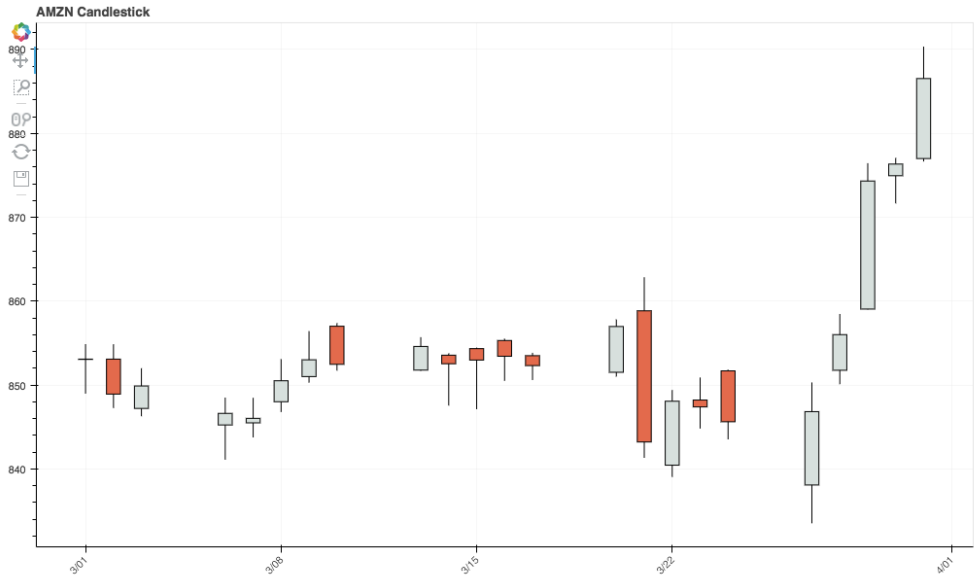

Overview
Data
Kernels
Discussion
Leaderboard
More
My Submissions
Submit Predictions
Upload Submission File
submission_20170420.csv (2.77 KB)
Complete
100%
2.77 KB
File Format
Your submission should be in CSV format. You can upload this in a zip/gz/rar/7z archive, if you prefer.
Number of Predictions
We expect the solution file to have 418 prediction rows. This file should have a header row. Please see sample submission file on the data page.
Styling with Markdown supported
Briefly describe your submission.
Make Submission

CHAPTER

SEVEN

MARKET PREDICTION TUTORIAL

MarketFlow Running Time: Approximately 6 minutes



Machine learning subsumes *technical analysis* because collectively, technical analysis is just a set of features for market prediction. We can use machine learning as a feature blender for moving averages, indicators such as RSI and ADX, and even representations of chart formations such as double tops and head-and-shoulder patterns.

We are not directly predicting net return in our models, although that is the ultimate goal. By characterizing the market with models, we can increase the Return On Investment (ROI). We have a wide range of dependent or target variables from which to choose, not just net return. There is more power in building a classifier rather than a more traditional regression model, so we want to define binary conditions such as whether or not today is going to be a trend day, rather than a numerical prediction of today's return.

In this tutorial, we will train a model that predicts whether or not the next day will have a larger-than-average range. This is important for deciding which system to deploy on the prediction day. If our model gives us predictive power, then we can filter out those days where trading a given system is a losing strategy.

Step 1: From the `examples` directory, change your directory:

```
cd "Trading Model"
```

Before running MarketFlow, let's briefly review the configuration files in the `config` directory:

`market.yml`: The MarketFlow configuration file

`model.yml`: The AlphaPy configuration file

In `market.yml`, we limit our model to six stocks in the target group `test`, going back 2000 trading days. You can define any group of stock symbols in the `groups` section, and then set the `target_group` attribute in the `market` section to the name of that group.

This is a 1-day forecast, but we also use those features that can be calculated at the market open, such as gap information in the `leaders` section. In the `features` section, we define many variables for moving averages, historical range, RSI, volatility, and volume.

Listing 1: **market.yml**

```
market:
    create_model    : True
    data_fractal    : 1d
    data_history    : 500
    forecast_period : 1
    fractal         : 1d
    lag_period      : 1
    leaders         : ['gap', 'gapbadown', 'gapbaup', 'gapdown', 'gapup']
    predict_history : 100
    schema          : yahoo
    subject         : stock
    target_group    : test

groups:
    all  : ['aaoi', 'aapl', 'acia', 'adbe', 'adi', 'adp', 'agn', 'aig', 'akam',
            'algn', 'alk', 'alxn', 'amat', 'amba', 'amd', 'amgn', 'amt', 'amzn',
            'antm', 'arch', 'asml', 'athn', 'atvi', 'auph', 'avgo', 'axp', 'ayx',
            'azo', 'ba', 'baba', 'bac', 'bby', 'bidu', 'biib', 'brcd', 'bvsn',
            'bwld', 'c', 'cacc', 'cara', 'casy', 'cat', 'cde', 'celg', 'cern',
            'chkp', 'chtr', 'clvs', 'cme', 'cmg', 'cof', 'cohr', 'comm', 'cost',
            'cpk', 'crm', 'crus', 'csco', 'ctsh', 'ctxs', 'csx', 'cvs', 'cybr',
            'data', 'ddd', 'deck', 'dgaz', 'dia', 'dis', 'dish', 'dnkn', 'dpz',
            'drys', 'dust', 'ea', 'ebay', 'edc', 'edz', 'eem', 'elli', 'eog',
            'esrx', 'etrm', 'ewh', 'ewt', 'expe', 'fang', 'fas', 'faz', 'fb',
            'fcx', 'fdx', 'ffiv', 'fit', 'five', 'fnsr', 'fslr', 'ftnt', 'gddy',
            'gdx', 'gdxj', 'ge', 'gild', 'gld', 'glw', 'gm', 'googl', 'gpro',
            'grub', 'gs', 'gwph', 'hal', 'has', 'hd', 'hdp', 'hlf', 'hog', 'hum',
            'ibb', 'ibm', 'ice', 'idxx', 'ilmn', 'ilmn', 'incy', 'intc', 'intu',
            'ip', 'isrg', 'iwm', 'ivv', 'iwf', 'iwm', 'jack', 'jcp', 'jdst', 'jnj',
            'jnpr', 'jnug', 'jpm', 'kite', 'klac', 'ko', 'kss', 'labd', 'labu',
            'len', 'lite', 'lmt', 'lnkd', 'lrcx', 'lulu', 'lvs', 'mbly', 'mcd',
            'mchp', 'mdy', 'meoh', 'mnst', 'mo', 'momo', 'mon', 'mrk', 'ms', 'msft',
            'mtb', 'mu', 'nflx', 'nfx', 'nke', 'ntap', 'ntes', 'ntnx', 'nugt',
            'nvda', 'nxpi', 'nxst', 'oii', 'oled', 'orcl', 'orly', 'p', 'panw',
            'pcln', 'pg', 'pm', 'pnra', 'prgo', 'pxd', 'pypl', 'qcom', 'qqq',
            'qrvo', 'rht', 'sam', 'sbux', 'sds', 'sgen', 'shld', 'shop', 'sig',
            'sina', 'siri', 'skx', 'slb', 'slv', 'smh', 'snap', 'sncr', 'soda',
            'splk', 'spy', 'stld', 'stmp', 'stx', 'svxy', 'swks', 'symc', 't',
            'tbt', 'teva', 'tgt', 'tho', 'tlt', 'tmo', 'tna', 'tqqq', 'trip',
            'tsla', 'ttwo', 'tvix', 'twlo', 'twtr', 'tza', 'uaa', 'ugaz', 'uhs',
            'ulta', 'ulti', 'unh', 'unp', 'upro', 'uri', 'ups', 'uri', 'uthr',
            'utx', 'uvxy', 'v', 'veev', 'viav', 'vlo', 'vmc', 'vrsn', 'vrtx', 'vrx',
            'vwo', 'vxx', 'vz', 'wday', 'wdc', 'wfc', 'wfm', 'wmt', 'wynn', 'x',
            'xbi', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlnx', 'xom', 'xlp', 'xlu',
            'xlv', 'xme', 'xom', 'wix', 'yelp', 'z']
    etf  : ['dia', 'dust', 'edc', 'edz', 'eem', 'ewh', 'ewt', 'fas', 'faz',
            'gld', 'hyg', 'iwm', 'ivv', 'iwf', 'jnk', 'mdy', 'nugt', 'qqq',
            'sds', 'smh', 'spy', 'tbt', 'tlt', 'tna', 'tvix', 'tza', 'upro',
```

(continues on next page)

(continued from previous page)

```
            'uvxy', 'vwo', 'vxx', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlp',
            'xlu', 'xlv', 'xme']
    tech : ['aapl', 'adbe', 'amat', 'amgn', 'amzn', 'avgo', 'baba', 'bidu',
            'brcd', 'csco', 'ddd', 'emc', 'expe', 'fb', 'fit', 'fslr', 'goog',
            'intc', 'isrg', 'lnkd', 'msft', 'nflx', 'nvda', 'pcln', 'qcom',
            'qqq', 'tsla', 'twtr']
    test : ['aapl', 'amzn', 'goog', 'fb', 'nvda', 'tsla']

features: ['abovema_3', 'abovema_5', 'abovema_10', 'abovema_20', 'abovema_50',
           'adx', 'atr', 'bigdown', 'bigup', 'diminus', 'diplus', 'doji',
           'gap', 'gapbadown', 'gapbaup', 'gapdown', 'gapup',
           'hc', 'hh', 'ho', 'hl', 'lc', 'lh', 'll', 'lo', 'hookdown', 'hookup',
           'inside', 'outside', 'madelta_3', 'madelta_5', 'madelta_7', 'madelta_10',
           'madelta_12', 'madelta_15', 'madelta_18', 'madelta_20', 'madelta',
           'net', 'netdown', 'netup', 'nr_3', 'nr_4', 'nr_5', 'nr_7', 'nr_8',
           'nr_10', 'nr_18', 'roi', 'roi_2', 'roi_3', 'roi_4', 'roi_5', 'roi_10',
           'roi_20', 'rr_1_4', 'rr_1_7', 'rr_1_10', 'rr_2_5', 'rr_2_7', 'rr_2_10',
           'rr_3_8', 'rr_3_14', 'rr_4_10', 'rr_4_20', 'rr_5_10', 'rr_5_20',
           'rr_5_30', 'rr_6_14', 'rr_6_25', 'rr_7_14', 'rr_7_35', 'rr_8_22',
           'rrhigh', 'rrlow', 'rrover', 'rrunder', 'rsi_3', 'rsi_4', 'rsi_5',
           'rsi_6', 'rsi_8', 'rsi_10', 'rsi_14', 'sep_3_3', 'sep_5_5', 'sep_8_8',
           'sep_10_10', 'sep_14_14', 'sep_21_21', 'sep_30_30', 'sep_40_40',
           'sephigh', 'seplow', 'trend', 'vma', 'vmover', 'vmratio', 'vmunder',
           'volatility_3', 'volatility_5', 'volatility', 'volatility_20',
           'wr_2', 'wr_3', 'wr', 'wr_5', 'wr_6', 'wr_7', 'wr_10']

aliases:
    atr         : 'ma_truerange'
    aver        : 'ma_hlrange'
    cma         : 'ma_close'
    cmax        : 'highest_close'
    cmin        : 'lowest_close'
    hc          : 'higher_close'
    hh          : 'higher_high'
    hl          : 'higher_low'
    ho          : 'higher_open'
    hmax        : 'highest_high'
    hmin        : 'lowest_high'
    lc          : 'lower_close'
    lh          : 'lower_high'
    ll          : 'lower_low'
    lo          : 'lower_open'
    lmax        : 'highest_low'
    lmin        : 'lowest_low'
    net         : 'net_close'
    netdown     : 'down_net'
    netup       : 'up_net'
    omax        : 'highest_open'
    omin        : 'lowest_open'
    rmax        : 'highest_hlrange'
    rmin        : 'lowest_hlrange'
    rr          : 'maratio_hlrange'
    rixc        : 'rindex_close_high_low'
    rixo        : 'rindex_open_high_low'
    roi         : 'netreturn_close'
    rsi         : 'rsi_close'
    sepma       : 'ma_sep'
```

(continues on next page)

(continued from previous page)

```
    vma         : 'ma_volume'
    vmratio     : 'maratio_volume'
    upmove      : 'net_high'

variables:
    abovema     : 'close > cma_50'
    belowma     : 'close < cma_50'
    bigup       : 'rrover & sephigh & netup'
    bigdown     : 'rrover & sephigh & netdown'
    doji        : 'sepdoji & rrunder'
    hookdown    : 'open > high[1] & close < close[1]'
    hookup      : 'open < low[1] & close > close[1]'
    inside      : 'low > low[1] & high < high[1]'
    madelta     : '(close - cma_50) / atr_10'
    nr          : 'hlrange == rmin_4'
    outside     : 'low < low[1] & high > high[1]'
    roihigh     : 'roi_5 >= 5'
    roilow      : 'roi_5 < -5'
    roiminus    : 'roi_5 < 0'
    roiplus     : 'roi_5 > 0'
    rrhigh      : 'rr_1_10 >= 1.2'
    rrlow       : 'rr_1_10 <= 0.8'
    rrover      : 'rr_1_10 >= 1.0'
    rrunder     : 'rr_1_10 < 1.0'
    sep         : 'rixc_1 - rixo_1'
    sepdoji     : 'abs(sep) <= 15'
    sephigh     : 'abs(sep_1_1) >= 70'
    seplow      : 'abs(sep_1_1) <= 30'
    trend       : 'rrover & sephigh'
    vmover      : 'vmratio >= 1'
    vmunder     : 'vmratio < 1'
    volatility  : 'atr_10 / close'
    wr          : 'hlrange == rmax_4'
```

In each of the tutorials, we experiment with different options in `model.yml` to run AlphaPy. Here, we first apply univariate feature selection and then run a random forest classifier with Recursive Feature Elimination, including Cross-Validation (RFECV). When you choose RFECV, the process takes much longer, so if you want to see more logging, then increase the `verbosity` level in the `pipeline` section.

Since stock prices are time series data, we apply the `runs_test` function to twelve features in the `treatments` section. Treatments are powerful because you can write any function to extrapolate new features from existing ones. AlphaPy provides some of these functions in the `alphapy.features` module, but it can also import external functions as well.

Our target variable is `rrover`, the ratio of the 1-day range to the 10-day average high/low range. If that ratio is greater than or equal to 1.0, then the value of `rrover` is True. This is what we are trying to predict.

Listing 2: **model.yml**

```
project:
    directory         : .
    file_extension    : csv
    submission_file   :
    submit_probas     : False

data:
    drop              : ['date', 'tag', 'open', 'high', 'low', 'close', 'volume',
↪'adjclose',
```

(continues on next page)

(continued from previous page)

```
                          'low[1]', 'high[1]', 'net', 'close[1]', 'rmin_3', 'rmin_4',
↪'rmin_5',
                          'rmin_7', 'rmin_8', 'rmin_10', 'rmin_18', 'pval', 'mval',
↪'vma',
                          'rmax_2', 'rmax_3', 'rmax_4', 'rmax_5', 'rmax_6', 'rmax_7',
↪'rmax_10']
    features           : '*'
    sampling           :
        option         : True
        method         : under_random
        ratio          : 0.5
    sentinel           : -1
    separator          : ','
    shuffle            : True
    split              : 0.4
    target             : rrover
    target_value       : True

model:
    algorithms         : ['RF']
    balance_classes    : True
    calibration        :
        option         : False
        type           : isotonic
    cv_folds           : 3
    estimators         : 501
    feature_selection  :
        option         : True
        percentage     : 50
        uni_grid       : [5, 10, 15, 20, 25]
        score_func     : f_classif
    grid_search        :
        option         : False
        iterations     : 100
        random         : True
        subsample      : True
        sampling_pct   : 0.25
    pvalue_level       : 0.01
    rfe                :
        option         : True
        step           : 10
    scoring_function   : 'roc_auc'
    type               : classification

features:
    clustering         :
        option         : False
        increment      : 3
        maximum        : 30
        minimum        : 3
    counts             :
        option         : False
    encoding           :
        rounding       : 3
        type           : factorize
    factors            : []
    interactions       :
```

(continues on next page)

(continued from previous page)

```
        option         : True
        poly_degree    : 2
        sampling_pct   : 5
    isomap             :
        option         : False
        components     : 2
        neighbors      : 5
    logtransform       :
        option         : False
    numpy              :
        option         : False
    pca                :
        option         : False
        increment      : 3
        maximum        : 15
        minimum        : 3
        whiten         : False
    scaling            :
        option         : True
        type           : standard
    scipy              :
        option         : False
    text               :
        ngrams         : 1
        vectorize      : False
    tsne               :
        option         : False
        components     : 2
        learning_rate  : 1000.0
        perplexity     : 30.0
    variance           :
        option         : True
        threshold      : 0.1

treatments:
    doji               : ['alphapy.features', 'runs_test', ['all'], 18]
    hc                 : ['alphapy.features', 'runs_test', ['all'], 18]
    hh                 : ['alphapy.features', 'runs_test', ['all'], 18]
    hl                 : ['alphapy.features', 'runs_test', ['all'], 18]
    ho                 : ['alphapy.features', 'runs_test', ['all'], 18]
    rrhigh             : ['alphapy.features', 'runs_test', ['all'], 18]
    rrlow              : ['alphapy.features', 'runs_test', ['all'], 18]
    rrover             : ['alphapy.features', 'runs_test', ['all'], 18]
    rrunder            : ['alphapy.features', 'runs_test', ['all'], 18]
    sephigh            : ['alphapy.features', 'runs_test', ['all'], 18]
    seplow             : ['alphapy.features', 'runs_test', ['all'], 18]
    trend              : ['alphapy.features', 'runs_test', ['all'], 18]

pipeline:
    number_jobs        : -1
    seed               : 10231
    verbosity          : 0

plots:
    calibration        : True
    confusion_matrix   : True
    importances        : True
```

(continues on next page)

(continued from previous page)

```
    learning_curve     : True
    roc_curve          : True

xgboost:
    stopping_rounds    : 20
```

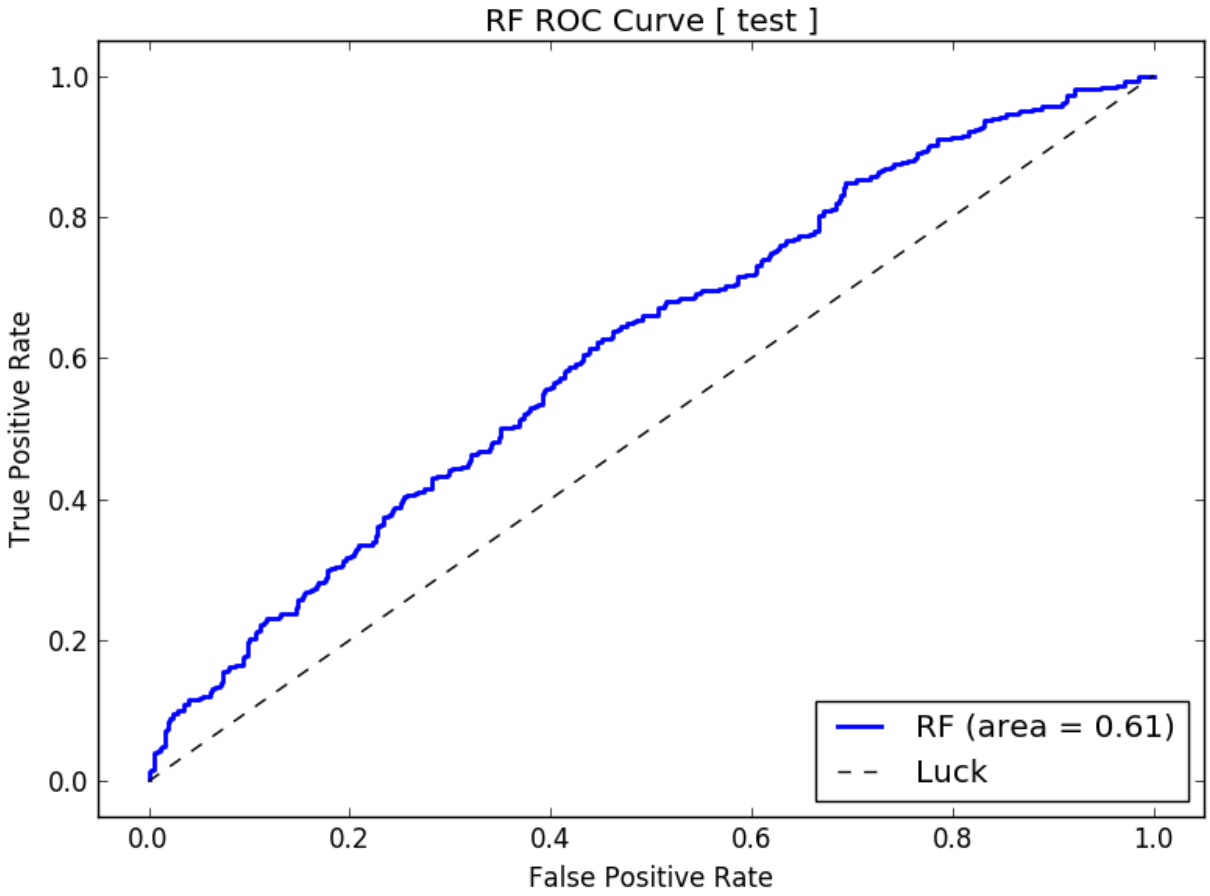
Step 2: Now, let's run MarketFlow:

```
mflow --pdate 2017-10-01
```

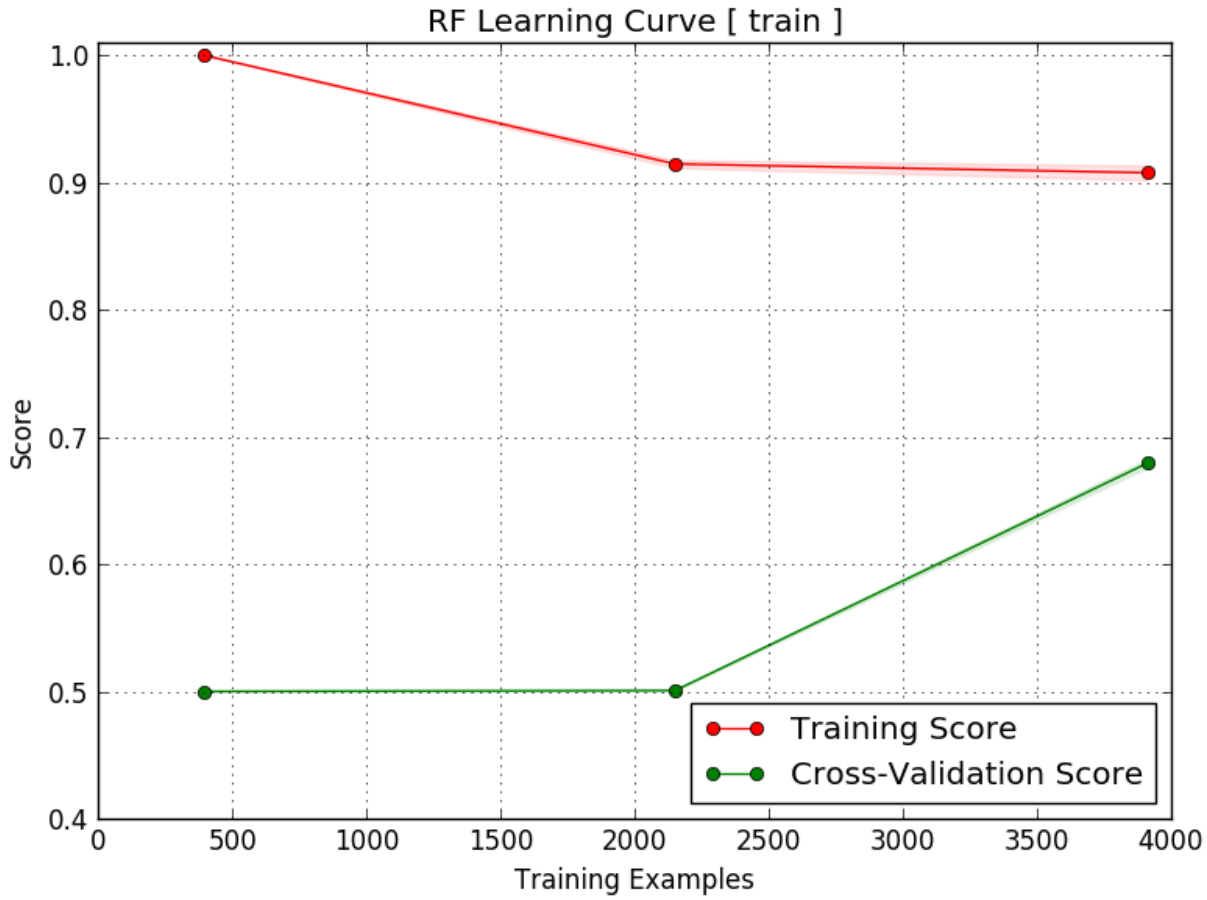
As `mflow` runs, you will see the progress of the workflow, and the logging output is saved in `market_flow.log`. When the workflow completes, your project structure will look like this, with a different datestamp:

```
Trading Model
├── market_flow.log
├── config
    ├── algos.yml
    ├── market.yml
    ├── model.yml
└── data
└── input
    ├── test_20170420.csv
    ├── test.csv
    ├── train_20170420.csv
    ├── train.csv
└── model
    ├── feature_map_20170420.pkl
    ├── model_20170420.pkl
└── output
    ├── predictions_20170420.csv
    ├── probabilities_20170420.csv
    ├── rankings_20170420.csv
└── plots
    ├── calibration_test.png
    ├── calibration_train.png
    ├── confusion_test_RF.png
    ├── confusion_train_RF.png
    ├── feature_importance_train_RF.png
    ├── learning_curve_train_RF.png
    ├── roc_curve_test.png
    ├── roc_curve_train.png
```

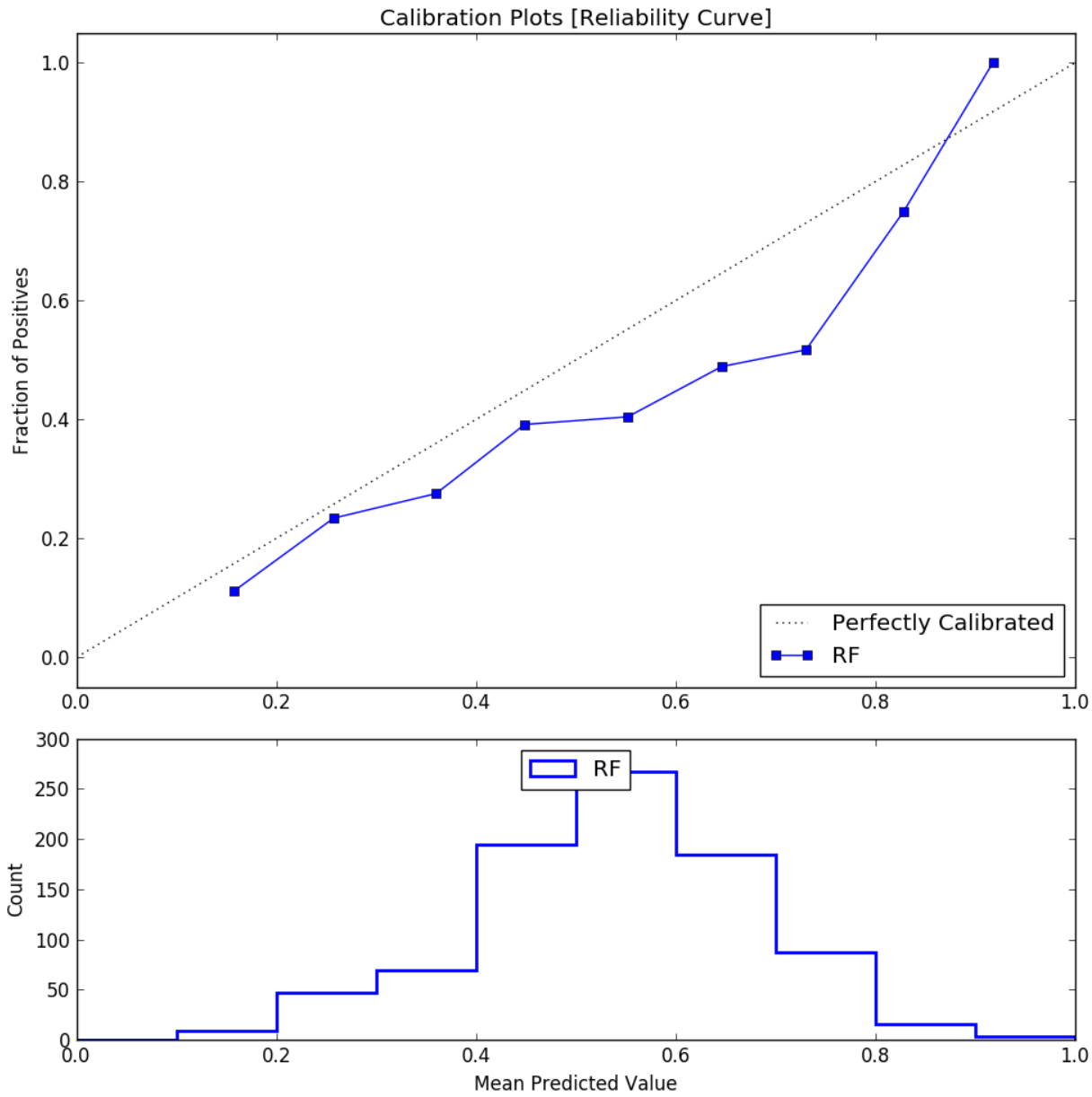
Let's look at the results in the `plots` directory. Since our scoring function was `roc_auc`, we examine the ROC Curve first. The AUC is approximately 0.61, which is not very high but in the context of the stock market, we may still be able to derive some predictive power. Further, we are running the model on a relatively small sample of stocks, as denoted by the jittery line of the ROC Curve.



We can benefit from more samples, as the learning curve shows that the training and cross-validation lines have yet to converge.



The good news is that even with a relatively small number of testing points, the Reliability Curve slopes upward from left to right, with the dotted line denoting a perfect classifier.



To get better accuracy, we can raise our threshold to find the best candidates, since they are ranked by probability, but this also means limiting our pool of stocks. Let's take a closer look at the rankings file.

Step 3: From the command line, enter:

```
jupyter notebook
```

Step 4: Click on the notebook named:

```
A Trading Model.ipynb
```

Step 5: Run the commands in the notebook, making sure that when you read in the rankings file, change the date to match the result from the `ls` command.

`Conclusion` We can predict large-range days with some confidence, but only at a higher probability threshold. This is important for choosing the correct system on any given day. We can achieve better results with more data, so we

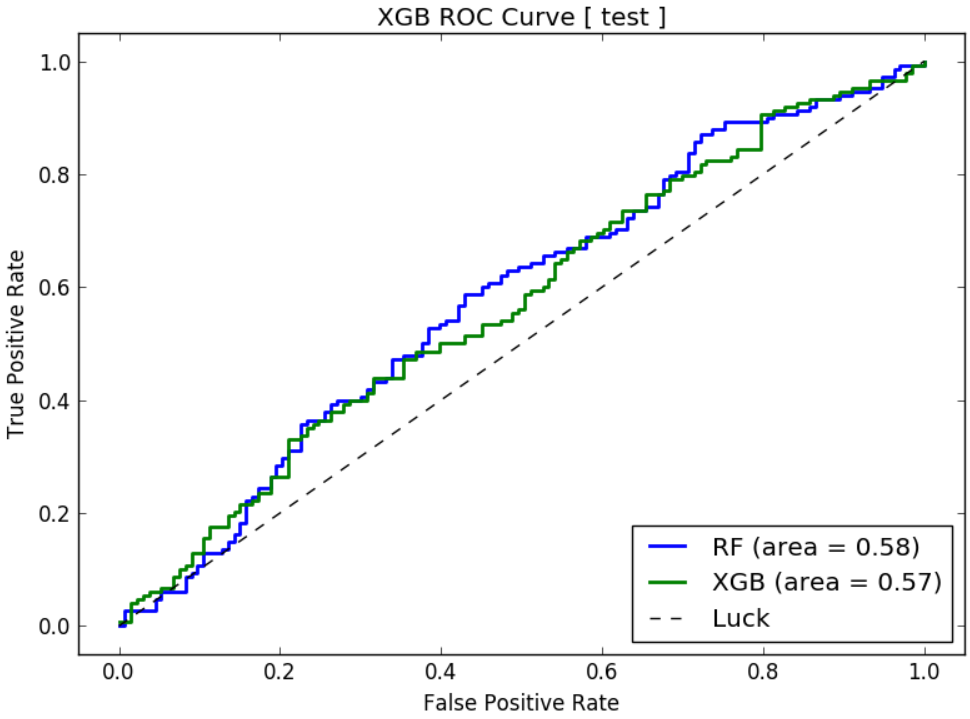
recommend expanding the stock universe, e.g., a group with at least 100 members going five years back.

CHAPTER

EIGHT

NCAA BASKETBALL TUTORIAL

SportFlow Running Time: Approximately 15 minutes



In this tutorial, we use machine learning to predict whether or not an NCAA Men's Basketball team will cover the *spread*. The spread is set by Las Vegas bookmakers to balance the betting; it is a way of giving points to the underdog to encourage bets on both sides.

SportFlow starts with the basic data and derives time series features based on streaks and runs (not the baseball runs). In the table below, the game data includes both *line* and *over_under* information consolidated from various sports Web sites. For example, a line of -9 means the home team is favored by 9 points. A line of +3 means the away team is favored by 3 points; the line is always relative to the home team. An over_under is the predicted total score for the game, with a bet being placed on whether not the final total will be under or over that amount.

Table 1: NCAA Basketball Data

season	date	away.team	away.score	home.team	home.score	line	over_under
2015	2015-11-13	COLO	62	ISU	68	-10	151
2015	2015-11-13	SDAK	69	WRST	77	-6.5	136

continues on next page

Table 1 – continued from previous page

2015	2015-11-13	WAG	57	SJU	66	-5.5	142
2015	2015-11-13	JVST	83	CMU	89	-18	142.5
2015	2015-11-13	NIAG	50	ODU	67	-18	132
2015	2015-11-13	ALBY	65	UK	78	-20	132.5
2015	2015-11-13	TEM	67	UNC	91	-9.5	145
2015	2015-11-13	NKU	61	WVU	107	-23.5	147.5
2015	2015-11-13	SIE	74	DUKE	92	-24	155
2015	2015-11-13	WCU	72	CIN	97	-20	132
2015	2015-11-13	MSM	56	MD	80	-21.5	140
2015	2015-11-13	CHAT	92	UGA	90	-10.5	136
2015	2015-11-13	SEMO	53	DAY	84	-19	140
2015	2015-11-13	DART	67	HALL	84	-11	136
2015	2015-11-13	CAN	85	HOF	96	-10.5	150.5
2015	2015-11-13	JMU	87	RICH	75	-9	137.5
2015	2015-11-13	EIU	49	IND	88	-25	150
2015	2015-11-13	FAU	55	MSU	82	-23.5	141
2015	2015-11-13	SAM	45	LOU	86	-23	142
2015	2015-11-13	MIOH	72	XAV	81	-15.5	144.5
2015	2015-11-13	PRIN	64	RID	56	1	137
2015	2015-11-13	IUPU	72	INST	70	-8	135.5
2015	2015-11-13	SAC	66	ASU	63	-18	144
2015	2015-11-13	AFA	75	SIU	77	-5.5	131
2015	2015-11-13	UNCO	72	KU	109	-29	147.5
2015	2015-11-13	BALL	53	BRAD	54	3	135
2015	2015-11-13	USD	45	USC	83	-12.5	140
2015	2015-11-13	UTM	57	OKST	91	-12	141.5
2015	2015-11-13	COR	81	GT	116	-17	130
2015	2015-11-13	MOST	65	ORU	80	-4.5	133.5
2015	2015-11-13	DREX	81	JOES	82	-9.5	127.5
2015	2015-11-13	WMRY	85	NCST	68	-12.5	149
2015	2015-11-13	SF	78	UIC	75	1.5	148.5
2015	2015-11-13	PEAY	41	VAN	80	-24.5	144
2015	2015-11-13	CSN	71	NIU	83	-9.5	134.5
2015	2015-11-13	UCSB	60	OMA	59	-2.5	157.5
2015	2015-11-13	UTSA	64	LOYI	76	-14	138.5
2015	2015-11-13	BRWN	65	SPU	77	-2	130.5
2015	2015-11-13	NAU	70	WSU	82	-10.5	145

Step 1: First, from the `examples` directory, change your directory:

```
cd NCAAB
```

Before running SportFlow, let's briefly review the configuration files in the `config` directory:

`sport.yml`: The SportFlow configuration file

`model.yml`: The AlphaPy configuration file

In `sport.yml`, the first three items are used for `random_scoring`, which we will not be doing here. By default, we will create a model based on all `seasons` and calculate short-term streaks of 3 with the `rolling_window`.

Listing 1: **sport.yml**

```
sport:
    league           : NCAAB
    points_max       : 100
    points_min       : 50
    random_scoring   : False
    seasons          : []
    rolling_window   : 3
```

In each of the tutorials, we experiment with different options in `model.yml` to run AlphaPy. Here, we will run a random forest classifier with Recursive Feature Elimination and Cross-Validation (RFECV), and then an XGBoost classifier. We will also perform a random grid search, which increases the total running time to approximately 15 minutes. You can get in some two-ball dribbling while waiting for SportFlow to finish.

In the `features` section, we identify the `factors` generated by SportFlow. For example, we want to treat the various streaks as factors. Other options are `interactions`, standard `scaling`, and a `threshold` for removing low-variance features.

Our target variable is `won_on_spread`, a Boolean indicator of whether or not the home team covered the spread. This is what we are trying to predict.

Listing 2: **model.yml**

```
project:
    directory          : .
    file_extension     : csv
    submission_file    :
    submit_probas      : False

data:
    drop               : ['Unnamed: 0', 'index', 'season', 'date', 'home.team', 'away.
↪team',
                          'home.score', 'away.score', 'total_points', 'point_margin_
↪game',
                          'won_on_points', 'lost_on_points', 'cover_margin_game',
                          'lost_on_spread', 'overunder_margin', 'over', 'under']
    features           : '*'
    sampling           :
        option         : False
        method         : under_random
        ratio          : 0.0
    sentinel           : -1
    separator          : ','
    shuffle            : False
    split              : 0.4
    target             : won_on_spread
    target_value       : True

model:
    algorithms         : ['RF', 'XGB']
    balance_classes    : False
    calibration        :
        option         : False
        type           : isotonic
    cv_folds           : 3
    estimators         : 201
```

(continues on next page)

(continued from previous page)

```
    feature_selection :
        option        : False
        percentage    : 50
        uni_grid      : [5, 10, 15, 20, 25]
        score_func    : f_classif
    grid_search       :
        option        : True
        iterations    : 50
        random        : True
        subsample     : False
        sampling_pct  : 0.25
    pvalue_level      : 0.01
    rfe               :
        option        : True
        step          : 5
    scoring_function  : 'roc_auc'
    type              : classification

features:
    clustering        :
        option        : False
        increment     : 3
        maximum       : 30
        minimum       : 3
    counts            :
        option        : False
    encoding          :
        rounding      : 3
        type          : factorize
    factors           : ['line', 'delta.wins', 'delta.losses', 'delta.ties',
                         'delta.point_win_streak', 'delta.point_loss_streak',
                         'delta.cover_win_streak', 'delta.cover_loss_streak',
                         'delta.over_streak', 'delta.under_streak']
    interactions      :
        option        : True
        poly_degree   : 2
        sampling_pct  : 5
    isomap            :
        option        : False
        components    : 2
        neighbors     : 5
    logtransform      :
        option        : False
    numpy             :
        option        : False
    pca               :
        option        : False
        increment     : 3
        maximum       : 15
        minimum       : 3
        whiten        : False
    scaling           :
        option        : True
        type          : standard
    scipy             :
        option        : False
    text              :
```

(continues on next page)

(continued from previous page)

```
        ngrams          : 1
        vectorize       : False
    tsne                :
        option          : False
        components      : 2
        learning_rate   : 1000.0
        perplexity      : 30.0
    variance            :
        option          : True
        threshold       : 0.1

pipeline:
    number_jobs         : -1
    seed                : 13201
    verbosity           : 0

plots:
    calibration         : True
    confusion_matrix    : True
    importances         : True
    learning_curve      : True
    roc_curve           : True

xgboost:
    stopping_rounds     : 30
```

Step 2: Now, let's run SportFlow:

```
sflow --pdate 2016-03-01
```

As `sflow` runs, you will see the progress of the workflow, and the logging output is saved in `sport_flow.log`. When the workflow completes, your project structure will look like this, with a different datestamp:

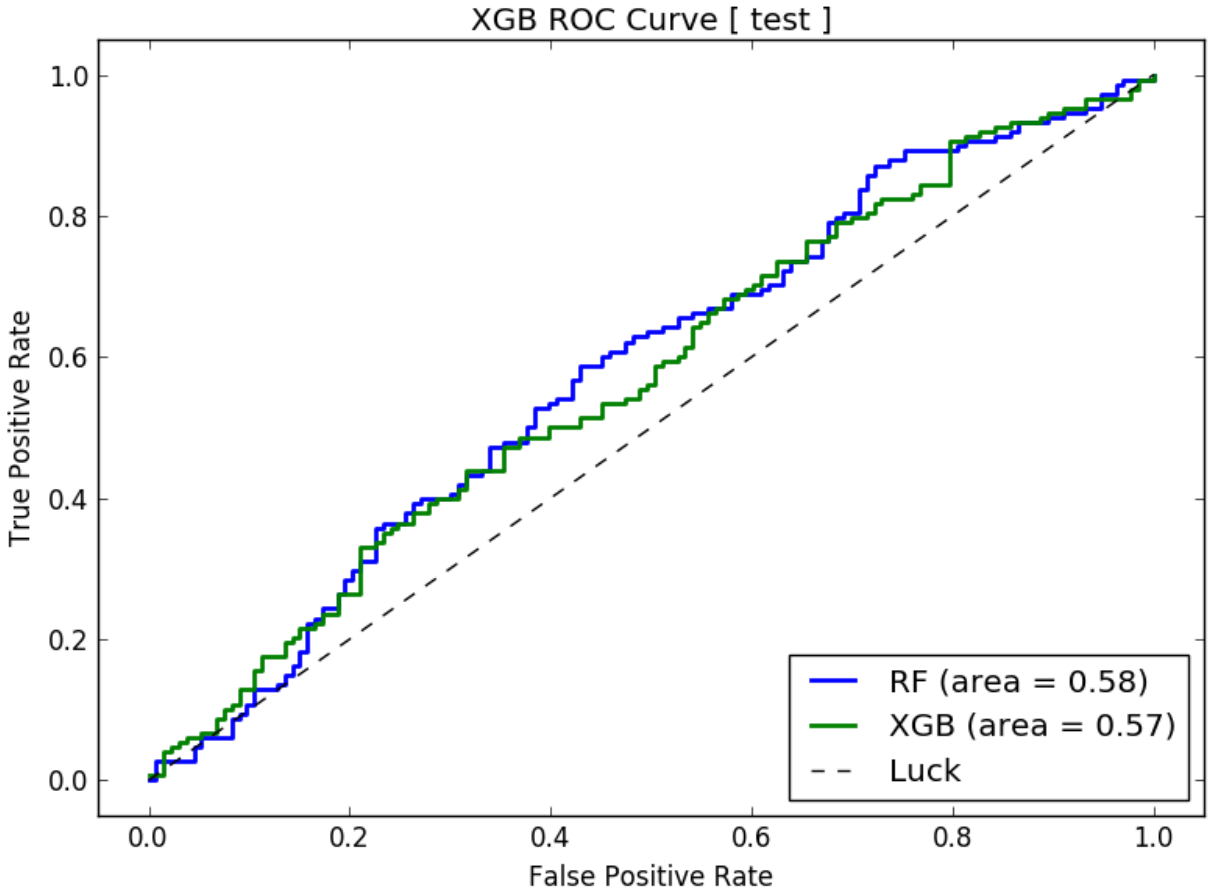
```
NCAAB
├── sport_flow.log
├── config
    ├── algos.yml
    ├── sport.yml
    ├── model.yml
└── data
    ├── ncaab_game_scores_1g.csv
└── input
    ├── test.csv
    ├── train.csv
└── model
    ├── feature_map_20170427.pkl
    ├── model_20170427.pkl
└── output
    ├── predictions_20170427.csv
    ├── probabilities_20170427.csv
    ├── rankings_20170427.csv
└── plots
    ├── calibration_test.png
    ├── calibration_train.png
    ├── confusion_test_RF.png
    ├── confusion_test_XGB.png
```

(continues on next page)

(continued from previous page)

```
    ├── confusion_train_RF.png
    ├── confusion_train_XGB.png
    ├── feature_importance_train_RF.png
    ├── feature_importance_train_XGB.png
    ├── learning_curve_train_RF.png
    ├── learning_curve_train_XGB.png
    ├── roc_curve_test.png
    ├── roc_curve_train.png
```

Depending upon the model parameters and the prediction date, the AUC of the ROC Curve will vary between 0.54 and 0.58. This model is barely passable, but we are getting a slight edge even with our basic data. We will need more game samples to have any confidence in our predictions.



After a model is created, we can run `sflow` in `predict` mode. Just specify the prediction date `pdate`, and SportFlow will make predictions for all cases in the `predict.csv` file on or after the specified date. Note that the `predict.csv` file is generated on the fly in predict mode and stored in the `input` directory.

Step 3: Now, let's run SportFlow in predict mode, where all results will be stored in the `output` directory:

```
sflow --predict --pdate 2016-03-15
```

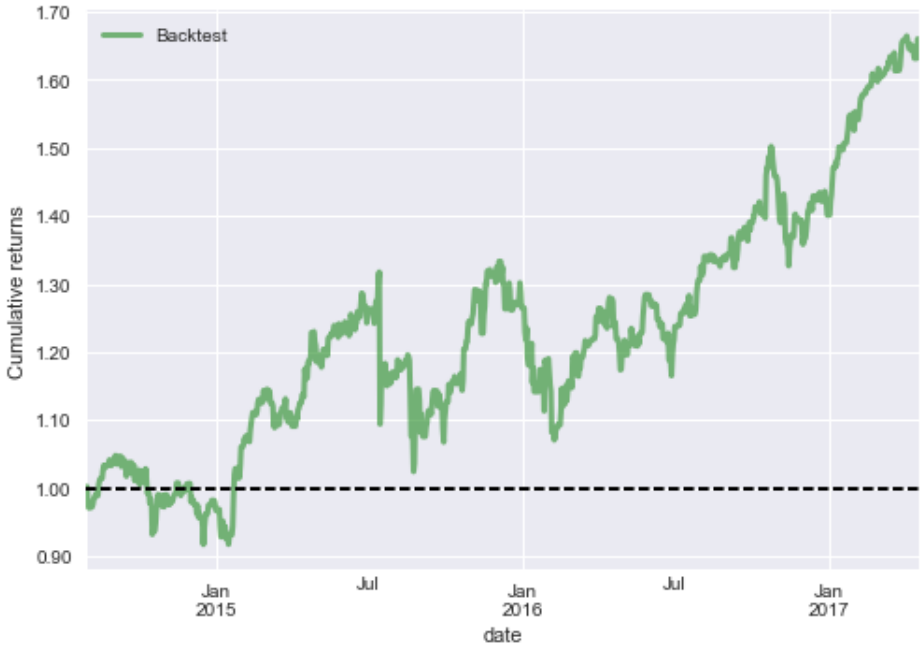
`Conclusion` Even with just one season of NCAA Men's Basketball data, our model predicts between 52-54% accuracy. To attain better accuracy, we need more historical data vis a vis the number of games and other types of information such as individual player statistics. If you want to become a professional bettor, then you need at least 56% winners to break the bank.

CHAPTER

NINE

TRADING SYSTEM TUTORIAL

MarketFlow Running Time: Approximately 1 minute



A trading system is a set of automated rules for buying and selling stocks, options, futures, and other instruments. Trading is considered to be both an art and a science; the scientific branch is known as *technical analysis*. Many technicians spend their lives chasing the Holy Grail: a system that will make them rich simply by detecting common patterns. Technicians in history such as Edwards, Elliott, Fibonacci, Gann, and Gartley show us visually appealing charts, but there is no scientific evidence proving that these techniques actually work.

Trading systems generally operate in two contexts: trend and counter-trend. A system that follows the trend tries to stay in one direction as long as possible. A system that bucks the trend reverses direction at certain support and resistance levels, also known as fading the trend. With MarketFlow, you can implement either type of system using our long/short strategy.

In this tutorial, we are going to test a simple long/short system. If today's closing price is greater than yesterday's close, then we go long. If today's close is lower than yesterday's, then we go short, so we always have a position in the market.

Step 1: From the `examples` directory, change your directory:

```
cd "Trading System"
```

Before running MarketFlow, let's review the `market.yml` file in the `config` directory. Since we are just running a system, we really don't need the `model.yml` file, but if you have a project where the system is derived from a model, then you will want to maintain both files.

In `market.yml`, we will test our system on five stocks in the target group `faang`, going back 1000 trading days. We need to define only two features: `hc` for higher close, and `lc` for lower close. We name the system `closer`, which requires just a `longentry` and a `shortentry`. There are no exit conditions and no holding period, so we will always have a position in each stock.

Listing 1: **market.yml**

```
market:
    create_model    : False
    data_fractal    : 1d
    data_history    : 500
    forecast_period : 1
    fractal         : 1d
    lag_period      : 1
    leaders         : []
    predict_history : 50
    schema          : quandl_wiki
    subject         : stock
    target_group    : faang

system:
    name       : 'closer'
    holdperiod : 0
    longentry  : hc
    longexit   :
    shortentry : lc
    shortexit  :
    scale      : False

groups:
    faang      : ['fb', 'aapl', 'amzn', 'nflx', 'googl']

features       : ['hc', 'lc']

aliases:
    hc         : 'higher_close'
    lc         : 'lower_close'
```

Step 2: Now, let's run MarketFlow:

```
mflow
```

As `mflow` runs, you will see the progress of the workflow, and the logging output is saved in `market_flow.log`. When the workflow completes, your project structure will look like this, with an additional directory `systems`:

```
Trading System
├── market_flow.log
├── config
    ├── algos.yml
    ├── market.yml
    ├── model.yml
└── data
└── input
└── model
```

(continues on next page)

(continued from previous page)

```
└── output
└── plots
└── systems
    ├── faang_closer_positions_1d.csv
    ├── faang_closer_returns_1d.csv
    ├── faang_closer_trades_1d.csv
    ├── faang_closer_transactions_1d.csv
```

MarketFlow records position, return, and transaction data in the `systems` directory, so now we can analyze our results with *Pyfolio*.

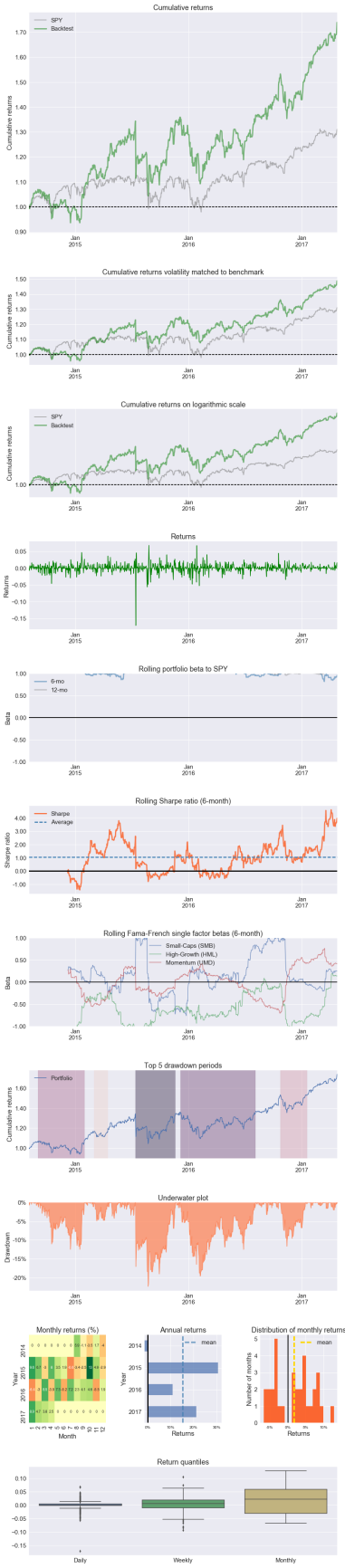
Step 3: From the command line, enter:

```
jupyter notebook
```

Step 4: Click on the notebook named:

```
A Trading System.ipynb
```

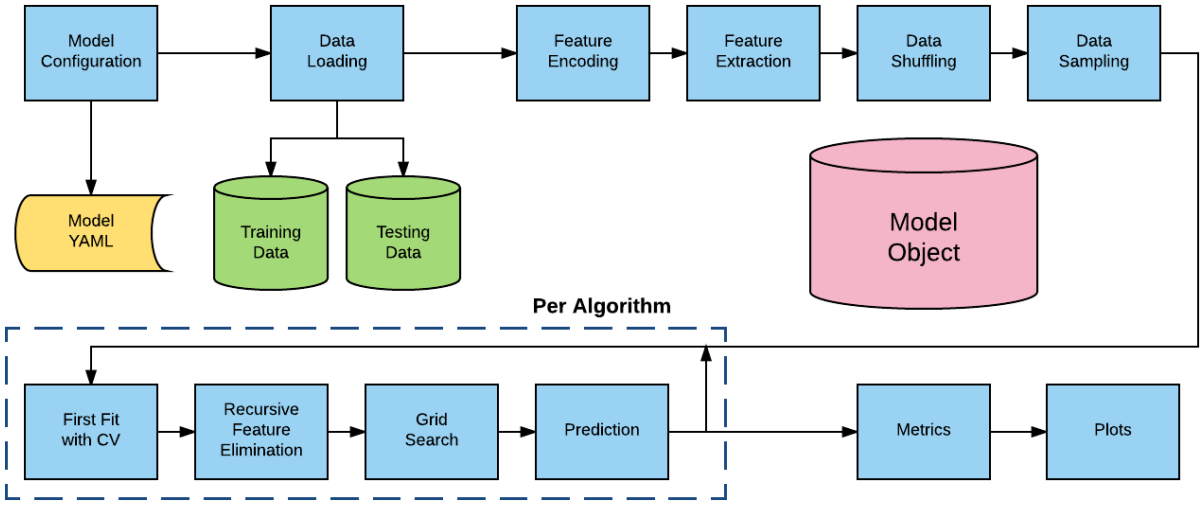
You should obtain the following results in your notebook.


Cumulative returns
SPY
Backtest
Cumulative returns volatility matched to benchmark
Cumulative returns on logarithmic scale
Returns
Rolling portfolio beta to SPY
6-mo
12-mo
Beta
Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
Rolling Fama-French single factor betas (6-month)
Small Caps (SMB)
High-Growth (HML)
Momentum (UMD)
Top 5 drawdown periods
Portfolio
Underwater plot
Drawdown
Monthly returns (%)
Annual returns
mean
Distribution of monthly returns
Number of months
Return quantiles
Daily
Weekly
Monthly

CHAPTER

TEN

ALPHAPY



10.1 Model Object Creation

AlphaPy first reads the `model.yml` file and then displays the model parameters as confirmation that the file was read successfully. As shown in the example below, the Random Forest (RF) and XGBoost (XGB) algorithms are used to build the model. From the model specifications, a `Model` object will be created.

All of the model parameters are listed in alphabetical order. At a minimum, scan for `algorithms`, `features`, `model_type`, and `target` to verify their accuracy, i.e., that you are running the right model. The `verbosity` parameter will control the degree of output that you see when running the pipeline.

Listing 1: **alphapy.log**

```
[12/30/17 23:17:49]␣
↪INFO         ********************************************************************************
[12/30/17 23:17:49] INFO         AlphaPy Start
[12/30/17 23:17:49]␣
↪INFO         ********************************************************************************
[12/30/17 23:17:49] INFO         Model Configuration
[12/30/17 23:17:49] INFO         No Treatments Found
[12/30/17 23:17:49] INFO         MODEL PARAMETERS:
[12/30/17 23:17:49] INFO         algorithms        = ['RF', 'XGB']
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:17:49] INFO        balance_classes   = True
[12/30/17 23:17:49] INFO        calibration       = False
[12/30/17 23:17:49] INFO        cal_type          = sigmoid
[12/30/17 23:17:49] INFO        calibration_plot  = False
[12/30/17 23:17:49] INFO        clustering        = True
[12/30/17 23:17:49] INFO        cluster_inc       = 3
[12/30/17 23:17:49] INFO        cluster_max       = 30
[12/30/17 23:17:49] INFO        cluster_min       = 3
[12/30/17 23:17:49] INFO        confusion_matrix  = True
[12/30/17 23:17:49] INFO        counts            = True
[12/30/17 23:17:49] INFO        cv_folds          = 3
[12/30/17 23:17:49] INFO        directory         = .
[12/30/17 23:17:49] INFO        extension         = csv
[12/30/17 23:17:49] INFO        drop              = ['PassengerId']
[12/30/17 23:17:49] INFO        encoder           = <Encoders.factorize: 3>
[12/30/17 23:17:50] INFO        esr               = 20
[12/30/17 23:17:50] INFO        factors           = []
[12/30/17 23:17:50] INFO        features [X]      = *
[12/30/17 23:17:50] INFO        feature_selection = False
[12/30/17 23:17:50] INFO        fs_percentage     = 50
[12/30/17 23:17:50] INFO        fs_score_func     = <function f_classif at␣
↪0x112f1dd90>
[12/30/17 23:17:50] INFO        fs_uni_grid       = [5, 10, 15, 20, 25]
[12/30/17 23:17:50] INFO        grid_search       = True
[12/30/17 23:17:50] INFO        gs_iters          = 50
[12/30/17 23:17:50] INFO        gs_random         = True
[12/30/17 23:17:50] INFO        gs_sample         = False
[12/30/17 23:17:50] INFO        gs_sample_pct     = 0.200000
[12/30/17 23:17:50] INFO        importances       = True
[12/30/17 23:17:50] INFO        interactions      = True
[12/30/17 23:17:50] INFO        isomap            = False
[12/30/17 23:17:50] INFO        iso_components    = 2
[12/30/17 23:17:50] INFO        iso_neighbors     = 5
[12/30/17 23:17:50] INFO        isample_pct       = 10
[12/30/17 23:17:50] INFO        learning_curve    = True
[12/30/17 23:17:50] INFO        logtransform      = False
[12/30/17 23:17:50] INFO        lv_remove         = True
[12/30/17 23:17:50] INFO        lv_threshold      = 0.100000
[12/30/17 23:17:50] INFO        model_type        = <ModelType.classification: 1>
[12/30/17 23:17:50] INFO        n_estimators      = 51
[12/30/17 23:17:50] INFO        n_jobs            = -1
[12/30/17 23:17:50] INFO        ngrams_max        = 3
[12/30/17 23:17:50] INFO        numpy             = True
[12/30/17 23:17:50] INFO        pca               = False
[12/30/17 23:17:50] INFO        pca_inc           = 1
[12/30/17 23:17:50] INFO        pca_max           = 10
[12/30/17 23:17:50] INFO        pca_min           = 2
[12/30/17 23:17:50] INFO        pca_whiten        = False
[12/30/17 23:17:50] INFO        poly_degree       = 5
[12/30/17 23:17:50] INFO        pvalue_level      = 0.010000
[12/30/17 23:17:50] INFO        rfe               = True
[12/30/17 23:17:50] INFO        rfe_step          = 3
[12/30/17 23:17:50] INFO        roc_curve         = True
[12/30/17 23:17:50] INFO        rounding          = 2
[12/30/17 23:17:50] INFO        sampling          = False
[12/30/17 23:17:50] INFO        sampling_method   = <SamplingMethod.under_random: 12>
[12/30/17 23:17:50] INFO        sampling_ratio    = 0.500000
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:17:50] INFO        scaler_option     = True
[12/30/17 23:17:50] INFO        scaler_type       = <Scalers.standard: 2>
[12/30/17 23:17:50] INFO        scipy             = False
[12/30/17 23:17:50] INFO        scorer            = roc_auc
[12/30/17 23:17:50] INFO        seed              = 42
[12/30/17 23:17:50] INFO        sentinel          = -1
[12/30/17 23:17:50] INFO        separator         = ,
[12/30/17 23:17:50] INFO        shuffle           = False
[12/30/17 23:17:50] INFO        split             = 0.400000
[12/30/17 23:17:50] INFO        submission_file   = gender_submission
[12/30/17 23:17:50] INFO        submit_probas     = False
[12/30/17 23:17:50] INFO        target [y]        = Survived
[12/30/17 23:17:50] INFO        target_value      = 1
[12/30/17 23:17:50] INFO        treatments        = None
[12/30/17 23:17:50] INFO        tsne              = False
[12/30/17 23:17:50] INFO        tsne_components   = 2
[12/30/17 23:17:50] INFO        tsne_learn_rate   = 1000.000000
[12/30/17 23:17:50] INFO        tsne_perplexity   = 30.000000
[12/30/17 23:17:50] INFO        vectorize         = False
[12/30/17 23:17:50] INFO        verbosity         = 0
[12/30/17 23:17:50] INFO        Creating directory ./data
```

10.2 Data Ingestion

Data are loaded from both the training file and the test file. Any features that you wish to remove from the data are then dropped. Statistics about the shape of the data and the target variable proportions are logged.

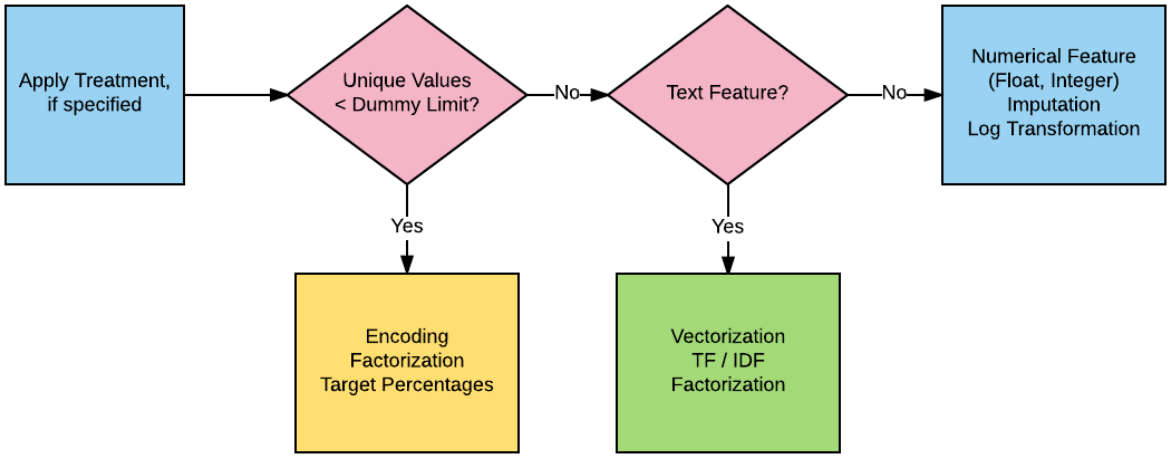
Listing 2: **alphapy.log**

```
[12/30/17 23:17:50] INFO        Creating directory ./model
[12/30/17 23:17:50] INFO        Creating directory ./output
[12/30/17 23:17:50] INFO        Creating directory ./plots
[12/30/17 23:17:50] INFO        Creating Model
[12/30/17 23:17:50] INFO        Calling Pipeline
[12/30/17 23:17:50] INFO        Training Pipeline
[12/30/17 23:17:50] INFO        Loading Data
[12/30/17 23:17:50] INFO        Loading data from ./input/train.csv
[12/30/17 23:17:50] INFO        Found target Survived in data frame
[12/30/17 23:17:50] INFO        Labels (y) found for Partition.train
[12/30/17 23:17:50] INFO        Loading Data
[12/30/17 23:17:50] INFO        Loading data from ./input/test.csv
[12/30/17 23:17:50] INFO        Target Survived not found in Partition.test
[12/30/17 23:17:50] INFO        Saving New Features in Model
[12/30/17 23:17:50] INFO        Original Feature Statistics
[12/30/17 23:17:50] INFO        Number of Training Rows    : 891
[12/30/17 23:17:50] INFO        Number of Training Columns : 11
[12/30/17 23:17:50] INFO        Unique Training Values for Survived : [0 1]
[12/30/17 23:17:50] INFO        Unique Training Counts for Survived : [549 342]
```

10.3 Feature Processing

There are two stages to feature processing. First, you may want to transform a column of a dataframe into a different format or break up a feature into its respective components. This is known as a *treatment*, and it is a one-to-many transformation. For example, a date feature can be extracted into day, month, and year.

The next stage is feature type determination, which applies to all features, regardless of whether or not a treatment has been previously applied. The unique number of a feature's values dictates whether or not that feature is a factor. If the given feature is a factor, then a specific type of encoding is applied. Otherwise, the feature is generally either text or a number.



In the example below, each feature's type is identified along with the unique number of values. For factors, a specific type of encoding is selected, as specified in the `model.yml` file. For text, you can choose either count vectorization and TF-IDF or just plain factorization. Numerical features have both imputation and log-transformation options.

Listing 3: **alphapy.log**

```
[12/30/17 23:17:50] INFO        Number of Testing Rows     : 418
[12/30/17 23:17:50] INFO        Number of Testing Columns  : 11
[12/30/17 23:17:50] INFO        Original Features : Index(['PassengerId', 'Pclass',
↪'Name', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
[12/30/17 23:17:50] INFO        Feature Count     : 11
[12/30/17 23:17:50] INFO        Applying Treatments
[12/30/17 23:17:50] INFO        No Treatments Specified
[12/30/17 23:17:50] INFO        New Feature Count : 11
[12/30/17 23:17:50] INFO        Dropping Features: ['PassengerId']
[12/30/17 23:17:50] INFO        Original Feature Count : 11
[12/30/17 23:17:50] INFO        Reduced Feature Count  : 10
[12/30/17 23:17:50] INFO        Writing data frame to ./input/train_20171230.csv
[12/30/17 23:17:50] INFO        Writing data frame to ./input/test_20171230.csv
[12/30/17 23:17:50] INFO        Creating Cross-Tabulations
[12/30/17 23:17:50] INFO        Original Features : Index(['Pclass', 'Name', 'Sex',
↪'Age', 'SibSp', 'Parch', 'Ticket', 'Fare',
       'Cabin', 'Embarked'],
      dtype='object')
[12/30/17 23:17:50] INFO        Feature Count     : 10
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:17:50] INFO        Creating Count Features
[12/30/17 23:17:50] INFO        NA Counts
[12/30/17 23:17:50] INFO        Number Counts
[12/30/17 23:17:50] INFO        New Feature Count : 21
[12/30/17 23:17:50] INFO        Creating Base Features
[12/30/17 23:17:50] INFO        Feature 1: Pclass is a numerical feature of type␣
↪int64 with 3 unique values
[12/30/17 23:17:50] INFO        Feature 2: Name is a text feature [12:82] with 1307␣
↪unique values
[12/30/17 23:17:50] INFO        Feature 2: Name => Factorization
[12/30/17 23:17:50] INFO        Feature 3: Sex is a text feature [4:6] with 2 unique␣
↪values
[12/30/17 23:17:50] INFO        Feature 3: Sex => Factorization
[12/30/17 23:17:50] INFO        Feature 4: Age is a numerical feature of type float64␣
↪with 99 unique values
[12/30/17 23:17:50] INFO        Feature 5: SibSp is a numerical feature of type int64␣
↪with 7 unique values
[12/30/17 23:17:50] INFO        Feature 6: Parch is a numerical feature of type int64␣
↪with 8 unique values
[12/30/17 23:17:50] INFO        Feature 7: Ticket is a text feature [3:18] with 929␣
↪unique values
[12/30/17 23:17:50] INFO        Feature 7: Ticket => Factorization
[12/30/17 23:17:50] INFO        Feature 8: Fare is a numerical feature of type␣
↪float64 with 282 unique values
[12/30/17 23:17:50] INFO        Feature 9: Cabin is a text feature [1:15] with 187␣
↪unique values
[12/30/17 23:17:50] INFO        Feature 9: Cabin => Factorization
[12/30/17 23:17:50] INFO        Feature 10: Embarked is a text feature [1:1] with 4␣
↪unique values
[12/30/17 23:17:50] INFO        Feature 10: Embarked => Factorization
[12/30/17 23:17:50] INFO        Feature 11: nan_count is a numerical feature of type␣
↪int64 with 3 unique values
[12/30/17 23:17:50] INFO        Feature 12: count_0 is a numerical feature of type␣
↪int64 with 4 unique values
[12/30/17 23:17:50] INFO        Feature 13: count_1 is a numerical feature of type␣
↪int64 with 4 unique values
[12/30/17 23:17:50] INFO        Feature 14: count_2 is a numerical feature of type␣
↪int64 with 4 unique values
[12/30/17 23:17:50] INFO        Feature 15: count_3 is a numerical feature of type␣
↪int64 with 4 unique values
[12/30/17 23:17:50] INFO        Feature 16: count_4 is a numerical feature of type␣
↪int64 with 3 unique values
[12/30/17 23:17:50] INFO        Feature 17: count_5 is a numerical feature of type␣
↪int64 with 2 unique values
[12/30/17 23:17:50] INFO        Feature 18: count_6 is a numerical feature of type␣
↪int64 with 2 unique values
[12/30/17 23:17:50] INFO        Feature 19: count_7 is a numerical feature of type␣
↪int64 with 2 unique values
[12/30/17 23:17:50] INFO        Feature 20: count_8 is a numerical feature of type␣
↪int64 with 3 unique values
[12/30/17 23:17:50] INFO        Feature 21: count_9 is a numerical feature of type␣
↪int64 with 3 unique values
[12/30/17 23:17:50] INFO        New Feature Count : 21
[12/30/17 23:17:50] INFO        Scaling Base Features
[12/30/17 23:17:50] INFO        Creating NumPy Features
[12/30/17 23:17:50] INFO        NumPy Feature: sum
[12/30/17 23:17:50] INFO        NumPy Feature: mean
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:17:50] INFO        NumPy Feature: standard deviation
[12/30/17 23:17:50] INFO        NumPy Feature: variance
[12/30/17 23:17:50] INFO        NumPy Feature Count : 4
[12/30/17 23:17:50] INFO        New Feature Count : 25
[12/30/17 23:17:50] INFO        Creating Clustering Features
[12/30/17 23:17:50] INFO        Cluster Minimum   : 3
[12/30/17 23:17:50] INFO        Cluster Maximum   : 30
[12/30/17 23:17:50] INFO        Cluster Increment : 3
[12/30/17 23:17:50] INFO        k = 3
[12/30/17 23:17:50] INFO        k = 6
[12/30/17 23:17:50] INFO        k = 9
[12/30/17 23:17:50] INFO        k = 12
[12/30/17 23:17:50] INFO        k = 15
[12/30/17 23:17:50] INFO        k = 18
```

As AlphaPy runs, you can see the number of new features that are generated along the way, depending on which features you selected in the `features` section of the `model.yml` file. For interactions, you specify the polynomial degree and the percentage of the interactions that you would like to retain in the model. Be careful of the polynomial degree, as the number of interaction terms is exponential.

Listing 4: **alphapy.log**

```
[12/30/17 23:17:50] INFO        k = 21
[12/30/17 23:17:50] INFO        k = 24
[12/30/17 23:17:50] INFO        k = 27
[12/30/17 23:17:51] INFO        k = 30
[12/30/17 23:17:51] INFO        Clustering Feature Count : 10
[12/30/17 23:17:51] INFO        New Feature Count : 35
[12/30/17 23:17:51] INFO        Saving New Features in Model
[12/30/17 23:17:51] INFO        Creating Interactions
[12/30/17 23:17:51] INFO        Initial Feature Count  : 35
[12/30/17 23:17:51] INFO        Generating Polynomial Features
[12/30/17 23:17:51] INFO        Interaction Percentage : 10
[12/30/17 23:17:51] INFO        Polynomial Degree      : 5
[12/30/17 23:17:51] INFO        Polynomial Feature Count : 15
```

10.4 Feature Selection

There are two types of feature selection:

- Univariate Selection
- Recursive Feature Elimination (RFE)

Univariate selection finds the informative features based on a percentile of the highest scores, using a scoring function such as ANOVA F-Scores or Chi-squared statistics. There are scoring functions for both classification and regression.

RFE is more time-consuming, but has cross-validation with a configurable scoring function and step size. We also recommend using a seed for reproducible results, as the resulting support vector (a ranking of the features) can vary dramatically across runs.

Listing 5: **alphapy.log**

```
[12/30/17 23:17:51] INFO        Getting Class Weights
[12/30/17 23:17:51] INFO        Class Weight for target Survived [1]: 1.605263
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:17:51] INFO        Getting All Estimators
[12/30/17 23:17:51] INFO        Algorithm Configuration
```

10.5 Model Estimation

A classification model is highly dependent on the class proportions. If you're trying to predict a rare pattern with high accuracy, then training for accuracy will be useless because a dumb classifier could just predict the majority class and be right most of the time. As a result, **AlphaPy** gives data scientists the ability to undersample majority classes or oversample minority classes. There are even techniques that combine the two, e.g., SMOTE or ensemble sampling.

Before estimation, we need to apply sampling and possibly shuffling to improve cross-validation. For example, time series data is ordered, and you may want to eliminate that dependency.

At the beginning of the estimation phase, we read in all of the algorithms from the `algos.yml` file and then select those algorithms used in this particular model. The process is iterative for each algorithm: initial fit, feature selection, grid search, and final fit.

Listing 6: **alphapy.log**

```
[12/30/17 23:17:51] INFO        New Total Feature Count  : 50
[12/30/17 23:17:51] INFO        Saving New Features in Model
[12/30/17 23:17:51] INFO        Removing Low-Variance Features
[12/30/17 23:17:51] INFO        Low-Variance Threshold  : 0.10
[12/30/17 23:17:51] INFO        Original Feature Count  : 50
[12/30/17 23:17:51] INFO        Reduced Feature Count   : 50
[12/30/17 23:17:51] INFO        Saving New Features in Model
[12/30/17 23:17:51] INFO        Skipping Shuffling
[12/30/17 23:17:51] INFO        Skipping Sampling
[12/30/17 23:17:51] INFO        Getting Class Weights
[12/30/17 23:17:51] INFO        Class Weight for target Survived [1]: 1.605263
[12/30/17 23:17:51] INFO        Getting All Estimators
[12/30/17 23:17:51] INFO        Algorithm Configuration
[12/30/17 23:17:51] INFO        Selecting Models
[12/30/17 23:17:51] INFO        Algorithm: RF
[12/30/17 23:17:51] INFO        Fitting Initial Model
[12/30/17 23:17:51] INFO        Recursive Feature Elimination with CV
[12/30/17 23:18:14] INFO        RFECV took 22.72 seconds for step 3 and 3 folds
[12/30/17 23:18:14] INFO        Algorithm: RF, Selected Features: 20, Ranking: [ 2  1␣
↪ 1  1  5  9  1  1  2  6  8  7  8  6  7 10 11 11 11 10 10  1  1  1  1
  9  6  9  5  1  5  4  2  4  1  1  1  3  7  1  1  8  1  1  4  1  1  3  3  1]
[12/30/17 23:18:14] INFO        Randomized Grid Search
[12/30/17 23:19:08] INFO        Grid Search took 54.03 seconds for 50 candidate␣
↪parameter settings.
[12/30/17 23:19:08] INFO        Model with rank: 1
[12/30/17 23:19:08] INFO        Mean validation score: 0.863 (std: 0.014)
[12/30/17 23:19:08] INFO        Parameters: {'est__n_estimators': 501, 'est__min_
↪samples_split': 5, 'est__min_samples_leaf': 3, 'est__max_depth': 7, 'est__criterion
↪': 'entropy', 'est__bootstrap': True}
[12/30/17 23:19:08] INFO        Model with rank: 2
[12/30/17 23:19:08] INFO        Mean validation score: 0.862 (std: 0.015)
[12/30/17 23:19:08] INFO        Parameters: {'est__n_estimators': 201, 'est__min_
↪samples_split': 10, 'est__min_samples_leaf': 2, 'est__max_depth': 7, 'est__criterion
↪': 'entropy', 'est__bootstrap': True}
[12/30/17 23:19:08] INFO        Model with rank: 3
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:19:08] INFO        Mean validation score: 0.861 (std: 0.014)
[12/30/17 23:19:08] INFO        Parameters: {'est__n_estimators': 101, 'est__min_
↪samples_split': 2, 'est__min_samples_leaf': 3, 'est__max_depth': 7, 'est__criterion
↪': 'entropy', 'est__bootstrap': True}
[12/30/17 23:19:08] INFO        Algorithm: RF, Best Score: 0.8627, Best Parameters: {
↪'est__n_estimators': 501, 'est__min_samples_split': 5, 'est__min_samples_leaf': 3,
↪'est__max_depth': 7, 'est__criterion': 'entropy', 'est__bootstrap': True}
[12/30/17 23:19:08] INFO        Final Model Predictions for RF
[12/30/17 23:19:08] INFO        Skipping Calibration
[12/30/17 23:19:08] INFO        Making Predictions
[12/30/17 23:19:09] INFO        Predictions Complete
[12/30/17 23:19:09] INFO        Algorithm: XGB
[12/30/17 23:19:09] INFO        Fitting Initial Model
[12/30/17 23:19:09] INFO        No RFE Available for XGB
[12/30/17 23:19:09] INFO        Randomized Grid Search
[12/30/17 23:19:32] INFO        Grid Search took 23.44 seconds for 50 candidate␣
↪parameter settings.
[12/30/17 23:19:32] INFO        Model with rank: 1
[12/30/17 23:19:32] INFO        Mean validation score: 0.863 (std: 0.020)
[12/30/17 23:19:32] INFO        Parameters: {'est__subsample': 0.6, 'est__n_estimators
↪': 21, 'est__min_child_weight': 1.1, 'est__max_depth': 12, 'est__learning_rate': 0.
↪1, 'est__colsample_bytree': 0.7}
[12/30/17 23:19:32] INFO        Model with rank: 2
[12/30/17 23:19:32] INFO        Mean validation score: 0.856 (std: 0.014)
[12/30/17 23:19:32] INFO        Parameters: {'est__subsample': 0.5, 'est__n_estimators
↪': 51, 'est__min_child_weight': 1.0, 'est__max_depth': 8, 'est__learning_rate': 0.
↪01, 'est__colsample_bytree': 0.7}
[12/30/17 23:19:32] INFO        Model with rank: 3
```

10.6 Grid Search

There are two types of grid search for model hyperparameters:

- Full Grid Search
- Randomized Grid Search

A full grid search is exhaustive and can be the most time-consuming task of the pipeline. We recommend that you save the full grid search until the end of your model development, and in the interim use a randomized grid search with a fixed number of iterations. The results of the top 3 grid searches are ranked by mean validation score, and the best estimator is saved for making predictions.

Listing 7: **alphapy.log**

```
[12/30/17 23:17:51] INFO        Selecting Models
[12/30/17 23:17:51] INFO        Algorithm: RF
[12/30/17 23:17:51] INFO        Fitting Initial Model
[12/30/17 23:17:51] INFO        Recursive Feature Elimination with CV
[12/30/17 23:18:14] INFO        RFECV took 22.72 seconds for step 3 and 3 folds
[12/30/17 23:18:14] INFO        Algorithm: RF, Selected Features: 20, Ranking: [ 2  1␣
↪ 1  1  5  9  1  1  2  6  8  7  8  6  7 10 11 11 11 10 10  1  1  1  1
  9  6  9  5  1  5  4  2  4  1  1  1  3  7  1  1  8  1  1  4  1  1  3  3  1]
[12/30/17 23:18:14] INFO        Randomized Grid Search
[12/30/17 23:19:08] INFO        Grid Search took 54.03 seconds for 50 candidate␣
↪parameter settings.
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:19:08] INFO        Model with rank: 1
[12/30/17 23:19:08] INFO        Mean validation score: 0.863 (std: 0.014)
[12/30/17 23:19:08] INFO        Parameters: {'est__n_estimators': 501, 'est__min_
↪samples_split': 5, 'est__min_samples_leaf': 3, 'est__max_depth': 7, 'est__criterion
↪': 'entropy', 'est__bootstrap': True}
```

10.7 Model Evaluation

Each model is evaluated using all of the metrics available in *scikit-learn* to give you a sense of how other scoring functions compare. Metrics are calculated on the training data for every algorithm. If test labels are present, then metrics are also calculated for the test data.

Listing 8: **alphapy.log**

```
[12/30/17 23:19:32] INFO        Final Model Predictions for XGB
[12/30/17 23:19:32] INFO        Skipping Calibration
[12/30/17 23:19:32] INFO        Making Predictions
[12/30/17 23:19:32] INFO        Predictions Complete
[12/30/17 23:19:32] INFO        Blending Models
[12/30/17 23:19:32] INFO        Blending Start: 2017-12-30 23:19:32.734086
[12/30/17 23:19:32] INFO        Blending Complete: 0:00:00.010781
[12/30/17 23:19:32]␣
↪INFO        ================================================================================
[12/30/17 23:19:32] INFO        Metrics for: Partition.train
[12/30/17 23:19:32] INFO        ------------------------------------------------------
↪--------------------------
[12/30/17 23:19:32] INFO        Algorithm: RF
[12/30/17 23:19:32] INFO        accuracy: 0.895622895623
[12/30/17 23:19:32] INFO        adjusted_rand_score: 0.623145355109
[12/30/17 23:19:32] INFO        average_precision: 0.939127507197
[12/30/17 23:19:32] INFO        confusion_matrix: [[530  19]
 [ 74 268]]
[12/30/17 23:19:32] INFO        explained_variance: 0.574782432706
[12/30/17 23:19:32] INFO        f1: 0.852146263911
[12/30/17 23:19:32] INFO        mean_absolute_error: 0.104377104377
[12/30/17 23:19:32] INFO        median_absolute_error: 0.0
[12/30/17 23:19:32] INFO        neg_log_loss: 0.299193724458
[12/30/17 23:19:32] INFO        neg_mean_squared_error: 0.104377104377
[12/30/17 23:19:32] INFO        precision: 0.933797909408
[12/30/17 23:19:32] INFO        r2: 0.558671268335
[12/30/17 23:19:32] INFO        recall: 0.783625730994
[12/30/17 23:19:32] INFO        roc_auc: 0.954665047561
[12/30/17 23:19:32] INFO        ------------------------------------------------------
↪--------------------------
[12/30/17 23:19:32] INFO        Algorithm: XGB
[12/30/17 23:19:32] INFO        accuracy: 0.915824915825
[12/30/17 23:19:32] INFO        adjusted_rand_score: 0.689583531462
[12/30/17 23:19:32] INFO        average_precision: 0.958117084885
[12/30/17 23:19:32] INFO        confusion_matrix: [[532  17]
 [ 58 284]]
[12/30/17 23:19:32] INFO        explained_variance: 0.653042746514
[12/30/17 23:19:32] INFO        f1: 0.883359253499
[12/30/17 23:19:32] INFO        mean_absolute_error: 0.0841750841751
[12/30/17 23:19:32] INFO        median_absolute_error: 0.0
```

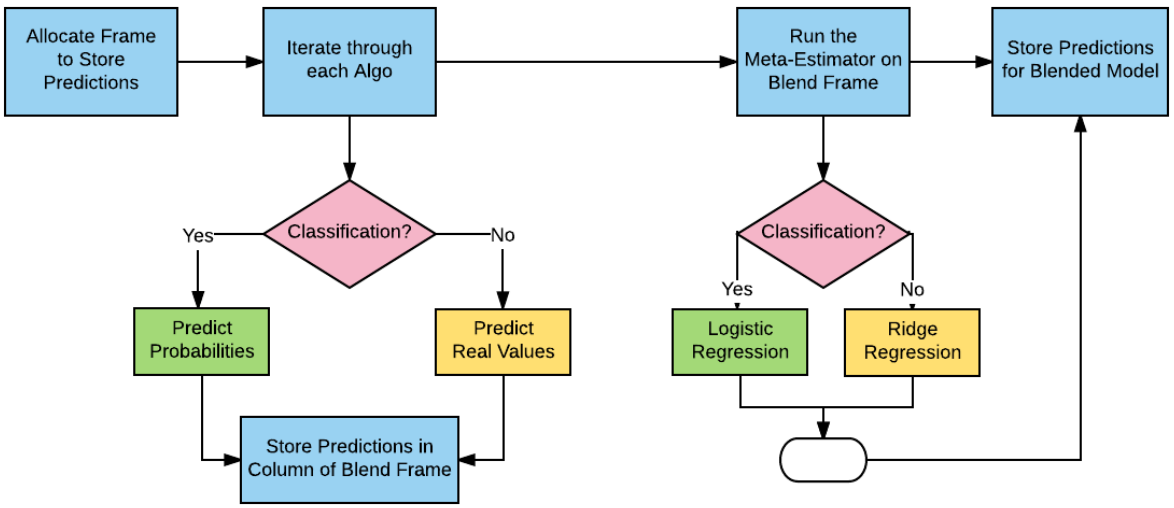
(continues on next page)

(continued from previous page)

```
[12/30/17 23:19:32] INFO        neg_log_loss: 0.295991903662
[12/30/17 23:19:32] INFO        neg_mean_squared_error: 0.0841750841751
[12/30/17 23:19:32] INFO        precision: 0.943521594684
```

10.8 Model Selection

10.8.1 Blended Model



Listing 9: **alphapy.log**

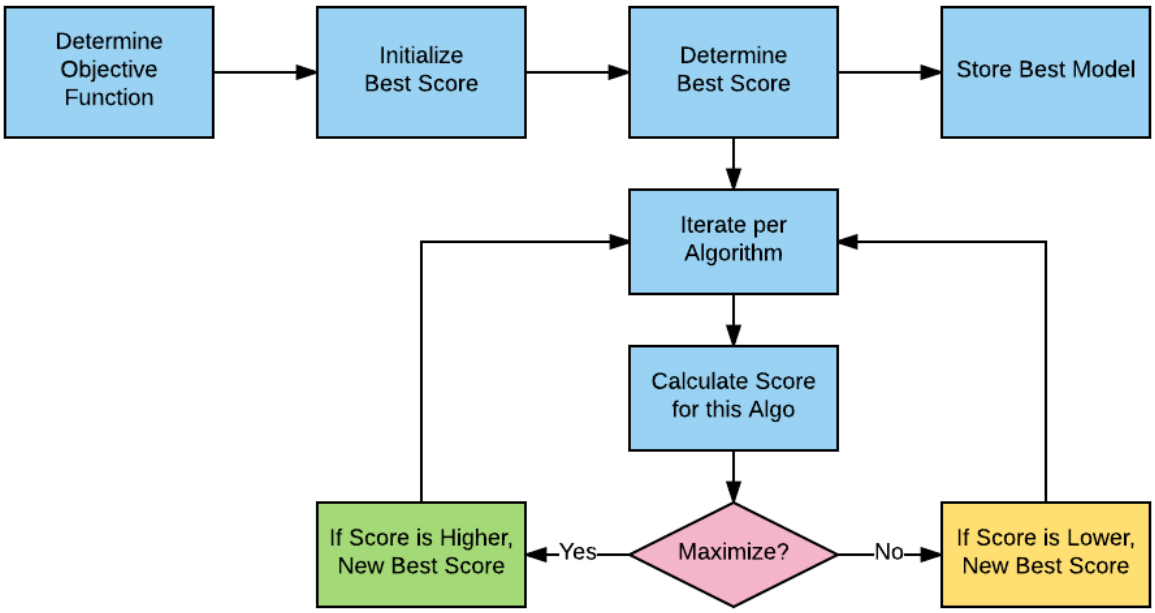
```
[12/30/17 23:19:32] INFO        Mean validation score: 0.855 (std: 0.023)
[12/30/17 23:19:32] INFO        Parameters: {'est__subsample': 0.5, 'est__n_estimators
↪': 21, 'est__min_child_weight': 1.0, 'est__max_depth': 7, 'est__learning_rate': 0.
↪05, 'est__colsample_bytree': 0.6}
[12/30/17 23:19:32] INFO        Algorithm: XGB, Best Score: 0.8627, Best Parameters: {
↪'est__subsample': 0.6, 'est__n_estimators': 21, 'est__min_child_weight': 1.1, 'est__
↪max_depth': 12, 'est__learning_rate': 0.1, 'est__colsample_bytree': 0.7}
```

10.8.2 Best Model

The best model is selected from the score of:

- a model for each algorithm, and
- a *blended* model

Depending on the scoring function, best model selection is based on whether the score must be minimized or maximized. For example, the Area Under the Curve (AUC) must be maximized, and negative log loss must be minimized.



When more than one algorithm is scored in the estimation stage, the final step is to combine the predictions of each one and create the blended model, i.e., the predictions from the independent models are used as training features. For classification, AlphaPy uses logistic regression, and for regression, we use ridge regression.

Listing 10: **alphapy.log**

```
[12/30/17 23:19:32] INFO        precision: 0.943521594684
[12/30/17 23:19:32] INFO        r2: 0.644089732528
[12/30/17 23:19:32] INFO        recall: 0.830409356725
[12/30/17 23:19:32] INFO        roc_auc: 0.971127728246
[12/30/17 23:19:32]␣
↪INFO        ================================================================================
[12/30/17 23:19:32] INFO        Metrics for: Partition.test
[12/30/17 23:19:32] INFO        No labels for generating Partition.test metrics
[12/30/17 23:19:32]␣
↪INFO        ================================================================================
[12/30/17 23:19:32] INFO        Selecting Best Model
[12/30/17 23:19:32] INFO        Scoring for: Partition.train
```

10.9 Plot Generation

The user has the option of generating the following plots:

- Calibration Plot
- Confusion Matrix
- Feature Importances
- Learning Curve
- ROC Curve

All plots are saved to the `plots` directory of your project.

Listing 11: **alphapy.log**

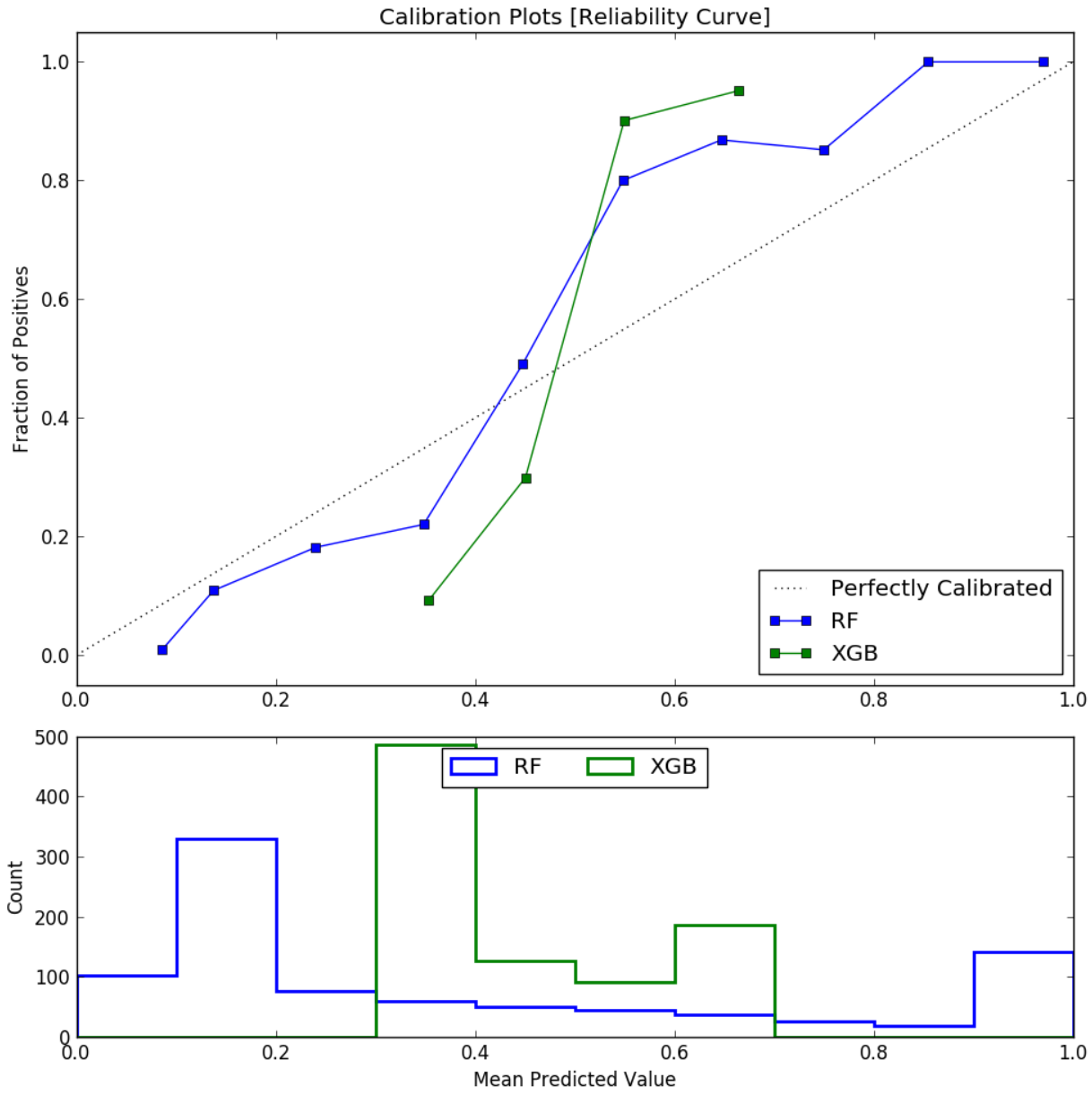
```
[12/30/17 23:19:32] INFO        Scoring for: Partition.train
[12/30/17 23:19:32] INFO        Best Model Selection Start: 2017-12-30 23:19:32.779281
[12/30/17 23:19:32] INFO        Scoring RF Model
[12/30/17 23:19:32] INFO        Scoring XGB Model
[12/30/17 23:19:32] INFO        Scoring BLEND Model
[12/30/17 23:19:32] INFO        Best Model is XGB with a roc_auc score of 0.9711
[12/30/17 23:19:32] INFO        Best Model Selection Complete: 0:00:00.000801
[12/30/17 23:19:32]␣
↪INFO        ================================================================================
[12/30/17 23:19:32] INFO        Generating Plots for partition: train
[12/30/17 23:19:32] INFO        Generating Calibration Plot
[12/30/17 23:19:32] INFO        Calibration for Algorithm: RF
[12/30/17 23:19:32] INFO        Calibration for Algorithm: XGB
[12/30/17 23:19:32] INFO        Writing plot to ./plots/calibration_train.png
[12/30/17 23:19:33] INFO        Generating Confusion Matrices
[12/30/17 23:19:33] INFO        Confusion Matrix for Algorithm: RF
[12/30/17 23:19:33] INFO        Confusion Matrix:
[12/30/17 23:19:33] INFO        [[530  19]
 [ 74 268]]
[12/30/17 23:19:33] INFO        Writing plot to ./plots/confusion_train_RF.png
[12/30/17 23:19:33] INFO        Confusion Matrix for Algorithm: XGB
[12/30/17 23:19:33] INFO        Confusion Matrix:
[12/30/17 23:19:33] INFO        [[532  17]
 [ 58 284]]
[12/30/17 23:19:33] INFO        Writing plot to ./plots/confusion_train_XGB.png
[12/30/17 23:19:33] INFO        Generating ROC Curves
[12/30/17 23:19:33] INFO        ROC Curve for Algorithm: RF
[12/30/17 23:19:33] INFO        ROC Curve for Algorithm: XGB
[12/30/17 23:19:33] INFO        Writing plot to ./plots/roc_curve_train.png
[12/30/17 23:19:33] INFO        Generating Learning Curves
[12/30/17 23:19:33] INFO        Algorithm Configuration
[12/30/17 23:19:33] INFO        Learning Curve for Algorithm: RF
[12/30/17 23:19:35] INFO        Writing plot to ./plots/learning_curve_train_RF.png
[12/30/17 23:19:35] INFO        Learning Curve for Algorithm: XGB
[12/30/17 23:19:36] INFO        Writing plot to ./plots/learning_curve_train_XGB.png
[12/30/17 23:19:36] INFO        Generating Feature Importance Plots
[12/30/17 23:19:36] INFO        Feature Importances for Algorithm: RF
[12/30/17 23:19:36] INFO        Feature Ranking:
[12/30/17 23:19:36] INFO        1. Feature 2 (0.106345)
[12/30/17 23:19:36] INFO        2. Feature 36 (0.074083)
[12/30/17 23:19:36] INFO        3. Feature 39 (0.055330)
[12/30/17 23:19:36] INFO        4. Feature 3 (0.050339)
[12/30/17 23:19:36] INFO        5. Feature 7 (0.049228)
[12/30/17 23:19:36] INFO        6. Feature 23 (0.044868)
[12/30/17 23:19:36] INFO        7. Feature 22 (0.042925)
[12/30/17 23:19:36] INFO        8. Feature 42 (0.042850)
[12/30/17 23:19:36] INFO        9. Feature 21 (0.039954)
[12/30/17 23:19:36] INFO        10. Feature 24 (0.038563)
[12/30/17 23:19:36] INFO        Writing plot to ./plots/feature_importance_train_RF.
↪png
[12/30/17 23:19:36] INFO        Feature Importances for Algorithm: XGB
[12/30/17 23:19:36] INFO        Feature Ranking:
[12/30/17 23:19:36] INFO        1. Feature 2 (0.142857)
[12/30/17 23:19:36] INFO        2. Feature 3 (0.120301)
```

(continues on next page)

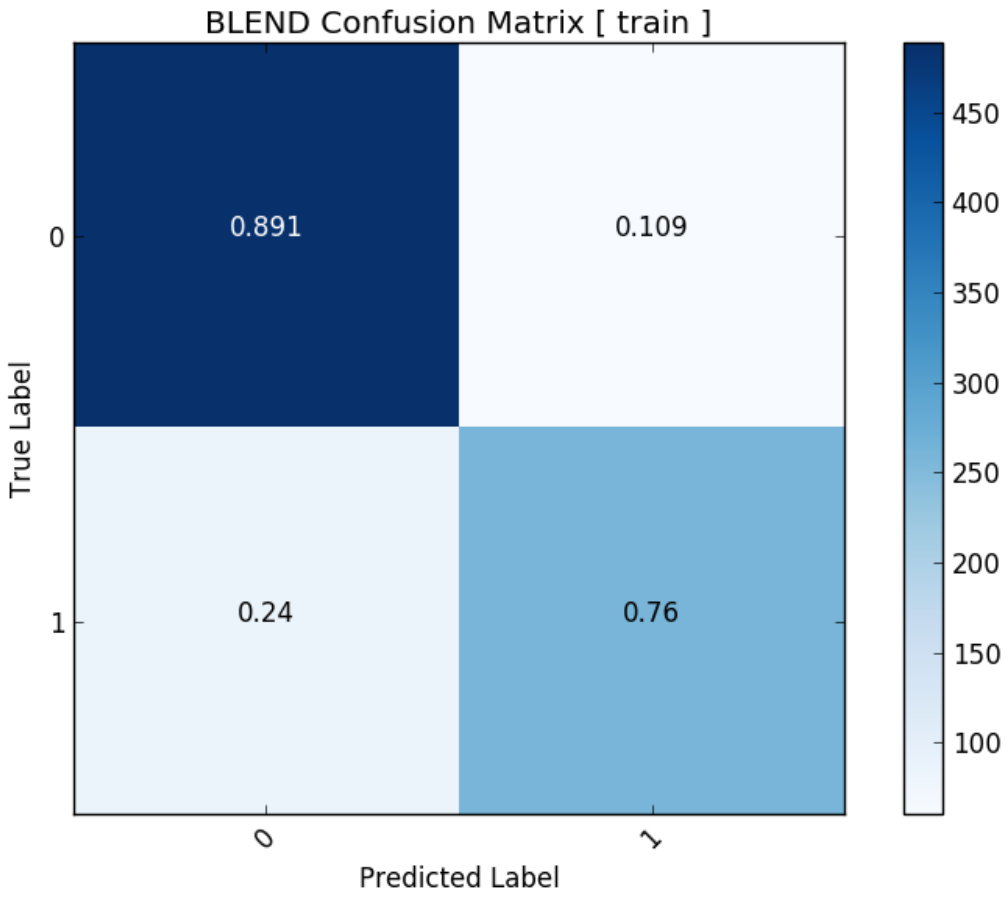
(continued from previous page)

```
[12/30/17 23:19:36] INFO        3. Feature 0 (0.116541)
[12/30/17 23:19:36] INFO        4. Feature 7 (0.109023)
[12/30/17 23:19:36] INFO        5. Feature 31 (0.105263)
```

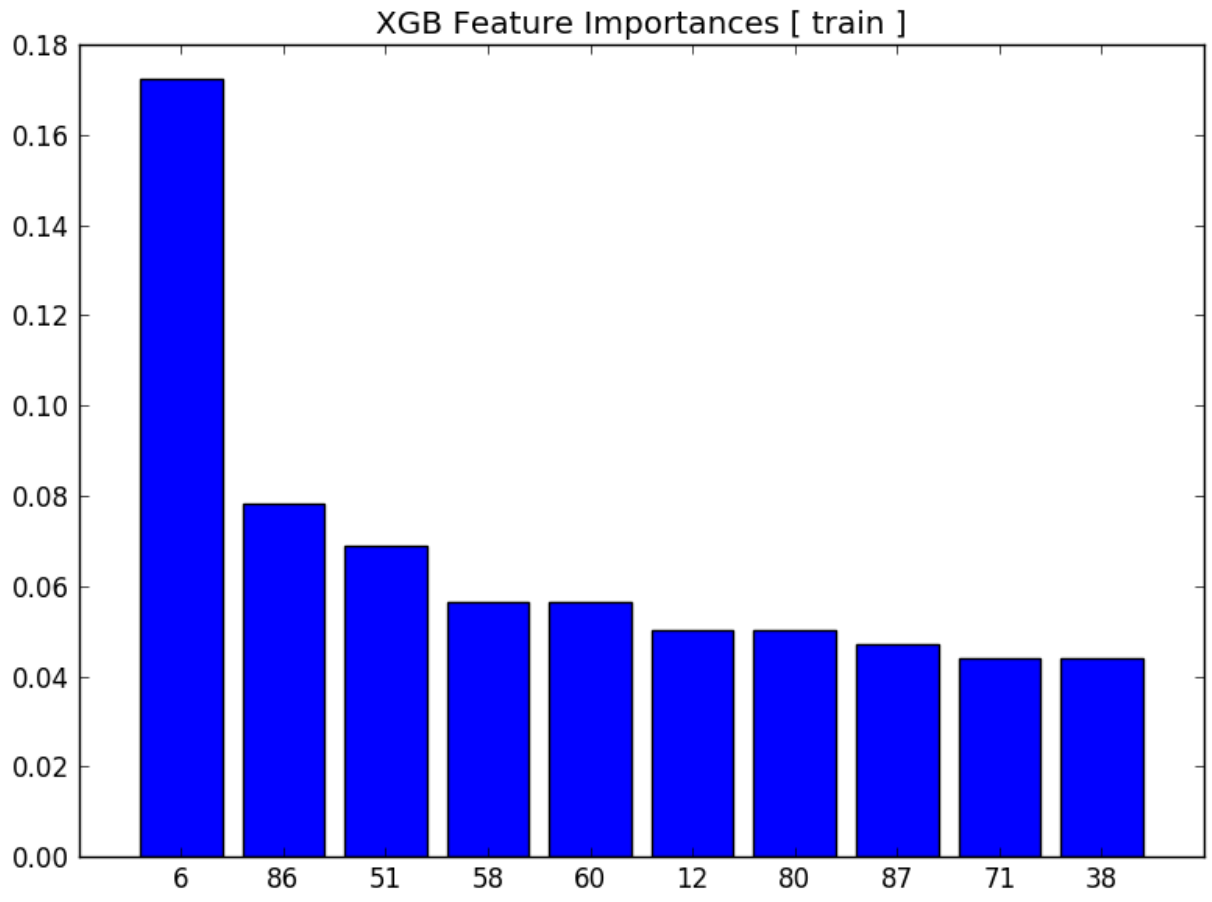
10.9.1 Calibration Plot



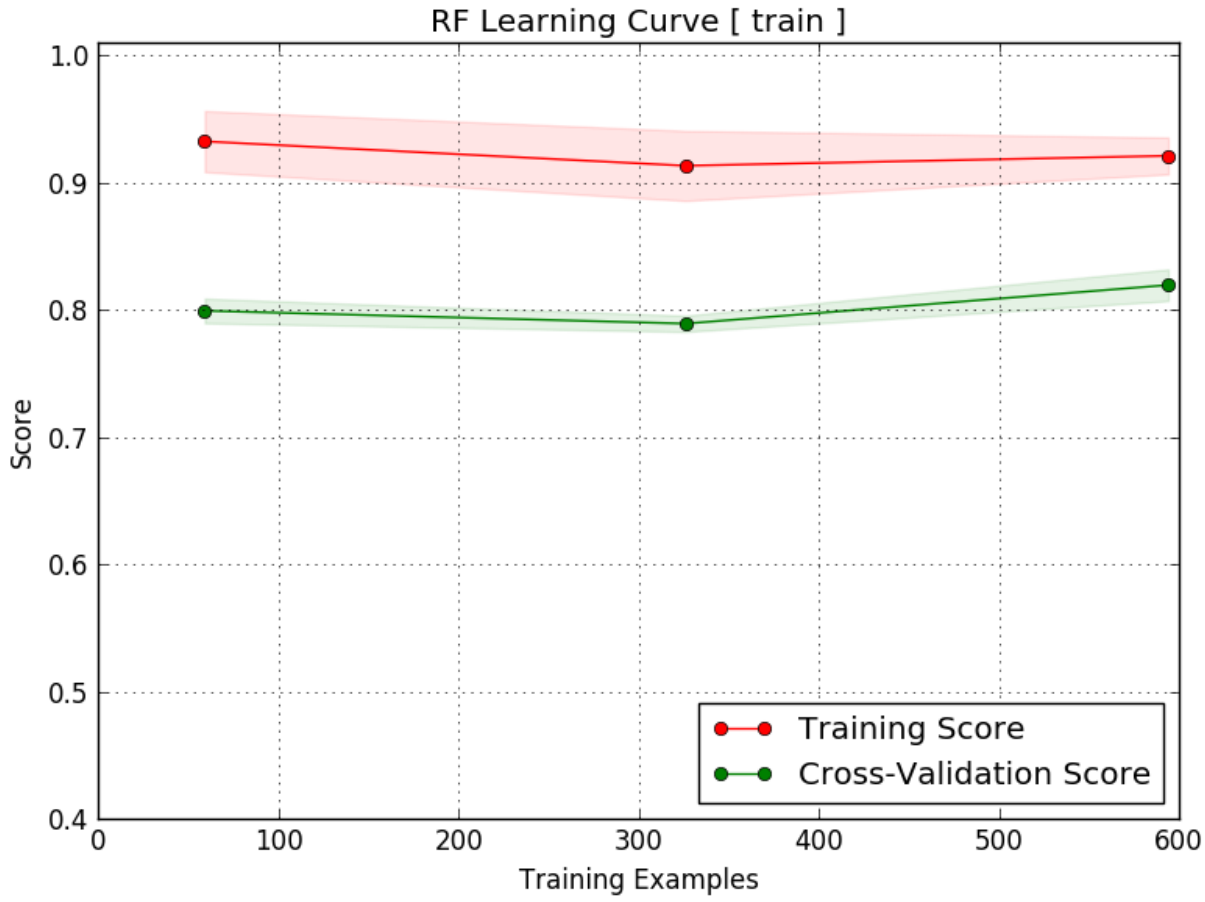
10.9.2 Confusion Matrix



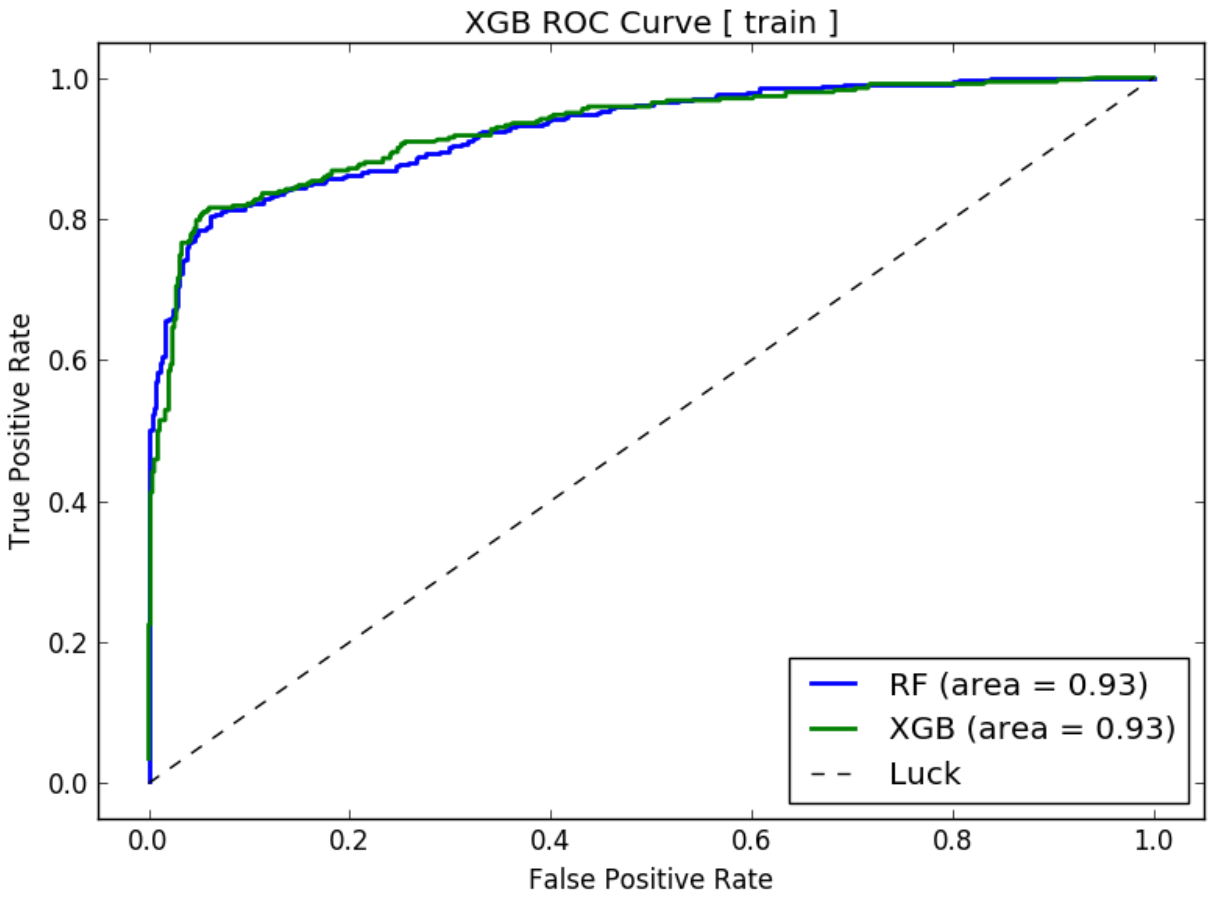
10.9.3 Feature Importances



10.9.4 Learning Curve



10.9.5 ROC Curve



10.10 Final Results

- The model object is stored in Pickle (.pkl) format in the `models` directory of the project. The model is loaded later in prediction mode.
- The feature map is stored in Pickle (.pkl) format in the `models` directory. The feature map is restored for prediction mode.
- Predictions are stored in the project's `output` directory.
- Sorted rankings of predictions are stored in `output`.
- Any submission files are stored in `output`.

Listing 12: **alphapy.log**

```
[12/30/17 23:19:36] INFO        6. Feature 23 (0.075188)
[12/30/17 23:19:36] INFO        7. Feature 40 (0.071429)
[12/30/17 23:19:36] INFO        8. Feature 8 (0.033835)
[12/30/17 23:19:36] INFO        9. Feature 4 (0.030075)
[12/30/17 23:19:36] INFO        10. Feature 9 (0.030075)
```

(continues on next page)

(continued from previous page)

```
[12/30/17 23:19:36] INFO        Writing plot to ./plots/feature_importance_train_XGB.
↪png
[12/30/17 23:19:36]␣
↪INFO        ================================================================================
[12/30/17 23:19:36] INFO        Saving Model Predictor
[12/30/17 23:19:36] INFO        Writing model predictor to ./model/model_20171230.pkl
[12/30/17 23:19:36] INFO        Saving Feature Map
[12/30/17 23:19:36] INFO        Writing feature map to ./model/feature_map_20171230.
↪pkl
[12/30/17 23:19:36] INFO        Loading data from ./input/test.csv
```

CHAPTER

ELEVEN

PROJECT STRUCTURE

11.1 Setup

Your initial configuration must have the following directories and files. The directories `config`, `data`, and `input` store input, and the directories `model`, `output`, and `plots` store output:

```
project
├── config
    ├── model.yml
    ├── algos.yml
└── data
└── input
    ├── train.csv
    ├── test.csv
└── model
└── output
└── plots
```

The top-level directory is the main project directory with a unique name. There are six required subdirectories:

config: This directory contains all of the YAML files. At a minimum, it must contain `model.yml` and `algos.yml`.

data: If required, any data for the domain pipeline is stored here. Data from this directory will be transformed into `train.csv` and `test.csv` in the `input` directory.

input: The training file `train.csv` and the testing file `test.csv` are stored here. Note that these file names can be named anything as configured in the `model.yml` file.

model: The final model is dumped here as a pickle file in the format `model_[yyyymmdd].pkl`.

output: This directory contains predictions, probabilities, rankings, and any submission files:

- `predictions_[yyyymmdd].csv`
- `probabilities_[yyyymmdd].csv`
- `rankings_[yyyymmdd].csv`
- `submission_[yyyymmdd].csv`

plots: All generated plots are stored here. The file name has the following elements:

- plot name
- 'train' or 'test'
- algorithm abbreviation

- format suffix

For example, a calibration plot for the testing data for all algorithms will be named `calibration_test.png`. The file name for a confusion matrix for XGBoost training data will be `confusion_train_XGB.png`.

11.2 Model Configuration

Here is an example of a model configuration file. It is written in YAML and is divided into logical sections reflecting the stages of the pipeline. Within each section, you can control different aspects for experimenting with model results. Please refer to the following sections for more detail.

Listing 1: **model.yml**

```
project:
    directory          : .
    file_extension     : csv
    submission_file    : 'gender_submission'
    submit_probas      : False

data:
    drop               : ['PassengerId']
    features           : '*'
    sampling           :
        option         : False
        method         : under_random
        ratio          : 0.5
    sentinel           : -1
    separator          : ','
    shuffle            : False
    split              : 0.4
    target             : Survived
    target_value       : 1

model:
    algorithms         : ['RF', 'XGB']
    balance_classes    : True
    calibration        :
        option         : False
        type           : sigmoid
    cv_folds           : 3
    estimators         : 51
    feature_selection  :
        option         : False
        percentage     : 50
        uni_grid       : [5, 10, 15, 20, 25]
        score_func     : f_classif
    grid_search        :
        option         : True
        iterations     : 50
        random         : True
        subsample      : False
        sampling_pct   : 0.2
    pvalue_level       : 0.01
    rfe                :
        option         : True
        step           : 3
```

(continues on next page)

(continued from previous page)

```
    scoring_function  : roc_auc
    type              : classification

features:
    clustering        :
        option        : True
        increment     : 3
        maximum       : 30
        minimum       : 3
    counts            :
        option        : True
    encoding          :
        rounding      : 2
        type          : factorize
    factors           : []
    interactions      :
        option        : True
        poly_degree   : 5
        sampling_pct  : 10
    isomap            :
        option        : False
        components    : 2
        neighbors     : 5
    logtransform      :
        option        : False
    numpy             :
        option        : True
    pca               :
        option        : False
        increment     : 1
        maximum       : 10
        minimum       : 2
        whiten        : False
    scaling           :
        option        : True
        type          : standard
    scipy             :
        option        : False
    text              :
        ngrams        : 3
        vectorize     : False
    tsne              :
        option        : False
        components    : 2
        learning_rate : 1000.0
        perplexity    : 30.0
    variance          :
        option        : True
        threshold     : 0.1

pipeline:
    number_jobs       : -1
    seed              : 42
    verbosity         : 0

plots:
    calibration       : True
```

(continues on next page)

(continued from previous page)

```
    confusion_matrix   : True
    importances        : True
    learning_curve     : True
    roc_curve          : True

xgboost:
    stopping_rounds    : 20
```

11.2.1 Project Section

The `project` section has the following keys:

directory: The full specification of the project location

file_extension: The extension is usually `csv` but could also be `tsv` or other types using different delimiters between values

submission_file: The file name of the submission template, which is usually provided in Kaggle competitions

submit_probas: Set the value to `True` if submitting probabilities, or set to `False` if the predictions are the actual labels or real values.

Listing 2: **model.yml**

```
project:
    directory          : .
    file_extension     : csv
    submission_file    : 'gender_submission'
    submit_probas      : False
```

Warning: If you do not supply a value on the right-hand side of the colon [:], then Python will interpret that key as having a `None` value, which is correct. Do not spell out *None*; otherwise, the value will be interpreted as the string 'None'.

11.2.2 Data Section

The `data` section has the following keys:

drop: A list of features to be dropped from the data frame

features: A list of features for training. `'*'` means all features will be used in training.

sampling: Resample imbalanced classes with one of the sampling methods in `alphapy.data.SamplingMethod`

sentinel: The designated value to replace any missing values

separator: The delimiter separating values in the training and test files

shuffle: If `True`, randomly shuffle the data.

split: The proportion of data to include in training, which is a fraction between 0 and 1

target: The name of the feature that designates the label to predict

target_value: The value of the target label to predict

Listing 3: **model.yml**

```
data:
    drop              : ['PassengerId']
    features          : '*'
    sampling          :
        option        : False
        method        : under_random
        ratio         : 0.5
    sentinel          : -1
    separator         : ','
    shuffle           : False
    split             : 0.4
    target            : Survived
    target_value      : 1
```

11.2.3 Model Section

The `model` section has the following keys:

algorithms: The list of algorithms to test for model selection. Refer to *Algorithms Configuration* for the abbreviation codes.

balance_classes: If `True`, calculate sample weights to offset the majority class when training a model.

calibration: Calibrate final probabilities for a classification. Refer to the scikit-learn documentation for Calibration.

cv_folds: The number of folds for cross-validation

estimators: The number of estimators to be used in the machine learning algorithm, e.g., the number of trees in a random forest

feature_selection: Perform univariate feature selection based on percentile. Refer to the scikit-learn documentation for FeatureSelection.

grid_search: The grid search is either random with a fixed number of iterations, or it is a full grid search. Refer to the scikit-learn documentation for GridSearch.

pvalue_level: The p-value threshold to determine whether or not a numerical feature is normally distributed.

rfe: Perform Recursive Feature Elimination (RFE). Refer to the scikit-learn documentation for RecursiveFeatureElimination.

scoring_function: The scoring function is an objective function for model evaluation. Use one of the values in ScoringFunction.

type: The model type is either `classification` or `regression`.

Listing 4: **model.yml**

```
model:
    algorithms        : ['RF', 'XGB']
    balance_classes   : True
    calibration       :
        option        : False
        type          : sigmoid
    cv_folds          : 3
    estimators        : 51
```

(continues on next page)

(continued from previous page)

```
    feature_selection :
        option        : False
        percentage    : 50
        uni_grid      : [5, 10, 15, 20, 25]
        score_func    : f_classif
    grid_search       :
        option        : True
        iterations    : 50
        random        : True
        subsample     : False
        sampling_pct  : 0.2
    pvalue_level      : 0.01
    rfe               :
        option        : True
        step          : 3
    scoring_function  : roc_auc
    type              : classification
```

11.2.4 Features Section

The `features` section has the following keys:

clustering: For clustering, specify the minimum and maximum number of clusters and the increment from min-to-max.

counts: Create features that record counts of the NA values, zero values, and the digits 1-9 in each row.

encoding: Encode factors from features, selecting an encoding type and any rounding if necessary. Refer to `alphapy.features.Encoders` for the encoding type.

factors: The list of features that are factors.

interactions: Calculate polynomical interactions of a given degree, and select the percentage of interactions included in the feature set.

isomap: Use isomap embedding. Refer to isomap.

logtransform: For numerical features that do not fit a normal distribution, perform a log transformation.

numpy: Calculate the total, mean, standard deviation, and variance of each row.

pca: For Principal Component Analysis, specify the minimum and maximum number of components, the increment from min-to-max, and whether or not whitening is applied.

scaling: To scale features, specify `standard` or `minmax`.

scipy: Calculate skew and kurtosis for row distributions.

text: If there are text features, then apply vectorization and TF-IDF. If vectorization does not work, then apply factorization.

tsne: Perform t-distributed Stochastic Neighbor Embedding (TSNE), which can be very memory-intensive. Refer to TSNE.

variance: Remove low-variance features using a specified threshold. Refer to VAR.

Listing 5: **model.yml**

```
features:
    clustering         :
        option         : True
        increment      : 3
        maximum        : 30
        minimum        : 3
    counts             :
        option         : True
    encoding           :
        rounding       : 2
        type           : factorize
    factors            : []
    interactions       :
        option         : True
        poly_degree    : 5
        sampling_pct   : 10
    isomap             :
        option         : False
        components     : 2
        neighbors      : 5
    logtransform       :
        option         : False
    numpy              :
        option         : True
    pca                :
        option         : False
        increment      : 1
        maximum        : 10
        minimum        : 2
        whiten         : False
    scaling            :
        option         : True
        type           : standard
    scipy              :
        option         : False
    text               :
        ngrams         : 3
        vectorize      : False
    tsne               :
        option         : False
        components     : 2
        learning_rate  : 1000.0
        perplexity     : 30.0
    variance           :
        option         : True
        threshold      : 0.1
```

11.2.5 Treatments Section

Treatments are special functions for feature extraction. In the `treatments` section below, we are applying treatments to two features *doji* and *hc*. Within the Python list, we are calling the `runs_test` function of the module *alphapy.features*. The module name is always the first element of the list, and the the function name is always the second element of the list. The remaining elements of the list are the actual parameters to the function.

Listing 6: **model.yml**

```
treatments:
    doji : ['alphapy.features', 'runs_test', ['all'], 18]
    hc   : ['alphapy.features', 'runs_test', ['all'], 18]
```

Here is the code for the `runs_test` function, which calculates runs for Boolean features. For a treatment function, the first and second arguments are always the same. The first argument `f` is the data frame, and the second argument `c` is the column (or feature) to which we are going to apply the treatment. The remaining function arguments correspond to the actual parameters that were specified in the configuration file, in this case `wfuncs` and `window`.

Listing 7: **features.py**

```
def runs_test(f, c, wfuncs, window):
    fc = f[c]
    all_funcs = {'runs'   : runs,
                 'streak' : streak,
                 'rtotal' : rtotal,
                 'zscore' : zscore}
    # use all functions
    if 'all' in wfuncs:
        wfuncs = all_funcs.keys()
    # apply each of the runs functions
    new_features = pd.DataFrame()
    for w in wfuncs:
        if w in all_funcs:
            new_feature = fc.rolling(window=window).apply(all_funcs[w])
            new_feature.fillna(0, inplace=True)
            frames = [new_features, new_feature]
            new_features = pd.concat(frames, axis=1)
        else:
            logger.info("Runs Function %s not found", w)
    return new_features
```

When the `runs_test` function is invoked, a new data frame is created, as multiple feature columns may be generated from a single treatment function. These new features are returned and appended to the original data frame.

11.2.6 Pipeline Section

The `pipeline` section has the following keys:

`number_jobs`: Number of jobs to run in parallel [-1 for all cores]

`seed`: A random seed integer to ensure reproducible results

`verbosity`: The logging level from 0 (no logging) to 10 (highest)

Listing 8: **model.yml**

```
pipeline:
    number_jobs        : -1
    seed               : 42
    verbosity          : 0
```

11.2.7 Plots Section

To turn on the automatic generation of any plot in the `plots` section, simply set the corresponding value to `True`.

Listing 9: **model.yml**

```
plots:
    calibration        : True
    confusion_matrix   : True
    importances        : True
    learning_curve     : True
    roc_curve          : True
```

11.2.8 XGBoost Section

The `xgboost` section has the following keys:

`stopping_rounds`: early stopping rounds for XGBoost

Listing 10: **model.yml**

```
xgboost:
    stopping_rounds    : 20
```

11.3 Algorithms Configuration

Each algorithm has its own section in the `algos.yml` file, e.g., **AB** or **RF**. The following elements are required for every algorithm entry in the YAML file:

`model_type`: Specify `classification` or `regression`

`params` The initial parameters for the first fitting

`grid`: The grid search dictionary for hyperparameter tuning of an estimator. If you are using randomized grid search, then make sure that the total number of grid combinations exceeds the number of random iterations.

`scoring`: Set to `True` if a specific scoring function will be applied.

Note: The parameters `n_estimators`, `n_jobs`, `seed`, and `verbosity` are informed by the `model.yml` file. When the estimators are created, the proper values for these parameters are automatically substituted in the `algos.yml` file on a global basis.

Listing 11: **algos.yml**

```
#
# Algorithms
#

AB:
    # AdaBoost
    model_type : classification
    params     : {"n_estimators" : n_estimators,
                  "random_state" : seed}
    grid       : {"n_estimators" : [10, 50, 100, 150, 200],
                  "learning_rate" : [0.2, 0.5, 0.7, 1.0, 1.5, 2.0],
                  "algorithm" : ['SAMME', 'SAMME.R']}
    scoring    : True

GB:
    # Gradient Boosting
    model_type : classification
    params     : {"n_estimators" : n_estimators,
                  "max_depth" : 3,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {"loss" : ['deviance', 'exponential'],
                  "learning_rate" : [0.05, 0.1, 0.15],
                  "n_estimators" : [50, 100, 200],
                  "max_depth" : [3, 5, 10],
                  "min_samples_split" : [2, 3],
                  "min_samples_leaf" : [1, 2]}
    scoring    : True

GBR:
    # Gradient Boosting Regression
    model_type : regression
    params     : {"n_estimators" : n_estimators,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {}
    scoring    : False

KNN:
    # K-Nearest Neighbors
    model_type : classification
    params     : {"n_jobs" : n_jobs}
    grid       : {"n_neighbors" : [3, 5, 7, 10],
                  "weights" : ['uniform', 'distance'],
                  "algorithm" : ['ball_tree', 'kd_tree', 'brute', 'auto'],
                  "leaf_size" : [10, 20, 30, 40, 50]}
    scoring    : False

KNR:
    # K-Nearest Neighbor Regression
    model_type : regression
    params     : {"n_jobs" : n_jobs}
    grid       : {}
    scoring    : False
```

(continues on next page)

(continued from previous page)

```
LOGR:
    # Logistic Regression
    model_type : classification
    params     : {"random_state" : seed,
                  "n_jobs" : n_jobs,
                  "verbose" : verbosity}
    grid       : {"penalty" : ['l2'],
                  "C" : [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 1e4,␣
↪1e5, 1e6, 1e7],
                  "fit_intercept" : [True, False],
                  "solver" : ['newton-cg', 'lbfgs', 'liblinear', 'sag']}
    scoring    : True

LR:
    # Linear Regression
    model_type : regression
    params     : {"n_jobs" : n_jobs}
    grid       : {"fit_intercept" : [True, False],
                  "normalize" : [True, False],
                  "copy_X" : [True, False]}
    scoring    : False

LSVC:
    # Linear Support Vector Classification
    model_type : classification
    params     : {"C" : 0.01,
                  "max_iter" : 2000,
                  "penalty" : 'l1',
                  "dual" : False,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {"C" : np.logspace(-2, 10, 13),
                  "penalty" : ['l1', 'l2'],
                  "dual" : [True, False],
                  "tol" : [0.0005, 0.001, 0.005],
                  "max_iter" : [500, 1000, 2000]}
    scoring    : False

LSVM:
    # Linear Support Vector Machine
    model_type : classification
    params     : {"kernel" : 'linear',
                  "probability" : True,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {"C" : np.logspace(-2, 10, 13),
                  "gamma" : np.logspace(-9, 3, 13),
                  "shrinking" : [True, False],
                  "tol" : [0.0005, 0.001, 0.005],
                  "decision_function_shape" : ['ovo', 'ovr']}
    scoring    : False

NB:
    # Naive Bayes
    model_type : classification
    params     : {}
    grid       : {"alpha" : [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 2.0, 5.0, 10.0],
```

(continues on next page)

(continued from previous page)

```
                   "fit_prior" : [True, False]}
    scoring     : True

RBF:
    # Radial Basis Function
    model_type : classification
    params     : {"kernel" : 'rbf',
                  "probability" : True,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {"C" : np.logspace(-2, 10, 13),
                  "gamma" : np.logspace(-9, 3, 13),
                  "shrinking" : [True, False],
                  "tol" : [0.0005, 0.001, 0.005],
                  "decision_function_shape" : ['ovo', 'ovr']}
    scoring    : False

RF:
    # Random Forest
    model_type : classification
    params     : {"n_estimators" : n_estimators,
                  "max_depth" : 10,
                  "min_samples_split" : 5,
                  "min_samples_leaf" : 3,
                  "bootstrap" : True,
                  "criterion" : 'entropy',
                  "random_state" : seed,
                  "n_jobs" : n_jobs,
                  "verbose" : verbosity}
    grid       : {"n_estimators" : [21, 51, 101, 201, 501],
                  "max_depth" : [5, 7, 10, 20],
                  "min_samples_split" : [2, 3, 5, 10],
                  "min_samples_leaf" : [1, 2, 3],
                  "bootstrap" : [True, False],
                  "criterion" : ['gini', 'entropy']}
    scoring    : True

RFR:
    # Random Forest Regression
    model_type : regression
    params     : {"n_estimators" : n_estimators,
                  "random_state" : seed,
                  "n_jobs" : n_jobs,
                  "verbose" : verbosity}
    grid       : {}
    scoring    : False

SVM:
    # Support Vector Machine
    model_type : classification
    params     : {"probability" : True,
                  "random_state" : seed,
                  "verbose" : verbosity}
    grid       : {"C" : np.logspace(-2, 10, 13),
                  "gamma" : np.logspace(-9, 3, 13),
                  "shrinking" : [True, False],
                  "tol" : [0.0005, 0.001, 0.005],
```

(continues on next page)

(continued from previous page)

```
                  "decision_function_shape" : ['ovo', 'ovr']}
    scoring     : False

TF_DNN:
    # Google TensorFlow Deep Neural Network
    model_type : classification
    params      : {"feature_columns" : [tf.contrib.layers.real_valued_column("",
↪dimension=4)],
                   "n_classes" : 2,
                   "hidden_units" : [20, 40, 20]}
    grid        : {}
    scoring     : False

XGB:
    # XGBoost Binary
    model_type : classification
    params      : {"objective" : 'binary:logistic',
                   "n_estimators" : n_estimators,
                   "seed" : seed,
                   "max_depth" : 6,
                   "learning_rate" : 0.1,
                   "min_child_weight" : 1.1,
                   "subsample" : 0.9,
                   "colsample_bytree" : 0.9,
                   "nthread" : n_jobs,
                   "silent" : True}
    grid        : {"n_estimators" : [21, 51, 101, 201, 501],
                   "max_depth" : [5, 6, 7, 8, 9, 10, 12, 15, 20],
                   "learning_rate" : [0.01, 0.02, 0.05, 0.1, 0.2],
                   "min_child_weight" : [1.0, 1.1],
                   "subsample" : [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
                   "colsample_bytree" : [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]}
    scoring     : False

XGBM:
    # XGBoost Multiclass
    model_type : multiclass
    params      : {"objective" : 'multi:softmax',
                   "n_estimators" : n_estimators,
                   "seed" : seed,
                   "max_depth" : 10,
                   "learning_rate" : 0.1,
                   "min_child_weight" : 1.1,
                   "subsample" : 0.9,
                   "colsample_bytree" : 0.9,
                   "nthread" : n_jobs,
                   "silent" : True}
    grid        : {}
    scoring     : False

XGBR:
    # XGBoost Regression
    model_type : regression
    params      : {"objective" : 'reg:linear',
                   "n_estimators" : n_estimators,
                   "seed" : seed,
                   "max_depth" : 10,
```

(continues on next page)

(continued from previous page)

```
                  "learning_rate" : 0.1,
                  "min_child_weight" : 1.1,
                  "subsample" : 0.9,
                  "colsample_bytree" : 0.9,
                  "seed" : seed,
                  "nthread" : n_jobs,
                  "silent" : True}
    grid        : {}
    scoring     : False

XT:
    # Extra Trees
    model_type : classification
    params      : {"n_estimators" : n_estimators,
                   "random_state" : seed,
                   "n_jobs" : n_jobs,
                   "verbose" : verbosity}
    grid        : {"n_estimators" : [21, 51, 101, 201, 501, 1001, 2001],
                   "max_features" : ['auto', 'sqrt', 'log2'],
                   "max_depth" : [3, 5, 7, 10, 20, 30],
                   "min_samples_split" : [2, 3],
                   "min_samples_leaf" : [1, 2],
                   "bootstrap" : [True, False],
                   "warm_start" : [True, False]}
    scoring     : True

XTR:
    # Extra Trees Regression
    model_type : regression
    params      : {"n_estimators" : n_estimators,
                   "random_state" : seed,
                   "n_jobs" : n_jobs,
                   "verbose" : verbosity}
    grid        : {}
    scoring     : False
```

11.4 Final Output

This is an example of your file structure after running the pipeline:

```
project
├── alphapy.log
├── config
│   ├── algos.yml
│   ├── model.yml
└── data
└── input
│   ├── test.csv
│   ├── train.csv
└── model
│   ├── feature_map_20170325.pkl
│   ├── model_20170325.pkl
└── output
    ├── predictions_20170325.csv
```

(continues on next page)

(continued from previous page)

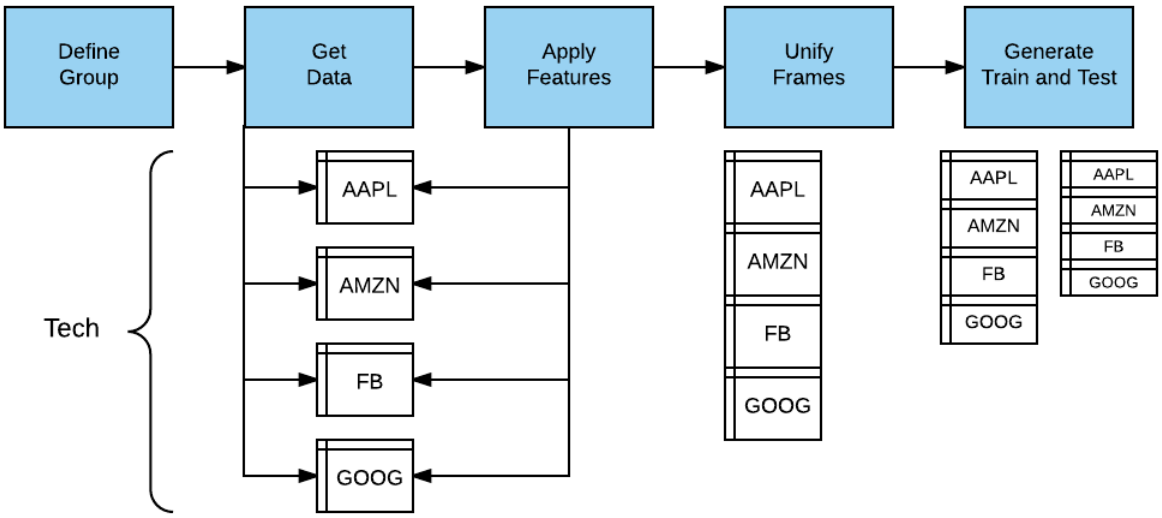
```
    ├── probabilities_20170325.csv
    ├── rankings_20170325.csv
    ├── submission_20170325.csv
└── plots
    ├── calibration_train.png
    ├── confusion_train_RF.png
    ├── confusion_train_XGB.png
    ├── feature_importance_train_RF.png
    ├── feature_importance_train_XGB.png
    ├── learning_curve_train_RF.png
    ├── learning_curve_train_XGB.png
    ├── roc_curve_train.png
```

CHAPTER

TWELVE

MARKETFLOW

MarketFlow transforms financial market data into machine learning models for making market predictions. The platform gets stock price data from Yahoo Finance (end-of-day) and Google Finance (intraday), transforming the data into canonical form for training and testing. MarketFlow is powerful because you can easily apply new features to groups of stocks simultaneously using our *Variable Definition Language* (VDL). All of the dataframes are aggregated and split into training and testing files for input into *AlphaPy*.



12.1 Data Sources

MarketFlow gets daily stock prices from Yahoo Finance and intraday stock prices from Google Finance. Both data sources have the standard primitives: `Open`, `High`, `Low`, `Close`, and `Volume`. For daily data, there is a `Date` timestamp and for intraday data, there is a `Datetime` timestamp. We augment the intraday data with a `bar_number` field to mark the end of the trading day. All trading days do not end at 4:00 pm EST, as there are holiday trading days that are shortened.

Table 1: Amazon Daily Stock Prices (Source: Yahoo)

Date	Open	High	Low	Close	Volume
2017-03-01	853.05	854.83	849.01	853.08	2752000
2017-03-02	853.08	854.82	847.28	848.91	2129200
2017-03-03	847.20	851.99	846.27	849.88	1941100
2017-03-06	845.23	848.49	841.12	846.61	2598400
2017-03-07	845.48	848.46	843.75	846.02	2217800
2017-03-08	848.00	853.07	846.79	850.50	2286500
2017-03-09	851.00	856.40	850.31	853.00	2040600
2017-03-10	857.00	857.35	851.72	852.46	2422000
2017-03-13	851.77	855.69	851.71	854.59	1906100
2017-03-14	853.55	853.75	847.55	852.53	2128700
2017-03-15	854.33	854.45	847.11	852.97	2556700
2017-03-16	855.30	855.50	850.51	853.42	1832600
2017-03-17	853.49	853.83	850.64	852.31	3380700
2017-03-20	851.51	857.80	851.01	856.97	2223300
2017-03-21	858.84	862.80	841.31	843.20	4349100
2017-03-22	840.43	849.37	839.05	848.06	2636200
2017-03-23	848.20	850.89	844.80	847.38	1945700
2017-03-24	851.68	851.80	843.53	845.61	2118300
2017-03-27	838.07	850.30	833.50	846.82	2754200
2017-03-28	851.75	858.46	850.10	856.00	3033000
2017-03-29	859.05	876.44	859.02	874.32	4464400
2017-03-30	874.95	877.06	871.66	876.34	2745800
2017-03-31	877.00	890.35	876.65	886.54	3910700

Note: Normal market hours are 9:30 am to 4:00 pm EST. Here, we retrieved the data from the CST time zone, one hour ahead.

Table 2: Amazon Intraday Stock Prices (Source: Google)

datetime	open	high	low	close	volume	bar_number	end_of_day
2017-03-31 08:30:00	877.00	877.06	876.66	877.06	43910	0	False
2017-03-31 08:40:00	877.00	881.74	876.65	880.66	192152	1	False
2017-03-31 08:50:00	881.00	886.01	880.45	884.67	249892	2	False
2017-03-31 09:00:00	884.83	886.08	883.50	883.90	148034	3	False
2017-03-31 09:10:00	883.90	886.44	883.84	885.72	118518	4	False
2017-03-31 09:20:00	885.82	886.39	884.68	886.30	76880	5	False
2017-03-31 09:30:00	886.14	886.74	885.07	885.73	74180	6	False
2017-03-31 09:40:00	885.80	886.20	885.13	886.20	77154	7	False
2017-03-31 09:50:00	886.21	887.61	885.77	887.51	86971	8	False
2017-03-31 10:00:00	887.59	888.35	886.83	887.81	111998	9	False
2017-03-31 10:10:00	887.80	888.72	887.59	888.60	64497	10	False
2017-03-31 10:20:00	888.62	890.35	888.44	889.82	101562	11	False
2017-03-31 10:30:00	889.81	889.96	888.83	889.83	42580	12	False
2017-03-31 10:40:00	889.70	889.92	887.32	887.61	88559	13	False
2017-03-31 10:50:00	887.68	889.58	887.66	889.01	45492	14	False
2017-03-31 11:00:00	889.12	889.26	887.25	888.34	39841	15	False
2017-03-31 11:10:00	888.52	889.00	887.66	887.66	24525	16	False

continues on next page

Table 2 – continued from previous page

2017-03-31 11:20:00	887.83	888.74	887.67	888.14	35031	17	False
2017-03-31 11:30:00	888.14	888.87	888.10	888.87	24460	18	False
2017-03-31 11:40:00	888.91	889.47	888.76	888.96	38921	19	False
2017-03-31 11:50:00	888.87	889.06	888.47	888.89	35439	20	False
2017-03-31 12:00:00	888.86	889.50	888.82	889.40	25933	21	False
2017-03-31 12:10:00	889.40	889.94	889.35	889.86	35120	22	False
2017-03-31 12:20:00	889.90	890.25	889.51	889.57	38429	23	False
2017-03-31 12:30:00	889.70	889.79	889.20	889.79	18435	24	False
2017-03-31 12:40:00	889.80	889.93	889.28	889.50	25481	25	False
2017-03-31 12:50:00	889.60	889.99	889.50	889.77	26536	26	False
2017-03-31 13:00:00	889.70	889.73	888.33	888.49	35556	27	False
2017-03-31 13:10:00	888.31	888.64	887.80	888.45	39215	28	False
2017-03-31 13:20:00	888.58	888.58	887.09	887.15	43771	29	False
2017-03-31 13:30:00	887.18	888.40	887.05	888.13	36830	30	False
2017-03-31 13:40:00	888.22	888.99	887.68	888.78	29510	31	False
2017-03-31 13:50:00	888.79	888.99	888.35	888.58	30370	32	False
2017-03-31 14:00:00	888.66	888.82	887.02	887.02	48011	33	False
2017-03-31 14:10:00	887.02	888.30	886.80	888.15	41046	34	False
2017-03-31 14:20:00	888.14	889.00	888.06	888.55	38660	35	False
2017-03-31 14:30:00	888.58	888.83	888.30	888.40	39304	36	False
2017-03-31 14:40:00	888.40	888.60	888.01	888.45	57289	37	False
2017-03-31 14:50:00	888.39	889.32	888.16	888.17	105594	38	False
2017-03-31 15:00:00	888.43	888.54	886.54	886.94	518134	39	True

Note: You can get Google intraday data going back a maximum of 50 days. If you want to build your own historical record, then we recommend that you save the data on an ongoing basis for a a larger backtesting window.

12.2 Domain Configuration

The market configuration file (`market.yml`) is written in YAML and is divided into logical sections reflecting different parts of **MarketFlow**. This file is stored in the `config` directory of your project, along with the `model.yml` and `algos.yml` files. The `market` section has the following parameters:

`data_history`: Number of periods of historical data to retrieve.

`forecast_period`: Number of periods to forecast for the target variable.

`fractal`: The time quantum for the data feed, represented by an integer followed by a character code. The string "1d" is one day, and "5m" is five minutes.

`leaders`: A list of features that are coincident with the target variable. For example, with daily stock market data, the `Open` is considered to be a leader because it is recorded at the market open. In contrast, the daily `High` or `Low` cannot be known until the the market close.

`predict_history`: This is the minimum number of periods required to derive all of the features in prediction mode on a given date. If you use a rolling mean of 50 days, then the `predict_history` should be set to at least 50 to have a valid value on the prediction date.

`schema`: This string uniquely identifies the subject matter of the data. A schema could be `prices` for identifying market data.

`target_group`: The name of the group selected from the `groups` section, e.g., a set of stock symbols.

Listing 1: **market.yml**

```
market:
    data_history    : 2000
    forecast_period : 1
    fractal         : 1d
    leaders         : ['gap', 'gapbadown', 'gapbaup', 'gapdown', 'gapup']
    predict_history : 100
    schema          : prices
    target_group    : test

groups:
    all  : ['aaoi', 'aapl', 'acia', 'adbe', 'adi', 'adp', 'agn', 'aig', 'akam',
            'algn', 'alk', 'alxn', 'amat', 'amba', 'amd', 'amgn', 'amt', 'amzn',
            'antm', 'arch', 'asml', 'athn', 'atvi', 'auph', 'avgo', 'axp', 'ayx',
            'azo', 'ba', 'baba', 'bac', 'bby', 'bidu', 'biib', 'brcd', 'bvsn',
            'bwld', 'c', 'cacc', 'cara', 'casy', 'cat', 'cde', 'celg', 'cern',
            'chkp', 'chtr', 'clvs', 'cme', 'cmg', 'cof', 'cohr', 'comm', 'cost',
            'cpk', 'crm', 'crus', 'csco', 'ctsh', 'ctxs', 'csx', 'cvs', 'cybr',
            'data', 'ddd', 'deck', 'dgaz', 'dia', 'dis', 'dish', 'dnkn', 'dpz',
            'drys', 'dust', 'ea', 'ebay', 'edc', 'edz', 'eem', 'elli', 'eog',
            'esrx', 'etrm', 'ewh', 'ewt', 'expe', 'fang', 'fas', 'faz', 'fb',
            'fcx', 'fdx', 'ffiv', 'fit', 'five', 'fnsr', 'fslr', 'ftnt', 'gddy',
            'gdx', 'gdxj', 'ge', 'gild', 'gld', 'glw', 'gm', 'googl', 'gpro',
            'grub', 'gs', 'gwph', 'hal', 'has', 'hd', 'hdp', 'hlf', 'hog', 'hum',
            'ibb', 'ibm', 'ice', 'idxx', 'ilmn', 'ilmn', 'incy', 'intc', 'intu',
            'ip', 'isrg', 'iwm', 'ivv', 'iwf', 'iwm', 'jack', 'jcp', 'jdst', 'jnj',
            'jnpr', 'jnug', 'jpm', 'kite', 'klac', 'ko', 'kss', 'labd', 'labu',
            'len', 'lite', 'lmt', 'lnkd', 'lrcx', 'lulu', 'lvs', 'mbly', 'mcd',
            'mchp', 'mdy', 'meoh', 'mnst', 'mo', 'momo', 'mon', 'mrk', 'ms', 'msft',
            'mtb', 'mu', 'nflx', 'nfx', 'nke', 'ntap', 'ntes', 'ntnx', 'nugt',
            'nvda', 'nxpi', 'nxst', 'oii', 'oled', 'orcl', 'orly', 'p', 'panw',
            'pcln', 'pg', 'pm', 'pnra', 'prgo', 'pxd', 'pypl', 'qcom', 'qqq',
            'qrvo', 'rht', 'sam', 'sbux', 'sds', 'sgen', 'shld', 'shop', 'sig',
            'sina', 'siri', 'skx', 'slb', 'slv', 'smh', 'snap', 'sncr', 'soda',
            'splk', 'spy', 'stld', 'stmp', 'stx', 'svxy', 'swks', 'symc', 't',
            'tbt', 'teva', 'tgt', 'tho', 'tlt', 'tmo', 'tna', 'tqqq', 'trip',
            'tsla', 'ttwo', 'tvix', 'twlo', 'twtr', 'tza', 'uaa', 'ugaz', 'uhs',
            'ulta', 'ulti', 'unh', 'unp', 'upro', 'uri', 'ups', 'uri', 'uthr',
            'utx', 'uvxy', 'v', 'veev', 'viav', 'vlo', 'vmc', 'vrsn', 'vrtx', 'vrx',
            'vwo', 'vxx', 'vz', 'wday', 'wdc', 'wfc', 'wfm', 'wmt', 'wynn', 'x',
            'xbi', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlnx', 'xom', 'xlp', 'xlu',
            'xlv', 'xme', 'xom', 'wix', 'yelp', 'z']
    etf  : ['dia', 'dust', 'edc', 'edz', 'eem', 'ewh', 'ewt', 'fas', 'faz',
            'gld', 'hyg', 'iwm', 'ivv', 'iwf', 'jnk', 'mdy', 'nugt', 'qqq',
            'sds', 'smh', 'spy', 'tbt', 'tlt', 'tna', 'tvix', 'tza', 'upro',
            'uvxy', 'vwo', 'vxx', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlp',
            'xlu', 'xlv', 'xme']
    tech : ['aapl', 'adbe', 'amat', 'amgn', 'amzn', 'avgo', 'baba', 'bidu',
            'brcd', 'csco', 'ddd', 'emc', 'expe', 'fb', 'fit', 'fslr', 'goog',
            'intc', 'isrg', 'lnkd', 'msft', 'nflx', 'nvda', 'pcln', 'qcom',
            'qqq', 'tsla', 'twtr']
    test : ['aapl', 'amzn', 'goog', 'fb', 'nvda', 'tsla']

features: ['abovema_3', 'abovema_5', 'abovema_10', 'abovema_20', 'abovema_50',
           'adx', 'atr', 'bigdown', 'bigup', 'diminus', 'diplus', 'doji',
           'gap', 'gapbadown', 'gapbaup', 'gapdown', 'gapup',
```

(continues on next page)

(continued from previous page)

```
            'hc', 'hh', 'ho', 'hl', 'lc', 'lh', 'll', 'lo', 'hookdown', 'hookup',
            'inside', 'outside', 'madelta_3', 'madelta_5', 'madelta_7', 'madelta_10',
            'madelta_12', 'madelta_15', 'madelta_18', 'madelta_20', 'madelta',
            'net', 'netdown', 'netup', 'nr_3', 'nr_4', 'nr_5', 'nr_7', 'nr_8',
            'nr_10', 'nr_18', 'roi', 'roi_2', 'roi_3', 'roi_4', 'roi_5', 'roi_10',
            'roi_20', 'rr_1_4', 'rr_1_7', 'rr_1_10', 'rr_2_5', 'rr_2_7', 'rr_2_10',
            'rr_3_8', 'rr_3_14', 'rr_4_10', 'rr_4_20', 'rr_5_10', 'rr_5_20',
            'rr_5_30', 'rr_6_14', 'rr_6_25', 'rr_7_14', 'rr_7_35', 'rr_8_22',
            'rrhigh', 'rrlow', 'rrover', 'rrunder', 'rsi_3', 'rsi_4', 'rsi_5',
            'rsi_6', 'rsi_8', 'rsi_10', 'rsi_14', 'sep_3_3', 'sep_5_5', 'sep_8_8',
            'sep_10_10', 'sep_14_14', 'sep_21_21', 'sep_30_30', 'sep_40_40',
            'sephigh', 'seplow', 'trend', 'vma', 'vmover', 'vmratio', 'vmunder',
            'volatility_3', 'volatility_5', 'volatility', 'volatility_20',
            'wr_2', 'wr_3', 'wr', 'wr_5', 'wr_6', 'wr_7', 'wr_10']

aliases:
    atr         : 'ma_truerange'
    aver        : 'ma_hlrange'
    cma         : 'ma_close'
    cmax        : 'highest_close'
    cmin        : 'lowest_close'
    hc          : 'higher_close'
    hh          : 'higher_high'
    hl          : 'higher_low'
    ho          : 'higher_open'
    hmax        : 'highest_high'
    hmin        : 'lowest_high'
    lc          : 'lower_close'
    lh          : 'lower_high'
    ll          : 'lower_low'
    lo          : 'lower_open'
    lmax        : 'highest_low'
    lmin        : 'lowest_low'
    net         : 'net_close'
    netdown     : 'down_net'
    netup       : 'up_net'
    omax        : 'highest_open'
    omin        : 'lowest_open'
    rmax        : 'highest_hlrange'
    rmin        : 'lowest_hlrange'
    rr          : 'maratio_hlrange'
    rixc        : 'rindex_close_high_low'
    rixo        : 'rindex_open_high_low'
    roi         : 'netreturn_close'
    rsi         : 'rsi_close'
    sepma       : 'ma_sep'
    vma         : 'ma_volume'
    vmratio     : 'maratio_volume'
    upmove      : 'net_high'

variables:
    abovema     : 'close > cma_50'
    belowma     : 'close < cma_50'
    bigup       : 'rrover & sephigh & netup'
    bigdown     : 'rrover & sephigh & netdown'
    doji        : 'sepdoji & rrunder'
    hookdown    : 'open > high[1] & close < close[1]'
```

(continues on next page)

(continued from previous page)

```
    hookup     : 'open < low[1] & close > close[1]'
    inside     : 'low > low[1] & high < high[1]'
    madelta    : '(close - cma_50) / atr_10'
    nr         : 'hlrange == rmin_4'
    outside    : 'low < low[1] & high > high[1]'
    roihigh    : 'roi_5 >= 5'
    roilow     : 'roi_5 < -5'
    roiminus   : 'roi_5 < 0'
    roiplus    : 'roi_5 > 0'
    rrhigh     : 'rr_1_10 >= 1.2'
    rrlow      : 'rr_1_10 <= 0.8'
    rrover     : 'rr_1_10 >= 1.0'
    rrunder    : 'rr_1_10 < 1.0'
    sep        : 'rixc_1 - rixo_1'
    sepdoji    : 'abs(sep) <= 15'
    sephigh    : 'abs(sep_1_1) >= 70'
    seplow     : 'abs(sep_1_1) <= 30'
    trend      : 'rrover & sephigh'
    vmover     : 'vmratio >= 1'
    vmunder    : 'vmratio < 1'
    volatility : 'atr_10 / close'
    wr         : 'hlrange == rmax_4'
```

12.3 Group Analysis

The cornerstone of MarketFlow is the *Analysis*. You can create models and forecasts for different groups of stocks. The purpose of the analysis object is to gather data for all of the group members and then consolidate the data into train and test files. Further, some features and the target variable have to be adjusted (lagged) to avoid data leakage.

A group is simply a collection of symbols for analysis. In this example, we create different groups for technology stocks, ETFs, and a smaller group for testing. To create a model for a given group, simply set the `target_group` in the `market` section of the market.yml file and run `mflow`.

Listing 2: **market.yml**

```
groups:
    all  : ['aaoi', 'aapl', 'acia', 'adbe', 'adi', 'adp', 'agn', 'aig', 'akam',
            'algn', 'alk', 'alxn', 'amat', 'amba', 'amd', 'amgn', 'amt', 'amzn',
            'antm', 'arch', 'asml', 'athn', 'atvi', 'auph', 'avgo', 'axp', 'ayx',
            'azo', 'ba', 'baba', 'bac', 'bby', 'bidu', 'biib', 'brcd', 'bvsn',
            'bwld', 'c', 'cacc', 'cara', 'casy', 'cat', 'cde', 'celg', 'cern',
            'chkp', 'chtr', 'clvs', 'cme', 'cmg', 'cof', 'cohr', 'comm', 'cost',
            'cpk', 'crm', 'crus', 'csco', 'ctsh', 'ctxs', 'csx', 'cvs', 'cybr',
            'data', 'ddd', 'deck', 'dgaz', 'dia', 'dis', 'dish', 'dnkn', 'dpz',
            'drys', 'dust', 'ea', 'ebay', 'edc', 'edz', 'eem', 'elli', 'eog',
            'esrx', 'etrm', 'ewh', 'ewt', 'expe', 'fang', 'fas', 'faz', 'fb',
            'fcx', 'fdx', 'ffiv', 'fit', 'five', 'fnsr', 'fslr', 'ftnt', 'gddy',
            'gdx', 'gdxj', 'ge', 'gild', 'gld', 'glw', 'gm', 'googl', 'gpro',
            'grub', 'gs', 'gwph', 'hal', 'has', 'hd', 'hdp', 'hlf', 'hog', 'hum',
            'ibb', 'ibm', 'ice', 'idxx', 'ilmn', 'ilmn', 'incy', 'intc', 'intu',
            'ip', 'isrg', 'iwm', 'ivv', 'iwf', 'iwm', 'jack', 'jcp', 'jdst', 'jnj',
            'jnpr', 'jnug', 'jpm', 'kite', 'klac', 'ko', 'kss', 'labd', 'labu',
            'len', 'lite', 'lmt', 'lnkd', 'lrcx', 'lulu', 'lvs', 'mbly', 'mcd',
            'mchp', 'mdy', 'meoh', 'mnst', 'mo', 'momo', 'mon', 'mrk', 'ms', 'msft',
```

(continues on next page)

(continued from previous page)

```
            'mtb', 'mu', 'nflx', 'nfx', 'nke', 'ntap', 'ntes', 'ntnx', 'nugt',
            'nvda', 'nxpi', 'nxst', 'oii', 'oled', 'orcl', 'orly', 'p', 'panw',
            'pcln', 'pg', 'pm', 'pnra', 'prgo', 'pxd', 'pypl', 'qcom', 'qqq',
            'qrvo', 'rht', 'sam', 'sbux', 'sds', 'sgen', 'shld', 'shop', 'sig',
            'sina', 'siri', 'skx', 'slb', 'slv', 'smh', 'snap', 'sncr', 'soda',
            'splk', 'spy', 'stld', 'stmp', 'stx', 'svxy', 'swks', 'symc', 't',
            'tbt', 'teva', 'tgt', 'tho', 'tlt', 'tmo', 'tna', 'tqqq', 'trip',
            'tsla', 'ttwo', 'tvix', 'twlo', 'twtr', 'tza', 'uaa', 'ugaz', 'uhs',
            'ulta', 'ulti', 'unh', 'unp', 'upro', 'uri', 'ups', 'uri', 'uthr',
            'utx', 'uvxy', 'v', 'veev', 'viav', 'vlo', 'vmc', 'vrsn', 'vrtx', 'vrx',
            'vwo', 'vxx', 'vz', 'wday', 'wdc', 'wfc', 'wfm', 'wmt', 'wynn', 'x',
            'xbi', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlnx', 'xom', 'xlp', 'xlu',
            'xlv', 'xme', 'xom', 'wix', 'yelp', 'z']
    etf  : ['dia', 'dust', 'edc', 'edz', 'eem', 'ewh', 'ewt', 'fas', 'faz',
            'gld', 'hyg', 'iwm', 'ivv', 'iwf', 'jnk', 'mdy', 'nugt', 'qqq',
            'sds', 'smh', 'spy', 'tbt', 'tlt', 'tna', 'tvix', 'tza', 'upro',
            'uvxy', 'vwo', 'vxx', 'xhb', 'xiv', 'xle', 'xlf', 'xlk', 'xlp',
            'xlu', 'xlv', 'xme']
    tech : ['aapl', 'adbe', 'amat', 'amgn', 'amzn', 'avgo', 'baba', 'bidu',
            'brcd', 'csco', 'ddd', 'emc', 'expe', 'fb', 'fit', 'fslr', 'goog',
            'intc', 'isrg', 'lnkd', 'msft', 'nflx', 'nvda', 'pcln', 'qcom',
            'qqq', 'tsla', 'twtr']
    test : ['aapl', 'amzn', 'goog', 'fb', 'nvda', 'tsla']
```

12.4 Variables and Aliases

Because market analysis encompasses a wide array of technical indicators, you can define features using the *Variable Definition Language* (VDL). The concept is simple: flatten out a function call and its parameters into a string, and that string represents the variable name. You can use the technical analysis functions in AlphaPy, or define your own.

Let's define a feature that indicates whether or not a stock is above its 50-day closing moving average. The *alphapy.market_variables* module has a function `ma` to calculate a rolling mean. It has two parameters: the name of the dataframe's column and the period over which to calculate the mean. So, the corresponding variable name is `ma_close_50`.

Typically, a moving average is calculated with the closing price, so we can define an alias `cma` which represents the closing moving average. An alias is simply a substitution mechanism for replacing one string with an abbreviation. Instead of `ma_close_50`, we can now refer to `cma_50` using an alias.

Finally, we can define the variable `abovema` with a relational expression. Note that numeric values in the expression can be substituted when defining features, e.g., `abovema_20`.

Listing 3: **market.yml**

```
features: ['abovema_50']

aliases:
    cma         : 'ma_close'

variables:
    abovema     : 'close > cma_50'
```

Here are more examples of aliases.

Listing 4: **market.yml**

```
aliases:
    atr         : 'ma_truerange'
    aver        : 'ma_hlrange'
    cma         : 'ma_close'
    cmax        : 'highest_close'
    cmin        : 'lowest_close'
    hc          : 'higher_close'
    hh          : 'higher_high'
    hl          : 'higher_low'
    ho          : 'higher_open'
    hmax        : 'highest_high'
    hmin        : 'lowest_high'
    lc          : 'lower_close'
    lh          : 'lower_high'
    ll          : 'lower_low'
    lo          : 'lower_open'
    lmax        : 'highest_low'
    lmin        : 'lowest_low'
    net         : 'net_close'
    netdown     : 'down_net'
    netup       : 'up_net'
    omax        : 'highest_open'
    omin        : 'lowest_open'
    rmax        : 'highest_hlrange'
    rmin        : 'lowest_hlrange'
    rr          : 'maratio_hlrange'
    rixc        : 'rindex_close_high_low'
    rixo        : 'rindex_open_high_low'
    roi         : 'netreturn_close'
    rsi         : 'rsi_close'
    sepma       : 'ma_sep'
    vma         : 'ma_volume'
    vmratio     : 'maratio_volume'
    upmove      : 'net_high'
```

Variable expressions are valid Python expressions, with the addition of offsets to reference previous values.

Listing 5: **market.yml**

```
variables:
    abovema     : 'close > cma_50'
    belowma     : 'close < cma_50'
    bigup       : 'rrover & sephigh & netup'
    bigdown     : 'rrover & sephigh & netdown'
    doji        : 'sepdoji & rrunder'
    hookdown    : 'open > high[1] & close < close[1]'
    hookup      : 'open < low[1] & close > close[1]'
    inside      : 'low > low[1] & high < high[1]'
    madelta     : '(close - cma_50) / atr_10'
    nr          : 'hlrange == rmin_4'
    outside     : 'low < low[1] & high > high[1]'
    roihigh     : 'roi_5 >= 5'
    roilow      : 'roi_5 < -5'
    roiminus    : 'roi_5 < 0'
    roiplus     : 'roi_5 > 0'
```

(continues on next page)

(continued from previous page)

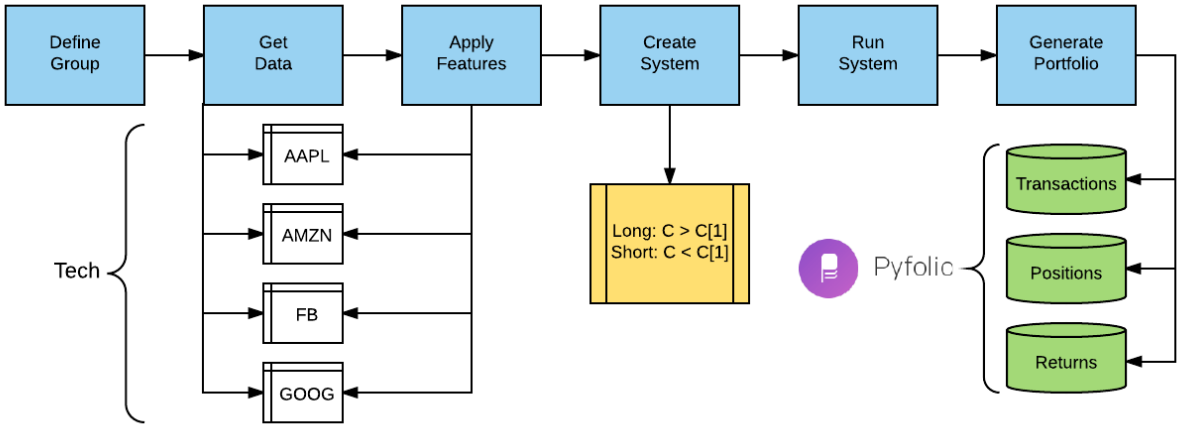
```
    rrhigh     : 'rr_1_10 >= 1.2'
    rrlow      : 'rr_1_10 <= 0.8'
    rrover     : 'rr_1_10 >= 1.0'
    rrunder    : 'rr_1_10 < 1.0'
    sep        : 'rixc_1 - rixo_1'
    sepdoji    : 'abs(sep) <= 15'
    sephigh    : 'abs(sep_1_1) >= 70'
    seplow     : 'abs(sep_1_1) <= 30'
    trend      : 'rrover & sephigh'
    vmover     : 'vmratio >= 1'
    vmunder    : 'vmratio < 1'
    volatility : 'atr_10 / close'
    wr         : 'hlrange == rmax_4'
```

Once the aliases and variables are defined, a foundation is established for defining all of the features that you want to test.

Listing 6: **market.yml**

```
features: ['abovema_3', 'abovema_5', 'abovema_10', 'abovema_20', 'abovema_50',
           'adx', 'atr', 'bigdown', 'bigup', 'diminus', 'diplus', 'doji',
           'gap', 'gapbadown', 'gapbaup', 'gapdown', 'gapup',
           'hc', 'hh', 'ho', 'hl', 'lc', 'lh', 'll', 'lo', 'hookdown', 'hookup',
           'inside', 'outside', 'madelta_3', 'madelta_5', 'madelta_7', 'madelta_10',
           'madelta_12', 'madelta_15', 'madelta_18', 'madelta_20', 'madelta',
           'net', 'netdown', 'netup', 'nr_3', 'nr_4', 'nr_5', 'nr_7', 'nr_8',
           'nr_10', 'nr_18', 'roi', 'roi_2', 'roi_3', 'roi_4', 'roi_5', 'roi_10',
           'roi_20', 'rr_1_4', 'rr_1_7', 'rr_1_10', 'rr_2_5', 'rr_2_7', 'rr_2_10',
           'rr_3_8', 'rr_3_14', 'rr_4_10', 'rr_4_20', 'rr_5_10', 'rr_5_20',
           'rr_5_30', 'rr_6_14', 'rr_6_25', 'rr_7_14', 'rr_7_35', 'rr_8_22',
           'rrhigh', 'rrlow', 'rrover', 'rrunder', 'rsi_3', 'rsi_4', 'rsi_5',
           'rsi_6', 'rsi_8', 'rsi_10', 'rsi_14', 'sep_3_3', 'sep_5_5', 'sep_8_8',
           'sep_10_10', 'sep_14_14', 'sep_21_21', 'sep_30_30', 'sep_40_40',
           'sephigh', 'seplow', 'trend', 'vma', 'vmover', 'vmratio', 'vmunder',
           'volatility_3', 'volatility_5', 'volatility', 'volatility_20',
           'wr_2', 'wr_3', 'wr', 'wr_5', 'wr_6', 'wr_7', 'wr_10']
```

12.5 Trading Systems



MarketFlow provides two out-of-the-box trading systems. The first is a long/short system that you define using the system features in the configuration file `market.yml`. When MarketFlow detects a system in the file, it knows to execute that particular long/short strategy.

Listing 7: **market.yml**

```
market:
    data_history    : 1000
    forecast_period : 1
    fractal         : 1d
    leaders         : []
    predict_history : 50
    schema          : prices
    target_group    : faang

system:
    name       : 'closer'
    holdperiod : 0
    longentry  : hc
    longexit   :
    shortentry : lc
    shortexit  :
    scale      : False

groups:
    faang      : ['fb', 'aapl', 'amzn', 'nflx', 'googl']

features       : ['hc', 'lc']

aliases:
    hc         : 'higher_close'
    lc         : 'lower_close'
```

name: Unique identifier for the trading system.

holdperiod: Number of periods to hold an open position.

longentry: A conditional feature to establish when to open a long position.

longexit: A conditional feature to establish when to close a long position.

shortentry: A conditional feature to establish when to open a short position.

shortexit: A conditional feature to establish when to close a short position.

scale: When `True`, add to a position in the same direction. The default action is not to scale positions.

The second system is an *open range breakout* strategy. The premise of the system is to wait for an established high-low range in the first n minutes (e.g., 30) and then wait for a breakout of either the high or the low, especially when the range is relatively narrow. Typically, a stop-loss is set at the other side of the breakout range.

After a system runs, four output files are stored in the `system` directory; the first three are formatted for analysis by Quantopian's **pyfolio** package. The last file is the list of trades generated by MarketFlow based on the system specifications.

- [group]_[system]_transactions_[fractal].csv
- [group]_[system]_positions_[fractal].csv
- [group]_[system]_returns_[fractal].csv
- [group]_[system]_trades_[fractal].csv

If we developed a moving average crossover system on daily data for technology stocks, then the trades file could be named:

    tech_xma_trades_1d.csv

The important point here is to reserve a namespace for different combinations of groups, systems, and fractals to compare performance over space and time.

12.6 Model Configuration

MarketFlow runs on top of AlphaPy, so the `model.yml` file has the same format. In the following example, note the use of treatments to calculate runs for a set of features.

Listing 8: **model.yml**

```
project:
    directory         : .
    file_extension    : csv
    submission_file   :
    submit_probas     : False

data:
    drop              : ['date', 'tag', 'open', 'high', 'low', 'close', 'volume',
↪'adjclose',
                         'low[1]', 'high[1]', 'net', 'close[1]', 'rmin_3', 'rmin_4',
↪'rmin_5',
                         'rmin_7', 'rmin_8', 'rmin_10', 'rmin_18', 'pval', 'mval',
↪'vma',
                         'rmax_2', 'rmax_3', 'rmax_4', 'rmax_5', 'rmax_6', 'rmax_7',
↪'rmax_10']
    features          : '*'
    sampling          :
        option        : True
        method        : under_random
        ratio         : 0.5
    sentinel          : -1
```

(continues on next page)

(continued from previous page)

```
    separator         : ','
    shuffle           : True
    split             : 0.4
    target            : rrover
    target_value      : True

model:
    algorithms        : ['RF']
    balance_classes   : True
    calibration       :
        option        : False
        type          : isotonic
    cv_folds          : 3
    estimators        : 501
    feature_selection :
        option        : True
        percentage    : 50
        uni_grid      : [5, 10, 15, 20, 25]
        score_func    : f_classif
    grid_search       :
        option        : False
        iterations    : 100
        random        : True
        subsample     : True
        sampling_pct  : 0.25
    pvalue_level      : 0.01
    rfe               :
        option        : True
        step          : 10
    scoring_function  : 'roc_auc'
    type              : classification

features:
    clustering        :
        option        : False
        increment     : 3
        maximum       : 30
        minimum       : 3
    counts            :
        option        : False
    encoding          :
        rounding      : 3
        type          : factorize
    factors           : []
    interactions      :
        option        : True
        poly_degree   : 2
        sampling_pct  : 5
    isomap            :
        option        : False
        components    : 2
        neighbors     : 5
    logtransform      :
        option        : False
    numpy             :
        option        : False
    pca               :
```

(continues on next page)

(continued from previous page)

```
        option        : False
        increment     : 3
        maximum       : 15
        minimum       : 3
        whiten        : False
    scaling           :
        option        : True
        type          : standard
    scipy             :
        option        : False
    text              :
        ngrams        : 1
        vectorize     : False
    tsne              :
        option        : False
        components    : 2
        learning_rate : 1000.0
        perplexity    : 30.0
    variance          :
        option        : True
        threshold     : 0.1

treatments:
    doji              : ['alphapy.features', 'runs_test', ['all'], 18]
    hc                : ['alphapy.features', 'runs_test', ['all'], 18]
    hh                : ['alphapy.features', 'runs_test', ['all'], 18]
    hl                : ['alphapy.features', 'runs_test', ['all'], 18]
    ho                : ['alphapy.features', 'runs_test', ['all'], 18]
    rrhigh            : ['alphapy.features', 'runs_test', ['all'], 18]
    rrlow             : ['alphapy.features', 'runs_test', ['all'], 18]
    rrover            : ['alphapy.features', 'runs_test', ['all'], 18]
    rrunder           : ['alphapy.features', 'runs_test', ['all'], 18]
    sephigh           : ['alphapy.features', 'runs_test', ['all'], 18]
    seplow            : ['alphapy.features', 'runs_test', ['all'], 18]
    trend             : ['alphapy.features', 'runs_test', ['all'], 18]

pipeline:
    number_jobs       : -1
    seed              : 10231
    verbosity         : 0

plots:
    calibration       : True
    confusion_matrix  : True
    importances       : True
    learning_curve    : True
    roc_curve         : True

xgboost:
    stopping_rounds   : 20
```

12.7 Creating the Model

First, change the directory to your project location, where you have already followed the *Project Structure* specifications:

```
cd path/to/project
```

Run this command to train a model:

```
mflow
```

Usage:

```
mflow [--train | --predict] [--tdate yyyy-mm-dd] [--pdate yyyy-mm-dd]
```

--train	Train a new model and make predictions (Default)
--predict	Make predictions from a saved model
--tdate	The training date in format YYYY-MM-DD (Default: Earliest Date in the Data)
--pdate	The prediction date in format YYYY-MM-DD (Default: Today's Date)

12.8 Running the Model

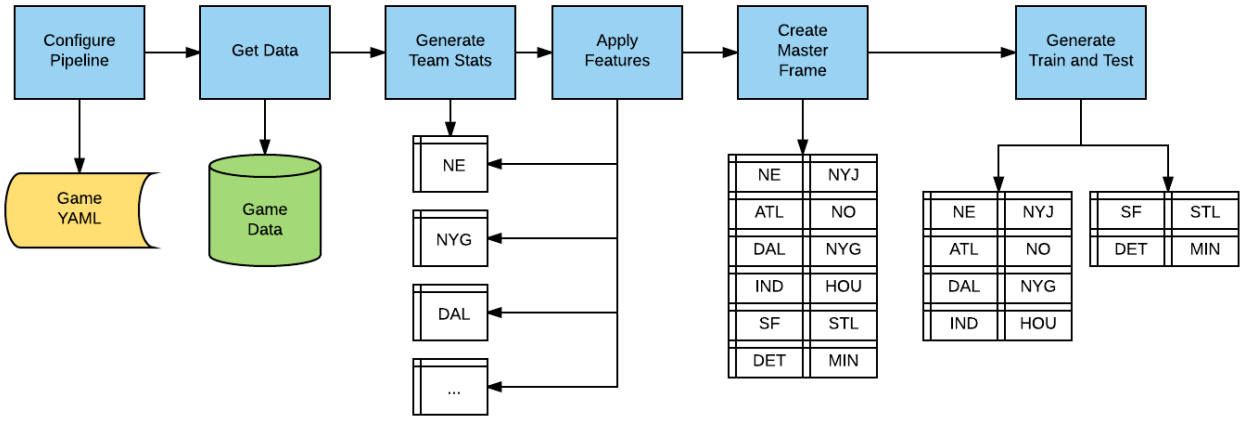
In the project location, run `mflow` with the `predict` flag. MarketFlow will automatically create the `predict.csv` file using the `pdate` option:

```
mflow --predict [--pdate yyyy-mm-dd]
```

CHAPTER

THIRTEEN

SPORTFLOW



SportFlow applies machine learning algorithms to predict game outcomes for matches in any team sport. We created binary features (for classification) to determine whether or not a team will win the game or even more importantly, cover the spread. We also try to predict whether or not a game's total points will exceed the *over/under*.

Of course, there are practical matters to predicting a game's outcome. The strength of supervised learning is to improve an algorithm's performance with lots of data. While major-league baseball has a total of 2,430 games per year, pro football has only 256 games per year. College football and basketball are somewhere in the middle of this range.

The other complication is determining whether or not a model for one sport can be used for another. The advantage is that combining sports gives us more data. The disadvantage is that each sport has unique characteristics that could make a unified model infeasible. Still, we can combine the game data to test an overall model.

13.1 Data Sources

SportFlow starts with minimal game data (lines and scores) and expands these data into temporal features such as runs and streaks for all of the features. Currently, we do not incorporate player data or other external factors, but there are some excellent open-source packages such as BurntSushi's *nflgame* Python code. For its initial version, SportFlow game data must be in the format below:

Table 1: NCAA Basketball Data

season	date	away.team	away.score	home.team	home.score	line	over_under
2015	2015-11-13	COLO	62	ISU	68	-10	151

continues on next page

Table 1 – continued from previous page

2015	2015-11-13	SDAK	69	WRST	77	-6.5	136
2015	2015-11-13	WAG	57	SJU	66	-5.5	142
2015	2015-11-13	JVST	83	CMU	89	-18	142.5
2015	2015-11-13	NIAG	50	ODU	67	-18	132
2015	2015-11-13	ALBY	65	UK	78	-20	132.5
2015	2015-11-13	TEM	67	UNC	91	-9.5	145
2015	2015-11-13	NKU	61	WVU	107	-23.5	147.5
2015	2015-11-13	SIE	74	DUKE	92	-24	155
2015	2015-11-13	WCU	72	CIN	97	-20	132
2015	2015-11-13	MSM	56	MD	80	-21.5	140
2015	2015-11-13	CHAT	92	UGA	90	-10.5	136
2015	2015-11-13	SEMO	53	DAY	84	-19	140
2015	2015-11-13	DART	67	HALL	84	-11	136
2015	2015-11-13	CAN	85	HOF	96	-10.5	150.5
2015	2015-11-13	JMU	87	RICH	75	-9	137.5
2015	2015-11-13	EIU	49	IND	88	-25	150
2015	2015-11-13	FAU	55	MSU	82	-23.5	141
2015	2015-11-13	SAM	45	LOU	86	-23	142
2015	2015-11-13	MIOH	72	XAV	81	-15.5	144.5
2015	2015-11-13	PRIN	64	RID	56	1	137
2015	2015-11-13	IUPU	72	INST	70	-8	135.5
2015	2015-11-13	SAC	66	ASU	63	-18	144
2015	2015-11-13	AFA	75	SIU	77	-5.5	131
2015	2015-11-13	UNCO	72	KU	109	-29	147.5
2015	2015-11-13	BALL	53	BRAD	54	3	135
2015	2015-11-13	USD	45	USC	83	-12.5	140
2015	2015-11-13	UTM	57	OKST	91	-12	141.5
2015	2015-11-13	COR	81	GT	116	-17	130
2015	2015-11-13	MOST	65	ORU	80	-4.5	133.5
2015	2015-11-13	DREX	81	JOES	82	-9.5	127.5
2015	2015-11-13	WMRY	85	NCST	68	-12.5	149
2015	2015-11-13	SF	78	UIC	75	1.5	148.5
2015	2015-11-13	PEAY	41	VAN	80	-24.5	144
2015	2015-11-13	CSN	71	NIU	83	-9.5	134.5
2015	2015-11-13	UCSB	60	OMA	59	-2.5	157.5
2015	2015-11-13	UTSA	64	LOYI	76	-14	138.5
2015	2015-11-13	BRWN	65	SPU	77	-2	130.5
2015	2015-11-13	NAU	70	WSU	82	-10.5	145

The SportFlow logic is split-apply-combine, as the data are first split along team lines, then team statistics are calculated and applied, and finally the team data are inserted into the overall model frame.

13.2 Domain Configuration

The SportFlow configuration file is minimal. You can simulate random scoring to compare with a real model. Further, you can experiment with the rolling window for run and streak calculations.

Listing 1: **sport.yml**

```
sport:
    points_max      : 100
    points_min      : 50
    random_scoring  : False
    seasons         : []
    rolling_window  : 3
```

points_max: Maximum number of simulated points to assign to any single team.

points_min: Minimum number of simulated points to assign to any single team.

random_scoring: If `True`, assign random point values to games [Default: `False`].

seasons: The yearly list of seasons to evaluate.

rolling_window: The period over which streaks are calculated.

13.3 Model Configuration

SportFlow runs on top of AlphaPy, so the `model.yml` file has the same format.

Listing 2: **model.yml**

```
project:
    directory         : .
    file_extension    : csv
    submission_file   :
    submit_probas     : False

data:
    drop              : ['Unnamed: 0', 'index', 'season', 'date', 'home.team', 'away.
↪team',
                         'home.score', 'away.score', 'total_points', 'point_margin_
↪game',
                         'won_on_points', 'lost_on_points', 'cover_margin_game',
                         'lost_on_spread', 'overunder_margin', 'over', 'under']
    features          : '*'
    sampling          :
        option        : False
        method        : under_random
        ratio         : 0.0
    sentinel          : -1
    separator         : ','
    shuffle           : False
    split             : 0.4
    target            : won_on_spread
    target_value      : True

model:
```

(continues on next page)

(continued from previous page)

```
    algorithms        : ['RF', 'XGB']
    balance_classes   : False
    calibration       :
        option        : False
        type          : isotonic
    cv_folds          : 3
    estimators        : 201
    feature_selection :
        option        : False
        percentage    : 50
        uni_grid      : [5, 10, 15, 20, 25]
        score_func    : f_classif
    grid_search       :
        option        : True
        iterations    : 50
        random        : True
        subsample     : False
        sampling_pct  : 0.25
    pvalue_level      : 0.01
    rfe               :
        option        : True
        step          : 5
    scoring_function  : 'roc_auc'
    type              : classification

features:
    clustering        :
        option        : False
        increment     : 3
        maximum       : 30
        minimum       : 3
    counts            :
        option        : False
    encoding          :
        rounding      : 3
        type          : factorize
    factors           : ['line', 'delta.wins', 'delta.losses', 'delta.ties',
                         'delta.point_win_streak', 'delta.point_loss_streak',
                         'delta.cover_win_streak', 'delta.cover_loss_streak',
                         'delta.over_streak', 'delta.under_streak']
    interactions      :
        option        : True
        poly_degree   : 2
        sampling_pct  : 5
    isomap            :
        option        : False
        components    : 2
        neighbors     : 5
    logtransform      :
        option        : False
    numpy             :
        option        : False
    pca               :
        option        : False
        increment     : 3
        maximum       : 15
        minimum       : 3
```

(continues on next page)

(continued from previous page)

```
        whiten          : False
    scaling             :
        option          : True
        type            : standard
    scipy               :
        option          : False
    text                :
        ngrams          : 1
        vectorize       : False
    tsne                :
        option          : False
        components      : 2
        learning_rate   : 1000.0
        perplexity      : 30.0
    variance            :
        option          : True
        threshold       : 0.1

pipeline:
    number_jobs         : -1
    seed                : 13201
    verbosity           : 0

plots:
    calibration         : True
    confusion_matrix    : True
    importances         : True
    learning_curve      : True
    roc_curve           : True

xgboost:
    stopping_rounds     : 30
```

13.4 Creating the Model

First, change the directory to your project location, where you have already followed the *Project Structure* specifications:

```
cd path/to/project
```

Run this command to train a model:

```
sflow
```

Usage:

```
sflow [--train | --predict] [--tdate yyyy-mm-dd] [--pdate yyyy-mm-dd]
```

--train	Train a new model and make predictions (Default)
--predict	Make predictions from a saved model
--tdate	The training date in format YYYY-MM-DD (Default: Earliest Date in the Data)
--pdate	The prediction date in format YYYY-MM-DD (Default: Today's Date)

13.5 Running the Model

In the project location, run `sflow` with the `predict` flag. SportFlow will automatically create the `predict.csv` file using the `pdate` option:

```
sflow --predict [--pdate yyyy-mm-dd]
```

CHAPTER

FOURTEEN

ALPHAPY

14.1 alphapy package

14.1.1 Submodules

14.1.2 alphapy.__main__ module

alphapy.__main__.**main**(*args=None*)
AlphaPy Main Program

Notes

(1) Initialize logging.

(2) Parse the command line arguments.

(3) Get the model configuration.

(4) Create the model object.

(5) Call the main AlphaPy pipeline.

alphapy.__main__.**main_pipeline**(*model*)
AlphaPy Main Pipeline

Parameters **model** (*alphapy.Model*) – The model specifications for the pipeline.

Returns **model** – The final model.

Return type alphapy.Model

alphapy.__main__.**prediction_pipeline**(*model*)
AlphaPy Prediction Pipeline

Parameters **model** (*alphapy.Model*) – The model object for controlling the pipeline.

Returns **None**

Return type None

Notes

The saved model is loaded from disk, and predictions are made on the new testing data.

alphapy.__main__.**training_pipeline**(*model*)
AlphaPy Training Pipeline

Parameters **model** (*alphapy.Model*) – The model object for controlling the pipeline.

Returns **model** – The final results are stored in the model object.

Return type alphapy.Model

Raises **KeyError** – If the number of columns of the train and test data do not match, then this exception is raised.

14.1.3 alphapy.alias module

class alphapy.alias.**Alias**(*name*, *expr*, *replace=False*)
Bases: object

Create a new alias as a key-value pair. All aliases are stored in `Alias.aliases`. Duplicate keys or values are not allowed, unless the `replace` parameter is `True`.

Parameters

- **name** (*str*) – Alias key.
- **expr** (*str*) – Alias value.
- **replace** (*bool, optional*) – Replace the current key-value pair if it already exists.

Variables ***Alias.aliases*** (*dict*) – Class variable for storing all known aliases

Examples

```
>>> Alias('atr', 'ma_truerange')
>>> Alias('hc', 'higher_close')
```

aliases = {}

alphapy.alias.**get_alias**(*alias*)
Find an alias value with the given key.

Parameters **alias** (*str*) – Key for finding the alias value.

Returns **alias_value** – Value for the corresponding key.

Return type str

Examples

```
>>> alias_value = get_alias('atr')
>>> alias_value = get_alias('hc')
```

14.1.4 alphapy.analysis module

class alphapy.analysis.**Analysis**(*model*, *group*)
Bases: object

Create a new analysis for a group. All analyses are stored in Analysis.analyses. Duplicate keys are not allowed.

Parameters

- **model** (*alphapy.Model*) – Model object for the analysis.
- **group** (*alphapy.Group*) – The group of members in the analysis.

Variables ***Analysis.analyses*** (*dict*) – Class variable for storing all known analyses

analyses = {}

alphapy.analysis.**analysis_name**(*gname*, *target*)
Get the name of the analysis.

Parameters

- **gname** (*str*) – Group name.
- **target** (*str*) – Target of the analysis.

Returns **name** – Value for the corresponding key.

Return type str

alphapy.analysis.**run_analysis**(*analysis*, *lag_period*, *forecast_period*, *leaders*, *predict_history*, *splits=True*)
Run an analysis for a given model and group.

First, the data are loaded for each member of the analysis group. Then, the target value is lagged for the forecast_period, and any leaders are lagged as well. Each frame is split along the predict_date from the analysis, and finally the train and test files are generated.

Parameters

- **analysis** (*alphapy.Analysis*) – The analysis to run.
- **lag_period** (*int*) – The number of lagged features for the analysis.
- **forecast_period** (*int*) – The period for forecasting the target of the analysis.
- **leaders** (*list*) – The features that are contemporaneous with the target.
- **predict_history** (*int*) – The number of periods required for lookback calculations.
- **splits** (*bool, optional*) – If True, then the data for each member of the analysis group are in separate files.

Returns **analysis** – The completed analysis.

Return type alphapy.Analysis

14.1.5 alphapy.calendrical module

alphapy.calendrical.**biz_day_month**(*rdate*)
Calculate the business day of the month.

Parameters **rdate** (*int*) – RDate date format.

Returns **bdm** – Business day of month.

Return type int

alphapy.calendrical.**biz_day_week**(*rdate*)
Calculate the business day of the week.

Parameters **rdate** (*int*) – RDate date format.

Returns **bdw** – Business day of week.

Return type int

alphapy.calendrical.**christmas_day**(*gyear*, *observed*)
Get Christmas Day for a given year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **observed** (*bool*) – False if the exact date, True if the weekday.

Returns **xmas** – Christmas Day in RDate format.

Return type int

alphapy.calendrical.**cinco_de_mayo**(*gyear*)
Get Cinco de Mayo for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **cinco_de_mayo** – Cinco de Mayo in RDate format.

Return type int

alphapy.calendrical.**day_of_week**(*rdate*)
Get the ordinal day of the week.

Parameters **rdate** (*int*) – RDate date format.

Returns **dw** – Ordinal day of the week.

Return type int

alphapy.calendrical.**day_of_year**(*gyear*, *gmonth*, *gday*)
Calculate the day number of the given calendar year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **dy** – Day number of year in RDate format.

Return type int

alphapy.calendrical.**days_left_in_year**(*gyear*, *gmonth*, *gday*)
Calculate the number of days remaining in the calendar year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **days_left** – Calendar days remaining in RDate format.

Return type int

`alphapy.calendrical.`**`easter_day`**(*gyear*)
Get Easter Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **ed** – Easter Day in RDate format.

Return type int

`alphapy.calendrical.`**`expand_dates`**(*date_list*)

`alphapy.calendrical.`**`fathers_day`**(*gyear*)
Get Father's Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **fathers_day** – Father's Day in RDate format.

Return type int

`alphapy.calendrical.`**`first_kday`**(*k*, *gyear*, *gmonth*, *gday*)
Calculate the first kday in RDate format.

Parameters

- **k** (*int*) – Day of the week.
- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **fkd** – first-kday in RDate format.

Return type int

`alphapy.calendrical.`**`gdate_to_rdate`**(*gyear*, *gmonth*, *gday*)
Convert Gregorian date to RDate format.

Parameters

- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **rdate** – RDate date format.

Return type int

`alphapy.calendrical.`**`get_holiday_names`**()
Get the list of defined holidays.

Returns **holidays** – List of holiday names.

Return type list of str

alphapy.calendrical.**get_nth_kday_of_month**(*gday*, *gmonth*, *gyear*)
Convert Gregorian date to RDate format.

Parameters

- **gday** (*int*) – Gregorian day.
- **gmonth** (*int*) – Gregorian month.
- **gyear** (*int*) – Gregorian year.

Returns **nth** – Ordinal number of a given day's occurrence within the month, for example, the third Friday of the month.

Return type int

alphapy.calendrical.**get_rdate**(*row*)
Extract RDate from a dataframe.

Parameters **row** (*pandas.DataFrame*) – Row of a dataframe containing year, month, and day.

Returns **rdate** – RDate date format.

Return type int

alphapy.calendrical.**good_friday**(*gyear*)
Get Good Friday for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **gf** – Good Friday in RDate format.

Return type int

alphapy.calendrical.**halloween**(*gyear*)
Get Halloween for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **halloween** – Halloween in RDate format.

Return type int

alphapy.calendrical.**independence_day**(*gyear*, *observed*)
Get Independence Day for a given year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **observed** (*bool*) – False if the exact date, True if the weekday.

Returns **d4j** – Independence Day in RDate format.

Return type int

alphapy.calendrical.**kday_after**(*rdate*, *k*)
Calculate the day after a given RDate.

Parameters

- **rdate** (*int*) – RDate date format.
- **k** (*int*) – Day of the week.

Returns **kda** – kday-after in RDate format.

Return type int

alphapy.calendrical.**kday_before**(*rdate*, *k*)
Calculate the day before a given RDate.

Parameters

- **rdate** (*int*) – RDate date format.
- **k** (*int*) – Day of the week.

Returns **kdb** – kday-before in RDate format.

Return type int

alphapy.calendrical.**kday_nearest**(*rdate*, *k*)
Calculate the day nearest a given RDate.

Parameters

- **rdate** (*int*) – RDate date format.
- **k** (*int*) – Day of the week.

Returns **kdn** – kday-nearest in RDate format.

Return type int

alphapy.calendrical.**kday_on_after**(*rdate*, *k*)
Calculate the day on or after a given RDate.

Parameters

- **rdate** (*int*) – RDate date format.
- **k** (*int*) – Day of the week.

Returns **kdoa** – kday-on-or-after in RDate format.

Return type int

alphapy.calendrical.**kday_on_before**(*rdate*, *k*)
Calculate the day on or before a given RDate.

Parameters

- **rdate** (*int*) – RDate date format.
- **k** (*int*) – Day of the week.

Returns **kdob** – kday-on-or-before in RDate format.

Return type int

alphapy.calendrical.**labor_day**(*gyear*)
Get Labor Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **lday** – Labor Day in RDate format.

Return type int

alphapy.calendrical.**last_kday**(*k*, *gyear*, *gmonth*, *gday*)
Calculate the last kday in RDate format.

Parameters

- **k** (*int*) – Day of the week.
- **gyear** (*int*) – Gregorian year.

- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **lkd** – last-kday in RDate format.

Return type int

alphapy.calendrical.**leap_year**(*gyear*)
Determine if this is a Gregorian leap year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **leap_year** – True if a Gregorian leap year, else False.

Return type bool

alphapy.calendrical.**memorial_day**(*gyear*)
Get Memorial Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **md** – Memorial Day in RDate format.

Return type int

alphapy.calendrical.**mlk_day**(*gyear*)
Get Martin Luther King Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **mlkday** – Martin Luther King Day in RDate format.

Return type int

alphapy.calendrical.**mothers_day**(*gyear*)
Get Mother's Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **mothers_day** – Mother's Day in RDate format.

Return type int

alphapy.calendrical.**new_years_day**(*gyear*, *observed*)
Get New Year's day for a given year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **observed** (*bool*) – False if the exact date, True if the weekday.

Returns **nyday** – New Year's Day in RDate format.

Return type int

alphapy.calendrical.**next_event**(*rdate*, *events*)
Find the next event after a given date.

Parameters

- **rdate** (*int*) – RDate date format.
- **events** (*list of RDate (int)*) – Monthly events in RDate format.

Returns **event** – Next event in RDate format.

Return type RDate (int)

alphapy.calendrical.**next_holiday**(*rdate*, *holidays*)
Find the next holiday after a given date.

Parameters

- **rdate** (*int*) – RDate date format.
- **holidays** (*dict of RDate (int)*) – Holidays in RDate format.

Returns **holiday** – Next holiday in RDate format.

Return type RDate (int)

alphapy.calendrical.**nth_bizday**(*n*, *gyear*, *gmonth*)
Calculate the nth business day in a month.

Parameters

- **n** (*int*) – Number of the business day to get.
- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.

Returns **bizday** – Nth business day of a given month in RDate format.

Return type int

alphapy.calendrical.**nth_kday**(*n*, *k*, *gyear*, *gmonth*, *gday*)
Calculate the nth-kday in RDate format.

Parameters

- **n** (*int*) – Occurrence of a given day counting in either direction.
- **k** (*int*) – Day of the week.
- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

Returns **nthkday** – nth-kday in RDate format.

Return type int

alphapy.calendrical.**presidents_day**(*gyear*)
Get President's Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **prezday** – President's Day in RDate format.

Return type int

alphapy.calendrical.**previous_event**(*rdate*, *events*)
Find the previous event before a given date.

Parameters

- **rdate** (*int*) – RDate date format.
- **events** (*list of RDate (int)*) – Monthly events in RDate format.

Returns **event** – Previous event in RDate format.

Return type RDate (int)

alphapy.calendrical.**previous_holiday**(*rdate*, *holidays*)
Find the previous holiday before a given date.

Parameters

- **rdate** (*int*) – RDate date format.
- **holidays** (*dict of RDate (int)*) – Holidays in RDate format.

Returns **holiday** – Previous holiday in RDate format.

Return type RDate (int)

alphapy.calendrical.**rdate_to_gdate**(*rdate*)
Convert RDate format to Gregorian date format.

Parameters **rdate** (*int*) – RDate date format.

Returns

- **gyear** (*int*) – Gregorian year.
- **gmonth** (*int*) – Gregorian month.
- **gday** (*int*) – Gregorian day.

alphapy.calendrical.**rdate_to_gyear**(*rdate*)
Convert RDate format to Gregorian year.

Parameters **rdate** (*int*) – RDate date format.

Returns **gyear** – Gregorian year.

Return type int

alphapy.calendrical.**saint_patricks_day**(*gyear*)
Get Saint Patrick's day for a given year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **observed** (*bool*) – False if the exact date, True if the weekday.

Returns **patricks** – Saint Patrick's Day in RDate format.

Return type int

alphapy.calendrical.**set_events**(*n*, *k*, *gyear*, *gday*)
Define monthly events for a given year.

Parameters

- **n** (*int*) – Occurrence of a given day counting in either direction.
- **k** (*int*) – Day of the week.
- **gyear** (*int*) – Gregorian year for the events.
- **gday** (*int*) – Gregorian day representing the first day to consider.

Returns **events** – Monthly events in RDate format.

Return type list of RDate (int)

Example

```
>>> # Options Expiration (Third Friday of every month)
>>> set_events(3, 5, 2017, 1)
```

alphapy.calendrical.**set_holidays**(*gyear*, *observe*)
Determine if this is a Gregorian leap year.

Parameters

- **gyear** (*int*) – Value for the corresponding key.
- **observe** (*bool*) – True to get the observed date, otherwise False.

Returns **holidays** – Set of holidays in RDate format for a given year.

Return type dict of int

alphapy.calendrical.**subtract_dates**(*gyear1*, *gmonth1*, *gday1*, *gyear2*, *gmonth2*, *gday2*)
Calculate the difference between two Gregorian dates.

Parameters

- **gyear1** (*int*) – Gregorian year of first date.
- **gmonth1** (*int*) – Gregorian month of first date.
- **gday1** (*int*) – Gregorian day of first date.
- **gyear2** (*int*) – Gregorian year of successive date.
- **gmonth2** (*int*) – Gregorian month of successive date.
- **gday2** (*int*) – Gregorian day of successive date.

Returns **delta_days** – Difference in days in RDate format.

Return type int

alphapy.calendrical.**thanksgiving_day**(*gyear*)
Get Thanksgiving Day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **tday** – Thanksgiving Day in RDate format.

Return type int

alphapy.calendrical.**valentines_day**(*gyear*)
Get Valentine's day for a given year.

Parameters **gyear** (*int*) – Gregorian year.

Returns **valentines** – Valentine's Day in RDate format.

Return type int

alphapy.calendrical.**veterans_day**(*gyear*, *observed*)
Get Veteran's day for a given year.

Parameters

- **gyear** (*int*) – Gregorian year.
- **observed** (*bool*) – False if the exact date, True if the weekday.

Returns **veterans** – Veteran's Day in RDate format.

Return type int

14.1.6 alphapy.data module

`alphapy.data.`**`convert_data`**(*df*, *index_column*, *intraday_data*)
Convert the market data frame to canonical format.

Parameters

- **df** (*pandas.DataFrame*) – The intraday dataframe.
- **index_column** (*str*) – The name of the index column.
- **intraday_data** (*bool*) – Flag set to True if the frame contains intraday data.

Returns **df** – The canonical dataframe with date/time index.

Return type pandas.DataFrame

`alphapy.data.`**`enhance_intraday_data`**(*df*)
Add columns to the intraday dataframe.

Parameters **df** (*pandas.DataFrame*) – The intraday dataframe.

Returns **df** – The dataframe with bar number and end-of-day columns.

Return type pandas.DataFrame

`alphapy.data.`**`get_data`**(*model*, *partition*)
Get data for the given partition.

Parameters

- **model** (*alphapy.Model*) – The model object describing the data.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns

- **X** (*pandas.DataFrame*) – The feature set.
- **y** (*pandas.Series*) – The array of target values, if available.

`alphapy.data.`**`get_google_data`**(*schema*, *subschema*, *symbol*, *intraday_data*, *data_fractal*, *from_date*, *to_date*, *lookback_period*)
Get data from Google.

Parameters

- **schema** (*str*) – The schema (including any subschema) for this data feed.
- **subschema** (*str*) – Any subschema for this data feed.
- **symbol** (*str*) – A valid stock symbol.
- **intraday_data** (*bool*) – If True, then get intraday data.
- **data_fractal** (*str*) – Pandas offset alias.
- **from_date** (*str*) – Starting date for symbol retrieval.
- **to_date** (*str*) – Ending date for symbol retrieval.
- **lookback_period** (*int*) – The number of periods of data to retrieve.

Returns **df** – The dataframe containing the market data.

Return type pandas.DataFrame

alphapy.data.**get_google_intraday_data**(*symbol*, *lookback_period*, *fractal*)

Get Google Finance intraday data.

We get intraday data from the Google Finance API, even though it is not officially supported. You can retrieve a maximum of 50 days of history, so you may want to build your own database for more extensive backtesting.

Parameters

- **symbol** (*str*) – A valid stock symbol.
- **lookback_period** (*int*) – The number of days of intraday data to retrieve, capped at 50.
- **fractal** (*str*) – The intraday frequency, e.g., "5m" for 5-minute data.

Returns **df** – The dataframe containing the intraday data.

Return type pandas.DataFrame

alphapy.data.**get_iex_data**(*schema*, *subschema*, *symbol*, *intraday_data*, *data_fractal*, *from_date*, *to_date*, *lookback_period*)

Get data from IEX.

Parameters

- **schema** (*str*) – The schema (including any subschema) for this data feed.
- **subschema** (*str*) – Any subschema for this data feed.
- **symbol** (*str*) – A valid stock symbol.
- **intraday_data** (*bool*) – If True, then get intraday data.
- **data_fractal** (*str*) – Pandas offset alias.
- **from_date** (*str*) – Starting date for symbol retrieval.
- **to_date** (*str*) – Ending date for symbol retrieval.
- **lookback_period** (*int*) – The number of periods of data to retrieve.

Returns **df** – The dataframe containing the market data.

Return type pandas.DataFrame

alphapy.data.**get_market_data**(*model*, *market_specs*, *group*, *lookback_period*, *intraday_data=False*)

Get data from an external feed.

Parameters

- **model** (*alphapy.Model*) – The model object describing the data.
- **market_specs** (*dict*) – The specifications for controlling the MarketFlow pipeline.
- **group** (*alphapy.Group*) – The group of symbols.
- **lookback_period** (*int*) – The number of periods of data to retrieve.
- **intraday_data** (*bool*) – If True, then get intraday data.

Returns **n_periods** – The maximum number of periods actually retrieved.

Return type int

alphapy.data.**get_pandas_data**(*schema*, *subschema*, *symbol*, *intraday_data*, *data_fractal*, *from_date*, *to_date*, *lookback_period*)

Get Pandas Web Reader data.

Parameters

- **schema** (*str*) – The schema (including any subschema) for this data feed.
- **subschema** (*str*) – Any subschema for this data feed.
- **symbol** (*str*) – A valid stock symbol.
- **intraday_data** (*bool*) – If True, then get intraday data.
- **data_fractal** (*str*) – Pandas offset alias.
- **from_date** (*str*) – Starting date for symbol retrieval.
- **to_date** (*str*) – Ending date for symbol retrieval.
- **lookback_period** (*int*) – The number of periods of data to retrieve.

Returns **df** – The dataframe containing the market data.

Return type pandas.DataFrame

`alphapy.data.`**`get_quandl_data`**(*schema*, *subschema*, *symbol*, *intraday_data*, *data_fractal*, *from_date*, *to_date*, *lookback_period*)

Get Quandl data.

Parameters

- **schema** (*str*) – The schema for this data feed.
- **subschema** (*str*) – Any subschema for this data feed.
- **symbol** (*str*) – A valid stock symbol.
- **intraday_data** (*bool*) – If True, then get intraday data.
- **data_fractal** (*str*) – Pandas offset alias.
- **from_date** (*str*) – Starting date for symbol retrieval.
- **to_date** (*str*) – Ending date for symbol retrieval.
- **lookback_period** (*int*) – The number of periods of data to retrieve.

Returns **df** – The dataframe containing the market data.

Return type pandas.DataFrame

`alphapy.data.`**`get_yahoo_data`**(*schema*, *subschema*, *symbol*, *intraday_data*, *data_fractal*, *from_date*, *to_date*, *lookback_period*)

Get Yahoo data.

Parameters

- **schema** (*str*) – The schema (including any subschema) for this data feed.
- **subschema** (*str*) – Any subschema for this data feed.
- **symbol** (*str*) – A valid stock symbol.
- **intraday_data** (*bool*) – If True, then get intraday data.
- **data_fractal** (*str*) – Pandas offset alias.
- **from_date** (*str*) – Starting date for symbol retrieval.
- **to_date** (*str*) – Ending date for symbol retrieval.
- **lookback_period** (*int*) – The number of periods of data to retrieve.

Returns **df** – The dataframe containing the market data.

Return type pandas.DataFrame

alphapy.data.**sample_data**(*model*)
Sample the training data.

Sampling is configured in the `model.yml` file (data:sampling:method) You can learn more about resampling techniques here [IMB].

Parameters **model** (*alphapy.Model*) – The model object describing the data.

Returns **model** – The model object with the sampled data.

Return type alphapy.Model

alphapy.data.**shuffle_data**(*model*)
Randomly shuffle the training data.

Parameters **model** (*alphapy.Model*) – The model object describing the data.

Returns **model** – The model object with the shuffled data.

Return type alphapy.Model

14.1.7 alphapy.estimators module

class alphapy.estimators.**Estimator**(*algorithm*, *model_type*, *estimator*, *grid*)
Bases: `object`

Store information about each estimator.

Parameters

- **algorithm** (*str*) – Abbreviation representing the given algorithm.
- **model_type** (*enum ModelType*) – The machine learning task for this algorithm.
- **estimator** (*function*) – A scikit-learn, TensorFlow, or XGBoost function.
- **grid** (*dict*) – The dictionary of hyperparameters for grid search.

alphapy.estimators.**create_keras_model**(*nlayers*, *layer1=None*, *layer2=None*, *layer3=None*, *layer4=None*, *layer5=None*, *layer6=None*, *layer7=None*, *layer8=None*, *layer9=None*, *layer10=None*, *optimizer=None*, *loss=None*, *metrics=None*)
Create a Keras Sequential model.

Parameters

- **nlayers** (*int*) – Number of layers of the Sequential model.
- **layer1...layer10** (*str*) – Ordered layers of the Sequential model.
- **optimizer** (*str*) – Compiler optimizer for the Sequential model.
- **loss** (*str*) – Compiler loss function for the Sequential model.
- **metrics** (*str*) – Compiler evaluation metric for the Sequential model.

Returns **model** – Compiled Keras Sequential Model.

Return type keras.models.Sequential

alphapy.estimators.**find_optional_packages**()

alphapy.estimators.**get_algos_config**(*cfg_dir*)
Read the algorithms configuration file.

Parameters **cfg_dir** (*str*) – The directory where the configuration file `algos.yml` is stored.

Returns **specs** – The specifications for determining which algorithms to run.

Return type dict

`alphapy.estimators.`**`get_estimators`**(*model*)
Define all the AlphaPy estimators based on the contents of the `algos.yml` file.

Parameters **model** (*alphapy.Model*) – The model object containing global AlphaPy parameters.

Returns **estimators** – All of the estimators required for running the pipeline.

Return type dict

14.1.8 alphapy.features module

`alphapy.features.`**`apply_transform`**(*fname*, *df*, *fparams*)
Apply a transform function to a column of the dataframe.

Parameters

- **fname** (*str*) – Name of the column to be treated in the dataframe `df`.
- **df** (*pandas.DataFrame*) – Dataframe containing the column `fname`.
- **fparams** (*list*) – The module, function, and parameter list of the transform function

Returns **new_features** – The set of features after applying a transform function.

Return type pandas.DataFrame

`alphapy.features.`**`apply_transforms`**(*model*, *X*)
Apply special functions to the original features.

Parameters

- **model** (*alphapy.Model*) – Model specifications indicating any transforms.
- **X** (*pandas.DataFrame*) – Combined train and test data, or just prediction data.

Returns **all_features** – All features, including transforms.

Return type pandas.DataFrame

Raises **`IndexError`** – The number of transform rows must match the number of rows in `X`.

`alphapy.features.`**`create_clusters`**(*features*, *model*)
Cluster the given features.

Parameters

- **features** (*numpy array*) – The features to cluster.
- **model** (*alphapy.Model*) – The model object with the clustering parameters.

Returns

- **cfeatures** (*numpy array*) – The calculated clusters.
- **cnames** (*list*) – The cluster feature names.

References

You can find more information on clustering here [CLUS].

alphapy.features.**create_crosstabs**(*model*)
Create cross-tabulations for categorical variables.

Parameters **model** (*alphapy.Model*) – The model object containing the data.

Returns **model** – The model object with the updated feature map.

Return type alphapy.Model

alphapy.features.**create_features**(*model*, *X*, *X_train*, *X_test*, *y_train*)
Create features for the train and test set.

Parameters

- **model** (*alphapy.Model*) – Model object with the feature specifications.
- **X** (*pandas.DataFrame*) – Combined train and test data.
- **X_train** (*pandas.DataFrame*) – Training data.
- **X_test** (*pandas.DataFrame*) – Testing data.
- **y_train** (*pandas.DataFrame*) – Target variable for training data.

Returns **all_features** – The new features.

Return type numpy array

Raises **TypeError** – Unrecognized data type.

alphapy.features.**create_interactions**(*model*, *X*)
Create feature interactions based on the model specifications.

Parameters

- **model** (*alphapy.Model*) – Model object with train and test data.
- **X** (*numpy array*) – Feature Matrix.

Returns **all_features** – The new interaction features.

Return type numpy array

Raises **TypeError** – Unknown model type when creating interactions.

alphapy.features.**create_isomap_features**(*features*, *model*)
Create Isomap features.

Parameters

- **features** (*numpy array*) – The input features.
- **model** (*alphapy.Model*) – The model object with the Isomap parameters.

Returns

- **ifeatures** (*numpy array*) – The Isomap features.
- **inames** (*list*) – The Isomap feature names.

Notes

Isomaps are very memory-intensive. Your process will be killed if you run out of memory.

References

You can find more information on Principal Component Analysis here [ISO].

`alphapy.features.`**`create_numpy_features`**(*base_features*, *sentinel*)
Calculate the sum, mean, standard deviation, and variance of each row.

Parameters

- **base_features** (*numpy array*) – The feature dataframe.
- **sentinel** (*float*) – The number to be imputed for NaN values.

Returns

- **np_features** (*numpy array*) – The calculated NumPy features.
- **np_fnames** (*list*) – The NumPy feature names.

`alphapy.features.`**`create_pca_features`**(*features*, *model*)
Apply Principal Component Analysis (PCA) to the features.

Parameters

- **features** (*numpy array*) – The input features.
- **model** (*alphapy.Model*) – The model object with the PCA parameters.

Returns

- **pfeatures** (*numpy array*) – The PCA features.
- **pnames** (*list*) – The PCA feature names.

References

You can find more information on Principal Component Analysis here [PCA].

`alphapy.features.`**`create_scipy_features`**(*base_features*, *sentinel*)
Calculate the skew, kurtosis, and other statistical features for each row.

Parameters

- **base_features** (*numpy array*) – The feature dataframe.
- **sentinel** (*float*) – The number to be imputed for NaN values.

Returns

- **sp_features** (*numpy array*) – The calculated SciPy features.
- **sp_fnames** (*list*) – The SciPy feature names.

`alphapy.features.`**`create_tsne_features`**(*features*, *model*)
Create t-SNE features.

Parameters

- **features** (*numpy array*) – The input features.
- **model** (*alphapy.Model*) – The model object with the t-SNE parameters.

Returns

- **tfeatures** (*numpy array*) – The t-SNE features.
- **tnames** (*list*) – The t-SNE feature names.

References

You can find more information on the t-SNE technique here [TSNE].

`alphapy.features.`**`drop_features`**(*X*, *drop*)

Drop any specified features.

Parameters

- **X** (*pandas.DataFrame*) – The dataframe containing the features.
- **drop** (*list*) – The list of features to remove from `X`.

Returns **X** – The dataframe without the dropped features.

Return type pandas.DataFrame

`alphapy.features.`**`float_factor`**(*x*, *rounding*)

Convert a floating point number to a factor.

Parameters

- **x** (*float*) – The value to convert to a factor.
- **rounding** (*int*) – The number of places to round.

Returns **ffactor** – The resulting factor.

Return type int

`alphapy.features.`**`get_factors`**(*model*, *X_train*, *X_test*, *y_train*, *fnum*, *fname*, *nvalues*, *dtype*, *encoder*, *rounding*, *sentinel*)

Convert the original feature to a factor.

Parameters

- **model** (*alphapy.Model*) – Model object with the feature specifications.
- **X_train** (*pandas.DataFrame*) – Training dataframe containing the column `fname`.
- **X_test** (*pandas.DataFrame*) – Testing dataframe containing the column `fname`.
- **y_train** (*pandas.Series*) – Training series for target variable.
- **fnum** (*int*) – Feature number, strictly for logging purposes
- **fname** (*str*) – Name of the text column in the dataframe `df`.
- **nvalues** (*int*) – The number of unique values.
- **dtype** (*str*) – The values `'float64'`, `'int64'`, or `'bool'`.
- **encoder** (*alphapy.features.Encoders*) – Type of encoder to apply.
- **rounding** (*int*) – Number of places to round.
- **sentinel** (*float*) – The number to be imputed for NaN values.

Returns

- **all_features** (*numpy array*) – The features that have been transformed to factors.

- **all_fnames** (*list*) – The feature names for the encodings.

alphapy.features.**get_numerical_features**(*fnum*, *fname*, *df*, *nvalues*, *dt*, *sentinel*, *logt*, *plevel*)
Transform numerical features with imputation and possibly log-transformation.

Parameters

- **fnum** (*int*) – Feature number, strictly for logging purposes
- **fname** (*str*) – Name of the numerical column in the dataframe `df`.
- **df** (*pandas.DataFrame*) – Dataframe containing the column `fname`.
- **nvalues** (*int*) – The number of unique values.
- **dt** (*str*) – The values `'float64'`, `'int64'`, or `'bool'`.
- **sentinel** (*float*) – The number to be imputed for NaN values.
- **logt** (*bool*) – If `True`, then log-transform numerical values.
- **plevel** (*float*) – The p-value threshold to test if a feature is normally distributed.

Returns

- **new_values** (*numpy array*) – The set of imputed and transformed features.
- **new_fnames** (*list*) – The new feature name(s) for the numerical variable.

alphapy.features.**get_polynomials**(*features*, *poly_degree*)
Generate interactions that are products of distinct features.

Parameters

- **features** (*pandas.DataFrame*) – Dataframe containing the features for generating interactions.
- **poly_degree** (*int*) – The degree of the polynomial features.

Returns

- **poly_features** (*numpy array*) – The interaction features only.
- **poly_fnames** (*list*) – List of polynomial feature names.

References

You can find more information on polynomial interactions here [POLY].

alphapy.features.**get_text_features**(*fnum*, *fname*, *df*, *nvalues*, *vectorize*, *ngrams_max*)
Transform text features with count vectorization and TF-IDF, or alternatively factorization.

Parameters

- **fnum** (*int*) – Feature number, strictly for logging purposes
- **fname** (*str*) – Name of the text column in the dataframe `df`.
- **df** (*pandas.DataFrame*) – Dataframe containing the column `fname`.
- **nvalues** (*int*) – The number of unique values.
- **vectorize** (*bool*) – If `True`, then attempt count vectorization.
- **ngrams_max** (*int*) – The maximum number of n-grams for count vectorization.

Returns

- **new_features** (*numpy array*) – The vectorized or factorized text features.
- **new_fnames** (*list*) – The new feature name(s) for the numerical variable.

References

To use count vectorization and TF-IDF, you can find more information here [TFE].

`alphapy.features.`**`impute_values`**(*feature*, *dt*, *sentinel*)

Impute values for a given data type. The *median* strategy is applied for floating point values, and the *most frequent* strategy is applied for integer or Boolean values.

Parameters

- **feature** (*pandas.Series or numpy.array*) – The feature for imputation.
- **dt** (*str*) – The values `'float64'`, `'int64'`, or `'bool'`.
- **sentinel** (*float*) – The number to be imputed for NaN values.

Returns **imputed** – The feature after imputation.

Return type numpy.array

Raises **`TypeError`** – Data type `dt` is invalid for imputation.

References

You can find more information on feature imputation here [IMP].

`alphapy.features.`**`remove_lv_features`**(*model*, *X*)

Remove low-variance features.

Parameters

- **model** (*alphapy.Model*) – Model specifications for removing features.
- **X** (*numpy array*) – The feature matrix.

Returns **X_reduced** – The reduced feature matrix.

Return type numpy array

References

You can find more information on low-variance feature selection here [LV].

`alphapy.features.`**`save_features`**(*model*, *X_train*, *X_test*, *y_train=None*, *y_test=None*)

Save new features to the model.

Parameters

- **model** (*alphapy.Model*) – Model object with train and test data.
- **X_train** (*numpy array*) – Training features.
- **X_test** (*numpy array*) – Testing features.
- **y_train** (*numpy array*) – Training labels.
- **y_test** (*numpy array*) – Testing labels.

Returns **model** – Model object with new train and test data.

Return type alphapy.Model

alphapy.features.**select_features**(*model*)
Select features with univariate selection.

Parameters **model** (*alphapy.Model*) – Model object with the feature selection specifications.

Returns **model** – Model object with the revised number of features.

Return type alphapy.Model

References

You can find more information on univariate feature selection here [UNI].

14.1.9 alphapy.frame module

class alphapy.frame.**Frame**(*name*, *space*, *df*)
Bases: object

Create a new Frame that points to a dataframe in memory. All frames are stored in `Frame.frames`. Names must be unique.

Parameters

- **name** (*str*) – Frame key.
- **space** (*alphapy.Space*) – Namespace of the given frame.
- **df** (*pandas.DataFrame*) – The contents of the actual dataframe.

Variables ***frames*** (*dict*) – Class variable for storing all known frames

Examples

```
>>> Frame('tech', Space('stock', 'prices', '5m'), df)
```

frames = {}

alphapy.frame.**dump_frames**(*group*, *directory*, *extension*, *separator*)
Save a group of data frames to disk.

Parameters

- **group** (*alphapy.Group*) – The collection of frames to be saved to the file system.
- **directory** (*str*) – Full directory specification.
- **extension** (*str*) – File name extension, e.g., `csv`.
- **separator** (*str*) – The delimiter between fields in the file.

Returns None

Return type None

alphapy.frame.**frame_name**(*name*, *space*)
Get the frame name for the given name and space.

Parameters

- **name** (*str*) – Group name.

- **space** (*alphapy.Space*) – Context or namespace for the given group name.

Returns **fname** – Frame name.

Return type str

Examples

```
>>> fname = frame_name('tech', Space('stock', 'prices', '1d'))
# 'tech_stock_prices_1d'
```

`alphapy.frame.`**`load_frames`**(*group*, *directory*, *extension*, *separator*, *splits=False*)

Read a group of dataframes into memory.

Parameters

- **group** (*alphapy.Group*) – The collection of frames to be read into memory.
- **directory** (*str*) – Full directory specification.
- **extension** (*str*) – File name extension, e.g., `csv`.
- **separator** (*str*) – The delimiter between fields in the file.
- **splits** (*bool, optional*) – If `True`, then all the members of the group are stored in separate files corresponding with each member. If `False`, then the data are stored in a single file.

Returns **all_frames** – The list of pandas dataframes loaded from the file location. If the files cannot be located, then `None` is returned.

Return type list

`alphapy.frame.`**`read_frame`**(*directory*, *filename*, *extension*, *separator*, *index_col=None*, *squeeze=False*)

Read a delimiter-separated file into a data frame.

Parameters

- **directory** (*str*) – Full directory specification.
- **filename** (*str*) – Name of the file to read, excluding the `extension`.
- **extension** (*str*) – File name extension, e.g., `csv`.
- **separator** (*str*) – The delimiter between fields in the file.
- **index_col** (*str, optional*) – Column to use as the row labels in the dataframe.
- **squeeze** (*bool, optional*) – If the data contains only one column, then return a pandas Series.

Returns **df** – The pandas dataframe loaded from the file location. If the file cannot be located, then `None` is returned.

Return type pandas.DataFrame

`alphapy.frame.`**`sequence_frame`**(*df*, *target*, *forecast_period=1*, *leaders=[]*, *lag_period=1*)

Create sequences of lagging and leading values.

Parameters

- **df** (*pandas.DataFrame*) – The original dataframe.
- **target** (*str*) – The target variable for prediction.
- **forecast_period** (*int*) – The period for forecasting the target of the analysis.

- **leaders** (*list*) – The features that are contemporaneous with the target.
- **lag_period** (*int*) – The number of lagged rows for prediction.

Returns **new_frame** – The transformed dataframe with variable sequences.

Return type pandas.DataFrame

`alphapy.frame.`**`write_frame`**(*df*, *directory*, *filename*, *extension*, *separator*, *index=False*, *index_label=None*, *columns=None*)

Write a dataframe into a delimiter-separated file.

Parameters

- **df** (*pandas.DataFrame*) – The pandas dataframe to save to a file.
- **directory** (*str*) – Full directory specification.
- **filename** (*str*) – Name of the file to write, excluding the `extension`.
- **extension** (*str*) – File name extension, e.g., `csv`.
- **separator** (*str*) – The delimiter between fields in the file.
- **index** (*bool, optional*) – If `True`, write the row names (index).
- **index_label** (*str, optional*) – A column label for the `index`.
- **columns** (*str, optional*) – A list of column names.

Returns **None**

Return type None

14.1.10 alphapy.globals module

class `alphapy.globals.`**`Encoders`**(*value*)

Bases: `enum.Enum`

AlphaPy Encoders.

These are the encoders used in AlphaPy, as configured in the `model.yml` file (features:encoding:type) You can learn more about encoders here [ENC].

`backdiff = 1`

`basen = 2`

`binary = 3`

`catboost = 4`

`hashing = 5`

`helmert = 6`

`jstein = 7`

`leaveone = 8`

`mestimate = 9`

`onehot = 10`

`ordinal = 11`

`polynomial = 12`

sum = 13

target = 14

woe = 15

class alphapy.globals.**ModelType**(*value*)
Bases: enum.Enum

AlphaPy Model Types.

Note: One-Class Classification oneclass is not yet implemented.

classification = 1

clustering = 2

multiclass = 3

oneclass = 4

regression = 5

class alphapy.globals.**Objective**(*value*)
Bases: enum.Enum

Scoring Function Objectives.

Best model selection is based on the scoring or Objective function, which must be either maximized or minimized. For example, roc_auc is maximized, while neg_log_loss is minimized.

maximize = 1

minimize = 2

class alphapy.globals.**Orders**
Bases: object

System Order Types.

Variables

- ***le*** (*str*) – long entry
- ***se*** (*str*) – short entry
- ***lx*** (*str*) – long exit
- ***sx*** (*str*) – short exit
- ***lh*** (*str*) – long exit at the end of the holding period
- ***sh*** (*str*) – short exit at the end of the holding period

le = 'le'

lh = 'lh'

lx = 'lx'

se = 'se'

sh = 'sh'

sx = 'sx'

class alphapy.globals.**Partition**(*value*)
Bases: enum.Enum

AlphaPy Partitions.

predict = 1

test = 2

train = 3

class alphapy.globals.**SamplingMethod**(*value*)
Bases: enum.Enum

AlphaPy Sampling Methods.

These are the data sampling methods used in AlphaPy, as configured in the `model.yml` file (data:sampling:method) You can learn more about resampling techniques here [IMB].

ensemble_bc = 1

ensemble_easy = 2

over_random = 3

over_smote = 4

over_smoteb = 5

over_smotesv = 6

overunder_smote_enn = 7

overunder_smote_tomek = 8

under_cluster = 9

under_ncr = 10

under_nearmiss = 11

under_random = 12

under_tomek = 13

class alphapy.globals.**Scalers**(*value*)
Bases: enum.Enum

AlphaPy Scalers.

These are the scaling methods used in AlphaPy, as configured in the `model.yml` file (features:scaling:type) You can learn more about feature scaling here [SCALE].

minmax = 1

standard = 2

14.1.11 alphapy.group module

class alphapy.group.**Group**(*name*, *space=<alphapy.space.Space object>*, *dynamic=True*, *members={}*)

Bases: object

Create a new Group that contains common members. All defined groups are stored in `Group.groups`. Group names must be unique.

Parameters

- **name** (*str*) – Group name.
- **space** (*alphapy.Space, optional*) – Namespace for the given group.
- **dynamic** (bool, optional, default `True`) – Flag for defining whether or not the group membership can change.
- **members** (*set, optional*) – The initial members of the group, especially if the new group is fixed, e.g., not `dynamic`.

Variables ***groups*** (*dict*) – Class variable for storing all known groups

Examples

```
>>> Group('tech')
```

add(*newlist*)

Add new members to the group.

Parameters **newlist** (*list*) – New members or identifiers to add to the group.

Returns **None**

Return type None

Notes

New members cannot be added to a fixed or non-dynamic group.

groups = {}

member(*item*)

Find a member in the group.

Parameters **item** (*str*) – The member to find the group.

Returns **member_exists** – Flag indicating whether or not the member is in the group.

Return type bool

remove(*remlist*)

Read in data from the given directory in a given format.

Parameters **remlist** (*list*) – The list of members to remove from the group.

Returns **None**

Return type None

Notes

Members cannot be removed from a fixed or non-dynamic group.

14.1.12 alphapy.market_flow module

alphapy.market_flow.**get_market_config**()
Read the configuration file for MarketFlow.

Parameters **None** (*None*)

Returns **specs** – The parameters for controlling MarketFlow.

Return type dict

alphapy.market_flow.**main**(*args=None*)
MarketFlow Main Program

Notes

(1) Initialize logging.

(2) Parse the command line arguments.

(3) Get the market configuration.

(4) Get the model configuration.

(5) Create the model object.

(6) Call the main MarketFlow pipeline.

Raises **ValueError** – Training date must be before prediction date.

alphapy.market_flow.**market_pipeline**(*model*, *market_specs*)
AlphaPy MarketFlow Pipeline

Parameters

- **model** (*alphapy.Model*) – The model object for AlphaPy.
- **market_specs** (*dict*) – The specifications for controlling the MarketFlow pipeline.

Returns **model** – The final results are stored in the model object.

Return type alphapy.Model

Notes

(1) Define a group.

(2) Get the market data.

(3) Apply system features.

(4) Create an analysis.

(5) Run the analysis, which calls AlphaPy.

14.1.13 alphapy.model module

class alphapy.model.**Model**(*specs*)
Bases: object

Create a new model.

Parameters **specs** (*dict*) – The model specifications obtained by reading the `model.yml` file.

Variables

- **specs** (*dict*) – The model specifications.
- **X_train** (*pandas.DataFrame*) – Training features in matrix format.
- **X_test** (*pandas.Series*) – Testing features in matrix format.
- **y_train** (*pandas.DataFrame*) – Training labels in vector format.
- **y_test** (*pandas.Series*) – Testing labels in vector format.
- **algolist** (*list*) – Algorithms to use in training.
- **estimators** (*dict*) – Dictionary of estimators (key: algorithm)
- **importances** (*dict*) – Feature Importances (key: algorithm)
- **coefs** (*dict*) – Coefficients, if applicable (key: algorithm)
- **support** (*dict*) – Support Vectors, if applicable (key: algorithm)
- **preds** (*dict*) – Predictions or labels (keys: algorithm, partition)
- **probas** (*dict*) – Probabilities from classification (keys: algorithm, partition)
- **metrics** (*dict*) – Model evaluation metrics (keys: algorith, partition, metric)

Raises **KeyError** – Model specs must include the key *algorithms*, which is stored in `algolist`.

alphapy.model.**first_fit**(*model*, *algo*, *est*)
Fit the model before optimization.

Parameters

- **model** (*alphapy.Model*) – The model object with specifications.
- **algo** (*str*) – Abbreviation of the algorithm to run.
- **est** (*alphapy.Estimator*) – The estimator to fit.

Returns **model** – The model object with the initial estimator.

Return type alphapy.Model

Notes

AlphaPy fits an initial model because the user may choose to get a first score without any additional feature selection or grid search. XGBoost is a special case because it has the advantage of an `eval_set` and `early_stopping_rounds`, which can speed up the estimation phase.

alphapy.model.**generate_metrics**(*model*, *partition*)
Generate model evaluation metrics for all estimators.

Parameters

- **model** (*alphapy.Model*) – The model object with stored predictions.

- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns **model** – The model object with the completed metrics.

Return type alphapy.Model

Notes

AlphaPy takes a brute-force approach to calculating each metric. It calls every scikit-learn function without exception. If the calculation fails for any reason, then the evaluation will still continue without error.

References

For more information about model evaluation and the associated metrics, refer to [EVAL].

`alphapy.model.`**`get_model_config`**`()`
Read in the configuration file for AlphaPy.

Parameters **None** (*None*)

Returns **specs** – The parameters for controlling AlphaPy.

Return type dict

Raises **`ValueError`** – Unrecognized value of a `model.yml` field.

`alphapy.model.`**`load_feature_map`**`(`*model*, *directory*`)`
Load the feature map from storage. By default, the most recent feature map is loaded into memory.

Parameters

- **model** (*alphapy.Model*) – The model object to contain the feature map.
- **directory** (*str*) – Full directory specification of the feature map's location.

Returns **model** – The model object containing the feature map.

Return type alphapy.Model

`alphapy.model.`**`load_predictor`**`(`*directory*`)`
Load the model predictor from storage. By default, the most recent model is loaded into memory.

Parameters **directory** (*str*) – Full directory specification of the predictor's location.

Returns **predictor** – The scoring function.

Return type function

`alphapy.model.`**`make_predictions`**`(`*model*, *algo*, *calibrate*`)`
Make predictions for the training and testing data.

Parameters

- **model** (*alphapy.Model*) – The model object with specifications.
- **algo** (*str*) – Abbreviation of the algorithm to make predictions.
- **calibrate** (*bool*) – If `True`, calibrate the probabilities for a classifier.

Returns **model** – The model object with the predictions.

Return type alphapy.Model

Notes

For classification, calibration is a precursor to making the actual predictions. In this case, AlphaPy predicts both labels and probabilities. For regression, real values are predicted.

alphapy.model.**predict_best**(*model*)
Select the best model based on score.

Parameters **model** (*alphapy.Model*) – The model object with all of the estimators.

Returns **model** – The model object with the best estimator.

Return type alphapy.Model

Notes

Best model selection is based on a scoring function. If the objective is to minimize (e.g., negative log loss), then we select the model with the algorithm that has the lowest score. If the objective is to maximize, then we select the algorithm with the highest score (e.g., AUC).

For multiple algorithms, AlphaPy always creates a blended model. Therefore, the best algorithm that is selected could actually be the blended model itself.

alphapy.model.**predict_blend**(*model*)
Make predictions from a blended model.

Parameters **model** (*alphapy.Model*) – The model object with all of the estimators.

Returns **model** – The model object with the blended estimator.

Return type alphapy.Model

Notes

For classification, AlphaPy uses logistic regression for creating a blended model. For regression, ridge regression is applied.

alphapy.model.**save_feature_map**(*model*, *timestamp*)
Save the feature map to disk.

Parameters

- **model** (*alphapy.Model*) – The model object containing the feature map.
- **timestamp** (*str*) – Date in yyyy-mm-dd format.

Returns **None**

Return type None

alphapy.model.**save_model**(*model*, *tag*, *partition*)
Save the results in the model file.

Parameters

- **model** (*alphapy.Model*) – The model object to save.
- **tag** (*str*) – A unique identifier for the output files, e.g., a date stamp.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns **None**

Return type None

Notes

The following components are extracted from the model object and saved to disk:

- Model predictor (via joblib/pickle)
- Predictions
- Probabilities (classification only)
- Rankings
- Submission File (optional)

`alphapy.model.`**`save_predictions`**(*model*, *tag*, *partition*)
Save the predictions to disk.

Parameters

- **model** (*alphapy.Model*) – The model object to save.
- **tag** (*str*) – A unique identifier for the output files, e.g., a date stamp.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns

- **preds** (*numpy array*) – The prediction vector.
- **probas** (*numpy array*) – The probability vector.

`alphapy.model.`**`save_predictor`**(*model*, *timestamp*)
Save the time-stamped model predictor to disk.

Parameters

- **model** (*alphapy.Model*) – The model object that contains the best estimator.
- **timestamp** (*str*) – Date in yyyy-mm-dd format.

Returns **None**

Return type None

14.1.14 alphapy.optimize module

`alphapy.optimize.`**`grid_report`**(*results*, *n_top=3*)
Report the top grid search scores.

Parameters

- **results** (*dict of numpy arrays*) – Mean test scores for each grid search iteration.
- **n_top** (*int, optional*) – The number of grid search results to report.

Returns **None**

Return type None

`alphapy.optimize.`**`hyper_grid_search`**(*model*, *estimator*)
Return the best hyperparameters for a grid search.

Parameters

- **model** (*alphapy.Model*) – The model object with grid search parameters.
- **estimator** (*alphapy.Estimator*) – The estimator containing the hyperparameter grid.

Returns **model** – The model object with the grid search estimator.

Return type alphapy.Model

Notes

To reduce the time required for grid search, use either randomized grid search with a fixed number of iterations or a full grid search with subsampling. AlphaPy uses the scikit-learn Pipeline with feature selection to reduce the feature space.

References

For more information about grid search, refer to [GRID].

To learn about pipelines, refer to [PIPE].

`alphapy.optimize.`**`rfecv_search`**(*model*, *algo*)

Return the best feature set using recursive feature elimination with cross-validation.

Parameters

- **model** (*alphapy.Model*) – The model object with RFE parameters.
- **algo** (*str*) – Abbreviation of the algorithm to run.

Returns **model** – The model object with the RFE support vector and the best estimator.

Return type alphapy.Model

Notes

If a scoring function is available, then AlphaPy can perform RFE with Cross-Validation (CV), as in this function; otherwise, it just does RFE without CV.

References

For more information about Recursive Feature Elimination, refer to [RFECV].

14.1.15 alphapy.plots module

`alphapy.plots.`**`generate_plots`**(*model*, *partition*)

Generate plots while running the pipeline.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns **None**

Return type None

alphapy.plots.**get_partition_data**(*model*, *partition*)
Get the X, y pair for a given model and partition

Parameters

- **model** (*alphapy.Model*) – The model object with partition data.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns

- **X** (*numpy array*) – The feature matrix.
- **y** (*numpy array*) – The target vector.

Raises **TypeError** – Partition must be train or test.

alphapy.plots.**get_plot_directory**(*model*)
Get the plot output directory of a model.

Parameters **model** (*alphapy.Model*) – The model object with directory information.

Returns **plot_directory** – The output directory to write the plot.

Return type str

alphapy.plots.**plot_boundary**(*model*, *partition*, *f1=0*, *f2=1*)
Display a comparison of classifiers

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.
- **f1** (*int*) – Number of the first feature to compare.
- **f2** (*int*) – Number of the second feature to compare.

Returns **None**

Return type None

References

Code excerpts from authors:

- Gael Varoquaux
- Andreas Muller

http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

alphapy.plots.**plot_box**(*df*, *x*, *y*, *hue*, *tag='eda'*, *directory=None*)
Display a Box Plot.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the `x` and `y` features.
- **x** (*str*) – Variable name in `df` to display along the x-axis.
- **y** (*str*) – Variable name in `df` to display along the y-axis.
- **hue** (*str*) – Variable name to be used as hue, i.e., another data dimension.
- **tag** (*str*) – Unique identifier for the plot.

- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

http://seaborn.pydata.org/generated/seaborn.boxplot.html

`alphapy.plots.`**`plot_calibration`**(*model*, *partition*)

Display scikit-learn calibration plots.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns None

Return type None

References

Code excerpts from authors:

- Alexandre Gramfort <alexandre.gramfort@telecom-paristech.fr>
- Jan Hendrik Metzen <jhm@informatik.uni-bremen.de>

http://scikit-learn.org/stable/auto_examples/calibration/plot_calibration_curve.html#sphx-glr-auto-examples-calibration-plot-calibration-curve-py

`alphapy.plots.`**`plot_candlestick`**(*df*, *symbol*, *datecol='date'*, *directory=None*)

Plot time series data.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the `target` feature.
- **symbol** (*str*) – Unique identifier of the data to plot.
- **datecol** (*str, optional*) – The name of the date column.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

Notes

The dataframe `df` must contain these columns:

- `open`
- `high`
- `low`
- `close`

References

http://bokeh.pydata.org/en/latest/docs/gallery/candlestick.html

alphapy.plots.**plot_confusion_matrix**(*model*, *partition*)
Draw the confusion matrix.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns None

Return type None

References

http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

alphapy.plots.**plot_distribution**(*df*, *target*, *tag='eda'*, *directory=None*)
Display a Distribution Plot.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the `target` feature.
- **target** (*str*) – The target variable for the distribution plot.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

http://seaborn.pydata.org/generated/seaborn.distplot.html

alphapy.plots.**plot_facet_grid**(*df*, *target*, *frow*, *fcol*, *tag='eda'*, *directory=None*)
Plot a Seaborn faceted histogram grid.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the features.
- **target** (*str*) – The target variable for contrast.
- **frow** (*list of str*) – Feature names for the row elements of the grid.
- **fcol** (*list of str*) – Feature names for the column elements of the grid.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

http://seaborn.pydata.org/generated/seaborn.FacetGrid.html

alphapy.plots.**plot_importance**(*model*, *partition*)
Display scikit-learn feature importances.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns None

Return type None

References

http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

alphapy.plots.**plot_learning_curve**(*model*, *partition*)
Generate learning curves for a given partition.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns None

Return type None

References

http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

alphapy.plots.**plot_partial_dependence**(*est*, *X*, *features*, *fnames*, *tag*, *n_jobs=- 1*, *verbosity=0*, *directory=None*)
Display a Partial Dependence Plot.

Parameters

- **est** (*estimator*) – The scikit-learn estimator for calculating partial dependence.
- **X** (*numpy array*) – The data on which the estimator was trained.
- **features** (*list of int*) – Feature numbers of `X`.
- **fnames** (*list of str*) – The feature names to plot.
- **tag** (*str*) – Unique identifier for the plot
- **n_jobs** (*int, optional*) – The maximum number of parallel jobs.
- **verbosity** (*int, optional*) – The amount of logging from 0 (minimum) and higher.
- **directory** (*str*) – Directory where the plot will be stored.

Returns None

Return type None.

References

http://scikit-learn.org/stable/auto_examples/ensemble/plot_partial_dependence.html#sphx-glr-auto-examples-ensemble-plot-partial-dependence-py

alphapy.plots.**plot_roc_curve**(*model*, *partition*)
Display ROC Curves with Cross-Validation.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.

Returns None

Return type None

References

http://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc

alphapy.plots.**plot_scatter**(*df*, *features*, *target*, *tag='eda'*, *directory=None*)
Plot a scatterplot matrix, also known as a pair plot.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the features.
- **features** (*list of str*) – The features to compare in the scatterplot.
- **target** (*str*) – The target variable for contrast.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

https://seaborn.pydata.org/examples/scatterplot_matrix.html

alphapy.plots.**plot_swarm**(*df*, *x*, *y*, *hue*, *tag='eda'*, *directory=None*)
Display a Swarm Plot.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the `x` and `y` features.
- **x** (*str*) – Variable name in `df` to display along the x-axis.
- **y** (*str*) – Variable name in `df` to display along the y-axis.
- **hue** (*str*) – Variable name to be used as hue, i.e., another data dimension.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

http://seaborn.pydata.org/generated/seaborn.swarmplot.html

alphapy.plots.**plot_time_series**(*df*, *target*, *tag='eda'*, *directory=None*)
Plot time series data.

Parameters

- **df** (*pandas.DataFrame*) – The dataframe containing the `target` feature.
- **target** (*str*) – The target variable for the time series plot.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification of the plot location.

Returns None

Return type None.

References

http://seaborn.pydata.org/generated/seaborn.tsplot.html

alphapy.plots.**plot_validation_curve**(*model*, *partition*, *pname*, *prange*)
Generate scikit-learn validation curves.

Parameters

- **model** (*alphapy.Model*) – The model object with plotting specifications.
- **partition** (*alphapy.Partition*) – Reference to the dataset.
- **pname** (*str*) – Name of the hyperparameter to test.
- **prange** (*numpy array*) – The values of the hyperparameter that will be evaluated.

Returns None

Return type None

References

http://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html#sphx-glr-auto-examples-model-selection-plot-validation-curve-py

alphapy.plots.**write_plot**(*vizlib*, *plot*, *plot_type*, *tag*, *directory=None*)
Save the plot to a file, or display it interactively.

Parameters

- **vizlib** (*str*) – The visualization library: `'matplotlib'`, `'seaborn'`, or `'bokeh'`.
- **plot** (*module*) – Plotting context, e.g., `plt`.
- **plot_type** (*str*) – Type of plot to generate.
- **tag** (*str*) – Unique identifier for the plot.
- **directory** (*str, optional*) – The full specification for the directory location. if `directory` is *None*, then the plot is displayed interactively.

Returns None

Return type None.

Raises **ValueError** – Unrecognized data visualization library.

References

Visualization Libraries:

- Matplotlib : http://matplotlib.org/
- Seaborn : https://seaborn.pydata.org/
- Bokeh : http://bokeh.pydata.org/en/latest/

14.1.16 alphapy.portfolio module

class alphapy.portfolio.**Portfolio**(*group_name*, *tag*, *space=<alphapy.space.Space object>*, *maxpos=10*, *posby='close'*, *kopos=0*, *koby='-profit'*, *restricted=False*, *weightby='quantity'*, *startcap=100000*, *margin=0.5*, *mincash=0.2*, *fixedfrac=0.1*, *maxloss=0.1*)

Bases: object

Create a new portfolio with a unique name. All portfolios are stored in `Portfolio.portfolios`.

Parameters

- **group_name** (*str*) – The group represented in the portfolio.
- **tag** (*str*) – A unique identifier.
- **space** (*alphapy.Space, optional*) – Namespace for the portfolio.
- **maxpos** (*int, optional*) – The maximum number of positions.
- **posby** (*str, optional*) – The denominator for position sizing.
- **kopos** (*int, optional*) – The number of positions to kick out from the portfolio.
- **koby** (*str, optional*) – The "kick out" criteria. For example, a `koby` value of '-profit' means the three least profitable positions will be closed.
- **restricted** (*bool, optional*) – If `True`, then the portfolio is limited to a maximum number of positions `maxpos`.
- **weightby** (*str, optional*) – The weighting variable to balance the portfolio, e.g., by closing price, by volatility, or by any column.
- **startcap** (*float, optional*) – The amount of starting capital.
- **margin** (*float, optional*) – The amount of margin required, expressed as a fraction.
- **mincash** (*float, optional*) – Minimum amount of cash on hand, expressed as a fraction of the total portfolio value.
- **fixedfrac** (*float, optional*) – The fixed fraction for any given position.
- **maxloss** (*float, optional*) – Stop loss for any given position.

Variables

- ***portfolios*** (*dict*) – Class variable for storing all known portfolios
- **value** (*float*) – Class variable for storing all known portfolios
- **netprofit** (*float*) – Net profit ($) since previous valuation.

- **netreturn** (*float*) – Net return (%) since previous valuation
- **totalprofit** (*float*) – Total profit ($) since inception.
- **totalreturn** (*float*) – Total return (%) since inception.

portfolios = {}

class alphapy.portfolio.**Position**(*portfolio*, *name*, *opendate*)

Bases: object

Create a new position in the portfolio.

Parameters

- **portfolio** (*alphaPy.portfolio*) – The portfolio that will contain the position.
- **name** (*str*) – A unique identifier such as a stock symbol.
- **opendate** (*datetime*) – Date the position is opened.

Variables

- **date** (*timedate*) – Current date of the position.
- **name** (*str*) – A unique identifier.
- **status** (*str*) – State of the position: 'opened' or 'closed'.
- **mpos** (*str*) – Market position 'long' or 'short'.
- **quantity** (*float*) – The net size of the position.
- **price** (*float*) – The current price of the instrument.
- **value** (*float*) – The total dollar value of the position.
- **profit** (*float*) – The net profit of the current position.
- **netreturn** (*float*) – The Return On Investment (ROI), or net return.
- **opened** (*datetime*) – Date the position is opened.
- **held** (*int*) – The holding period since the position was opened.
- **costbasis** (*float*) – Overall cost basis.
- **trades** (*list of Trade*) – The executed trades for the position so far.
- **ntrades** (*int*) – Total number of trades.
- **pdata** (*pandas DataFrame*) – Price data for the given name.
- **multiplier** (*float*) – Multiple for instrument type (e.g., 1.0 for stocks).

class alphapy.portfolio.**Trade**(*name*, *order*, *quantity*, *price*, *tdate*)

Bases: object

Initiate a trade.

Parameters

- **name** (*str*) – The symbol to trade.
- **order** (*alphapy.Orders*) – Long or short trade for entry or exit.
- **quantity** (*int*) – The quantity for the order.
- **price** (*str*) – The execution price of the trade.
- **tdate** (*datetime*) – The date and time of the trade.

Variables ***states*** (*list of str*) – Trade state names for a dataframe.

```
states = ['name', 'order', 'quantity', 'price']
```

alphapy.portfolio.**add_position**(*p*, *name*, *pos*)

Add a position to a portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio that will hold the position.
- **name** (*int*) – Unique identifier for the position, e.g., a stock symbol.
- **pos** (*alphapy.Position*) – New position to add to the portfolio.

Returns **p** – Portfolio with the new position.

Return type alphapy.Portfolio

alphapy.portfolio.**allocate_trade**(*p*, *pos*, *trade*)

Determine the trade allocation for a given portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio that will hold the new position.
- **pos** (*alphapy.Position*) – Position to update.
- **trade** (*alphapy.Trade*) – The proposed trade.

Returns **allocation** – The trade size that can be allocated for the portfolio.

Return type float

alphapy.portfolio.**balance**(*p*, *tdate*, *cashlevel*)

Balance the portfolio using a weighting variable.

Rebalancing is the process of equalizing a portfolio's positions using some criterion. For example, if a portfolio is *dollar-weighted*, then one position can increase in proportion to the rest of the portfolio, i.e., its fraction of the overall portfolio is greater than the other positions. To make the portfolio "equal dollar", then some positions have to be decreased and others decreased.

The rebalancing process is periodic (e.g., once per month) and generates a series of trades to balance the positions. Other portfolios are *volatility-weighted* because a more volatile stock has a greater effect on the beta, i.e., the more volatile the instrument, the smaller the position size.

Technically, any type of weight can be used for rebalancing, so AlphaPy gives the user the ability to specify a `weightby` column name.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio to rebalance.
- **tdate** (*datetime*) – The rebalancing date.
- **cashlevel** (*float*) – The cash level to maintain during rebalancing.

Returns **p** – The rebalanced portfolio.

Return type alphapy.Portfolio

Notes

Warning: The portfolio management functions `balance`, `kick_out`, and `stop_loss` are not part of the main **StockStream** pipeline, and thus have not been thoroughly tested. Feel free to exercise the code and report any issues.

`alphapy.portfolio.`**`close_position`**(*p*, *position*, *tdate*)
Close the position and remove it from the portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio holding the position.
- **position** (*alphapy.Position*) – Position to close.
- **tdate** (*datetime*) – The date for pricing the closed position.

Returns **p** – Portfolio with the removed position.

Return type alphapy.Portfolio

`alphapy.portfolio.`**`delete_portfolio`**(*p*)
Delete the portfolio.

Parameters **p** (*alphapy.Portfolio*) – Portfolio to delete.

Returns **None**

Return type None

`alphapy.portfolio.`**`deposit_portfolio`**(*p*, *cash*, *tdate*)
Deposit cash into a given portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio to accept the deposit.
- **cash** (*float*) – Cash amount to deposit.
- **tdate** (*datetime*) – The date of deposit.

Returns **p** – Portfolio with the added cash.

Return type alphapy.Portfolio

`alphapy.portfolio.`**`exec_trade`**(*p*, *name*, *order*, *quantity*, *price*, *tdate*)
Execute a trade for a portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio in which to trade.
- **name** (*str*) – The symbol to trade.
- **order** (*alphapy.Orders*) – Long or short trade for entry or exit.
- **quantity** (*int*) – The quantity for the order.
- **price** (*str*) – The execution price of the trade.
- **tdate** (*datetime*) – The date and time of the trade.

Returns **tsize** – The executed trade size.

Return type float

Other Parameters **Frame.frames** (*dict*) – Dataframe for the price data.

alphapy.portfolio.**gen_portfolio**(*model*, *system*, *group*, *tframe*, *startcap=100000*, *posby='close'*)

Create a portfolio from a trades frame.

Parameters

- **model** (*alphapy.Model*) – The model with specifications.
- **system** (*str*) – Name of the system.
- **group** (*alphapy.Group*) – The group of instruments in the portfolio.
- **tframe** (*pandas.DataFrame*) – The input trade list from running the system.
- **startcap** (*float*) – Starting capital.
- **posby** (*str*) – The position sizing column in the price dataframe.

Returns **p** – The generated portfolio.

Return type alphapy.Portfolio

Raises **MemoryError** – Could not allocate Portfolio.

Notes

This function also generates the files required for analysis by the *pyfolio* package:

- Returns File
- Positions File
- Transactions File

alphapy.portfolio.**kick_out**(*p*, *tdate*)

Trim the portfolio based on filter criteria.

To reduce a portfolio's positions, AlphaPy can rank the positions on some criterion, such as open profit or net return. On a periodic basis, the worst performers can be culled from the portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – The portfolio for reducing positions.
- **tdate** (*datetime*) – The date to trim the portfolio positions.

Returns **p** – The reduced portfolio.

Return type alphapy.Portfolio

Notes

Warning: The portfolio management functions `kick_out`, `balance`, and `stop_loss` are not part of the main **StockStream** pipeline, and thus have not been thoroughly tested. Feel free to exercise the code and report any issues.

alphapy.portfolio.**portfolio_name**(*group_name*, *tag*)

Return the name of the portfolio.

Parameters

- **group_name** (*str*) – The group represented in the portfolio.
- **tag** (*str*) – A unique identifier.

Returns **port_name** – Portfolio name.

Return type str

`alphapy.portfolio.`**`remove_position`**(*p*, *name*)
Remove a position from a portfolio by name.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio with the current position.
- **name** (*int*) – Unique identifier for the position, e.g., a stock symbol.

Returns **p** – Portfolio with the deleted position.

Return type alphapy.Portfolio

`alphapy.portfolio.`**`stop_loss`**(*p*, *tdate*)
Trim the portfolio based on stop-loss criteria.

Parameters

- **p** (*alphapy.Portfolio*) – The portfolio for reducing positions based on `maxloss`.
- **tdate** (*datetime*) – The date to trim any underperforming positions.

Returns **p** – The reduced portfolio.

Return type alphapy.Portfolio

Notes

Warning: The portfolio management functions `stop_loss`, `balance`, and `kick_out` are not part of the main **StockStream** pipeline, and thus have not been thoroughly tested. Feel free to exercise the code and report any issues.

`alphapy.portfolio.`**`update_portfolio`**(*p*, *pos*, *trade*)
Update the portfolio positions.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio holding the position.
- **pos** (*alphapy.Position*) – Position to update.
- **trade** (*alphapy.Trade*) – Trade for updating the position and portfolio.

Returns **p** – Portfolio with the revised position.

Return type alphapy.Portfolio

`alphapy.portfolio.`**`update_position`**(*position*, *trade*)
Add the new trade to the position and revalue.

Parameters

- **position** (*alphapy.Position*) – The position to be update.
- **trade** (*alphapy.Trade*) – Trade for updating the position.

Returns **position** – New value of the position.

Return type alphapy.Position

alphapy.portfolio.**valuate_portfolio**(*p*, *tdate*)
Value the portfolio based on the current positions.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio for calculating profit and return.
- **tdate** (*datetime*) – The date of valuation.

Returns **p** – Portfolio with the new valuation.

Return type alphapy.Portfolio

alphapy.portfolio.**valuate_position**(*position*, *tdate*)
Valuate the position for the given date.

Parameters

- **position** (*alphapy.Position*) – The position to be valued.
- **tdate** (*timedate*) – Date to value the position.

Returns **position** – New value of the position.

Return type alphapy.Position

Notes

An Example of Cost Basis

Date	Shares	Price	Amount
11/09/16	+100	10.0	1,000
12/14/16	+200	15.0	3,000
04/05/17	-500	20.0	10,000
All	800		14,000

The cost basis is calculated as the total value of all trades (14,000) divided by the total number of shares traded (800), so 14,000 / 800 = 17.5, and the net position is -200.

alphapy.portfolio.**withdraw_portfolio**(*p*, *cash*, *tdate*)
Withdraw cash from a given portfolio.

Parameters

- **p** (*alphapy.Portfolio*) – Portfolio to accept the withdrawal.
- **cash** (*float*) – Cash amount to withdraw.
- **tdate** (*datetime*) – The date of withdrawal.

Returns **p** – Portfolio with the withdrawn cash.

Return type alphapy.Portfolio

14.1.17 alphapy.space module

class alphapy.space.**Space**(*subject='stock'*, *schema='prices'*, *fractal='1d'*)
Bases: object

Create a new namespace.

Parameters

- **subject** (*str*) – An identifier for a group of related items.
- **schema** (*str*) – The data related to the subject.
- **fractal** (*str*) – The time fractal of the data, e.g., "5m" or "1d".

alphapy.space.**space_name**(*subject*, *schema*, *fractal*)
Get the namespace string.

Parameters

- **subject** (*str*) – An identifier for a group of related items.
- **schema** (*str*) – The data related to the subject.
- **fractal** (*str*) – The time fractal of the data, e.g., "5m" or "1d".

Returns **name** – The joined namespace string.

Return type str

14.1.18 alphapy.sport_flow module

alphapy.sport_flow.**add_features**(*frame*, *fdict*, *flen*, *prefix=''*)
Add new features to a dataframe with the specified dictionary.

Parameters

- **frame** (*pandas.DataFrame*) – The dataframe to extend with new features defined by fdict.
- **fdict** (*dict*) – A dictionary of column names (key) and data types (value).
- **flen** (*int*) – Length of frame.
- **prefix** (*str, optional*) – Prepend all columns with a prefix.

Returns **frame** – The dataframe with the added features.

Return type pandas.DataFrame

alphapy.sport_flow.**generate_delta_data**(*frame*, *fdict*, *prefix1*, *prefix2*)
Subtract two similar columns to get the delta value.

Parameters

- **frame** (*pandas.DataFrame*) – The input model frame.
- **fdict** (*dict*) – A dictionary of column names (key) and data types (value).
- **prefix1** (*str*) – The prefix of the first team.
- **prefix2** (*str*) – The prefix of the second team.

Returns **frame** – The completed dataframe with the delta data.

Return type pandas.DataFrame

alphapy.sport_flow.**generate_team_frame**(*team*, *tf*, *home_team*, *away_team*, *window*)
Calculate statistics for each team.

Parameters

- **team** (*str*) – The abbreviation for the team.
- **tf** (*pandas.DataFrame*) – The initial team frame.
- **home_team** (*str*) – Label for the home team.
- **away_team** (*str*) – Label for the away team.
- **window** (*int*) – The value for the rolling window to calculate means and sums.

Returns **tf** – The completed team frame.

Return type pandas.DataFrame

alphapy.sport_flow.**get_day_offset**(*date_vector*)
Compute the day offsets between games.

Parameters **date_vector** (*pandas.Series*) – The date column.

Returns **day_offset** – A vector of day offsets between adjacent dates.

Return type pandas.Series

alphapy.sport_flow.**get_losses**(*point_margin*)
Determine a loss based on the point margin.

Parameters **point_margin** (*int*) – The point margin can be positive, zero, or negative.

Returns **lost** – If the point margin is less than 0, return 1, else 0.

Return type int

alphapy.sport_flow.**get_point_margin**(*row*, *score*, *opponent_score*)
Get the point margin for a game.

Parameters

- **row** (*pandas.Series*) – The row of a game.
- **score** (*int*) – The score for one team.
- **opponent_score** (*int*) – The score for the other team.

Returns **point_margin** – The resulting point margin (0 if NaN).

Return type int

alphapy.sport_flow.**get_series_diff**(*series*)
Perform the difference operation on a series.

Parameters **series** (*pandas.Series*) – The series for the `diff` operation.

Returns **new_series** – The differenced series.

Return type pandas.Series

alphapy.sport_flow.**get_sport_config**()
Read the configuration file for SportFlow.

Parameters **None** (*None*)

Returns **specs** – The parameters for controlling SportFlow.

Return type dict

alphapy.sport_flow.**get_streak**(*series*, *start_index*, *window*)
Calculate the current streak.

Parameters

- **series** (*pandas.Series*) – A Boolean series for calculating streaks.
- **start_index** (*int*) – The offset of the series to start counting.
- **window** (*int*) – The period over which to count.

Returns **streak** – The count value for the current streak.

Return type int

alphapy.sport_flow.**get_team_frame**(*game_frame*, *team*, *home*, *away*)
Calculate statistics for each team.

Parameters

- **game_frame** (*pandas.DataFrame*) – The game frame for a given season.
- **team** (*str*) – The team abbreviation.
- **home** (*int*) – The label of the home team column.
- **away** (*int*) – The label of the away team column.

Returns **team_frame** – The extracted team frame.

Return type pandas.DataFrame

alphapy.sport_flow.**get_ties**(*point_margin*)
Determine a tie based on the point margin.

Parameters **point_margin** (*int*) – The point margin can be positive, zero, or negative.

Returns **tied** – If the point margin is equal to 0, return 1, else 0.

Return type int

alphapy.sport_flow.**get_wins**(*point_margin*)
Determine a win based on the point margin.

Parameters **point_margin** (*int*) – The point margin can be positive, zero, or negative.

Returns **won** – If the point margin is greater than 0, return 1, else 0.

Return type int

alphapy.sport_flow.**insert_model_data**(*mf*, *mpos*, *mdict*, *tf*, *tpos*, *prefix*)
Insert a row from the team frame into the model frame.

Parameters

- **mf** (*pandas.DataFrame*) – The model frame for a single season.
- **mpos** (*int*) – The position in the model frame where to insert the row.
- **mdict** (*dict*) – A dictionary of column names (key) and data types (value).
- **tf** (*pandas.DataFrame*) – The team frame for a season.
- **tpos** (*int*) – The position of the row in the team frame.
- **prefix** (*str*) – The prefix to join with the `mdict` key.

Returns **mf** – The .

Return type pandas.DataFrame

alphapy.sport_flow.**main**(*args=None*)
The main program for SportFlow.

Notes

(1) Initialize logging.

(2) Parse the command line arguments.

(3) Get the game configuration.

(4) Get the model configuration.

(5) Generate game frames for each season.

(6) Create statistics for each team.

(7) Merge the team frames into the final model frame.

(8) Run the AlphaPy pipeline.

Raises **ValueError** – Training date must be before prediction date.

14.1.19 alphapy.system module

class alphapy.system.**System**(*name*, *longentry*, *shortentry=None*, *longexit=None*, *shortexit=None*, *holdperiod=0*, *scale=False*)
Bases: object

Create a new system. All systems are stored in System.systems. Duplicate names are not allowed.

Parameters

- **name** (*str*) – The system name.
- **longentry** (*str*) – Name of the conditional feature for a long entry.
- **shortentry** (*str, optional*) – Name of the conditional feature for a short entry.
- **longexit** (*str, optional*) – Name of the conditional feature for a long exit.
- **shortexit** (*str, optional*) – Name of the conditional feature for a short exit.
- **holdperiod** (*int, optional*) – Holding period of a position.
- **scale** (*bool, optional*) – Add to a position for a signal in the same direction.

Variables ***systems*** (*dict*) – Class variable for storing all known systems

Examples

```
>>> System('closer', hc, lc)
```

systems = {}

alphapy.system.**run_system**(*model*, *system*, *group*, *intraday=False*, *quantity=1*)
Run a system for a given group, creating a trades frame.

Parameters

- **model** (*alphapy.Model*) – The model object with specifications.

- **system** (*alphapy.System*) – The system to run.
- **group** (*alphapy.Group*) – The group of symbols to trade.
- **intraday** (*bool, optional*) – If true, this is an intraday system.
- **quantity** (*float, optional*) – The amount to trade for each symbol, e.g., number of shares

Returns **tf** – All of the trades for this `group`.

Return type pandas.DataFrame

`alphapy.system.`**`trade_system`**(*model*, *system*, *space*, *intraday*, *name*, *quantity*)

Trade the given system.

Parameters

- **model** (*alphapy.Model*) – The model object with specifications.
- **system** (*alphapy.System*) – The long/short system to run.
- **space** (*alphapy.Space*) – Namespace of instrument prices.
- **intraday** (*bool*) – If True, then run an intraday system.
- **name** (*str*) – The symbol to trade.
- **quantity** (*float*) – The amount of the `name` to trade, e.g., number of shares

Returns **tradelist** – List of trade entries and exits.

Return type list

Other Parameters **Frame.frames** (*dict*) – All of the data frames containing price data.

14.1.20 alphapy.transforms module

`alphapy.transforms.`**`abovema`**(*f*, *c*, *p=50*)

Determine those values of the dataframe that are above the moving average.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period of the moving average.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

`alphapy.transforms.`**`adx`**(*f*, *p=14*)

Calculate the Average Directional Index (ADX).

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with all columns required for calculation. If you are applying ADX through `vapply`, then these columns are calculated automatically.
- **p** (*int*) – The period over which to calculate the ADX.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

The Average Directional Movement Index (ADX) was invented by J. Welles Wilder in 1978 [WIKI_ADX]. Its value reflects the strength of trend in any given instrument.

alphapy.transforms.**belowma**(*f*, *c*, *p=50*)
Determine those values of the dataframe that are below the moving average.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period of the moving average.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

alphapy.transforms.**c2max**(*f*, *c1*, *c2*)
Take the maximum value between two columns in a dataframe.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the two columns `c1` and `c2`.
- **c1** (*str*) – Name of the first column in the dataframe `f`.
- **c2** (*str*) – Name of the second column in the dataframe `f`.

Returns **max_val** – The maximum value of the two columns.

Return type float

alphapy.transforms.**c2min**(*f*, *c1*, *c2*)
Take the minimum value between two columns in a dataframe.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the two columns `c1` and `c2`.
- **c1** (*str*) – Name of the first column in the dataframe `f`.
- **c2** (*str*) – Name of the second column in the dataframe `f`.

Returns **min_val** – The minimum value of the two columns.

Return type float

alphapy.transforms.**diff**(*f*, *c*, *n=1*)
Calculate the n-th order difference for the given variable.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **n** (*int*) – The number of times that the values are differenced.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**diminus**(*f*, *p=14*)
Calculate the Minus Directional Indicator (-DI).

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.
- **p** (*int*) – The period over which to calculate the -DI.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

A component of the average directional index (ADX) that is used to measure the presence of a downtrend. When the -DI is sloping downward, it is a signal that the downtrend is getting stronger [IP_NDI].

`alphapy.transforms.`**`diplus`**(*f*, *p=14*)
Calculate the Plus Directional Indicator (+DI).

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.
- **p** (*int*) – The period over which to calculate the +DI.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

A component of the average directional index (ADX) that is used to measure the presence of an uptrend. When the +DI is sloping upward, it is a signal that the uptrend is getting stronger [IP_PDI].

`alphapy.transforms.`**`dminus`**(*f*)
Calculate the Minus Directional Movement (-DM).

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Directional movement is negative (minus) when the prior low minus the current low is greater than the current high minus the prior high. This so-called Minus Directional Movement (-DM) equals the prior low minus the current low, provided it is positive. A negative value would simply be entered as zero [SC_ADX].

`alphapy.transforms.`**`dmplus`**(*f*)
Calculate the Plus Directional Movement (+DM).

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Directional movement is positive (plus) when the current high minus the prior high is greater than the prior low minus the current low. This so-called Plus Directional Movement (+DM) then equals the current high minus the prior high, provided it is positive. A negative value would simply be entered as zero [SC_ADX].

`alphapy.transforms.`**`down`**(*f*, *c*)

Find the negative values in the series.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

`alphapy.transforms.`**`dpc`**(*f*, *c*)

Get the negative values, with positive values zeroed.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with column `c`.
- **c** (*str*) – Name of the column.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

`alphapy.transforms.`**`ema`**(*f*, *c*, *p=20*)

Calculate the mean on a rolling basis.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period over which to calculate the rolling mean.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

An exponential moving average (EMA) is a type of moving average that is similar to a simple moving average, except that more weight is given to the latest data [IP_EMA].

`alphapy.transforms.`**`extract_bizday`**(*f*, *c*)

Extract business day of month and week.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the date column `c`.
- **c** (*str*) – Name of the date column in the dataframe `f`.

Returns **date_features** – The dataframe containing the date features.

Return type pandas.DataFrame

alphapy.transforms.**extract_date**(*f*, *c*)
Extract date into its components: year, month, day, dayofweek.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the date column `c`.
- **c** (*str*) – Name of the date column in the dataframe `f`.

Returns **date_features** – The dataframe containing the date features.

Return type pandas.DataFrame

alphapy.transforms.**extract_time**(*f*, *c*)
Extract time into its components: hour, minute, second.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the time column `c`.
- **c** (*str*) – Name of the time column in the dataframe `f`.

Returns **time_features** – The dataframe containing the time features.

Return type pandas.DataFrame

alphapy.transforms.**gap**(*f*)
Calculate the gap percentage between the current open and the previous close.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `open` and `close`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

A gap is a break between prices on a chart that occurs when the price of a stock makes a sharp move up or down with no trading occurring in between [IP_GAP].

alphapy.transforms.**gapbadown**(*f*)
Determine whether or not there has been a breakaway gap down.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `open` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

A breakaway gap represents a gap in the movement of a stock price supported by levels of high volume [IP_BAGAP].

alphapy.transforms.**gapbaup**(*f*)
Determine whether or not there has been a breakaway gap up.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `open` and `high`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

A breakaway gap represents a gap in the movement of a stock price supported by levels of high volume [IP_BAGAP].

`alphapy.transforms.`**`gapdown`**(*f*)

Determine whether or not there has been a gap down.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `open` and `close`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

A gap is a break between prices on a chart that occurs when the price of a stock makes a sharp move up or down with no trading occurring in between [IP_GAP].

`alphapy.transforms.`**`gapup`**(*f*)

Determine whether or not there has been a gap up.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `open` and `close`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

A gap is a break between prices on a chart that occurs when the price of a stock makes a sharp move up or down with no trading occurring in between [IP_GAP].

`alphapy.transforms.`**`gtval`**(*f*, *c1*, *c2*)

Determine whether or not the first column of a dataframe is greater than the second.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the two columns `c1` and `c2`.
- **c1** (*str*) – Name of the first column in the dataframe `f`.
- **c2** (*str*) – Name of the second column in the dataframe `f`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

`alphapy.transforms.`**`gtval0`**(*f*, *c1*, *c2*)

For positive values in the first column of the dataframe that are greater than the second column, get the value in the first column, otherwise return zero.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the two columns `c1` and `c2`.
- **c1** (*str*) – Name of the first column in the dataframe `f`.
- **c2** (*str*) – Name of the second column in the dataframe `f`.

Returns **new_val** – A positive value or zero.

Return type float

alphapy.transforms.**higher**(*f*, *c*, *o=1*)
Determine whether or not a series value is higher than the value `o` periods back.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **o** (*int, optional*) – Offset value for shifting the series.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

alphapy.transforms.**highest**(*f*, *c*, *p=20*)
Calculate the highest value on a rolling basis.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period over which to calculate the rolling maximum.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**hlrange**(*f*, *p=1*)
Calculate the Range, the difference between High and Low.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.
- **p** (*int*) – The period over which the range is calculated.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**lower**(*f*, *c*, *o=1*)
Determine whether or not a series value is lower than the value `o` periods back.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **o** (*int, optional*) – Offset value for shifting the series.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

alphapy.transforms.**lowest**(*f*, *c*, *p=20*)
Calculate the lowest value on a rolling basis.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period over which to calculate the rolling minimum.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**ma**(*f*, *c*, *p=20*)
Calculate the mean on a rolling basis.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period over which to calculate the rolling mean.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

In statistics, a moving average (rolling average or running average) is a calculation to analyze data points by creating series of averages of different subsets of the full data set [WIKI_MA].

alphapy.transforms.**maratio**(*f*, *c*, *p1=1*, *p2=10*)
Calculate the ratio of two moving averages.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p1** (*int*) – The period of the first moving average.
- **p2** (*int*) – The period of the second moving average.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**mval**(*f*, *c*)
Get the negative value, otherwise zero.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.

Returns **new_val** – Negative value or zero.

Return type float

alphapy.transforms.**net**(*f*, *c='close'*, *o=1*)
Calculate the net change of a given column.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **o** (*int, optional*) – Offset value for shifting the series.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Net change is the difference between the closing price of a security on the day's trading and the previous day's closing price. Net change can be positive or negative and is quoted in terms of dollars [IP_NET].

`alphapy.transforms.`**`netreturn`**(*f*, *c*, *o=1*)

Calculate the net return, or Return On Invesment (ROI)

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **o** (*int, optional*) – Offset value for shifting the series.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

ROI measures the amount of return on an investment relative to the original cost. To calculate ROI, the benefit (or return) of an investment is divided by the cost of the investment, and the result is expressed as a percentage or a ratio [IP_ROI].

`alphapy.transforms.`**`pchange1`**(*f*, *c*, *o=1*)

Calculate the percentage change within the same variable.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **o** (*int*) – Offset to the previous value.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

`alphapy.transforms.`**`pchange2`**(*f*, *c1*, *c2*)

Calculate the percentage change between two variables.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the two columns `c1` and `c2`.
- **c1** (*str*) – Name of the first column in the dataframe `f`.
- **c2** (*str*) – Name of the second column in the dataframe `f`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

`alphapy.transforms.`**`pval`**(*f*, *c*)

Get the positive value, otherwise zero.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.

Returns **new_val** – Positive value or zero.

Return type float

alphapy.transforms.**rindex**(*f*, *ci*, *ch*, *cl*, *p=1*)
Calculate the *range index* spanning a given period `p`.

The **range index** is a number between 0 and 100 that relates the value of the index column `ci` to the high column `ch` and the low column `cl`. For example, if the low value of the range is 10 and the high value is 20, then the range index for a value of 15 would be 50%. The range index for 18 would be 80%.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the columns `ci`, `ch`, and `cl`.
- **ci** (*str*) – Name of the index column in the dataframe `f`.
- **ch** (*str*) – Name of the high column in the dataframe `f`.
- **cl** (*str*) – Name of the low column in the dataframe `f`.
- **p** (*int*) – The period over which the range index of column `ci` is calculated.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**rsi**(*f*, *c*, *p=14*)
Calculate the Relative Strength Index (RSI).

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `net`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **p** (*int*) – The period over which to calculate the RSI.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Developed by J. Welles Wilder, the Relative Strength Index (RSI) is a momentum oscillator that measures the speed and change of price movements [SC_RSI].

alphapy.transforms.**rtotal**(*vec*)
Calculate the running total.

Parameters **vec** (*pandas.Series*) – The input array for calculating the running total.

Returns **running_total** – The final running total.

Return type int

Example

```
>>> vec.rolling(window=20).apply(rtotal)
```

alphapy.transforms.**runs**(*vec*)
Calculate the total number of runs.

Parameters **vec** (*pandas.Series*) – The input array for calculating the number of runs.

Returns **runs_value** – The total number of runs.

Return type int

Example

```
>>> vec.rolling(window=20).apply(runs)
```

alphapy.transforms.**runs_test**(*f*, *c*, *wfuncs*, *window*)
Perform a runs test on binary series.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.
- **wfuncs** (*list*) – The set of runs test functions to apply to the column:

  `'all'`: Run all of the functions below.

  `'rtotal'`: The running total over the `window` period.

  `'runs'`: Total number of runs in `window`.

  `'streak'`: The length of the latest streak.

  `'zscore'`: The Z-Score over the `window` period.
- **window** (*int*) – The rolling period.

Returns **new_features** – The dataframe containing the runs test features.

Return type pandas.DataFrame

References

For more information about runs tests for detecting non-randomness, refer to [RUNS].

alphapy.transforms.**split_to_letters**(*f*, *c*)
Separate text into distinct characters.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the text column in the dataframe `f`.

Returns **new_feature** – The array containing the new feature.

Return type pandas.Series

Example

The value 'abc' becomes 'a b c'.

alphapy.transforms.**streak**(*vec*)
Determine the length of the latest streak.

Parameters **vec** (*pandas.Series*) – The input array for calculating the latest streak.

Returns **latest_streak** – The length of the latest streak.

Return type int

Example

```
>>> vec.rolling(window=20).apply(streak)
```

alphapy.transforms.**texplode**(*f*, *c*)
Get dummy values for a text column.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the text column in the dataframe `f`.

Returns **dummies** – The dataframe containing the dummy variables.

Return type pandas.DataFrame

Example

This function is useful for columns that appear to have separate character codes but are consolidated into a single column. Here, the column `c` is transformed into five dummy variables.

c	0_a	1_x	1_b	2_x	2_z
abz	1	0	1	0	1
abz	1	0	1	0	1
axx	1	1	0	1	0
abz	1	0	1	0	1
axz	1	1	0	0	1

alphapy.transforms.**truehigh**(*f*)
Calculate the *True High* value.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Today's high, or the previous close, whichever is higher [TS_TR].

alphapy.transforms.**truelow**(*f*)
Calculate the *True Low* value.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

Today's low, or the previous close, whichever is lower [TS_TR].

alphapy.transforms.**truerange**(*f*)
Calculate the *True Range* value.

Parameters **f** (*pandas.DataFrame*) – Dataframe with columns `high` and `low`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

References

True High - True Low [TS_TR].

alphapy.transforms.**up**(*f*, *c*)
Find the positive values in the series.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str*) – Name of the column in the dataframe `f`.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

alphapy.transforms.**upc**(*f*, *c*)
Get the positive values, with negative values zeroed.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe with column `c`.
- **c** (*str*) – Name of the column.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (float)

alphapy.transforms.**xmadown**(*f*, *c='close'*, *pfast=20*, *pslow=50*)
Determine those values of the dataframe that are below the moving average.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str, optional*) – Name of the column in the dataframe `f`.

- **pfast** (*int, optional*) – The period of the fast moving average.
- **pslow** (*int, optional*) – The period of the slow moving average.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

In the statistics of time series, and in particular the analysis of financial time series for stock trading purposes, a moving-average crossover occurs when, on plotting two moving averages each based on different degrees of smoothing, the traces of these moving averages cross [WIKI_XMA].

`alphapy.transforms.`**`xmaup`**(*f*, *c='close'*, *pfast=20*, *pslow=50*)

Determine those values of the dataframe that are below the moving average.

Parameters

- **f** (*pandas.DataFrame*) – Dataframe containing the column `c`.
- **c** (*str, optional*) – Name of the column in the dataframe `f`.
- **pfast** (*int, optional*) – The period of the fast moving average.
- **pslow** (*int, optional*) – The period of the slow moving average.

Returns **new_column** – The array containing the new feature.

Return type pandas.Series (bool)

References

In the statistics of time series, and in particular the analysis of financial time series for stock trading purposes, a moving-average crossover occurs when, on plotting two moving averages each based on different degrees of smoothing, the traces of these moving averages cross [WIKI_XMA].

`alphapy.transforms.`**`zscore`**(*vec*)

Calculate the Z-Score.

Parameters **vec** (*pandas.Series*) – The input array for calculating the Z-Score.

Returns **zscore** – The value of the Z-Score.

Return type float

References

To calculate the Z-Score, you can find more information here [ZSCORE].

Example

```
>>> vec.rolling(window=20).apply(zscore)
```

14.1.21 alphapy.utilities module

alphapy.utilities.**get_datestamp**()
Returns today's datestamp.

Returns **datestamp** – The valid date string in YYYY-mm-dd format.

Return type str

alphapy.utilities.**most_recent_file**(*directory*, *file_spec*)
Find the most recent file in a directory.

Parameters

- **directory** (*str*) – Full directory specification.
- **file_spec** (*str*) – Wildcard search string for the file to locate.

Returns **file_name** – Name of the file to read, excluding the `extension`.

Return type str

alphapy.utilities.**np_store_data**(*data*, *dir_name*, *file_name*, *extension*, *separator*)
Store NumPy data in a file.

Parameters

- **data** (*numpy array*) – The model component to store
- **dir_name** (*str*) – Full directory specification.
- **file_name** (*str*) – Name of the file to read, excluding the `extension`.
- **extension** (*str*) – File name extension, e.g., `csv`.
- **separator** (*str*) – The delimiter between fields in the file.

Returns **None**

Return type None

alphapy.utilities.**remove_list_items**(*elements*, *alist*)
Remove one or more items from the given list.

Parameters

- **elements** (*list*) – The items to remove from the list `alist`.
- **alist** (*list*) – Any object of any type can be a list item.

Returns **sublist** – The subset of items after removal.

Return type list

Examples

```
>>> test_list = ['a', 'b', 'c', test_func]
>>> remove_list_items([test_func], test_list)  # ['a', 'b', 'c']
```

alphapy.utilities.**subtract_days**(*date_string*, *ndays*)

Subtract a number of days from a given date.

Parameters

- **date_string** (*str*) – An alphanumeric string in the format %Y-%m-%d.
- **ndays** (*int*) – Number of days to subtract.

Returns **new_date_string** – The adjusted date string in the format %Y-%m-%d.

Return type str

Examples

```
>>> subtract_days('2017-11-10', 31)   # '2017-10-10'
```

alphapy.utilities.**valid_date**(*date_string*)

Determine whether or not the given string is a valid date.

Parameters **date_string** (*str*) – An alphanumeric string in the format %Y-%m-%d.

Returns **date_string** – The valid date string.

Return type str

Raises **ValueError** – Not a valid date.

Examples

```
>>> valid_date('2016-7-1')   # datetime.datetime(2016, 7, 1, 0, 0)
>>> valid_date('345')        # ValueError: Not a valid date
```

alphapy.utilities.**valid_name**(*name*)

Determine whether or not the given string is a valid alphanumeric string.

Parameters **name** (*str*) – An alphanumeric identifier.

Returns **result** – `True` if the name is valid, else `False`.

Return type bool

Examples

```
>>> valid_name('alpha')   # True
>>> valid_name('!alpha')  # False
```

14.1.22 alphapy.variables module

class alphapy.variables.**Variable**(*name*, *expr*, *replace=False*)
Bases: object

Create a new variable as a key-value pair. All variables are stored in `Variable.variables`. Duplicate keys or values are not allowed, unless the `replace` parameter is `True`.

Parameters

- **name** (*str*) – Variable key.
- **expr** (*str*) – Variable value.
- **replace** (*bool, optional*) – Replace the current key-value pair if it already exists.

Variables ***variables*** (*dict*) – Class variable for storing all known variables

Examples

```
>>> Variable('rrunder', 'rr_3_20 <= 0.9')
>>> Variable('hc', 'higher_close')
```

variables = {}

alphapy.variables.**allvars**(*expr*)
Get the list of valid names in the expression.

Parameters **expr** (*str*) – A valid expression conforming to the Variable Definition Language.

Returns **vlist** – List of valid variable names.

Return type list

alphapy.variables.**vapply**(*group*, *vname*, *vfuncs=None*)
Apply a variable to multiple dataframes.

Parameters

- **group** (*alphapy.Group*) – The input group.
- **vname** (*str*) – The variable to apply to the `group`.
- **vfuncs** (*dict, optional*) – Dictionary of external modules and functions.

Returns **None**

Return type None

Other Parameters **Frame.frames** (*dict*) – Global dictionary of dataframes

See also:

vunapply()

alphapy.variables.**vexec**(*f*, *v*, *vfuncs=None*)
Add a variable to the given dataframe.

This is the core function for adding a variable to a dataframe. The default variable functions are already defined locally in `alphapy.transforms`; however, you may want to define your own variable functions. If so, then the `vfuncs` parameter will contain the list of modules and functions to be imported and applied by the `vexec` function.

To write your own variable function, your function must have a pandas *DataFrame* as an input parameter and must return a pandas *DataFrame* with the new variable(s).

Parameters

- **f** (*pandas.DataFrame*) – Dataframe to contain the new variable.
- **v** (*str*) – Variable to add to the dataframe.
- **vfuncs** (*dict, optional*) – Dictionary of external modules and functions.

Returns **f** – Dataframe with the new variable.

Return type pandas.DataFrame

Other Parameters **Variable.variables** (*dict*) – Global dictionary of variables

`alphapy.variables.`**`vmapply`**(*group*, *vs*, *vfuncs=None*)

Apply multiple variables to multiple dataframes.

Parameters

- **group** (*alphapy.Group*) – The input group.
- **vs** (*list*) – The list of variables to apply to the `group`.
- **vfuncs** (*dict, optional*) – Dictionary of external modules and functions.

Returns **None**

Return type None

See also:

vmunapply()

`alphapy.variables.`**`vmunapply`**(*group*, *vs*)

Remove a list of variables from multiple dataframes.

Parameters

- **group** (*alphapy.Group*) – The input group.
- **vs** (*list*) – The list of variables to remove from the `group`.

Returns **None**

Return type None

See also:

vmapply()

`alphapy.variables.`**`vparse`**(*vname*)

Parse a variable name into its respective components.

Parameters **vname** (*str*) – The name of the variable.

Returns

- **vxlag** (*str*) – Variable name without the `lag` component.
- **root** (*str*) – The base variable name without the parameters.
- **plist** (*list*) – The parameter list.
- **lag** (*int*) – The offset starting with the current value [0] and counting back, e.g., an offset [1] means the previous value of the variable.

Notes

AlphaPy makes feature creation easy. The syntax of a variable name maps to a function call:

xma_20_50 => xma(20, 50)

Examples

```
>>> vparse('xma_20_50[1]')
# ('xma_20_50', 'xma', ['20', '50'], 1)
```

alphapy.variables.**vsub**(*v*, *expr*)

Substitute the variable parameters into the expression.

This function performs the parameter substitution when applying features to a dataframe. It is a mechanism for the user to override the default values in any given expression when defining a feature, instead of having to programmatically call a function with new values.

Parameters

- **v** (*str*) – Variable name.
- **expr** (*str*) – The expression for substitution.

Returns The expression with the new, substituted values.

Return type newexpr

alphapy.variables.**vtree**(*vname*)

Get all of the antecedent variables.

Before applying a variable to a dataframe, we have to recursively get all of the child variables, beginning with the starting variable's expression. Then, we have to extract the variables from all the subsequent expressions. This process continues until all antecedent variables are obtained.

Parameters **vname** (*str*) – A valid variable stored in `Variable.variables`.

Returns **all_variables** – The variables that need to be applied before `vname`.

Return type list

Other Parameters **Variable.variables** (*dict*) – Global dictionary of variables

alphapy.variables.**vunapply**(*group*, *vname*)

Remove a variable from multiple dataframes.

Parameters

- **group** (*alphapy.Group*) – The input group.
- **vname** (*str*) – The variable to remove from the `group`.

Returns **None**

Return type None

Other Parameters **Frame.frames** (*dict*) – Global dictionary of dataframes

See also:

vapply()

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[CLUS] http://scikit-learn.org/stable/modules/clustering.html

[ISO] http://scikit-learn.org/stable/modules/manifold.html#isomap

[PCA] http://scikit-learn.org/stable/modules/decomposition.html#pca

[TSNE] http://scikit-learn.org/stable/modules/manifold.html#t-distributed-stochastic-neighbor-embedding-t-sne

[POLY] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

[IMP] http://scikit-learn.org/stable/modules/preprocessing.html#imputation

[LV] http://scikit-learn.org/stable/modules/feature_selection.html#variance-threshold

[UNI] http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

[ENC] https://github.com/scikit-learn-contrib/categorical-encoding

[IMB] https://github.com/scikit-learn-contrib/imbalanced-learn

[SCALE] http://scikit-learn.org/stable/modules/preprocessing.html

[EVAL] http://scikit-learn.org/stable/modules/model_evaluation.html

[GRID] http://scikit-learn.org/stable/modules/grid_search.html#grid-search

[PIPE] http://scikit-learn.org/stable/modules/pipeline.html#pipeline

[RFECV] http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html

[WIKI_ADX] https://en.wikipedia.org/wiki/Average_directional_movement_index

[IP_NDI] http://www.investopedia.com/terms/n/negativedirectionalindicator.asp

[IP_PDI] http://www.investopedia.com/terms/p/positivedirectionalindicator.asp

[SC_ADX] http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:average_directional_index_adx

[IP_EMA] http://www.investopedia.com/terms/e/ema.asp

[IP_GAP] http://www.investopedia.com/terms/g/gap.asp

[IP_BAGAP] http://www.investopedia.com/terms/b/breakawaygap.asp

[WIKI_MA] https://en.wikipedia.org/wiki/Moving_average

[IP_NET] http://www.investopedia.com/terms/n/netchange.asp

[IP_ROI] http://www.investopedia.com/terms/r/returnoninvestment.asp

[SC_RSI] http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_index_rsi

[RUNS] http://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm

[TS_TR] http://help.tradestation.com/09_01/tradestationhelp/charting_definitions/true_range.htm

[WIKI_XMA] https://en.wikipedia.org/wiki/Moving_average_crossover

[ZSCORE] https://en.wikipedia.org/wiki/Standard_score

PYTHON MODULE INDEX

a

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

W

X

Z