supplychainpy Documentation

Release 0.0.4

Kevin Fasusi

Nov 07, 2017

Contents

Contents:

CHAPTER 1

Change Log

1.1 0.0.5

1.1.1 Application

- [Bug Fix] Using Flask's web server for the Dashboard on a public route on a standalone server (–host 0.0.0.0)
- [Bug Fix] Javascript error while loading dashboard.
- [Bug Fix] Pip install error (Log.txt FileNotFound)
- [New Feature] Basic ability to run Monte Carlo Simulation and view summarised results in reporting suite.
- [Update] Load scripts use multi-processing for forecast calculations when processing data file.
- [Update] Load scripts using batch process.
- [Update] Debug commandline argument for viewing logging output '–debug'.
- [Update] Use Chat Bot from commandline with *-c* flag. EXPERIMENTAL
- [Update] Recommendation generator takes into account forecasts
- [Update] Flask Blueprints used for reporting views.

1.1.2 Documentation

- [New] Wiki started on GitHub for more responsive updates to documentation including changes to source during development.
- [Update] Tutorial.

1.2 0.0.4

Release 0.0.4 has breaking API changes. Namespaces have changed in this release. All the modules previously in the "demand" package are now inside the "inventory" package. If you have been using the "model_inventory" module, then nothing has changed, there will not be any break in contracts.

1.2.1 Application

- [Update] Explicit internal and public API.
- [Update] Excess, shortages added to the UncertainDemand order_summary.
- [Update] Moved abc_xyz.py, analyse_uncertain_demand.py, economic_order_quantity.py and eoq.pyx from "demand" to "inventory" package
- [Update] "demand" package now contains: evolutionary_algorithms.py, forecast_demand.py and regression.py
- [Update] retail_price added to *model_inventory.analyse_orders*.
- [Update] backlog added to the data format for loading into the analysis.
- [Update] Unit Tests.
- [Update] Docstrings.
- [New Feature] Analytic Hierarchy Process.
- [New Feature] API supports Pandas *DataFrame*.
- [New Feature] Browser based reporting suite, with charts, data summaries and integrated chat bot.
- [New Feature] Dash Bot, a basic chat bot assistant for the data in the reporting suite. Query data using natural language.
- [New Feature] Command line interface for processing .csv to database, launching reports and chat bot.
- [New Feature] "Model_Demand" module containing simple exponential smoothing and holts trend corrected exponential smoothing.
- [New Feature] Summarise and filter your analysis.
- [New Feature] Holts Trend Corrected Exponential Smoothing Forecast and optimised variant (evolutionary algorithm for optimised alpha and gamma)
- [New Feature] Simple Exponential Smoothing (evolutionary algorithm for optimised alpha).
- [New Feature] Evolutionary Algorithms for Smoothing Level Constants (converges on better smoothing levels using genetic algorithm)
- [New Feature] SKU and inventory profile recommendations generator.

1.2.2 Documentation

- [New] Reporting Suite Walk Through.
- [Update] Tutorial.
- [Update] Quick Guide.
- [New] Declare public API explicitly. describe and document each module and function, give an example also add to website tutorial as Jupyter notebook.
- [New] Docker for supplychainpy quick guide.

- [New] Analytic Hierarchy Process quick guide.
- [New] Inventory Modeling.
- [New] Demand Planning with Pandas.

1.3 0.0.3

1.3.1 Application

- Compiled Cython (eoq and simulation modules) for OS X, Windows and Linux.
- Removed z_value, file_type, file_path and reorder_cost parameters from simulate.run_monte_carlo.

1.3.2 Documentation

- Update Quick Guide

1.4 0.0.2

1.4.1 Application

- Added monte carlo simulation and simulation summary using Cython optimisation.
- Added orders analysis optimisation, based on results of the monte carlo simulation.
- Added simulate module to api.
- Added weighted moving average forecast.
- Added moving average forecast.
- Added mean absolute deviation.
- Updated economic order quantity using Cython optimisation.
- Updated unit tests.

1.4.2 Documentation

- Updates Quick Guide.
- Updated Tutorial.
- Updated README.md
- Added Formulas and Equations.
- Updated data.csv.

1.5 0.0.1

1.5.1 Application

- Added inventory analysis for uncertain demand. Analyse orders from .csv, .txt or from dict.
- Added inventory analysis summary for uncertain demand. ABC XYZ, economic order quantity (EOQ), reorder level (ROL), demand variability and safety stock.

1.5.2 Documentation

- Added Quick Guide.
- Added Tutorial.
- Added Installation.

1.6 Options

Currently using the reporting suite feature, requires the command line. Using a nix or PowerShell console, the typical command issued for processing a file and generating a report would be:

$ supplychainpy <filename> -a -loc <absolute-path-to-current-directory> -l

The command line arguments can be viewed by using the *–help* command in the cli.

- *-l, –launch*
  - Launches supplychainpy reporting GUI for setting port and launching the default browser. The reporting suite is hosted inside the browser and defaults to port 5000. The GUI provides the opportunity to change the port if necessary. The *-l* flag is a boolean flag, and its inclusion is essential if the aim is to launch into the reporting suite.
- *-loc*
  - Flag for the aboslute path to the current directory. The path is required to locate a current reporting.db or a store as the new location in the settings.
- *-a, –analyse*
  - Initiates the analyisis of the file name supplied as the first argument directly after the *supplychainpy* command e.g.:

    $ supplychainpy <filename> -a -loc <absolute-path-to-current-directory> -l
- *-lx, –launch-console*
  - Launches supplychainpy reporting on the default port, without GUI interface. Uses default port (5000) unless another port is specified. Appropriate for running the reports on a sever. Currently, this is only for testing. The Werkzeug web server that supplied with flask serves the pages for the reports. Werkzeug is not a production web server. It is sufficient as a local web server for the reporting application on a client system. In the coming releases deployment using a more robust web server such as Nginx or Gunicorn will be documented for Server type implementation.
- *-cur*
  - The flag Sets the currency for the analysis and should match the raw data. IMPORTANT: Currency conversion does not occur by setting this flag. The default currency is US Dollars (USD).

- *–host*
  - Sets the host for the server (defaults 127.0.0.1).
- *–debug*
  - Runs in debug mode.
- *-p, –port*
  - Specify Port to use for local server e.g. 8080 (default: 5000).
- *-c*
  - Enter chat mode with the Dash bot from the command line.

CHAPTER 2

Installation

The easiest way to install supplychainpy is via pip:

```
pip install supplychainpy
python -m textblob.dowload_copora
```

The option also exists to install from source. Clone the package from Github.

2.1 Python Version

- Python 3.5

2.2 Dependencies

- Numpy
- Pandas
- Flask
- Flask-Restful
- Flask-Restless
- Flask-Script
- Flask-SqlAlchemy
- Flask-Uploads
- Flask-WTF
- Scipy
- SqlAlchemy

- TextBlob

2.3 Optional Dependencies

- matplotlib
- openpyxl
- xlwings

Installing the Anaconda package may be preferable as it comes with Python 3.5 and all the dependencies.

CHAPTER 3

Quick Guide

Warning: The library is currently under development and in planning stages. The library should not be used in production at this time.

3.1 Overview

Supplychainpy is a Python library for supply chain analysis, modelling and simulation. The library assists a workflow that is reliant on spreadsheets.

This quick guide assumes analysts have the requisite domain knowledge, and predominantly use Excel. Some knowledge of Python or programming is assumed, although those new to data analysis or using Python will likely be able to follow with assistance from other material.

The following guide assumes that the supplychainpy library has already been installed. If not, please use the instructions for *Installation*.

3.2 Up and Running

Typically, inventory analysis requires several formulas, manual processes, possibly some pivot tables and in some cases VBA. Using the supplychainpy library can reduce the time taken and effort made for the same analysis.

A simple analysis for an individual SKU can be carried out by using:

```
>>> from supplychainpy import model_inventory
>>> yearly_demand = {'jan': 75, 'feb': 75, 'mar': 75, 'apr': 75,
>>>                  'may': 75, 'jun': 75, 'jul': 25,'aug': 25,
>>>                  'sep': 25, 'oct': 25, 'nov': 25, 'dec': 25}
>>> summary = model_inventory.analyse_orders(self._yearly_demand,  sku_id='RX983-90',␣
↪lead_time= Decimal(3),
>>>                                          unit_cost=Decimal(50.99), reorder_
↪cost=Decimal(400),
```

```
>>>                                                 z_value=Decimal(1.28), retail_
↪price=Decimal(600), quantity_on_hand=Decimal(390)))
>>> print(summary)
```

```
{'revenue': '360000',
'total_orders': '600',
'orders': {'feb': 75, 'dec': 25, 'jan': 75, 'jun': 75, 'may': 75, 'mar': 75, 'aug':␣
↪25, 'sep': 25, 'jul': 25, 'oct': 25, 'nov': 25, 'apr': 75},
'shortages': '0',
'reorder_level': '142',
'safety_stock': '55',
'average_orders': '50',
'standard_deviation': '25',
'excess_stock': '161',
'sku': 'RX983-90',
'ABC_XYZ_Classification': '',
'demand_variability': '0.500',
'reorder_quantity': '56',
'quantity_on_hand': '390',
'currency': 'USD',
'unit_cost': '50.99'}
```

Note: The signature for the *analysed_orders* function has changed. Moving from release-0.0.3 to release-0.0.4, **Retail price** and **quantity on hand** are required arguments.

The same analysis can be made by supplying a pre-formatted *.csv*, *.txt* or Pandas *DataFrame* containing several SKU or entire inventory profile. The format for the file can be found ‘ here <https://github.com/KevinFasusi/supplychainpy/blob/master/supplychainpy/sample_data/complete_dataset_small.csv>‘_ An example using file:

```
>>> from supplychainpy.model_inventory import analyse
>>> from supplychainpy.sample_data.config import ABS_FILE_PATH
>>> from decimal import Decimal
>>> analysed_data = analyse(file_path=ABS_FILE_PATH['COMPLETE_CSV_SM'],
...                         z_value=Decimal(1.28),
...                         reorder_cost=Decimal(400),
...                         retail_price=Decimal(455),
...                         file_type='csv',
...                         currency='USD')
>>> analysis = [demand.orders_summary() for demand in analysed_data]
```

```
{'quantity_on_hand': '1003',
'currency': 'USD',
'orders': {'demand': ('1509', '1855', '2665', '1841', '1231', '2598', '1988', '1988',
↪'2927', '2707', '731', '2598')},
'economic_order_variable_cost': '15708.41',
 'ABC_XYZ_Classification': 'BY',
 'reorder_level': '4069',
 'safety_stock': '1165',
 'shortages': '5969',
 'demand_variability': '0.314',
 'excess_stock': '0',
 'standard_deviation': '644',
 'average_orders': '2053.1667',
 'unit_cost': '1001',
 'economic_order_quantity': '44',
```

```
 'reorder_quantity': '13',
 'revenue': '123190000',
 'sku': 'KR202-209',
 'total_orders': '24638'},
```

The library also supports Pandas using a *DataFrame*. The following example shows how to use the library to perform an inventory analysis if a *DataFrame* is the preference:

```
>>> import pandas as pd
>>> r_df = pd.read_csv(ABS_FILE_PATH['COMPLETE_CSV_SM'])
>>> analyse_kv = dict(
...     df=raw_df,
...     start=1,
...     interval_length=12,
...     interval_type='months',
...     z_value=Decimal(1.28),
...     reorder_cost=Decimal(400),
...     retail_price=Decimal(455),
...     currency='USD'
... )
>>> analysis_df = analyse(**analyse_kv)
```

3.2.1 Summarising the Analysis

Use the *describe_sku* method a retrieve a summary for a specific skus:

```
>>> from supplychainpy.inventory.summarise import Inventory
>>> from supplychainpy.model_inventory import analyse
>>> from supplychainpy.sample_data.config import ABS_FILE_PATH
>>> from decimal import Decimal
>>> analysed_data = analyse(file_path=ABS_FILE_PATH['COMPLETE_CSV_SM'],
...                         z_value=Decimal(1.28),
...                         reorder_cost=Decimal(400),
...                         retail_price=Decimal(455),
...                         file_type='csv',
...                         currency='USD')
>>> filtered_summary = Inventory(processed_orders=analysed_orders)
>>> sku_summary = [summary for summary in filtered_summary.describe_sku('KR202-209')]
>>> print(sku_summary)
```

```
{'economic_order_quantity': '44',
'ABC_XYZ_Classification': 'BY',
'sku': 'KR202-209',
'shortages': '5969',
'demand_variability': '0.314',
'reorder_level': '4069',
'reorder_quantity': '13',
'unit_cost': '1001',
'currency': 'UNKNOWN',
'standard_deviation': '644',
'revenue': '123190000',
'average_orders': '2053.1667',
'safety_stock': '1165',
'quantity_on_hand': '1003',
'orders': {'demand': ('1509', '1855', '2665', '1841', '1231', '2598', '1988', '1988',
↪'2927', '2707', '731', '2598')},
```

```
'excess_stock': '0',
'economic_order_variable_cost': '15708.41',
'total_orders': '24638'}
```

For more coverage of the library please take a look at the Jupyter notebooks is available from here . The content of notebooks can be found in *Inventory Modeling and Analysis Made Easy with Supplychainpy* and *Using supplychainpy with Pandas, Jupyter and Matplotlib*.

CHAPTER 4

Supplychainpy Reporting Suite

To further indicate the merit of the library in a more direct way and showcase the possibilities offered, release 0.0.4 debuts a reporting feature. The supplychainpy reporting feature allows analysts to get a quick overview of their analysis and provides a tool for communicating their insights very quickly. The reports bring some of the data analysis capabilities of the library to life with a complimentary suite of charts, tables, KPIs and an interactive Bot.

The reports aim to:

1. provide the ability to visualise data and spot trends, allowing analysts to get a "feel" for their data.
2. provide a set of generic default reports, to showcase some general uses cases and highlight the capabilities of the library.
3. identify areas of interest for further exploration.

4.1 Launch from Cli

The command line arguments can be viewed using the *–help* flag. There are several options for launching the reports. Using the the reporting function generates a sqlite database called *reporting*. To process a CSV file and initiate the reporting suite directly after, navigate to a directory suitable for storing the CSV and resulting database. Use the following commands:

Linux and Mac

```
supplychainpy filename.csv -a -loc ~/absolute/path/to/current/directory -l -cur EUR
```

Windows

```
supplychainpy filename.csv -a -loc drive:\absolute\path\to\current\directory -l -cur␣
↪EUR
```

Importantly the the currency flag (-cur) if unspecified will default to USD. Other optional arguments include the host (–host default: 127.0.0.1) and port (-p default: 5000) arguments. Setting the host and ports allows the *-l* arguments can be replaced by the *-lx*. The *-l* arguments launch a small intermediary GUI for setting the port before launching the

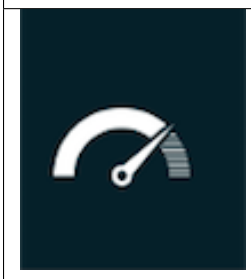
reports in a web browser. The *-lx* argument start the reporting process but does not launch a GUI or a browser window and instead expects the user to open the browser and navigate to the address hosting the reports as specified in the CLI.

4.2 Reporting Suite Walk Through

The reporting suite launches on the 'Dashboard' page. The Dashboard is split into three section: Classification Breakdown, Top 10 Shortages and Top 10 Excess. The Dashboard hosts three toggle switches at the top of the screen for toggling each section in and out of view.

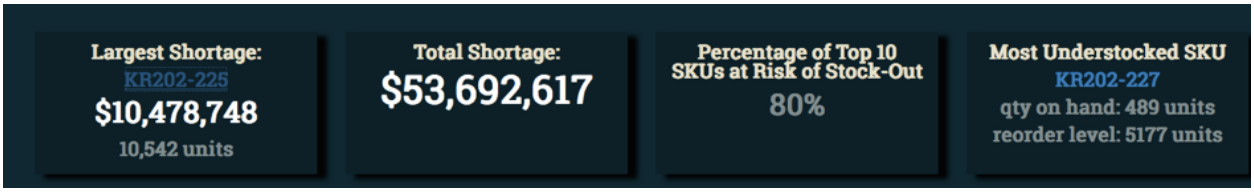
4.2.1 Navigation

The Navigation panel is hidden to the left of the browser and can be toggled into view by clicking on the 'supplychainpy' logo icon.

Icon	Description
	The analysis icon navigates to the pages hosting the raw analysis data in tabular form.
	The dashboard icon navigates to the pages hosting the top level summary of the inventory profile.
	The news feed icon navigates to the pages hosting the recommendations feed.
	The bot icon navigates to the pages hosting the chat bot, for interrogating the data using natural language.
	The contact icon navigates to a page containing the projects contact details.

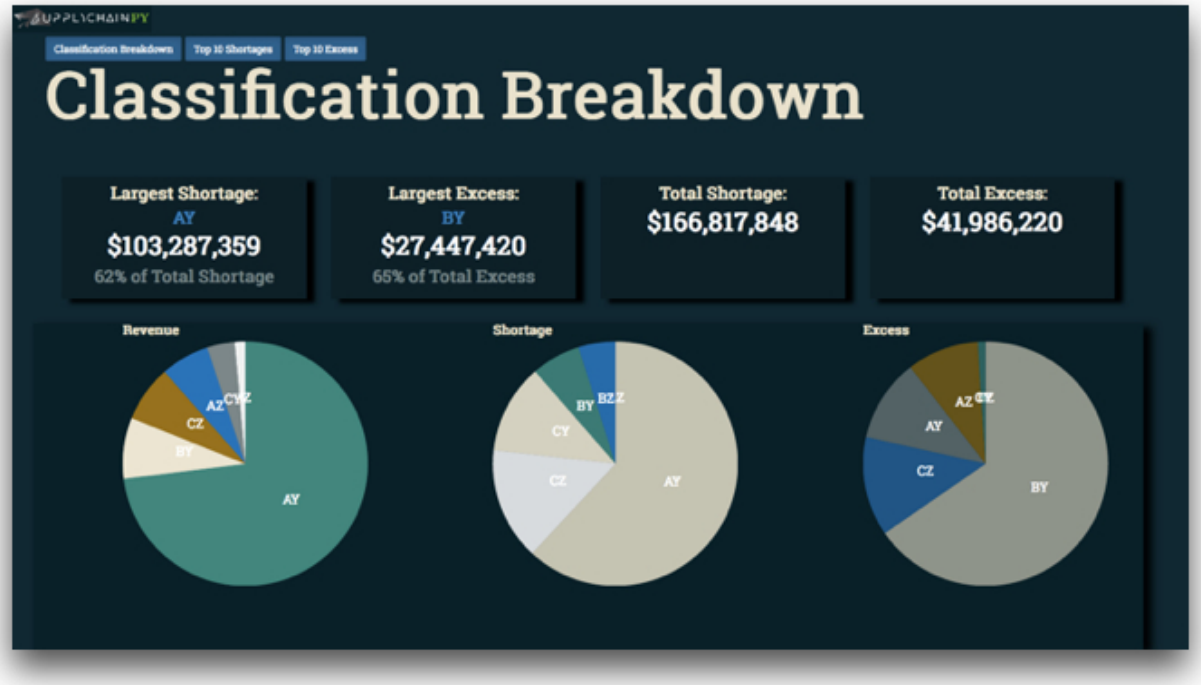
4.2.2 The Head up Display (HUD)

The reports use what we like to refer to as a "Heads up Display" (HUD) to highlight key values and KPIs. As seen below, the HUD is a row of boxes (Slates), that sits on top of the main charts, analysis and tabular breakdowns.



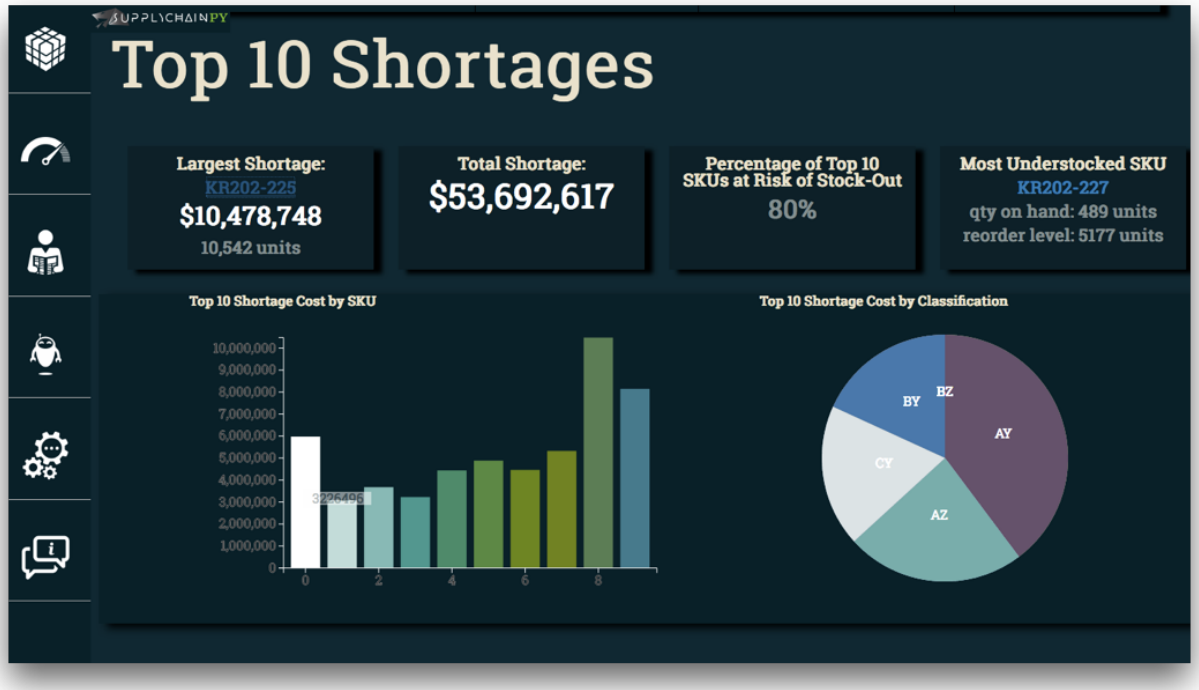
4.2.3 Classification Breakdown

The classification breakdown summarises the Inventory Profile using Pareto and variance analysis, to indicate the contribution the SKU makes to revenue and the variability in demand respectively. This section shows which classification has the largest excess and shortage as well as breaking down revenue, shortages and excess by category.



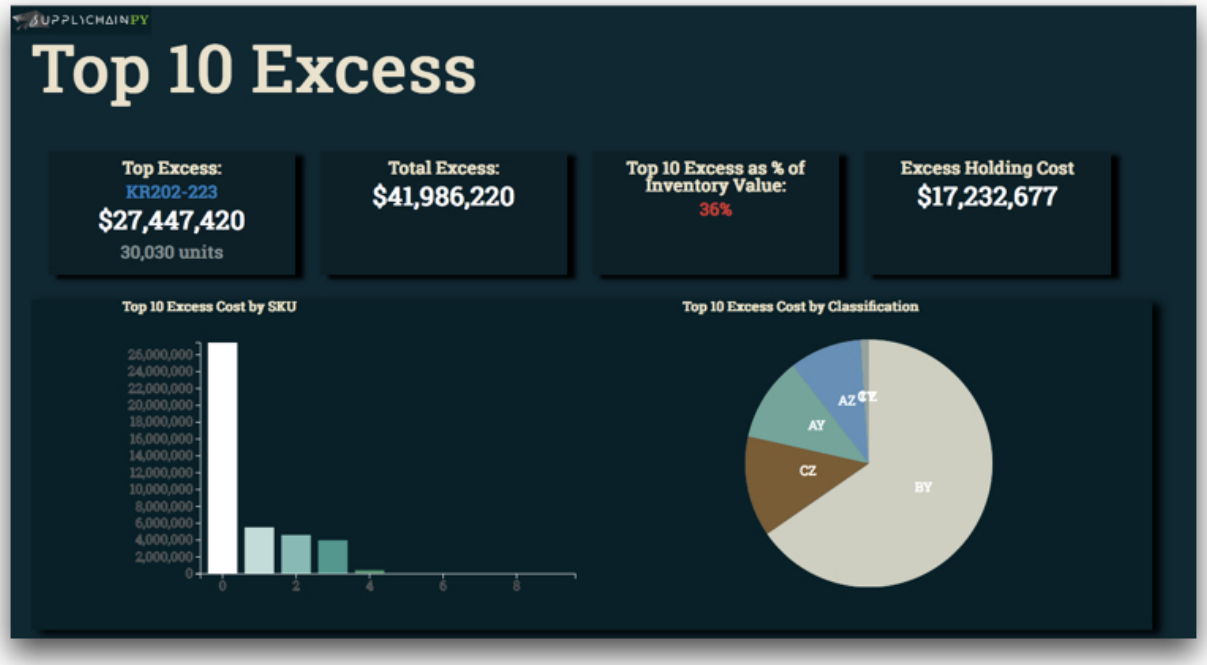
4.2.4 Top 10 Shortages

This section indicates which 10 SKUs are responsible for the most shortages.



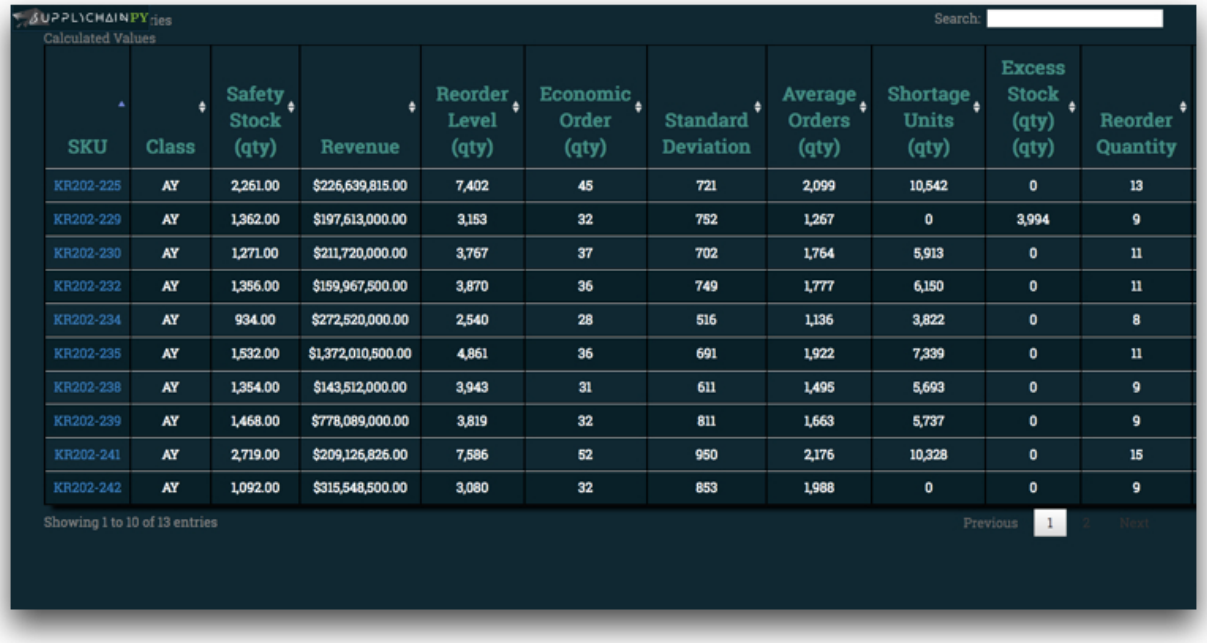
4.2.5 Top 10 Excess

This section indicates which 10 SKUs are responsible for the most excess stock based on their unit cost.



4.2.6 Analysis Cube

The analysis cube page hosts the tabulated data for all data points within the analysis.



SUPPLYCHAINPYries

Calculated Values

Search:

SKU	Class	Safety Stock (qty)	Revenue	Reorder Level (qty)	Economic Order (qty)	Standard Deviation	Average Orders (qty)	Shortage Units (qty)	Excess Stock (qty) (qty)	Reorder Quantity
KR202-225	AY	2,261.00	$226,639,815.00	7,402	45	721	2,099	10,542	0	13
KR202-229	AY	1,362.00	$197,613,000.00	3,153	32	752	1,267	0	3,994	9
KR202-230	AY	1,271.00	$211,720,000.00	3,767	37	702	1,764	5,913	0	11
KR202-232	AY	1,356.00	$159,967,500.00	3,870	36	749	1,777	6,150	0	11
KR202-234	AY	934.00	$272,520,000.00	2,540	28	516	1,136	3,822	0	8
KR202-235	AY	1,532.00	$1,372,010,500.00	4,861	36	691	1,922	7,339	0	11
KR202-238	AY	1,354.00	$143,512,000.00	3,943	31	611	1,495	5,693	0	9
KR202-239	AY	1,468.00	$778,089,000.00	3,819	32	811	1,663	5,737	0	9
KR202-241	AY	2,719.00	$209,126,826.00	7,586	52	950	2,176	10,328	0	15
KR202-242	AY	1,092.00	$315,548,500.00	3,080	32	853	1,988	0	0	9

Showing 1 to 10 of 13 entries

Previous 1 2 Next

4.2.7 Dash the Bot

The Chat Bot provides a simply method of querying the analysis using natural language.

4.2.8 Recommendations Feed

The recommendation feed for all the auto-generated recommendation for each SKU.

CHAPTER 5

Inventory Modeling and Analysis Made Easy with Supplychainpy

The following is taken from the jupyter notebook title '0.0.4-Inventory-Modeling-v1' found here . For a more interactive experience please retrieve this notebook and run with jupyter.

This workbook assumes some familiarity and proficiency in programming with Python. Understanding list comprehensions, conditional logic, loops and functions are a basic prequisite for continuing with this workbook.

Typically, inventory analysis using Excel requires several formulas, manual processes, possibly some pivot tables and in some cases VBA to achieve. Using the supplychainpy library can reduce the time taken and effort made for the same analysis.

```
from supplychainpy.model_inventory import analyse
from decimal import Decimal
from supplychainpy.sample_data.config import ABS_FILE_PATH
```

The first two imports are manditory, the second import is for using the sample data in the `supplychainpy` library. When working with a different file, supply the file path to the `file_path` parameter. The data supplied for analysis can be from a `csv` or a database ETL process.

The sample data is a `csv` formatted file:

```
with open(ABS_FILE_PATH['COMPLETE_CSV_SM'],'r') as raw_data:
    for line in raw_data:
        print(line)
```

```
Sku,jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,unit cost,lead-time,retail_price,
↪quantity_on_hand,backlog

KR202-209,1509,1855,2665,1841,1231,2598,1988,1988,2927,2707,731,2598,1001,2,5000,1003,
↪10

KR202-210,1006,206,2588,670,2768,2809,1475,1537,919,2525,440,2691,394,2,1300,3224,10

KR202-211,1840,2284,850,983,2737,1264,2002,1980,235,1489,218,525,434,4,1200,390,10

KR202-212,104,2262,350,528,2570,1216,1101,2755,2856,2381,1867,2743,474,3,10,390,10
```

```
KR202-213,489,954,1112,199,919,330,561,2372,921,1587,1532,1512,514,1,2000,2095,10

KR202-214,2416,2010,2527,1409,1059,890,2837,276,987,2228,1095,1396,554,2,1800,55,10

KR202-215,403,1737,753,1982,2775,380,1561,1230,1262,2249,824,743,594,1,2500,4308,10

KR202-216,2908,929,684,2618,1477,1508,765,43,2550,2157,937,1201,634,3,3033,34,10

KR202-217,2799,2197,1647,2263,224,2987,2366,588,1140,869,1707,1180,674,3,5433,390,10

KR202-218,1333,402,804,318,1408,830,1028,534,1871,2730,2022,94,714,2,3034,3535,10

KR202-219,813,969,745,1001,2732,1987,717,599,2722,171,639,2108,754,3,5000,334,10

KR202-220,1481,905,1067,2513,861,1670,650,2630,1245,997,1936,2780,794,3,7500,3434,10

KR202-221,771,2941,1360,2714,1801,1744,1428,1660,436,578,1956,1101,834,2,4938,4433,10

KR202-222,2349,4,345,524,340,2698,2137,1164,498,1583,1241,2965,874,2,4922,3435,10

KR202-223,2045,2055,552,81,2780,176,2316,1475,2566,1678,1553,2745,914,1,4894,34533,10

KR202-224,2482,1887,1911,1446,2939,1241,1281,692,119,627,1941,1383,954,2,2942,33,10

KR202-225,2744,2770,2697,1726,1776,2264,332,2420,2722,1161,1986,2587,994,6,8999,2000,
↪10

KR202-226,2509,914,903,877,1859,2263,383,593,236,189,920,1686,1034,3,4342,4344,10

KR202-227,368,2502,2955,2994,1270,2884,2208,699,854,877,2320,160,1074,3,4920,489,10

KR202-228,1468,1109,2464,2799,948,589,2858,1140,501,2691,93,1060,1114,2,15000,9439,10

KR202-229,2114,198,1479,1249,1475,744,407,2280,226,2285,796,1948,1154,2,13000,8939,10

KR202-230,1023,1150,1672,2026,1590,441,2484,2300,2928,1082,2064,2412,1194,2,10000,349,
↪10

KR202-231,482,546,299,2304,2953,1029,1863,2809,454,927,2488,2341,1234,4,9999,3434,10

KR202-232,614,2138,962,2017,2398,2963,2189,1804,414,2016,1350,2464,1274,2,7500,234,10

KR202-233,2395,2521,2157,728,1028,43,138,826,570,2825,181,787,1314,4,6000,349,10

KR202-234,1336,1478,865,533,1562,422,2287,1302,1230,1059,1153,399,1354,2,20000,324,10

KR202-235,2565,2762,2721,1431,845,2163,2413,2227,1753,740,1139,2300,1394,3,59500,850,
↪10

KR202-236,1912,1726,1569,316,71,2082,108,174,1974,609,2896,566,1434,3,2300,4930,10

KR202-237,2153,1112,16,130,590,2619,2576,2390,2567,1531,842,242,1474,2,4500,9483,10

KR202-238,1417,2044,1981,1936,2377,780,1544,1521,51,1056,1876,1356,1514,3,8000,839,10

KR202-239,2717,2186,2300,677,2157,2328,1917,2519,561,281,1162,1146,1554,2,39000,433,10
```

```
KR202-240,1015,741,2754,2925,2302,695,2869,440,406,1083,2334,1015,1594,3,3943,390,10

KR202-241,3050,1507,3637,1112,1963,1675,898,1986,2262,3895,1229,2904,769,5,8007,2125,
↪10

KR202-242,1875,2368,830,823,868,1409,1845,3095,3247,1894,2558,3048,1819,1,13225,1253,
↪10

KR202-243,1717,593,3006,2935,3139,2753,3247,3845,1720,3413,3399,2799,1120,3,14682,
↪1128,10

KR202-244,2383,2046,2487,3827,1674,3118,2849,2233,3888,2566,2216,3817,1067,5,11997,
↪1191,10

KR202-245,1115,2694,3038,3366,1058,2724,2863,1930,1787,838,3087,1565,1623,2,12876,611,
↪10

KR202-246,3108,1197,2472,1264,3179,3638,1268,1581,3456,1630,1788,2288,608,2,6548,2192,
↪10

KR202-247,3439,1854,652,1827,1645,2257,2733,1337,2034,2106,877,2409,1578,2,10463,1017,
↪10
```

It is probable that getting the data to this format will require 'extracting' from a database and 'transforming' data before 'loading' into the `analyse` function. This can be achieved with an orm like slqalchemy or using the driver for the database in question. Supplychainpy works with Pandas so performing the transformations using Pandas may be an idea. The `DataFrame` or file passed as an argument must be identical to the format above (future versions of supplychainpy will be more lenient and attempt to identify if a minimum requirement has been met).

The ETL process is not covered in this workbook but on the 'roadmap' for `supplychainpy` is the automation of this process.

So now that we have the data in the correct format we can proceed with the anlysis.

```
#%%timeit
analysed_inventory_profile= analyse(file_path=ABS_FILE_PATH['COMPLETE_CSV_SM'],
                                                                    z_value=Decimal(1.28),
                                                                    reorder_
↪cost=Decimal(400),
                                                                    file_type='csv')
```

The variable `analysed_inventory_profile` now contains a collection (list) of `UncertainDemand` objects (one per SKU). Each object contains the analysis for each SKU. The analysis include the following:

- safety stock
- total_orders
- standard_deviation
- quantity_on_hand': '1003
- economic_order_variable_cost
- sku
- economic_order_quantity
- unit_cost
- demand_variability

- average_orders
- excess_stock
- currency
- ABC_XYZ_Classification
- shortages
- reorder_level
- revenue
- reorder_quantity
- safety_stock
- orders

The listed summary items can be retrieved by calling the method `orders_summary()` on each object. The quickest way to do this is with a list comprehension.

```
analysis_summary = [demand.orders_summary() for demand in analysed_inventory_profile]
```

For the intrepid reader who did not heed the warning about the prerequisites and is now scratching their head wondering "what manner of black magic is this," here is a quick overview on list comprehensions. In short, the above code is similar to the code below:

```
analysis_summary =[]
for demand in analysed_inventory_profile:
    analysis_summary.append(demand.orders_summary())
```

The former is much more readable and in truth quite addictive (hence the warning, the love for list comprehensions runs deep).

5.1 Exploring the results

To make sense of the results we can filter our analysis using standard python scripting techniques. For example to retrieve the whole `summary` for the SKU `KR202-209` we can do something like this:

```
%%timeit
sku_summary = [demand.orders_summary() for demand in analysed_inventory_profile if␣
↪demand.orders_summary().get('sku')== 'KR202-209']
#print(sku_summary)
```

```
1000 loops, best of 3: 885 µs per loop
```

The inventory classification ABC XYZ denotes the SKUs contribution to revenue and demand volatility. `AX` SKUs typically exhibit steady demand and contribute 80% of the revenue value for the period being analysed. Further explanation on `ABC XYZ` analysis can be found here.

As a more traditional way of grouping SKUs by behaviour, it is also likely to be used for generating inventory policies and for further exploration of the inventory profile. To retrive all the summaries for a particular classification, we could do something like this:

```
ay_classification_summary = [demand.orders_summary() for demand in analysed_inventory_
↪profile if demand.orders_summary().get('ABC_XYZ_Classification')== 'AY']
print(ay_classification_summary)
```

```
[{'total_orders': '25185', 'standard_deviation': '721', 'quantity_on_hand': '2000',
↪'economic_order_variable_cost': '15826.20', 'sku': 'KR202-225', 'economic_order_
↪quantity': '45', 'unit_cost': '994', 'demand_variability': '0.344', 'average_orders
↪': '2098.75', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_XYZ_Classification':
↪'AY', 'shortages': '10542', 'reorder_level': '7402', 'revenue': '226639815',
↪'reorder_quantity': '13', 'safety_stock': '2261', 'orders': {'demand': ('2744',
↪'2770', '2697', '1726', '1776', '2264', '332', '2420', '2722', '1161', '1986', '2587
↪')}}, {'total_orders': '15201', 'standard_deviation': '752', 'quantity_on_hand':
↪'8939', 'economic_order_variable_cost': '13248.03', 'sku': 'KR202-229', 'economic_
↪order_quantity': '32', 'unit_cost': '1154', 'demand_variability': '0.594', 'average_
↪orders': '1266.75', 'excess_stock': '3994', 'currency': 'UNKNOWN', 'ABC_XYZ_
↪Classification': 'AY', 'shortages': '0', 'reorder_level': '3153', 'revenue':
↪'197613000', 'reorder_quantity': '9', 'safety_stock': '1362', 'orders': {'demand': (
↪'2114', '198', '1479', '1249', '1475', '744', '407', '2280', '226', '2285', '796',
↪'1948')}}, {'total_orders': '21172', 'standard_deviation': '702', 'quantity_on_hand
↪': '349', 'economic_order_variable_cost': '15903.60', 'sku': 'KR202-230', 'economic_
↪order_quantity': '37', 'unit_cost': '1194', 'demand_variability': '0.398', 'average_
↪orders': '1764.3333', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_XYZ_
↪Classification': 'AY', 'shortages': '5913', 'reorder_level': '3767', 'revenue':
↪'211720000', 'reorder_quantity': '11', 'safety_stock': '1271', 'orders': {'demand':␣
↪('1023', '1150', '1672', '2026', '1590', '441', '2484', '2300', '2928', '1082',
↪'2064', '2412')}}, {'total_orders': '21329', 'standard_deviation': '749', 'quantity_
↪on_hand': '234', 'economic_order_variable_cost': '16488.55', 'sku': 'KR202-232',
↪'economic_order_quantity': '36', 'unit_cost': '1274', 'demand_variability': '0.422',
↪ 'average_orders': '1777.4167', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_
↪XYZ_Classification': 'AY', 'shortages': '6150', 'reorder_level': '3870', 'revenue':
↪'159967500', 'reorder_quantity': '11', 'safety_stock': '1356', 'orders': {'demand':␣
↪('614', '2138', '962', '2017', '2398', '2963', '2189', '1804', '414', '2016', '1350
↪', '2464')}}, {'total_orders': '13626', 'standard_deviation': '516', 'quantity_on_
↪hand': '324', 'economic_order_variable_cost': '13586.45', 'sku': 'KR202-234',
↪'economic_order_quantity': '28', 'unit_cost': '1354', 'demand_variability': '0.454',
↪ 'average_orders': '1135.5', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_XYZ_
↪Classification': 'AY', 'shortages': '3822', 'reorder_level': '2540', 'revenue':
↪'272520000', 'reorder_quantity': '8', 'safety_stock': '934', 'orders': {'demand': (
↪'1336', '1478', '865', '533', '1562', '422', '2287', '1302', '1230', '1059', '1153',
↪ '399')}}, {'total_orders': '23059', 'standard_deviation': '691', 'quantity_on_hand
↪': '850', 'economic_order_variable_cost': '17933.46', 'sku': 'KR202-235', 'economic_
↪order_quantity': '36', 'unit_cost': '1394', 'demand_variability': '0.360', 'average_
↪orders': '1921.5833', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_XYZ_
↪Classification': 'AY', 'shortages': '7339', 'reorder_level': '4861', 'revenue':
↪'1372010500', 'reorder_quantity': '11', 'safety_stock': '1532', 'orders': {'demand
↪': ('2565', '2762', '2721', '1431', '845', '2163', '2413', '2227', '1753', '740',
↪'1139', '2300')}}, {'total_orders': '19951', 'standard_deviation': '811', 'quantity_
↪on_hand': '433', 'economic_order_variable_cost': '17612.47', 'sku': 'KR202-239',
↪'economic_order_quantity': '32', 'unit_cost': '1554', 'demand_variability': '0.488',
↪ 'average_orders': '1662.5833', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_
↪XYZ_Classification': 'AY', 'shortages': '5737', 'reorder_level': '3819', 'revenue':
↪'778089000', 'reorder_quantity': '9', 'safety_stock': '1468', 'orders': {'demand': (
↪'2717', '2186', '2300', '677', '2157', '2328', '1917', '2519', '561', '281', '1162',
↪ '1146')}}, {'total_orders': '26118', 'standard_deviation': '950', 'quantity_on_hand
↪': '2125', 'economic_order_variable_cost': '14175.73', 'sku': 'KR202-241',
↪'economic_order_quantity': '52', 'unit_cost': '769', 'demand_variability': '0.437',
↪'average_orders': '2176.5', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_XYZ_
↪Classification': 'AY', 'shortages': '10328', 'reorder_level': '7586', 'revenue':
↪'209126826', 'reorder_quantity': '15', 'safety_stock': '2719', 'orders': {'demand':␣
↪('3050', '1507', '3637', '1112', '1963', '1675', '898', '1986', '2262', '3895',
↪'1229', '2904')}}, {'total_orders': '23860', 'standard_deviation': '853', 'quantity_
↪on_hand': '1253', 'economic_order_variable_cost': '20838.38', 'sku': 'KR202-242',
↪'economic_order_quantity': '32', 'unit_cost': '1819', 'demand_variability': '0.429',
↪ 'average_orders': '1988.3333', 'excess_stock': '0', 'currency': 'UNKNOWN', 'ABC_
↪XYZ_Classification': 'AY', 'shortages': '0', 'reorder_level': '3080', 'revenue':
↪'315348500', 'reorder_quantity': '9', 'safety_stock': '1092', 'orders': {'demand': (
↪'1875', '2368', '830', '823', '868', '1409', '1845', '3095', '3247', '1894', '2558',
↪ '3048')}}, {'total_orders': '32566', 'standard_deviation': '882', 'quantity_on_hand
↪': '1128', 'economic_order_variable_cost': '19103.09', 'sku': 'KR202-243',
```

Using a built-in feature of the library provides a quicker way to filter the results. For example a quciker way to filter for SKU `KR202-209`, is through the use of `Inventory` class in the `summarise` module.

```
from supplychainpy.inventory.summarise import Inventory
filtered_summary = Inventory(analysed_inventory_profile)
```

```
%%timeit
sku_summary = [summary for summary in filtered_summary.describe_sku('KR202-209')]
#print(sku_summary)
```

```
10000 loops, best of 3: 190 µs per loop
```

Using the import Inventory specifically built to filter the analysis is faster and syntactically cleaner for eaier to read and understand code. The `Inventory` summary class also provides a more detailed summary of the SKU with additional KPIs and metric in context of the whole inventory profile. The summary ranks and performs some comparative analysis for more insight.

The descriptive summary includes:

- shortage_rank
- min_orders
- excess_units
- revenue_rank
- excess_rank
- average_orders
- gross_profit_margin
- markup_percentage
- max_order
- shortage_cost
- quantity_on_hand
- inventory_turns
- sku_id
- retail_price
- revenue_rank
- shortage_units
- unit_cost
- classification
- safety_stock_cost
- safety_stock_units
- safety_stock_rank
- percentage_contribution_revenue
- gross_profit_margin

- shortage_rank
- inventory_traffic_light
- unit_cost_rank
- excess_cost
- excess_units
- markup_percentage
- revenue

This is a pretty comprehensive list of descriptors to use for further analysis.

Further summaries can be retrieved, for instance summaries at the inventory classification level of detail can be quite useful when exploring inventory policies:

```
classification_summary =  [summary for summary in filtered_summary.abc_xyz_
↪summary(classification=('AY',), category=('revenue',))]
print(classification_summary)
```

```
[{'AY': {'revenue': 5372496600.0}}]
```

Now we know the total revenue generated by the `AY` SKU class. There is another, slightly more fun way to arrive at this number using `Dash` but more on that latter.

```
top_10_safety_stock_skus =  [summary.get('sku')for summary in filtered_summary.rank_
↪summary(attribute='safety_stock', count=10)]
print(top_10_safety_stock_skus)
```

```
['KR202-241', 'KR202-231', 'KR202-233', 'KR202-227', 'KR202-225', 'KR202-212', 'KR202-
↪240', 'KR202-244', 'KR202-236', 'KR202-211', 'KR202-243']
```

Lets add the safety_stock and create a tuple to see that the results explicitly.

```
top_10_safety_stock_values =  [(summary.get('sku'), summary.get('safety_stock'))for␣
↪summary in filtered_summary.rank_summary(attribute='safety_stock', count=10)]
print(top_10_safety_stock_values)
```

```
[('KR202-241', '2719'), ('KR202-231', '2484'), ('KR202-233', '2472'), ('KR202-227',
↪'2277'), ('KR202-225', '2261'), ('KR202-212', '2164'), ('KR202-240', '2120'), (
↪'KR202-244', '2054'), ('KR202-236', '2045'), ('KR202-211', '2020'), ('KR202-243',
↪'1954')]
```

We can then pass back the list of `top_10_safety_stock_skus` back into the inventory filter and get their breakdown.

```
top_10_safety_stock_summary = [summary for summary in filtered_summary.describe_
↪sku(*top_10_safety_stock_skus)]
#print(top_10_safety_stock_summary)
```

We have only covered a few use cases but we have already achieved a significant amount of analysis with relativley few line of code. The equivalent in Excel would require much more work and many more formulas.

CHAPTER 6

Using supplychainpy with Pandas, Jupyter and Matplotlib

The following is taken from the jupyter notebook title '0.0.4-Using-Supplychainpy-and-Pandas-v1' found here . For a more interactive experience please retrieve this notebook and run with jupyter.

To use supplychainpy with Pandas, we first read a csv file to a Pandas DataFrame.

```
%matplotlib inline

import matplotlib
import pandas as pd

from supplychainpy.model_inventory import analyse
from supplychainpy.model_demand import simple_exponential_smoothing_forecast
from supplychainpy.sample_data.config import ABS_FILE_PATH

raw_df =pd.read_csv(ABS_FILE_PATH['COMPLETE_CSV_SM'])
print(raw_df)
```

```
          Sku   jan   feb   mar   apr   may   jun   jul   aug   sep   oct  0 ␣
↪ KR202-209  1509  1855  2665  1841  1231  2598  1988  1988  2927  2707
1   KR202-210  1006   206  2588   670  2768  2809  1475  1537   919  2525
2   KR202-211  1840  2284   850   983  2737  1264  2002  1980   235  1489
3   KR202-212   104  2262   350   528  2570  1216  1101  2755  2856  2381
4   KR202-213   489   954  1112   199   919   330   561  2372   921  1587
5   KR202-214  2416  2010  2527  1409  1059   890  2837   276   987  2228
6   KR202-215   403  1737   753  1982  2775   380  1561  1230  1262  2249
7   KR202-216  2908   929   684  2618  1477  1508   765    43  2550  2157
8   KR202-217  2799  2197  1647  2263   224  2987  2366   588  1140   869
9   KR202-218  1333   402   804   318  1408   830  1028   534  1871  2730
10  KR202-219   813   969   745  1001  2732  1987   717   599  2722   171
11  KR202-220  1481   905  1067  2513   861  1670   650  2630  1245   997
12  KR202-221   771  2941  1360  2714  1801  1744  1428  1660   436   578
13  KR202-222  2349     4   345   524   340  2698  2137  1164   498  1583
14  KR202-223  2045  2055   552    81  2780   176  2316  1475  2566  1678
15  KR202-224  2482  1887  1911  1446  2939  1241  1281   692   119   627
```

16	KR202-225	2744	2770	2697	1726	1776	2264	332	2420	2722	1161
17	KR202-226	2509	914	903	877	1859	2263	383	593	236	189
18	KR202-227	368	2502	2955	2994	1270	2884	2208	699	854	877
19	KR202-228	1468	1109	2464	2799	948	589	2858	1140	501	2691
20	KR202-229	2114	198	1479	1249	1475	744	407	2280	226	2285
21	KR202-230	1023	1150	1672	2026	1590	441	2484	2300	2928	1082
22	KR202-231	482	546	299	2304	2953	1029	1863	2809	454	927
23	KR202-232	614	2138	962	2017	2398	2963	2189	1804	414	2016
24	KR202-233	2395	2521	2157	728	1028	43	138	826	570	2825
25	KR202-234	1336	1478	865	533	1562	422	2287	1302	1230	1059
26	KR202-235	2565	2762	2721	1431	845	2163	2413	2227	1753	740
27	KR202-236	1912	1726	1569	316	71	2082	108	174	1974	609
28	KR202-237	2153	1112	16	130	590	2619	2576	2390	2567	1531
29	KR202-238	1417	2044	1981	1936	2377	780	1544	1521	51	1056
30	KR202-239	2717	2186	2300	677	2157	2328	1917	2519	561	281
31	KR202-240	1015	741	2754	2925	2302	695	2869	440	406	1083
32	KR202-241	3050	1507	3637	1112	1963	1675	898	1986	2262	3895
33	KR202-242	1875	2368	830	823	868	1409	1845	3095	3247	1894
34	KR202-243	1717	593	3006	2935	3139	2753	3247	3845	1720	3413
35	KR202-244	2383	2046	2487	3827	1674	3118	2849	2233	3888	2566
36	KR202-245	1115	2694	3038	3366	1058	2724	2863	1930	1787	838
37	KR202-246	3108	1197	2472	1264	3179	3638	1268	1581	3456	1630
38	KR202-247	3439	1854	652	1827	1645	2257	2733	1337	2034	2106

	nov	dec	unit cost	lead-time	retail_price	quantity_on_hand	backlog
0	731	2598	1001	2	5000	1003	10
1	440	2691	394	2	1300	3224	10
2	218	525	434	4	1200	390	10
3	1867	2743	474	3	10	390	10
4	1532	1512	514	1	2000	2095	10
5	1095	1396	554	2	1800	55	10
6	824	743	594	1	2500	4308	10
7	937	1201	634	3	3033	34	10
8	1707	1180	674	3	5433	390	10
9	2022	94	714	2	3034	3535	10
10	639	2108	754	3	5000	334	10
11	1936	2780	794	3	7500	3434	10
12	1956	1101	834	2	4938	4433	10
13	1241	2965	874	2	4922	3435	10
14	1553	2745	914	1	4894	34533	10
15	1941	1383	954	2	2942	33	10
16	1986	2587	994	6	8999	2000	10
17	920	1686	1034	3	4342	4344	10
18	2320	160	1074	3	4920	489	10
19	93	1060	1114	2	15000	9439	10
20	796	1948	1154	2	13000	8939	10
21	2064	2412	1194	2	10000	349	10
22	2488	2341	1234	4	9999	3434	10
23	1350	2464	1274	2	7500	234	10
24	181	787	1314	4	6000	349	10
25	1153	399	1354	2	20000	324	10
26	1139	2300	1394	3	59500	850	10
27	2896	566	1434	3	2300	4930	10
28	842	242	1474	2	4500	9483	10

```
29  1876  1356       1514          3          8000                839       10
30  1162  1146       1554          2         39000                433       10
31  2334  1015       1594          3          3943                390       10
32  1229  2904        769          5          8007               2125       10
33  2558  3048       1819          1         13225               1253       10
34  3399  2799       1120          3         14682               1128       10
35  2216  3817       1067          5         11997               1191       10
36  3087  1565       1623          2         12876                611       10
37  1788  2288        608          2          6548               2192       10
38   877  2409       1578          2         10463               1017       10
```

Passing a Pandas `DataFrame` as a keyword parameter (df=) returns a DataFrame with the inventory profile analysed. Excluding the import statements this can be achieved in 3 lines of code. There are several columns, so the print statement has been limited to a few.

```
orders_df = raw_df[['Sku','jan','feb','mar','apr', 'may', 'jun', 'jul', 'aug', 'sep',
↪'oct', 'nov', 'dec']]
#orders_df.set_index('Sku')
analysis_df = analyse(df=raw_df, start=1, interval_length=12, interval_type='months')
print(analysis_df[['sku','quantity_on_hand', 'excess_stock', 'shortages', 'ABC_XYZ_
↪Classification']])
```

```
          sku quantity_on_hand excess_stock shortages ABC_XYZ_Classification
0   KR202-209             1003            0       5969                     BY
1   KR202-210             3224            0          0                     CY
2   KR202-211              390            0       7099                     CY
3   KR202-212              390            0       7759                     CY
4   KR202-213             2095            0          0                     CY
5   KR202-214               55            0       5824                     CY
6   KR202-215             4308          732          0                     CY
7   KR202-216               34            0       6999                     CY
8   KR202-217              390            0       7245                     BY
9   KR202-218             3535            0          0                     CZ
10  KR202-219              334            0       5917                     CZ
11  KR202-220             3434            0          0                     BY
12  KR202-221             4433            0          0                     BY
13  KR202-222             3435            0          0                     CZ
14  KR202-223            34533        30030          0                     BY
15  KR202-224               33            0       5580                     CY
16  KR202-225             2000            0      10542                     AY
17  KR202-226             4344            0          0                     CZ
18  KR202-227              489            0       7587                     BZ
19  KR202-228             9439         3572          0                     AZ
20  KR202-229             8939         3994          0                     AY
21  KR202-230              349            0       5913                     AY
22  KR202-231             3434            0          0                     AZ
23  KR202-232              234            0       6150                     AY
24  KR202-233              349            0       6856                     CZ
25  KR202-234              324            0       3822                     AY
26  KR202-235              850            0       7339                     AY
27  KR202-236             4930            0          0                     CZ
28  KR202-237             9483         3742          0                     CZ
29  KR202-238              839            0       5693                     BY
30  KR202-239              433            0       5737                     AY
31  KR202-240              390            0       7094                     CZ
32  KR202-241             2125            0      10328                     AY
33  KR202-242             1253            0          0                     AY
34  KR202-243             1128            0      10227                     AY
```

```
35  KR202-244             1191           0     13200                    AY
36  KR202-245              611           0      7081                    AY
37  KR202-246             2192           0         0                    AY
38  KR202-247             1017           0      5776                    AY
```

Before we can make a forecast we need to select a SKU from the `analysis_df` variable, slice the row to retrive only orders data and convert to a `Series`.

```
row_ds = raw_df[raw_df['Sku']=='KR202-212'].squeeze()
print(row_ds[1:12])
```

```
jan     104
feb    2262
mar     350
apr     528
may    2570
jun    1216
jul    1101
aug    2755
sep    2856
oct    2381
nov    1867
Name: 3, dtype: object
```

Now that we have a `series` of orders data fro the SKU `KR202-212`, we can now perform a forecast using the `model_demand` module. We can perform a simple_exponential_smoothing_forecast by passing the forecasting function the orders data using the keyword parameter `ds=`.

```
ses_df = simple_exponential_smoothing_forecast(ds=row_ds[1:12], length=12, smoothing_
↪level_constant=0.5)
print(ses_df)
```

```
{'statistics': {'pvalue': 0.0047852515832242743, 'test_statistic': 3.8634855288615153,
↪ 'std_residuals': 4793.7283216530095, 'intercept': 377.59999999999991, 'trend':␣
↪True, 'slope': 224.4909090909091, 'slope_standard_error': 58.105797838218294},
↪'alpha': 0.5, 'forecast_breakdown': [{'squared_error': 2345353.024793389, 'alpha':␣
↪0.5, 'demand': 104, 'one_step_forecast': 1635.4545454545455, 't': 1, 'level_
↪estimates': 869.72727272727275, 'forecast_error': -1531.4545454545455}, {'squared_
↪error': 1938423.3471074379, 'alpha': 0.5, 'demand': 2262, 'one_step_forecast': 869.
↪72727272727275, 't': 2, 'level_estimates': 1565.8636363636365, 'forecast_error':␣
↪1392.2727272727273}, {'squared_error': 1478324.3822314052, 'alpha': 0.5, 'demand':␣
↪350, 'one_step_forecast': 1565.8636363636365, 't': 3, 'level_estimates': 957.
↪93181818181824, 'forecast_error': -1215.8636363636365}, {'squared_error': 184841.
↪36828512402, 'alpha': 0.5, 'demand': 528, 'one_step_forecast': 957.93181818181824,
↪'t': 4, 'level_estimates': 742.96590909090912, 'forecast_error': -429.
↪93181818181824}, {'squared_error': 3338053.5693440083, 'alpha': 0.5, 'demand': 2570,
↪ 'one_step_forecast': 742.96590909090912, 't': 5, 'level_estimates': 1656.
↪4829545454545, 'forecast_error': 1827.034090909091}, {'squared_error': 194025.
↪23324509294, 'alpha': 0.5, 'demand': 1216, 'one_step_forecast': 1656.4829545454545,
↪'t': 6, 'level_estimates': 1436.2414772727273, 'forecast_error': -440.4829545454545}
↪, {'squared_error': 112386.84808400051, 'alpha': 0.5, 'demand': 1101, 'one_step_
↪forecast': 1436.2414772727273, 't': 7, 'level_estimates': 1268.6207386363635,
↪'forecast_error': -335.24147727272725}, {'squared_error': 2209323.3086119094, 'alpha
↪': 0.5, 'demand': 2755, 'one_step_forecast': 1268.6207386363635, 't': 8, 'level_
↪estimates': 2011.8103693181818, 'forecast_error': 1486.3792613636365}, {'squared_
↪error': 712656.13255070464, 'alpha': 0.5, 'demand': 2856, 'one_step_forecast': 2011.
↪8103693181818, 't': 9, 'level_estimates': 2433.905184659091, 'forecast_error': 844.
↪18963068181824}, {'squared_error': 2798.9585638125168, 'alpha': 0.5, 'demand': 2381,
↪ 'one_step_forecast': 2433.905184659091, 't': 10, 'level_estimates': 2407.
↪4525923295455, 'forecast_error': -52.905184659090992}, {'squared_error': 292089.
↪0045557259, 'alpha': 0.5, 'demand': 1867, 'one_step_forecast': 2407.4525923295455,
↪'t': 11, 'level_estimates': 2137.226296164773, 'forecast_error': -540.4525923295455}
↪], 'mape': 100.69830747447692, 'forecast': [2137.226296164773, 2137.226296164773,␣
↪2137.226296164773, 2137.226296164773, 2137.226296164773]}
```

```
print(ses_df.get('forecast', 'UNKNOWN'))
```

```
[2137.226296164773, 2137.226296164773, 2137.226296164773, 2137.226296164773, 2137.
↪226296164773]
```

If we check the statistcs for the forecast we can see whether there is a linear trend and subsequently if the forecast is useful.

```
print(ses_df.get('statistics', 'UNKNOWN'),'\n mape: {}'.format(ses_df.get('mape',
↪'UNKNOWN')))
```

```
{'pvalue': 0.0047852515832242743, 'test_statistic': 3.8634855288615153, 'std_residuals
↪': 4793.7283216530095, 'intercept': 377.59999999999991, 'trend': True, 'slope': 224.
↪4909090909091, 'slope_standard_error': 58.105797838218294}
 mape: 100.69830747447692
```

The breakdown of the forecast is also returned with the `forecast` and `statistics`.

```
print(ses_df.get('forecast_breakdown', 'UNKNOWN'))
```

```
[{'squared_error': 2345353.024793389, 'alpha': 0.5, 'demand': 104, 'one_step_forecast
↪': 1635.4545454545455, 't': 1, 'level_estimates': 869.72727272727275, 'forecast_
↪error': -1531.4545454545455}, {'squared_error': 1938423.3471074379, 'alpha': 0.5,
↪'demand': 2262, 'one_step_forecast': 869.72727272727275, 't': 2, 'level_estimates':␣
↪1565.8636363636365, 'forecast_error': 1392.2727272727273}, {'squared_error':␣
↪1478324.3822314052, 'alpha': 0.5, 'demand': 350, 'one_step_forecast': 1565.
↪8636363636365, 't': 3, 'level_estimates': 957.93181818181824, 'forecast_error': -
↪1215.8636363636365}, {'squared_error': 184841.36828512402, 'alpha': 0.5, 'demand':␣
↪528, 'one_step_forecast': 957.93181818181824, 't': 4, 'level_estimates': 742.
↪96590909090912, 'forecast_error': -429.93181818181824}, {'squared_error': 3338053.
↪5693440083, 'alpha': 0.5, 'demand': 2570, 'one_step_forecast': 742.96590909090912,
↪'t': 5, 'level_estimates': 1656.4829545454545, 'forecast_error': 1827.034090909091},
↪ {'squared_error': 194025.23324509294, 'alpha': 0.5, 'demand': 1216, 'one_step_
↪forecast': 1656.4829545454545, 't': 6, 'level_estimates': 1436.2414772727273,
↪'forecast_error': -440.4829545454545}, {'squared_error': 112386.84808400051, 'alpha
↪': 0.5, 'demand': 1101, 'one_step_forecast': 1436.2414772727273, 't': 7, 'level_
↪estimates': 1268.6207386363635, 'forecast_error': -335.24147727272725}, {'squared_
↪error': 2209323.3086119094, 'alpha': 0.5, 'demand': 2755, 'one_step_forecast': 1268.
↪6207386363635, 't': 8, 'level_estimates': 2011.8103693181818, 'forecast_error':␣
↪1486.3792613636365}, {'squared_error': 712656.13255070464, 'alpha': 0.5, 'demand':␣
↪2856, 'one_step_forecast': 2011.8103693181818, 't': 9, 'level_estimates': 2433.
↪905184659091, 'forecast_error': 844.18963068181824}, {'squared_error': 2798.
↪9585638125168, 'alpha': 0.5, 'demand': 2381, 'one_step_forecast': 2433.905184659091,
↪ 't': 10, 'level_estimates': 2407.4525923295455, 'forecast_error': -52.
↪905184659090992}, {'squared_error': 292089.0045557259, 'alpha': 0.5, 'demand': 1867,
↪ 'one_step_forecast': 2407.4525923295455, 't': 11, 'level_estimates': 2137.
↪226296164773, 'forecast_error': -540.4525923295455}]
```

We can convert the `forecast_breakdown` back into a `DataFrame`.

```
forecast_breakdown_df = pd.DataFrame(ses_df.get('forecast_breakdown', 'UNKNOWN'))
print(forecast_breakdown_df)
```

```
    alpha  demand  forecast_error  level_estimates  one_step_forecast  0     ␣
↪0.5     104    -1531.454545       869.727273        1635.454545
```

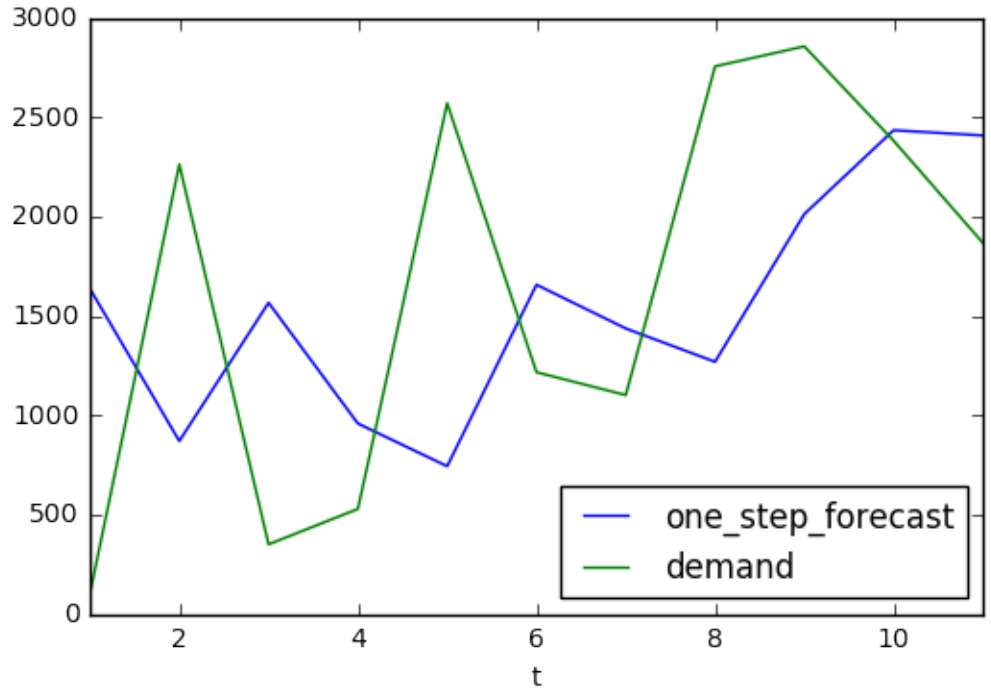
```
1      0.5    2262     1392.272727      1565.863636         869.727273
2      0.5     350    -1215.863636       957.931818        1565.863636
3      0.5     528     -429.931818       742.965909         957.931818
4      0.5    2570     1827.034091      1656.482955         742.965909
5      0.5    1216     -440.482955      1436.241477        1656.482955
6      0.5    1101     -335.241477      1268.620739        1436.241477
7      0.5    2755     1486.379261      2011.810369        1268.620739
8      0.5    2856      844.189631      2433.905185        2011.810369
9      0.5    2381      -52.905185      2407.452592        2433.905185
10     0.5    1867     -540.452592      2137.226296        2407.452592
```

```
    squared_error   t
0    2.345353e+06   1
1    1.938423e+06   2
2    1.478324e+06   3
3    1.848414e+05   4
4    3.338054e+06   5
5    1.940252e+05   6
6    1.123868e+05   7
7    2.209323e+06   8
8    7.126561e+05   9
9    2.798959e+03  10
10   2.920890e+05  11
```

Let's look at the `demand` and the `one_step_forecast` in a chart.

```
forecast_breakdown_df.plot(x='t', y=['one_step_forecast','demand'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10e1be400>
```



Using `y = mx + c` we can also create the data points for the regression line.

```
regression = {'regression': [(ses_df.get('statistics')['slope']* i ) + ses_df.get(
↪'statistics')['intercept'] for i in range(1,12)]}
print(regression)
```

```
{'regression': [602.09090909090901, 826.58181818181811, 1051.0727272727272, 1275.
↪5636363636363, 1500.0545454545454, 1724.5454545454545, 1949.0363636363636, 2173.
↪5272727272727, 2398.0181818181818, 2622.5090909090909, 2847.0]}
```

We can add the regression data points to the forecast breakdwn DataFrame.

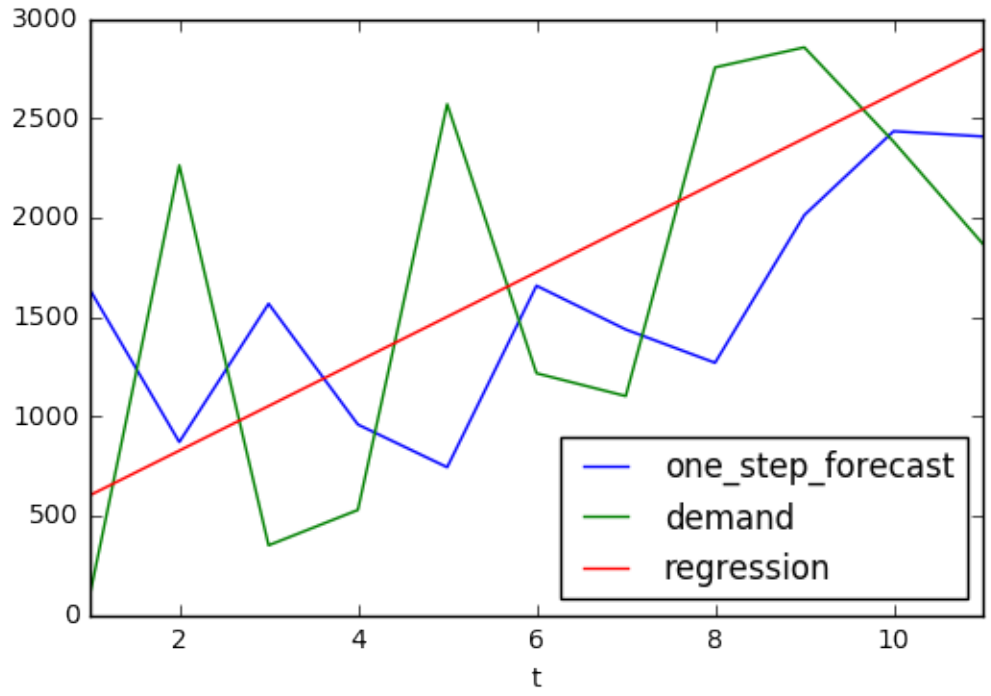
```
forecast_breakdown_df['regression'] = regression.get('regression')
print(forecast_breakdown_df)
```

```
    alpha  demand  forecast_error  level_estimates  one_step_forecast  0
↪0.5     104    -1531.454545       869.727273        1635.454545
1     0.5    2262     1392.272727      1565.863636         869.727273
2     0.5     350    -1215.863636       957.931818        1565.863636
3     0.5     528     -429.931818       742.965909         957.931818
4     0.5    2570     1827.034091      1656.482955         742.965909
5     0.5    1216     -440.482955      1436.241477        1656.482955
6     0.5    1101     -335.241477      1268.620739        1436.241477
7     0.5    2755     1486.379261      2011.810369        1268.620739
8     0.5    2856      844.189631      2433.905185        2011.810369
9     0.5    2381      -52.905185      2407.452592        2433.905185
10    0.5    1867     -540.452592      2137.226296        2407.452592

    squared_error   t   regression
0    2.345353e+06   1   602.090909
1    1.938423e+06   2   826.581818
2    1.478324e+06   3  1051.072727
3    1.848414e+05   4  1275.563636
4    3.338054e+06   5  1500.054545
5    1.940252e+05   6  1724.545455
6    1.123868e+05   7  1949.036364
7    2.209323e+06   8  2173.527273
8    7.126561e+05   9  2398.018182
9    2.798959e+03  10  2622.509091
10   2.920890e+05  11  2847.000000
```

```
forecast_breakdown_df.plot(x='t', y=['one_step_forecast','demand', 'regression'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110a83b38>
```



We have a choice now, we can use another alpha and repeat the analysis to reduce the Standard Error or use supplychainpy's `optimise=True` parameter to use an evolutionary algorithm and get closer to an optimal solution.

```
opt_ses_df = simple_exponential_smoothing_forecast(ds=row_ds[1:12], length=12,␣
↪smoothing_level_constant=0.4,optimise=True)
print(opt_ses_df)
```

```
{'statistics': {'pvalue': 0.0047852515832242743, 'test_statistic': 3.8634855288615153,
↪ 'std_residuals': 4793.7283216530095, 'intercept': 377.59999999999991, 'trend':␣
↪True, 'slope': 224.4909090909091, 'slope_standard_error': 58.105797838218294},
↪'optimal_alpha': 0.006889829296806371, 'mape': 209.37388042679993, 'standard_error
↪': 1097.3575476759161, 'forecast_breakdown': [{'squared_error': 2345353.024793389,
↪'alpha': 0.006889829296806371, 'demand': 104, 'one_step_forecast': 1635.
↪4545454545455, 't': 1, 'level_estimates': 1624.9030850605454, 'forecast_error': -
↪1531.4545454545455}, {'squared_error': 405892.47902537062, 'alpha': 0.
↪006889829296806371, 'demand': 2262, 'one_step_forecast': 1624.9030850605454, 't': 2,
↪ 'level_estimates': 1629.2925740500002, 'forecast_error': 637.09691493945456}, {
↪'squared_error': 1636589.4900194753, 'alpha': 0.006889829296806371, 'demand': 350,
↪'one_step_forecast': 1629.2925740500002, 't': 3, 'level_estimates': 1620.
↪4784665941236, 'forecast_error': -1279.2925740500002}, {'squared_error': 1193509.
↪1999718475, 'alpha': 0.006889829296806371, 'demand': 528, 'one_step_forecast': 1620.
↪4784665941236, 't': 4, 'level_estimates': 1612.9514764488533, 'forecast_error': -
↪1092.4784665941236}, {'squared_error': 915941.87643142976, 'alpha': 0.
↪006889829296806371, 'demand': 2570, 'one_step_forecast': 1612.9514764488533, 't': 5,
↪ 'level_estimates': 1619.5453774048813, 'forecast_error': 957.04852355114667}, {
↪'squared_error': 162848.87162484805, 'alpha': 0.006889829296806371, 'demand': 1216,
↪'one_step_forecast': 1619.5453774048813, 't': 6, 'level_estimates': 1616.
↪7650186410463, 'forecast_error': -403.54537740488126}, {'squared_error': 266013.
↪5544537988, 'alpha': 0.006889829296806371, 'demand': 1101, 'one_step_forecast':␣
↪1616.7650186410463, 't': 7, 'level_estimates': 1613.2114857053452, 'forecast_error
↪': -515.76501864104625}, {'squared_error': 1303681.0113751951, 'alpha': 0.
↪006889829296806371, 'demand': 2755, 'one_step_forecast': 1613.2114857053452, 't': 8,
↪ 'level_estimates': 1621.0782136618895, 'forecast_error': 1141.7885142946548}, {
↪'squared_error': 1525031.8183725097, 'alpha': 0.006889829296806371, 'demand': 2856,
↪'one_step_forecast': 1621.0782136618895, 't': 9, 'level_estimates': 1629.
↪5866139646664, 'forecast_error': 1234.9217863381105}, {'squared_error': 564638.
↪07671308529, 'alpha': 0.006889829296806371, 'demand': 2381, 'one_step_forecast':␣
↪1629.5866139646664, 't': 10, 'level_estimates': 1634.7637239257851, 'forecast_error
↪': 751.41338603533359}, {'squared_error': 53933.687924818943, 'alpha': 0.
↪006889829296806371, 'demand': 1867, 'one_step_forecast': 1634.7637239257851, 't':␣
```

```
print(opt_ses_df.get('statistics', 'UNKNOWN'),'\n mape: {}'.format(opt_ses_df.get(
↪'mape', 'UNKNOWN')))
```

```
{'pvalue': 0.0047852515832242743, 'test_statistic': 3.8634855288615153, 'std_residuals
↪': 4793.7283216530095, 'intercept': 377.59999999999991, 'trend': True, 'slope': 224.
↪4909090909091, 'slope_standard_error': 58.105797838218294}
 mape: 209.37388042679993
```

```
print(opt_ses_df.get('forecast', 'UNKNOWN'))
```

```
[1636.3637922244625, 1636.3637922244625, 1636.3637922244625, 1636.3637922244625, 1636.
↪3637922244625]
```

```
optimised_regression = {'regression': [(opt_ses_df.get('statistics')['slope']* i ) +␣
↪opt_ses_df.get('statistics')['intercept'] for i in range(1,12)]}
print(optimised_regression)
```

```
{'regression': [602.09090909090901, 826.58181818181811, 1051.0727272727272, 1275.
↪5636363636363, 1500.0545454545454, 1724.5454545454545, 1949.0363636363636, 2173.
↪5272727272727, 2398.0181818181818, 2622.5090909090909, 2847.0]}
```

```
opt_forecast_breakdown_df = pd.DataFrame(opt_ses_df.get('forecast_breakdown', 'UNKNOWN
↪'))
```

We can compare the MAPE of our previous forecast with the optimised simple exponential smoothing forecast to see which is a better forecast.

```
opt_forecast_breakdown_df['regression'] = optimised_regression.get('regression')
print(opt_forecast_breakdown_df)
```

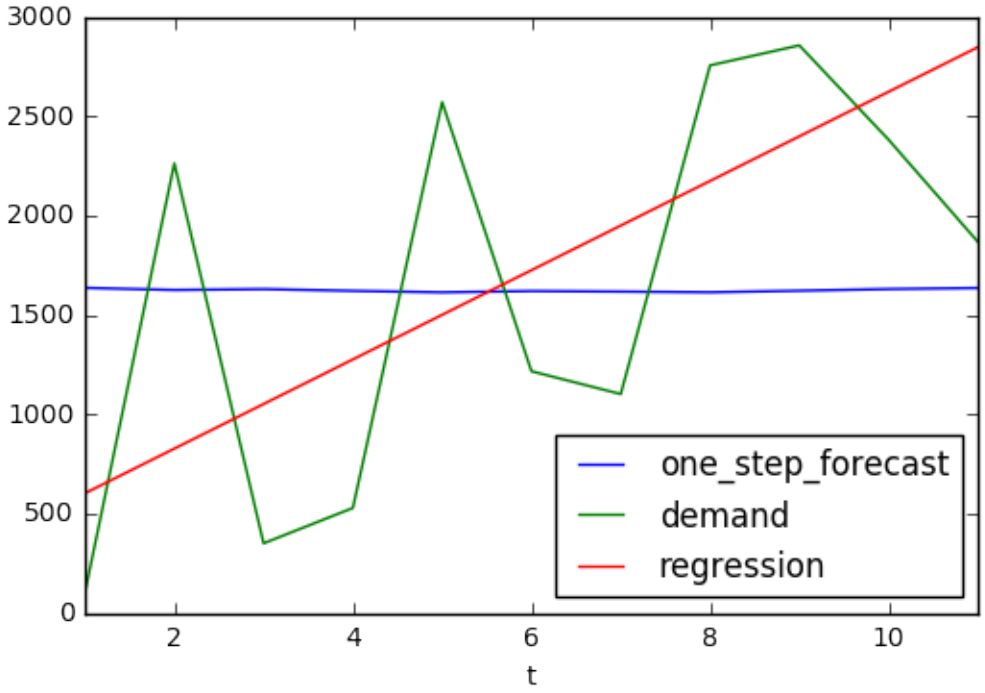
```
      alpha  demand  forecast_error  level_estimates  one_step_forecast  0  ␣
↪0.00689     104    -1531.454545      1624.903085        1635.454545
1   0.00689    2262      637.096915      1629.292574        1624.903085
2   0.00689     350    -1279.292574      1620.478467        1629.292574
3   0.00689     528    -1092.478467      1612.951476        1620.478467
4   0.00689    2570      957.048524      1619.545377        1612.951476
5   0.00689    1216     -403.545377      1616.765019        1619.545377
6   0.00689    1101     -515.765019      1613.211486        1616.765019
7   0.00689    2755     1141.788514      1621.078214        1613.211486
8   0.00689    2856     1234.921786      1629.586614        1621.078214
9   0.00689    2381      751.413386      1634.763724        1629.586614
10  0.00689    1867      232.236276      1636.363792        1634.763724

    squared_error   t   regression
0    2.345353e+06   1   602.090909
1    4.058925e+05   2   826.581818
2    1.636589e+06   3  1051.072727
3    1.193509e+06   4  1275.563636
4    9.159419e+05   5  1500.054545
5    1.628489e+05   6  1724.545455
6    2.660136e+05   7  1949.036364
7    1.303681e+06   8  2173.527273
```

```
8     1.525032e+06   9  2398.018182
9     5.646221e+05  10  2622.509091
10    5.393369e+04  11  2847.000000
```

```
opt_forecast_breakdown_df.plot(x='t', y=['one_step_forecast','demand', 'regression'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110a98f98>
```



CHAPTER 7

Analytic Hierarchy Process

As of release 0.0.4, Supplychainpy will have the facility for computing the AHP of a given set of criteria and alternative options. For an overview of the process, please visit this blog post

Below is a code snippet for the AHP:

```
>>> from supplychainpy.model_decision import analytical_hierarchy_process
>>> lorry_cost = {'scania': 55000, 'iveco': 79000, 'volvo': 59000, 'navistar': 66000}
>>> criteria = ('style', 'reliability', 'comfort', 'fuel_economy')
>>> criteria_scores = [ (1 / 1, 2 / 1, 7 / 1, 9 / 1), (1 / 2, 1 / 1, 5 / 1, 5 / 1),␣
↪(1 / 7, 1 / 5, 1 / 1, 5 / 1),(1 / 9, 1 / 5, 1 / 5, 1 / 1)]
>>> options = ('scania', 'iveco', 'navistar', 'volvo' )
>>>     option_scores = {
>>> 'style': [(1 / 1, 1 / 3, 5 / 1, 1 / 5), (3 / 1, 1 / 1, 2 / 1, 3 / 1), (1 / 3, 1 /␣
↪5, 1 / 1, 1 / 5), (5 / 1, 1 / 3, 5 / 1, 1 / 1)],
>>> 'reliability': [(1 / 1, 1 / 3, 3 / 1, 1 / 7), (3 / 1, 1 / 1, 5 / 1, 1 / 5), (1 /␣
↪3, 1 / 5, 1 / 1, 1 / 5), (7 / 1, 5 / 1, 5 / 1, 1 / 1)],
>>> 'comfort': [(1 / 1, 5 / 1, 5 / 1, 1 / 7), (1 / 5, 1 / 1, 2 / 1, 1 / 7), (1 / 3, 1␣
↪/ 5, 1 / 1, 1 / 5), (7 / 1, 7 / 1, 5 / 1, 1 / 1)],
>>> 'fuel_economy': (11, 9, 10, 12)}
>>> lorry_decision = analytical_hierarchy_process(criteria=criteria,
...                                               criteria_scores=criteria_scores,
...                                               options=options,
...                                               option_scores=option_scores,
...                                               quantitative_criteria=('fuel_economy',),
...                                               item_cost=lorry_cost)
```

The results of the AHP:

```
{'analytical_hierarchy': {'iveco': 0.20541585500041709, 'scania': 0.21539971200341132,
↪ 'volvo': 0.5677817531137912, 'navistar': 0.011402679882380324}, 'cost_benefit_
↪ratios': {'iveco': 0.67345198031782316, 'scania': 1.0143368256160643, 'volvo': 2.
↪4924656619741006, 'navistar': 0.044746880144492483}
```

CHAPTER 8

Monte Carlo Simulation

After analysing the orders, the results for safety stock may not adequately calculate the service level required. The complexity of the supply chain operation may include randomness an analytical model does not capture. A simulation is useful for giving a dynamic view of a complex process. The simulation replicates some of the complexity of the system over time.

The code below returns a transaction report covering the number of periods specified, multiplied by the number of runs requested. The higher the number of runs the more accurately the simulation captures the dynamics of the system, when summarised later. The simulation is limited by the assumptions inherent in the simulations design (detailed in the calculations).

To start we need to analyse the orders again like we did in the inventory analysis above:

```
>>> from supplychainpy.model_inventory import analyse_orders_abcxyz_from_file
>>> orders_analysis = analyse_orders_abcxyz_from_file(file_path="data.csv", z_
↪value=Decimal(1.28),
>>>                                         reorder_cost=Decimal(5000), file_type="csv
↪")
```

The orders are then passed as a parameter to the monte carlo simulation:

```
>>> from supplychainpy.model_inventory import analyse_orders_abcxyz_from_file
>>> from supplychainpy import simulate
>>> orders_analysis = analyse_orders_abcxyz_from_file(file_path="data.csv", z_
↪value=Decimal(1.28),
>>>                                         reorder_cost=Decimal(5000), file_type="csv
↪")
>>>
>>> sim = simulate.run_monte_carlo(orders_analysis=orders_analysis.orders, file_path=
↪"data.csv", z_value=Decimal(1.28), runs=100,
>>>                                reorder_cost=Decimal(4000), file_type="csv", period_
↪length=12)
>>> for transaction in sim:
>>>     print(transaction)
```

The Monte Carlo simulation generates normally distributed random demand, based on the historical data. The demand

for each SKU is then used in each period to model a probable transaction history. The output below are the sales for one SKU over the year for 100 runs (1 run shown).

```
[{'delivery': '0', 'quantity_sold': '1354', 'po_received': '', 'po_quantity': '3630',
↪'opening_stock': '1446',
'shortage_units': '0', 'closing_stock': '1355', 'revenue': '541946', 'demand': '92',
↪'index': '1', 'po_raised':
'PO 31', 'period': '1', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost': '0'}]
[{'delivery': '0', 'quantity_sold': '1354', 'po_received': '', 'po_quantity': '6268',
↪'opening_stock': '1355',
'shortage_units': '1283', 'closing_stock': '0', 'revenue': '541946', 'demand': '2638',
↪ 'index': '1', 'po_raised':
'PO 41', 'period': '2', 'backlog': '1283', 'sku_id': 'KR202-209', 'shortage_cost':
↪'154032'}]
[{'delivery': '3630', 'quantity_sold': '1520', 'po_received': 'PO 31', 'po_quantity':
↪'3464', 'opening_stock': '0',
'shortage_units': '0', 'closing_stock': '2805', 'revenue': '608381', 'demand': '826',
↪'index': '1', 'po_raised':
'PO 51', 'period': '3', 'backlog': '1283', 'sku_id': 'KR202-209', 'shortage_cost': '0
↪'}]
[{'delivery': '6269', 'quantity_sold': '7753', 'po_received': 'PO 41', 'po_quantity':
↪'0', 'opening_stock': '2805',
'shortage_units': '0', 'closing_stock': '7754', 'revenue': '3101401', 'demand': '1320
↪', 'index': '1',
'po_raised': '', 'period': '4', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost
↪': '0'}]
[{'delivery': '3464', 'quantity_sold': '10203', 'po_received': 'PO 51', 'po_quantity
↪': '0', 'opening_stock': '7754',
'shortage_units': '0', 'closing_stock': '10204', 'revenue': '4081460', 'demand': '1014
↪', 'index': '1',
'po_raised': '', 'period': '5', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost
↪': '0'}]
[{'delivery': '0', 'quantity_sold': '8926', 'po_received': '', 'po_quantity': '0',
↪'opening_stock': '10204',
'shortage_units': '0', 'closing_stock': '8927', 'revenue': '3570654', 'demand': '1277
↪', 'index': '1',
'po_raised': '','period': '6', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost
↪': '0'}]
[{'delivery': '0', 'quantity_sold': '7284', 'po_received': '', 'po_quantity': '0',
↪'opening_stock': '8927',
'shortage_units': '0', 'closing_stock': '7285', 'revenue': '2913927', 'demand': '1642
↪', 'index': '1',
'po_raised': '','period': '7', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost
↪': '0'}]
[{'delivery': '0', 'quantity_sold': '6387', 'po_received': '', 'po_quantity': '0',
↪'opening_stock': '7285',
'shortage_units': '0', 'closing_stock': '6387', 'revenue': '2554819', 'demand': '898',
↪ 'index': '1',
'po_raised': '','period': '8', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost
↪': '0'}]
[{'delivery': '0', 'quantity_sold': '4708', 'po_received': '', 'po_quantity': '276',
↪'opening_stock': '6387',
'shortage_units': '0', 'closing_stock': '4709', 'revenue': '1883461', 'demand': '1678
↪', 'index': '1', 'po_raised':
'PO 111', 'period': '9', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost': '0'}]
[{'delivery': '0', 'quantity_sold': '2954', 'po_received': '', 'po_quantity': '2030',
↪'opening_stock': '4709',
'shortage_units': '0', 'closing_stock': '2955', 'revenue': '1181806', 'demand': '1754
↪', 'index': '1', 'po_raised':
```

```
'PO 121', 'period': '10', 'backlog': '0', 'sku_id': 'KR202-209', 'shortage_cost': '0'}
↪]
[{'delivery': '276', 'quantity_sold': '674', 'po_received': 'PO 111', 'po_quantity':
↪'4310',
'opening_stock': '2955', 'shortage_units': '0', 'closing_stock': '674', 'revenue':
↪'269654', 'demand': '2557',
'index': '1', 'po_raised': 'PO 131', 'period': '11', 'backlog': '0', 'sku_id': 'KR202-
↪209', 'shortage_cost': '0'}]
[{'delivery': '2031', 'quantity_sold': '947', 'po_received': 'PO 121', 'po_quantity':
↪'4037',
'opening_stock': '674', 'shortage_units': '0', 'closing_stock': '947', 'revenue':
↪'378903', 'demand': '1757',
'index': '1', 'po_raised': 'PO 141', 'period': '12', 'backlog': '0', 'sku_id': 'KR202-
↪209', 'shortage_cost': '0'}]
```

After running the Monte Carlo simulation, the results can be passed as a parameter for summary:

```
>>> from supplychainpy.model_inventory import analyse_orders_abcxyz_from_file
>>> from supplychainpy import simulate
>>> orders_analysis = analyse_orders_abcxyz_from_file(file_path="data.csv", z_
↪value=Decimal(1.28),
>>>                                         reorder_cost=Decimal(5000), file_type="csv
↪")
>>>
>>> sim = simulate.run_monte_carlo(orders_analysis=orders_analysis.orders, runs=100,␣
↪period_length=12)
>>>
>>> sim_window = simulate.summarize_window(simulation_frame=sim, period_length=12)
>>> for r in i:
>>>         print(r)
```

The result is a transactions summary for each SKU, over every run (100) requested. It is important to note that each run will have a different randomly generated demand. Due to the randomised demand, the transaction summary for the same SKU will differ over consecutive runs. The spread of data captures the statistically probable distribution of demand the SKU can expect. However the more runs (thousands, tens of thousands), the more useful the result.

```
{'standard_deviation_backlog': 250.43961347997646, 'variance_quantity_sold': 4045303.
↪0763888955,
'total_shortage_units': 672.0, 'average_closing_stock': 3028.416748046875, 'maximum_
↪opening_stock': 6279.0,
'minimum_closing_stock': 0.0, 'maximum_shortage_units': 672.0, 'variance_backlog':␣
↪62720.0,
'average_quantity_sold': 3091.583251953125, 'minimum_backlog': 0.0, 'maximum_backlog
↪': 672.0,
'minimum_opening_stock': 0.0, 'standard_deviation_opening_stock': 2082.4554600412375,
↪'sku_id': 'KR202-230',
'standard_deviation_revenue': 2011.2938811593137, 'maximum_quantity_sold': 6278.0,
'average_opening_stock': 2994.916748046875, 'minimum_quantity_sold': 537.0, 'maximum_
↪closing_stock': 6279.0,
'stockout_percentage': 0.0833333358168602, 'variance_opening_stock': 4336620.
↪7430555625,
'variance_shortage_units': 34496.0, 'standard_deviation_closing_stock': 2096.
↪713160255569,
'average_backlog': 112.0, 'variance_closing_stock': 4396206.0763888955,
'standard_deviation_shortage_cost': 185.7309882599024, 'minimum_shortage_units': 0.0,
↪'index': '22'}
```

The *summarize_window* returns max, min, averages and standard deviations for the primary values from the transaction

summary.

The last method summarises the runs into one transaction summary for each SKU. Similar in content to the previous summary, however, this summary aggregates the simulation runs.

```
>>> from supplychainpy.model_inventory import analyse_orders_abcxyz_from_file
>>> from supplychainpy import simulate
>>>
>>> orders_analysis = analyse_orders_abcxyz_from_file(file_path="data.csv", z_
↪value=Decimal(1.28),
>>>                                         reorder_cost=Decimal(5000), file_type="csv
↪")
>>>
>>> sim = simulate.run_monte_carlo(orders_analysis=orders_analysis.orders, runs=100,␣
↪period_length=12)
>>>
>>> sim_window = simulate.summarize_window(simulation_frame=sim, period_length=12)
>>>
>>> sim_frame= simulate.summarise_frame(sim_window)
>>>
>>> for transaction_summary in sim_frame:
>>>         print(transaction_summary)
```

Below is 1 of 32 results for 32 SKUs ran 100 times.

```
{'standard_deviation_quantity_sold': '2228', 'average_backlog': '0', 'standard_
↪deviation_closing_stock': '2228',
'maximum_quantity_sold': 7901.0, 'sku_id': 'KR202-209', 'minimum_quantity_sold': 407.
↪0, 'minimum_backlog': 0.0,
'average_closing_stock': '3592', 'average_shortage_units': '0', 'variance_opening_
↪stock': '2287',
'minimum_opening_stock': 407, 'maximum_opening_stock': 7901, 'minimum_closing_stock':␣
↪407, 'service_level': '100.00',
'maximum_closing_stock': 7901, 'average_quantity_sold': '3592', 'standard_deviation_
↪backlog': '0',
'maximum_backlog': 0.0}
```

An optimisation option exists, if after running the Monte Carlo analysis, the behaviour in the transaction summary is not favourable. If most SKUs are not achieving their desired service level or have large quantities of backlog etc., then you can use:

```
>>> from supplychainpy.model_inventory import analyse_orders_abcxyz_from_file
>>> from supplychainpy import simulate
>>>
>>> orders_analysis = analyse_orders_abcxyz_from_file(file_path="data.csv", z_
↪value=Decimal(1.28),
>>>                                         reorder_cost=Decimal(5000), file_type="csv
↪")
>>>
>>> sim = simulate.run_monte_carlo(orders_analysis=orders_analysis.orders, runs=100,␣
↪period_length=12)
>>>
>>> sim_window = simulate.summarize_window(simulation_frame=sim, period_length=12)
>>>
>>> sim_frame= simulate.summarise_frame(sim_window)
>>>
>>> optimised_orders = simulate.optimise_service_level(service_level=95.0, frame_
↪summary=sim_frame,
>>>                                                    orders_analysis=orders_analysis.orders,
↪ runs=100, percentage_increase=1.30)
```

The *optimise_service_level* methods take a value for the desired service level, the transaction summary of the Monte Carlo simulation and the original orders analysis. The service level achieved in the Monte Carlo analysis is reviewed and compared with the desired service level. If below a threshold, then the safety stock is increased, and the full Monte Carlo simulation runs again. The supplied variable percentage_increase specifies the growth in safety stock.

This optimisation step will take as long, if not longer, than the first Monte Carlo simulation because the optimisation step runs the simulation again to simulate transactions based on the new safety stock values. Please take this into consideration and adjust your expectation for this optimisation step. This feature is in development as is the whole library but this feature will change in the next release.

For further details on the implementation, please view the deep-dive blog posts for each release.

CHAPTER 9

Supplychainpy with Docker

The docker image for supplychainpy uses the continuumio/anaconda3 image, with a pre-installed version of supplychainpy and all the dependencies.

```
docker run -ti -v directory/on/client:directory/in/container --name fruit-smoothie -
↪p5000:5000 supplychainpy/suchpy bash
```

The port, container name and directories can be changed as needed. Use a shared volume (as shown above) to present a CSV to the container for generating the report.

Make sure you specify the host as "0.0.0.0" for the reporting instance running in the container.

```
supplychainpy data.csv -a -loc / -lx --host 0.0.0.0
```

CHAPTER 10

Formulas and Equations

The formal expression of the formulas and equations used in the library are detailed here.

10.1 Lead-time Demand

$$LD = LT \times D$$

where: LD = Lead-time Demand

LT = Lead-time

D = Demand

10.2 Standard Deviation of Lead-time demand

$$\sigma_{LTD} = \sqrt{LT \times \sigma_D^2 + D^2 \times \sigma_{LT}^2}$$

where: sigma_{LTD} = Standard deviation of lead-time demand

LT = Lead-time

D = Demand

10.3 Reorder Level

The formula used for calculating the reorder level is:

$$RL = LT \times D + Z \times \sigma \times \sqrt{LT}$$

where: Z = service level

LT = Lead-time

D = Demand

10.4 Safety Stock

The formula used for safety stock is:

$$SS = Z \times \sigma \times \sqrt{LT}$$

where: SS = Safety Stock

Z = service level

LT = Lead-time

10.5 Economic Order Quantity (eoq)

The economic order quantity is calculated using:

$$eoq_0 = \sqrt{\frac{2 \times R \times D}{HC}}$$

where: R = Reorder Cost

D = Demand

HC = Holding Cost

CHAPTER 11

Indices and tables

- genindex
- modindex
- search