Introduction to Python

for Cloud Computing

Gregor von Laszewski, laszewski@gmail.com

2022-06-07 21:38:01.720523

Contents

1 PREFACE

Tue 07 Jun 2022 09:37:54 PM EDT

1.0.1 Contributors

This book has recieved contributions by many people. As this book is part of a larger collection of material, you als may find outhors that have not directly contributed to this document. As the book is part of a larger collection of information, it is too difficult to distinguish individual contributions and to check the contribution to this volume. Contributors are sorted by the first letter of their combined Firstname and Lastname and if not available by their github ID. Please, note that the authors are identified through git logs in addition to some contributors added by hand. The git repository from which this document is derived contains more than the documents included in this document. Thus not everyone in this list may have directly contributed to this document. However if you find someone missing that has contributed (they may not have used this particular git) please let us know. We will add you. The contributors that we are aware of include:

> Anand Sriramulu, Andrew Goldfarb, Ankita Rajendra Alshi, Anthony Duer, Arnav, Ashok Reddy Singam, Averill Cate, Jr, Ben Tracy, Bertolt Sobolik, Bo Feng, Brad Pope, Brijesh, Dave DeMeulenaere, David Drummond, De'Angelo Rutledge, Eliyah Ben Zayin, Eric Bower, Fugang Wang, Geoffrey C. Fox, Gerald Manipon, Gregor von Laszewski, Hyungro Lee, Ian Sims, IzoldaIU, J.P, Javier Diaz, Jeevan Reddy Rachepalli, Jonathan Beckford, Jonathan Branam, Josh Goodman, Juliette Zerick, Keith Hickman, Keli Fine, Kenneth Jones, Lamcloud, Mallik Challa, Mani Kagita, Miao Jiang, Mihir Shanishchara, Min Chen, Murali Cheruvu, Orly Esteban, Pulasthi Supun, Pulasthi Supun Wickramasinghe, Pulkit Maloo, Qianqian Tang, Rahul, Ravinder Lambadi, RhondaFischer, Richa Rastogi, Ritesh Tandon, Robert Knuuti, Saber Sheybani, Sachith Withana, Sandeep Kumar Khandelwal, Sheri Sanders, Shivani Katukota, Silvia Karim, Swarnima H. Sowani, Tharak Vangalapat, Tim Whitson, Tyler Balson, Vafa Andalibi, Vibhatha Abeykoon, Vineet Barshikar, XinGu, Yu Luo, ahilgenkamp, aralshi, azebrowski, bfeng, bkegerreis, brandonfischer99, btpope, garbeandy, harshadpitkar, himanshu3jul, hrbahramian, isims1, janumudvari, jessicazhu44, joshish-iu, juaco77, karankotz, keithhickman08, kkp, mallik3006, manjunathsivan, niranda perera, prateekazam, qianqian tang, rajni-cs, rirasto, sahancha, shilpasingh21, swsachith, tafaltens, toshreyanjain, trawat87, tvangalapat, varunjoshi01, vineetb-gh, wang542, xianghang mi, zhengyili4321

2 INTRODUCTION

2.0.1 Introduction to Python

Learning Objectives

- Learn quickly Python under the assumption you know a programming language
- Work with modules
- Understand docopts and cmd
- Conduct some Python examples to refresh your Python knowledge
- Learn about the `map` function in Python
- Learn how to start subprocesses and redirect their output
- Learn more advanced constructs such as multiprocessing and Queues
- Understand why we do not use `anaconda`
- Get familiar with `venv`

Python is an easy-to-learn programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's simple syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, ht tps://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation. The Python interpreter can be extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

Python is an interpreted, dynamic, high-level programming language suitable for a wide range of applications.

The philosophy of Python is summarized in The Zen of Python as follows:

- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

The main features of Python are:

- Use of indentation whitespace to indicate blocks
- Object orient paradigm
- Dynamic typing

- Interpreted runtime
- Garbage collected memory management
- a large standard library
- a large repository of third-party libraries

Python is used by many companies and is applied for web development, scientific computing, embedded applications, artificial intelligence, software development, and information security, to name a few.

The material collected here introduces the reader to the basic concepts and features of the Python language and system. After you have worked through the material you will be able to:

- use Python
- use the interactive Python interface
- understand the basic syntax of Python
- write and run Python programs
- have an overview of the standard library
- install Python libraries using venv for multi-Python interpreter development.

This book does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules.

In order to conduct this lesson you need

- A computer with Python 3.10.5 ro newer
- Familiarity with command line usage
- A text editor such as PyCharm, emacs, vi, or others. You should identify which works best for you and set it up.

2.0.1.1 References

Some important additional information can be found on the following Web pages.

- Python
- Pip
- Virtualenv
- NumPy
- SciPy
- Matplotlib

- Pandas
- pyenv
- PyCharm

Python module of the week is a Web site that provides a number of short examples on how to use some elementary python modules. Not all modules are equally useful and you should decide if there are better alternatives. However, for beginners, this site provides a number of good examples

- Python 2: https://pymotw.com/2/
- Python 3: https://pymotw.com/3/

3 INSTALLATION

3.0.1 Python Installation

Learning Objectives

- Learn how to install Python.
- Find additional information about Python.
- Make sure your Computer supports Python.

In this section, we explain how to install python 3.10 on a computer. Likely much of the code will work with earlier versions, but we do the development in Python on the newest version of Python available at https://www.python.org/downloads .

3.0.1.1 Hardware In general, using Python does not require any special hardware. We have installed Python not only on PC's and Laptops but also on Raspberry PI's and Lego Mindstorms.

However, there are some things to consider when developing code. If you use many programs on your desktop and run them all at the same time, you discover that in a up-to-date operating systems you will quickly run out of memory. This is not really a Python issue, but caused by other programs you may run on your computer This is especially true if you use Web browsers and editors such as PyCharm, which we highly recommend. Furthermore, as you likely have lots of disk access, make sure to use a fast HDD we recommend using SSDs or NVMe storage.

A typical modern developer PC or Laptop has *16GB RAM* and an *SSD*. You can certainly do Python on a $35-$75 Raspberry PI, but you probably will not be able to run PyCharm. There are many alternative editors with less memory footprint available.

3.0.1.2 Python 3.10

Here we discuss how to install Python 3.10 or newer on your operating system. It is typically advantageous to use a newer version of Python, so you can leverage the latest features. Please be aware that many operating systems come with older versions that may or may not work for you. You always can start with the version that is installed and if you run into issues update later.

3.0.1.2.1 Python 3.10 on macOS

First, you want tio install a number of useful tools on your macOS. This includes git, make, and a c compiler. All this can be installed with Xcode which is available from

- https://apps.apple.com/us/app/xcode/id497799835

Once you have installed it, you need to install macOS XCode command-line tools:

```
$ xcode-select --install
```

The easiest installation of Python is to use the installation from https://www.python.org/downloads. Please, visit the page and follow the instructions to install the python `.pkg` file. After this install, you have python3 available from the command line.

3.0.1.2.2 Python 3.10 on macOS via Homebrew

Homebrew provides you with an alternative installation. However we noticed that Homebrew may not provide you with the newest version, so we recommend using the install from python.org if you can.

To use this install method, you need to install Homebrew first. Start the process by installing first `homebrew`. Install `homebrew` using the instruction documented on their web page:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
   ↪ Homebrew/install/master/install)"
```

Now you can install Python using:

```
$ brew install python@3.10
```

3.0.1.3 Python 3.10 on Ubuntu 20.04 The default version of Python on Ubuntu 20.04 may note yet be 3.01.2. Thus we recommend to check it with

```
$ python3 --version
$ python3.10 --version
```

If one ov the versions returns 3.10.5 you are all set. However if not we recommend you update your python version as you can benefit from newer version while either installing them through python.org or adding them as follows:

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
$ sudo apt install software-properties-common -y
$ sudo add-apt-repository ppa:deadsnakes/ppa -y
$ sudo apt-get install python3.10 python3-dev -y
```

Now you can verify the version with

```
$ python3.10 --version
```

which should be `3.10.5` or newer.

Now we will create a new virtual environment:

```
$ python3.10 -m venv ~/ENV3
```

Now you must edit the `~/.bashrc` file and add the following line at the end:

```
alias ENV3="source ~/ENV3/bin/activate"
ENV3
```

Now activate the virtual environment using:

```
$ source ~/.bashrc
```

3.0.1.4 Python 3.10.5 on Ubuntu 20.04 from Source We can also install python from source. We have seen perfomance improvements in some systems when we compilled it from source compared to conda and other prebuild python versions. The compilation includes a parameter CORES that indicates how many parallel cores are use. By default we use all of them, however to reduce load you can for example use just half of them. YOu can find out the number of cores with

```
nproc
```

Here the full instalation script

```
$ PYTHON_VESRION="3.10.5"
$ CORES=`nproc`
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
$ sudo apt install -y build-essential zlib1g-dev libncurses5-dev
$ sudo apt install -y libgdbm-dev libnss3-dev libssl-dev libreadline-
   ↪ dev
$ sudo apt install -y libffi-dev libsqlite3-dev wget libbz2-dev
$ cd /tmp
$ wget https://www.python.org/ftp/python/$PYTHON_VERSION/Python-
   ↪ $PYTHON_VERSION.tgz
$ tar -xf Python-$PYTHON_VERSION.tgz
$ cd Python-$PYTHON_VERSION
$ ./configure --enable-optimizations
$ make -j $CORES
$ sudo make altinstall
$ python3.10 --version
$ sudo apt install python3-pip
$ python3.10 -m venv ~/ENV3
$ pip install pip -U
$ which python
$ which pip
```

The which commands should have the directory anme ENV3 in it.

3.0.1.5 Python 3.10.5 on Windows 10

Python 3.10.5 can be installed on Windows 10 using: https://www.python.org/downloads

Let us assume you choose the Web-based installer than you click on the file in the edge browser (make sure the account you use has administrative privileges). Follow the instructions that the installer gives. Important is that you select at one point `[x] Add to Path`. There will be an empty checkmark about this that you will click on.

Once it is installed chose a terminal and execute

```
python --version
```

However, if you have installed conda for some reason, you need to read up on how to install 3.10.5 Python in conda or identify how to run conda and python.org at the same time. We often see others are giving the wrong installation instructions. Please also be aware that when you uninstall conda it is not sufficient to just delete it. You will have to make sure that you unset the system variables automatically set at install time. This includes. modifications on Linux and or Mac in `.zprofile`, `.bashrc` and `.bash_profile`. In windows, PATH and other environment variables may have been modified.

3.0.1.5.1 Python in the Linux Subsystem An alternative is to use Python from within the Linux Subsystem. It has some limitations, and you will need to explore how to access the file system in the subsystem to have a smooth integration between your Windows host so you can, for example, use PyCharm.

To activate the Linux Subsystem, please follow the instructions at

- https://docs.microsoft.com/en-us/windows/wsl/install-win10

A suitable distribution would be

- https://www.microsoft.com/en-us/p/ubuntu-1804-lts/9n9tngvndl3q?activetab=pivot:overviewtab

However, as it may use an older version of Python, you may want to update it as previously discussed

3.0.1.6 Using venv This step is needed if you have not yet already installed a `venv` for Python to make sure you are not interfering with your system python. Not using a venv could have catastrophic consequences and the destruction of your operating system tools if they really on Python. The use of `venv` is simple. For our purposes we assume that you use the directory:

```
~/ENV3
```

Follow these steps first:

First cd to your home directory. Then execute

```
$ python3 -m venv  ~/ENV3
$ source ~/ENV3/bin/activate
```

You can add at the end of your `.bashrc` (ubuntu) or `.bash_profile` or .zprofile‘ (macOS) file the line

If you like to activate it when you start a new terminal, please add this line to your `.bashrc` or `.bash_profile` or `.zprofile` file.

```
$ source ~/ENV3/bin/activate
```

3.0.1.6.1 How to Automatically Start Git Bash in venv on Windows

On Windows the Git Bash can be set to automatically use this venv by issuing these commands:

```
$ cd ~
$ vi .bashrc
```

Press the `i` key and then type

```
source ~/ENV3/Scripts/activate
```

After typing this command press `Enter` and then press the `Esc` key

```
:wq
```

After typing this command press `Enter`. Now every time a new instance of Git Bash is launched, it will automatically be within a virtual environment. The first time Git Bash is restarted after configuring this, it will show an error, but this is normal.

3.0.1.6.2 Confirm Python is installed

Now you are ready to install Cloudmesh.

Check if you have the right version of Python installed with

```
$ python --version
```

To make sure you have an up to date version of pip issue the command

```
$ pip install pip -U
```

3.0.1.7 Install Python 3.10 via Anaconda or Miniconda

We are not recommending ether to use conda or anaconda. If you do so, it is your responsibility to update the information in this section in regards to it.

Anaconda is a popular and large distribution of the python ecosystem frequently used by Data Scientists. Unlike the other python installers, anaconda installs additional tools beyond what is normally considered part of python - such as the conda package management system and an opinionated set of preinstalled packages. Some practitioners consider these additional package installs as bloat as it increases the installation footprint of python to include several third-party packages, which a user may not need.

Miniconda, in contrast installs only what is required to execute the conda packaging system, allowing users to build their own anaconda distribution from scratch without the additional libraries.

As of writing, only the conda-forge channel for anaconda supports python versions greater than 3.10. While you can change channels during install to install the latest versions of python, it is generally not recommended as anaconda does not guarantee that all of its libraries will work properly with the community version of python.

Installing anaconda is straightforward, and all it requires is for users to go through their guided procedures based on what OS you are using. You can find the latest instructions for

- Windows
- MacOS
- Linux

in the conda instalation quide

conda init

When installing Anaconda, it is important to not add it to the path or by running `conda init`. Doing either of these two steps will modify your command prompts and system configuration to register the `(base)` environment of anaconda by default, which will adversely interact with other versions of python installed outside of anaconda. This can result in unexpected command execution or runtime errors.

Instead, later in the instructions will configure your system to only expose the conda command and not add any additional commands and keeping your path clear of any unintentional binaries.

3.0.1.7.1 Configuring conda to be on the path There are many ways to place the `conda` command onto your path. To prevent polluting the command line path, we choose to only expose the `conda` command and none of the environment's underlying binaries. This can be done by doing the following for each OS.

Windows

Run the following at the command line.

```
setx PATH <path_to_conda_install>\condabin;%PATH%
```

If you are using git-bash or equivalent, follow the Linux/MacOS instructions.

Linux / MacOS

Add the following line to your .bashrc, .bash_profile, or .zprofile

```
source <path_to_conda_install>/etc/profile.d/conda.sh
```

3.0.1.7.2 Installing Python via conda Once you have installed conda, you can install Python 3.10 in a virtual environment with conda please use

```
$ conda create -n ENV3 -c conda-forge python=3.10.5 pip
```

Optionally, you can omit the python version to get the latest community version as well.

It is very important that you run the latest version of pip along with python. Failure to do so may result in errors when installing packages designed to use newer versions of pip.

Activating, Inspecting, and Deactivating Conda

Once the above steps have been completed, you must activate your environment for the newly installed version of python and pip are made available. You will need to run this command each time you open a new command window, or you can make it active by default by appending the line to your bash configuration located at `~/.bashrc`, `~/.bash_profile`, or your Zsh profile located at `~/.zprofile`.

To activate our `ENV3` environment, run

```
conda activate ENV3
```

This will augment your current command prompt to use this python environment and all of it's dependencies. Your command prompt will also change to indicate that you are running within this environment by prepending `(ENV3)` to your prompt.

Note that doing this will also expose any anaconda managed programs to your command line (such as openssl and sqlite3). You can see details on your environment by running the command `conda info`. This shows you many important troubleshooting details about your environment, including its name, the path where the environment has been configured, and key paths to conda's configuration.

Additionally, you can inspect the current packages installed in the environment by running the command `conda list`, which will show both conda packages and python packages, their version and build, and the channel (or source) they were installed (main, conda-forge, and pypi are the most commonly seen).

When you are done using this conda environment and the software packages it has installed, you can deactivate the environment by running

```
conda deactivate
```

and your shell will remove the conda environment from your current shell's path.

3.0.1.7.3 Version Test Regardless of which version you install, you must do a version test to make sure you have the correct python and pip versions:

```
$ python --version
$ pip --version
```

If you installed everything correctly, you should see the below versions or newer for each tool:

```
Python 3.10.5
pip 21.3.1
```

If you see an older version of pip, you can update it with

```
pip install -U pip
```

Or with conda,

```
conda update -c conda-forge pip
```

4 FIRST STEPS

4.0.1 Interactive Python

Python can be used interactively. You can enter the interactive mode by entering the interactive loop by executing the command:

```
$ python
```

You will see something like the following:

```
$ python
Python 3.10.5 (main, Jan 18 2022, 10:10:24) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information
   ↪ .
>>>
```

The `>>>` is the prompt used by the interpreter. This is similar to bash where commonly `$` is used.

Sometimes it is convenient to show the prompt when illustrating an example. This is to provide some context for what we are doing. If you are following along you will not need to type in the prompt.

This interactive python process does the following:

- *read* your input commands
- *evaluate* your command
- *print* the result of the evaluation
- *loop* back to the beginning.

This is why you may see the interactive loop referred to as a **REPL**: **R**ead-**E**valuate-**P**rint-**L**oop.

4.0.1.1 REPL (Read Eval Print Loop)

There are many different types beyond what we have seen so far, such as **dictionaries**, **lists**, **sets**. One handy way of using the interactive python is to get the type of a value using type():

```
>>> type(42)
<type 'int'>
>>> type('hello')
<type 'str'>
>>> type(3.14)
<type 'float'>
```

You can also ask for help about something using help():

```
>>> help(int)
>>> help(list)
>>> help(str)
```

Using help() opens up a help message within a pager. To navigate you can use the spacebar to go down a page w to go up a page, the arrow keys to go up/down line-by-line, or q to exit.

4.0.1.2 Interpreter Although the interactive mode provides a convenient tool to test things out you will see quickly that for our class we want to use the python interpreter from the command line. Let us assume the program is called `prg.py`. Once you have written it in that file you simply can call it with

```
$ python prg.py
```

It is important to name the program with meaningful names.

4.0.2 Editors

This section is meant to give an overview of the Python editing tools needed for completing this course. There are many other alternatives; however, we do recommend using PyCharm.

4.0.2.1 PyCharm PyCharm is an Integrated Development Environment (IDE) used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, and integration with git.


Python 8:56 Pycharm

4.0.2.2 Python in 45 minutes Next is an additional community YouTube video about the Python programming language. Naturally, there are many alternatives to this video, but it is probably a good start. It also uses PyCharm which we recommend.


Python 43:16 PyCharm

How much you want to understand Python is a bit up to you. While it is good to know classes and inheritance, you may be able to get away without using it for this class. However, we do recommend that you learn it.

PyCharm Installation:

Method 1: Download and install it from the PyCharm website. This is easy and if no automated install is required we recommend this method. Students and teachers can apply for a free professional version. Please note that Jupyter notebooks can only be viewed in the professional version.

Method 2: PyCharm Installation on ubuntu using umake

```
$ sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make
$ sudo apt-get update
$ sudo apt-get install ubuntu-make
```

Once the `umake` command is installed, use the next command to install PyCharm community edition:

```
$ umake ide pycharm
```

If you want to remove PyCharm installed using umake command, use this:

```
$ umake -r ide pycharm
```

Method 2: PyCharm installation on ubuntu using PPA

```
$ sudo add-apt-repository ppa:mystic-mirage/pycharm
$ sudo apt-get update
$ sudo apt-get install pycharm-community
```

PyCharm also has a Professional (paid) version that can be installed using the following command:

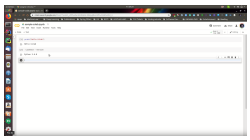
```
$ sudo apt-get install pycharm
```

Once installed, go to your VM dashboard and search for PyCharm.

4.0.3 Google Colab

In this section, we are going to introduce you, how to use Google Colab to run deep learning models.

4.0.3.1 Introduction to Google Colab This video contains the introduction to Google Colab. In this section we will be learning how to start a Google Colab project.



4.0.3.2 Programming in Google Colab In this video, we will learn how to create a simple, Colab Notebook.

Required Installations

```
pip install numpy
```

4.0.3.3 Benchmarking in Google Colab with Cloudmesh In this video, we learn how to do a basic benchmark with Cloudmesh tools. Cloudmesh StopWatch will be used in this tutorial.

Required Installations

```
pip install numpy
pip install cloudmesh-installer
pip install cloudmesh-common
```

5 LANGUAGE

5.0.1 Language

5.0.1.1 Statements and Strings Let us explore the syntax of Python while starting with a print statement

```
print("Hello world from Python!")
```

This will print on the terminal

```
Hello world from Python!
```

The print function was given a **string** to process. A string is a sequence of characters. A **character** can be an alphabetic (A through Z, lower and upper case), numeric (any of the digits), white space (spaces, tabs, newlines, etc), syntactic directives (comma, colon, quotation, exclamation, etc), and so forth. A string is just a sequence of the character and typically indicated by surrounding the characters in double-quotes.

Standard output is discussed in the Section Linux.

So, what happened when you pressed Enter? The interactive Python program read the line `print ("Hello world from Python!")`, split it into the print statement and the `"Hello world from Python!"` string, and then executed the line, showing you the output.

5.0.1.2 Comments

Comments in Python are followed by a `#`:

```
# This is a comment
```

5.0.1.3 Variables

You can store data into a **variable** to access it later. For instance:

```
hello = 'Hello world from Python!'
print(hello)
```

This will print again

```
Hello world from Python!
```

5.0.1.4 Data Types

5.0.1.4.1 Booleans

A **boolean** is a value that can have the values `True` or `False`. You can combine booleans with **boolean operators** such as `and` and `or`

```
print(True and True) # True
print(True and False) # False
print(False and False) # False
print(True or True) # True
print(True or False) # True
print(False or False) # False
```

5.0.1.4.2 Numbers The interactive interpreter can also be used as a calculator. For instance, say we wanted to compute a multiple of 21:

```
print(21 * 2) # 42
```

We saw here the print statement again. We passed in the result of the operation 21 * 2. An **integer** (or **int**) in Python is a numeric value without a fractional component (those are called **floating point** numbers, or **float** for short).

The mathematical operators compute the related mathematical operation to the provided numbers. Some operators are:

Operator	Function
*	multiplication
/	division
+	addition
-	subtraction
**	exponent

Exponentiation x^y is written as x**y is x to the yth power.

You can combine **float**s and **int**s:

```
print(3.14 * 42 / 11 + 4 - 2) # 13.9890909091
print(2**3) # 8
```

Note that **operator precedence** is important. Using parenthesis to indicate affect the order of operations gives a difference results, as expected:

```
print(3.14 * (42 / 11) + 4 - 2) # 11.42
print(1 + 2 * 3 - 4 / 5.0) # 6.2
print( (1 + 2) * (3 - 4) / 5.0 ) # -0.6
```

5.0.1.5 Module Management A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. A module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

5.0.1.5.1 Import Statement When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. The from…import Statement Python's from statement lets you import specific attributes from a module into the current namespace. It is preferred to use for each import its own line such as:

```
import numpy
import matplotlib
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

5.0.1.5.2 The from ... import Statement Python's from statement lets you import specific attributes from a module into the current namespace. The from … import has the following syntax:

```
from datetime import datetime
```

5.0.1.6 Date Time in Python The `datetime` module supplies classes for manipulating dates and times in both simple and complex ways. While date and time arithmetic is supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation. For related functionality, see also the time and calendar modules.

The import Statement You can use any Python source file as a module by executing an import statement in some other Python source file.

```
from datetime import datetime
```

This module offers a generic date/time string parser which is able to parse most known formats to represent a date and/or time.

```
from dateutil.parser import parse
```

pandas is an open-source Python library for data analysis that needs to be imported.

```
import pandas as pd
```

Create a string variable with the class start time

```
fall_start = '08-21-2018'
```

Convert the string to datetime format

```
datetime.strptime(fall_start, '%m-%d-%Y') \#
datetime.datetime(2017, 8, 21, 0, 0)
```

Creating a list of strings as dates

```
class_dates = [
    '8/25/2017',
    '9/1/2017',
    '9/8/2017',
    '9/15/2017',
    '9/22/2017',
    '9/29/2017']
```

Convert Class_dates strings into `datetime` format and save the list into variable a

```
a = [datetime.strptime(x, '%m/%d/%Y') for x in class_dates]
```

Use parse() to attempt to auto-convert common string formats. Parser must be a string or character stream, not list.

```
parse(fall_start) # datetime.datetime(2017, 8, 21, 0, 0)
```

Use parse() on every element of the Class_dates string.

```
[parse(x) for x in class_dates]
# [datetime.datetime(2017, 8, 25, 0, 0),
#  datetime.datetime(2017, 9, 1, 0, 0),
#  datetime.datetime(2017, 9, 8, 0, 0),
#  datetime.datetime(2017, 9, 15, 0, 0),
#  datetime.datetime(2017, 9, 22, 0, 0),
#  datetime.datetime(2017, 9, 29, 0, 0)]
```

Use parse, but designate that the day is first.

```
parse (fall_start, dayfirst=True)
# datetime.datetime(2017, 8, 21, 0, 0)
```

Create a `dataframe`. A DataFrame is a tabular data structure comprised of rows and columns, akin to a spreadsheet, database table. DataFrame is a group of Series objects that share an index (the column names).

```
import pandas as pd
data = {
  'dates': [
    '8/25/2017 18:47:05.069722',
    '9/1/2017 18:47:05.119994',
    '9/8/2017 18:47:05.178768',
    '9/15/2017 18:47:05.230071',
    '9/22/2017 18:47:05.230071',
    '9/29/2017 18:47:05.280592'],
  'complete': [1, 0, 1, 1, 0, 1]}
df = pd.DataFrame(
  data,
  columns = ['dates','complete'])
print(df)
#                  dates  complete
#  0  8/25/2017 18:47:05.069722 1
#  1   9/1/2017 18:47:05.119994 0
#  2   9/8/2017 18:47:05.178768 1
#  3  9/15/2017 18:47:05.230071 1
#  4  9/22/2017 18:47:05.230071 0
#  5  9/29/2017 18:47:05.280592 1
```

Convert `df[`date`]` from string to datetime

```
import pandas as pd
pd.to_datetime(df['dates'])
# 0   2017-08-25 18:47:05.069722
# 1   2017-09-01 18:47:05.119994
# 2   2017-09-08 18:47:05.178768
# 3   2017-09-15 18:47:05.230071
# 4   2017-09-22 18:47:05.230071
# 5   2017-09-29 18:47:05.280592
# Name: dates, dtype: datetime64[ns]
```

5.0.1.7 Control Statements

5.0.1.7.1 Comparison Computer programs do not only execute instructions. Occasionally, a choice needs to be made. Such as a choice is based on a condition. Python has several conditional operators:

Operator	Function
>	greater than
<	smaller than
==	equals
!=	is not

Conditions are always combined with variables. A program can make a choice using the if keyword. For example:

```
x = int(input("Guess x:"))
if x == 4:
   print('Correct!')
```

In this example, *You guessed correctly!* will only be printed if the variable x equals four. Python can also execute multiple conditions using the `elif` and `else` keywords.

```
x = int(input("Guess x:"))
if x == 4:
```

```
    print('Correct!')
elif abs(4 - x) == 1:
    print('Wrong, but close!')
else:
    print('Wrong, way off!')
```

5.0.1.7.2 Iteration To repeat code, the `for` keyword can be used. For example, to display the numbers from 1 to 10, we could write something like this:

```
for i in range(1, 11):
   print('Hello!')
```

The second argument to the `range`, *11*, is not inclusive, meaning that the loop will only get to *10* before it finishes. Python itself starts counting from 0, so this code will also work:

```
for i in range(0, 10):
   print(i + 1)
```

In fact, the range function defaults to starting value of *0*, so it is equivalent to:

```
for i in range(10):
   print(i + 1)
```

We can also nest loops inside each other:

```
for i in range(0,10):
    for j in range(0,10):
        print(i,' ',j)
```

In this case, we have two nested loops. The code will iterate over the entire coordinate range (0,0) to (9,9)

5.0.1.8 Datatypes

5.0.1.8.1 Lists see: https://www.tutorialspoint.com/python/python_lists.htm

Lists in Python are ordered sequences of elements, where each element can be accessed using a 0-based index.

To define a list, you simply list its elements between square brackets '[ ]':

```
names = [
  'Albert',
  'Jane',
  'Liz',
  'John',
  'Abby']
# access the first element of the list
names[0]
# 'Albert'
# access the third element of the list
names[2]
# 'Liz'
```

You can also use a negative index if you want to start counting elements from the end of the list. Thus, the last element has index *-1*, the second before the last element has index *-2* and so on:

```
# access the last element of the list
names[-1]
# 'Abby'
# access the second last element of the list
names[-2]
# 'John'
```

Python also allows you to take whole slices of the list by specifying a beginning and end of the slice separated by a colon

```
# the middle elements, excluding first and last
names[1:-1]
# ['Jane', 'Liz', 'John']
```

As you can see from the example, the starting index in the slice is inclusive and the ending one, exclusive.

Python provides a variety of methods for manipulating the members of a list.

You can add elements with append':

```
names.append('Liz')
names
# ['Albert', 'Jane', 'Liz',
```

```
#  'John', 'Abby', 'Liz']
```

As you can see, the elements in a list need not be unique.

Merge two lists with ‘extend’:

```
names.extend(['Lindsay', 'Connor'])
names
# ['Albert', 'Jane', 'Liz', 'John',
#  'Abby', 'Liz', 'Lindsay', 'Connor']
```

Find the index of the first occurrence of an element with ‘index’:

```
names.index('Liz') \# 2
```

Remove elements by value with ‘remove’:

```
names.remove('Abby')
names
# ['Albert', 'Jane', 'Liz', 'John',
#  'Liz', 'Lindsay', 'Connor']
```

Remove elements by index with ‘pop’:

```
names.pop(1)
# 'Jane'
names
# ['Albert', 'Liz', 'John',
#  'Liz', 'Lindsay', 'Connor']
```

Notice that pop returns the element being removed, while remove does not.

If you are familiar with stacks from other programming languages, you can use insert and ‘pop’:

```
names.insert(0, 'Lincoln')
names
# ['Lincoln', 'Albert', 'Liz',
#  'John', 'Liz', 'Lindsay', 'Connor']
names.pop()
# 'Connor'
names
```

```
# ['Lincoln', 'Albert', 'Liz',
#  'John', 'Liz', 'Lindsay']
```

The Python documentation contains a full list of list operations.

To go back to the range function you used earlier, it simply creates a list of numbers:

```
range(10)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(2, 10, 2)
# [2, 4, 6, 8]
```

5.0.1.8.2 Sets Python lists can contain duplicates as you saw previously:

```
names = ['Albert', 'Jane', 'Liz',
         'John', 'Abby', 'Liz']
```

When we do not want this to be the case, we can use a set:

```
unique_names = set(names)
unique_names
# set(['Lincoln', 'John', 'Albert', 'Liz', 'Lindsay'])
```

Keep in mind that the *set* is an unordered collection of objects, thus we can not access them by index:

```
unique_names[0]
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
#   TypeError: 'set' object does not support indexing
```

However, we can convert a set to a list easily:

```
unique_names = list(unique_names)
unique_names [`Lincoln', `John', `Albert', `Liz', `Lindsay']
unique_names[0]
# `Lincoln'
```

Notice that in this case, the order of elements in the new list matches the order in which the elements were displayed when we create the set. We had

`set(['Lincoln', 'John', 'Albert', 'Liz', 'Lindsay'])`

and now we have

`['Lincoln', 'John', 'Albert', 'Liz', 'Lindsay'])`

You should not assume this is the case in general. That is, do not make any assumptions about the order of elements in a set when it is converted to any type of sequential data structure.

You can change a set's contents using the add, remove and update methods which correspond to the append, remove and extend methods in a list. In addition to these, *set* objects support the operations you may be familiar with from mathematical sets: *union*, *intersection*, *difference*, as well as operations to check containment. You can read about this in the Python documentation for sets.

5.0.1.8.3 Removal and Testing for Membership in Sets One important advantage of a `set` over a `list` is that **access to elements is fast**. If you are familiar with different data structures from a Computer Science class, the Python list is implemented by an array, while the set is implemented by a hash table.

We will demonstrate this with an example. Let us say we have a list and a set of the same number of elements (approximately 100 thousand):

```
import sys, random, timeit
nums_set = set([random.randint(0, sys.maxint) for _ in range(10**5)])
nums_list = list(nums_set)
len(nums_set)
# 100000
```

We will use the timeit Python module to time 100 operations that test for the existence of a member in either the list or set:

```
timeit.timeit('random.randint(0, sys.maxint) in nums',
              setup='import random; nums=%s' % str(nums_set), number
                 ↪ =100)
# 0.0004038810729980469
timeit.timeit('random.randint(0, sys.maxint) in nums',
              setup='import random; nums=%s' % str(nums_list), number
                 ↪ =100)
# 0.398054122924804
```

The exact duration of the operations on your system will be different, but the takeaway will be the same: searching for an element in a set is orders of magnitude faster than in a list. This is important to keep in mind when you work with large amounts of data.

5.0.1.8.4 Dictionaries One of the very important data structures in python is a dictionary also referred to as `dict`.

A dictionary represents a key value store:

```
computer = {
  'name': 'mycomputer',
  'memory': 16,
  'kind': 'Laptop'
  }
print("computer['name']: ", computer['name'])
# computer['name']:  mycomputer
print("computer['memory']: ", computer['memory'])
# computer['Age']:  16
```

A convenient for to print by named attributes is

```
print("{name} {memory}'.format(**computer))
```

This form of printing with the format statement and a reference to data increases the readability of the print statements.

You can delete elements with the following commands:

```
del computer['name'] # remove entry with key 'name'
# computer
# {'Age': 100, 'Class': 'Scientist'}
computer.clear()     # remove all entries in dict
# computer
# {}
del computer         # delete entire dictionary
# computer
# Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
#  NameError: name 'computer' is not defined
```

You can iterate over a dict:

```
computer = {
  'name': 'mycomputer',
  'memory': 16,
  'kind': 'Laptop'
  }
for item in computer:
  print(item, computer[item])

# name mycomputer
# memory 16
# kind laptop
```

5.0.1.8.5 Dictionary Keys and Values You can retrieve both the keys and values of a dictionary using the keys() and values() methods of the dictionary, respectively:

```
computer.keys() # ['name', 'memory', 'kind']
computer.values() # ['mycomputer', 'memory', 'kind']
```

Both methods return lists. Please remember howver that the keys and order in which the elements are returned are not necessarily the same. It is important to keep this in mind:

>  *You cannot make any assumptions about the order in which the elements of a dictionary will be returned by the keys() and values() methods.*

However, you can assume that if you call `keys()` and `values()` in sequence, the order of elements will at least correspond in both methods.

5.0.1.8.6 Counting with Dictionaries One application of dictionaries that frequently comes up is counting the elements in a sequence. For example, say we have a sequence of coin flips:

```
import random
die_rolls = [
  random.choice(['heads', 'tails']) for _ in range(10)
]
# die_rolls
```

```
# ['heads', 'tails', 'heads',
#  'tails', 'heads', 'heads',
   'tails', 'heads', 'heads', 'heads']
```

The actual list die_rolls will likely be different when you execute this on your computer since the outcomes of the die rolls are random.

To compute the probabilities of heads and tails, we could count how many heads and tails we have in the list:

```
counts = {'heads': 0, 'tails': 0}
for outcome in die_rolls:
   assert outcome in counts
   counts[outcome] += 1
print('Probability of heads: %.2f' % (counts['heads'] / len(die_rolls
   ↪ )))
# Probability of heads: 0.70

print('Probability of tails: %.2f' % (counts['tails'] / sum(counts.
   ↪ values())))
# Probability of tails: 0.30
```

In addition to how we use the dictionary counts to count the elements of coin_flips, notice a couple of things about this example:

1. We used the assert outcome in the `count` statement. The assert statement in Python allows you to easily insert debugging statements in your code to help you discover errors more quickly. assert statements are executed whenever the internal Python `__debug__` variable is set to True, which is always the case unless you start Python with the -O option which allows you to run *optimized* Python.
2. When we computed the probability of tails, we used the built-in `sum` function, which allowed us to quickly find the total number of coin flips. The `sum` is one of many built-in functions you can read about here.

5.0.1.9 Functions You can reuse code by putting it inside a function that you can call in other parts of your programs. Functions are also a good way of grouping code that logically belongs together in one coherent whole. A function has a unique name in the program. Once you call a function, it will execute its body which consists of one or more lines of code:

```
def check_triangle(a, b, c):
return \
    a < b + c and a > abs(b - c) and \
    b < a + c and b > abs(a - c) and \
    c < a + b and c > abs(a - b)

    print(check_triangle(4, 5, 6))
```

The def keyword tells Python we are defining a function. As part of the definition, we have the function name, check_triangle, and the parameters of the function – variables that will be populated when the function is called.

We call the function with arguments 4, 5, and 6, which are passed in order into the parameters a, b, and c. A function can be called several times with varying parameters. There is no limit to the number of function calls.

It is also possible to store the output of a function in a variable, so it can be reused.

```
def check_triangle(a, b, c):
  return \
     a < b + c and a > abs(b - c) and \
     b < a + c and b > abs(a - c) and \
     c < a + b and c > abs(a - b)

    result = check_triangle(4, 5, 6)
    print(result)
```

5.0.1.10 Classes A class is an encapsulation of data and the processes that work on them. The data is represented in member variables, and the processes are defined in the methods of the class (methods are functions inside the class). For example, let's see how to define a Triangle class:

```
class Triangle(object):

  def __init__(self, length, width,
               height, angle1, angle2, angle3):
     if not self._sides_ok(length, width, height):
         print('The sides of the triangle are invalid.')
     elif not self._angles_ok(angle1, angle2, angle3):
```

```
            print('The angles of the triangle are invalid.')

        self._length = length
        self._width = width
        self._height = height

        self._angle1 = angle1
        self._angle2 = angle2
        self._angle3 = angle3

    def _sides_ok(self, a, b, c):
        return \
            a < b + c and a > abs(b - c) and \
            b < a + c and b > abs(a - c) and \
            c < a + b and c > abs(a - b)

    def _angles_ok(self, a, b, c):
        return a + b + c == 180

triangle = Triangle(4, 5, 6, 35, 65, 80)
```

Python has full object-oriented programming (OOP) capabilities, however we can not cover all of them in this section, so if you need more information please refer to the Python docs on classes and OOP.

5.0.1.11 Modules Now write this simple program and save it:

```
print("Hello Cloud!")
```

As a check, make sure the file contains the expected contents on the command line:

```
$ cat hello.py
print("Hello Cloud!")
```

To execute your program pass the file as a parameter to the python command:

```
$ python hello.py
Hello Cloud!
```

Files in which Python code is stored are called **module**s. You can execute a Python module from the command line like you just did, or you can import it in other Python code using the import statement.

Let us write a more involved Python program that will receive as input the lengths of the three sides of a triangle, and will output whether they define a valid triangle. A triangle is valid if the length of each side is less than the sum of the lengths of the other two sides and greater than the difference of the lengths of the other two sides.:

```
"""Usage: check_triangle.py [-h] LENGTH WIDTH HEIGHT

Check if a triangle is valid.

Arguments:
  LENGTH     The length of the triangle.
  WIDTH      The width of the triangle.
  HEIGHT     The height of the triangle.

Options:
-h --help
"""
from docopt import docopt

if __name__ == '__main__':
  arguments = docopt(__doc__)
  a, b, c = int(arguments['LENGTH']),
            int(arguments['WIDTH']),
            int(arguments['HEIGHT'])
  valid_triangle = \
      a < b + c and a > abs(b - c) and \
      b < a + c and b > abs(a - c) and \
      c < a + b and c > abs(a - b)
  print('Triangle with sides %d, %d and %d is valid: %r' % (
      a, b, c, valid_triangle
  ))
```

Assuming we save the program in a file called `check_triangle.py`, we can run it like so:

```
$ python check_triangle.py 4 5 6
Triangle with sides 4, 5, and 6 is valid: True
```

Let us break this down a bit.

1. We've defined a boolean expression that tells us if the sides that were input define a valid triangle. The result of the expression is stored in the valid_triangle variable. inside are true, and False otherwise.
2. We've used the backslash symbol `\` to format our code nicely. The backslash simply indicates that the current line is being continued on the next line.
3. When we run the program, we do the check if `__name__ == '__main__'`. `__name__` is an internal Python variable that allows us to tell whether the current file is being run from the command line (value `__name__`), or is being imported by a module (the value will be the name of the module). Thus, with this statement, we arre just making sure the program is being run by the command line.
4. We are using the `docopt` module to handle command line arguments. The advantage of using this module is that it generates a usage help statement for the program and enforces command line arguments automatically. All of this is done by parsing the docstring at the top of the file.
5. In the print function, we are using Python's string formatting capabilities to insert values into the string we are displaying.

5.0.1.12 Lambda Expressions As opposed to normal functions in Python which are defined using the `def` keyword, lambda functions in Python are anonymous functions that do not have a name and are defined using the `lambda` keyword. The generic syntax of a lambda function is in the form of `lambda arguments: expression`, as shown in the following example:

```
greeter = lambda x: print('Hello %s!'%x)
print(greeter('Albert'))
```

As you could probably guess, the result is:

```
Hello Albert!
```

Now consider the following examples:

```
power2 = lambda x: x ** 2
```

The `power2` function defined in the expression, is equivalent to the following definition:

```
def power2(x):
    return x ** 2
```

Lambda functions are useful when you need a function for a short period. Note that they can also be very useful when passed as an argument with other built-in functions that take a function as an argument, e.g. `filter()` and `map()`. In the next example, we show how a lambda function can be combined with the `filer` function. Consider the array `all_names` which contains five words that rhyme together. We want to filter the words that contain the word `name`. To achieve this, we pass the function `lambda x: 'name' in x` as the first argument. This lambda function returns `True` if the word `name` exists as a substring in the string `x`. The second argument of `filter` function is the array of names, i.e. `all_names`.

```
all_names = ['surname', 'rename', 'nickname', 'acclaims', 'defame']
filtered_names = list(filter(lambda x: 'name' in x, all_names))
print(filtered_names)
# ['surname', 'rename', 'nickname']
```

As you can see, the names are successfully filtered as we expected.

In Python, the filter function returns a filter object or the iterator which gets lazily evaluated which means neither we can access the elements of the filter object with index nor we can use `len()` to find the length of the filter object.

```
list_a = [1, 2, 3, 4, 5]
filter_obj = filter(lambda x: x % 2 == 0, list_a)
# Convert the filer obj to a list
even_num = list(filter_obj)
print(even_num)
# Output: [2, 4]
```

In Python, we can have a small usually a single linear anonymous function called Lambda function which can have any number of arguments just like a normal function but with only one expression with no return statement. The result of this expression can be applied to a value.

Basic Syntax:

```
lambda arguments : expression
```

For example, a function in python

```
def multiply(a, b):
   return a*b

#call the function
```

```
multiply(3*5) #outputs: 15
```

The same function can be written as Lambda function. This function named as multiply is having 2 arguments and returns their multiplication.

Lambda equivalent for this function would be:

```
multiply = Lambda a, b : a*b

print(multiply(3, 5))
# outputs: 15
```

Here a and b are the 2 arguments and a*b is the expression whose value is returned as an output.

Also, we don't need to assign the Lambda function to a variable.

```
(lambda a, b : a*b)(3*5)
```

Lambda functions are mostly passed as a parameter to a function which expects a function objects like in map or filter.

5.0.1.12.1 map The basic syntax of the map function is

```
map(function_object, iterable1, iterable2,...)
```

map functions expect a function object and any number of iterable like a list or dictionary. It executes the function_object for each element in the sequence and returns a list of the elements modified by the function object.

Example:

```
def multiply(x):
   return x * 2

map(multiply2, [2, 4, 6, 8])
# Output [4, 8, 12, 16]
```

If we want to write the same function using Lambda

```
map(lambda x: x*2, [2, 4, 6, 8])
# Output [4, 8, 12, 16]
```

5.0.1.12.2 dictionary Now, let us see how we can iterate over a dictionary using map and lambda Let us say we have a dictionary object

```
dict_movies = [
    {'movie': 'avengers', 'comic': 'marvel'},
    {'movie': 'superman', 'comic': 'dc'}]
```

We can iterate over this dictionary and read the elements of it using map and lambda functions in following way:

```
map(lambda x : x['movie'], dict_movies)  # Output: ['avengers', '
    ↪ superman']
map(lambda x : x['comic'],  dict_movies)  # Output: ['marvel', 'dc']
map(lambda x : x['movie'] == "avengers", dict_movies)
# Output: [True, False]
```

In Python3, map function returns an iterator or map object which gets lazily evaluated which means neither we can access the elements of the map object with index nor we can use len() to find the length of the map object. We can force convert the map output i.e. the map object to list as shown next:

```
map_output = map(lambda x: x*2, [1, 2, 3, 4])
print(map_output)
# Output: map object: <map object at 0x04D6BAB0>
list_map_output = list(map_output)
print(list_map_output) # Output: [2, 4, 6, 8]
```

5.0.1.13 Iterators In Python, an iterator protocol is defined using two methods: `__iter()__` and `next()` . The former returns the iterator object and latter returns the next element of a sequence. Some advantages of iterators are as follows:

- Readability
- Supports sequences of infinite length
- Saving resources

There are several built-in objects in Python which implement iterator protocol, e.g. string, list, dictionary. In the following example, we create a new class that follows the iterator protocol. We then use the class to generate `log2` of numbers:

```
from math import log2
```

```
class LogTwo:
    "Implements an iterator of log two"

    def __init__(self,last = 0):
        self.last = last

    def __iter__(self):
        self.current_num = 1
        return self

    def __next__(self):
        if self.current_num <= self.last:
            result = log2(self.current_num)
            self.current_num += 1
            return result
        else:
            raise StopIteration

L = LogTwo(5)
i = iter(L)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
```

As you can see, we first create an instance of the class and assign its `__iter()__` function to a variable called `i`. Then by calling the `next()` function four times, we get the following output:

```
$ python iterator.py
0.0
1.0
1.584962500721156
2.0
```

As you probably noticed, the lines are `log2()` of 1, 2, 3, 4 respectively.

5.0.1.14 Generators Before we go to Generators, please understand Iterators. Generators are also Iterators but they can only be iterated over once. That is because generators do not store the values in memory instead they generate the values on the go. If we want to print those values then we can either simply iterate over them or use the for loop.

5.0.1.14.1 Generators with function For example, we have a function named as multiplyBy10 which prints all the input numbers multiplied by 10.

```
def multiplyBy10(numbers):
   result = []
   for i in numbers:
      result.append(i*10)
   return result

new_numbers = multiplyBy10([1,2,3,4,5])

print new_numbers  #Output: [10, 20, 30, 40 ,50]
```

Now, if we want to use Generators here then we will make the following changes.

```
def multiplyBy10(numbers):
   for i in numbers:
      yield(i*10)

new_numbers = multiplyBy10([1,2,3,4,5])

print new_numbers  #Output: Generators object
```

In Generators, we use yield() function in place of return(). So when we try to print new_numbers list now, it just prints Generators object. The reason for this is because Generators do not hold any value in memory, it yields one result at a time. So essentially it is just waiting for us to ask for the next result. To print the next result we can just say print next(new_numbers) , so how it is working is its reading the first value and squaring it and yielding out value 1. Also in this case, we can just print next(new_numbers) 5 times to print all numbers and if we do it for the 6th time then we will get an error StopIteration which means Generators has exhausted its limit and it has no 6th element to print.

```
print next(new_numbers)  #Output: 1
```

5.0.1.14.2 Generators using for loop If we now want to print the complete list of squared values then we can just do:

```
def multiplyBy10(numbers):
   for i in numbers:
      yield(i*10)

new_numbers = multiplyBy10([1,2,3,4,5])

for num in new_numbers:
   print num
```

The output will be:

```
10
20
30
40
50
```

5.0.1.14.3 Generators with List Comprehension Python has something called List Comprehension, if we use this then we can replace the complete function def with just:

```
new_numbers = [x*10 for x in [1,2,3,4,5]]
print new_numbers  #Output: [10, 20, 30, 40 ,50]
```

Here the point to note is square brackets [] in line 1 is very important. If we change it to () then again we will start getting Generators object.

```
new_numbers = (x*10 for x in [1,2,3,4,5])
print new_numbers  #Output: Generators object
```

We can get the individual elements again from Generators if we do a for loop over `new_numbers`, as we did previously. Alternatively, we can convert it into a list and then print it.

```
new_numbers = (x*10 for x in [1,2,3,4,5])
print list(new_numbers)  #Output: [10, 20, 30, 40 ,50]
```

But here if we convert this into a list then we lose on performance, which we will just see next.

5.0.1.14.4 Why use Generators? Generators are better with Performance because it does not hold the values in memory and here with the small examples we provide it is not a big deal since we are dealing with a small amount of data but just consider a scenario where the records are in millions of data set. And if we try to convert millions of data elements into a list then that will make an impact on memory and performance because everything will in memory.

Let us see an example of how Generators help in Performance. First, without Generators, normal function taking 1 million records and returns the result[people] for 1 million.

```
names = ['John', 'Jack', 'Adam', 'Steve', 'Rick']
majors = ['Math',
          'CompScience',
          'Arts',
          'Business',
          'Economics']

# prints the memory before we run the function
memory = mem_profile.memory_usage_resource()
print (f'Memory (Before): {memory}Mb')

def people_list(people):
   result = []
   for i in range(people):
      person = {
            'id' : i,
            'name' : random.choice(names),
            'major' : randon.choice(majors)
            }
      result.append(person)
   return result

t1 = time.clock()
people = people_list(10000000)
t2 = time.clock()

# prints the memory after we run the function
memory = mem_profile.memory_usage_resource()
print (f'Memory (After): {memory}Mb')
print ('Took {time} seconds'.format(time=t2-t1))
```

```
#Output
Memory (Before): 15Mb
Memory (After): 318Mb
Took 1.2 seconds
```

I am just giving approximate values to compare it with the next execution but we just try to run it we will see a serious consumption of memory with a good amount of time taken.

```
names = ['John', 'Jack', 'Adam', 'Steve', 'Rick']
majors = ['Math',
          'CompScience',
          'Arts',
          'Business',
          'Economics']

# prints the memory before we run the function
memory = mem_profile.memory_usage_resource()
print (f'Memory (Before): {memory}Mb')
def people_generator(people):
   for i in xrange(people):
      person = {
            'id' : i,
            'name' : random.choice(names),
            'major' : randon.choice(majors)
        }
      yield person

t1 = time.clock()
people = people_list(10000000)
t2 = time.clock()

# prints the memory after we run the function
memory = mem_profile.memory_usage_resource()
print (f'Memory (After): {memory}Mb')
print ('Took {time} seconds'.format(time=t2-t1))

#Output
```

```
Memory (Before): 15Mb
Memory (After): 15Mb
Took 0.01 seconds
```

Now after running the same code using Generators, we will see a significant amount of performance boost with almost 0 Seconds. And the reason behind this is that in the case of Generators, we do not keep anything in memory so the system just reads 1 at a time and yields that.

6 CLOUDMESH

6.0.1 Introduction

Learning Objectives

- Introduction to the cloudmesh API
- Using cmd5 via cms
- Introduction to cloudmesh convenience API for output, dotdict, shell, stopwatch, benchmark management
- Creating your own cms commands
- Cloudmesh configuration file
- Cloudmesh inventory

In this chapter, we like to introduce you to cloudmesh which provides you with a number of convenient methods to interface with the local system, but also with cloud services. We will start while focussing on some simple APIs and then gradually introduce the cloudmesh shell which not only provides a shell but also a command line interface so you can use cloudmesh from a terminal. This dual ability is quite useful as we can write cloudmesh scripts, but can also invoke the functionality from the terminal. This is quite an important distinction from other tools that only allow command line interfaces.

Moreover, we also show you that it is easy to create new commands and add them dynamically to the cloudmesh shell via simple pip installs.

Cloudmesh is an evolving project and you have the opportunity to improve if you see some features missing.

The manual of cloudmesh can be found at

- https://cloudmesh.github.io/cloudmesh-manual

The API documentation is located at

- https://cloudmesh.github.io/cloudmesh-manual/api/index.html#cloudmesh-api

We will initially focus on a subset of this functionality.

6.0.2 Installation

The installation of cloudmesh is simple and can technically be done via pip by a user. However you are not a user, you are a developer. Cloudmesh is distributed in different topical repositories and in order for developers to easily interact with them we have written a convenient `cloudmesh-installer` program.

As a developer, you must also use a python virtual environment to avoid affecting your system-wide Python installation. This can be achieved while using Python3 from python.org or via conda. However, we do recommend that you use python.org as this is the vanilla python that most developers in the world use. Conda is often used by users of Python if they do not need to use bleeding-edge but older prepackaged Python tools and libraries.

6.0.2.1 Prerequisite We require you to create a python virtual environment and activate it. How to do this was discussed in Section 3.0.1. Please create the ENV3 environment. Please activate it.

6.0.2.2 Basic Install Cloudmesh can install for developers a number of `bundles`. A bundle is a set of git repositories that are needed for a particular install. For us, we are mostly interested in the bundles `cms`, `cloud`, `storage`. We will introduce you to other bundles throughout this documentation.

If you like to find out more about the details of this you can look at cloudmesh-installer which will be regularly updated.

To make use of the bundle and the easy installation for developers please install the cloudmesh-installer via pip, but make sure you do this in a python virtual env as discussed previously. If not you may impact your system negatively. Please note that we are not responsible for fixing your computer. Naturally, you can also use a virtual machine, if you prefer. It is also important that we create a uniform development environment. In our case, we create an empty directory called `cm` in which we place the bundle.

```
$ mkdir cm
$ cd cm
$ pip install cloudmesh-installer
```

To see the bundle you can use

```
$ cloudmesh-installer bundles
```

We will start with the basic cloudmesh functionality at this time and only install the shell and some common APIs.

```
$ cloudmesh-installer git clone cms
$ cloudmesh-installer install cms
```

These commands download and install cloudmesh shell into your environment. It is important that you use the `-e` flag

To see if it works you can use the command

```
$ cms help
```

You will see an output. If this does not work for you, and you can not figure out the issue, please contact us so we can identify what went wrong.

For more information, please visit our Installation Instructions for Developers

6.0.3 Output

Cloudmesh provides a number of convenient API's to make the output easier or more fanciful.

These API's include

- Console
- Banner
- Heading
- VERBOSE

6.0.3.1 Console Print is the usual function to output to the terminal. However, often we like to have colored output that helps us in the notification to the user. For this reason, we have a simple `Console` class that has several built-in features. You can even switch and define your own color schemes.

```
from cloudmesh.common.console import Console

msg = "my message"
Console.ok(msg) # prins a green message
Console.error(msg) # prins a red message proceeded with ERROR
Console.msg(msg) # prins a regular black message
```

In case of the error message we also have convenient flags that allow us to include the traceback in the output.

```
Console.error(msg, prefix=True, traceflag=True)
```

The prefix can be switched on and off with the `prefix` flag, while the `traceflag` switches on and of if the trace should be set.

The verbosity of the output is controlled via variables that are stored in the `~/.cloudmesh` directory.

```
from cloudmesh.common.variables import Variables

variables = Variables()

variables['debug'] = True
variables['trace'] = True
variables['verbose'] = 10
```

For more features, see API: Console

6.0.3.2 Banner

In case you need a banner you can do this with

```
from cloudmesh.common.util import banner

banner("my text")
```

For more features, see API: Banner

6.0.3.3 Heading

A particularly useful function is `HEADING()` which prints the method name.

```
from cloudmesh.common.util import HEADING

class Example(object):

    def doit(self):
        HEADING()
        print ("Hello")
```

The invocation of the `HEADING()` function doit prints a banner with the name information. The reason we did not do it as a decorator is that you can place the `HEADING()` function in an arbitrary location of the method body.

For more features, see API: Heading

6.0.3.4 VERBOSE *Note: VERBOSE is not supported in jupyter notebooks*

VERBOSE is a very useful method allowing you to print a dictionary. Not only will it print the dict, but it will also provide you with the information in which file it is used and which line number. It will even print the name of the `dict` that you use in your code.

To use this you will have to enable the debugging methods for cloudmesh as discussed in Section 6.0.3.1

```
from cloudmesh.common.debug import VERBOSE

m = {"key": "value"}
VERBOSE(m)
```

For more features, please see VERBOSE

6.0.3.5 Using print and pprint In many cases, it may be sufficient to use `print` and `pprint` for debugging. However, as the code is big and you may forget where you placed print statements or the print statements may have been added by others, we recommend that you use the VERBOSE function. If you use `print` or `pprint` we recommend using a unique prefix, such as:

```
from pprint import pprint

d = {"sample": "value"}
```

```
print("MYDEBUG:")
pprint (d)

# or with print

print("MYDEBUG:", d)
```

6.0.4 Dictionaries

6.0.4.1 Dotdict For simple dictionaries we sometimes like to simplify the notation with a `.` instead of using the `[]`:

You can achieve this with `dotdict`

```
from cloudmesh.common.dotdict import dotdict

data = {
    "name": "Gregor"
}

data = dotdict(data)
```

Now you can either call

```
data["name"]
```

or

```
data.name
```

This is especially useful in if conditions as it may be easier to read and write

```
if data.name is "Gregor":

    print("this is quite readable")
```

and is the same as

```
if data["name"] is "Gregor":
```

```
    print("this is quite readable")
```

For more features, see API: dotdict

6.0.4.2 FlatDict In some cases, it is useful to be able to flatten out dictionaries that contain dicts within dicts. For this, we can use `FlatDict`.

```
from cloudmesh.common.Flatdict import FlatDict

data = {
    "name": "Gregor",
    "address": {
        "city": "Bloomington",
        "state": "IN"

    }
}

flat = FlatDict(data, sep=".")
```

This will be converted to a dict with the following structure.

```

flat = {
    "name": "Gregor"
    "address.city": "Bloomington",
    "address.state": "IN"
}
```

With `sep` you can change the sepaerator between the nested dict attributes. For more features, see API: dotdict

6.0.4.3 Printing Dicts In case we want to print dicts and lists of dicts in various formats, we have included a simple `Printer` that can print a dict in yaml, json, table, and csv format.

The function can even guess from the passed parameters what the input format is and uses the appropriate internal function.

A common example is

```
from pprint import pprint
from cloudmesh.common.Printer import Printer

data = [
    {
        "name": "Gregor",
        "address": {
            "street": "Funny Lane 11",
            "city": "Cloudville"
        }
    },
    {
        "name": "Albert",
        "address": {
            "street": "Memory Lane 1901",
            "city": "Cloudnine"
        }
    }
]


pprint(data)

table = Printer.flatwrite(data,
                          sort_keys=["name"],
                          order=["name", "address.street", "address.
                             ↪ city"],
                          header=["Name", "Street", "City"],
                          output='table')

print(table)
```

For more features, see API: Printer

More examples are available in the source code as tests

6.0.5 Shell

Python provides a sophisticated method for starting background processes. However, in many cases, it is quite complex to interact with it. It also does not provide convenient wrappers that we can use to start them in a pythonic fashion. For this reason, we have written a primitive `Shell` class that provides just enough functionality to be useful in many cases.

Let us review some examples where `result` is set to the output of the command being executed.

```
from cloudmesh.common.Shell import Shell

result = Shell.execute('pwd')
print(result)

result = Shell.execute('ls', ["-l", "-a"])
print(result)

result = Shell.execute('ls', "-l -a")
print(result)
```

Sometimes it may just be more convenient to use a string instead of the array for passing the command. Here you can use

```
from cloudmesh.common.Shell import Shell

result = Shell.run('pwd')
print(result)

result = Shell.run('ls -l -a")
print(result)
```

For many common commands, we provide built-in functions. For example:

```
result = Shell.ls("-aux")
print(result)

result = Shell.ls("-a", "-u", "-x")
print(result)

result = Shell.pwd()
```

```
print(result)
```

The list includes (naturally the commands that must be available on your OS. If the shell command is not available on your OS, please help us improving the code to either provide functions that work on your OS or develop with us platform-independent functionality of a subset of the functionality for the shell command that we may benefit from.

- `VBoxManage(cls, *args)`
- `bash(cls, *args)`
- `blockdiag(cls, *args)`
- `brew(cls, *args)`
- `cat(cls, *args)`
- `check_output(cls, *args, **kwargs)`
- `check_python(cls)`
- `cm(cls, *args)`
- `cms(cls, *args)`
- `command_exists(cls, name)`
- `dialog(cls, *args)`
- `edit(filename)`
- `execute(cls,*args)`
- `fgrep(cls, *args)`
- `find_cygwin_executables(cls)`
- `find_lines_with(cls, lines, what)`
- `get_python(cls)`
- `git(cls, *args)`
- `grep(cls, *args)`
- `head(cls, *args)`
- `install(cls, name)`
- `install(cls, name)`
- `keystone(cls, *args)`
- `kill(cls, *args)`
- `live(cls, command, cwd=None)`
- `ls(cls, *args)`
- `mkdir(cls, directory)`
- `mongod(cls, *args)`
- `nosetests(cls, *args)`
- `nova(cls, *args)`

- `operating_system(cls)`
- `pandoc(cls, *args)`
- `ping(cls, host=None, count=1)`
- `pip(cls, *args)`
- `ps(cls, *args)`
- `pwd(cls, *args)`
- `rackdiag(cls, *args)`
- `remove_line_with(cls, lines, what)`
- `rm(cls, *args)`
- `rsync(cls, *args)`
- `scp(cls, *args)`
- `sh(cls, *args)`
- `sort(cls, *args)`
- `ssh(cls, *args)`
- `sudo(cls, *args)`
- `tail(cls, *args)`
- `terminal(cls, command='pwd')`
- `terminal_type(cls)`
- `unzip(cls, source_filename, dest_dir)`
- `vagrant(cls, *args)`
- `version(cls, name)`
- `which(cls, command)`

For more features, please see Shell

6.0.6 StopWatch

Often you find yourself in a situation where you like to measure the time between two events. We provide a simple `StopWatch` that allows you not only to measure a number of times but also to print them out in a convenient format.

```
from cloudmesh.common.StopWatch import StopWatch
from time import sleep

StopWatch.start("test")
sleep(1)
StopWatch.stop("test")
```

```
print (StopWatch.get("test"))
```

To print, you can simply also use:

```
StopWatch.benchmark()
```

For more features, please see StopWatch

6.0.7 Cloudmesh Command Shell

6.0.7.1 CMD5 Python's CMD (https://docs.python.org/2/library/cmd.html) is a very useful package to create command line shells. However, it does not allow the dynamic integration of newly defined commands. Furthermore, additions to CMD need to be done within the same source tree. To simplify developing commands by a number of people and to have a dynamic plugin mechanism, we developed cmd5. It is a rewrite of our earlier efforts in cloudmesh client and cmd3.

6.0.7.1.1 Resources The source code for cmd5 is located in GitHub:

- https://github.com/cloudmesh/cmd5

We have discussed in Section 6.0.2 how to install cloudmesh as a developer and have access to the source code in a directory called `cm`. As you read this document we assume you are a developer and can skip the next section.

6.0.7.1.2 Installation from source WARNING: DO NOT EXECUTE THIS IF YOU ARE A DEVELOPER OR YOUR ENVIRONMENT WILL NOT PROPERLY WORK. YOU LIKELY HAVE ALREADY INSTALLED CMD5 IF YOU USED THE CLOUDMESH INSTALLER.

However, if you are a user of cloudmesh you can install it with

```
$ pip install cloudmesh-cmd5
```

6.0.7.1.3 Execution To run the shell you can activate it with the cms command. cms stands for cloudmesh shell:

```
(ENV2) $ cms
```

It will print the banner and enter the shell:

```
+-------------------------------------------------------+
|   ____ _                 _                      _     |
|  / ___| | ___  _   _  __| |_ __ ___   ___  ___| |__  |
| | |   | |/ _ \| | | |/ _` | '_ ` _ \ / _ \/ __| '_ \ |
| | |___| | (_) | |_| | (_| | | | | | |  __/\__ \ | | | |
|  \____|_|\___/ \__,_|\__,_|_| |_| |_|\___||___/_| |_| |
+-------------------------------------------------------+
|                  Cloudmesh CMD5 Shell                 |
+-------------------------------------------------------+

cms>
```

To see the list of commands you can say:

```
cms> help
```

To see the manual page for a specific command, please use:

```
help COMMANDNAME
```

6.0.7.1.4 Create your own Extension One of the most important features of CMD5 is its ability to extend it with new commands. This is done via packaged namespaces. We recommend you name it cloudmesh-mycommand, where mycommand is the name of the command that you like to create. This can easily be done while using the sys* cloudmesh command (we suggest you use a different name than `gregor` maybe your firstname):

```
$ cms sys command generate gregor
```

It will download a template from cloudmesh called `cloudmesh-bar` and generate a new directory `cloudmesh-gregor` with all the needed files to create your own command and register it dynamically with cloudmesh. All you have to do is to `cd` into the directory and install the code:

```
$ cd cloudmesh-gregor
$ python setup.py install
# pip install .
```

Adding your command is easy. It is important that all objects are defined in the command itself and that no global variables be used to allow each shell command to stand alone. Naturally, you should

develop API libraries outside of the cloudmesh shell command and reuse them to keep the command code as small as possible. We place the command in:

```
cloudmsesh/mycommand/command/gregor.py
```

Now you can go ahead and modify your command in that directory. It will look similar to (if you used the command name `gregor`):

```
from cloudmesh.shell.command import command
from cloudmesh.shell.command import PluginCommand

class GregorCommand(PluginCommand):

    @command
    def do_gregor(self, args, arguments):
        """
        ::
          Usage:
                gregor -f FILE
                gregor list
          This command does some useful things.
          Arguments:
              FILE   a file name
          Options:
              -f      specify the file
        """
        print(arguments)
        if arguments.FILE:
           print("You have used file: ", arguments.FILE)
        return ""
```

An important difference to other CMD solutions is that our commands can leverage (besides the standard definition), `docopts` as a way to define the manual page. This allows us to use arguments as dict and use simple if conditions to interpret the command. Using `docopts` has the advantage that contributors are forced to think about the command and its options and document them from the start. Previously we did not use but argparse and click. However, we noticed that for our contributors both systems lead to commands that were either not properly documented or the developers delivered ambiguous commands that resulted in confusion and wrong usage by subsequent users. Hence, we do recommend that you use docopts for documenting cmd5 commands. The transformation is enabled

by the @command decorator that generates a manual page and creates a proper help message for the shell automatically. Thus there is no need to introduce a separate help method as would normally be needed in CMD while reducing the effort it takes to contribute new commands in a dynamic fashion.

6.0.7.1.5 Bug: Quotes We have one bug in cmd5 that relates to the use of quotes on the command-line

For example, you need to say

```
$ cms gregor -f \"file name with spaces\"
```

If you like to help us fix this that would be great. it requires the use of shlex. Unfortunately, we did not yet time to fix this "feature".

6.0.8 Exercises

When doing your assignment, make sure you label the programs appropriately with comments that clearly identify the assignment. Place all assignments in a folder on GitHub named "cloudmesh-exercises"

For example, name the program solving E.Cloudmesh.Common.1 `e-cloudmesh-1.py` and so on. For more complex assignments you can name them as you like, as long as in the file you have a comment such as

`# fa19-516-000 E.Cloudmesh.Common.1`

at the beginning of the file. Please **do not store** any screenshots in your GitHub repository of your working program.

6.0.8.1 Cloudmesh Common E.Cloudmesh.Common.1

> Develop a program that demonstrates the use of `banner`, `HEADING`, and `VERBOSE`.

E.Cloudmesh.Common.2

> Develop a program that demonstrates the use of `dotdict`.

E.Cloudmesh.Common.3

> Develop a program that demonstrates the use of `FlatDict`.

E.Cloudmesh.Common.4

> Develop a program that demonstrates the use of `cloudmesh.common.Shell`.

E.Cloudmesh.Common.5

> Develop a program that demonstrates the use of `cloudmesh.common.StopWatch`.

6.0.8.2 Cloudmesh Shell E.Cloudmesh.Shell.1

> Install cmd5 and the command `cms` on your computer.

E. Cloudmesh.Shell.2

> Write a new command with your `firstname` as the command name.

E.Cloudmesh.Shell.3

> Write a new command and experiment with docopt syntax and argument interpretation of the dict with if conditions.

E.Cloudmesh.Shell.4

> If you have useful extensions that you like us to add by default, please work with us.

E.Cloudmesh.Shell.5

> At this time one needs to quote in some commands the `"` in the shell command line. Develop and test code that fixes this.

7 LIBRARIES

7.0.1 Python Modules

Often you may need functionality that is not present in Python's standard library. In this case, you have two option:

- implement the features yourself
- use a third-party library that has the desired features.

Often you can find a previous implementation of what you need. Since this is a common situation, there is a service supporting it: the Python Package Index (or PyPi for short).

Our task here is to install the autopep8 tool from PyPi. This will allow us to illustrate the use of virtual environments using the `venv` and installing and uninstalling PyPi packages using pip.

7.0.1.1 Updating Pip You must have the newest version of pip installed for your version of python. Let us assume your python is registered with python and you use venv, than you can update pip with

```
pip install -U pip
```

without interfering with a potential system-wide installed version of pip that may be needed by the system default version of python. See the section about venv for more details

7.0.1.2 Using pip to Install Packages Let us now look at another important tool for Python development: the Python Package Index, or PyPI for short. PyPI provides a large set of third-party Python packages.

To install a package from PyPI, use the pip command. We can search for PyPI for packages:

```
$ pip search --trusted-host pypi.python.org autopep8 pylint
```

It appears that the top two results are what we want, thus install them:

```
$ pip install --trusted-host pypi.python.org autopep8 pylint
```

This will cause pip to download the packages from PyPI, extract them, check their dependencies and install those as needed, then install the requested packages.

7.0.1.3 GUI

7.0.1.3.1 GUIZero Install `guizero` with the following command:

```
sudo pip install guizero
```

For a comprehensive tutorial on `guizero`, click here.

7.0.1.3.2 Kivy You can install Kivy on macOS as follows:

```
brew install pkg-config sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer
pip install -U Cython
pip install kivy
pip install pygame
```

A hello world program for kivy is included in the cloudmesh.robot repository. Which you can find here

- https://github.com/cloudmesh/cloudmesh.robot/tree/master/projects/kivy

To run the program, please download it or execute it in cloudmesh.robot as follows:

```
cd cloudmesh.robot/projects/kivy
python swim.py
```

To create stand-alone packages with kivy, please see:

```
-  https://kivy.org/docs/guide/packaging-osx.html
```

7.0.1.4 Formatting and Checking Python Code

First, get the bad code:

```
$ wget --no-check-certificate http://git.io/pXqb -O bad_code_example.
   ↪ py
```

Examine the code:

```
$ emacs bad_code_example.py
```

As you can see, this is very dense and hard to read. Cleaning it up by hand would be a time-consuming and error-prone process. Luckily, this is a common problem so there exist a couple of packages to help in this situation.

7.0.1.5 Using autopep8

We can now run the bad code through autopep8 to fix formatting problems:

```
$ autopep8 bad_code_example.py >code_example_autopep8.py
```

Let us look at the result. This is considerably better than before. It is easy to tell what the example1 and example2 functions are doing.

It is a good idea to develop a habit of using autopep8 in your python-development workflow. For instance: use autopep8 to check a file, and if it passes, make any changes in place using the `-i` flag:

```
$ autopep8 file.py    # check output to see of passes
$ autopep8 -i file.py # update in place
```

If you use pyCharm you can use a similar function while pressing on Inspect Code.

7.0.1.6 Writing Python 3 Compatible Code To write python 2 and 3 compatible code we recommend that you take a look at: http://python-future.org/compatible_idioms.html

7.0.1.7 Using Python on FutureSystems This is only important if you use Futuresystems resources.

To use Python you must log in to your FutureSystems account. Then at the shell prompt execute the following command:

```
$ module load python
```

This will make the python and virtualenv commands available to you.

The details of what the module load command does are described in the future lesson modules.

7.0.1.8 Ecosystem

7.0.1.8.1 pypi The Python Package Index is a large repository of software for the Python programming language containing a large number of packages, many of which can be found on pypi. The nice thing about pypi is that many packages can be installed with the program 'pip'.

To do so you have to locate the <package_name> for example with the search function in pypi and say on the command line:

```
$ pip install <package_name>
```

where `package_name` is the string name of the package. an example would be the package called cloudmesh_client which you can install with:

```
$ pip install cloudmesh_client
```

If all goes well the package will be installed.

7.0.1.8.2 Alternative Installations The basic installation of python is provided by python.org. However, others claim to have alternative environments that allow you to install python. This includes

- Canopy
- Anaconda
- IronPython

Typically they include not only the python compiler but also several useful packages. It is fine to use such environments for the class, but it should be noted that in both cases not every python library may be available for install in the given environment. For example, if you need to use cloudmesh client, it may not be available as conda or Canopy package. This is also the case for many other cloud-related and useful python libraries. Hence, we do recommend that if you are new to python to use the distribution from python.org, and use pip and virtualenv.

Additionally, some python versions have platform-specific libraries or dependencies. For example, coca libraries, `.NET`, or other frameworks are examples. For the assignments and the projects, such platform-dependent libraries are not to be used.

If however, you can write a platform-independent code that works on Linux, macOS, and Windows while using the python.org version but develop it with any of the other tools that are just fine. However, it is up to you to guarantee that this independence is maintained and implemented. You do have to write requirements.txt files that will install the necessary python libraries in a platform-independent fashion. The homework assignment PRG1 has even a requirement to do so.

In order to provide platform independence we have given in the class a *minimal* python version that we have tested with hundreds of students: python.org. If you use any other version, that is your decision. Additionally, some students not only use python.org but have used iPython which is fine too. However, this class is not only about python, but also about how to have your code run on any platform. The homework is designed so that you can identify a setup that works for you.

However, we have concerns if you for example wanted to use chameleon cloud which we require you to access with cloudmesh. cloudmesh is not available as conda, canopy, or other framework packages. Cloudmesh client is available form pypi which is standard and should be supported by the frameworks. We have not tested cloudmesh on any other python version than python.org which is the open-source community standard. None of the other versions are standard.

In fact, we had students over the summer using canopy on their machines and they got confused as they now had multiple python versions and did not know how to switch between them and activate the correct version. Certainly, if you know how to do that, then feel free to use canopy, and if you want to use canopy all this is up to you. However, the homework and project require you to make your program portable to python.org. If you know how to do that even if you use canopy, anaconda, or any other python version that is fine. Graders will test your programs on a python.org installation and not canopy, anaconda, ironpython while using virtualenv. It is obvious why. If you do not know that answer you may want to think about that every time they test a program they need to do a new virtualenv and run vanilla python in it. If we were to run two installs in the same system, this will not work as we do not know if one student will cause a side effect for another. Thus we as instructors do not just have to look at your code but code of hundreds of students with different setups. This is a non-scalable solution as every time we test out code from a student we would have to wipe out the OS, install it new,

install a new version of whatever python you have elected, become familiar with that version, and so on and on. This is the reason why the open-source community is using python.org. We follow best practices. Using other versions is not a community best practice, but may work for an individual.

We have however in regards to using other python versions additional bonus projects such as

- deploy run and document cloudmesh on ironpython
- deploy run and document cloudmesh on anaconda, develop script to generate a conda package form github
- deploy run and document cloudmesh on canopy, develop script to generate a conda package form github
- deploy run and document cloudmesh on ironpython
- other documentation that would be useful

7.0.1.9 Resources If you are unfamiliar with programming in Python, we also refer you to some of the numerous online resources. You may wish to start with Learn Python or the book Learn Python the Hard Way. Other options include Tutorials Point or Code Academy, and the Python wiki page contains a long list of references for learning as well. Additional resources include:

- https://virtualenvwrapper.readthedocs.io
- https://github.com/yyuu/pyenv
- https://amaral.northwestern.edu/resources/guides/pyenv-tutorial
- https://godjango.com/96-django-and-python-3-how-to-setup-pyenv-for-multiple-pythons/
- https://www.accelebrate.com/blog/the-many-faces-of-python-and-how-to-manage-them/
- http://ivory.idyll.org/articles/advanced-swc/
- http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html
- http://www.youtube.com/watch?v=0vJJlVBVTFg
- http://www.korokithakis.net/tutorials/python/
- http://www.afterhoursprogramming.com/tutorial/Python/Introduction/
- http://www.greenteapress.com/thinkpython/thinkCSpy.pdf
- https://docs.python.org/3.3/tutorial/modules.html
- https://www.learnpython.org/en/Modules/_and/_Packages
- https://docs.python.org/2/library/datetime.html
- https://chrisalbon.com/python/strings/_to/_datetime.html

A very long list of useful information is also available from

- https://github.com/vinta/awesome-python
- https://github.com/rasbt/python_reference

This list may be useful as it also contains links to data visualization and manipulation libraries, and AI tools and libraries. Please note that for this class you can reuse such libraries if not otherwise stated.

7.0.1.9.1 Jupyter Notebook Tutorials A Short Introduction to Jupyter Notebooks and NumPy To view the notebook, open this link in a background tab https://nbviewer.jupyter.org/ and copy and paste the following link in the URL input area https://cloudmesh.github.io/classes/lesson/prg/Jupyter-NumPy-tutorial-I523-F2017.ipynb Then hit Go.

7.0.1.10 Exercises

E.Python.Lib.1:

> Write a python program called iterate.py that accepts an integer n from the command line. Pass this integer to a function called iterate.

> The iterate function should then iterate from 1 to n. If the i-th number is a multiple of three, print *multiple of 3*, if a multiple of 5 print *multiple of 5*, if a multiple of both print *multiple of 3 and 5*, else print the value.

E:Python.Lib.2:

> 1. Create a pyenv or virtualenv ~/ENV

> 2. Modify your ~/.bashrc shell file to activate your environment upon login.

> 3. Install the docopt python package using pip

> 4. Write a program that uses docopt to define a command line program. Hint: modify the iterate program.

> 5. Demonstrate the program works.

7.0.2 Data Management

Obviously when dealing with big data we may not only be dealing with data in one format but in many different formats. It is important that you will be able to master such formats and seamlessly integrate in your analysis. Thus we provide some simple examples on which different data formats exist and how to use them.

7.0.2.1 Formats

7.0.2.1.1 Pickle Python pickle allows you to save data in a python native format into a file that can later be read in by other programs. However, the data format may not be portable among different python versions thus the format is often not suitable to store information. Instead we recommend for standard data to use either json or yaml.

```
import pickle

flavor = {
    "small": 100,
    "medium": 1000,
    "large": 10000
    }

pickle.dump( flavor, open( "data.p", "wb" ) )
```

To read it back in use

```
flavor = pickle.load( open( "data.p", "rb" ) )
```

7.0.2.1.2 Text Files To read text files into a variable called content you can use

```
content = open('filename.txt', 'r').read()
```

You can also use the following code while using the convenient `with` statement

```
with open('filename.txt','r') as file:
    content = file.read()
```

To split up the lines of the file into an array you can do

```
with open('filename.txt','r') as file:
    lines = file.read().splitlines()
```

This cam also be done with the build in `readlines` function

```
lines = open('filename.txt','r').readlines()
```

In case the file is too big you will want to read the file line by line:

```
with open('filename.txt','r') as file:
    line = file.readline()
    print (line)
```

7.0.2.1.3 CSV Files Often data is contained in comma separated values (CSV) within a file. To read such files you can use the csv package.

```
import csv
with open('data.csv', 'rb') as f:
   contents = csv.reader(f)
for row in content:
    print row
```

Using pandas you can read them as follows.

```
import pandas as pd
df = pd.read_csv("example.csv")
```

There are many other modules and libraries that include CSV read functions. In case you need to split a single line by comma, you may also use the `split` function. However, remember it swill split at every comma, including those contained in quotes. So this method although looking originally convenient has limitations.

7.0.2.1.4 Excel spread sheets Pandas contains a method to read Excel files

```
import pandas as pd
filename = 'data.xlsx'
data = pd.ExcelFile(file)
df = data.parse('Sheet1')
```

7.0.2.1.5 YAML YAML is a very important format as it allows you easily to structure data in hierarchical fields It is frequently used to coordinate programs while using yaml as the specification for configuration files, but also data files. To read in a yaml file the following code can be used

```
import yaml
```

```
with open('data.yaml', 'r') as f:
    content = yaml.load(f)
```

The nice part is that this code can also be used to verify if a file is valid yaml. To write data out we can use

```
with open('data.yml', 'w') as f:
    yaml.dump(data, f, default_flow_style=False)
```

The flow style set to false formats the data in a nice readable fashion with indentations.

7.0.2.1.6 JSON

```
import json
with open('strings.json') as f:
    content = json.load(f)
```

7.0.2.1.7 XML XML format is extensively used to transport data across the web. It has a hierarchical data format, and can be represented in the form of a tree.

A Sample XML data looks like:

```
<data>
    <items>
        <item name="item-1"></item>
        <item name="item-2"></item>
        <item name="item-3"></item>
    </items>
</data>
```

Python provides the ElementTree XML API to parse and create XML data.

Importing XML data from a file:

```
import xml.etree.ElementTree as ET
tree = ET.parse('data.xml')
root = tree.getroot()
```

Reading XML data from a string directly:

```
root = ET.fromstring(data_as_string)
```

Iterating over child nodes in a root:

```
for child in root:
    print(child.tag, child.attrib)
```

Modifying XML data using ElementTree:

- Modifying text within a tag of an element using .text method:

```
tag.text = new_data
tree.write('output.xml')
```

- Adding/modifying an attribute using .set() method:

```
tag.set('key', 'value')
tree.write('output.xml')
```

Other Python modules used for parsing XML data include

- minidom: https://docs.python.org/3/library/xml.dom.minidom.html
- BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/

7.0.2.1.8 RDF To read RDF files you will need to install RDFlib with

```
$ pip install rdflib
```

This will than allow you to read RDF files

```
from rdflib.graph import Graph
g = Graph()
g.parse("filename.rdf", format="format")
for entry in g:
   print(entry)
```

Good examples on using RDF are provided on the RDFlib Web page at https://github.com/RDFLib/rdflib

From the Web page we showcase also how to directly process RDF data from the Web

```
import rdflib
g=rdflib.Graph()
```

```
g.load('http://dbpedia.org/resource/Semantic_Web')

for s,p,o in g:
    print s,p,o
```

7.0.2.1.9 PDF The Portable Document Format (PDF) has been made available by Adobe Inc. royalty free. This has enabled PDF to become a world wide adopted format that also has been standardized in 2008 (ISO/IEC 32000-1:2008, https://www.iso.org/standard/51502.html). A lot of research is published in papers making PDF one of the de-facto standards for publishing. However, PDF is difficult to parse and is focused on high quality output instead of data representation. Nevertheless, tools to manipulate PDF exist:

PDFMiner https://pypi.python.org/pypi/pdfminer/ allows the simple translation of PDF into text that than can be further mined. The manual page helps to demonstrate some examples http://euske.github.io/pdfminer/index.html.

pdf-parser.py https://blog.didierstevens.com/programs/pdf-tools/ parses pdf documents and identifies some structural elements that can than be further processed.

If you know about other tools, let us know.

7.0.2.1.10 HTML A very powerful library to parse HTML Web pages is provided with https://www.crummy.com/software/BeautifulSoup/

More details about it are provided in the documentation page https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Beautiful Soup is a python library to parse, process and edit HTML documents.

To install Beautiful Soup, use `pip` command as follows:

```
$ pip install beautifulsoup4
```

In order to process HTML documents, a parser is required. Beautiful Soup supports the HTML parser included in Python's standard library, but it also supports a number of third-party Python parsers like the `lxml` parser which is commonly used [1].

Following command can be used to install `lxml` parser

```
$ pip install lxml
```

To begin with, we import the package and instantiate an object as follows for a html document `html_handle`:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_handle, `lxml`)
```

Now, we will discuss a few functions, attributes and methods of Beautiful Soup.

prettify function

`prettify()` method will turn a Beautiful Soup parse tree into a nicely formatted Unicode string, with a separate line for each HTML/XML tag and string. It is analgous to `pprint()` function. The object created above can be viewed by printing the prettfied version of the document as follows:

```
print(soup.prettify())
```

tag Object

A `tag` object refers to tags in the HTML document. It is possible to go down to the inner levels of the DOM tree. To access a tag `div` under the tag `body`, it can be done as follows:

```
body_div = soup.body.div
print(body_div.prettify())
```

The `attrs` attribute of the tag object returns a dictionary of all the defined attributes of the HTML tag as keys.

has_attr() method

To check if a `tag` object has a specific attribute, `has_attr()` method can be used.

```
if body_div.has_attr('p'):
    print('The value of \'p\' attribute is:', body_div['p'])
```

tag object attributes

- `name` - This attribute returns the name of the tag selected.
- `attrs` - This attribute returns a dictionary of all the defined attributes of the HTML tag as keys.
- `contents` - This attribute returns a list of contents enclosed within the HTML tag
- `string` - This attribute which returns the text enclosed within the HTML tag. This returns `None` if there are multiple children
- `strings` - This overcomes the limitation of `string` and returns a generator of all strings enclosed within the given tag

Following code showcases usage of the above discussed attributes:

```
body_tag = soup.body

print("Name of the tag:', body_tag.name)

attrs = body_tag.attrs
print('The attributes defined for body tag are:', attrs)

print('The contents of \'body\' tag are:\n', body_tag.contents)

print('The string value enclosed in \'body\' tag is:', body_tag.
   ↪ string)

for s in body_tag.strings:
    print(repr(s))
```

Searching the Tree

- `find()` function takes a filter expression as argument and returns the first match found
- `findall()` function returns a list of all the matching elements

```
search_elem = soup.find('a')
print(search_elem.prettify())

search_elems = soup.find_all("a", class_="sample")
pprint(search_elems)
```

- `select()` function can be used to search the tree using CSS selectors

```
# Select `a` tag with class `sample`
a_tag_elems = soup.select('a.sample')
print(a_tag_elems)
```

7.0.2.1.11 ConfigParser

- https://pymotw.com/2/ConfigParser/

7.0.2.1.12 ConfigDict

- https://github.com/cloudmesh/cloudmesh-common/blob/master/cloudmesh/common/ConfigDict.py

7.0.2.2 Encryption Often we need to protect the information stored in a file. This is achieved with encryption. There are many methods of supporting encryption and even if a file is encrypted it may be target to attacks. Thus it is not only important to encrypt data that you do not want others to se but also to make sure that the system on which the data is hosted is secure. This is especially important if we talk about big data having a potential large effect if it gets into the wrong hands.

To illustrate one type of encryption that is non trivial we have chosen to demonstrate how to encrypt a file with an ssh key. In case you have openssl installed on your system, this can be achieved as follows.

```
#! /bin/sh

# Step 1. Creating a file with data
echo "Big Data is the future." > file.txt

# Step 2. Create the pem
openssl rsa -in ~/.ssh/id_rsa -pubout  > ~/.ssh/id_rsa.pub.pem

# Step 3. look at the pem file to illustrate how it looks like (
   ↪ optional)
cat ~/.ssh/id_rsa.pub.pem

# Step 4. encrypt the file into secret.txt
openssl rsautl -encrypt -pubin -inkey ~/.ssh/id_rsa.pub.pem -in
   ↪ file.txt -out secret.txt

# Step 5. decrypt the file and print the contents to stdout
openssl rsautl -decrypt -inkey ~/.ssh/id_rsa -in secret.txt
```

Most important here are Step 4 that encrypts the file and Step 5 that decrypts the file. Using the Python os module it is straight forward to implement this. However, we are providing in cloudmesh a convenient class that makes the use in python very simple.

```
from cloudmesh.common.ssh.encrypt import EncryptFile
```

```
e = EncryptFile('file.txt', 'secret.txt')
e.encrypt()
e.decrypt()
```

In our class we initialize it with the locations of the file that is to be encrypted and decrypted. To initiate that action just call the methods `encrypt` and `decrypt`.

7.0.2.3 Database Access see: https://www.tutorialspoint.com/python/python_database_access.htm

7.0.2.4 SQLite

- https://www.sqlite.org/index.html
- https://docs.python.org/3/library/sqlite3.html

7.0.2.4.1 Exercises E:Encryption.1:

> Test the shell script to replicate how this example works

E:Encryption.2:

> Test the cloudmesh encryption class

E:Encryption.3:

> What other encryption methods exist. Can you provide an example and contribute to the section?

E:Encryption.4:

> What is the issue of encryption that make it challenging for Big Data

E:Encryption.5:

> Given a test dataset with many files text files, how long will it take to encrypt and decrypt them on various machines. Write a benchmark that you test. Develop this benchmark as a group, test out the time it takes to execute it on a variety of platforms.

7.0.3 Plotting with matplotlib

A brief overview of plotting with matplotlib along with examples is provided. First, matplotlib must be installed, which can be accomplished with pip install as follows:

```
$ pip install matplotlib
```

We will start by plotting a simple line graph using built in NumPy functions for sine and cosine. This first step is to import the proper libraries shown next.

```
import numpy as np
import matplotlib.pyplot as plt
```

Next, we will define the values for the x-axis, we do this with the linspace option in numpy. The first two parameters are the starting and ending points, these must be scalars. The third parameter is optional and defines the number of samples to be generated between the starting and ending points, this value must be an integer. Additional parameters for the linspace utility can be found here:

```
x = np.linspace(-np.pi, np.pi, 16)
```

Now we will use the sine and cosine functions in order to generate y values, for this we will use the values of x for the argument of both our sine and cosine functions i.e. $cos(x)$.

```
cos = np.cos(x)
sin = np.sin(x)
```

You can display the values of the three parameters we have defined by typing them in a python shell.

```
x
array([-3.14159265, -2.72271363, -2.30383461, -1.88495559,
   ↪ -1.46607657,
    -1.04719755, -0.62831853, -0.20943951, 0.20943951, 0.62831853,
    1.04719755, 1.46607657, 1.88495559, 2.30383461, 2.72271363,
    3.14159265])
```

Having defined x and y values we can generate a line plot and since we imported matplotlib.pyplot as plt we simply use plt.plot.

```
plt.plot(x,cos)
```

We can display the plot using plt.show() which will pop up a figure displaying the plot defined.

```
plt.show()
```

Additionally, we can add the sine line to outline graph by entering the following.

```
plt.plot(x,sin)
```

Invoking plt.show() now will show a figure with both sine and cosine lines displayed. Now that we have a figure generated it would be useful to label the x and y-axis and provide a title. This is done by the following three commands:

```
plt.xlabel("X - label (units)")
plt.ylabel("Y - label (units)")
plt.title("A clever Title for your Figure")
```

Along with axis labels and a title another useful figure feature may be a legend. In order to create a legend you must first designate a label for the line, this label will be what shows up in the legend. The label is defined in the initial plt.plot(x,y) instance, next is an example.

```
plt.plot(x,cos, label="cosine")
```

Then in order to display the legend, the following command is issued:

```
plt.legend(loc='upper right')
```

The location is specified by using upper or lower and left or right. Naturally, all these commands can be combined and put in a file with the .py extension and run from the command line.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 16)
cos = np.cos(x)
sin = np.sin(x)
plt.plot(x,cos, label="cosine")
plt.plot(x,sin, label="sine")

plt.xlabel("X - label (units)")
plt.ylabel("Y - label (units)")
plt.title("A clever Title for your Figure")
```

```
plt.legend(loc='upper right')

plt.show()
```

link error

An example of a bar chart is preceded next using data from [T:fast-cars].

```
import matplotlib.pyplot as plt

x = [' Toyota Prius',
     'Tesla Roadster ',
     ' Bugatti Veyron',
     ' Honda Civic ',
     ' Lamborghini Aventador ']
horse_power = [120, 288, 1200, 158, 695]

x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, horse_power, color='green')
plt.xlabel("Car Model")
plt.ylabel("Horse Power (Hp)")
plt.title("Horse Power for Selected Cars")

plt.xticks(x_pos, x)

plt.show()
```

You can customize plots further by using plt.style.use(), in python 3. If you provide the following command inside a python command shell you will see a list of available styles.

```
print(plt.style.available)
```

An example of using a predefined style is shown next.

```
plt.style.use('seaborn')
```

Up to this point, we have only showcased how to display figures through python output, however web browsers are a popular way to display figures. One example is Bokeh, the following lines can be

entered in a python shell and the figure is outputted to a browser.

```
from bokeh.io import show
from bokeh.plotting import figure

x_values = [1, 2, 3, 4, 5]
y_values = [6, 7, 2, 3, 6]

p = figure()
p.circle(x=x_values, y=y_values)
show(p)
```

7.0.4 DocOpts

When we want to design command line arguments for python programs we have many options. However, as our approach is to create documentation first, docopts provides also a good approach for Python. The code for it is located at

- https://github.com/docopt/docopt

It can be installed with

```
$ pip install docopt
```

Sample programs are located at

- https://github.com/docopt/docopt/blob/master/examples/options_example.py

A sample program of using doc opts for our purposes looks as follows

```
"""Cloudmesh VM management

Usage:
  cm-go vm start NAME [--cloud=CLOUD]
  cm-go vm stop NAME [--cloud=CLOUD]
  cm-go set --cloud=CLOUD
  cm-go -h | --help
  cm-go --version

Options:
```

```
  -h --help     Show this screen.
  --version     Show version.
  --cloud=CLOUD  The name of the cloud.
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.

ARGUMENTS:
  NAME     The name of the VM`
"""
from docopt import docopt

if __name__ == '__main__':
    arguments = docopt(__doc__, version='1.0.0rc2')
    print(arguments)
```

Another good feature of using docopts is that we can use the same verbal description in other programming languages as showcased in this book.

7.0.5 OpenCV

Learning Objectives

- Provide some simple calculations so we can test cloud services.
- Showcase some elementary OpenCV functions
- Show an environmental image analysis application using Secchi disks

---

OpenCV (Open Source Computer Vision Library) is a library of thousands of algorithms for various applications in computer vision and machine learning. It has C++, C, Python, Java, and MATLAB interfaces and supports Windows, Linux, Android, and Mac OS. In this section, we will explain the basic features of this library, including the implementation of a simple example.

7.0.5.1 Overview OpenCV has many functions for image and video processing. The pipeline starts with reading the images, low-level operations on pixel values, preprocessing e.g. denoising, and then multiple steps of higher-level operations which vary depending on the application. OpenCV covers the whole pipeline, especially providing a large set of library functions for high-level operations. A simpler library for image processing in Python is Scipy's multi-dimensional image processing package (scipy.ndimage).

7.0.5.2 Installation OpenCV for Python can be installed on Linux in multiple ways, namely PyPI(Python Package Index), Linux package manager (apt-get for Ubuntu), Conda package manager, and also building from source. You are recommended to use PyPI. Here's the command that you need to run on Ubuntu:

```
$ sudo apt update
$ sudo apt install libopencv-dev python3-opencv
$ python3 -c "import cv2; print(cv2.__version__)"
```

The last commnd will print the version of opencv you have. In order to test, import the module in Python command line:

```
import cv2
```

If it does not raise an error, it is installed correctly. Otherwise, try to solve the error.

For installation on Windows, see:

- https://docs.opencv.org/4.x/d3/d52/tutorial_windows_install.html

Note that building from source can take a long time and may not be feasible for deploying to limited platforms such as Raspberry Pi.

7.0.5.3 A Simple Example In this example, an image is loaded. A simple processing is performed, and the result is written to a new image.

7.0.5.3.1 Loading an image

```
import cv2

img = cv2.imread('images/opencv/4.2.01.tiff')
```

The image was downloaded from USC standard database:

http://sipi.usc.edu/database/database.php?volume=misc&image=9

7.0.5.3.2 Displaying the image The image is saved in a numpy array. Each pixel is represented with 3 values (R,G,B). This provides you with access to manipulate the image at the level of single pixels. You can display the image using imshow function as well as Matplotlib's imshow function.

You can display the image using imshow function:

```
cv2.imshow('Original',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

or you can use Matplotlib. If you have not installed Matplotlib before, install it using:

```
$ pip install matplotlib
```

Now you can use:

```
import matplotlib.pyplot as plt
plt.imshow(img)
```

which results in Figure 1

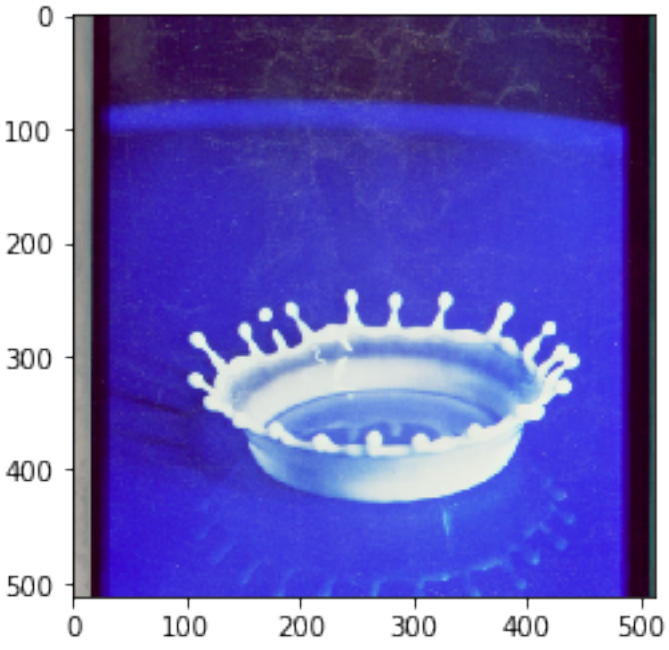


Figure 1: Image display

7.0.5.3.3 Scaling and Rotation

Scaling (resizing) the image relative to different axis

```
res = cv2.resize(img,
                 None,
                 fx=1.2,
                 fy=0.7,
                 interpolation=cv2.INTER_CUBIC)
plt.imshow(res)
```

which results in Figure 2

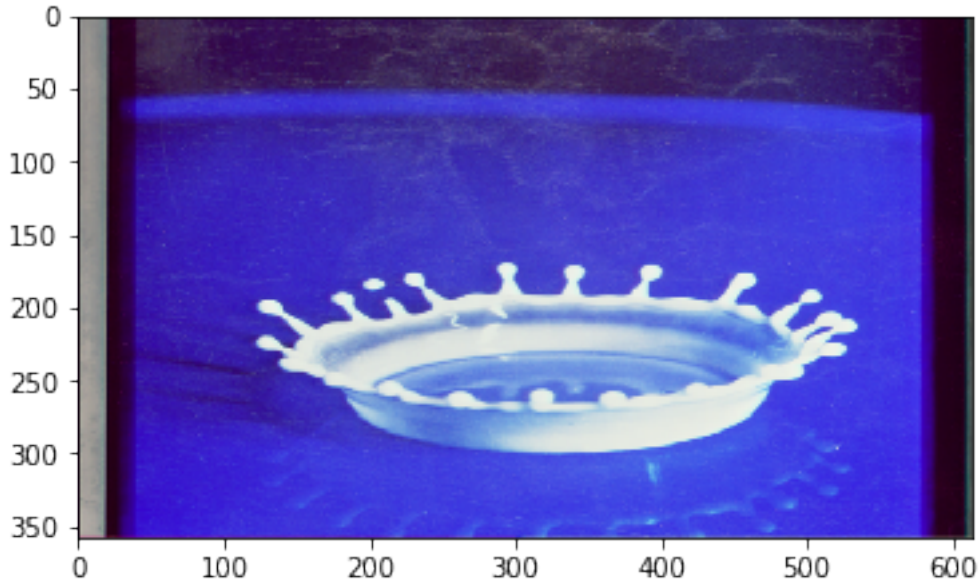


Figure 2: Scaling and rotation

Rotation of the image for an angle of t

```
rows,cols,_ = img.shape
t = 45
M = cv2.getRotationMatrix2D((cols/2,rows/2),t,1)
dst = cv2.warpAffine(img,M,(cols,rows))

plt.imshow(dst)
```

which results in Figure 3

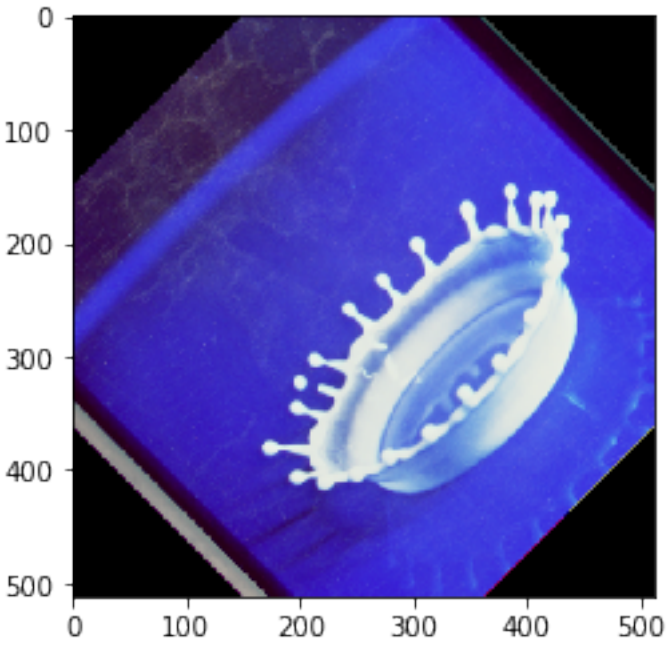


Figure 3: image

7.0.5.3.4 Gray-scaling

```
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img2, cmap='gray')
```

which results in +Figure 4

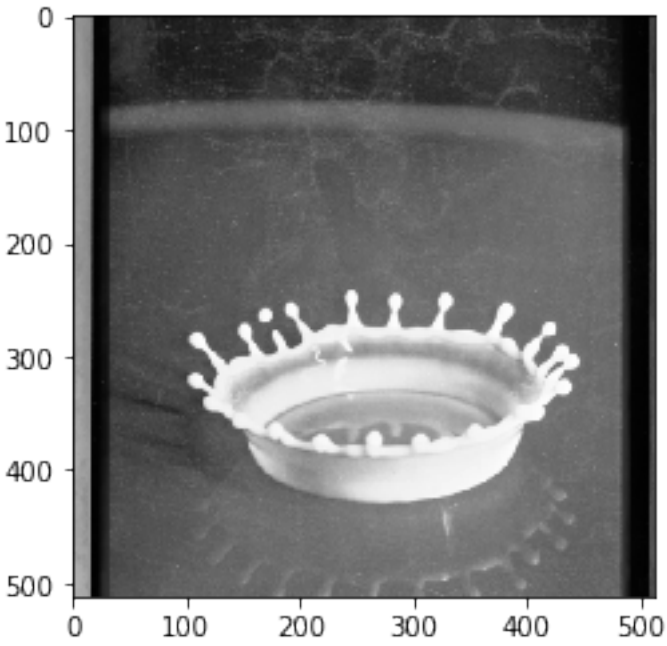


Figure 4: Gray sacling

7.0.5.3.5 Image Thresholding

```
ret,thresh =    cv2.threshold(img2,127,255,cv2.THRESH_BINARY)
plt.subplot(1,2,1), plt.imshow(img2, cmap='gray')
plt.subplot(1,2,2), plt.imshow(thresh, cmap='gray')
```

which results in Figure 5

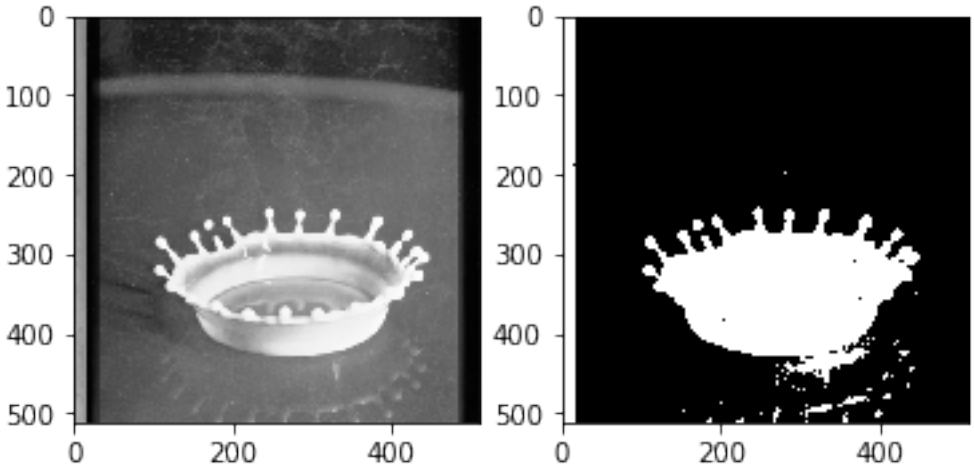


Figure 5: Image Thresholding

7.0.5.3.6 Edge Detection Edge detection using Canny edge detection algorithm

```
edges = cv2.Canny(img2,100,200)

plt.subplot(121),plt.imshow(img2,cmap = 'gray')
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
```

which results in Figure 6

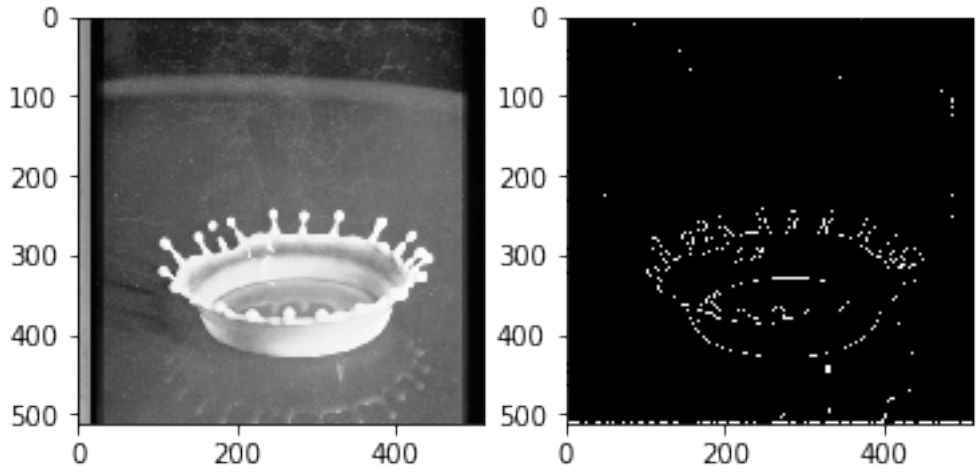


Figure 6: Edge detection

7.0.5.4 Additional Features OpenCV has implementations of many machine learning techniques such as KMeans and Support Vector Machines can be put into use with only a few lines of code. It also has functions especially for video analysis, feature detection, object recognition, and many more. You can find out more about them on their website

OpenCV

7.0.6 Secchi Disk

We are developing an autonomous robot boat that you can be part of developing within this class. The robot bot is measuring turbidity or water clarity. Traditionally this has been done with a Secchi disk. The use of the Secchi disk is as follows:

1. Lower the Secchi disk into the water.
2. Measure the point when you can no longer see it
3. Record the depth at various levels and plot in a geographical 3D map

One of the things we can do is take a video of the measurement instead of a human recording them. Then we can analyze the video automatically to see how deep a disk was lowered. This is a classical image analysis program. You are encouraged to identify algorithms that can identify the depth. The simplest seems to be to do a histogram at a variety of depth steps and measure when the histogram no longer changes significantly. The depth of that image will be the measurement we look for.

Thus if we analyze the images we need to look at the image and identify the numbers on the measuring tape, as well as the visibility of the disk.

To showcase how such a disk looks like we refer to the image showcasing different Secchi disks. For our purpose the black-white contrast Secchi disk works well. See Figure 7

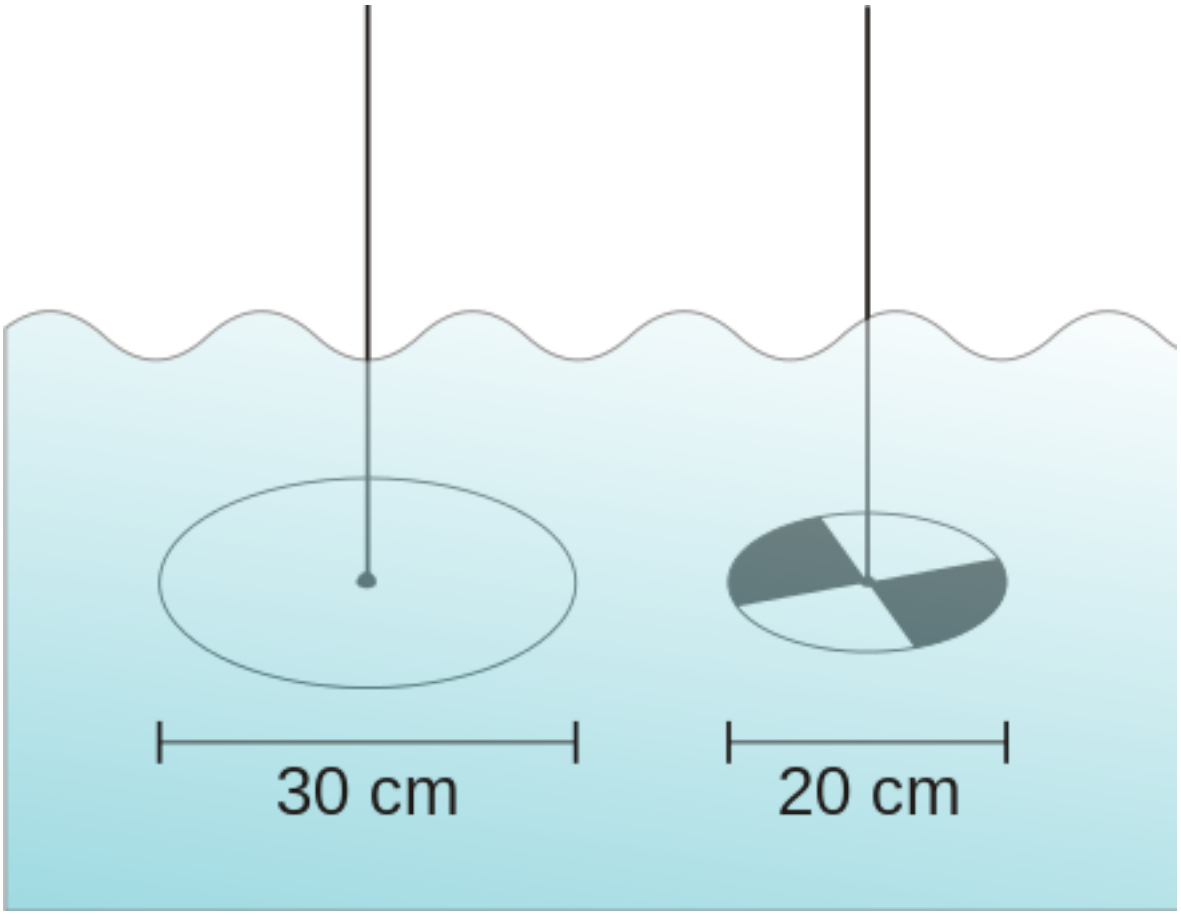


Figure 7: Secchi disk types. A marine style on the left and the freshwater version on the right wikipedia.

More information about Secchi Disk can be found at:

- https://en.wikipedia.org/wiki/Secchi/_disk

We have included next a couple of examples while using some obviously useful OpenCV methods. Surprisingly, the use of the edge detection that comes to mind first to identify if we still can see the disk, seems too complicated to use for analysis. We at this time believe the histogram will be sufficient.

Please inspect our examples.

7.0.6.1 Setup for OSX First lest setup the OpenCV environment for OSX. Naturally, you will have to update the versions based on your versions of python. When we tried the install of OpenCV on macOS, the setup was slightly more complex than other packages. This may have changed by now and if you have improved instructions, please let us know. However, we do not want to install it via Anaconda out of the obvious reason that anaconda installs too many other things.

```
import os, sys
from os.path import expanduser
os.path
home = expanduser("~")
sys.path.append('/usr/local/Cellar/opencv/3.3.1_1/lib/python3.6/site-
   ↪ packages/')
sys.path.append(home + '/.pyenv/versions/OPENCV/lib/python3.6/site-
   ↪ packages/')
import cv2
cv2.__version__
! pip install numpy > tmp.log
! pip install matplotlib >> tmp.log
%matplotlib inline
```

7.0.6.2 Step 1: Record the video Record the video on the robot

We have done this for you and will provide you with images and videos if you are interested in analyzing them. See Figure 8

7.0.6.3 Step 2: Analyse the images from the Video For now, we just selected 4 images from the video

```
import cv2
import matplotlib.pyplot as plt

img1 = cv2.imread('secchi/secchi1.png')
```

```
img2 = cv2.imread('secchi/secchi2.png')
img3 = cv2.imread('secchi/secchi3.png')
img4 = cv2.imread('secchi/secchi4.png')

figures = []
fig = plt.figure(figsize=(18, 16))
for i in range(1,13):
  figures.append(fig.add_subplot(4,3,i))
count = 0
for img in [img1,img2,img3,img4]:
  figures[count].imshow(img)

  color = ('b','g','r')
  for i,col in enumerate(color):
      histr = cv2.calcHist([img],[i],None,[256],[0,256])
      figures[count+1].plot(histr,color = col)

  figures[count+2].hist(img.ravel(),256,[0,256])

  count += 3

print("Legend")
print("First column = image of Secchi disk")
print("Second column = histogram of colors in image")
print("Third column = histogram of all values")

plt.show()
```

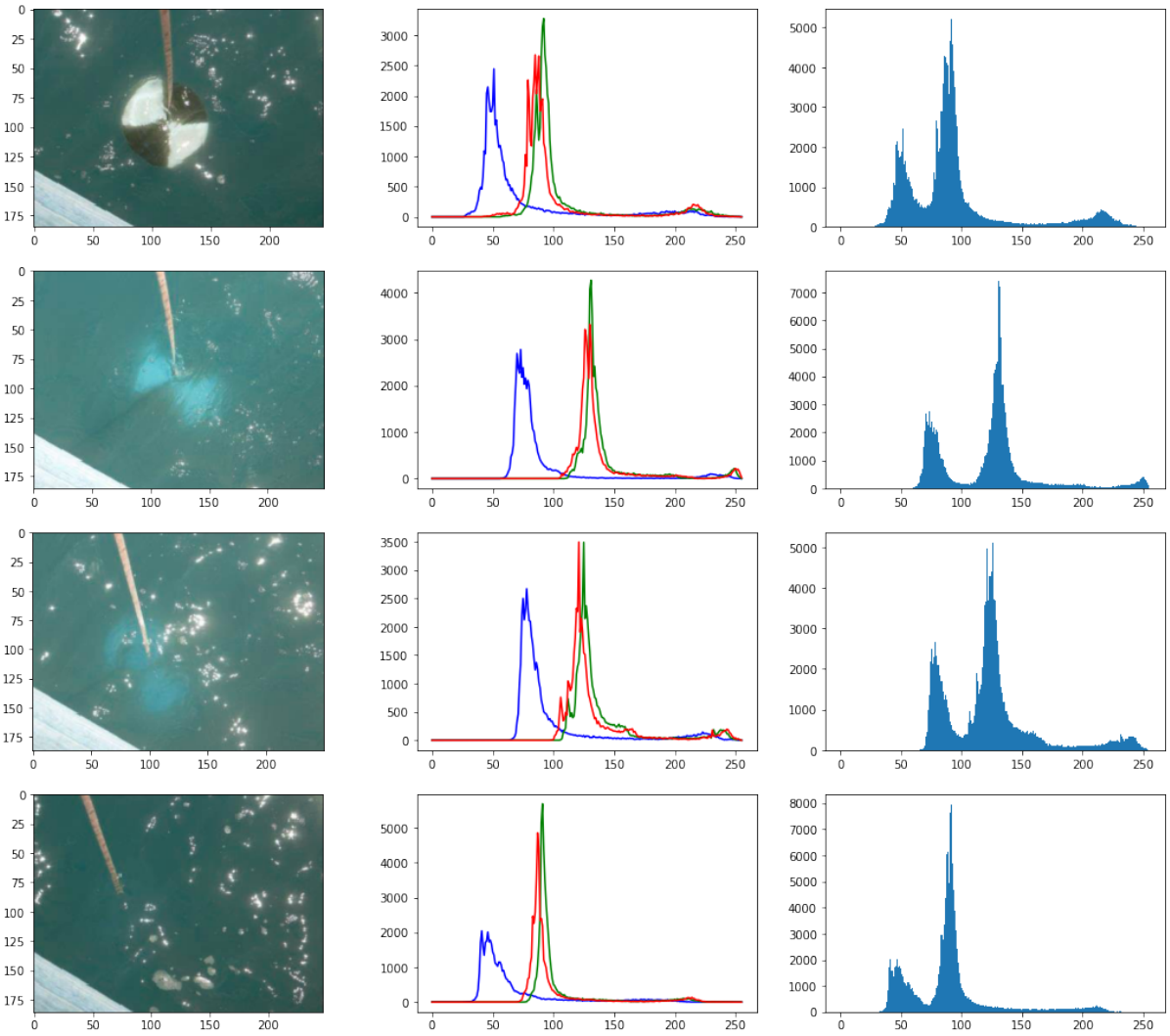


Figure 8: Histogram

7.0.6.3.1 Image Thresholding

See Figure 9, Figure 10, Figure 11, Figure 12

```
def threshold(img):
  ret,thresh = cv2.threshold(img,150,255,cv2.THRESH_BINARY)
  plt.subplot(1,2,1), plt.imshow(img, cmap='gray')
  plt.subplot(1,2,2), plt.imshow(thresh, cmap='gray')

threshold(img1)
threshold(img2)
threshold(img3)
```

```
threshold(img4)
```

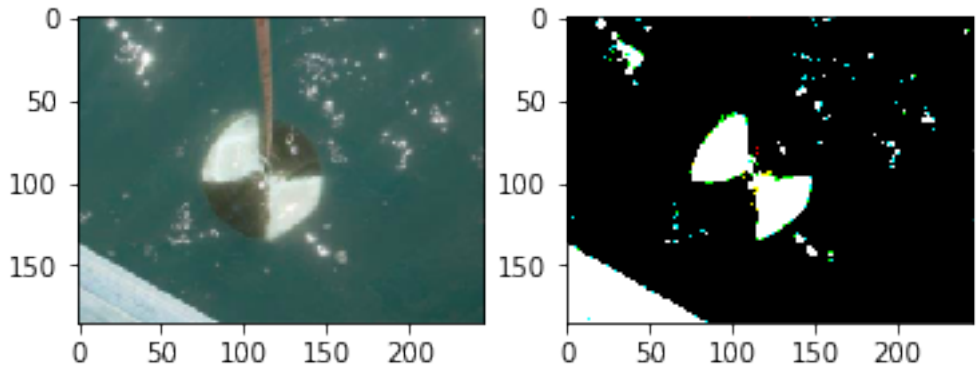


Figure 9: Threshold 1, `threshold(img1)`

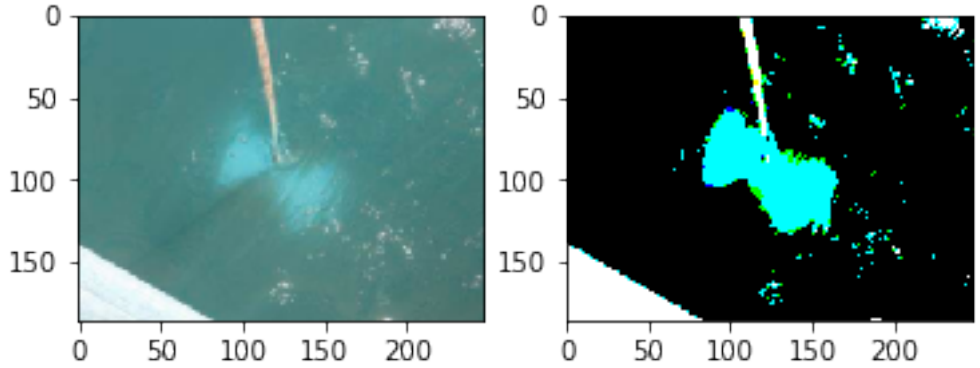


Figure 10: Threshold 2, `threshold(img2)`

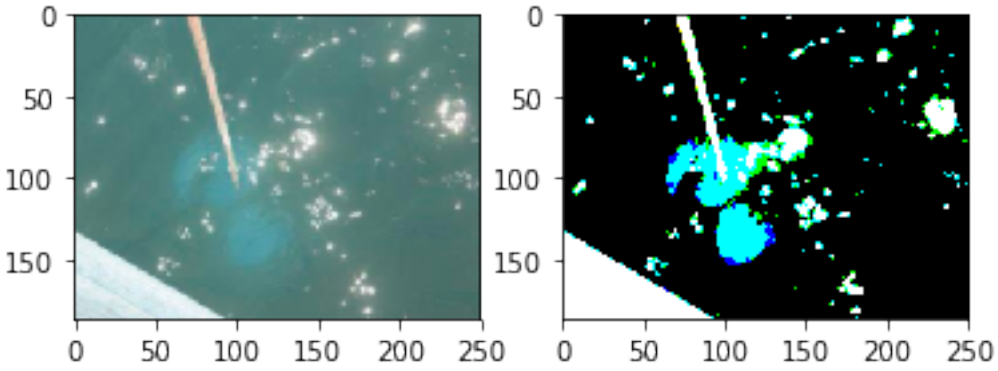


Figure 11: Threshold 3, `threshold(img3)`

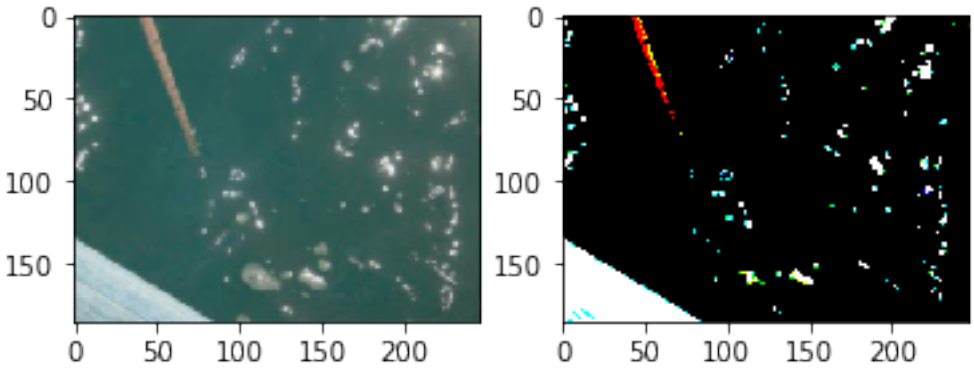


Figure 12: Threshold 4, `threshold(img4)`

7.0.6.3.2 Edge Detection See Figure 13, Figure 14, Figure 15, Figure 16, Figure 17. Edge detection using Canny edge detection algorithm

```
def find_edge(img):
  edges = cv2.Canny(img,50,200)
  plt.subplot(121),plt.imshow(img,cmap = 'gray')
  plt.subplot(122),plt.imshow(edges,cmap = 'gray')

find_edge(img1)
find_edge(img2)
find_edge(img3)
find_edge(img4)
```

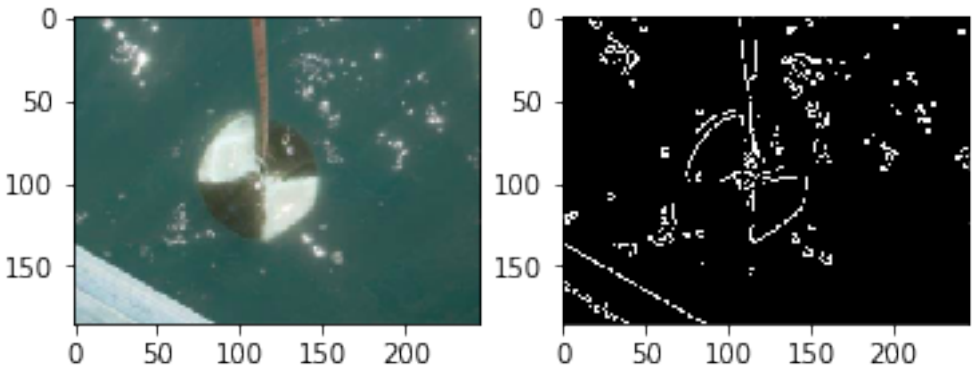


Figure 13: Edge Detection 1, `find_edge(img1)`

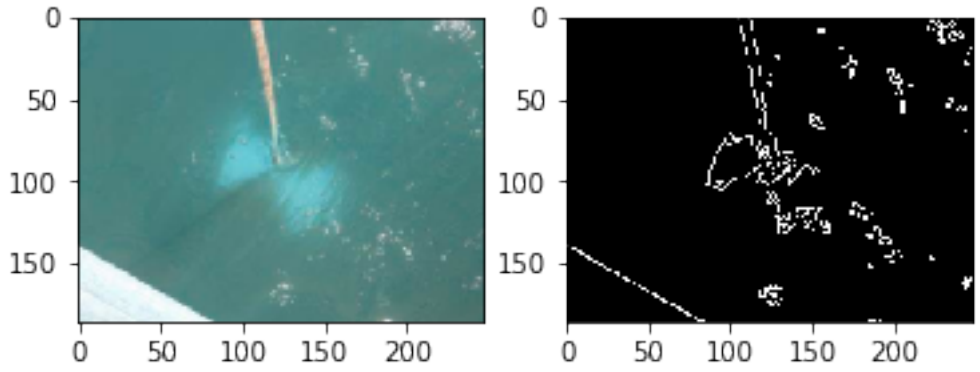


Figure 14: Edge Detection 2, `find_edge(img2)`

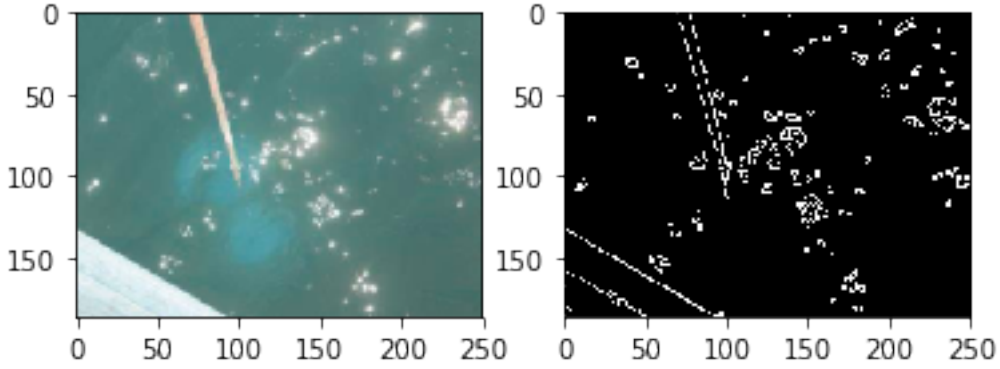


Figure 15: Edge Detection 3, `find_edge(img3)`

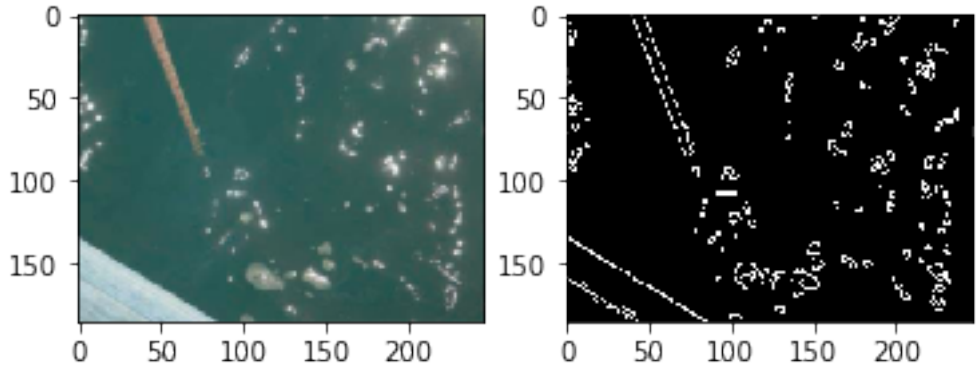


Figure 16: Edge Detection 4, , `find_edge(img4)`

7.0.6.3.3 Black and white

```
bw1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
plt.imshow(bw1, cmap='gray')
```

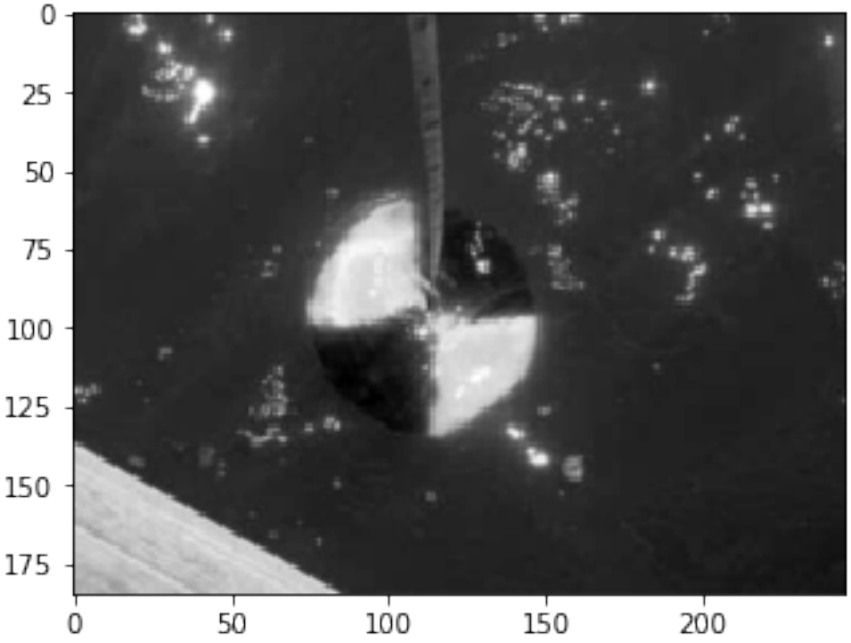


Figure 17: Back White conversion

8 DATA

8.0.1 Data Formats

8.0.1.1 YAML The term *YAML* stand for "YAML Ainot Markup Language". According to the Web Page at

- http://yaml.org/

"YAML is a human friendly data serialization standard for all programming languages." There are multiple versions of YAML existing and one needs to take care of that your software supports the right version. The current version is YAML 1.2.

YAML is often used for configuration and in many cases can also be used as XML replacement. Important is tat YAM in contrast to XML removes the tags while replacing them with indentation. This has naturally the advantage that it is mor easily to read, however, the format is strict and needs to adhere to proper

indentation. Thus it is important that you check your YAML files for correctness, either by writing for example a python program that read your yaml file, or an online YAML checker such as provided at

- http://www.yamllint.com/

An example on how to use yaml in python is provided in our next example. Please note that YAML is a superset of JSON. Originally YAML was designed as a markup language. However as it is not document oriented but data oriented it has been recast and it does no longer classify itself as markup language.

```
import os
import sys
import yaml

try:
    yamlFilename = os.sys.argv[1]
    yamlFile = open(yamlFilename, "r")
except:
    print("filename does not exist")
    sys.exit()
try:
   yaml.load(yamlFile.read())
except:
   print("YAML file is not valid.")
```

Resources:

- http://yaml.org/
- https://en.wikipedia.org/wiki/YAML
- http://www.yamllint.com/

8.0.1.2 JSON The term JSON stand for *JavaScript Object Notation*. It is targeted as an open-standard file format that emphasizes on integration of human-readable text to transmit data objects. The data objects contain attribute value pairs. Although it originates from JavaScript, the format itself is language independent. It uses brackets to allow organization of the data. PLease note that YAML is a superset of JSON and not all YAML documents can be converted to JSON. Furthermore JSON does not support comments. For these reasons we often prefer to us YAMl instead of JSON. However JSON data can easily be translated to YAML as well as XML.

Resources:

- https://en.wikipedia.org/wiki/JSON

- https://www.json.org/

8.0.1.3 XML XML stands for *Extensible Markup Language*. XML allows to define documents with the help of a set of rules in order to make it machine readable. The emphasize here is on machine readable as document in XML can become quickly complex and difficult to understand for humans. XML is used for documents as well as data structures.

A tutorial about XML is available at

- https://www.w3schools.com/xml/default.asp

Resources:

- https://en.wikipedia.org/wiki/XML

9 MONGO

9.0.1 MongoDB in Python

Learning Objectives

- Introduction to basic MongoDB knowledge
- Use of MongoDB via PyMongo
- Use of MongoEngine MongoEngine and Object-Document mapper,
- Use of Flask-Mongo

In today's era, NoSQL databases have developed an enormous potential to process the unstructured data efficiently. Modern information is complex, extensive, and may not have pre-existing relationships. With the advent of the advanced search engines, machine learning, and Artificial Intelligence, technology expectations to process, store, and analyze such data have grown tremendously [2]. The NoSQL database engines such as MongoDB, Redis, and Cassandra have successfully overcome the traditional relational database challenges such as scalability, performance, unstructured data growth, agile sprint cycles, and growing needs of processing data in real-time with minimal hardware processing power [3]. The NoSQL databases are a new generation of engines that do not necessarily require SQL language and

are sometimes also called *Not Only SQL* databases. However, most of them support various third-party open connectivity drivers that can map NoSQL queries to SQL's. It would be safe to say that although NoSQL databases are still far from replacing the relational databases, they are adding an immense value when used in hybrid IT environments in conjunction with relational databases, based on the application specific needs [3]. We will be covering the MongoDB technology, its driver PyMongo, its object-document mapper MongoEngine, and the Flask-PyMongo micro-web framework that make MongoDB more attractive and user-friendly.

9.0.1.1 Cloudmesh MongoDB Usage Quickstart

Before you read on we like you to read this quickstart. The easiest way for many of the activities we do to interact with MongoDB is to use our cloudmesh functionality. This prelude section is not intended to describe all the details, but get you started quickly while leveraging cloudmesh

This is done via the cloudmesh cmd5 and the cloudmesh_community/cm code:

- https://cloudmesh-community.github.io/cm/

To install mongo on for example macOS you can use

```
$ cms admin mongo install
```

To start, stop and see the status of mongo you can use

```
$ cms admin mongo start
$ cms admin mongo stop
$ cms admin mongo status
```

To add an object to Mongo, you simply have to define a dict with predefined values for `kind` and `cloud` . In future such attributes can be passed to the function to determine the MongoDB collection.

```
from cloudmesh.mongo.DataBaseDecorator import DatabaseUpdate

@DatabaseUpdate
def test():
  data ={
    "kind": "test",
    "cloud": "testcloud",
    "value": "hello"
  }
```

```
  return data

result = test()
```

When you invoke the function it will automatically store the information into MongoDB. Naturally this requires that the `~/.cloudmesh/cloudmesh.yaml` file is properly configured.

9.0.1.2 MongoDB Today MongoDB is one of leading NoSQL database which is fully capable of handling dynamic changes, processing large volumes of complex and unstructured data, easily using object-oriented programming features; as well as distributed system challenges [4]. At its core, MongoDB is an open source, cross-platform, document database mainly written in C++ language.

9.0.1.2.1 Installation MongoDB can be installed on various Unix Platforms, including Linux, Ubuntu, Amazon Linux, etc [5]. This section focuses on installing MongoDB on Ubuntu 18.04 Bionic Beaver used as a standard OS for a virtual machine used as a part of Big Data Application Class during the 2018 Fall semester.

Installation procedure

Before installing, it is recommended to configure the non-root user and provide the administrative privileges to it, in order to be able to perform general MongoDB admin tasks. This can be accomplished by login as the root user in the following manner [6].

```
$ adduser mongoadmin
$ usermod -aG sudo sammy
```

When logged in as a regular user, one can perform actions with superuser privileges by typing *sudo* before each command [6].

Once the user set up is completed, one can login as a regular user (mongoadmin) and use the following instructions to install MongoDB.

To update the Ubuntu packages to the most recent versions, use the next command:

```
$ sudo apt update
```

To install the MongoDB package:

```
$ sudo apt install -y mongodb
```

To check the service and database status:

```
$ sudo systemctl status mongodb
```

Verifying the status of a successful MongoDB installation can be confirmed with an output similar to this:

```
$ mongodb.service - An object/document-oriented database
    Loaded: loaded (/lib/systemd/system/mongodb.service; enabled;
       ↪ vendor preset: enabled)
    Active: **active** (running) since Sat 2018-11-15 07:48:04 UTC; 2
       ↪ min 17s ago
      Docs: man:mongod(1)
  Main PID: 2312 (mongod)
     Tasks: 23 (limit: 1153)
    CGroup: /system.slice/mongodb.service
            - 2312 /usr/bin/mongod --unixSocketPrefix=/run/mongodb --
               ↪ config /etc/mongodb.conf
```

To verify the configuration, more specifically the installed version, server, and port, use the following command:

```
$ mongo --eval 'db.runCommand({ connectionStatus: 1 })'
```

Similarly, to restart MongoDB, use the following:

```
$ sudo systemctl restart mongodb
```

To allow access to MongoDB from an outside hosted server one can use the following command which opens the fire-wall connections [5].

```
$ sudo ufw allow from your_other_server_ip/32 to any port 27017
```

Status can be verified by using:

```
$ sudo ufw status
```

Other MongoDB configurations can be edited through the */etc/mongodb.conf* files such as port and hostnames, file paths.

```
$ sudo nano /etc/mongodb.conf
```

Also, to complete this step, a server's IP address must be added to the bindIP value [5].

```
$ logappend=true

  bind_ip = 127.0.0.1,your_server_ip
  *port = 27017*
```

MongoDB is now listening for a remote connection that can be accessed by anyone with appropriate credentials [5].

9.0.1.2.2 Collections and Documents Each database within Mongo environment contains collections which in turn contain documents. Collections and documents are analogous to tables and rows respectively to the relational databases. The document structure is in a key-value form which allows storing of complex data types composed out of field and value pairs. Documents are objects which correspond to native data types in many programming languages, hence a well defined, embedded document can help reduce expensive joins and improve query performance. The `_id` field helps to identify each document uniquely [3].

MongoDB offers flexibility to write records that are not restricted by column types. The data storage approach is flexible as it allows one to store data as it grows and to fulfill varying needs of applications and/or users. It supports JSON like binary points known as BSON where data can be stored without specifying the type of data. Moreover, it can be distributed to multiple machines at high speed. It includes a sharding feature that partitions and spreads the data out across various servers. This makes MongoDB an excellent choice for cloud data processing. Its utilities can load high volumes of data at high speed which ultimately provides greater flexibility and availability in a cloud-based environment [2].

The dynamic schema structure within MongoDB allows easy testing of the small sprints in the Agile project management life cycles and research projects that require frequent changes to the data structure with minimal downtime. Contrary to this flexible process, modifying the data structure of relational databases can be a very tedious process [2].

Collection example

The following collection example for a person named *Albert* includes additional information such as age, status, and group [7].

```
{
 name: "Albert"
 age: "21"
 status: "Open"
```

```
 group: ["AI" , "Machine Learning"]
}
```

Document structure

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

Collection Operations

If collection does not exists, MongoDB database will create a collection by default.

```
> db.myNewCollection1.insertOne( { x: 1 } )
> db.myNewCollection2.createIndex( { y: 1 } )
```

9.0.1.2.3 MongoDB Querying The data retrieval patterns, the frequency of data manipulation statements such as insert, updates, and deletes may demand for the use of indexes or incorporating the sharding feature to improve query performance and efficiency of MongoDB environment [3]. One of the significant difference between relational databases and NoSQL databases are joins. In the relational database, one can combine results from two or more tables using a common column, often called as *key*. The native table contains the *primary key* column while the referenced table contains a *foreign key*. This mechanism allows one to make changes in a single row instead of changing all rows in the referenced table. This action is referred to as *normalization*. MongoDB is a document database and mainly contains denormalized data which means the data is repeated instead of indexed over a specific key. If the same data is required in more than one table, it needs to be repeated. This constraint has been eliminated in MongoDB's new version 3.2. The new release introduced a *$lookup* feature which more likely works as a left-outer-join. Lookups are restricted to aggregated functions which means that data usually need some type of filtering and grouping operations to be conducted beforehand. For this reason, joins in MongoDB require more complicated querying compared to the traditional relational database joins. Although at this time, *lookups* are still very far from replacing *joins*, this is a prominent feature that can resolve some of the relational data challenges for MongoDB [8]. MongoDB queries support regular expressions as well as range asks for specific fields that eliminate the need of returning entire documents [3]. MongoDB collections do not enforce document structure like SQL

databases which is a compelling feature. However, it is essential to keep in mind the needs of the applications[2].

Mongo Queries examples

The queries can be executed from Mongo shell as well as through scripts.

To query the data from a MongoDB collection, one would use MongoDB's *find()* method.

```
> db.COLLECTION_NAME.find()
```

The output can be formatted by using the *pretty()* command.

```
> db.mycol.find().pretty()
```

The MongoDB insert statements can be performed in the following manner:

```
> db.COLLECTION_NAME.insert(document)
```

> "The *$lookup* command performs a left-outer-join to an unsharded collection in the same database to filter in documents from the *joined* collection for processing" [9].

```
$ {
    $lookup:
      {
        from: <collection to join>,
        localField: <field from the input documents>,
        foreignField: <field from the documents of the "from"
           ↪ collection>,
        as: <output array field>
      }
  }
```

This operation is equivalent to the following SQL operation:

```
 $ SELECT *, <output array field>
   FROM collection
   WHERE <output array field> IN (SELECT *
                                  FROM <collection to join>
                                  WHERE <foreignField> = <collection.
                                     ↪ localField>);`
```

To perform a Like Match (Regex), one would use the following command:

```
> db.products.find( { sku: { $regex: /789$/ } } )
```

9.0.1.2.4 MongoDB Basic Functions When it comes to the technical elements of MongoDB, it posses a rich interface for importing and storage of external data in various formats. By using the Mongo *Import/Export* tool, one can easily transfer contents from JSON, CSV, or TSV files into a database. MongoDB supports CRUD (create, read, update, delete) operations efficiently and has detailed documentation available on the product website. It can also query the geospatial data, and it is capable of storing geospatial data in GeoJSON objects. The *aggregation* operation of the MongoDB process data records and returns computed results. MongoDB aggregation framework is modeled on the concept of data pipelines [10].

Import/Export functions examples

To import JSON documents, one would use the following command:

```
$ mongoimport --db users --collection contacts --file contacts.json
```

The CSV import uses the input file name to import a collection, hence, the collection name is optional [10].

```
$ mongoimport --db users --type csv --headerline --file /opt/backups/
   ↪ contacts.csv
```

> "*Mongoexport* is a utility that produces a JSON or CSV export of data stored in a MongoDB instance" [10].

```
$ mongoexport --db test --collection traffic --out traffic.json
```

9.0.1.2.5 Security Features Data security is a crucial aspect of the enterprise infrastructure management and is the reason why MongoDB provides various security features such as ole based access control, numerous authentication options, and encryption. It supports mechanisms such as SCRAM, LDAP, and Kerberos authentication. The administrator can create role/collection-based access control; also roles can be predefined or custom. MongoDB can audit activities such as DDL, CRUD statements, authentication and authorization operations [11].

Collection based access control example

A user defined role can contain the following privileges [11].

```
$ privileges: [
   { resource: { db: "products", collection: "inventory" }, actions:
      ↪ [ "find", "update"] },
   { resource: { db: "products", collection: "orders" },  actions: [
      ↪ "find" ] }
 ]
```

9.0.1.2.6 MongoDB Cloud Service In regards to the cloud technologies, MongoDB also offers fully automated cloud service called *Atlas* with competitive pricing options. Mongo Atlas Cloud interface offers interactive GUI for managing cloud resources and deploying applications quickly. The service is equipped with geographically distributed instances to ensure no single point failure. Also, a well-rounded performance monitoring interface allows users to promptly detect anomalies and generate index suggestions to optimize the performance and reliability of the database. Global technology leaders such as Google, Facebook, eBay, and Nokia are leveraging MongoDB and *Atlas* cloud services making MongoDB one of the most popular choices among the NoSQL databases [12].

9.0.1.3 PyMongo PyMongo is the official Python driver or distribution that allows work with a NoSQL type database called *MongoDB* [13]. The first version of the driver was developed in 2009 [14], only two years after the development of MongoDB was started. This driver allows developers to combine both Python's versatility and MongoDB's flexible schema nature into successful applications. Currently, this driver supports MongoDB versions 2.6, 3.0, 3.2, 3.4, 3.6, and 4.0 [15]. MongoDB and Python represent a compatible fit considering that BSON (binary JSON) used in this NoSQL database is very similar to Python dictionaries, which makes the collaboration between the two even more appealing [16]. For this reason, dictionaries are the recommended tools to be used in PyMongo when representing documents [17].

9.0.1.3.1 Installation Prior to being able to exploit the benefits of Python and MongoDB simultaneously, the PyMongo distribution must be installed using *pip*. To install it on all platforms, the following command should be used [18]:

```
$ python -m pip install pymongo
```

Specific versions of PyMongo can be installed with command lines such as in our example where the 3.5.1 version is installed [18].

```
$ python -m pip install pymongo==3.5.1
```

A single line of code can be used to upgrade the driver as well [18].

```
$ python -m pip install --upgrade pymongo
```

Furthermore, the installation process can be completed with the help of the *easy_install* tool, which requires users to use the following command [18].

```
$ python -m easy_install pymongo
```

To do an upgrade of the driver using this tool, the following command is recommended [18]:

```
$ python -m easy_install -U pymongo
```

There are many other ways of installing PyMongo directly from the source, however, they require for C extension dependencies to be installed prior to the driver installation step, as they are the ones that skim through the sources on GitHub and use the most up-to-date links to install the driver [18].

To check if the installation was completed accurately, the following command is used in the Python console [19].

```
import pymongo
```

If the command returns zero exceptions within the Python shell, one can consider for the PyMongo installation to have been completed successfully.

9.0.1.3.2 Dependencies The PyMongo driver has a few dependencies that should be taken into consideration prior to its usage. Currently, it supports CPython 2.7, 3.4+, PyPy, and PyPy 3.5+ interpreters [15]. An optional dependency that requires some additional components to be installed is the GSSAPI authentication [15]. For the Unix based machines, it requires *pykerberos*, while for the Windows machines *WinKerberos* is needed to fullfill this requirement [15]. The automatic installation of this dependency can be done simultaneously with the driver installation, in the following manner:

```
$ python -m pip install pymongo[gssapi]
```

Other third-party dependencies such as *ipaddress*, *certifi*, or *wincerstore* are necessary for connections with help of TLS/SSL and can also be simultaneously installed along with the driver installation [15].

9.0.1.3.3 Running PyMongo with Mongo Deamon Once PyMongo is installed, the Mongo deamon can be run with a very simple command in a new terminal window [19].

```
$ mongod
```

9.0.1.3.4 Connecting to a database using MongoClient In order to be able to establish a connection with a database, a MongoClient class needs to be imported, which sub-sequentially allows the MongoClient object to communicate with the database [19].

```
from pymongo import MongoClient
client = MongoClient()
```

This command allows a connection with a default, local host through port 27017, however, depending on the programming requirements, one can also specify those by listing them in the client instance or use the same information via the Mongo URI format [19].

9.0.1.3.5 Accessing Databases Since MongoClient plays a server role, it can be used to access any desired databases in an easy way. To do that, one can use two different approaches. The first approach would be doing this via the *attribute* method where the name of the desired database is listed as an attribute, and the second approach, which would include a dictionary-style access [19]. For example, to access a database called *cloudmesh_community*, one would use the following commands for the attribute and for the dictionary method, respectively.

```
db = client.cloudmesh_community
db = client['cloudmesh_community']
```

9.0.1.3.6 Creating a Database Creating a database is a straight forward process. First, one must create a MongoClient object and specify the connection (IP address) as well as the name of the database they are trying to create [20]. The example of this command is presented in the followng section:

```
import pymongo
client = pymongo.MongoClient('mongodb://localhost:27017/')
db = client['cloudmesh']
```

9.0.1.3.7 Inserting and Retrieving Documents (Querying) Creating documents and storing data using PyMongo is equally easy as accessing and creating databases. In order to add new data, a collection must be specified first. In this example, a decision is made to use the *cloudmesh* group of documents.

```
cloudmesh = db.cloudmesh
```

Once this step is completed, data may be inserted using the *insert_one()* method, which means that only one document is being created. Of course, insertion of multiple documents at the same time is possible as well with use of the *insert_many()* method [19]. An example of this method is as follows:

```
course_info = {
     'course': 'Big Data Applications and Analytics',
     'instructor': ' Gregor von Laszewski',
     'chapter': 'technologies'
}
result = cloudmesh.insert_one(course_info)`
```

Another example of this method would be to create a collection. If we wanted to create a collection of students in the *cloudmesh_community*, we would do it in the following manner:

```
student = [ {'name': 'John', 'st_id': 52642},
    {'name': 'Mercedes', 'st_id': 5717},
    {'name': 'Anna', 'st_id': 5654},
    {'name': 'Greg', 'st_id': 5423},
    {'name': 'Amaya', 'st_id': 3540},
    {'name': 'Cameron', 'st_id': 2343},
    {'name': 'Bozer', 'st_id': 4143},
    {'name': 'Cody', 'price': 2165} ]

client = MongoClient('mongodb://localhost:27017/')

with client:
    db = client.cloudmesh
    db.students.insert_many(student)
```

Retrieving documents is equally simple as creating them. The *find_one()* method can be used to retrieve one document [19]. An implementation of this method is given in the following example.

```
gregors_course = cloudmesh.find_one({'instructor':'Gregor von
   ↪ Laszewski'})
```

Similarly, to retieve multiple documents, one would use the *find()* method instead of the *find_one()*. For example, to find all courses thought by professor von Laszewski, one would use the following command:

```
gregors_course = cloudmesh.find({'instructor':'Gregor von Laszewski'
    ↪ })
```

One thing that users should be cognizant of when using the *find()* method is that it does not return results in an array format but as a *cursor* object, which is a combination of methods that work together to help with data querying [19]. In order to return individual documents, iteration over the result must be completed [19].

9.0.1.3.8 Limiting Results When it comes to working with large databases it is always useful to limit the number of query results. PyMongo supports this option with its *limit()* method [20]. This method takes in one parameter which specifies the number of documents to be returned [20]. For example, if we had a collection with a large number of cloud technologies as individual documents, one could modify the query results to return only the top 10 technologies. To do this, the following example could be utilized:

```
client = pymongo.MongoClient('mongodb://localhost:27017/')
    db = client['cloudmesh']
    col = db['technologies']
    topten = col.find().limit(10)
```

9.0.1.3.9 Updating Collection Updating documents is very similar to inserting and retrieving the same. Depending on the number of documents to be updated, one would use the *update_one()* or *update_many()* method [20]. Two parameters need to be passed in the *update_one()* method for it to successfully execute. The first argument is the query object that specifies the document to be changed, and the second argument is the object that specifies the new value in the document. An example of the *update_one()* method in action is the following:

```
myquery = { 'course': 'Big Data Applications and Analytics' }
newvalues = { '$set': { 'course': 'Cloud Computing' } }
```

Updating all documents that fall under the same criteria can be done with the *update_many* method [20]. For example, to update all documents in which course title starts with letter *B* with a different instructor information, we would do the following:

```
client = pymongo.MongoClient('mongodb://localhost:27017/')
db = client['cloudmesh']
col = db['courses']
query = { 'course': { '$regex': '^B' } }
```

```
newvalues = { '$set': { 'instructor': 'Gregor von Laszewski' } }

edited = col.update_many(query, newvalues)
```

9.0.1.3.10 Counting Documents Counting documents can be done with one simple operation called *count_documents()* instead of using a full query [21]. For example, we can count the documents in the *cloudmesh_commpunity* by using the following command:

```
cloudmesh = count_documents({})
```

To create a more specific count, one would use a command similar to this:

```
cloudmesh = count_documents({'author': 'von Laszewski'})
```

This technology supports some more advanced querying options as well. Those advanced queries allow one to add certain contraints and narrow down the results even more. For example, to get the courses thought by professor von Laszewski after a certain date, one would use the following command:

```
d = datetime.datetime(2017, 11, 12, 12)
for course in cloudmesh.find({'date': {'$lt': d}}).sort('author'):
    pprint.pprint(course)
```

9.0.1.3.11 Indexing Indexing is a very important part of querying. It can greately improve query performance but also add functionality and aide in storing documents [21].

> "To create a unique index on a key that rejects documents whose value for that key already exists in the index" [21].

We need to firstly create the index in the following manner:

```
result = db.profiles.create_index([('user_id', pymongo.ASCENDING)],
unique=True)
sorted(list(db.profiles.index_information()))
```

This command acutally creates two different indexes. The first one is the *_id* , created by MongoDB automatically, and the second one is the *user_id*, created by the user.

The purpose of those indexes is to cleverly prevent future additions of invalid *user_ids* into a collection.

9.0.1.3.12 Sorting Sorting on the server-side is also avaialable via MongoDB. The PyMongo *sort()* method is equivalent to the SQL *order by* statement and it can be performed as *pymongo.ascending* and *pymongo.descending* [22]. This method is much more efficient as it is being completed on the server-side, compared to the sorting completed on the client side. For example, to return all users with first name *Gregor* sorted in descending order by birthdate we would use a command such as this:

```
users = cloudmesh.users.find({'firstname':'Gregor'}).sort(('
   ↪ dateofbirth', pymongo.DESCENDING))
for user in users:
   print user.get('email')
```

9.0.1.3.13 Aggregation Aggregation operations are used to process given data and produce summarized results. Aggregation operations collect data from a number of documents and provide collective results by grouping data. PyMongo in its documentation offers a separate framework that supports data aggregation. This aggregation framework can be used to

> "provide projection capabilities to reshape the returned data" [23].

In the aggregation pipeline, documents pass through multiple pipeline stages which convert documents into result data. The basic pipeline stages include filters. Those filters act like document transformation by helping change the document output form. Other pipelines help group or sort documents with specific fields. By using native operations from MongoDB, the pipeline operators are efficient in aggregating results.

The *addFields* stage is used to add new fields into documents. It reshapes each document in stream, similarly to the *project* stage. The output document will contain existing fields from input documents and the newly added fields [24]]. The following example shows how to add *student details* into a document.

```
 db.cloudmesh_community.aggregate([
{
       $addFields: {
       "document.StudentDetails": {
       $concat:['$document.student.FirstName', '$document.student.
          ↪ LastName']
           }
       }
   } ])
```

The *bucket* stage is used to categorize incoming documents into groups based on specified expressions. Those groups are called *buckets* [24]. The following example shows the *bucket* stage in action.

```
db.user.aggregate([
{ "$group": {
  "_id": {
    "city": "$city",
    "age": {
      "$let": {
        "vars": {
 "age": { "$subtract" :[{ "$year": new Date() },{ "$year": "$birthDay
    ↪ " }] }},
        "in": {
          "$switch": {
            "branches": [
              { "case": { "$lt": [ "$$age", 20 ] }, "then": 0 },
              { "case": { "$lt": [ "$$age", 30 ] }, "then": 20 },
              { "case": { "$lt": [ "$$age", 40 ] }, "then": 30 },
              { "case": { "$lt": [ "$$age", 50 ] }, "then": 40 },
              { "case": { "$lt": [ "$$age", 200 ] }, "then": 50 }
            ] }  }  } } },
  "count": { "$sum": 1 }}})
```

In the *bucketAuto* stage, the boundaries are automatically determined in an attempt to evenly distribute documents into a specified number of buckets. In the following operation, input documents are grouped into four buckets according to the values in the price field [24].

```
db.artwork.aggregate( [
  {
    $bucketAuto: {
        groupBy: "$price",
        buckets: 4
    }
  }
 ] )
```

The *collStats* stage returns statistics regarding a collection or view [24].

```
db.matrices.aggregate( [ { $collStats: { latencyStats: { histograms:
   ↪ true } }
```

```
 } ] )
```

The *count* stage passes a document to the next stage that contains the number documents that were input to the stage [24].

```
db.scores.aggregate(  [    {
   $match: {        score: {          $gt: 80    } }  },
 {      $count: "passing_scores"  } ])
```

The *facet* stage helps process multiple aggregation pipelines in a single stage [24].

```
db.artwork.aggregate( [ {
   $facet: {  "categorizedByTags": [   { $unwind: "$tags" },
       { $sortByCount: "$tags" }  ],  "categorizedByPrice": [
       // Filter out documents without a price e.g., _id: 7
       { $match: { price: { $exists: 1 } } },
      { $bucket: { groupBy: "$price",
          boundaries: [  0, 150, 200, 300, 400 ],
          default: "Other",
          output: { "count": { $sum: 1 },
            "titles": { $push: "$title" }
          } }        }], "categorizedByYears(Auto)": [
      { $bucketAuto: { groupBy: "$year",buckets: 4 }
      } ]}}])
```

The *geoNear* stage returns an ordered stream of documents based on the proximity to a geospatial point. The output documents include an additional distance field and can include a location identifier field [24].

```
db.places.aggregate([
 {    $geoNear: {
      near: { type: "Point", coordinates: [ -73.99279 , 40.719296 ]
         ↪ },
      distanceField: "dist.calculated",
      maxDistance: 2,
      query: { type: "public" },
      includeLocs: "dist.location",
      num: 5,
      spherical: true
```

```
   }  }])
```

The *graphLookup* stage performs a recursive search on a collection. To each output document, it adds a new array field that contains the traversal results of the recursive search for that document [24].

```
db.travelers.aggregate( [
 {
    $graphLookup: {
       from: "airports",
       startWith: "$nearestAirport",
       connectFromField: "connects",
       connectToField: "airport",
       maxDepth: 2,
       depthField: "numConnections",
       as: "destinations"
    }
 }
] )
```

The *group* stage consumes the document data per each distinct group. It has a RAM limit of 100 MB. If the stage exceeds this limit, the *group* produces an error [24].

```
db.sales.aggregate(
 [
    {
      $group : {
         _id : { month: { $month: "$date" }, day: { $dayOfMonth: "
            ↪ $date" },
         year: { $year: "$date" } },
         totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] }
            ↪  },
         averageQuantity: { $avg: "$quantity" },
         count: { $sum: 1 }
       }
    }
 ]
)
```

The *indexStats* stage returns statistics regarding the use of each index for a collection [24].

```
db.orders.aggregate( [ { $indexStats: { } } ] )
```

The *limit* stage is used for controlling the number of documents passed to the next stage in the pipeline [24].

```
db.article.aggregate(
  { $limit : 5 }
)
```

The *listLocalSessions* stage gives the session information currently connected to mongos or mongod instance [24].

```
db.aggregate( [  { $listLocalSessions: { allUsers: true } } ] )
```

The *listSessions* stage lists out all session that have been active long enough to propagate to the *system.sessions* collection [24].

```
 use config
 db.system.sessions.aggregate( [  { $listSessions: { allUsers: true }
    ↪  } ] )
```

The *lookup* stage is useful for performing outer joins to other collections in the same database [24].

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from"
          ↪ collection>,
       as: <output array field>
     }
}
```

The *match* stage is used to filter the document stream. Only matching documents pass to next stage [24].

```
db.articles.aggregate(
    [ { $match : { author : "dave" } } ]
)
```

The *project* stage is used to reshape the documents by adding or deleting the fields.

```
db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] )
```

The *redact* stage reshapes stream documents by restricting information using information stored in documents themselves [24].

```
  db.accounts.aggregate(
  [
    { $match: { status: "A" } },
    {
      $redact: {
        $cond: {
          if: { $eq: [ "$level", 5 ] },
          then: "$$PRUNE",
          else: "$$DESCEND"
        }      }    }  ]);
```

The *replaceRoot* stage is used to replace a document with a specified embedded document [24].

```
  db.produce.aggregate( [
   {
     $replaceRoot: { newRoot: "$in_stock" }
   }
] )
```

The *sample* stage is used to sample out data by randomly selecting number of documents form input [24].

```
  db.users.aggregate(
   [ { $sample: { size: 3 } } ]
)
```

The *skip* stage skips specified initial number of documents and passes remaining documents to the pipeline [24].

```
db.article.aggregate(
   { $skip : 5 }
);
```

The *sort* stage is useful while reordering document stream by a specified sort key [24].

```
db.users.aggregate(
   [
     { $sort : { age : -1, posts: 1 } }
   ]
)
```

The *sortByCounts* stage groups the incoming documents based on a specified expression value and counts documents in each distinct group [24].

```
db.exhibits.aggregate(
[ { $unwind: "$tags" },  { $sortByCount: "$tags" } ] )
```

The *unwind* stage deconstructs an array field from the input documents to output a document for each element [24].

```
db.inventory.aggregate( [ { $unwind: "$sizes" } ] )
db.inventory.aggregate( [ { $unwind: { path: "$sizes" } } ] )
```

The *out* stage is used to write aggregation pipeline results into a collection. This stage should be the last stage of a pipeline [24].

```
db.books.aggregate( [
                  { $group : { _id : "$author", books: { $push: "
                     ↪ $title" } } },
                     { $out : "authors" }
                  ] )
```

Another option from the *aggregation operations* is the Map/Reduce framework, which essentially includes two different functions, *map* and *reduce*. The first one provides the key value pair for each tag in the array, while the latter one

> “sums over all of the emitted values for a given key” [23].

The last step in the Map/Reduce process it to call the *map_reduce()* function and iterate over the results [23]. The Map/Reduce operation provides result data in a collection or returns results in-line. One can perform subsequent operations with the same input collection if the output of the same is written to a collection [25]. An operation that produces results in a in-line form must provide results with in the BSON document size limit. The current limit for a BSON document is 16 MB. These types of

operations are not supported by views [25]. The PyMongo's API supports all features of the MongoDB's Map/Reduce engine [26]. Moreover, Map/Reduce has the ability to get more detailed results by passing *full_response=True* argument to the *map_reduce()* function [26].

9.0.1.3.14 Deleting Documents from a Collection The deletion of documents with PyMongo is fairly straight forward. To do so, one would use the *remove()* method of the PyMongo Collection object [22]. Similarly to the reads and updates, specification of documents to be removed is a must. For example, removal of the entire document collection with a score of 1, would required one to use the following command:

```
cloudmesh.users.remove({"score":1, safe=True})
```

The *safe* parameter set to *True* ensures the operation was completed [22].

9.0.1.3.15 Copying a Database Copying databases within the same mongod instance or between different mongod servers is made possible with the *command()* method after connecting to the desired mongod instance [27]. For example, to copy the *cloudmesh* database and name the new database *cloudmesh_copy*, one would use the *command()* method in the following manner:

```
client.admin.command('copydb',
                     fromdb='cloudmesh',
                     todb='cloudmesh_copy')
```

There are two ways to copy a database between servers. If a server is not password-prodected, one would not need to pass in the credentials nor to authenticate to the admin database [27]. In that case, to copy a database one would use the following command:

```
client.admin.command('copydb',
                     fromdb='cloudmesh',
                     todb='cloudmesh_copy',
                     fromhost='source.example.com')
```

On the other hand, if the server where we are copying the database to is protected, one would use this command instead:

```
client = MongoClient('target.example.com',
                     username='administrator',
                     password='pwd')
client.admin.command('copydb',
```

```
                    fromdb='cloudmesh',
                    todb='cloudmesh_copy',
                    fromhost='source.example.com')
```

9.0.1.3.16 PyMongo Strengths One of PyMongo strengths is that allows document creation and querying natively

> "through the use of existing language features such as nested dictionaries and lists" [22].

For moderately experienced Python developers, it is very easy to learn it and quickly feel comfortable with it.

> "For these reasons, MongoDB and Python make a powerful combination for rapid, iterative development of horizontally scalable backend applications" [22].

According to [22], MongoDB is very applicable to modern applications, which makes PyMongo equally valuable [22].

9.0.1.4 MongoEngine

> "MongoEngine is an Object-Document Mapper, written in Python for working with MongoDB" [28].

It is actually a library that allows a more advanced communication with MongoDB compared to PyMongo. As MongoEngine is technically considered to be an object-document mapper(ODM), it can also be considered to be

> "equivalent to a SQL-based object relational mapper(ORM)" [19].

The primary technique why one would use an ODM includes *data conversion* between computer systems that are not compatible with each other [29]. For the purpose of converting data to the appropriate form, a *virtual object database* must be created within the utilized programming language [29]. This library is also used to define schemata for documents within MongoDB, which ultimately helps with minimizing coding errors as well defining methods on existing fields [30]. It is also very beneficial to the overall workflow as it tracks changes made to the documents and aids in the document saving process [31].

9.0.1.4.1 Installation The installation process for this technology is fairly simple as it is considered to be a library. To install it, one would use the following command [32]:

```
$ pip install mongoengine
```

A *bleeding-edge* version of MongoEngine can be installed directly from GitHub by first cloning the repository on the local machine, virtual machine, or cloud.

9.0.1.4.2 Connecting to a database using MongoEngine Once installed, MongoEngine needs to be connected to an instance of the mongod, similarly to PyMongo [33]. The *connect()* function must be used to successfully complete this step and the argument that must be used in this function is the name of the desired database [33]. Prior to using this function, the function name needs to be imported from the MongoEngine library.

```
from mongoengine import connect
connect('cloudmesh_community')
```

Similarly to the MongoClient, MongoEngine uses the local host and port 27017 by default, however, the *connect()* function also allows specifying other hosts and port arguments as well [33].

```
connect('cloudmesh_community', host='196.185.1.62', port=16758)
```

Other types of connections are also supported (i.e. URI) and they can be completed by providing the URI in the *connect()* function [33].

9.0.1.4.3 Querying using MongoEngine To query MongoDB using MongoEngine an *objects attribute* is used, which is, technically, a part of the document class [34]. This attribute is called the *QuerySetManager* which in return

> "creates a new *QuerySet* object on access" [34].

To be able to access individual documents from a database, this object needs to be iterated over. For example, to return/print all students in the *cloudmesh_community* object (database), the following command would be used.

```
for user in cloudmesh_community.objects:
   print cloudmesh_community.student
```

MongoEngine also has a capability of query filtering which means that a keyword can be used within the called *QuerySet* object to retrieve specific information [34]. Let us say one would like to iterate over *cloudmesh_community* students that are natives of Indiana. To achieve this, one would use the following command:

```
indy_students = cloudmesh_community.objects(state='IN')
```

This library also allows the use of all operators except for the equality operator in its queries, and moreover, has the capability of handling *string queries*, *geo queries*, *list querying*, and querying of the raw PyMongo queries [34].

The string queries are useful in performing text operations in the conditional queries. A query to find a document exactly matching and with state *ACTIVE* can be performed in the following manner:

```
db.cloudmesh_community.find( State.exact("ACTIVE") )
```

The query to retrieve document data for names that start with a case sensitive *AL* can be written as:

```
db.cloudmesh_community.find( Name.startswith("AL") )
```

To perform an exact same query for the non-key-sensitive *AL* one would use the following command:

```
db.cloudmesh_community.find( Name.istartswith("AL") )
```

The MongoEngine allows data extraction of geographical locations by using Geo queries. The *geo_within* operator checks if a geometry is within a polygon.

```
  cloudmesh_community.objects(
            point__geo_within=[[[40, 5], [40, 6], [41, 6], [40, 5]]])
  cloudmesh_community.objects(
            point__geo_within={"type": "Polygon",
                 "coordinates": [[[40, 5], [40, 6], [41, 6], [40,
                    ↪ 5]]]})
```

The list query looks up the documents where the specified fields matches exactly to the given value. To match all pages that have the word *coding* as an item in the *tags* list one would use the following query:

```
  class Page(Document):
     tags = ListField(StringField())

  Page.objects(tags='coding')
```

Overall, it would be safe to say that MongoEngine has good compatibility with Python. It provides different functions to utilize Python easily with MongoDBand which makes this pair even more attractive to application developers.

9.0.1.5 Flask-PyMongo

> "Flask is a micro-web framework written in Python" [35].

It was developed after Django, and it is very pythonic in nature which implies that it is explicitly the targeting the Python user community. It is lightweight as it does not require additional tools or libraries and hence is classified as a Micro-Web framework. It is often used with MongoDB using PyMongo connector, and it treats data within MongoDB as searchable Python dictionaries. The applications such as Pinterest, LinkedIn, and the community web page for Flask are using the Flask framework. Moreover, it supports various features such as the RESTful request dispatching, secure cookies, Google app engine compatibility, and integrated support for unit testing, etc [35]. When it comes to connecting to a database, the connection details for MongoDB can be passed as a variable or configured in PyMongo constructor with additional arguments such as username and password, if required. It is important that versions of both Flask and MongoDB are compatible with each other to avoid functionality breaks [36].

9.0.1.5.1 Installation Flask-PyMongo can be installed with an easy command such as this:

```
$ pip install Flask-PyMongo
```

PyMongo can be added in the following manner:

```
from flask import Flask
from flask_pymongo import PyMongo
app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb://localhost:27017/
   ↪ cloudmesh_community"
mongo = PyMongo(app)
```

9.0.1.5.2 Configuration There are two ways to configure Flask-PyMongo. The first way would be to pass a MongoDB URI to the PyMongo constructor, while the second way would be to

> "assign it to the MONGO_URI Flask confiuration variable" [36].

9.0.1.5.3 Connection to multiple databases/servers Multiple PyMongo instances can be used to connect to multiple databases or database servers. To achieve this, once would use a command similar to the following:

```
app = Flask(__name__)
mongo1 = PyMongo(app, uri="mongodb://localhost:27017/
   ↪ cloudmesh_community_one")
mongo2 = PyMongo(app, uri="mongodb://localhost:27017/
   ↪ cloudmesh_community_two")
mongo3 = PyMongo(app, uri=
      "mongodb://another.host:27017/cloudmesh_community_Three")
```

9.0.1.5.4 Flask-PyMongo Methods Flask-PyMongo provides helpers for some common tasks. One of them is the *Collection.find_one_or_404* method shown in the following example:

```
@app.route("/user/<username>")
def user_profile(username):
    user = mongo.db.cloudmesh_community.find_one_or_404({"_id":
       ↪ username})
    return render_template("user.html", user=user)
```

This method is very similar to the MongoDB's *find_one()* method, however, instead of returning *None* it causes a *404 Not Found HTTP* status [36].

Similarly, the *PyMongo.send_file* and *PyMongo.save_file* methods work on the file-like objects and save them to GridFS using the given file name [36].

9.0.1.5.5 Additional Libraries Flask-MongoAlchemy and Flask-MongoEngine are the additional libraries that can be used to connect to a MongoDB database while using enhanced features with the Flask app. The Flask-MongoAlchemy is used as a proxy between Python and MongoDB to connect. It provides an option such as server or database based authentication to connect to MongoDB. While the default is set server based, to use a database-based authentication, the config value *MONGOALCHEMY_SERVER_AUTH* parameter must be set to *False* [37].

Flask-MongoEngine is the Flask extension that provides integration with the MongoEngine. It handles connection management for the apps. It can be installed through *pip* and set up very easily as well. The default configuration is set to the local host and port 27017. For the custom port and in cases where MongoDB is running on another server, the host and port must be explicitly specified in connect strings within the *MONGODB_SETTINGS* dictionary with *app.config*, along with the database username and password, in cases where a database authentication is enabled. The URI style connections are also supported and supply the URI as the host in the *MONGODB_SETTINGS* dictionary with *app.config*.

There are various custom query sets that are available within Flask-Mongoengine that are attached to Mongoengine's default queryset [38].

9.0.1.5.6 Classes and Wrappers Attributes such as *cx* and *db* in the PyMongo objects are the ones that help provide access to the MongoDB server [36]. To achieve this, one must pass the Flask app to the constructor or call *init_app()* [36].

> "Flask-PyMongo wraps PyMongo's MongoClient, Database, and Collection classes, and overrides their attribute and item accessors" [36].

This type of wrapping allows Flask-PyMongo to add methods to *Collection* while at the same time allowing a MongoDB-style dotted expressions in the code [36].

```
type(mongo.cx)
type(mongo.db)
type(mongo.db.cloudmesh_community)
```

Flask-PyMongo creates connectivity between Python and Flask using a MongoDB database and supports

> "extensions that can add application features as if they were implemented in Flask itself" [39],

hence, it can be used as an additional Flask functionality in Python code. The extensions are there for the purpose of supporting form validations, authentication technologies, object-relational mappers and framework related tools which ultimately adds a lot of strength to this micro-web framework [39]. One of the main reasons and benefits why it is frequently used with MongoDB is its capability of adding more control over databases and history [39].

9.0.2 Mongoengine

9.0.2.1 Introduction MongoEngine is a document mapper for working with mongoldb with python. To be able to use mongo engine MongodD should be already installed and running.

9.0.2.2 Install and connect Mongoengine can be installed by running:

```
    $ pip install mongo engine
```

This will install six, pymongo and mongoengine.

To connect to mongoldb use connect () function by specifying mongoldb instance name. You don't need to go to mongo shell but this can be done from unix shell or cmd line. In this case we are connecting to a database named student_db.

```
from mongo engine import * connect ('student_db')
```

If mongodb is running on a port different from default port , port number and host need to be specified. If mongoldb needs authentication username and password need to be specified.

9.0.2.3 Basics Mongodb does not enforce schemas. Comparing to RDBMS, Row in mongoldb is called a "document" and table can be compared to *Collection*. Defining a schema is helpful as it minimizes coding error's. To define a schema we create a class that inherits from document.

```
from mongoengine import *

class Student(Document):
    first_name = StringField(max_length=50)
    last_name = StringField(max_length=50)
```

Fields are not mandatory but if needed, set the required keyword argument to True. There are multiple values available for field types. Each field can be customized by by keyword argument. If each student is sending text messages to Universities central database , these can be stored using Mongodb. Each text can have different data types, some might have images or some might have url's. So we can create a class text and link it to student by using Reference field (similar to foreign key in RDBMS).

```
class Text(Document):
    title = StringField(max_length=120, required=True)
    author = ReferenceField(Student)
    meta = {'allow_inheritance': True}

class OnlyText(Text):
    content = StringField()

class ImagePost(Text):
    image_path = StringField()

class LinkPost(Text):
    link_url = StringField()
```

MongoDb supports adding tags to individual texts rather then storing them separately and then having them referenced.Similarly Comments can also be stored directly in a Text.

```
class Text(Document):
    title = StringField(max_length=120, required=True)
    author = ReferenceField(User)
    tags = ListField(StringField(max_length=30))
    comments = ListField(EmbeddedDocumentField(Comment))
```

For accessing data: if we need to get titles.

```
for text in OnlyText.objects:
    print(text.title)
```

Searching texts with tags.

```
for text in Text.objects(tags='mongodb'):
    print(text.title)
```

10 OTHER

10.0.1 Word Count with Parallel Python

We will demonstrate Python's `multiprocessing` API for parallel computation by writing a program that counts how many times each word in a collection of documents appear.

10.0.1.1 Generating a Document Collection Before we begin, let us write a script that will generate document collections by specifying the number of documents and the number of words per document. This will make benchmarking straightforward.

To keep it simple, the vocabulary of the document collection will consist of random numbers rather than the words of an actual language:

```
'''Usage: generate_nums.py [-h] NUM_LISTS INTS_PER_LIST MIN_INT
   ↪ MAX_INT DEST_DIR

Generate random lists of integers and save them
as 1.txt, 2.txt, etc.
```

```
Arguments:
   NUM_LISTS      The number of lists to create.
   INTS_PER_LIST  The number of integers in each list.
   MIN_NUM        Each generated integer will be >= MIN_NUM.
   MAX_NUM        Each generated integer will be <= MAX_NUM.
   DEST_DIR       A directory where the generated numbers will be
      ↪ stored.

Options:
  -h --help
'''

import os, random, logging
from docopt import docopt


def generate_random_lists(num_lists,
                          ints_per_list, min_int, max_int):
    return [[random.randint(min_int, max_int) \
        for i in range(ints_per_list)] for i in range(num_lists)]


if __name__ == '__main__':
   args = docopt(__doc__)
   num_lists, ints_per_list, min_int, max_int, dest_dir = [
      int(args['NUM_LISTS']),
      int(args['INTS_PER_LIST']),
      int(args['MIN_INT']),
      int(args['MAX_INT']),
      args['DEST_DIR']
   ]

   if not os.path.exists(dest_dir):
      os.makedirs(dest_dir)

   lists = generate_random_lists(num_lists,
                                 ints_per_list,
```

```
                                  min_int,
                                  max_int)
   curr_list = 1
   for lst in lists:
      with open(os.path.join(dest_dir, '%d.txt' % curr_list), 'w') as
         ↪  f:
     f.write(os.linesep.join(map(str, lst)))
  curr_list += 1
   logging.debug('Numbers written.')
```

Notice that we are using the docopt module that you should be familiar with from the Section [Python DocOpts](#s-python-docopts} to make the script easy to run from the command line.

You can generate a document collection with this script as follows:

```
python generate_nums.py 1000 10000 0 100 docs-1000-10000
```

10.0.1.2 Serial Implementation

A first serial implementation of wordcount is straightforward:

```
'''Usage: wordcount.py [-h] DATA_DIR

Read a collection of .txt documents and count how many times each
   ↪ word
appears in the collection.

Arguments:
  DATA_DIR  A directory with documents (.txt files).

Options:
  -h --help
'''

import os, glob, logging
from docopt import docopt

logging.basicConfig(level=logging.DEBUG)
```

```
def wordcount(files):
   counts = {}
   for filepath in files:
      with open(filepath, 'r') as f:
     words = [word.strip() for word in f.read().split()]
     for word in words:
        if word not in counts:
           counts[word] = 0
        counts[word] += 1
   return counts


if __name__ == '__main__':
   args = docopt(__doc__)
   if not os.path.exists(args['DATA_DIR']):
      raise ValueError('Invalid data directory: %s' % args['DATA_DIR'
         ↪ ])

   counts = wordcount(glob.glob(os.path.join(args['DATA_DIR'], '*.txt
      ↪ ')))
   logging.debug(counts)
```

10.0.1.3 Serial Implementation Using map and reduce

We can improve the serial implementation in anticipation of parallelizing the program by making use of Python's `map` and `reduce` functions.

In short, you can use `map` to apply the same function to the members of a collection. For example, to convert a list of numbers to strings, you could do:

```
import random
nums = [random.randint(1, 2) for _ in range(10)]
print(nums)
[2, 1, 1, 1, 2, 2, 2, 2, 2, 2]
print(map(str, nums))
['2', '1', '1', '1', '2', '2', '2', '2', '2', '2']
```

We can use reduce to apply the same function cumulatively to the items of a sequence. For example, to find the total of the numbers in our list, we could use `reduce` as follows:

```
def add(x, y):
    return x + y

print(reduce(add, nums))
17
```

We can simplify this even more by using a lambda function:

```
print(reduce(lambda x, y: x + y, nums))
17
```

You can read more about Python's lambda function in the docs.

With this in mind, we can reimplement the wordcount example as follows:

```
'''Usage: wordcount_mapreduce.py [-h] DATA_DIR

Read a collection of .txt documents and count how
many times each word
appears in the collection.

Arguments:
   DATA_DIR  A directory with documents (.txt files).

Options:
   -h --help
'''

import os, glob, logging
from docopt import docopt

logging.basicConfig(level=logging.DEBUG)

def count_words(filepath):
   counts = {}
   with open(filepath, 'r') as f:
      words = [word.strip() for word in f.read().split()]

  for word in words:
```

```
      if word not in counts:
         counts[word] = 0
      counts[word] += 1
   return counts


def merge_counts(counts1, counts2):
   for word, count in counts2.items():
      if word not in counts1:
      counts1[word] = 0
   counts1[word] += counts2[word]
   return counts1


if __name__ == '__main__':
   args = docopt(__doc__)
   if not os.path.exists(args['DATA_DIR']):
      raise ValueError('Invalid data directory: %s' % args['DATA_DIR'
         ↪ ])

      per_doc_counts = map(count_words,
                           glob.glob(os.path.join(args['DATA_DIR'],
                           '*.txt')))
   counts = reduce(merge_counts, [{}] + per_doc_counts)
   logging.debug(counts)
```

10.0.1.4 Parallel Implementation Drawing on the previous implementation using `map` and `reduce`, we can parallelize the implementation using Python's `multiprocessing` API:

```
'''Usage: wordcount_mapreduce_parallel.py [-h] DATA_DIR NUM_PROCESSES

Read a collection of .txt documents and count, in parallel, how many
times each word appears in the collection.

Arguments:
   DATA_DIR       A directory with documents (.txt files).
   NUM_PROCESSES  The number of parallel processes to use.
```

```
Options:
   -h --help
'''

import os, glob, logging
from docopt import docopt
from wordcount_mapreduce import count_words, merge_counts
from multiprocessing import Pool

logging.basicConfig(level=logging.DEBUG)

if __name__ == '__main__':
   args = docopt(__doc__)
   if not os.path.exists(args['DATA_DIR']):
      raise ValueError('Invalid data directory: %s' % args['DATA_DIR'
         ↪ ])
   num_processes = int(args['NUM_PROCESSES'])

   pool = Pool(processes=num_processes)

   per_doc_counts = pool.map(count_words,
                             glob.glob(os.path.join(args['DATA_DIR'],
                             '*.txt')))
   counts = reduce(merge_counts, [{}] + per_doc_counts)
   logging.debug(counts)
```

10.0.1.5 Benchmarking To time each of the examples, enter it into its own Python file and use Linux's `time` command:

```
$ time python wordcount.py docs-1000-10000
```

The output contains the real run time and the user run time. real is wall clock time - time from start to finish of the call. user is the amount of CPU time spent in user-mode code (outside the kernel) within the process, that is, only actual CPU time used in executing the process.

10.0.1.6 Excersises E.python.wordcount.1:

> Run the three different programs (serial, serial w/ map and reduce, parallel) and answer the following questions:
>
> 1. Is there any performance difference between the different versions of the program?
> 2. Does user time significantly differ from real time for any of the versions of the program?
> 3. Experiment with different numbers of processes for the parallel example, starting with 1. What is the performance gain when you goal from 1 to 2 processes? From 2 to 3? When do you stop seeing improvement? (this will depend on your machine architecture)

10.0.1.7 References

- Map, Filter and Reduce
- multiprocessing API

10.0.2 NumPy

NumPy is a popular library that is used by many other Python packages such as Pandas, SciPy, and scikit-learn. It provides a fast, simple-to-use way of interacting with numerical data organized in vectors and matrices. In this section, we will provide a short introduction to NumPy.

10.0.2.1 Installing NumPy The most common way of installing NumPy, if it wasn't included with your Python installation, is to install it via pip:

```
$ pip install numpy
```

If NumPy has already been installed, you can update to the most recent version using:

```
$ pip install -U numpy
```

You can verify that NumPy is installed by trying to use it in a Python program:

```
import numpy as np
```

Note that, by convention, we import NumPy using the alias 'np' - whenever you see 'np' sprinkled in example Python code, it's a good bet that it is using NumPy.

10.0.2.2 NumPy Basics At its core, NumPy is a container for n-dimensional data. Typically, 1-dimensional data is called an array and 2-dimensional data is called a matrix. Beyond 2-dimensions would be considered a multidimensional array. Examples, where you'll encounter these dimensions, may include:

- 1 Dimensional: time-series data such as audio, stock prices, or a single observation in a dataset.
- 2 Dimensional: connectivity data between network nodes, user-product recommendations, and database tables.
- 3+ Dimensional: network latency between nodes over time, video (RGB+time), and version-controlled datasets.

All of these data can be placed into NumPy's array object, just with varying dimensions.

10.0.2.3 Data Types: The Basic Building Blocks Before we delve into arrays and matrices, we will start with the most basic element of those: a single value. NumPy can represent data utilizing many different standard datatypes such as uint8 (an 8-bit **u**signed **int**eger), float64 (a 64-bit float), or str (a string). An exhaustive listing can be found at:

- https://docs.scipy.org/doc/numpy-1.15.0/user/basics.types.html

Before moving on, it is important to know about the tradeoff made when using different datatypes. For example, a uint8 can only contain values between 0 and 255. This, however, contrasts with float64 which can express any value from +/- 1.80e+308. So why wouldn't we just always use float64s? Though they allow us to be more expressive in terms of numbers, they also consume more memory. If we were working with a 12-megapixel image, for example, storing that image using uint8 values would require 3000 * 4000 * 8 = 96 million bits, or 91.55 MB of memory. If we were to store the same image utilizing float64, our image would consume 8 times as much memory: 768 million bits or 732.42 MB. It is important to use the right data type for the job to avoid consuming unnecessary resources or slowing down processing.

Finally, while NumPy will conveniently convert between datatypes, one must be aware of overflows when using smaller data types. For example:

```
a = np.array([6], dtype=np.uint8)
print(a)
>>>[6]
a = a + np.array([7], dtype=np.uint8)
print(a)
>>>[13]
a = a + np.array([245], dtype=np.uint8)
```

```
print(a)
>>>[2]
```

In this example, it makes sense that 6+7=13. But how does 13+245=2? Put simply, the object type (uint8) simply ran out of space to store the value and wrapped back around to the beginning. An 8-bit number is only capable of storing 2^8, or 256, unique values. An operation that results in a value above that range will 'overflow' and cause the value to wrap back around to zero. Likewise, anything below that range will 'underflow' and wrap back around to the end. In our example, 13+245 became 258, which was too large to store in 8 bits and wrapped back around to 0 and ended up at 2.

NumPy will, generally, try to avoid this situation by dynamically retyping to whatever datatype will support the result:

```
a = a + 260
print(test)
>>>[262]
```

Here, our addition caused our array, 'a', to be upscaled to use uint16 instead of uint8. Finally, NumPy offers convenience functions akin to Python's range() function to create arrays of sequential numbers:

```
X = np.arange(0.2,1,.1)
print(X)
>>>array([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], dtype=float32)
```

We can use this function to also generate parameters spaces that can be iterated on:

```
P = 10.0 ** np.arange(-7,1,1)
print(P)

for x,p in zip(X,P):
    print('%f, %f' % (x, p))
```

10.0.2.4 Arrays: Stringing Things Together With our knowledge of datatypes in hand, we can begin to explore arrays. Simply put, arrays can be thought of as a sequence of values (not neccesarily numbers). Arrays are 1 dimensional and can be created and accessed simply:

```
a = np.array([1, 2, 3])
print(type(a))
>>><class 'numpy.ndarray'>
```

```
print(a)
>>>[1 2 3]
print(a.shape)
>>>(3,)
a[0]
>>>1
```

Arrays (and, later, matrices) are zero-indexed. This makes it convenient when, for example, using Python's range() function to iterate through an array:

```
for i in range(3):
    print(a[i])
>>>1
>>>2
>>>3
```

Arrays are, also, mutable and can be changed easily:

```
a[0] = 42
print(a)
>>>array([42, 2, 3])
```

NumPy also includes incredibly powerful broadcasting features. This makes it very simple to perform mathematical operations on arrays that also makes intuitive sense:

```
a * 3
>>>array([3, 6, 9])
a**2
>>>array([1, 4, 9], dtype=int32)
```

Arrays can also interact with other arrays:

```
b = np.array([2, 3, 4])
print(a * b)
>>>array([ 2,  6, 12])
```

In this example, the result of multiplying together two arrays is to take the element-wise product while multiplying by a constant will multiply each element in the array by that constant. NumPy supports all of the basic mathematical operations: addition, subtraction, multiplication, division, and powers. It also includes an extensive suite of mathematical functions, such as log() and max(), which are covered later.

10.0.2.5 Matrices: An Array of Arrays Matrices can be thought of as an extension of arrays - rather than having one dimension, matrices have 2 (or more). Much like arrays, matrices can be created easily within NumPy:

```
m = np.array([[1, 2], [3, 4]])
print(m)
>>>[[1 2]
>>> [3 4]]
```

Accessing individual elements is similar to how we did it for arrays. We simply need to pass in a number of arguments equal to the number of dimensions:

```
m[1][0]
>>>3
```

In this example, our first index selected the row and the second selected the column - giving us our result of 3. Matrices can be extending out to any number of dimensions by simply using more indices to access specific elements (though use-cases beyond 4 may be somewhat rare).

Matrices support all of the normal mathematial functions such as +, -, *, and /. A special note: the * operator will result in an element-wise multiplication. Using @ or np.matmul() for matrix multiplication:

```
print(m-m)
print(m*m)
print(m/m)
```

More complex mathematical functions can typically be found within the NumPy library itself:

```
print(np.sin(x))
print(np.sum(x))
```

A full listing can be found at: https://docs.scipy.org/doc/numpy/reference/routines.math.html

10.0.2.6 Slicing Arrays and Matrices As one can imagine, accessing elements one-at-a-time is both slow and can potentially require many lines of code to iterate over every dimension in the matrix. Thankfully, NumPy incorporate a very powerful slicing engine that allows us to access ranges of elements easily:

```
m[1, :]
>>>array([3, 4])
```

The ':' value tells NumPy to select all elements in the given dimension. Here, we've requested all elements in the first row. We can also use indexing to request elements within a given range:

```
a = np.arange(0, 10, 1)
print(a)
>>>[0 1 2 3 4 5 6 7 8 9]
a[4:8]
>>>array([4, 5, 6, 7])
```

Here, we asked NumPy to give us elements 4 through 7 (ranges in Python are inclusive at the start and non-inclusive at the end). We can even go backwards:

```
a[-5:]
>>>array([5, 6, 7, 8, 9])
```

In the previous example, the negative value is asking NumPy to return the last 5 elements of the array. Had the argument been ':-5', NumPy would've returned everything BUT the last five elements:

```
a[:-5]
>>>array([0, 1, 2, 3, 4])
```

Becoming more familiar with NumPy's accessor conventions will allow you write more efficient, clearer code as it is easier to read a simple one-line accessor than it is a multi-line, nested loop when extracting values from an array or matrix.

10.0.2.7 Useful Functions The NumPy library provides several convenient mathematical functions that users can use. These functions provide several advantages to code written by users:

- They are open source typically have multiple contributors checking for errors.
- Many of them utilize a C interface and will run much faster than native Python code.
- They're written to very flexible.

NumPy arrays and matrices contain many useful aggregating functions such as max(), min(), mean(), etc These functions are usually able to run an order of magnitude faster than looping through the object, so it's important to understand what functions are available to avoid 'reinventing the wheel.' In addition, many of the functions are able to sum or average across axes, which make them extremely useful if your data has inherent grouping. To return to a previous example:

```
m = np.array([[1, 2], [3, 4]])
print(m)
```

```
>>>[[1 2]
>>> [3 4]]
m.sum()
>>>10
m.sum(axis=1)
>>>[3, 7]
m.sum(axis=0)
>>>[4, 6]
```

In this example, we created a 2x2 matrix containing the numbers 1 through 4. The sum of the matrix returned the element-wise addition of the entire matrix. Summing across axis 0 (rows) returned a new array with the element-wise addition across each row. Likewise, summing across axis 1 (columns) returned the columnar summation.

10.0.2.8 Linear Algebra Perhaps one of the most important uses for NumPy is its robust support for Linear Algebra functions. Like the aggregation functions described in the previous section, these functions are optimized to be much faster than user implementations and can utilize processesor level features to provide very quick computations. These functions can be accessed very easily from the NumPy package:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.matmul(a, b))
>>>[[19 22]
    [43 50]]
```

Included in within np.linalg are functions for calculating the Eigendecomposition of square matrices and symmetric matrices. Finally, to give a quick example of how easy it is to implement algorithms in NumPy, we can easily use it to calculate the cost and gradient when using simple Mean-Squared-Error (MSE):

```
cost = np.power(Y - np.matmul(X, weights)), 2).mean(axis=1)
gradient = np.matmul(X.T, np.matmul(X, weights) - y)
```

Finally, more advanced functions are easily available to users via the linalg library of NumPy as:

```
from numpy import linalg

A = np.diag((1,2,3))
```

```
w,v = linalg.eig(A)

print ('w =', w)
print ('v =', v)
```

10.0.2.9 NumPy Resources

- https://docs.scipy.org/doc/numpy
- http://cs231n.github.io/python-numpy-tutorial/#numpy
- https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.linalg.html
- https://en.wikipedia.org/wiki/Mean_squared_error

10.0.3 Scipy

SciPy is a library built around NumPy and has a number of off-the-shelf algorithms and operations implemented. These include algorithms from calculus (such as integration), statistics, linear algebra, image-processing, signal processing, machine learning.

To achieve this, SciPy bundles a number of useful open-source software for mathematics, science, and engineering. It includes the following packages:

NumPy, for managing N-dimensional arrays

SciPy library, to access fundamental scientific computing capabilities

Matplotlib, to conduct 2D plotting

IPython, for an Interactive console (see jupyter)

Sympy, for symbolic mathematics

pandas, for providing data structures and analysis

10.0.3.1 Introduction First, we add the usual scientific computing modules with the typical abbreviations, including sp for scipy. We could invoke scipy's statistical package as sp.stats, but for the sake of laziness, we abbreviate that too.

```
import numpy as np # import numpy
import scipy as sp # import scipy
from scipy import stats # refer directly to stats rather than sp.
   ↪ stats
```

```
import matplotlib as mpl # for visualization
from matplotlib import pyplot as plt # refer directly to pyplot
                                     # rather than mpl.pyplot
```

Now we create some random data to play with. We generate 100 samples from a Gaussian distribution centered at zero.

```
s = sp.randn(100)
```

How many elements are in the set?

```
print ('There are',len(s),'elements in the set')
```

What is the mean (average) of the set?

```
print ('The mean of the set is',s.mean())
```

What is the minimum of the set?

```
print ('The minimum of the set is',s.min())
```

What is the maximum of the set?

```
print ('The maximum of the set is',s.max())
```

We can use the scipy functions too. What's the median?

```
print ('The median of the set is',sp.median(s))
```

What about the standard deviation and variance?

```
print ('The standard deviation is',sp.std(s),
       'and the variance is',sp.var(s))
```

Isn't the variance the square of the standard deviation?

```
    print ('The square of the standard deviation is',sp.std(s)**2)
```

How close are the measures? The differences are close as the following calculation shows

```
    print ('The difference is',abs(sp.std(s)**2 - sp.var(s)))

    print ('And in decimal form, the difference is %0.16f' %
           (abs(sp.std(s)**2 - sp.var(s))))
```

How does this look as a histogram? See Figure 18, Figure 19, Figure 20

```
plt.hist(s) # yes, one line of code for a histogram
plt.show()
```

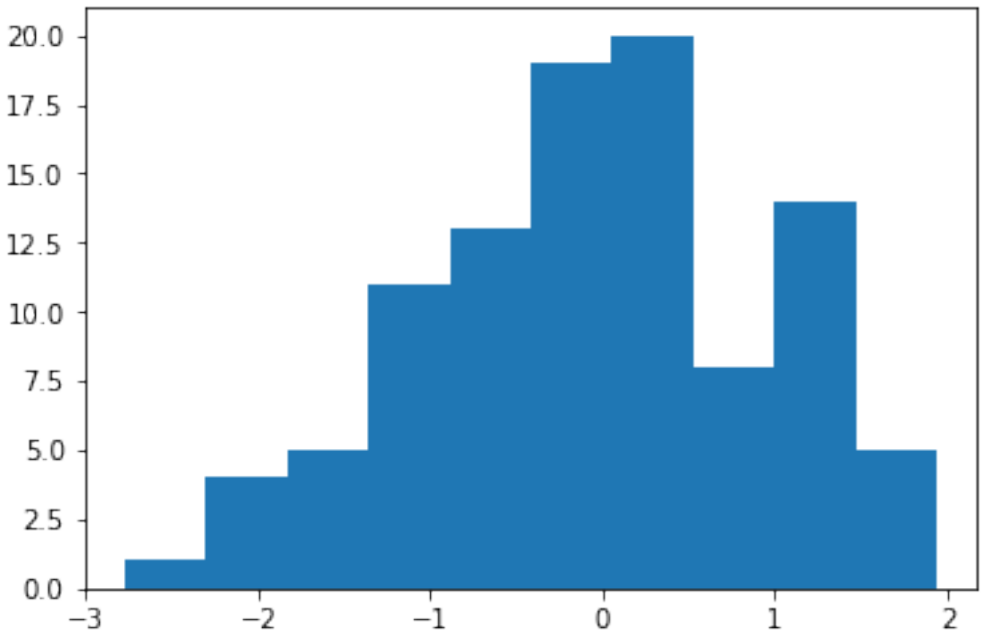


Figure 18: Histogram 1

Let us add some titles.

```
plt.clf() # clear out the previous plot

plt.hist(s)
plt.title("Histogram Example")
plt.xlabel("Value")
plt.ylabel("Frequency")
```

```
plt.show()
```

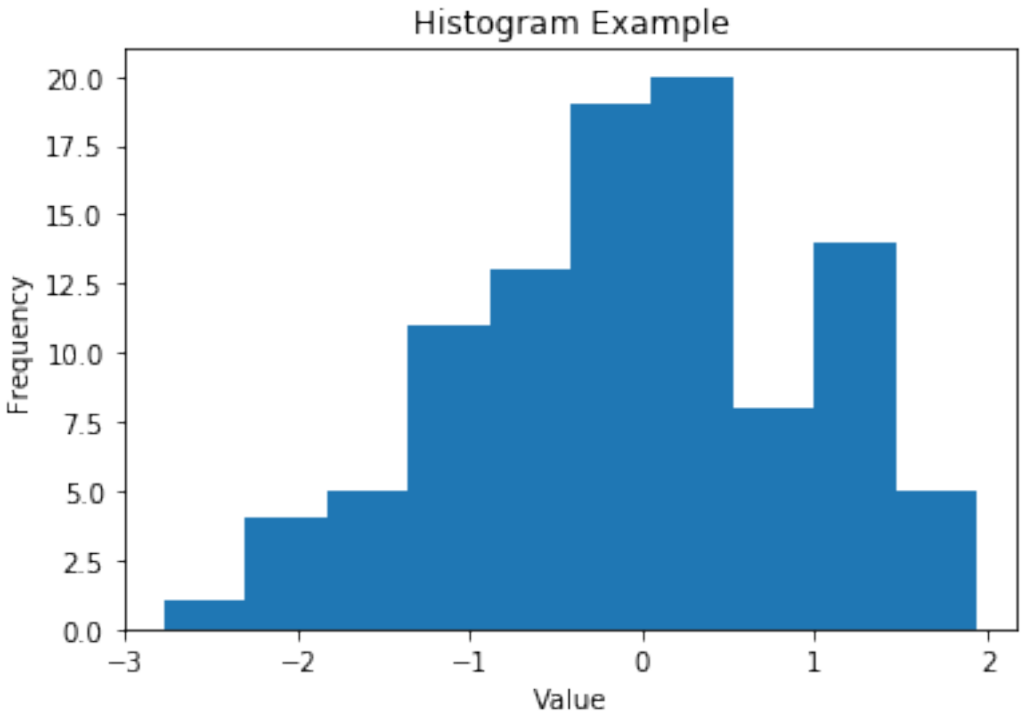


Figure 19: Histogram 2

Typically we do not include titles when we prepare images for inclusion in LaTeX. There we use the caption to describe what the figure is about.

```
plt.clf() # clear out the previous plot

plt.hist(s)
plt.xlabel("Value")
plt.ylabel("Frequency")

plt.show()
```

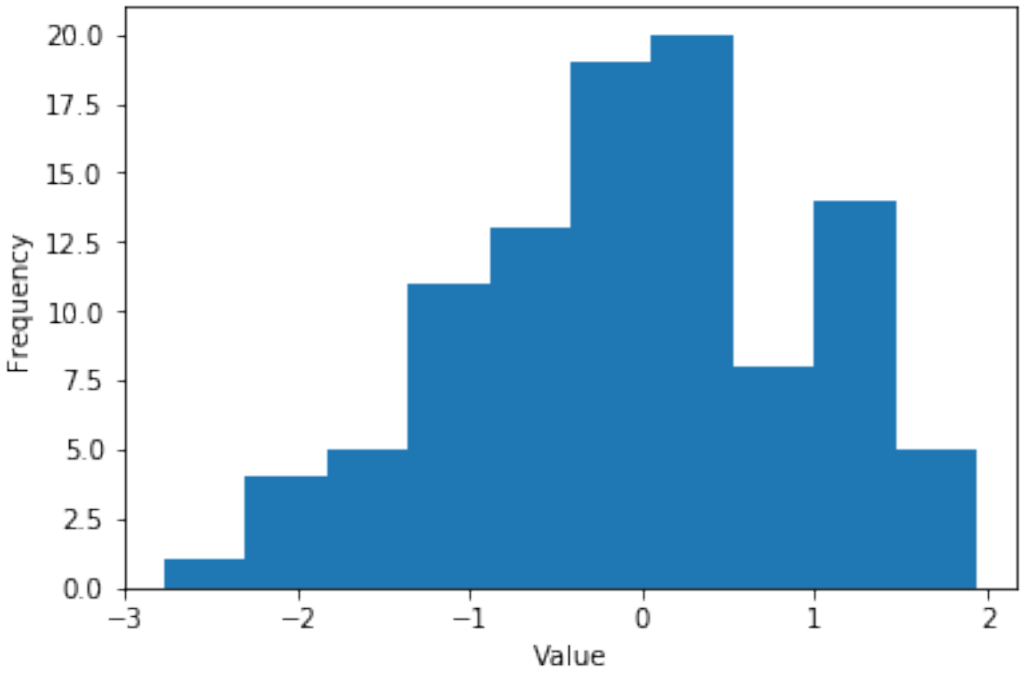


Figure 20: Histogram 3

Let us try out some linear regression or curve fitting. See @#fig:scipy-output_30_0

```
import random

def F(x):
    return 2*x - 2

def add_noise(x):
    return x + random.uniform(-1,1)

X = range(0,10,1)

Y = []
for i in range(len(X)):
    Y.append(add_noise(X[i]))

plt.clf() # clear out the old figure
plt.plot(X,Y,'.')
plt.show()
```

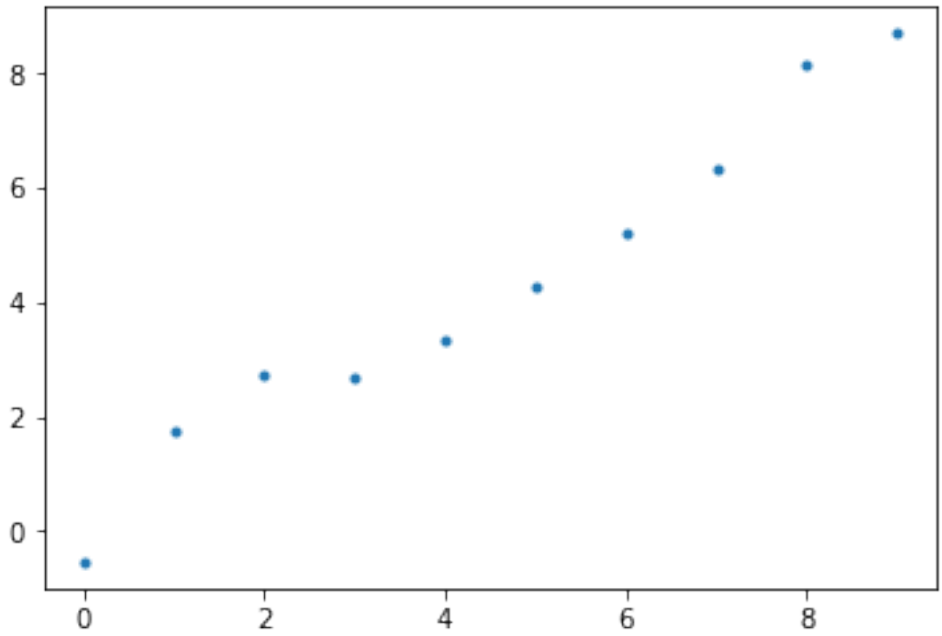


Figure 21: Result 1

Now let's try linear regression to fit the curve.

```
m, b, r, p, est_std_err = stats.linregress(X,Y)
```

What is the slope and y-intercept of the fitted curve?

```
print ('The slope is',m,'and the y-intercept is', b)

def Fprime(x): # the fitted curve
    return m*x + b
```

Now let's see how well the curve fits the data. We'll call the fitted curve F'.

```
X = range(0,10,1)

Yprime = []
for i in range(len(X)):
    Yprime.append(Fprime(X[i]))
```

```
plt.clf() # clear out the old figure

# the observed points, blue dots
plt.plot(X, Y, '.', label='observed points')

# the interpolated curve, connected red line
plt.plot(X, Yprime, 'r-', label='estimated points')

plt.title("Linear Regression Example") # title
plt.xlabel("x") # horizontal axis title
plt.ylabel("y") # vertical axis title
# legend labels to plot
plt.legend(['obsered points', 'estimated points'])

# comment out so that you can save the figure
#plt.show()
```

To save images into a PDF file for inclusion into LaTeX documents you can save the images as follows. Other formats such as png are also possible, but the quality is naturally not sufficient for inclusion in papers and documents. For that, you certainly want to use PDF. The save of the figure has to occur before you use the `show()` command. See Figure 22

```
plt.savefig("regression.pdf", bbox_inches='tight')

plt.savefig('regression.png')

plt.show()
```

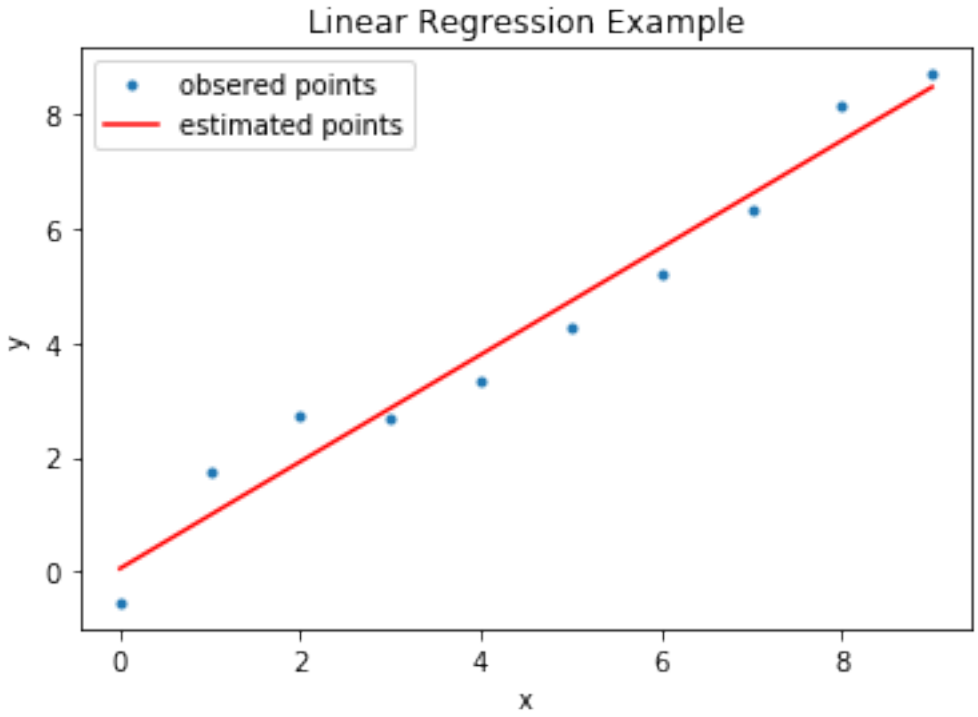


Figure 22: Result 2

10.0.3.2 References For more information about SciPy we recommend that you visit the following link

https://www.scipy.org/getting-started.html#learning-to-work-with-scipy

Additional material and inspiration for this section are from

- [ ] “Getting Started guide” https://www.scipy.org/getting-started.html
- [ ] Prasanth. “Simple statistics with SciPy.” Comfort at 1 AU. February 28, 2011. https://oneau.wordpress.com/2011/02/28/simple-statistics-with-scipy/.
- [ ] SciPy Cookbook. Lasted updated: 2015. http://scipy-cookbook.readthedocs.io/.

create bibtex entries

10.0.4 Scikit-learn

Learning Objectives

- Exploratory data analysis
- Pipeline to prepare data
- Full learning pipeline
- Fine tune the model
- Significance tests

10.0.4.1 Introduction to Scikit-learn Scikit learn is a Machine Learning specific library used in Python. Library can be used for data mining and analysis. It is built on top of NumPy, matplotlib and SciPy. Scikit Learn features Dimensionality reduction, clustering, regression and classification algorithms. It also features model selection using grid search, cross validation and metrics.

Scikit learn also enables users to preprocess the data which can then be used for machine learning using modules like preprocessing and feature extraction.

In this section we demonstrate how simple it is to use k-means in scikit learn.

10.0.4.2 Installation If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using pip

```
$ pip install numpy
$ pip install scipy -U
$ pip install -U scikit-learn
```

10.0.4.3 Supervised Learning Supervised Learning is used in machine learning when we already know a set of output predictions based on input characteristics and based on that we need to predict the target for a new input. Training data is used to train the model which then can be used to predict the output from a bounded set.

Problems can be of two types

1. Classification : Training data belongs to three or four classes/categories and based on the label we want to predict the class/category for the unlabeled data.
2. Regression : Training data consists of vectors without any corresponding target values. Clustering can be used for these type of datasets to determine discover groups of similar examples. Another way is density estimation which determine the distribution of data within the input space. Histogram is the most basic form.

10.0.4.4 Unsupervised Learning Unsupervised Learning is used in machine learning when we have the training set available but without any corresponding target. The outcome of the problem is to discover groups within the provided input. It can be done in many ways.

Few of them are listed here

1. Clustering : Discover groups of similar characteristics.
2. Density Estimation : Finding the distribution of data within the provided input or changing the data from a high dimensional space to two or three dimension.

10.0.4.5 Building a end to end pipeline for Supervised machine learning using Scikit-learn A data pipeline is a set of processing components that are sequenced to produce meaningful data. Pipelines are commonly used in Machine learning, since there is lot of data transformation and manipulation that needs to be applied to make data useful for machine learning. All components are sequenced in a way that the output of one component becomes input for the next and each of the component is self contained. Components interact with each other using data.

Even if a component breaks, the downstream component can run normally using the last output. Sklearn provide the ability to build pipelines that can be transformed and modeled for machine learning.

10.0.4.6 Steps for developing a machine learning model

1. Explore the domain space
2. Extract the problem definition
3. Get the data that can be used to make the system learn to solve the problem definition.
4. Discover and Visualize the data to gain insights
5. Feature engineering and prepare the data
6. Fine tune your model
7. Evaluate your solution using metrics
8. Once proven launch and maintain the model.

10.0.4.7 Exploratory Data Analysis

Example project = Fraud detection system

First step is to load the data into a dataframe in order for a proper analysis to be done on the attributes.

```
data = pd.read_csv('dataset/data_file.csv')
data.head()
```

Perform the basic analysis on the data shape and null value information.

```
print(data.shape)
print(data.info())
data.isnull().values.any()
```

Here is the example of few of the visual data analysis methods.

10.0.4.7.1 Bar plot

A bar chart or graph is a graph with rectangular bars or bins that are used to plot categorical values. Each bar in the graph represents a categorical variable and the height of the bar is proportional to the value represented by it.

Bar graphs are used:

To make comparisons between variables To visualize any trend in the data, i.e., they show the dependence of one variable on another Estimate values of a variable

```
plt.ylabel('Transactions')
plt.xlabel('Type')
data.type.value_counts().plot.bar()
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1a7891fb70>

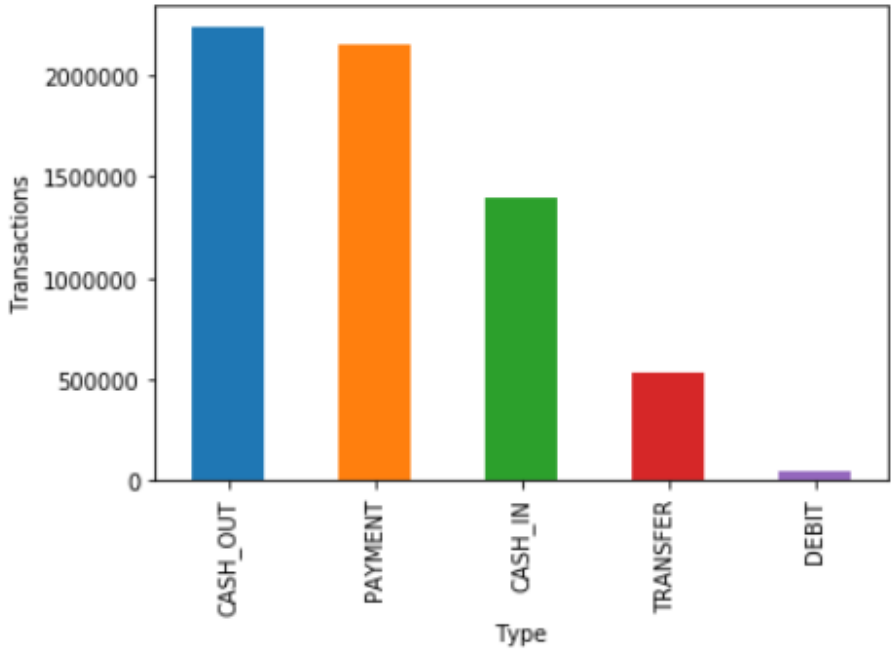


Figure 23: Example of scikit-learn barplots

10.0.4.7.2 Correlation between attributes Attributes in a dataset can be related based on differnt aspects.

Examples include attributes dependent on another or could be loosely or tightly coupled. Also example includes two variables can be associated with a third one.

In order to understand the relationship between attributes, correlation represents the best visual way to get an insight. Positive correlation meaning both attributes moving into the same direction. Negative correlation refers to opposte directions. One attributes values increase results in value decrease for other. Zero correlation is when the attributes are unrelated.

```
# compute the correlation matrix
corr = data.corr()

# generate a mask for the lower triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# set up the matplotlib figure
```

```
f, ax = plt.subplots(figsize=(18, 18))

# generate a custom diverging color map
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            square=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax);
```

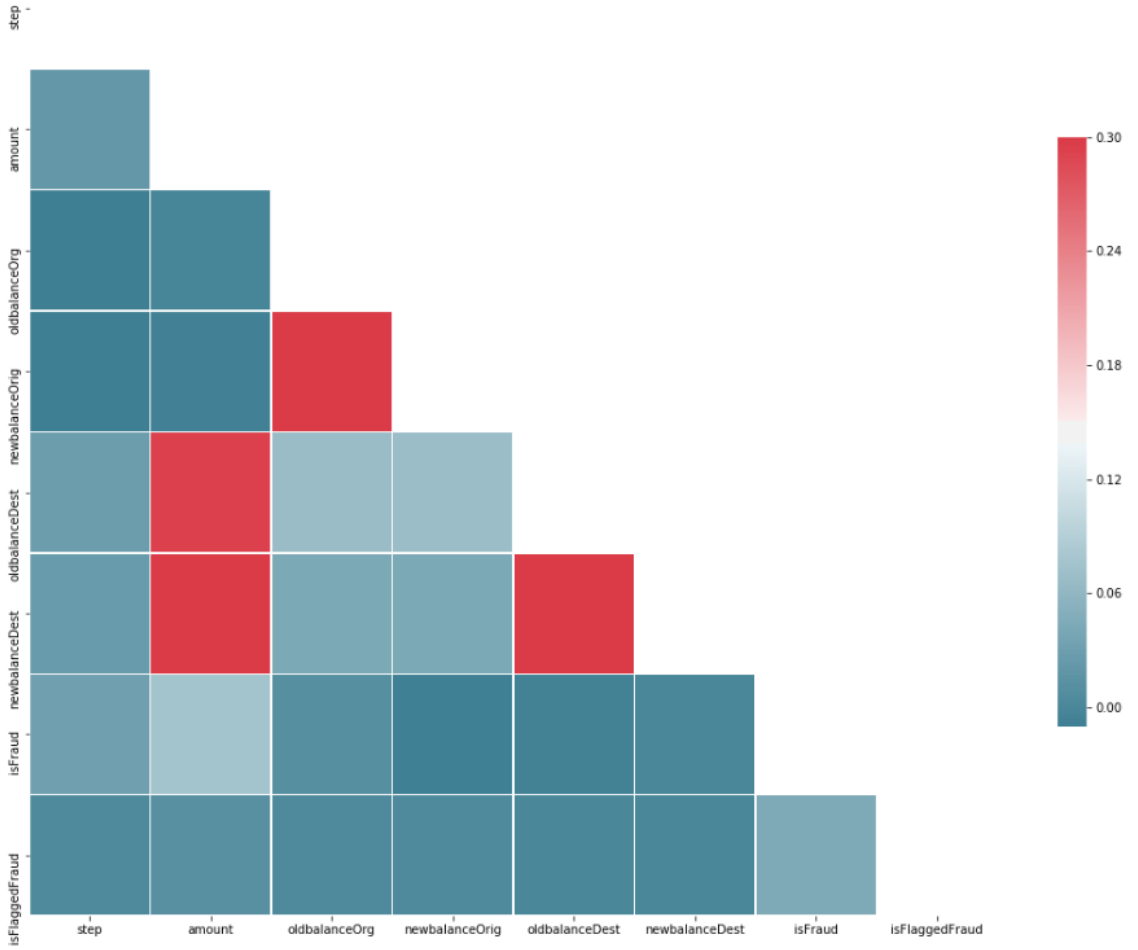


Figure 24: scikit-learn correlation array

10.0.4.7.3 Histogram Analysis of dataset attributes A histogram consists of a set of counts that represent the number of times some event occurred.

```
%matplotlib inline
data.hist(bins=30, figsize=(20,15))
plt.show()
```

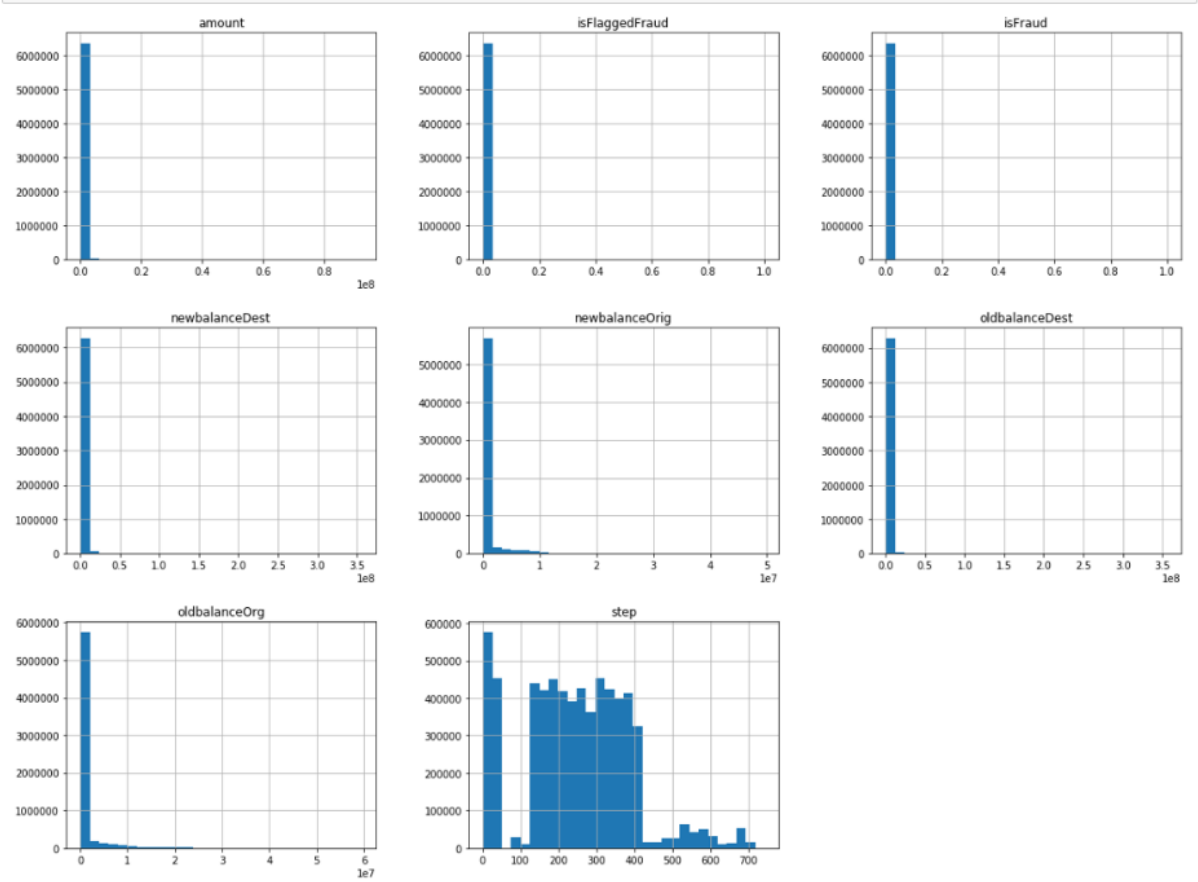


Figure 25: scikit-learn

10.0.4.7.4 Box plot Analysis Box plot analysis is useful in detecting whether a distribution is skewed and detect outliers in the data.

```
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
tmp = data.loc[(data.type == 'TRANSFER'), :]
```

```
a = sns.boxplot(x = 'isFlaggedFraud', y = 'amount', data = tmp, ax=
    ↪ axs[0][0])
axs[0][0].set_yscale('log')
b = sns.boxplot(x = 'isFlaggedFraud', y = 'oldbalanceDest', data =
    ↪ tmp, ax=axs[0][1])
axs[0][1].set(ylim=(0, 0.5e8))
c = sns.boxplot(x = 'isFlaggedFraud', y = 'oldbalanceOrg', data=tmp,
    ↪ ax=axs[1][0])
axs[1][0].set(ylim=(0, 3e7))
d = sns.regplot(x = 'oldbalanceOrg', y = 'amount', data=tmp.loc[(tmp.
    ↪ isFlaggedFraud ==1), :], ax=axs[1][1])
plt.show()
```

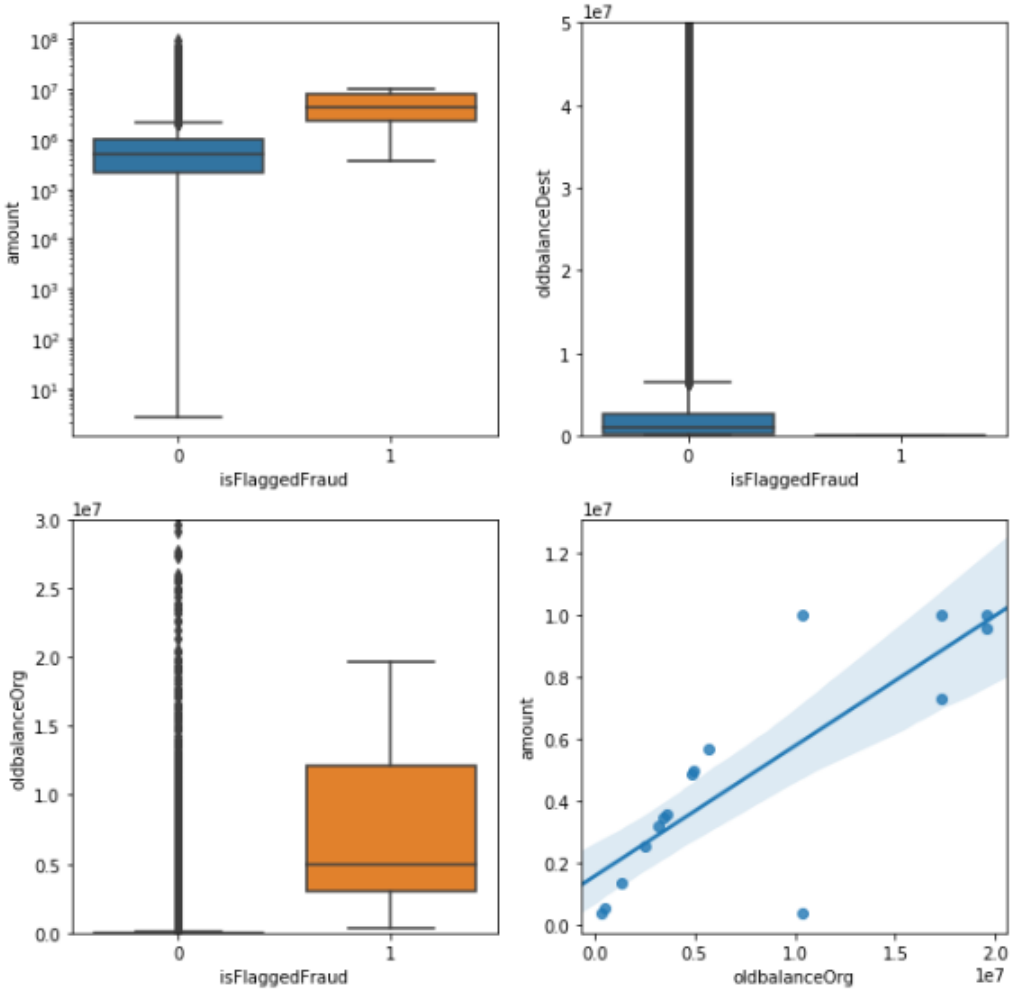


Figure 26: scikit-learn

10.0.4.7.5 Scatter plot Analysis The scatter plot displays values of two numerical variables as Cartesian coordinates.

```
plt.figure(figsize=(12,8))
sns.pairplot(data[['amount', 'oldbalanceOrg', 'oldbalanceDest', '
   ↪ isFraud']], hue='isFraud')
```

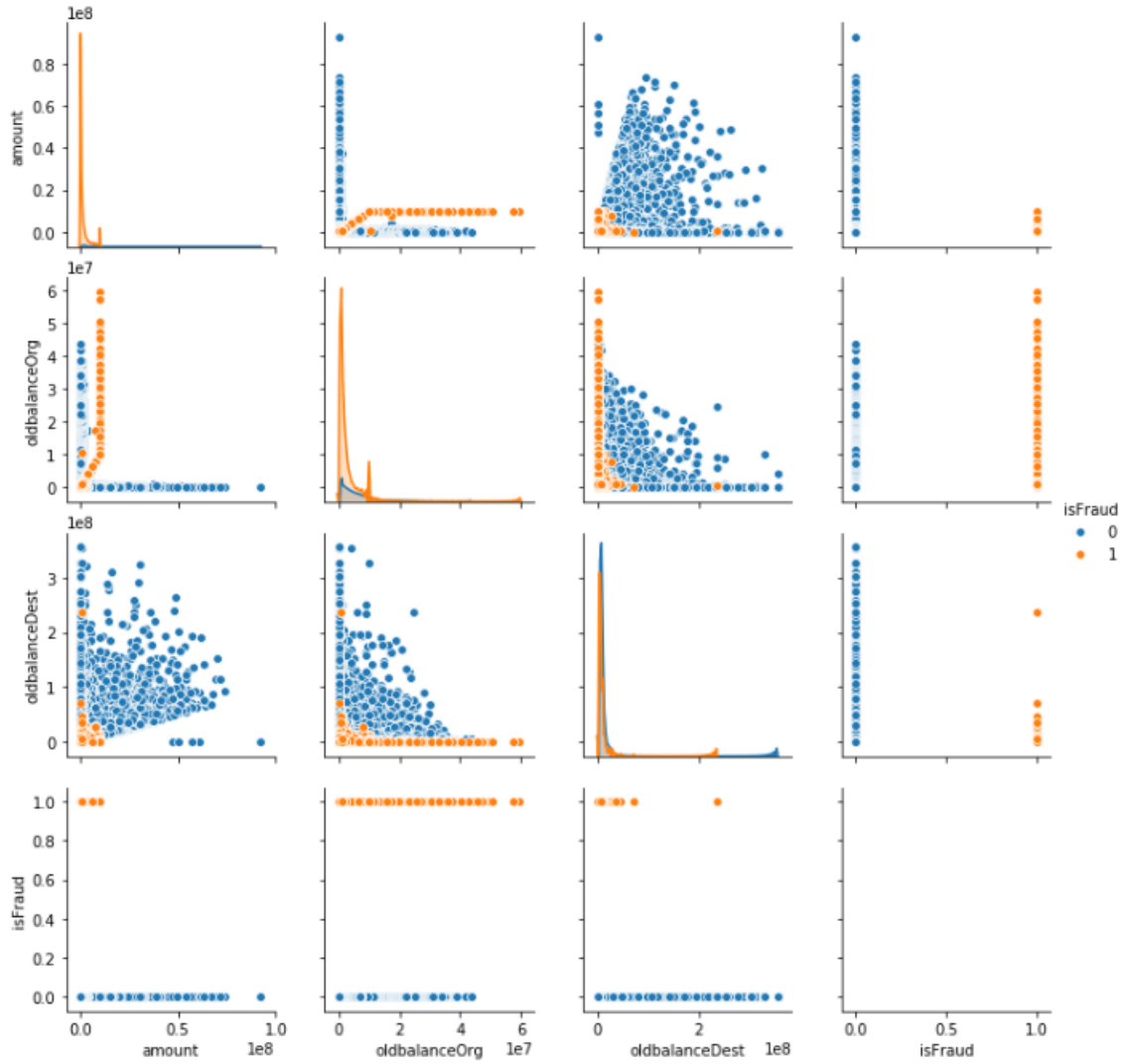


Figure 27: scikit-learn scatter plots

10.0.4.8 Data Cleansing - Removing Outliers If the transaction amount is lower than 5 percent of the all the transactions AND does not exceed USD 3000, we will exclude it from our analysis to reduce Type 1 costs If the transaction amount is higher than 95 percent of all the transactions AND exceeds USD 500000, we will exclude it from our analysis, and use a blanket review process for such transactions (similar to isFlaggedFraud column in original dataset) to reduce Type 2 costs

```
low_exclude = np.round(np.minimum(fin_samp_data.amount.quantile(0.05)
    ↪ , 3000), 2)
high_exclude = np.round(np.maximum(fin_samp_data.amount.quantile
```

```
    ↪ (0.95), 500000), 2)

###Updating Data to exclude records prone to Type 1 and Type 2 costs
low_data = fin_samp_data[fin_samp_data.amount > low_exclude]
data = low_data[low_data.amount < high_exclude]
```

10.0.4.9 Pipeline Creation Machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.

10.0.4.9.1 Defining DataFrameSelector to separate Numerical and Categorical attributes Sample function to seperate out Numerical and categorical attributes.

```
from sklearn.base import BaseEstimator, TransformerMixin

# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

10.0.4.9.2 Feature Creation / Additional Feature Engineering During EDA we identified that there are transactions where the balances do not tally after the transaction is completed.We believe this could potentially be cases where fraud is occurring. To account for this error in the transactions, we define two new features”errorBalanceOrig” and “errorBalanceDest”, calculated by adjusting the amount with the before and after balances for the Originator and Destination accounts.

Below, we create a function that allows us to create these features in a pipeline.

```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
```

```
amount_ix, oldbalanceOrg_ix, newbalanceOrig_ix, oldbalanceDest_ix,
   ↪ newbalanceDest_ix = 0, 1, 2, 3, 4

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self): # no *args or **kargs
        pass
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        errorBalanceOrig = X[:,newbalanceOrig_ix] +  X[:,amount_ix] -
           ↪   X[:,oldbalanceOrg_ix]
        errorBalanceDest = X[:,oldbalanceDest_ix] +  X[:,amount_ix]-
           ↪  X[:,newbalanceDest_ix]

        return np.c_[X, errorBalanceOrig, errorBalanceDest]
```

10.0.4.10 Creating Training and Testing datasets Training set includes the set of input examples that the model will be fit into or trained on by adjusting the parameters. Testing dataset is critical to test the generalizability of the model . By using this set, we can get the working accuracy of our model.

Testing set should not be exposed to model unless model training has not been completed. This way the results from testing will be more reliable.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size
   ↪ =0.30, random_state=42, stratify=y)
```

10.0.4.11 Creating pipeline for numerical and categorical attributes Identifying columns with Numerical and Categorical characteristics.

```
X_train_num = X_train[["amount","oldbalanceOrg", "newbalanceOrig", "
   ↪ oldbalanceDest", "newbalanceDest"]]
X_train_cat = X_train[["type"]]
X_model_col = ["amount","oldbalanceOrg", "newbalanceOrig", "
   ↪ oldbalanceDest", "newbalanceDest","type"]
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Imputer

num_attribs = list(X_train_num)
cat_attribs = list(X_train_cat)

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler())
    ])

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('cat_encoder', CategoricalEncoder(encoding="onehot-dense"))
    ])
```

10.0.4.12 Selecting the algorithm to be applied Algorithim selection primarily depends on the objective you are trying to solve and what kind of dataset is available. There are differnt type of algorithms which can be applied and we will look into few of them here.

10.0.4.12.1 Linear Regression This algorithm can be applied when you want to compute some continuous value. To predict some future value of a process which is currently running, you can go with regression algorithm.

Examples where linear regression can used are :

1. Predict the time taken to go from one place to another
2. Predict the sales for a future month
3. Predict sales data and improve yearly projections.

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
import time
scl= StandardScaler()
X_train_std = scl.fit_transform(X_train)
```

```
X_test_std = scl.transform(X_test)
start = time.time()
lin_reg = LinearRegression()
lin_reg.fit(X_train_std, y_train) #SKLearn's linear regression
y_train_pred = lin_reg.predict(X_train_std)
train_time = time.time()-start
```

10.0.4.12.2 Logistic Regression This algorithm can be used to perform binary classification. It can be used if you want a probabilistic framework. Also in case you expect to receive more training data in the future that you want to be able to quickly incorporate into your model.

1. Customer churn prediction.
2. Credit Scoring & Fraud Detection which is our example problem which we are trying to solve in this chapter.
3. Calculating the effectiveness of marketing campaigns.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, _, y_train, _ = train_test_split(X_train, y_train, stratify=
    ↪ y_train, train_size=subsample_rate, random_state=42)
X_test, _, y_test, _ = train_test_split(X_test, y_test, stratify=
    ↪ y_test, train_size=subsample_rate, random_state=42)

model_lr_sklearn = LogisticRegression(multi_class="multinomial", C=1
    ↪ e6, solver="sag", max_iter=15)
model_lr_sklearn.fit(X_train, y_train)

y_pred_test = model_lr_sklearn.predict(X_test)
acc = accuracy_score(y_test, y_pred_test)
results.loc[len(results)] = ["LR Sklearn", np.round(acc, 3)]
results
```

10.0.4.12.3 Decision trees Decision trees handle feature interactions and they're non-parametric. Doesnt support online learning and the entire tree needs to be rebuild when new traning dataset comes in. Memory consumption is very high.

Can be used for the following cases

1. Investment decisions
2. Customer churn
3. Banks loan defaulters
4. Build vs Buy decisions
5. Sales lead qualifications

```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
start = time.time()
dt.fit(X_train_std, y_train)
y_train_pred = dt.predict(X_train_std)
train_time = time.time() - start

start = time.time()
y_test_pred = dt.predict(X_test_std)
test_time = time.time() - start
```

10.0.4.12.4 K Means This algorithm is used when we are not aware of the labels and one needs to be created based on the features of objects. Example will be to divide a group of people into differnt subgroups based on common theme or attribute.

The main disadvantage of K-mean is that you need to know exactly the number of clusters or groups which is required. It takes a lot of iteration to come up with the best K.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV,
   ↪ PredefinedSplit
from sklearn.metrics import accuracy_score

X_train, _, y_train, _ = train_test_split(X_train, y_train, stratify=
   ↪ y_train, train_size=subsample_rate, random_state=42)
X_test, _, y_test, _ = train_test_split(X_test, y_test, stratify=
   ↪ y_test, train_size=subsample_rate, random_state=42)

model_knn_sklearn = KNeighborsClassifier(n_jobs=-1)
model_knn_sklearn.fit(X_train, y_train)

y_pred_test = model_knn_sklearn.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred_test)

results.loc[len(results)] = ["KNN Arbitary Sklearn", np.round(acc, 3)
   ↪ ]
results
```

10.0.4.12.5 Support Vector Machines SVM is a supervised ML technique and used for pattern recognition and classification problems when your data has exactly two classes. Its popular in text classification problems.

Few cases where SVM can be used is

1. Detecting persons with common diseases.
2. Hand-written character recognition
3. Text categorization
4. Stock market price prediction

10.0.4.12.6 Naive Bayes Naive Bayes is used for large datasets.This algoritm works well even when we have a limited CPU and memory available. This works by calculating bunch of counts. It requires less training data. The algorthim cant learn interation between features.

Naive Bayes can be used in real-world applications such as:

1. Sentiment analysis and text classification
2. Recommendation systems like Netflix, Amazon
3. To mark an email as spam or not spam
4. Face recognition

10.0.4.12.7 Random Forest Ranmdon forest is similar to Decision tree. Can be used for both regression and classification problems with large data sets.

Few case where it can be applied.

1. Predict patients for high risks.
2. Predict parts failures in manufacturing.
3. Predict loan defaulters.

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators = 400, criterion='mse',
   ↪ random_state=1, n_jobs=-1)
```

```
start = time.time()
forest.fit(X_train_std, y_train)
y_train_pred = forest.predict(X_train_std)
train_time = time.time() - start

start = time.time()
y_test_pred = forest.predict(X_test_std)
test_time = time.time() - start
```

10.0.4.12.8 Neural networks Neural network works based on weights of connections between neurons. Weights are trained and based on that the neural network can be utilized to predict the class or a quantity. They are resource and memory intensive.

Few cases where it can be applied.

1. Applied to unsupervised learning tasks, such as feature extraction.
2. Extracts features from raw images or speech with much less human intervention

10.0.4.12.9 Deep Learning using Keras Keras is most powerful and easy-to-use Python libraries for developing and evaluating deep learning models. It has the efficient numerical computation libraries Theano and TensorFlow.

10.0.4.12.10 XGBoost XGBoost stands for eXtreme Gradient Boosting. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It is engineered for efficiency of compute time and memory resources.

10.0.4.13 Scikit Cheat Sheet Scikit learning has put a very indepth and well explained flow chart to help you choose the right algorithm that I find very handy.

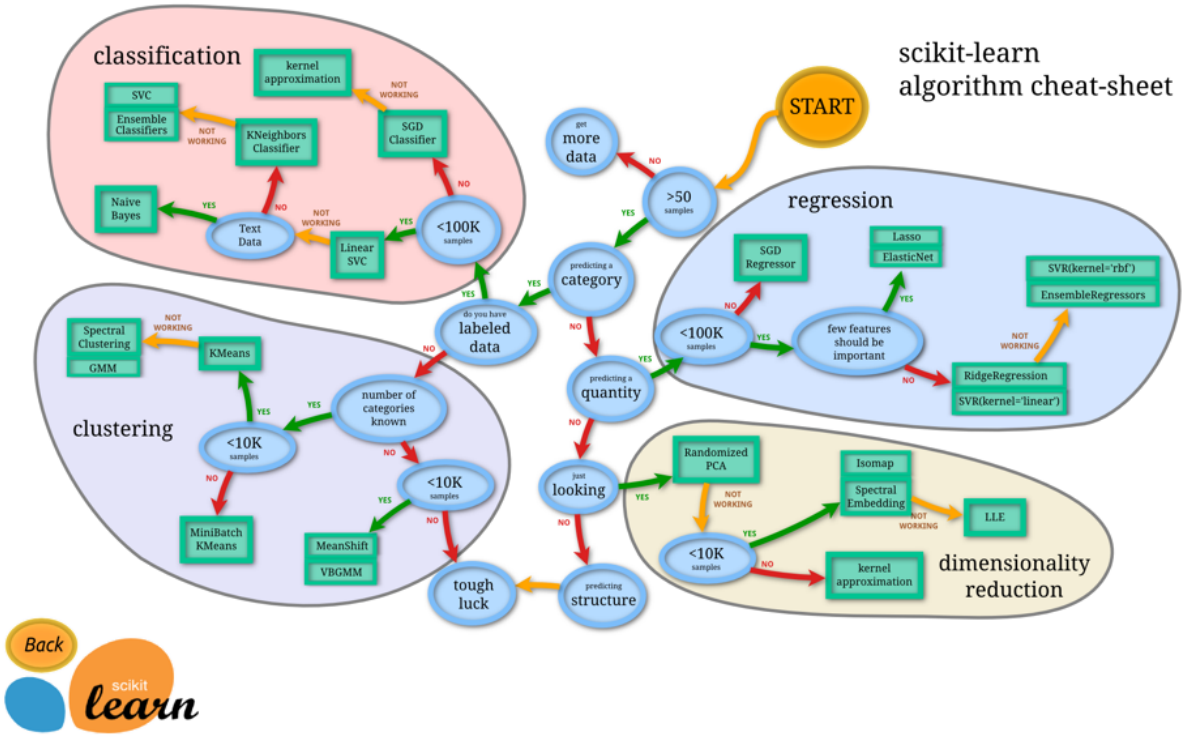


Figure 28: scikit-learn

10.0.4.14 Parameter Optimization Machine learning models are parameterized so that their behavior can be tuned for a given problem. These models can have many parameters and finding the best combination of parameters can be treated as a search problem.

A parameter is a configurationthat is part of the model and values can be derived from the given data.

1. Required by the model when making predictions.
2. Values define the skill of the model on your problem.
3. Estimated or learned from data.
4. Often not set manually by the practitioner.
5. Often saved as part of the learned model.

10.0.4.14.1 Hyperparameter optimization/tuning algorithms Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

Random search provide a statistical distribution for each hyperparameter from which values may be randomly sampled.

10.0.4.15 Experiments with Keras (deep learning), XGBoost, and SVM (SVC) compared to Logistic Regression(Baseline)

10.0.4.15.1 Creating a parameter grid

```
grid_param = [
                [{   #LogisticRegression
                   'model__penalty':['l1','l2'],
                   'model__C': [0.01, 1.0, 100]
                }],

                [{#keras
                    'model__optimizer': optimizer,
                    'model__loss': loss
                }],

                [{  #SVM
                   'model__C' :[0.01, 1.0, 100],
                   'model__gamma': [0.5, 1],
                   'model__max_iter':[-1]
                }],

                [{   #XGBClassifier
                    'model__min_child_weight': [1, 3, 5],
                    'model__gamma': [0.5],
                    'model__subsample': [0.6, 0.8],
                    'model__colsample_bytree': [0.6],
                    'model__max_depth': [3]

                }]
            ]
```

10.0.4.15.2 Implementing Grid search with models and also creating metrics from each of the model.

```
Pipeline(memory=None,
     steps=[('preparation', FeatureUnion(n_jobs=None,
       transformer_list=[('num_pipeline', Pipeline(memory=None,
     steps=[('selector', DataFrameSelector(attribute_names=['amount',
        ↪  'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', '
```

```
        ↪ newbalanceDest'])), ('attribs_adder',
        ↪ CombinedAttributesAdder()...penalty='l2', random_state=
        ↪ None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False))])
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from xgboost.sklearn import XGBClassifier
from sklearn.svm import SVC

test_scores = []
#Machine Learning Algorithm (MLA) Selection and Initialization
MLA = [
        linear_model.LogisticRegression(),
        keras_model,
        SVC(),
        XGBClassifier()

      ]

#create table to compare MLA metrics
MLA_columns = ['Name', 'Score', 'Accuracy_Score','ROC_AUC_score','
   ↪ final_rmse','Classification_error','Recall_Score','
   ↪ Precision_Score', 'mean_test_score', 'mean_fit_time', 'F1_Score
   ↪ ']
MLA_compare = pd.DataFrame(columns = MLA_columns)
Model_Scores = pd.DataFrame(columns = ['Name','Score'])

row_index = 0
for alg in MLA:

    #set name and parameters
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index, 'Name'] = MLA_name
    #MLA_compare.loc[row_index, 'Parameters'] = str(alg.get_params())
```

```
    full_pipeline_with_predictor = Pipeline([
        ("preparation", full_pipeline),  # combination of numerical
           ↪ and categorical pipelines
        ("model", alg)
    ])

    grid_search = GridSearchCV(full_pipeline_with_predictor,
       ↪ grid_param[row_index], cv=4, verbose=2, scoring='f1',
       ↪ return_train_score=True)

    grid_search.fit(X_train[X_model_col], y_train)
    y_pred = grid_search.predict(X_test)

    MLA_compare.loc[row_index, 'Accuracy_Score'] = np.round(
       ↪ accuracy_score(y_pred, y_test), 3)
    MLA_compare.loc[row_index, 'ROC_AUC_score'] = np.round(metrics.
       ↪ roc_auc_score(y_test, y_pred),3)
    MLA_compare.loc[row_index,'Score'] = np.round(grid_search.score(
       ↪ X_test, y_test),3)

    negative_mse = grid_search.best_score_
    scores = np.sqrt(-negative_mse)
    final_mse = mean_squared_error(y_test, y_pred)
    final_rmse = np.sqrt(final_mse)
    MLA_compare.loc[row_index, 'final_rmse'] = final_rmse

    confusion_matrix_var = confusion_matrix(y_test, y_pred)
    TP = confusion_matrix_var[1, 1]
    TN = confusion_matrix_var[0, 0]
    FP = confusion_matrix_var[0, 1]
    FN = confusion_matrix_var[1, 0]
    MLA_compare.loc[row_index,'Classification_error'] = np.round(((FP
       ↪  + FN) / float(TP + TN + FP + FN)), 5)
    MLA_compare.loc[row_index,'Recall_Score'] = np.round(metrics.
       ↪ recall_score(y_test, y_pred), 5)
    MLA_compare.loc[row_index,'Precision_Score'] = np.round(metrics.
       ↪ precision_score(y_test, y_pred), 5)
```

```
    MLA_compare.loc[row_index,'F1_Score'] = np.round(f1_score(y_test,
       ↪ y_pred), 5)


    MLA_compare.loc[row_index, 'mean_test_score'] = grid_search.
       ↪ cv_results_['mean_test_score'].mean()
    MLA_compare.loc[row_index, 'mean_fit_time'] = grid_search.
       ↪ cv_results_['mean_fit_time'].mean()

    Model_Scores.loc[row_index,'MLA Name'] = MLA_name
    Model_Scores.loc[row_index,'ML Score'] = np.round(metrics.
       ↪ roc_auc_score(y_test, y_pred),3)

    #Collect Mean Test scores for statistical significance test
    test_scores.append(grid_search.cv_results_['mean_test_score'])
    row_index+=1
```

```
In [108]: MLA_compare.sort_values(by = ['Score'], ascending = False, inplace = True)
          MLA_compare
```

Out[108]:

	Name	Score	Accuracy_Score	ROC_AUC_score	final_rmse	Classification_error	Recall_Score	Precision_Score	mean_test_score	mean_fit
3	XGBClassifier	1	1	1	0	0	1	1	0.815627	1
4	XGBClassifier_30%_Data	0.99	1	0.99	0.00511807	3e-05	0.97971	1	0.815627	1
5	XGBClassifier_All_Data	0.988	1	0.988	0.00557846	3e-05	0.97614	0.99975	0.815627	1
2	SVC	0.664	1	0.759	0.0205282	0.00042	0.51825	0.92208	0.372893	66
0	LogisticRegression	0.409	0.999	0.631	0.0246718	0.00061	0.26277	0.92308	0.226208	29
1	KerasClassifier	0.207	0.999	0.562	0.0275839	0.00076	0.12409	0.62963	0.0918313	5

Figure 29: scikit-learn

10.0.4.15.3 Results table from the Model evaluation with metrics.

10.0.4.15.4 ROC AUC Score AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

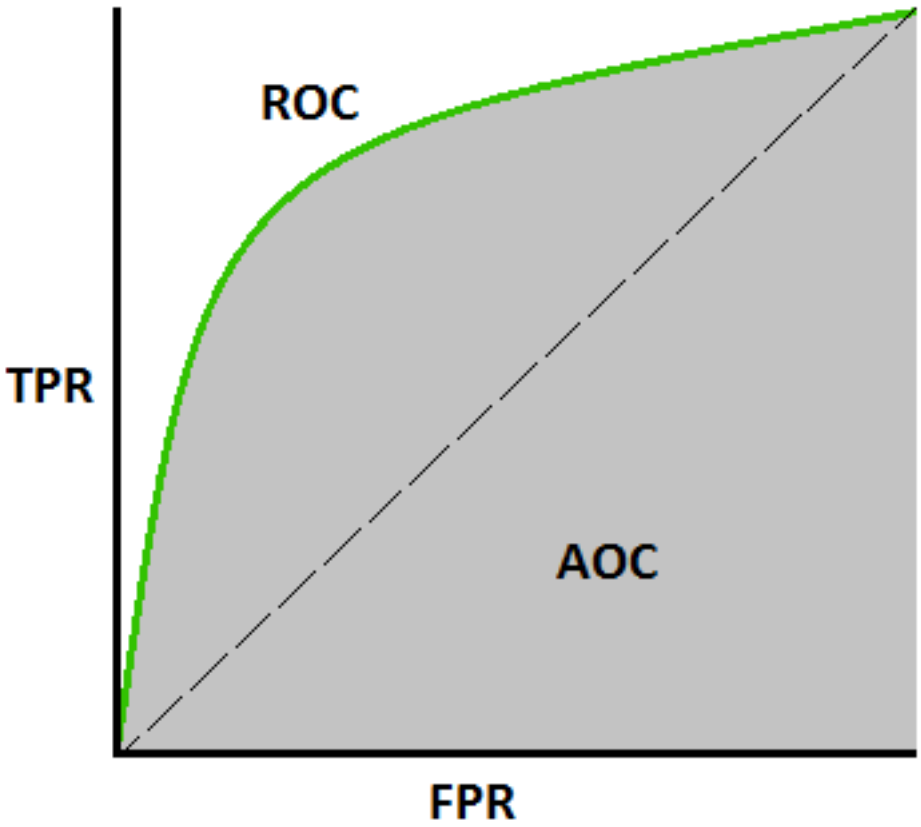


Figure 30: scikit-learn

```
In [109]: sns.barplot(x='ROC_AUC_score', y = 'Name', data = MLA_compare, color = 'm')

          plt.title('Machine Learning Algorithm ROC AUC Score \n')
          plt.xlabel('ROC_AUC Score (%)')
          plt.ylabel('Algorithm')
```

```
Out[109]: Text(0,0.5,'Algorithm')
```

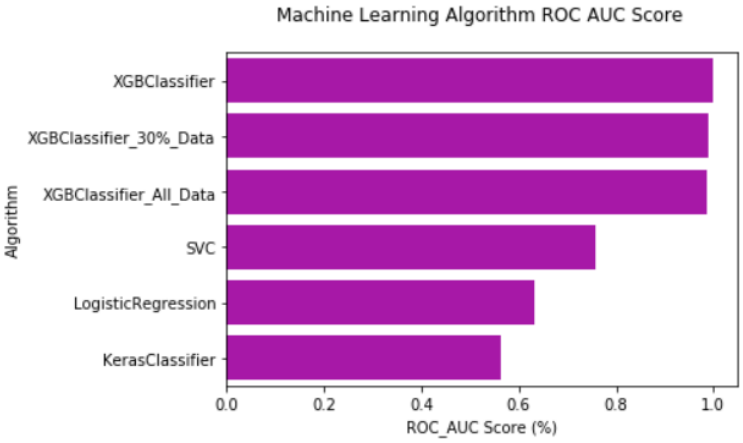


Figure 31: scikit-learn

10.0.4.16 K-means in scikit learn.

10.0.4.16.1 Import

10.0.4.17 K-means Algorithm In this section we demonstrate how simple it is to use k-means in scikit learn.

10.0.4.17.1 Import

```
from time import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

10.0.4.17.2 Create samples

```
    np.random.seed(42)

    digits = load_digits()
    data = scale(digits.data)
```

10.0.4.17.3 Create samples

```
    np.random.seed(42)

    digits = load_digits()
    data = scale(digits.data)

    n_samples, n_features = data.shape
    n_digits = len(np.unique(digits.target))
    labels = digits.target

    sample_size = 300

    print("n_digits: %d, \t n_samples %d, \t n_features %d" % (
       ↪ n_digits, n_samples, n_features))
    print(79 * '_')
    print('% 9s' % 'init' '    time  inertia    homo   compl  v-meas
       ↪     ARI AMI  silhouette')
    print("n_digits: %d, \t n_samples %d, \t n_features %d"
          % (n_digits, n_samples, n_features))


    print(79 * '_')
    print('% 9s' % 'init'
          '    time  inertia    homo   compl  v-meas     ARI AMI
             ↪ silhouette')


    def bench_k_means(estimator, name, data):
        t0 = time()
        estimator.fit(data)
        print('% 9s   %.2fs    %i   %.3f   %.3f   %.3f   %.3f   %.3f
           ↪    %.3f'
```

```
              % (name, (time() - t0), estimator.inertia_,
                 metrics.homogeneity_score(labels, estimator.labels_)
                    ↪ ,
                 metrics.completeness_score(labels, estimator.labels_
                    ↪ ),
                 metrics.v_measure_score(labels, estimator.labels_),
                 metrics.adjusted_rand_score(labels, estimator.
                    ↪ labels_),
                 metrics.adjusted_mutual_info_score(labels,
                    ↪ estimator.labels_),

                 metrics.silhouette_score(data, estimator.labels_,
                    ↪ metric='euclidean',sample_size=sample_size)))

    bench_k_means(KMeans(init='k-means++', n_clusters=n_digits,
       ↪ n_init=10), name="k-means++", data=data)

    bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init
       ↪ =10), name="random", data=data)

                 metrics.silhouette_score(data, estimator.labels_,
                                          metric='euclidean',
                                          sample_size=sample_size)))

    bench_k_means(KMeans(init='k-means++', n_clusters=n_digits,
       ↪ n_init=10),
                  name="k-means++", data=data)

    bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init
       ↪ =10),
                  name="random", data=data)


    # in this case the seeding of the centers is deterministic, hence
       ↪  we run the
    # kmeans algorithm only once with n_init=1
    pca = PCA(n_components=n_digits).fit(data)
```

```
    bench_k_means(KMeans(init=pca.components_,n_clusters=n_digits,
       ↪ n_init=1),name="PCA-based", data=data)
    print(79 * '_')
```

10.0.4.17.4 Visualize See Figure 32

```
    bench_k_means(KMeans(init=pca.components_,
                         n_clusters=n_digits, n_init=1),
                  name="PCA-based",
                  data=data)
    print(79 * '_')
```

10.0.4.17.5 Visualize See Figure 32

```
    reduced_data = PCA(n_components=2).fit_transform(data)
    kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
    kmeans.fit(reduced_data)

    # Step size of the mesh. Decrease to increase the quality of the
       ↪ VQ.
    h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

    # Plot the decision boundary. For that, we will assign a color to
       ↪  each
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].
       ↪ max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].
       ↪ max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
       ↪  y_max, h))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(1)
```

```
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize
       ↪ =2)
    # Plot the centroids as a white X
    centroids = kmeans.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('K-means clustering on the digits dataset (PCA-reduced
       ↪ data)\n'
              'Centroids are marked with white cross')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

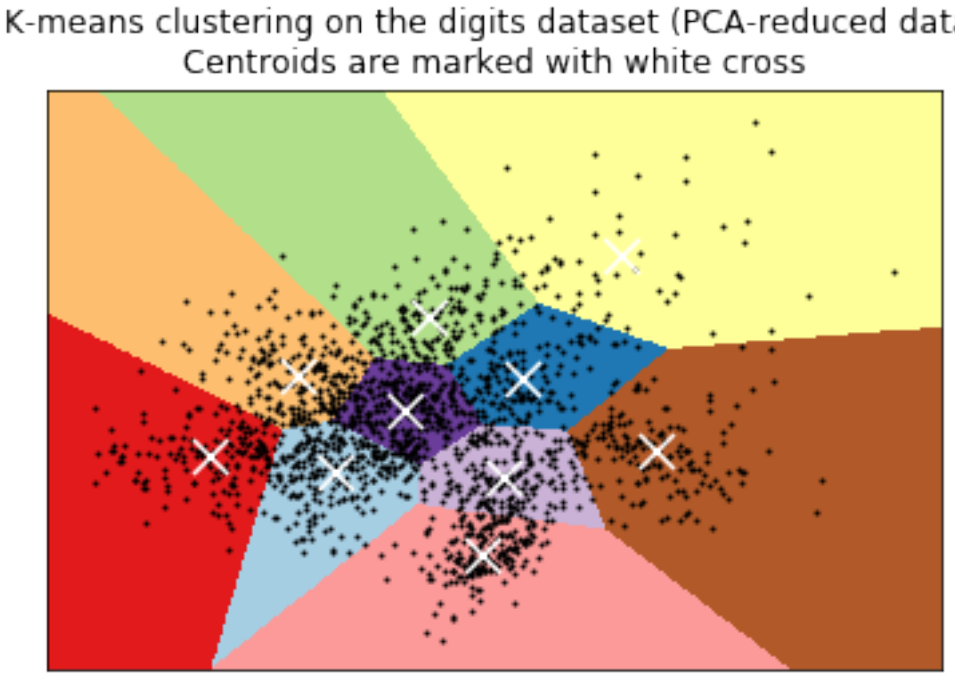


Figure 32: Result

10.0.5 Dask - Random Forest Feature Detection

10.0.5.1 Setup First we need our tools. pandas gives us the DataFrame, very similar to R's DataFrames. The DataFrame is a structure that allows us to work with our data more easily. It has nice features for slicing and transformation of data, and easy ways to do basic statistics.

numpy has some very handy functions that work on DataFrames.

10.0.5.2 Dataset We are using a dataset about the wine quality dataset, archived at UCI's Machine Learning Repository (http://archive.ics.uci.edu/ml/index.php).

```
import pandas as pd
import numpy as np
```

Now we will load our data. pandas makes it easy!

```
# red wine quality data, packed in a DataFrame
red_df = pd.read_csv('winequality-red.csv',sep=';',header=0,
    ↪ index_col=False)
```

```
# white wine quality data, packed in a DataFrame
white_df = pd.read_csv('winequality-white.csv',sep=';',header=0,
   ↪ index_col=False)

# rose? other fruit wines? plum wine? :(
```

Like in R, there is a .describe() method that gives basic statistics for every column in the dataset.

```
# for red wines
red_df.describe()
```

fixed acidity

volatile acidity

citric acid

residual sugar

chlorides

free sulfur dioxide

total sulfur dioxide

density

pH

sulphates

alcohol

quality

count

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

1599.000000

mean

8.319637

0.527821

0.270976

2.538806

0.087467

15.874922

46.467792

0.996747

3.311113

0.658149

10.422983

5.636023

std

1.741096

0.179060

0.194801

1.409928

0.047065

10.460157

32.895324

0.001887

0.154386

0.169507

1.065668

0.807569

min

4.600000

0.120000

0.000000

0.900000

0.012000

1.000000

6.000000

0.990070

2.740000

0.330000

8.400000

3.000000

25%

7.100000

0.390000

0.090000

1.900000

0.070000

7.000000

22.000000

0.995600

3.210000

0.550000

9.500000

5.000000

50%

7.900000

0.520000

0.260000

2.200000

0.079000

14.000000

38.000000

0.996750

3.310000

0.620000

10.200000

6.000000

75%

9.200000

0.640000

0.420000

2.600000

0.090000

21.000000

62.000000

0.997835

3.400000

0.730000

11.100000

6.000000

max

15.900000

1.580000

1.000000

15.500000

0.611000

72.000000

289.000000

1.003690

4.010000

2.000000

14.900000

8.000000

```
# for white wines
white_df.describe()
```

fixed acidity

volatile acidity

citric acid

residual sugar

chlorides

free sulfur dioxide

total sulfur dioxide

density

pH

sulphates

alcohol

quality

count

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

4898.000000

mean

6.854788

0.278241

0.334192

6.391415

0.045772

35.308085

138.360657

0.994027

3.188267

0.489847

10.514267

5.877909

std

0.843868

0.100795

0.121020

5.072058

0.021848

17.007137

42.498065

0.002991

0.151001

0.114126

1.230621

0.885639

min

3.800000

0.080000

0.000000

0.600000

0.009000

2.000000

9.000000

0.987110

2.720000

0.220000

8.000000

3.000000

25%

6.300000

0.210000

0.270000

1.700000

0.036000

23.000000

108.000000

0.991723

3.090000

0.410000

9.500000

5.000000

50%

6.800000

0.260000

0.320000

5.200000

0.043000

34.000000

134.000000

0.993740

3.180000

0.470000

10.400000

6.000000

75%

7.300000

0.320000

0.390000

9.900000

0.050000

46.000000

167.000000

0.996100

3.280000

0.550000

11.400000

6.000000

max

14.200000

1.100000

1.660000

65.800000

0.346000

289.000000

440.000000

1.038980

3.820000

1.080000

14.200000

9.000000

Sometimes it is easier to understand the data visually. A histogram of the white wine quality *data citric* acid samples is shown next. You can of course visualize other columns' data or other datasets. Just replace the DataFrame and column name (see Figure 33).

```
import matplotlib.pyplot as plt

def extract_col(df,col_name):
    return list(df[col_name])

col = extract_col(white_df,'citric acid') # can replace with another
   ↪ dataframe or column
plt.hist(col)

#TODO: add axes and such to set a good example
```

```
plt.show()
```

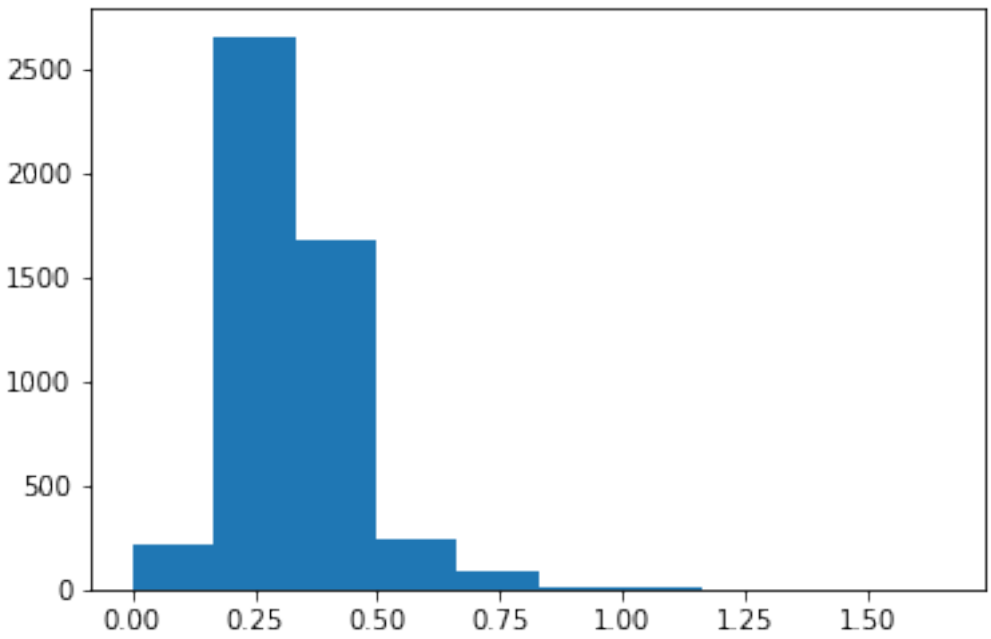


Figure 33: Histogram

10.0.5.3 Detecting Features Let us try out a some elementary machine learning models. These models are not always for prediction. They are also useful to find what features are most predictive of a variable of interest. Depending on the classifier you use, you may need to transform the data pertaining to that variable.

10.0.5.3.1 Data Preparation Let us assume we want to study what features are most correlated with pH. pH of course is real-valued, and continuous. The classifiers we want to use usually need labeled or integer data. Hence, we will transform the pH data, assigning wines with pH higher than average as `hi` (more basic or alkaline) and wines with pH lower than average as `lo` (more acidic).

```
# refresh to make Jupyter happy
red_df = pd.read_csv('winequality-red.csv',sep=';',header=0,
   ↪ index_col=False)
white_df = pd.read_csv('winequality-white.csv',sep=';',header=0,
   ↪ index_col=False)
```

```
#TODO: data cleansing functions here, e.g. replacement of NaN

# if the variable you want to predict is continuous, you can map
   ↪ ranges of values
# to integer/binary/string labels

# for example, map the pH data to 'hi' and 'lo' if a pH value is more
   ↪  than or
# less than the mean pH, respectively
M = np.mean(list(red_df['pH'])) # expect inelegant code in these
   ↪ mappings
Lf = lambda p: int(p < M)*'lo' + int(p >= M)*'hi' # some C-style
   ↪ hackery

# create the new classifiable variable
red_df['pH-hi-lo'] = map(Lf,list(red_df['pH']))

# and remove the predecessor
del red_df['pH']
```

Now we specify which dataset and variable you want to predict by assigning vlues to `SELECTED_DF` and `TARGET_VAR`, respectively.

We like to keep a parameter file where we specify data sources and such. This lets me create generic analytics code that is easy to reuse.

After we have specified what dataset we want to study, we split the training and test datasets. We then scale (normalize) the data, which makes most classifiers run better.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics

# make selections here without digging in code
SELECTED_DF = red_df # selected dataset
TARGET_VAR = 'pH-hi-lo' # the predicted variable

# generate nameless data structures
df = SELECTED_DF
```

```
target = np.array(df[TARGET_VAR]).ravel()
del df[TARGET_VAR] # no cheating

#TODO: data cleansing function calls here

# split datasets for training and testing
X_train, X_test, y_train, y_test = train_test_split(df,target,
   ↪ test_size=0.2)

# set up the scaler
scaler = StandardScaler()
scaler.fit(X_train)

# apply the scaler
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Now we pick a classifier. As you can see, there are many to try out, and even more in scikit-learn's documentation and many examples and tutorials. Random Forests are data science workhorses. They are the go-to method for most data scientists. Be careful relying on them though–they tend to overfit. We try to avoid overfitting by separating the training and test datasets.

10.0.5.4 Random Forest

```
# pick a classifier

from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
   ↪ ,ExtraTreeClassifier,ExtraTreeRegressor
from sklearn.ensemble import RandomForestClassifier,
   ↪ ExtraTreesClassifier

clf = RandomForestClassifier()
```

Now we will test it out with the default parameters.

Note that this code is boilerplate. You can use it interchangeably for most scikit-learn models.

```
# test it out

model = clf.fit(X_train,y_train)
```

```
pred = clf.predict(X_test)
conf_matrix = metrics.confusion_matrix(y_test,pred)

var_score = clf.score(X_test,y_test)

# the results
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
```

Now output the results. For Random Forests, we get a feature ranking. Relative importances usually exponentially decay. The first few highly-ranked features are usually the most important.

```
# for the sake of clarity
num_features = X_train.shape[1]
features = map(lambda x: df.columns[x],indices)
feature_importances = map(lambda x: importances[x],indices)

print 'Feature ranking:\n'

for i in range(num_features):
    feature_name = features[i]
    feature_importance = feature_importances[i]
    print '%s%f' % (feature_name.ljust(30), feature_importance)
```

```
Feature ranking:

fixed acidity                 0.269778
citric acid                   0.171337
density                       0.089660
volatile acidity              0.088965
chlorides                     0.082945
alcohol                       0.080437
total sulfur dioxide          0.067832
sulphates                     0.047786
free sulfur dioxide           0.042727
residual sugar                0.037459
quality                       0.021075
```

Sometimes it's easier to visualize. We'll use a bar chart. See Figure 34

```
plt.clf()
plt.bar(range(num_features),feature_importances)
plt.xticks(range(num_features),features,rotation=90)
plt.ylabel('relative importance (a.u.)')
plt.title('Relative importances of most predictive features')
plt.show()
```

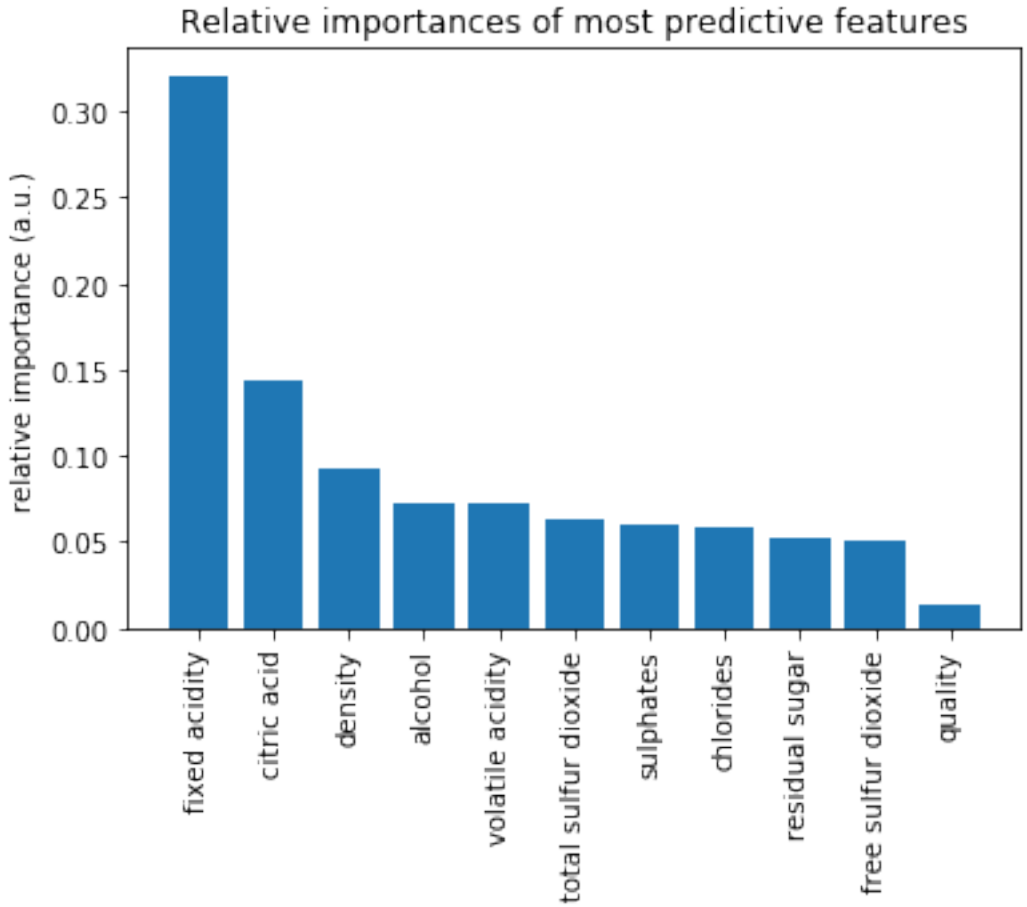


Figure 34: Result

```
import dask.dataframe as dd

red_df = dd.read_csv('winequality-red.csv',sep=';',header=0)
white_df = dd.read_csv('winequality-white.csv',sep=';',header=0)
```

10.0.5.5 Acknowledgement This notebook was developed by Juliette Zerick and Gregor von Laszewski

10.0.6 Parallel Computing in Python

In this module, we will review the available Python modules that can be used for parallel computing. Parallel computing can be in the form of either multi-threading or multi-processing. In multi-threading approach, the threads run in the same shared memory heap whereas in case of multi-processing, the memory heaps of processes are separate and independent, therefore the communication between the processes are a little bit more complex.

10.0.6.1 Multi-threading in Python Threading in Python is perfect for I/O operations where the process is expected to be idle regularly, e.g. web scraping. This is a very useful feature because several applications and scripts might spend the majority of their runtime waiting for network or data I/O. In several cases, e.g. web scraping, the resources, i.e. downloading from different websites, are most of the time-independent. Therefore the processor can download in parallel and join the result at the end.

10.0.6.1.1 `Thread` vs `Threading` There are two built-in modules in Python that are related to threading, namely `thread` and `threading`. The former module is deprecated for some time in `Python 2`, and in `Python 3` it is renamed to `_thread` for the sake of backward incompatibilities. The `_thread` module provides low-level threading API for multi-threading in Python, whereas the module `threading` builds a high-level threading interface on top of it.

The `Thread()` is the main method of the `threading` module, the two important arguments of which are `target`, for specifying the callable object, and `args` to pass the arguments for the target callable. We illustrate these in the following example:

```
import threading

def hello_thread(thread_num):
    print ("Hello from Thread ", thread_num)

if __name__ == '__main__':
    for thread_num in range(5):
        t = threading.Thread(target=hello_thread,arg=(thread_num,))
        t.start()
```

This is the output of the previous example:

```
In [1]: %run threading.py
Hello from Thread  0
Hello from Thread  1
Hello from Thread  2
Hello from Thread  3
Hello from Thread  4
```

In case you are not familiar with the `if __name__ == '__main__:'` statement, what it does is making sure that the code nested under this condition will be run only if you run your module as a program and it will not run in case your module is imported into another file.

10.0.6.1.2 Locks As mentioned prior, the memory space is shared between the threads. This is at the same time beneficial and problematic: it is beneficial in a sense that the communication between the threads becomes easy, however, you might experience a strange outcome if you let several threads change the same variable without caution, e.g. thread 2 changes variable `x` while thread 1 is working with it. This is when `lock` comes into play. Using `lock`, you can allow only one thread to work with a variable. In other words, only a single thread can hold the `lock`. If the other threads need to work with that variable, they have to wait until the other thread is done and the variable is "unlocked".

We illustrate this with a simple example:

```
import threading

global counter
counter = 0

def incrementer1():
    global counter
    for j in range(2):
        for i in range(3):
            counter += 1
            print("Greeter 1 incremented the counter by 1")
        print ("Counter is %d"%counter)

def incrementer2():
    global counter
    for j in range(2):
```

```
        for i in range(3):
            counter += 1
            print("Greeter 2 incremented the counter by 1")
        print ("Counter is now %d"%counter)


if __name__ == '__main__':
    t1 = threading.Thread(target = incrementer1)
    t2 = threading.Thread(target = incrementer2)

    t1.start()
    t2.start()
```

Suppose we want to print multiples of 3 between 1 and 12, i.e. 3, 6, 9 and 12. For the sake of argument, we try to do this using 2 threads and a nested for loop. Then we create a global variable called counter and we initialize it with 0. Then whenever each of the `incrementer1` or `incrementer2` functions are called, the `counter` is incremented by 3 twice (counter is incremented by 6 in each function call). If you run the previous code, you should be really lucky if you get the following as part of your output:

```
Counter is now 3
Counter is now 6
Counter is now 9
Counter is now 12
```

The reason is the conflict that happens between threads while incrementing the `counter` in the nested for loop. As you probably noticed, the first level for loop is equivalent to adding 3 to the counter and the conflict that might happen is not effective on that level but the nested for loop. Accordingly, the output of the previous code is different in every run. This is an example output:

```
$ python3 lock_example.py
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Counter is 4
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Greeter 1 incremented the counter by 1
Greeter 2 incremented the counter by 1
```

```
Greeter 1 incremented the counter by 1
Counter is 8
Greeter 1 incremented the counter by 1
Greeter 2 incremented the counter by 1
Counter is 10
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Counter is 12
```

We can fix this issue using a `lock` : whenever one of the function is going to increment the value by 3, it will `acquire()` the lock and when it is done the function will `release()` the lock. This mechanism is illustrated in the following code:

```
import threading

increment_by_3_lock = threading.Lock()

global counter
counter = 0

def incrementer1():
    global counter
    for j in range(2):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter += 1
            print("Greeter 1 incremented the counter by 1")
        print ("Counter is %d"%counter)
        increment_by_3_lock.release()

def incrementer2():
    global counter
    for j in range(2):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter += 1
            print("Greeter 2 incremented the counter by 1")
        print ("Counter is %d"%counter)
```

```
        increment_by_3_lock.release()

if __name__ == '__main__':
    t1 = threading.Thread(target = incrementer1)
    t2 = threading.Thread(target = incrementer2)

    t1.start()
    t2.start()
```

No matter how many times you run this code, the output would always be in the correct order:

```
$ python3 lock_example.py
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Counter is 3
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Greeter 1 incremented the counter by 1
Counter is 6
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Counter is 9
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Greeter 2 incremented the counter by 1
Counter is 12
```

Using the `Threading` module increases both the overhead associated with thread management as well as the complexity of the program and that is why in many situations, employing `multiprocessing` module might be a better approach.

10.0.6.2 Multi-processing in Python We already mentioned that multi-threading might not be sufficient in many applications and we might need to use `multiprocessing` sometimes, or better to say most of the time. That is why we are dedicating this subsection to this particular module. This module provides you with an API for spawning processes the way you spawn threads using

`threading` module. Moreover, some functionalities are not even available in `threading` module, e.g. the `Pool` class which allows you to run a batch of jobs using a *pool* of worker processes.

10.0.6.2.1 Process Similar to `threading` module which was employing `thread` (aka `_thread`) under the hood, `multiprocessing` employs the `Process` class. Consider the following example:

```
from multiprocessing import Process
import os

def greeter (name):
    proc_idx = os.getpid()
    print ("Process {0}: Hello {1}!".format(proc_idx,name))

if __name__ == '__main__':
    name_list = ['Harry', 'George', 'Dirk', 'David']
    process_list = []
    for name_idx, name in enumerate(name_list):
        current_process = Process(target=greeter, args=(name,))
        process_list.append(current_process)
        current_process.start()
    for process in process_list:
        process.join()
```

In this example, after importing the `Process` module we created a `greeter()` function that takes a `name` and greets that person. It also prints the `pid` (process identifier) of the process that is running it. Note that we used the `os` module to get the `pid`. In the bottom of the code after checking the `__name__='__main__'` condition, we create a series of `Process`es and `start` them. Finally in the last for loop and using the `join` method, we tell Python to wait for the processes to terminate. This is one of the possible outputs of the code:

```
$ python3 process_example.py
Process 23451: Hello Harry!
Process 23452: Hello George!
Process 23453: Hello Dirk!
Process 23454: Hello David!
```

10.0.6.2.2 Pool Consider the `Pool` class as a pool of worker processes. There are several ways for assigning jobs to the `Pool` class and we will introduce the most important ones in this section. These methods are categorized as `blocking` or `non-blocking`. The former means that after calling the API, it blocks the thread/process until it has the result or answer ready and the control returns only when the call completes. In the `non-blockin` on the other hand, the control returns immediately.

Synchronous `Pool.map()`

We illustrate the `Pool.map` method by re-implementing our previous greeter example using `Pool.map`:

```
from multiprocessing import Pool
import os

def greeter(name):
    pid = os.getpid()
    print("Process {0}: Hello {1}!".format(pid,name))

if __name__ == '__main__':
    names = ['Jenna', 'David','Marry', 'Ted','Jerry','Tom','Justin']
    pool = Pool(processes=3)
    sync_map = pool.map(greeter,names)
    print("Done!")
```

As you can see, we have seven names here but we do not want to dedicate each greeting to a separate process. Instead, we do the whole job of "greeting seven people" using "two processes". We create a pool of 3 processes with `Pool(processes=3)` syntax and then we map an iterable called `names` to the `greeter` function using `pool.map(greeter,names)`. As we expected, the greetings in the output will be printed from three different processes:

```
$ python poolmap_example.py
Process 30585: Hello Jenna!
Process 30586: Hello David!
Process 30587: Hello Marry!
Process 30585: Hello Ted!
Process 30585: Hello Jerry!
Process 30587: Hello Tom!
Process 30585: Hello Justin!
Done!
```

Note that `Pool.map()` is in `blocking` category and does not return the control to your script until it is done calculating the results. That is why `Done!` is printed after all of the greetings are over.

Asynchronous `Pool.map_async()`

As the name implies, you can use the `map_async` method, when you want assign many function calls to a pool of worker processes asynchronously. Note that unlike `map`, the order of the results is not guaranteed (as oppose to `map`) and the control is returned immediately. We now implement the previous example using `map_async`:

```
from multiprocessing import Pool
import os

def greeter(name):
    pid = os.getpid()
    print("Process {0}: Hello {1}!".format(pid,name))

if __name__ == '__main__':
    names = ['Jenna', 'David','Marry', 'Ted','Jerry','Tom','Justin']
    pool = Pool(processes=3)
    async_map = pool.map_async(greeter,names)
    print("Done!")
    async_map.wait()
```

As you probably noticed, the only difference (clearly apart from the `map_async` method name) is calling the `wait()` method in the last line. The `wait()` method tells your script to wait for the result of `map_async` before terminating:

```
$ python poolmap_example.py
Done!
Process 30740: Hello Jenna!
Process 30741: Hello David!
Process 30740: Hello Ted!
Process 30742: Hello Marry!
Process 30740: Hello Jerry!
Process 30741: Hello Tom!
Process 30742: Hello Justin!
```

Note that the order of the results are not preserved. Moreover, `Done!` is printer before any of the

results, meaning that if we do not use the `wait()` method, you probably will not see the result at all.

10.0.6.2.3 Locks The way `multiprocessing` module implements locks is almost identical to the way the `threading` module does. After importing `Lock` from `multiprocessing` all you need to do is to `acquire` it, do some computation and then `release` the lock. We will clarify the use of `Lock` by providing an example in next section about process communication.

10.0.6.2.4 Process Communication Process communication in `multiprocessing` is one of the most important, yet complicated, features for better use of this module. As oppose to `threading`, the `Process` objects will not have access to any shared variable by default, i.e. no shared memory space between the processes by default. This effect is illustrated in the following example:

```
from multiprocessing import Process, Lock, Value
import time

global counter
counter = 0

def incrementer1():
    global counter
    for j in range(2):
        for i in range(3):
            counter += 1
        print ("Greeter1: Counter is %d"%counter)

def incrementer2():
    global counter
    for j in range(2):
        for i in range(3):
            counter += 1
        print ("Greeter2: Counter is %d"%counter)


if __name__ == '__main__':

    t1 = Process(target = incrementer1 )
```

```
    t2 = Process(target = incrementer2 )
    t1.start()
    t2.start()
```

Probably you already noticed that this is almost identical to our example in `threading` section. Now, take a look at the strange output:

```
$ python communication_example.py
Greeter1: Counter is 3
Greeter1: Counter is 6
Greeter2: Counter is 3
Greeter2: Counter is 6
```

As you can see, it is as if the processes does not see each other. Instead of having two processes one counting to 6 and the other counting from 6 to 12, we have two processes counting to 6.

Nevertheless, there are several ways that `Process`es from `multiprocessing` can communicate with each other, including `Pipe`, `Queue`, `Value`, `Array` and `Manager`. `Pipe` and `Queue` are appropriate for inter-process message passing. To be more specific, `Pipe` is useful for process-to-process scenarios while `Queue` is more appropriate for process**es**-toprocess**es** ones. `Value` and `Array` are both used to provide synchronized access to a shared data (very much like shared memory) and `Managers` can be used on different data types. In the following sub-sections, we cover both `Value` and `Array` since they are both lightweight, yet useful, approach.

Value

The following example re-implements the broken example in the previous section. We fix the strange output, by using both `Lock` and `Value`:

```
from multiprocessing import Process, Lock, Value
import time

increment_by_3_lock = Lock()


def incrementer1(counter):
    for j in range(3):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter.value += 1
```

```
            time.sleep(0.1)
        print ("Greeter1: Counter is %d"%counter.value)
        increment_by_3_lock.release()

def incrementer2(counter):
    for j in range(3):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter.value += 1
            time.sleep(0.05)
        print ("Greeter2: Counter is %d"%counter.value)
        increment_by_3_lock.release()


if __name__ == '__main__':

    counter = Value('i',0)
    t1 = Process(target = incrementer1, args=(counter,))
    t2 = Process(target = incrementer2 , args=(counter,))
    t2.start()
    t1.start()
```

The usage of `Lock` object in this example is identical to the example in `threading` section. The usage of `counter` is on the other hand the novel part. First, note that counter is not a global variable anymore and instead it is a `Value` which returns a `ctypes` object allocated from a shared memory between the processes. The first argument `'i'` indicates a signed integer, and the second argument defines the initialization value. In this case we are assigning a signed integer in the shared memory initialized to size 0 to the `counter` variable. We then modified our two functions and pass this *shared* variable as an argument. Finally, we change the way we increment the `counter` since the counter is not a Python integer anymore but a `ctypes` signed integer where we can access its value using the `value` attribute. The output of the code is now as we expected:

```
$ python mp_lock_example.py
Greeter2: Counter is 3
Greeter2: Counter is 6
Greeter1: Counter is 9
Greeter1: Counter is 12
```

The last example related to parallel processing, illustrates the use of both `Value` and `Array`, as well as a technique to pass multiple arguments to a function. Note that the `Process` object does not accept multiple arguments for a function and therefore we need this or similar techniques for passing multiple arguments. Also, this technique can also be used when you want to pass multiple arguments to `map` or `map_async`:

```
from multiprocessing import Process, Lock, Value, Array
import time
from ctypes import c_char_p


increment_by_3_lock = Lock()


def incrementer1(counter_and_names):
    counter=  counter_and_names[0]
    names = counter_and_names[1]
    for j in range(2):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter.value += 1
            time.sleep(0.1)
        name_idx = counter.value//3 -1
        print ("Greeter1: Greeting {0}! Counter is {1}".format(names.
           ↪ value[name_idx],counter.value))
        increment_by_3_lock.release()

def incrementer2(counter_and_names):
    counter=  counter_and_names[0]
    names = counter_and_names[1]
    for j in range(2):
        increment_by_3_lock.acquire(True)
        for i in range(3):
            counter.value += 1
            time.sleep(0.05)
        name_idx = counter.value//3 -1
        print ("Greeter2: Greeting {0}! Counter is {1}".format(names.
           ↪ value[name_idx],counter.value))
```

```
        increment_by_3_lock.release()


if __name__ == '__main__':
    counter = Value('i',0)
    names = Array (c_char_p,4)
    names.value = ['James','Tom','Sam', 'Larry']
    t1 = Process(target = incrementer1, args=((counter,names),))
    t2 = Process(target = incrementer2 , args=((counter,names),))
    t2.start()
    t1.start()
```

In this example, we created a `multiprocessing.Array()` object and assigned it to a variable called `names`. As we mentioned before, the first argument is the `ctype` data type and since we want to create an array of strings with a length of 4 (second argument), we imported the `c_char_p` and passed it as the first argument.

Instead of passing the arguments separately, we merged both the `Value` and `Array` objects in a tuple and passed the tuple to the functions. We then modified the functions to unpack the objects in the first two lines in both functions. Finally, we changed the print statement in a way that each process greets a particular name. The output of the example is:

```
$ python3 mp_lock_example.py
Greeter2: Greeting James! Counter is 3
Greeter2: Greeting Tom! Counter is 6
Greeter1: Greeting Sam! Counter is 9
Greeter1: Greeting Larry! Counter is 12
```

10.0.7 Dask

Dask is a python-based parallel computing library for analytics. Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved concurrently.

Dask is composed of two components:

1. *Dynamic task scheduling optimized for computation.* This is similar to Airflow, Luigi, Celery, or Make, but optimized for interactive computational workloads.

2. *Big Data collections* like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments. These parallel collections run on top of the dynamic task schedulers.

Dask emphasizes the following virtues:

- *Familiar*: Provides parallelized NumPy array and Pandas DataFrame objects.
- *Flexible*: Provides a task scheduling interface for more custom workloads and integration with other projects.
- *Native*: Enables distributed computing in Pure Python with access to the PyData stack.
- *Fast*: Operates with low overhead, low latency, and minimal serialization necessary for fast numerical algorithms
- *Scales up*: Runs resiliently on clusters with 1000s of cores
- *Scales down*: Trivial to set up and run on a laptop in a single process
- *Responsive*: Designed with interactive computing in mind it provides rapid feedback and diagnostics to aid humans

The section is structured in a number of subsections addressing the following topics:

Foundations: an explanation of what Dask is, how it works, and how to use lower level primitives to set up computations. Casual users may wish to skip this section, although we consider it useful knowledge for all users.

Distributed Features: information on running Dask on the distributed scheduler, which enables scale-up to distributed settings and enhanced monitoring of task operations. The distributed scheduler is now generally the recommended engine for executing task work, even on single workstations or laptops.

Collections: convenient abstractions giving a familiar feel to big data.

Bags: Python iterators with a functional paradigm, such as found in func/iter-tools and toolz - generalize lists/generators to big data; this will seem very familiar to users of PySpark's RDD

Array: massive multi-dimensional numerical data, with Numpy functionality

Dataframe: massive tabular data, with Pandas functionality

10.0.7.1 How Dask Works

Dask is a computation tool for larger-than-memory datasets, parallel execution or delayed/background execution.

We can summarize the basics of Dask as follows:

- process data that does not fit into memory by breaking it into blocks and specifying task chains
- parallelize execution of tasks across cores and even nodes of a cluster

- move computation to the data rather than the other way around, to minimize communication overheads

We use for-loops to build basic tasks, Python iterators, and the Numpy (array) and Pandas (dataframe) functions for multi-dimensional or tabular data, respectively.

Dask allows us to construct a prescription for the calculation we want to carry out. A module named Dask.delayed lets us parallelize custom code. It is useful whenever our problem doesn't quite fit a high-level parallel object like dask.array or dask.dataframe but could still benefit from parallelism. Dask.delayed works by delaying our function evaluations and putting them into a dask graph. Here is a small example:

```
from dask import delayed

@delayed
def inc(x):
    return x + 1

@delayed
def add(x, y):
    return x + y
```

Here we have used the delayed annotation to show that we want these functions to operate lazily - to save the set of inputs and execute only on demand.

10.0.7.2 Dask Bag Dask-bag excels in processing data that can be represented as a sequence of arbitrary inputs. We'll refer to this as "messy" data, because it can contain complex nested structures, missing fields, mixtures of data types, etc. The functional programming style fits very nicely with standard Python iteration, such as can be found in the itertools module.

Messy data is often encountered at the beginning of data processing pipelines when large volumes of raw data are first consumed. The initial set of data might be JSON, CSV, XML, or any other format that does not enforce strict structure and datatypes. For this reason, the initial data massaging and processing is often done with Python lists, dicts, and sets.

These core data structures are optimized for general-purpose storage and processing. Adding streaming computation with iterators/generator expressions or libraries like itertools or toolz let us process large volumes in a small space. If we combine this with parallel processing then we can churn through a fair amount of data.

Dask.bag is a high level Dask collection to automate common workloads of this form. In a nutshell

```
dask.bag = map, filter, toolz + parallel execution
```

You can create a Bag from a Python sequence, from files, from data on S3, etc.

```
# each element is an integer
import dask.bag as db
b = db.from_sequence([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# each element is a text file of JSON lines
import os
b = db.read_text(os.path.join('data', 'accounts.*.json.gz'))

# Requires `s3fs` library
# each element is a remote CSV text file
b = db.read_text('s3://dask-data/nyc-taxi/2015/yellow_tripdata_2015
   ↪ -01.csv')
```

Bag objects hold the standard functional API found in projects like the Python standard library, toolz, or pyspark, including map, filter, groupby, etc.

As with Array and DataFrame objects, operations on Bag objects create new bags. Call the .compute() method to trigger execution.

```
def is_even(n):
    return n % 2 == 0

b = db.from_sequence([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
c = b.filter(is_even).map(lambda x: x ** 2)
c

# blocking form: wait for completion (which is very fast in this case
   ↪ )
c.compute()
```

For more details on Dask Bag check https://dask.pydata.org/en/latest/bag.html

10.0.7.3 Concurrency Features Dask supports a real-time task framework that extends Python's concurrent.futures interface. This interface is good for arbitrary task scheduling, like dask.delayed,

but is immediate rather than lazy, which provides some more flexibility in situations where the computations may evolve. These features depend on the second-generation task scheduler found in dask.distributed (which, despite its name, runs very well on a single machine).

Dask allows us to simply construct graphs of tasks with dependencies. We can find that graphs can also be created automatically for us using functional, Numpy, or Pandas syntax on data collections. None of this would be very useful if there weren't also a way to execute these graphs, in a parallel and memory-aware way. Dask comes with four available schedulers:

- `dask.threaded.get` : a scheduler backed by a thread pool
- `dask.multiprocessing.get` : a scheduler backed by a process pool
- `dask.async.get_sync` : a synchronous scheduler, good for debugging
- `distributed.Client.get` : a distributed scheduler for executing graphs on multiple machines.

Here is a simple program for dask.distributed library:

```
from dask.distributed import Client
client = Client('scheduler:port')

futures = []
for fn in filenames:
    future = client.submit(load, fn)
    futures.append(future)

summary = client.submit(summarize, futures)
summary.result()
```

For more details on Concurrent Features by Dask check https://dask.pydata.org/en/latest/futures.html

10.0.7.4 Dask Array Dask arrays implement a subset of the NumPy interface on large arrays using blocked algorithms and task scheduling. These behave like numpy arrays, but break a massive job into tasks that are then executed by a scheduler. The default scheduler uses threading but you can also use multiprocessing or distributed or even serial processing (mainly for debugging). You can tell the dask array how to break the data into chunks for processing.

```
import dask.array as da
f = h5py.File('myfile.hdf5')
x = da.from_array(f['/big-data'], chunks=(1000, 1000))
```

```
x - x.mean(axis=1).compute()
```

For more details on Dask Array check https://dask.pydata.org/en/latest/array.html

10.0.7.5 Dask DataFrame A Dask DataFrame is a large parallel dataframe composed of many smaller Pandas dataframes, split along the index. These pandas dataframes may live on disk for larger-than-memory computing on a single machine, or on many different machines in a cluster. Dask.dataframe implements a commonly used subset of the Pandas interface including elementwise operations, reductions, grouping operations, joins, timeseries algorithms, and more. It copies the Pandas interface for these operations exactly and so should be very familiar to Pandas users. Because Dask.dataframe operations merely coordinate Pandas operations they usually exhibit similar performance characteristics as are found in Pandas. To run the following code, save 'student.csv' file in your machine.

```
import pandas as pd
df = pd.read_csv('student.csv')
d = df.groupby(df.HID).Serial_No.mean()
print(d)

ID
101     1
102     2
104     3
105     4
106     5
107     6
109     7
111     8
201     9
202    10
Name: Serial_No, dtype: int64

import dask.dataframe as dd
df = dd.read_csv('student.csv')
dt = df.groupby(df.HID).Serial_No.mean().compute()
print (dt)

ID
101     1.0
```

```
102     2.0
104     3.0
105     4.0
106     5.0
107     6.0
109     7.0
111     8.0
201     9.0
202    10.0
Name: Serial_No, dtype: float64
```

For more details on Dask DataFrame check https://dask.pydata.org/en/latest/dataframe.html

10.0.7.6 Dask DataFrame Storage Efficient storage can dramatically improve performance, particularly when operating repeatedly from disk.

Decompressing text and parsing CSV files is expensive. One of the most effective strategies with medium data is to use a binary storage format like HDF5.

```
# be sure to shut down other kernels running distributed clients
from dask.distributed import Client
client = Client()
```

Create data if we don't have any

```
from prep import accounts_csvs
accounts_csvs(3, 1000000, 500)
```

First we read our csv data as before.

CSV and other text-based file formats are the most common storage for data from many sources, because they require minimal pre-processing, can be written line-by-line and are human-readable. Since Pandas' read_csv is well-optimized, CSVs are a reasonable input, but far from optimized, since reading required extensive text parsing.

```
import os
filename = os.path.join('data', 'accounts.*.csv')
filename

import dask.dataframe as dd
```

```
df_csv = dd.read_csv(filename)
df_csv.head()
```

HDF5 and netCDF are binary array formats very commonly used in the scientific realm.

Pandas contains a specialized HDF5 format, HDFStore. The dd.DataFrame.to_hdf method works exactly like the pd.DataFrame.to_hdf method.

```
target = os.path.join('data', 'accounts.h5')
target

%time df_csv.to_hdf(target, '/data')

df_hdf = dd.read_hdf(target, '/data')
df_hdf.head()
```

For more information on Dask DataFrame Storage, click http://dask.pydata.org/en/latest/dataframe-create.html

10.0.7.7 Links

- https://dask.pydata.org/en/latest/
- http://matthewrocklin.com/blog/work/2017/10/16/streaming-dataframes-1
- http://people.duke.edu/~ccc14/sta-663-2017/18A_Dask.html
- https://www.kdnuggets.com/2016/09/introducing-dask-parallel-programming.html
- https://pypi.python.org/pypi/dask/
- https://www.hdfgroup.org/2015/03/hdf5-as-a-zero-configuration-ad-hoc-scientific-database-for-python/
- https://github.com/dask/dask-tutorial

11 APPLICATIONS

11.0.1 Fingerprint Matching

> ⚠ Please note that NIST has temporarily removed the Fingerprint data set. We, unfortunately, do not have a copy of the dataset. If you have one, please notify us —

Python is a flexible and popular language for running data analysis pipelines. In this section, we will implement a solution for fingerprint matching.

11.0.1.1 Overview Fingerprint recognition refers to the automated method for verifying a match between two fingerprints and that is used to identify individuals and verify their identity. Fingerprints (Figure 35) are the most widely used form of biometric used to identify individuals.

Figure 35: Fingerprints

The automated fingerprint matching generally required the detection of different fingerprint features (aggregate characteristics of ridges, and minutia points) and then the use of fingerprint matching algorithm, which can do both one-to-one and one-to-many matching operations. Based on the number of matches a proximity score (distance or similarity) can be calculated.

We use the following NIST dataset for the study:

Special Database 14 - NIST Mated Fingerprint Card Pairs 2. (http://www.nist.gov/itl/iad/ig/special_dbases.cfm)

11.0.1.2 Objectives Match the fingerprint images from a probe set to a gallery set and report the match scores.

11.0.1.3 Prerequisites

For this work we will use the following algorithms:

- MINDTCT: The NIST minutiae detector, which automatically locates and records ridge ending and bifurcations in a fingerprint image. (http://www.nist.gov/itl/iad/ig/nbis.cfm)
- BOZORTH3: A NIST fingerprint matching algorithm, which is a minutiae-based fingerprint-matching algorithm. It can do both one-to-one and one-to-many matching operations. (http://www.nist.gov/itl/iad/ig/nbis.cfm)

In order to follow along, you must have the NBIS tools which provide `mindtct` and `bozorth3` installed. If you are on Ubuntu 16.04 Xenial, the following steps will accomplish this:

```
$ sudo apt-get update -qq
$ sudo apt-get install -y build-essential cmake unzip
$ wget "http://nigos.nist.gov:8080/nist/nbis/nbis_v5_0_0.zip"
$ unzip -d nbis nbis_v5_0_0.zip
$ cd nbis/Rel_5.0.0
$ ./setup.sh /usr/local --without-X11
$ sudo make
```

11.0.1.4 Implementation

1. Fetch the fingerprint images from the web
2. Call out to external programs to prepare and compute the match scored
3. Store the results in a database
4. Generate a plot to identify likely matches.

```
import urllib
import zipfile
import hashlib
```

we will be interacting with the operating system and manipulating files and their pathnames.

```
import os.path
import os
import sys
import shutil
import tempfile
```

Some general useful utilities

```
import itertools
import functools
import types
from pprint import pprint
```

Using the `attrs` library provides some nice shortcuts to defining objects

```
import attr
```

```
import sys
```

we will be randomly dividing the entire dataset, based on user input, into the probe and gallery stets

```
import random
```

we will need to call out to the NBIS software. we will also be using multiple processes to take advantage of all the cores on our machine

```
import subprocess
import multiprocessing
```

As for plotting, we will use `matplotlib`, though there are many alternatives.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Finally, we will write the results to a database.

```
import sqlite3
```

11.0.1.5 Utility functions

Next, we will define some utility functions:

```
def take(n, iterable):
    "Returns a generator of the first **n** elements of an iterable"
    return itertools.islice(iterable, n )
```

```
def zipWith(function, *iterables):
    "Zip a set of **iterables** together and apply **function** to
       ↪ each tuple"
    for group in itertools.izip(*iterables):
        yield function(*group)


def uncurry(function):
    "Transforms an N-arry **function** so that it accepts a single
       ↪ parameter of an N-tuple"
    @functools.wraps(function)
    def wrapper(args):
        return function(*args)
    return wrapper


def fetch_url(url, sha256, prefix='.', checksum_blocksize=2**20,
   ↪ dryRun=False):
    """Download a url.

    :param url: the url to the file on the web
    :param sha256: the SHA-256 checksum. Used to determine if the
       ↪ file was previously downloaded.
    :param prefix: directory to save the file
    :param checksum_blocksize: blocksize to used when computing the
       ↪ checksum
    :param dryRun: boolean indicating that calling this function
       ↪ should do nothing
    :returns: the local path to the downloaded file
    :rtype:

    """

    if not os.path.exists(prefix):
        os.makedirs(prefix)

    local = os.path.join(prefix, os.path.basename(url))
```

```
    if dryRun: return local

    if os.path.exists(local):
        print ('Verifying checksum')
        chk = hashlib.sha256()
        with open(local, 'rb') as fd:
            while True:
                bits = fd.read(checksum_blocksize)
                if not bits: break
                chk.update(bits)
        if sha256 == chk.hexdigest():
            return local

    print ('Downloading', url)

    def report(sofar, blocksize, totalsize):
        msg = '{}%\r'.format(100 * sofar * blocksize / totalsize,
           ↪ 100)
        sys.stderr.write(msg)

    urllib.urlretrieve(url, local, report)

    return local
```

11.0.1.6 Dataset we will now define some global parameters

First, the fingerprint dataset

```
DATASET_URL = 'https://s3.amazonaws.com/nist-srd/SD4/
   ↪ NISTSpecialDatabase4GrayScaleImagesofFIGS.zip'
DATASET_SHA256 = '4
   ↪ db6a8f3f9dc14c504180cbf67cdf35167a109280f121c901be37a80ac13c449
   ↪ '
```

We'll define how to download the dataset. This function is general enough that it could be used to retrieve most files, but we will default it to use the values from previous.

```
def prepare_dataset(url=None, sha256=None, prefix='.', skip=False):
```

```
    url = url or DATASET_URL
    sha256 = sha256 or DATASET_SHA256
    local = fetch_url(url, sha256=sha256, prefix=prefix, dryRun=skip)

    if not skip:
        print ('Extracting', local, 'to', prefix)
        with zipfile.ZipFile(local, 'r') as zip:
            zip.extractall(prefix)

    name, _ = os.path.splitext(local)
    return name


def locate_paths(path_md5list, prefix):
    with open(path_md5list) as fd:
        for line in itertools.imap(str.strip, fd):
            parts = line.split()
            if not len(parts) == 2: continue
            md5sum, path = parts
            chksum = Checksum(value=md5sum, kind='md5')
            filepath = os.path.join(prefix, path)
            yield Path(checksum=chksum, filepath=filepath)


def locate_images(paths):

    def predicate(path):
        _, ext = os.path.splitext(path.filepath)
        return ext in ['.png']

    for path in itertools.ifilter(predicate, paths):
        yield image(id=path.checksum.value, path=path)
```

11.0.1.7 Data Model we will define some classes so we have a nice API for working with the dataflow. We set `slots=True` so that the resulting objects will be more space-efficient.

11.0.1.7.1 Utilities Checksum

The checksum consists of the actual hash value (`value`) as well as a string representing the hashing algorithm. The validator enforces that the algorithm can only be one of the listed acceptable methods

```
@attr.s(slots=True)
class Checksum(object):
  value = attr.ib()
  kind = attr.ib(validator=lambda o, a, v: v in 'md5 sha1 sha224
     ↪ sha256 sha384 sha512'.split())
```

Path

`Path refers to an image's file path and associated` Checksum‘. We get the checksum “for”free” since the MD5 hash is provided for each image in the dataset.

```
@attr.s(slots=True)
class Path(object):
    checksum = attr.ib()
    filepath = attr.ib()
```

Image

The start of the data pipeline is the image. An `image` has an `id` (the md5 hash) and the path to the image.

```
@attr.s(slots=True)
class image(object):
    id = attr.ib()
    path = attr.ib()
```

11.0.1.7.2 Mindtct The next step in the pipeline is to apply the `mindtct` program from NBIS. A `mindtct` object, therefore, represents the results of applying `mindtct` on an `image` . The `xyt` output is needed for the next step, and the `image` attribute represents the image id.

```
@attr.s(slots=True)
class mindtct(object):
    image = attr.ib()
    xyt = attr.ib()

    def pretty(self):
        d = dict(id=self.image.id, path=self.image.path)
```

```
        return pprint(d)
```

We need a way to construct a `mindtct` object from an `image` object. A straightforward way of doing this would be to have a `from_image` `@staticmethod` or `@classmethod`, but that doesn't work well with `multiprocessing` as top-level functions work best as they need to be serialized.

```
def mindtct_from_image(image):
    imgpath = os.path.abspath(image.path.filepath)
    tempdir = tempfile.mkdtemp()
    oroot = os.path.join(tempdir, 'result')

    cmd = ['mindtct', imgpath, oroot]

    try:
        subprocess.check_call(cmd)

        with open(oroot + '.xyt') as fd:
            xyt = fd.read()

        result = mindtct(image=image.id, xyt=xyt)
        return result

    finally:
        shutil.rmtree(tempdir)
```

11.0.1.7.3 Bozorth3 The final step in the pipeline is running the `bozorth3` from NBIS. The `bozorth3` class represents the match being done: tracking the ids of the probe and gallery images as well as the match score.

Since we will be writing these instances out to a database, we provide some static methods for SQL statements. While there are many Object-Relational-Model (ORM) libraries available for Python, this approach keeps the current implementation simple.

```
@attr.s(slots=True)
class bozorth3(object):
    probe = attr.ib()
    gallery = attr.ib()
    score = attr.ib()
```

```
    @staticmethod
    def sql_stmt_create_table():
        return 'CREATE TABLE IF NOT EXISTS bozorth3' \
             + '(probe TEXT, gallery TEXT, score NUMERIC)'

    @staticmethod
    def sql_prepared_stmt_insert():
        return 'INSERT INTO bozorth3 VALUES (?, ?, ?)'

    def sql_prepared_stmt_insert_values(self):
        return self.probe, self.gallery, self.score
```

In order to work well with `multiprocessing`, we define a class representing the input parameters to `bozorth3` and a helper function to run `bozorth3`. This way the pipeline definition can be kept simple to a `map` to create the input and then a `map` to run the program.

As NBIS `bozorth3` can be called to compare one-to-one or one-to-many, we will also dynamically choose between these approaches depending on if the gallery attribute is a list or a single object.

```
@attr.s(slots=True)
class bozorth3_input(object):
    probe = attr.ib()
    gallery = attr.ib()

    def run(self):
        if isinstance(self.gallery, mindtct):
            return bozorth3_from_one_to_one(self.probe, self.gallery)
        elif isinstance(self.gallery, types.ListType):
            return bozorth3_from_one_to_many(self.probe, self.gallery
               ↪ )
        else:
            raise ValueError('Unhandled type for gallery: {}'.format(
               ↪ type(gallery)))
```

The next is the top-level function to running `bozorth3`. It accepts an instance of `bozorth3_input`. The is implemented as a simple top-level wrapper so that it can be easily passed to the `multiprocessing` library.

```
def run_bozorth3(input):
    return input.run()
```

Running Bozorth3

There are two cases to handle: 1. One-to-one probe to gallery sets 1. One-to-many probe to gallery sets

Both approaches are implemented next. The implementations follow the same pattern: 1. Create a temporary directory within with to work 1. Write the probe and gallery images to files in the temporary directory 1. Call the `bozorth3` executable 1. The match score is written to `stdout` which is captured and then parsed. 1. Return a `bozorth3` instance for each match 1. Make sure to clean up the temporary directory

One-to-one

```
def bozorth3_from_one_to_one(probe, gallery):
    tempdir = tempfile.mkdtemp()
    probeFile = os.path.join(tempdir, 'probe.xyt')
    galleryFile = os.path.join(tempdir, 'gallery.xyt')

    with open(probeFile,   'wb') as fd: fd.write(probe.xyt)
    with open(galleryFile, 'wb') as fd: fd.write(gallery.xyt)

    cmd = ['bozorth3', probeFile, galleryFile]

    try:
        result = subprocess.check_output(cmd)
        score = int(result.strip())
        return bozorth3(probe=probe.image, gallery=gallery.image,
            ↪ score=score)
    finally:
        shutil.rmtree(tempdir)
```

One-to-many

```
def bozorth3_from_one_to_many(probe, galleryset):
    tempdir = tempfile.mkdtemp()
    probeFile = os.path.join(tempdir, 'probe.xyt')
    galleryFiles = [os.path.join(tempdir, 'gallery%d.xyt' % i)
                    for i,_ in enumerate(galleryset)]
```

```
    with open(probeFile, 'wb') as fd: fd.write(probe.xyt)
    for galleryFile, gallery in itertools.izip(galleryFiles,
       ↪ galleryset):
        with open(galleryFile, 'wb') as fd: fd.write(gallery.xyt)

    cmd = ['bozorth3', '-p', probeFile] + galleryFiles

    try:
        result = subprocess.check_output(cmd).strip()
        scores = map(int, result.split('\n'))
        return [bozorth3(probe=probe.image, gallery=gallery.image,
           ↪ score=score)
                for score, gallery in zip(scores, galleryset)]
    finally:
        shutil.rmtree(tempdir)
```

11.0.1.8 Plotting For plotting, we will operate only on the database. we will select a small number of probe images and plot the score between them and the rest of the gallery images.

The `mk_short_labels` helper function will be defined next.

```
def plot(dbfile, nprobes=10):
    conn = sqlite3.connect(dbfile)
    results = pd.read_sql(
        "SELECT DISTINCT probe FROM bozorth3 ORDER BY score LIMIT '%s
           ↪ '" % nprobes,
        con=conn
    )
    shortlabels = mk_short_labels(results.probe)
    plt.figure()

    for i, probe in results.probe.iteritems():
        stmt = 'SELECT gallery, score FROM bozorth3 WHERE probe = ?
           ↪ ORDER BY gallery DESC'
        matches = pd.read_sql(stmt, params=(probe,), con=conn)
        xs = np.arange(len(matches), dtype=np.int)
```

```
        plt.plot(xs, matches.score, label='probe %s' % shortlabels[i
           ↪ ])

    plt.ylabel('Score')
    plt.xlabel('Gallery')
    plt.legend(bbox_to_anchor=(0, 0, 1, -0.2))
    plt.show()
```

The image ids are long hash strings. In order to minimize the amount of space on the figure the labels occupy, we provide a helper function to create a short label that still uniquely identifies each probe image in the selected sample

```
def mk_short_labels(series, start=7):
    for size in xrange(start, len(series[0])):
        if len(series) == len(set(map(lambda s: s[:size], series))):
            break
    return map(lambda s: s[:size], series)
```

11.0.1.9 Putting it all Together First, set up a temporary directory in which to work:

```
pool = multiprocessing.Pool()
prefix = '/tmp/fingerprint_example/'
if not os.path.exists(prefix):
    os.makedirs(prefix)
```

Next, we download and extract the fingerprint images from NIST:

```
%%time
dataprefix = prepare_dataset(prefix=prefix)
```

```
Verifying checksum Extracting
/tmp/fingerprint_example/NISTSpecialDatabase4GrayScaleImagesofFIGS.
   ↪ zip
to /tmp/fingerprint_example/ CPU times: user 3.34 s, sys: 645 ms,
total: 3.99 s Wall time: 4.01 s
```

Next, we will configure the location of the MD5 checksum file that comes with the download

```
md5listpath = os.path.join(prefix, '
   ↪ NISTSpecialDatabase4GrayScaleImagesofFIGS/sd04/sd04_md5.lst')
```

Load the images from the downloaded files to start the analysis pipeline

```
%%time
print('Loading images')
paths = locate_paths(md5listpath, dataprefix)
images = locate_images(paths)
mindtcts = pool.map(mindtct_from_image, images)
print('Done')
```

```
Loading images Done CPU times: user 187 ms, sys: 17 ms, total: 204 ms
Wall time: 1min 21s
```

We can examine one of the loaded images. Note that `image` refers to the MD5 checksum that came with the image and the `xyt` attribute represents the raw image data.

```
print(mindtcts[0].image)
print(mindtcts[0].xyt[:50])
```

```
98b15d56330cb17f1982ae79348f711d 14 146 214 6 25 238 22 37 25 51 180
   ↪ 20
30 332 214
```

For example purposes we will only a use a small percentage of the database, randomly selected, for pur probe and gallery datasets.

```
perc_probe = 0.001
perc_gallery = 0.1
```

```
%%time
print('Generating samples')
probes  = random.sample(mindtcts, int(perc_probe   * len(mindtcts)))
gallery = random.sample(mindtcts, int(perc_gallery * len(mindtcts)))
print('|Probes| =', len(probes))
print('|Gallery|=', len(gallery))
```

```
Generating samples = 4 = 400 CPU times: user 2 ms, sys: 0 ns, total:
   ↪ 2
ms Wall time: 993 micros
```

We can now compute the matching scores between the probe and gallery sets. This will use all cores available on this workstation.

```
%%time
print('Matching')
input = [bozorth3_input(probe=probe, gallery=gallery)
         for probe in probes]
bozorth3s = pool.map(run_bozorth3, input)
```

```
Matching CPU times: user 19 ms, sys: 1 ms, total: 20 ms Wall time:
   ↪ 1.07
s
```

`bozorth3s` is now a `list` of `lists` of `bozorth3` instances.

```
print('|Probes|  =', len(bozorth3s))
print('|Gallery| =', len(bozorth3s[0]))
print('Result:', bozorth3s[0][0])
```

```
= 4 = 400 Result: bozorth3(probe='caf9143b268701416fbed6a9eb2eb4cf',
gallery='22fa0f24998eaea39dea152e4a73f267', score=4)
```

Now add the results to the database

```
dbfile = os.path.join(prefix, 'scores.db')
conn = sqlite3.connect(dbfile)
cursor = conn.cursor()
cursor.execute(bozorth3.sql_stmt_create_table())
```

```
<sqlite3.Cursor at 0x7f8a2f677490>
```

```
%%time
for group in bozorth3s:
```

```
    vals = map(bozorth3.sql_prepared_stmt_insert_values, group)
    cursor.executemany(bozorth3.sql_prepared_stmt_insert(), vals)
    conn.commit()
    print('Inserted results for probe', group[0].probe)
```

```
Inserted results for probe caf9143b268701416fbed6a9eb2eb4cf Inserted
results for probe 55ac57f711eba081b9302eab74dea88e Inserted results
   ↪ for
probe 4ed2d53db3b5ab7d6b216ea0314beb4f Inserted results for probe
20f68849ee2dad02b8fb33ecd3ece507 CPU times: user 2 ms, sys: 3 ms,
   ↪ total:
5 ms Wall time: 3.57 ms
```

We now plot the results. Figure 36

```
plot(dbfile, nprobes=len(probes))
```

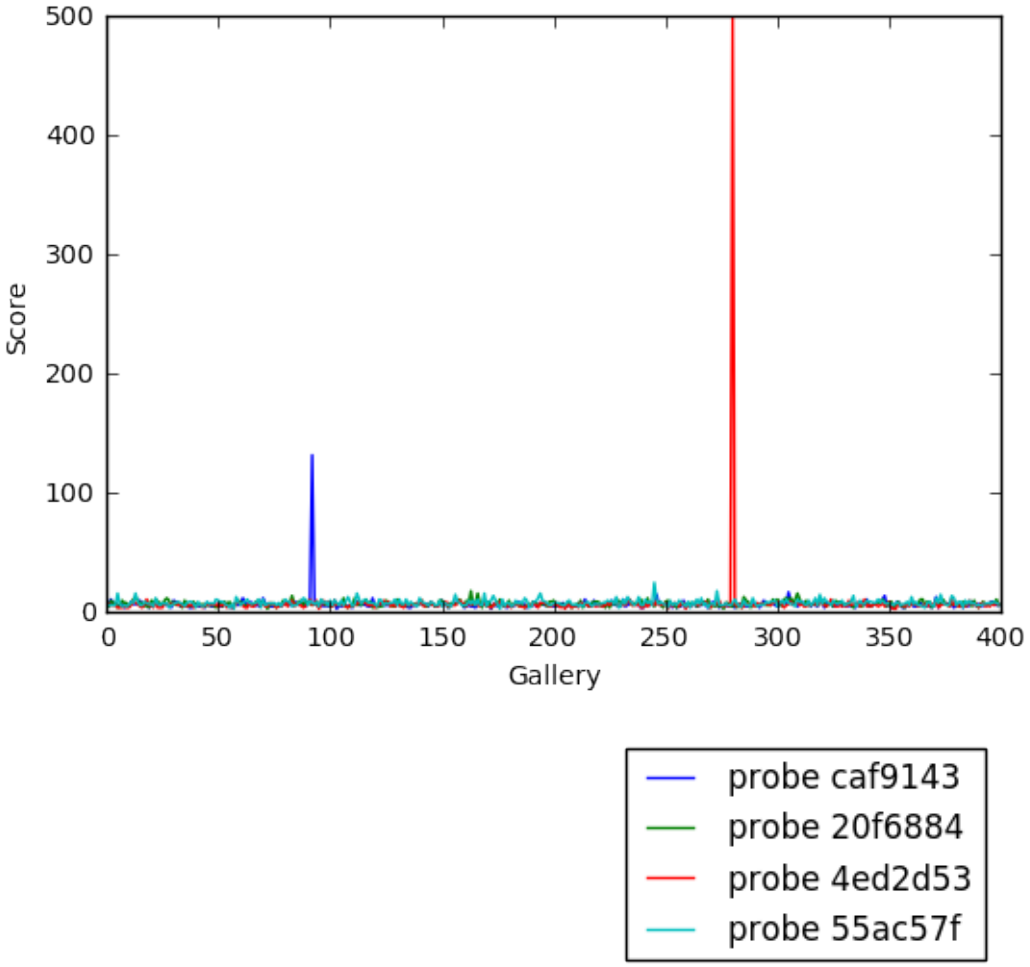


Figure 36: Result

```
cursor.close()
```

11.0.2 NIST Pedestrian and Face Detection 🐞[](https://github.com/cloudmesh-community/book/blob/master/chapters/prg/python/facedetection/facedetection.md)

Figure 37: No

Pedestrian and Face Detection uses OpenCV to identify people standing in a picture or a video and NIST use case in this document is built with Apache Spark and Mesos clusters on multiple compute nodes.

The example in this tutorial deploys software packages on OpenStack using Ansible with its roles. See Figure 38, Figure 39, Figure 40, Figure 41

Figure 38: Original

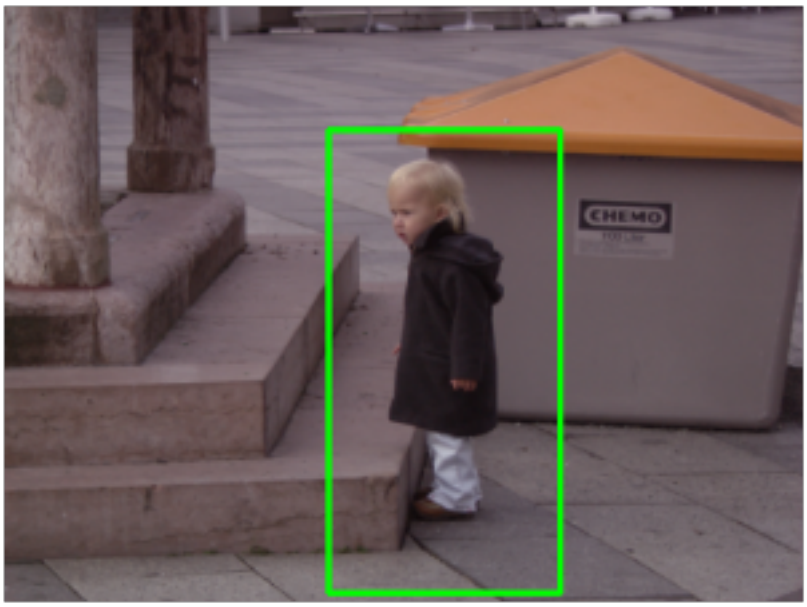


Figure 39: Pedestrian Detected

Figure 40: Original

Figure 41: Pedestrian and Face/eyes Detected

11.0.2.0.1 Introduction Human (pedestrian) detection and face detection have been studied during the last several years and models for them have improved along with Histograms of Oriented Gradients (HOG) for Human Detection [1]. OpenCV is a Computer Vision library including the SVM classifier and the HOG object detector for pedestrian detection and INRIA Person Dataset [2] is one of the popular samples for both training and testing purposes. In this document, we deploy Apache Spark on Mesos clusters to train and apply detection models from OpenCV using Python API.

INRIA Person Dataset

This dataset contains positive and negative images for training and test purposes with annotation files for upright persons in each image. 288 positive test images, 453 negative test images, 614 positive training images, and 1218 negative training images are included along with normalized 64x128 pixel formats. 970MB dataset is available to download [3].

HOG with SVM model

Histogram of Oriented Gradient (HOG) and Support Vector Machine (SVM) are used as object detectors and classifiers and built-in python libraries from OpenCV provide these models for human detection.

Ansible Automation Tool

Ansible is a python tool to install/configure/manage software on multiple machines with JSON files where system descriptions are defined. There are reasons why we use Ansible:

- Expandable: Leverages Python (default) but modules can be written in any language
- Agentless: no setup required on a managed node
- Security: Allows deployment from userspace; uses ssh for authentication
- Flexibility: only requires ssh access to privileged user
- Transparency: YAML Based script files express the steps of installing and configuring software
- Modularity: Single Ansible Role (should) contain all required commands and variables to deploy software package independently
- Sharing and portability: roles are available from the source (GitHub, bitbucket, GitLab, etc) or the Ansible Galaxy portal

We use Ansible roles to install software packages for Human and Face Detection which requires running OpenCV Python libraries on Apache Mesos with a cluster configuration. Dataset is also downloaded from the web using an ansible role.

11.0.2.0.2 Deployment by Ansible Ansible is to deploy applications and build clusters for batch-processing large datasets towards target machines e.g. VM instances on OpenStack and we use Ansible roles with *include* directive to organize layers of big data software stacks (BDSS). Ansible provides abstractions by Playbook Roles and reusability by Include statements. We define X application in X Ansible Role, for example, and use include statements to combine with other applications e.g. Y or Z. The layers exist in subdirectories (see next) to add modularity to your Ansible deployment. For example, there are five roles used in this example that are Apache Mesos in a scheduler layer, Apache Spark in a processing layer, an OpenCV library in an application layer, INRIA Person Dataset in a dataset layer, and a python script for human and face detection in an analytics layer. If you have an additional software package to add, you can simply add a new role in the main Ansible playbook with *include* directive. With this, your Ansible playbook maintains simple but flexible to add more roles without having a large single file which is getting difficult to read when it deploys more applications on multiple layers. The main Ansible playbook runs Ansible roles in order which look like:

```
```
include: sched/00-mesos.yml
include: proc/01-spark.yml
include: apps/02-opencv.yml
include: data/03-inria-dataset.yml
Include: anlys/04-human-face-detection.yml
```
```

Directory names e.g. sched, proc, data, or anlys indicate BDSS layers like: - sched: scheduler layer - proc: data processing layer - apps: application layer - data: dataset layer - anlys: analytics layer and two digits in the filename indicate an order of roles to be run.

11.0.2.0.3 Cloudmesh for Provisioning It is assumed that virtual machines are created by cloudmesh, the cloud management software. For example on OpenStack,

```
cm cluster create -N=6
```

command starts a set of virtual machine instances. The number of machines and groups for clusters e.g. namenodes and datanodes are defined in the Ansible inventory file, a list of target machines with groups, which will be generated once machines are ready to use by cloudmesh. Ansible roles install software and dataset on virtual clusters after that stage.

11.0.2.0.4 Roles Explained for Installation Mesos role is installed first as a scheduler layer for masters and slaves where mesos-master runs on the masters group and mesos-slave runs on the slaves group. Apache Zookeeper is included in the mesos role therefore mesos slaves find an elected mesos leader for the coordination. Spark, as a data processing layer, provides two options for distributed job processing, batch job processing via a cluster mode and real-time processing via a client mode. The Mesos dispatcher runs on a masters group to accept a batch job submission and Spark interactive shell, which is the client mode, provides real-time processing on any node in the cluster. Either way, Spark is installed later to detect a master (leader) host for a job submission. Other roles for OpenCV, INRIA Person Dataset and Human and Face Detection Python applications are followed by.

The following software is expected in the stacks according to the github:

- mesos cluster (master, worker)
- spark (with dispatcher for mesos cluster mode)
- openCV
- zookeeper
- INRIA Person Dataset
- Detection Analytics in Python
- [1] Dalal, Navneet, and Bill Triggs. “Histograms of oriented gradients for human detection.” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05). Vol. 1. IEEE,

  2005. [pdf]
- [2] http://pascal.inrialpes.fr/data/human/

- [3] ftp://ftp.inrialpes.fr/pub/lear/douze/data/INRIAPerson.tar
- [4] https://docs.python.org/2/library/configparser.html

Server groups for Masters/Slaves by Ansible inventory

We may separate compute nodes in two groups: masters and workers therefore Mesos masters and zookeeper quorums manage job requests and leaders and workers run actual tasks. Ansible needs group definitions in their inventory therefore software installation associated with a proper part can be completed.

Example of Ansible Inventory file (inventory.txt)

```
[masters]
10.0.5.67
10.0.5.68
10.0.5.69
[slaves]
10.0.5.70
10.0.5.71
10.0.5.72
```

11.0.2.0.5 Instructions for Deployment The following commands complete NIST Pedestrian and Face Detection deployment on OpenStack.

Cloning Pedestrian Detection Repository from Github

Roles are included as submodules that require `--recursive` option to checkout them all.

```
$ git clone --recursive https://github.com/futuresystems/pedestrian-
   ↪ and-face-detection.git
```

Change the following variable with actual ip addresses:

```
sample_inventory="""[masters]
10.0.5.67
10.0.5.68
10.0.5.69
[slaves]
10.0.5.70
10.0.5.71
10.0.5.72"""
```

Create an `inventory.txt` file with the variable in your local directory.

```
!printf "$sample_inventory" > inventory.txt
!cat inventory.txt
```

Add `ansible.cfg` file with options for ssh host key checking and login name.

```
ansible_config="""[defaults]
host_key_checking=false
remote_user=ubuntu"""
!printf "$ansible_config" > ansible.cfg
!cat ansible.cfg
```

Check accessibility by ansible ping like:

```
!ansible -m ping -i inventory.txt all
```

Make sure that you have a correct ssh key in your account otherwise you may encounter 'FAILURE' in the previous ping test.

Ansible Playbook

We use a main Ansible playbook to deploy software packages for NIST Pedestrian and Face detection which includes: - mesos - spark -zookeeper - opencv - INRIA Person dataset - Python script for the detection

```
!cd pedestrian-and-face-detection/ && ansible-playbook -i ../
   ↪ inventory.txt site.yml
```

The installation may take 30 minutes or an hour to complete.

11.0.2.0.6 OpenCV in Python Before we run our code for this project, let's try OpenCV first to see how it works.

Import cv2

Let us import opencv python module and we will use images from the online database image-net.org to test OpenCV image recognition. See Figure 42, Figure 43

```
import cv2
```

Let us download a mailbox image with a red color to see if opencv identifies the shape with a color. The example file in this tutorial is:

```
$ curl http://farm4.static.flickr.com/3061/2739199963_ee78af76ef.jpg
   ↪ > mailbox.jpg
```

```
100 167k 100 167k 0 0 686k 0 --:--:-- --:--:-- --:--:-- 684k
```

```
%matplotlib inline

from IPython.display import Image
mailbox_image = "mailbox.jpg"
Image(filename=mailbox_image)
```

Figure 42: Mailbox image

You can try other images. Check out the image-net.org for mailbox images: http://image-net.org/synset?wnid=n03710193

Image Detection

Just for a test, let's try to detect a red color shaped mailbox using opencv python functions.

There are key functions that we use: * cvtColor: to convert a color space of an image * inRange: to detect a mailbox based on the range of red color pixel values * np.array: to define the range of red color using a Numpy library for better calculation * findContours: to find a outline of the object * bitwise_and: to black-out the area of contours found

```
    import numpy as np
    import matplotlib.pyplot as plt

    # imread for loading an image
    img = cv2.imread(mailbox_image)
    # cvtColor for color conversion
    hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    # define range of red color in hsv
    lower_red1 = np.array([0, 50, 50])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([170, 50, 50])
    upper_red2 = np.array([180, 255, 255])

    # threshold the hsv image to get only red colors
    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
    mask = mask1 + mask2

    # find a red color mailbox from the image
    im2, contours,hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
       ↪ cv2.CHAIN_APPROX_SIMPLE)

    # bitwise_and to remove other areas in the image except the
       ↪ detected object
    res = cv2.bitwise_and(img, img, mask = mask)

    # turn off - x, y axis bar
    plt.axis("off")
    # text for the masked image
    cv2.putText(res, "masked image", (20,300), cv2.
       ↪ FONT_HERSHEY_SIMPLEX, 2, (255,255,255))
```

```
    # display
    plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
    plt.show()
```

Figure 43: Masked image

The red color mailbox is left alone in the image which we wanted to find in this example by OpenCV functions. You can try other images with different colors to detect the different shape of objects using findContours and inRange from opencv.

For more information, see the next useful links.

- contours features: http://docs.opencv.org/3.1.0/dd/d49/tutorial/_py/_contour/_features.html
- contours: http://docs.opencv.org/3.1.0/d4/d73/tutorial/_py/_contours/_begin.html
- red color in hsv: http://stackoverflow.com/questions/30331944/finding-red-color-using-python-opencv
- inrange: http://docs.opencv.org/master/da/d97/tutorial/_threshold/_inRange.html
- inrange: http://docs.opencv.org/3.0-beta/doc/py/_tutorials/py/_imgproc/py/_colorspaces/py/_colorspaces.html

- numpy: http://docs.opencv.org/3.0-beta/doc/py/_tutorials/py/_core/py/_basic/_ops/py/_basic/_ops.html

11.0.2.0.7 Human and Face Detection in OpenCV INRIA Person Dataset

We use INRIA Person dataset to detect upright people and faces in images in this example. Let us download it first.

```
$ curl ftp://ftp.inrialpes.fr/pub/lear/douze/data/INRIAPerson.tar >
   ↪ INRIAPerson.tar
```

```
100 969M 100 969M 0 0 8480k 0 0:01:57 0:01:57 –:–:– 12.4M
```

```
$ tar xvf INRIAPerson.tar > logfile && tail logfile
```

Face Detection using Haar Cascades

This section is prepared based on the opencv-python tutorial: http://docs.opencv.org/3.1.0/d7/d8b/tutorial/_py/_face/_detection.html#gsc.tab=0

There is a pre-trained classifier for face detection, download it from here:

```
$ curl https://raw.githubusercontent.com/opencv/opencv/master/data/
   ↪ haarcascades/haarcascade_frontalface_default.xml >
   ↪ haarcascade_frontalface_default.xml
```

```
100 908k 100 908k 0 0 2225k 0 –:–:– –:–:– –:–:– 2259k
```

This classifier XML file will be used to detect faces in images. If you like to create a new classifier, find out more information about training from here: http://docs.opencv.org/3.1.0/dc/d88/tutorial/_traincascade.html

Face Detection Python Code Snippet

Now, we detect faces from the first five images using the classifier. See Figure 44, Figure 45, Figure 46, Figure 47, Figure 48, Figure 49, Figure 50, Figure 51, Figure 52, Figure 53, Figure 54

```
# import the necessary packages
import numpy as np
import cv2
from os import listdir
```

```
from os.path import isfile, join
import matplotlib.pyplot as plt

mypath = "INRIAPerson/Test/pos/"
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default
    ↪ .xml')

onlyfiles = [join(mypath, f) for f in listdir(mypath) if isfile(join(
    ↪ mypath, f))]

cnt = 0
for filename in onlyfiles:
    image = cv2.imread(filename)
    image_grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(image_grayscale, 1.3, 5)
    if len(faces) == 0:
        continue

    cnt_faces = 1
    for (x,y,w,h) in faces:
        cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.putText(image, "face" + str(cnt_faces), (x,y-10), cv2.
            ↪ FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2)
        plt.figure()
        plt.axis("off")
        plt.imshow(cv2.cvtColor(image[y:y+h, x:x+w], cv2.
            ↪ COLOR_BGR2RGB))
        cnt_faces += 1
    plt.figure()
    plt.axis("off")
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    cnt = cnt + 1
    if cnt == 5:
        break
```

Figure 44: Example

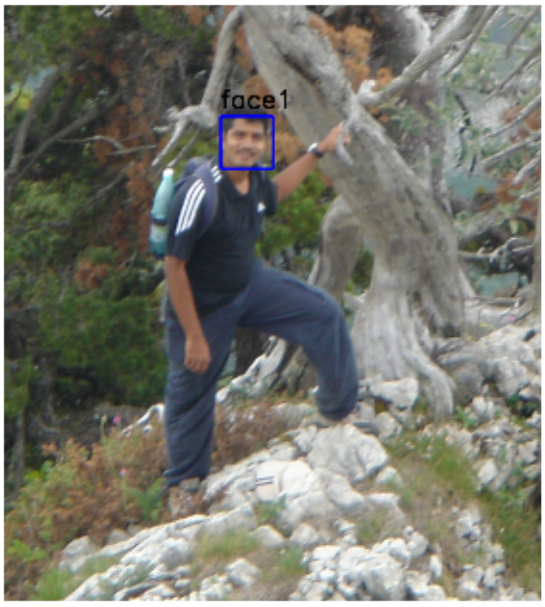


Figure 45: Example

Figure 46: Example

Figure 47: Example

Figure 48: Example

Figure 49: Example

Figure 50: Example

Figure 51: Example

Figure 52: Example

Figure 53: Example

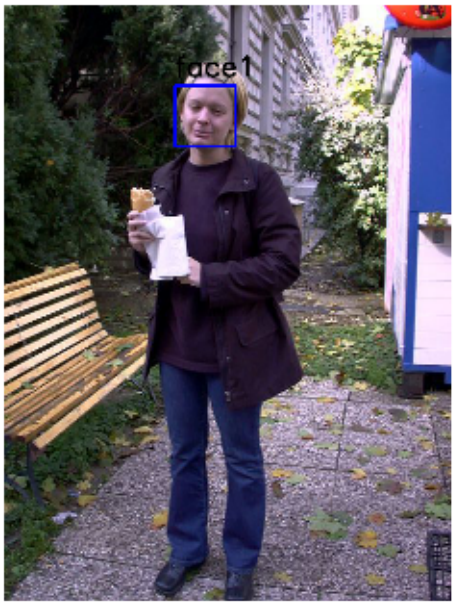


Figure 54: Example

11.0.2.0.8 Pedestrian Detection using HOG Descriptor We will use Histogram of Oriented Gradients (HOG) to detect a upright person from images. See Figure 55, Figure 56, Figure 57, Figure 58, Figure 59, Figure 60, Figure 61, Figure 62, Figure 63, Figure 64

Python Code Snippet

```
# initialize the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

cnt = 0
for filename in onlyfiles:
    img = cv2.imread(filename)
    orig = img.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # detect people in the image
    (rects, weights) = hog.detectMultiScale(img, winStride=(8, 8),
    padding=(16, 16), scale=1.05)
```

```
    # draw the final bounding boxes
    for (x, y, w, h) in rects:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

    plt.figure()
    plt.axis("off")
    plt.imshow(cv2.cvtColor(orig, cv2.COLOR_BGR2RGB))
    plt.figure()
    plt.axis("off")
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    cnt = cnt + 1
    if cnt == 5:
        break
```

Figure 55: Example

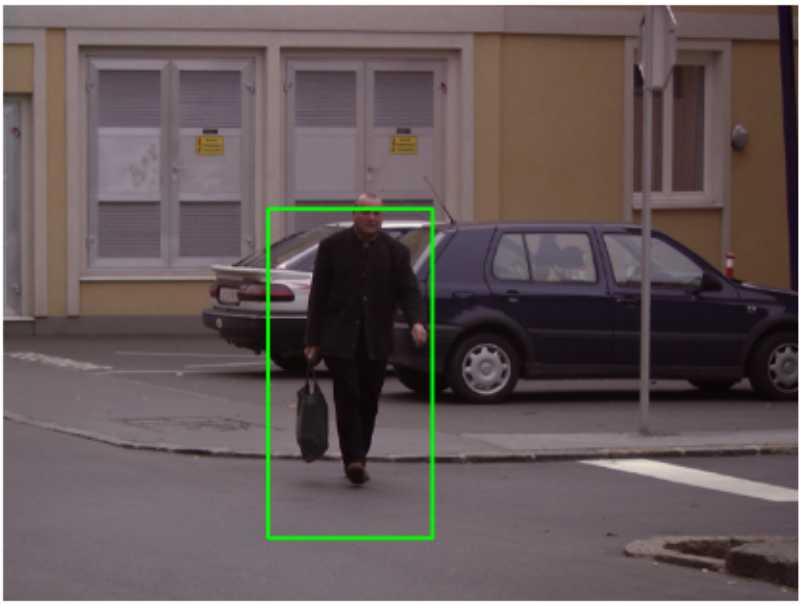


Figure 56: Example

Figure 57: Example

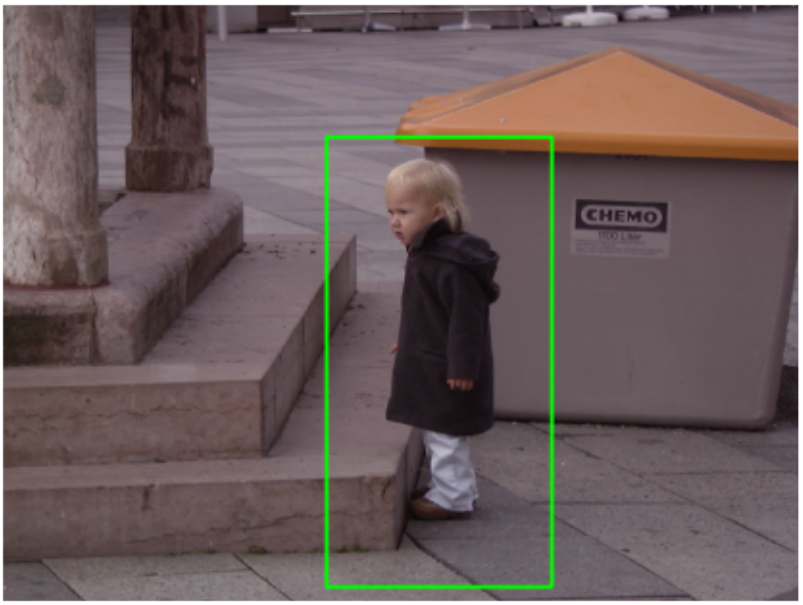


Figure 58: Example

Figure 59: Example

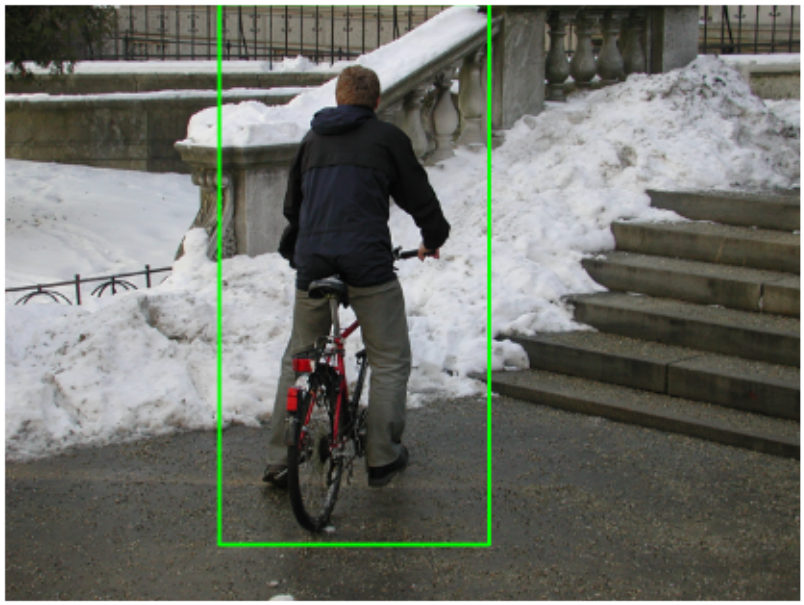


Figure 60: Example

Figure 61: Example

Figure 62: Example

Figure 63: Example

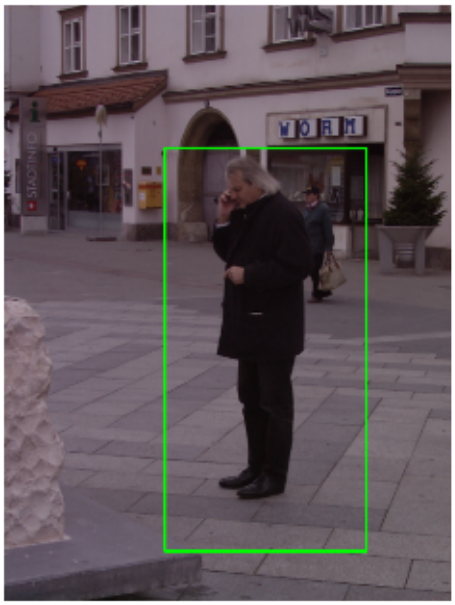


Figure 64: Example

11.0.2.0.9 Processing by Apache Spark INRIA Person dataset provides 100+ images and Spark can be used for image processing in parallel. We load 288 images from “Test/pos” directory.

Spark provides a special object ‘sc’ to connect between a spark cluster and functions in python code. Therefore, we can run python functions in parallel to detect objects in this example.

- *map* function is used to process pedestrian and face detection per image from the parallelize() function of ‘sc’ spark context.
- *collect* function merges results in an array.

  def apply_batch(imagePath): import cv2 import numpy as np # initialize the HOG descriptor/person detector hog = cv2.HOGDescriptor() hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector()) image = cv2.imread(imagePath) # detect people in the image (rects, weights) = hog.detectMultiScale(image, winStride=(8, 8), padding=(16, 16), scale=1.05) # draw the final bounding boxes for (x, y, w, h) in rects: cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2) return image

Parallelize in Spark Context

The list of image files is given to parallelize.

```
pd = sc.parallelize(onlyfiles)
```

Map Function (apply_batch)

The 'apply_batch' function that we created previously is given to map function to process in a spark cluster.

```
pdc = pd.map(apply_batch)
```

Collect Function

The result of each map process is merged into an array.

```
result = pdc.collect()
```

11.0.2.0.10 Results for 100+ images by Spark Cluster

```
for image in result:
    plt.figure()
    plt.axis("off")
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

12 REFERENCES

[1] L. Richardson, "Beautiful soup python package overview." Web Page, 2019 [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[2] C. WODEHOUSE, "Should you use MongoDB? A look at the leading NoSQL database." Web Page, 2018 [Online]. Available: https://www.upwork.com/hiring/data/should-you-use-mongodb-a-look-at-the-leading-nosql-database/

[3] Guru99, "Introduction to MongoDB." Web Page, 2018 [Online]. Available: https://www.guru99.com/mongodb-tutorials.html#1

[4] MongoDB, "Https://www.mongodb.com/." Web Page, 2018 [Online]. Available: https://docs.mongodb.com/manual/introduction/

[5] M. Papiernik, "How to install MongoDB on ubuntu 18.04." Web Page, Jun-2018 [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-18-04

[6] J. Ellingwood, “Initial server setup with ubuntu 18.04.” Web Page, Apr-2018 [Online]. Available: https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04

[7] MongoDB, *Databases and collections*, 4.0 ed. New York, New York, USA: MongoDB Inc, 2008 [Online]. Available: https://docs.mongodb.com/manual/core/databases-and-collections/

[8] J. M. Craig Buckler, “Using JOINs in MongoDB NoSQL databases.” Web Page, Sep-2016 [Online]. Available: https://www.sitepoint.com/using-joins-in-mongodb-nosql-databases/

[9] MongoDB, *Lookup (aggregation)*, 3.2 ed. New York City, New York, United States: MongoDB Inc, 2008 [Online]. Available: https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/

[10] MongoDB, *MongoDB package components - mongoexport*, 4.0 ed. New York City, New York, United States: MongoDB Inc, 2008 [Online]. Available: https://docs.mongodb.com/manual/reference/program/mongoexport/

[11] MongoDB, *Security*, 4.0 ed. New York City, New York, United States: MongoDB Inc, 2008 [Online]. Available: https://docs.mongodb.com/manual/security/

[12] MongoDB, “MongoDB atlas.” Web Page, 2018 [Online]. Available: https://www.mongodb.com/cloud/atlas

[13] I. MongoDB, “PyMongo 3.7.1 documentation.” Web Page, 2008 [Online]. Available: https://api.mongodb.com/python/current/api

[14] A. J. J. Davis, “Announcing PyMongo3.” Web Page, Apr-2015 [Online]. Available: https://emptysqua.re/blog/announcing-pymongo-3/

[15] M. Dirolf, “PyMongo.” Web Page, Jul-2018 [Online]. Available: https://github.com/mongodb/mongo-python-driver

[16] N. Leite, “MongoDB and python.” Web Page, Mar-2015 [Online]. Available: https://www.slideshare.net/NorbertoLeite/mongodb-and-python

[17] V. Oleynik, “How do you use MongoDB with python?” Web Page, Mar-2017 [Online]. Available: https://gearheart.io/blog/how-do-you-use-mongodb-with-python/

[18] I. MongoDB, “Installing / upgrading.” Web pages, 2008 [Online]. Available: http://api.mongodb.com/python/current/installation.html

[19] R. Python, “Introduction to MongoDB and python.” Web Page, 2016 [Online]. Available: https://realpython.com/introduction-to-mongodb-and-python/

[20] W3Schools, “Python MongoDB create database.” Web Page, 1999 [Online]. Available: https://www.w3schools.com/python/python_mongodb_create_db.asp

[21] Inc. MongoDB, “PyMongo 3.7.1 documentation.” Web Page, 2008 [Online]. Available: https://api.mongodb.com/python/current/tutorial.html

[22] N. O’Higgins, *PyMongo & python*. O’Reilly, 2011 [Online]. Available: http://img105.job1001.com/upload/adminnew/2015-04-07/1428393873-MHKX3LN.pdf

[23] Inc. MongoDB, “PyMongo 3.7.1 documentation.” Web Page, 2008 [Online]. Available: https://api.mongodb.com/python/current/examples/aggregation.html

[24] MongoDB, “PyMongo 3.7.2 documenation.” Web Page, 2008 [Online]. Available: https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

[25] MongoDB, “PyMongo 3.7.2 documenation.” Web Page, 2008 [Online]. Available: https://docs.mongodb.com/manual/core/map-reduce/

[26] MongoDB, “PyMongo v2.0 documentation.” Web Page, 2008 [Online]. Available: https://api.mongodb.com/python/2.0/examples/map_reduce.html

[27] MongoDB, “PyMongo 3.7.2 documenation.” Web Page, 2008 [Online]. Available: https://api.mongodb.com/python/current/examples/copydb.html

[28] MongoEngine, “MongoEngine user documentation.” Web Page, 2009 [Online]. Available: http://docs.mongoengine.org/

[29] Wikipedia, “Object-relational mapping.” Web Page, May-2009 [Online]. Available: https://en.wikipedia.org/wiki/Object-relational_mapping

[30] MongoDB, “Flask-MongoEngine.” Web Page, 2008 [Online]. Available: http://docs.mongoengine.org/guide/defining-documents.html

[31] MongoEngine, “User guide: Document instances.” Web Page, 2009 [Online]. Available: http://docs.mongoengine.org/guide/document-instances.html

[32] MongoEngine, “2.1 installing MongoEngine.” Web Page, 2009 [Online]. Available: http://docs.mongoengine.org/guide/installing.html

[33] MongoEngine, “2.2 connection to MongoDB.” Web Page, 2009 [Online]. Available: http://docs.mongoengine.org/guide/connecting.html

[34] MongoEngine, “User guide 2.5. Querying the database.” Web Page, 2009 [Online]. Available: http://docs.mongoengine.org/guide/querying.html

[35] Wikipedia, “Flask (web framework).” Web Page, 2010 [Online]. Available: https://en.wikipedia.org/wiki/Flask_(web_framework)

[36] MongoDB, “Flask-PyMongo.” Web Page, 2008 [Online]. Available: https://flask-pymongo.readthedocs.io/en/latest/

[37] MongoDB, “Flask MongoAlchemy.” Web Page, 2008 [Online]. Available: https://pythonhosted.org/Flask-MongoAlchemy/

[38] MongoDB, “Flask-MongoEngine.” Web Page, 2008 [Online]. Available: http://docs.mongoengine.org/projects/flask-mongoengine/en/latest/

[39] Wikipedia, “Flask (web framework).” Web Page, Oct-2018 [Online]. Available: https://en.wikipedia.org/wiki/Flask_(web_framework)