Datmo Documentation

Release 0.0.41-dev

Anand Sampat

Nov 29, 2019

Contents

Datmo is an open source model tracking and reproducibility tool for developers.

CHAPTER 1

Features

- One command environment setup (languages, frameworks, packages, etc)
- Tracking and logging for model config and results
- Project versioning (model state tracking)
- Experiment reproducibility (re-run tasks)
- Visualize + export experiment history

CHAPTER 2

Table of contents

2.1 Quickstart

2.1.1 Hello World

Setup:

- **docker (installed and running before starting)** Instructions for Ubuntu, MacOS, Windows
- **datmo** install with `$ pip install datmo`

Steps:

1. Clone this github project.

```
$ git clone https://github.com/datmo/hello-world.git
```

2. Move into the project, initialize it, and setup the environment using the datmo CLI,

```
$ cd hello-world
$ # Initialize the project using datmo
$ datmo init
$ # Set the name and description for the project
$ # Enter `y` to setup the environment
$ # Select `cpu`, `data-analytics`, `py27` based on the questions being asked
```

3. Now, run and view your first experiment using the following commands,

```
$ datmo run 'python script.py'
$ # check for your first run using ls command
$ datmo ls
```

4. Now let's change the environment and script for a new run,

To edit the environment,

```
$ vi datmo_environment/Dockerfile
```

Add the following line into this *Dockerfile*

```
RUN pip install catboost
```

To edit the script,

```
$ vi script.py
```

Uncomment the following lines in the script,

```
# import catboost
# print catboost.__version__
```

5. Now that we have updated the environment and script, let's run another experiment,

```
$ datmo run 'python script.py'
$ # check for your first run using ls command
$ datmo ls
```

6. With two test being tracked, we can now rerun any of the previous run with reprocibility,

```
$ # Select the earlier run-id to rerun the first experiment
$ datmo rerun < run-id >
```

Now, in this hello-world example, you have run two experiments, both which are tracked, and have rerun one of these tracked experiments.

2.1.2 Spinning up a TensorFlow Jupyter Notebook

0. Install Docker on your system

Find the proper version for your operating system and install Docker from this page. Check that Docker is installed and running before moving forward.

1. Install datmo using pip:

```
$ pip install datmo
```

2. Navigate to a new folder for your project and run:

```
$ datmo init
```

3. **Create a name and description. When prompted to setup an environment, respond with the following answers:**

   - Would you like to set up an environment? : `y`
   - Please select one of the above enviornment type (e.g. 1 or gpu): `cpu`
   - Please select one of the above environments (e.g. 1 or data-analytics): `keras-tensorflow`
   - Please select one of the above environment language (e.g. py27): `py27`

4. Open a jupyter notebook automatically with:

```
$ datmo notebook
```

Congrats, you now have a functional jupyter notebook with TensorFlow!

Testing it out:

1. Navigate to the notebook by typing the following into your browser:

```
localhost:8888/?token=UNIQUE_TOKEN_FROM_TERMINAL
```

2. Click

```
New --> Notebook:  Python2
```

3. In the first cell, paste in and run:

```
import tensorflow as tf
```

4. In the second cell paste and run:

```
# Define a constant
hello = tf.constant('Hello, TensorFlow!')

# Start tf session
sess = tf.Session()

# Run the op
print(sess.run(hello))
```

If your output is `Hello, TensorFlow!`, you're good to go!

2.2 Workflows

In order to run the example workflows, make sure that you have datmo properly installed with the latest stable or development version. You can install the latest stable version with the following command:

```
$ pip install datmo
```

Listed below are actions you might want to take with Datmo. For each we have listed if there are any example for each type of flow. You can navigate to the specific flow folder to find the exact instructions for each example.

2.2.1 Setting up your environment

Environment Setup Examples		
Feature	Scenario	Link
Project Environment Setup	For fresh repository	Docs
	For existing datmo project (pre-configured env)	GitHub
	For existing datmo project (bring your own env)	GitHub
Opening a Workspace	Opening a Jupyter Notebook	GitHub
	Opening RStudio	Github
	Opening JupyterLab	Github
	Opening a Terminal	Github

2.2.2 Running an experiment

Experiment Run Examples		
Method	Example	Link
CLI	Run a single experiment	Coming Soon
CLI + Python	Run a single experiment	Coming Soon
	Run two tasks and compare results	Coming Soon

2.2.3 Creating a Snapshot

Note: All of the following flows involve using the CLI to some extent, even in conjunction with the python SDK. The standalone CLI version, while the most manual method, is compatible with any language and files, even those not listed here.

Snapshot Create Examples		
Method	Example	Link
CLI	Iris dataset sk-learn classifier	GitHub
CLI + Python	Iris dataset sk-learn classifier	GitHub
CLI + Jupyter Notebook	Iris dataset sk-learn classifier	GitHub
CLI + R	Iris dataset caret decision tree	GitHub

You can view the latest examples on the master branch on Github

2.3 Tutorials

We keep a curated list of formally supported tutorials available on a secondary repository located here.

For ease of access, we've included them here in the documentation as well.

Python Tutorials		
Project	Tags	Datmo Features Used
Kaggle Titanic Survivor Prediction (CLI '_ / 'SDK in Jupyter Notebook)	AutoML, TPOT, SVM	`notebook`, `snapshot create`, `snapshot ls`
Face Recognition (CLI in Jupyter Notebook)	CV, dlib, face_recognition	`notebook`, `snapshot create`, `snapshot ls`
Keras Fashion MNIST (CLI in Jupyter Notebook)	CV, keras, tensorflow	`notebook`, `snapshot create`, `snapshot ls`
Kaggle Jigsaw Toxic Comment Identification (CLI in Jupyter Notebook)	NLP, capsule net, Keras	`notebook`, `snapshot create`, `snapshot ls`, `environment setup`

2.4 Datmo Concepts

2.4.1 Environments

Environments contain the hardware and software necessary for running code. These involve everything from programming languages, language-level packages/libraries, operating systems, and GPU drivers. Users can store multiple environments and choose which to use at the time of a task run.

You can setup environments on initialization with `$ datmo init` or at any point in time using `$ datmo environment setup`

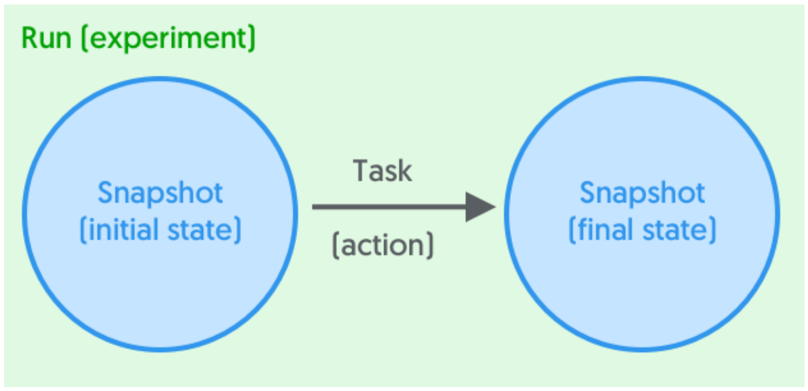
2.4.2 Workspaces

Workspaces are interactive programming environments/IDE's. Depending on which environment is chosen during setup, there

- Jupyter Notebook via `$ datmo notebook`
- RStudio via `$ datmo rstudio`
- JupyterLab via `$ datmo jupyterlab`
- Terminal via `$ datmo terminal`

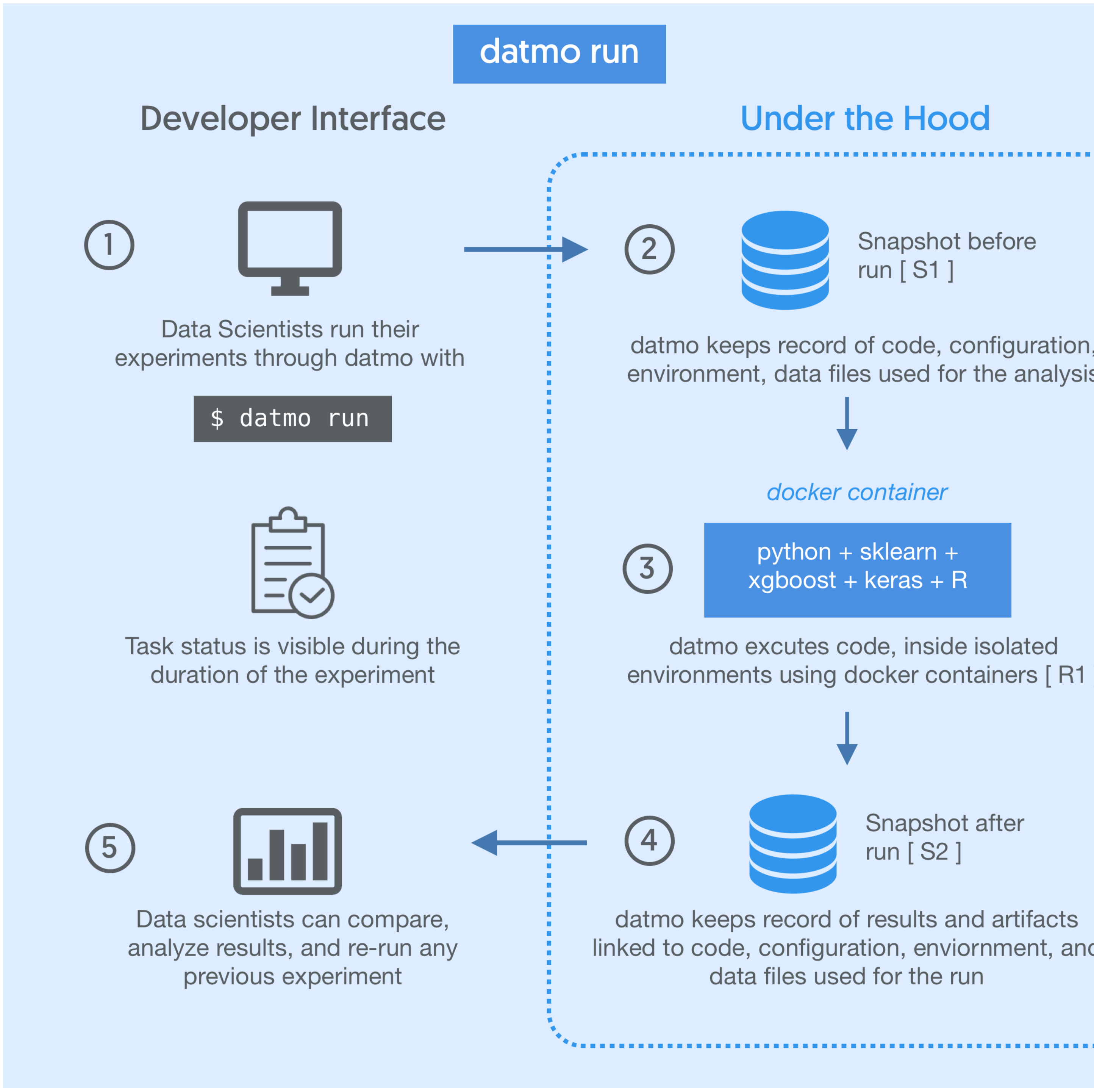
2.4.3 Runs

A **run** is comprised of *tasks* and *snapshots*. In Datmo's paradigm, states (snapshots) represent nodes, and tasks represent edges, in this case actions that are applied to states.



Each run contains the initial state (snapshot), followed by the action that was performed to it (task), as well as the final state of the repository (another snapshot).

For a typical use case, this would appear as follows:


datmo run
Developer Interface
Under the Hood
1
Data Scientists run their experiments through datmo with
$ datmo run
2
Snapshot before run [S1]
datmo keeps record of code, configuration, environment, data files used for the analysis
docker container
3
python + sklearn + xgboost + keras + R
datmo excutes code, inside isolated environments using docker containers [R1
Task status is visible during the duration of the experiment
4
Snapshot after run [S2]
datmo keeps record of results and artifacts linked to code, configuration, enviornment, and data files used for the run
5
Data scientists can compare, analyze results, and re-run any previous experiment

You can view all of your past runs at any time with `$ datmo run ls`

You can replicate a run at any time with the `$ datmo rerun` command.

2.4.4 Tasks

Tasks are command line actions or commands a user can perform within a project. For example, the commands `python train.py` or `python predict.py` would both be examples of tasks. Tasks are declarative entities and are applied to states (snapshots) to form execute runs.

2.4.5 Snapshots

For recording state, we have our own fundamental unit called a **Snapshot**. This enables the user to have a single point of reference for the model version, rather than having to worry about individually tracking each component. Snapshots contain five components, each of which is logged at the time of Snapshot creation simultaneously. When performing runs, snapshots are autogenerated to help you autosave your work.

- **Source code** is managed between snapshot versions automagically inside of a hidden `.datmo` folder that the user never has to interact with. Users can
- **Environment** (dependencies, packages, libraries, system env) are stored in environment files (typically Dockerfiles) for containerized task running and reproducibility on other systems. Datmo also currently autogenerates a *requirements.txt* file based on the packages imported by Python scripts in the repository.
- **Files** include visualizations, model weights files, datasets, and any other files present at the time of snapshot creation. For versioning models, large datasets or weights files are recommended to be stored as pointers to external sources in the _config_ property.
- **Configurations** are properties which alter your experiments (such as variable hyperparameters). Configurations are user defined, which can include (but are not limited to) algorithm type, framework, hyperparameters, external file locations, database queries, and more.
- **Metrics** are the values that help you assess your model (e.g. validation accuracy, training time, loss function score). These can be passed in from a memory-level variable/object in the Python SDK, or manually as a file or value via the CLI for all other languages.

2.5 Setting Up Your Environment

In Datmo, there are three ways to setup an environment:

1. *Using a Default Environment*
2. *Adding to a Default Environment*
3. *Bringing Your Own Environment*

2.5.1 Using a Default Environment

Default Datmo environments are maintained by the Datmo team and are guaranteed to work out of the box. We've provided many of the most popular environments for data science and AI, including:

- data-analytics (general data science/computational programming)
- kaggle
- keras/tensorflow
- mxnet
- pytorch
- caffe2
- base R and Python

You can setup a Datmo default environment either by saying yes when prompted at time of project initialization with `$ datmo init`, or at any point in time by selecting a provided environment from `$ datmo environment setup`.

2.5.2 Adding to a Default Environment

There are instances where you may want to add additional components to your environment. A common example would be installing additional language-level packages that weren't included in the default environment.

1. **To do this, first setup your default environment in one of two ways:**
    - If you haven't yet initialized a project, use `$ datmo init`
    - Otherwise, use `$ datmo environment setup`
2. **Once you've followed the prompts and selected the default environment to setup, you can find the environment file at:** `PROJECT-DIR/datmo_environment/Dockerfile`
3. **Open the Dockerfile in a text/code editor**
4. **Add the respective environment setup commands inside the Dockerfile:**

    Note: Docker has specific rules for writing command line arguments in Dockerfiles. For more details, see the RUN section of the official Docker documentation.

    Adding individual packages/libraries

    to add pandas using the pip package manager, you would add the following line to the Dockerfile: `RUN pip install pandas`

    Adding packages froma build/package list

    If you have a list of python packages you'd like to install, like a *requirements.txt* file, you can do so by appending the following to the end of your Dockerfile:

    ```
    COPY requirements.txt /tmp/requirements.txt
    RUN pip install -r /tmp/requirements.txt
    ```

2.5.3 Bringing Your Own Environment

There are two ways to write your own enviornment: with or without a Datmo base image.

Datmo base images aim to efficiently serve as a foundation for environments, including an operating system, necessary system level drivers, as well as a programming language, package, and workspaces. Datmo base images are tested and reliable, but the user has the option to bring their own as well if they would prefer.

A) With a datmo base image:

We have created public base images for all permutations of the following:

- Languages: Python 2.7, Python 3.5, R
- System Drivers: CPU, GPU (nvidia)

The datmo team maintains these images and ensures they will serve as a stable base for building environments on top of. To use one of them, you'll need to call one of them using `FROM` at the top of the Dockerfile.

1. Open a blank text file, and save it with the name `Dockerfile` in the `/datmo_environment` directory

2. Designate a base image

At the top of your Dockerfile, you will need a line of the following format:

```
FROM datmo/python-base:z
```

Note: The tag `z` is dependent on the language and system permutation (`py27`, `py35`, `r` and `cpu`, `gpu`)

Examples of valid dockerfile names would be as follows (list is not exhaustive):

i. datmo/python-base:py35-cpu
ii. datmo/python-base:py35-gpu
iii. datmo/python-base:py27-cpu
iv. datmo/python-base:py27-gpu

To see the full list of officially supported Python environment versions, check out the Dockerhub page here.

3. Designate installation of system level packages

All base datmo environments utilize Ubuntu, so the `apt-get` package tool will be used to install any necessary system dependencies.

In your Dockerfile, enumerate all system level packages with the following:

```
RUN apt-get install <package-name>
```

Note: For installing multiple system packages consecutively, read more about Docker's suggested syntax here.

---

4. Designate installation of language level packages

Most languages leverage some sort of package management tool. For example, Python utilizes pip, and is included in all python base datmo images.

To utilize your package manager to install packages through the Dockerfile, use the following line:

```
RUN pip install <python-package-name>
```

Note: For installing multiple language-level packages, follow the same guidelines listed above in the step 3 note.

B) Without a datmo base image:

1. Open a blank text file, and save it with the name `Dockerfile` in the `/datmo_environment` directory

2. Designate a base image

At the top of your Dockerfile, you will need a line of the following format:

```
FROM x/y:z
```

Where each variable represents the following Dockerhub information:

- x: user/organization account name
- y: Dockerfile name
- z: Dockerfile version

An example would be the following: `FROM kaggle/python:latest`

3. Designate installation of system level packages

Based on which operating system the base image utilizes, you will likely have a different package manager for installing system level utilities. Examples include `apt-get` for Ubuntu, `yum` for CentOS/Fedora, or `apk` on Alpine, and more.

In your Dockerfile, enumerate all system level package installations using your respective package manager with the following:

```
RUN apt-get install <package-name>
```

Note: For installing multiple system packages consecutively, read more about Docker's suggested syntax here.

4. Designate installation of language level packages

Most languages leverage some sort of package management tool. For example, Python utilizes pip, which may need to be installed as a system level resource first.

To utilize your language-level package manager to install packages through the Dockerfile, use the following line:

```
RUN pip install <python-package-name>
```

Note: For installing multiple language-level packages, follow the same guidelines listed above in the step 3 note.

5. Getting datmo workspaces to work with your custom environment

By running a fully custom environment image, you will need to add code snippets to your Dockerfile in order for some of datmo's aliases to work. Please make sure you have installed `pip` and `apt-get` during step 3.

Jupyter Notebook via `$ datmo notebook`

i. Add the following code snippet to your Dockerfile

```
# Jupyter
RUN pip --no-cache-dir install \
        ipykernel \
        jupyter \
        && \
    python -m ipykernel.kernelspec

# Set up our notebook config.
COPY jupyter_notebook_config_py2.py /root/.jupyter/
RUN mv /root/.jupyter/jupyter_notebook_config_py2.py /root/.jupyter/jupyter_
↪notebook_config.py

# Jupyter has issues with being run directly:
#   https://github.com/ipython/ipython/issues/7062
# We just add a little wrapper script.
```

(continues on next page)

(continued from previous page)

```
COPY run_jupyter.sh /
RUN chmod +x /run_jupyter.sh

# IPython
EXPOSE 8888
```

ii. Download the 3 patchfiles from here and move them into your `datmo_environment` folder along with your Dockerfile.

JupyterLab via `$ datmo jupyterlab`

i. Add the following code snippet to your Dockerfile

```
# Jupyter
   RUN pip --no-cache-dir install \
           ipykernel \
           jupyter \
           && \
       python -m ipykernel.kernelspec

   # Set up our notebook config.
   COPY jupyter_notebook_config_py2.py /root/.jupyter/
   RUN mv /root/.jupyter/jupyter_notebook_config_py2.py /root/.jupyter/
↪jupyter_notebook_config.py

   # Jupyter has issues with being run directly:
   #   https://github.com/ipython/ipython/issues/7062
   # We just add a little wrapper script.

   COPY run_jupyter.sh /
   RUN chmod +x /run_jupyter.sh

   # Jupyter lab
   RUN pip install jupyterlab==0.32.1

   # IPython
   EXPOSE 8888
```

ii. Download the 3 patchfiles from here and move them into your `datmo_environment` folder along with your Dockerfile.

RStudio via `$ datmo rstudio`

i. Add the following code snippet to your Dockerfile

```
# Rstudio
ENV DEBIAN_FRONTEND noninteractive
ENV CRAN_URL https://cloud.r-project.org/

RUN set -e \
      && ln -sf /bin/bash /bin/sh

RUN set -e \
      && apt-get -y update \
      && apt-get -y dist-upgrade \
      && apt-get -y install apt-transport-https gdebi-core libapparmor1␣
↪libcurl4-openssl-dev \
```

(continues on next page)

(continued from previous page)

```
                              libssl-dev libxml2-dev pandoc r-base \
      && apt-get -y autoremove \
      && apt-get clean

RUN set -e \
      && R -e "\
      update.packages(ask = FALSE, repos = '${CRAN_URL}'); \
      pkgs <- c('dbplyr', 'devtools', 'docopt', 'doParallel', 'foreach',
↪'gridExtra', 'rmarkdown', 'tidyverse'); \
      install.packages(pkgs = pkgs, dependencies = TRUE, repos = '${CRAN_URL}
↪'); \
      sapply(pkgs, require, character.only = TRUE);"

RUN set -e \
      && curl -sS https://s3.amazonaws.com/rstudio-server/current.ver \
        | xargs -I {} curl -sS http://download2.rstudio.org/rstudio-server-{}
↪-amd64.deb -o /tmp/rstudio.deb \
      && gdebi -n /tmp/rstudio.deb \
      && rm -rf /tmp/rstudio.deb

RUN set -e \
      && useradd -m -d /home rstudio \
      && echo rstudio:rstudio \
        | chpasswd

# expose for rstudio
EXPOSE 8787
```

ii. Download the 3 patchfiles from here and move them into your `datmo_environment` folder along with your Dockerfile.

2.6 Visualizing Snapshots

In Datmo, *Snapshots* are used as the primary record of state for a project, recording all components of a project state, including files, model weights, environment, configuration, stats, and other metadata. There are three primary ways to assess Snapshots using the CLI:

1. *View all Snapshots within a project*
2. *Inspect a single Snapshot*
3. *Compare two Snapshots*

2.6.1 View all Snapshots within a project

If you'd like to see a broad overview of all snapshots in the current datmo project, the user can do so with:

```
$ datmo snapshot ls
```

This will return the metadata for all snapshots in a table that resembles the following format:

```
+---------+------------+-------------------------------------------+--------
↪---------+--------------+-------+
|    id   |  created at |                  config                   |  ␣
↪stats          |    message    | label |
```

(continues on next page)

(continued from previous page)

```
+---------+-------------+--------------------------------------------+--------
↪---------+---------------+-------+
| 30f8366b|  2018-05-16 | {u'selected features': [u'Sex', u'Pclass',|{u
↪'accuracy':    |   auto-ml-2   |  None |
|         |   03:04:06  |  u'Age', u'Fare', u'Embarked', u'Title',   |   0.
↪8295964}    |               |        |
|         |             |   u'FarePerPerson', u'FamilySize']}        |       ␣
↪          |               |        |
| adf76fa7|  2018-05-16 | {u'selected features': [u'Sex', u'Pclass',|{u
↪'accuracy':    |   auto-ml-1   |  None |
|         |   01:24:53  |  u'Age', u'Fare', u'Embarked',             |   0.
↪8206278}    |               |        |
|         |             |   u'Fare', u'IsAlone', u'Title']}          |       ␣
↪          |               |        |
| 30803662|  2018-05-15 | {u'features analyzed': [u'Sex',            |    {} ␣
↪          |      EDA      |  None |
|         |   23:15:44  |  u'Pclass', u'FamilySize', u'IsAlone',     |       ␣
↪          |               |        |
|         |             |  u'Embarked', u'Fare', u'Age', u'Title']} |       ␣
↪          |               |        |
+---------+-------------+--------------------------------------------+--------
↪---------+---------------+-------+
```

2.6.2 Inspect a single Snapshot

If you'd like to see a detailed view of all properties pertaining to a specific snapshot, use:

```
$ datmo snapshot inspect <SNAPSHOT_ID>
```

This will return a detailed view of the snapshot that resembles the following:

```
Date: Tue Jul 17 15:38:04 2018 -0700
Session      -> a2084eeaf6a7c66509972ea4f8ca35027721e34e
Visible      -> True
Code         -> 1dadd5bbc73822ed90d9061c9003fc2556b9d40b
Environment  -> 47cc3cee2043f4e9026997e01c53918bad74f28a
Files        -> 155cc40f0d762712cd115e8262d1e8033aba727c
Config       -> {u'selected features': [u'Sex', u'Pclass', u'Age', u'Fare', u
↪'Embarked', u'FarePerPerson', u'FamilySize', u'Title']}
Stats        -> {u'accuracy': 0.8161434977578476}
```

2.6.3 Compare two Snapshots

There will often be times where you want to see the difference between two snapshots.

This is possible with the following command:

```
$ datmo snapshot diff <SNAPSHOT_ID_1> <SNAPSHOT_ID_2>
```

Resulting in an output resembling the following:

```
Attributes          Snapshot 1                                     Snapshot 2

id                  9d8e06ae0c8546465c1b0c200f1e84a33c049067  -> ␣
↪5e476e78b8b480506e117fdf8478c45d28020165
```

(continues on next page)

(continued from previous page)

```
created_at           Tue Jul 17 15:38:04 2018 -0700          ->  Tue Jul 17␣
↪15:30:08 2018 -0700
message              auto-ml-2                               ->  auto-ml-1
label                N/A                                     ->  N/A
code_id              1dadd5bbc73822ed90d9061c9003fc2556b9d40b  -> ␣
↪c78873227313f64c3362ee9b30432053036eef68
environment_id       47cc3cee2043f4e9026997e01c53918bad74f28a  -> ␣
↪47cc3cee2043f4e9026997e01c53918bad74f28a
file_collection_id   155cc40f0d762712cd115e8262d1e8033aba727c  -> ␣
↪155cc40f0d762712cd115e8262d1e8033aba727c
```

2.7 Command Line Utility

The command line utility for datmo is to be used in tandem with the SDK and will typically be your first contact with the datmo system. If using Python, see *Python SDK*.

If you are working within a repository already, you will want to run the `datmo init` within your repository in order to create your datmo project.

From there, you can create snapshots or run tasks using either the SDK or the CLI. At any given point you can find out more about all of your snapshots using the `datmo snapshot ls` command and see the status of any of your tasks with the `datmo task ls` command.

Sessions are a way for you to group together tasks and snapshots, but are completely optional. For example, if you want to run a set of hyperparameter experiments modifying some subset of hyperparameters you might want to do them in a designated session. Then you might try another set of hyperparameter sweeps which you would like to group into another session. By default, you will always be in the "default" session unless otherwise specified.

You can delve through more of the commands and each of their parameters below to learn more about each entity and how you can create different versions of them. You can also look through the Getting Started section in the README.

```
usage: datmo [-h]
             {init,version,status,cleanup,dashboard,configure,notebook,jupyterlab,
↪terminal,rstudio,run,ls,stop,delete,rerun,environment,snapshot}
             ...
```

2.7.1 commands

command	Possible choices: init, version, status, cleanup, dashboard, configure, notebook, jupyterlab, terminal, rstudio, run, ls, stop, delete, rerun, environment, snapshot

2.7.2 Sub-commands:

init

initialize project

```
datmo init [-h] [--name NAME] [--description DESCRIPTION] [--force]
```

Named Arguments

--name

--description

--force, -f, --no-prompt boolean if you want to run init without prompts

Default: False

version

datmo version

```
datmo version [-h]
```

status

project status

```
datmo status [-h]
```

cleanup

remove project

```
datmo cleanup [-h]
```

dashboard

start dashboard

```
datmo dashboard [-h]
```

configure

configure datmo

```
datmo configure [-h]
```

notebook

To run jupyter notebook

```
datmo notebook [-h] [--gpu] [--environment-id ENVIRONMENT_ID]
               [--environment-paths ENVIRONMENT_PATHS] [--mem-limit MEM_LIMIT]
               [--data DATA]
```

Named Arguments

--gpu	boolean if you want to run using GPUs Default: False
--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--mem-limit, -m	maximum amount of memory the notebook environment can use (these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes)
--data	list of absolute or relative filepath and/or dirpaths for data; can specify destination names with '>' (e.g. /path/to/dir, /path/to/dir>newdir, /path/to/file)

jupyterlab

To run jupyterlab

```
datmo jupyterlab [-h] [--gpu] [--environment-id ENVIRONMENT_ID]
                 [--environment-paths ENVIRONMENT_PATHS]
                 [--mem-limit MEM_LIMIT] [--data DATA]
```

Named Arguments

--gpu	boolean if you want to run using GPUs Default: False
--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--mem-limit, -m	maximum amount of memory the jupyterlab environment can use (these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes)
--data	list of absolute or relative filepath and/or dirpaths for data; can specify destination names with '>' (e.g. /path/to/dir, /path/to/dir>newdir, /path/to/file)

terminal

To run terminal

```
datmo terminal [-h] [--gpu] [--environment-id ENVIRONMENT_ID]
               [--environment-paths ENVIRONMENT_PATHS] [--mem-limit MEM_LIMIT]
               [--data DATA] [--ports PORTS]
```

Named Arguments

--gpu	boolean if you want to run using GPUs Default: False
--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--mem-limit, -m	maximum amount of memory the terminal environment can use (these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes)
--data	list of absolute or relative filepath and/or dirpaths for data; can specify destination names with '>' (e.g. /path/to/dir, /path/to/dir>newdir, /path/to/file)
--ports, -p	network port mapping during run (e.g. 8888:8888). Left is the host machine port and right is the environment port available during a run.

rstudio

To run Rstudio workspace

```
datmo rstudio [-h] [--environment-id ENVIRONMENT_ID]
              [--environment-paths ENVIRONMENT_PATHS] [--mem-limit MEM_LIMIT]
              [--data DATA]
```

Named Arguments

--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--mem-limit, -m	maximum amount of memory the rstudio environment can use (these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes)
--data	list of absolute or relative filepath and/or dirpaths for data; can specify destination names with '>' (e.g. /path/to/dir, /path/to/dir>newdir, /path/to/file)

run

run module

```
datmo run [-h] [--gpu] [--ports PORTS] [--environment-id ENVIRONMENT_ID]
          [--environment-paths ENVIRONMENT_PATHS] [--mem-limit MEM_LIMIT]
          [--interactive] [--data DATA]
          [cmd]
```

Positional Arguments

cmd	command to run within environment

Named Arguments

--gpu	boolean if you want to run using GPUs Default: False
--ports, -p	network port mapping during run (e.g. 8888:8888). Left is the host machine port and right is the environment port available during a run.
--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--mem-limit, -m	maximum amount of memory the task environment can use (these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes. e.g. 4g)
--interactive	run the environment in interactive mode (keeps STDIN open) Default: False
--data	list of absolute or relative filepath and/or dirpaths for data; can specify destination names with '>' (e.g. /path/to/dir, /path/to/dir>newdir, /path/to/file)

ls

list module

```
datmo ls [-h] [--format FORMAT] [--download] [--download-path DOWNLOAD_PATH]
```

Named Arguments

--format	output format ['table', 'csv'] Default: "table"
--download	boolean is true if user would like to download. use –download-path to specify a path Default: False
--download-path	checked only if download is specified. saves output to location specified

stop

stop runs

```
datmo stop [-h] [--id ID] [--all]
```

Named Arguments

--id	run id to stop
--all, -a	stop all datmo runs Default: False

delete

delete runs

```
datmo delete [-h] id
```

Positional Arguments

id	run id to delete

rerun

To rerun an experiment

```
datmo rerun [-h] id
```

Positional Arguments

id	run id to be rerun

environment

environment module

```
datmo environment [-h] {setup,create,update,delete,ls} ...
```

subcommands

subcommand	Possible choices: setup, create, update, delete, ls

Sub-commands:

setup

setup environment adds a predefined supported environment into your project environment directory

```
datmo environment setup [-h] [--name NAME] [--type TYPE]
                        [--framework FRAMEWORK] [--language LANGUAGE]
```

Named Arguments

--name	name of environment to be used for environment (e.g. my-new-environment). if none is given, a prompt will present the supported names
--type	type of environment to be used for environment (e.g. cpu). if none is given, a prompt will present the supported type

--framework	framework (and relevant libraries) to be used for environment (e.g. data-analytics). if none is given, a prompt will present the supported names
--language	language of environment to be used for environment (e.g. py27). if none is given, a prompt will present the supported language for the name and type

create

create environment using the definition paths given, if not looks in your project environment directory, or creates a default

```
datmo environment create [-h] [--paths PATHS] [--name NAME]
                         [--description DESCRIPTION]
```

Named Arguments

--paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--name, -n	name given to the environment
--description, -d	description of environment

update

update an environment by id

```
datmo environment update [-h] [--name NAME] [--description DESCRIPTION] id
```

Positional Arguments

id	environment id to update

Named Arguments

--name	new name for the environment
--description	new description for the environment

delete

delete a environment by id

```
datmo environment delete [-h] id
```

Positional Arguments

id	id of environment to delete

ls

list environments

```
datmo environment ls [-h] [--format FORMAT] [--download]
                     [--download-path DOWNLOAD_PATH]
```

Named Arguments

--format	output format ['table', 'csv'] Default: "table"
--download	boolean is true if user would like to download. use –download-path to specify a path Default: False
--download-path	checked only if download is specified. saves output to location specified

snapshot

Datmo snapshots allow you to save the state of your model and experiments by keeping track of your source code, environment, configuration, metrics and large files.

```
datmo snapshot [-h] {create,update,delete,ls,checkout,diff,inspect} ...
```

subcommands

subcommand	Possible choices: create, update, delete, ls, checkout, diff, inspect

Sub-commands:

create

Run snapshot create any time you want to save the results of your experiments. You can then view all snapshots with the *snapshot ls* command.

```
datmo snapshot create [-h] [--message MESSAGE] [--label LABEL]
                      [--session-id SESSION_ID] [--run-id RUN_ID]
                      [--environment-id ENVIRONMENT_ID]
                      [--environment-paths ENVIRONMENT_PATHS] [--paths PATHS]
                      [--config-filename CONFIG_FILENAME]
                      [--config-filepath CONFIG_FILEPATH] [--config CONFIG]
                      [--stats-filename STATS_FILENAME]
                      [--stats-filepath STATS_FILEPATH] [--stats STATS]
```

Named Arguments

--message, -m	message to describe snapshot

--label, -l	label snapshots with a category (e.g. best)
--session-id	user given session id
--run-id	specify run id to pull information from
--environment-id	environment id from environment object
--environment-paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--paths	list of absolute or relative filepaths and/or dirpaths to collect; can specify destination names with '>' (e.g. /path/to/file>hello, /path/to/file2, /path/to/dir>newdir)
--config-filename	filename to use to search for configuration JSON
--config-filepath	absolute filepath to use to search for configuration JSON
--config, -c	provide key, value pair for the config such as key:value, (e.g. accuracy:91.1). Left is the key and right is the value for it.
--stats-filename	filename to use to search for metrics JSON
--stats-filepath	absolute filepath to use to search for metrics JSON
--stats, -s	provide key, value pair for the stats such as key:value, (e.g. accuracy:91.1). Left is the key and right is the value for it.

update

update a snapshot by id

```
datmo snapshot update [-h] [--config CONFIG] [--stats STATS]
                      [--message MESSAGE] [--label LABEL]
                      id
```

Positional Arguments

id	snapshot id to update

Named Arguments

--config, -c	provide key, value pair for the config such as key:value, (e.g. accuracy:91.1). Left is the key and right is the value for it.
--stats, -s	provide key, value pair for the stats such as key:value, (e.g. accuracy:91.1). Left is the key and right is the value for it.
--message	new message for the snapshot
--label	new label for the snapshot

delete

delete a snapshot by id

```
datmo snapshot delete [-h] id
```

Positional Arguments

id	snapshot id to delete

ls

list snapshots

```
datmo snapshot ls [-h] [--session-id SESSION_ID] [--details] [--all]
                  [--format FORMAT] [--download]
                  [--download-path DOWNLOAD_PATH]
```

Named Arguments

--session-id	session id to filter
--details	show detailed snapshot information Default: False
--all	show all visible and hidden snapshots Default: False
--format	output format ['table', 'csv'] Default: "table"
--download	boolean is true if user would like to download. use –download-path to specify a path Default: False
--download-path	checked only if download is specified. saves output to location specified

checkout

checkout a snapshot by id

```
datmo snapshot checkout [-h] id
```

Positional Arguments

id	snapshot id to checkout

diff

view diff between 2 snapshots

```
datmo snapshot diff [-h] id_1 id_2
```

Positional Arguments

id_1	snapshot id 1
id_2	snapshot id 2

inspect

inspect a snapshot by id

```
datmo snapshot inspect [-h] id
```

Positional Arguments

id	snapshot id

2.8 Python SDK

Datmo's Python SDK is a way to create datmo snapshots and run tasks directly within your code. Although the SDK is not necessary for using datmo, it helps simplify the process of integrating your current code with current Python projects. If you aren't using Python, see *Command Line Utility*.

2.8.1 datmo.snapshot module

class `datmo.snapshot.`**`Snapshot`**(*snapshot_entity*)

Snapshot is an entity object to enable user access to properties

Parameters **`snapshot_entity`** (*`datmo.core.entity.snapshot.Snapshot`*) – core snapshot entity to reference

`id`

the id of the entity

Type str

`model_id`

the parent model id for the entity

Type str

`message`

long description of snapshot

Type str

code_id
code reference associated with the snapshot

Type str

environment_id
id for environment used to create snapshot

Type str

file_collection_id
file collection associated with the snapshot

Type str

config
key, value pairs of configurations

Type dict

stats
key, value pairs of metrics and statistics

Type dict

task_id
task id associated with snapshot

Type str

label
short description of snapshot

Type str

created_at

Type datetime.datetime

Raises `InvalidArgumentType`

files

get_files(*mode='r'*)
Returns a list of file objects for the snapshot

Parameters **mode** (*str*) – file object mode (default is "r" which signifies read mode)

Returns list of file objects associated with the snapshot

Return type list

datmo.snapshot.**create**(*message*, *label=None*, *run_id=None*, *environment_id=None*, *env=None*, *paths=None*, *config=None*, *stats=None*)
Create a snapshot within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **message** (*str*) – a description of the snapshot for later reference

- **label** (*str, optional*) – a short description of the snapshot for later reference (default is None, which means a blank label is stored)
- **run_id** (*str, optional*) – run object id to use to create snapshot if run id is passed then subsequent parameters would be ignored. when using run id, it will overwrite the following inputs

  environment_id: used to run the task,

  paths: this is the set of all files saved during the task

  config: nothing is passed into this variable. the user may add something to the config by passing in a dict for the config

  stats: the task.results are added into the stats variable of the snapshot.
- **environment_id** (*str, optional*) – provide the environment object id to use with this snapshot (default is None, which means it creates a default environment)
- **env** (*str or list, optional*) – the absolute file path for the environment definition path. env is not used if environment_id is also passed. this can be either a string or list (default is None, environment_id is also not passed, which will defer to the environment to find a default environment or will fail if not found)
- **paths** (*list, optional*) – list of absolute or relative filepaths and/or dirpaths to collect with destination names (e.g. "/path/to/file>hello", "/path/to/file2", "/path/to/dir>newdir")
- **config** (*dict, optional*) – provide the dictionary of configurations (default is None, which means it is empty)
- **stats** (*dict, optional*) – provide the dictionary of relevant statistics or metrics (default is None, which means it is empty)

Returns returns a Snapshot entity as defined above

Return type *Snapshot*

Examples

You can use this function within a project repository to save snapshots for later use. Once you have created this, you will be able to view the snapshot with the *datmo snapshot ls* cli command

```
>>> import datmo
>>> datmo.snapshot.create(message="my first snapshot", paths=["/path/to/a/large/
↪file"], config={"test": 0.4, "test2": "string"}, stats={"accuracy": 0.94})
```

You can also use the result of a task run in order to create a snapshot

```
>>> datmo.snapshot.create(message="my first snapshot from task", task_id=
↪"1jfkshg049")
```

datmo.snapshot.**delete**(*snapshot_id=None*)

Delete a snapshot within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters **snapshot_id** (*str*) – snapshot id to be updated

Returns returns a Snapshot entity

Return type snapshot entity

Examples

You can use this function within a project repository to delete a snapshot.

```
>>> import datmo
>>> datmo.snapshot.delete(snapshot_id="4L24adFfsa")
```

`datmo.snapshot.`**`ls`**(*filter=None*)

List snapshots within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters **`filter`** (*`str, optional`*) – a string to use to filter from message and label (default is to give all snapshots, unless provided a specific string. eg: best)

Returns returns a list of Snapshot entities (as defined above)

Return type list

Examples

You can use this function within a project repository to list snapshots.

```
>>> import datmo
>>> snapshots = datmo.snapshot.ls()
```

`datmo.snapshot.`**`update`**(*snapshot_id=None*, *config=None*, *stats=None*, *message=None*, *label=None*)

Update a snapshot within a project

The project must be created before this is implemented. You can do that by using the following command:

```
$ datmo init
```

Parameters

- **`snapshot_id`** (*`str`*) – snapshot id to be updated
- **`config`** (*`dict, optional`*) – provide the dictionary of configurations to update (default is None, which means it is not being updated)
- **`stats`** (*`dict, optional`*) – provide the dictionary of relevant statistics or metrics to update (default is None, which means it is not being updated)
- **`message`** (*`str, optional`*) – a string to use as a new message for the snapshot (default is the already given message to that snapshot, unless provided a specific string.)
- **`label`** (*`str, optional`*) – a string to use as a new label for the snapshot (default is the already given label to that snapshot, unless provided a specific string.)

Returns returns a Snapshot entity

Return type snapshot entity

Examples

You can use this function within a project repository to update a snapshot.

```
>>> import datmo
>>> snapshots = datmo.snapshot.update(snapshot_id="4L24adFfsa", config={"depth":
↪"10", "learning_rate": "0.91"},
...           stats={"acc": "91.34", "f1_score": "0.91"}, message="new message",␣
↪label="best")
```

2.9 Frequently Asked Questions

Q: What is the role of the datmo open source tool?

A: The open source project acts as a user-controlled project manager (available as both a CLI and Python SDK) that enables users to create, run, manage, and record all aspects of their experiments.

Q: Do I have to know how to use Docker to use Datmo?

A: Not at all! However, knowledge of Docker will be helpful for understanding how the environments are created and setup.

2.9.1 Environment Questions

Q: How can I add my own environments to be used with Datmo?

A: The `environment setup` command adds in a default environment provided by datmo in the `datmo_environment` directory. You can add in your own environment by modifying these files, or adding your own files to the `datmo_environment` directory (ie: Dockerfile, requirements.txt, package.json, etc). You can run a *datmo environment create* and use the environment ID at the time you run a task or run a workspace. You can also just directly run a task or workspace and Datmo will create a new environment from `datmo_environment` and will set the most recent environment that was setup as the default for running tasks.

Check out our guide full guide on *Bringing Your Own Environment* here.

Q: How does Datmo handle all of my different environments?

A: The default environment that will be used for running tasks at any given time is chosen by the Dockerfile that is present in the `datmo_environment` directory. The other environments locally available for your project, visible with `$ datmo environment ls` and can be selected by passing the environment ID in as a parameter at the time of a task run or workspace creation.

Q: I've made changes to the Dockerfile in my project, but the container environment isn't changing too. Why is this?

A: When running a task, Datmo always looks first inside the `datmo_environment` directory. If an environment is not present there, it will then use a Dockerfile from the project's root directory (if present). However, after the first run, Datmo creates an environment entity and Dockerfile that are replicas of the one used at the time of the initial run. Because of the priority of environment directories, Datmo will utilize the Dockerfile from the `datmo_environment` for subsequent runs, which means that changes to the original Dockerfile outside of `datmo_environment` will not appear in the environment Datmo has created/tracked. If you would like to change the environment, you can change it in the `datmo_environment` folder.

Q: Why does my environment have a different ID on different operating systems?

A: Environment IDs are unique hashes based on the content of the file(s). Due to differences in line separator characters on Windows and Linux/MacOS, this will cause the hashes to be different, despite the visible contents of the environment being the same. Windows utilizes `\r\n`, while Linux/MacOS use `\n`.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index