Proceedings of the REU2022

Cybertraining

Gregor von Laszewski, laszewski@gmail.com

04 June, 2022

Contents

1 USEFUL LINKS

1.1 REU 2022

1.1.1 Books

- https://cloudmesh-community.github.io/pub/vonLaszewski-python.pdf
- https://cloudmesh-community.github.io/pub/vonLaszewski-python.epub
- https://cloudmesh-community.github.io/pub/vonLaszewski-linux.pdf
- https://cloudmesh-community.github.io/pub/vonLaszewski-linux.epub
- https://cloudmesh.github.io/cloudmesh-mpi/report-mpi.pdf
- https://cloudmesh-community.github.io/pub/vonLaszewski-writing.pdf

- https://cloudmesh-community.github.io/pub/vonLaszewski-communicate.pdf
- https://cloudmesh-community.github.io/pub/vonLaszewski-reu2022.pdf

1.1.2 The repository for our report(s)

- https://github.com/cybertraining-dsc/reu2022

1.1.3 The many UVA technical help ticket systems

- ITS: https://virginia.edusupportcenter.com/shp/uva/helpcenter
- Rivanna: https://varesearchhelp.atlassian.net/servicedesk/customer/user/requests?page=1&reporter=all
- BII: https://uva-biocomplexity.atlassian.net/servicedesk/customer/user/requests?page=1&reporter=all&statuses=open&statuses=closed

1.1.4 Our technical slack

- https://cloudmesh-reu2022.slack.com

1.1.5 The administrative slack

- https://biocomplexity-eoo5671.slack.com/archives/C031CR0B2QG

1.1.6 Rivanna Information

- Research Computing: https://www.rc.virginia.edu/
- Rivanna Overview: https://www.rc.virginia.edu/userinfo/rivanna/overview/
- Research Computing Support Center: https://www.rc.virginia.edu/support/#office-hours
- access through ssh: https://www.rc.virginia.edu/userinfo/rivanna/login/
- access through Web Browser: https://shibidp.its.virginia.edu/idp/profile/SAML2/Redirect/SSO?execution=e2s1
- Rivanna Slide Show: https://learning.rc.virginia.edu/notes/rivanna-intro/
- Globus Data Transfer (not using for reu2022) https://www.rc.virginia.edu/userinfo/globus/
- Slurm Job Manager Documentation https://www.rc.virginia.edu/userinfo/rivanna/slurm/
- Rivanna Allocations: https://www.rc.virginia.edu/userinfo/rivanna/allocations/

1.1.7 GitBash (Windows only)

- https://git-scm.com/downloads

1.1.8 Pycharm

- pycharm community edition: https://www.jetbrains.com/pycharm/download
- md: https://www.jetbrains.com/help/pycharm/markdown.html
- (optional) extension makefile: https://plugins.jetbrains.com/plugin/9333-makefile-language
- (optional) rst: https://www.jetbrains.com/help/pycharm/restructured-text.html
- 80 char: https://intellij-support.jetbrains.com/hc/en-us/community/posts/206070859-How-do-I-enable-the-80-column-guideline-change
- background to white as i can not read white on black when in meetings with me I will not support anyone that has black background: https://www.jetbrains.com/help/pycharm/user-interface-themes.html use intelliJ Light. After the meeting you can set it to whatever.

1.1.9 Bibliography Management

- jabref: https://www.jabref.org/
- bibtex: https://en.wikipedia.org/wiki/BibTeX
- draft bibtex from urls: https://addons.mozilla.org/en-US/firefox/addon/bibitnow/ (has to be modified once copied. Not everything will work.

1.1.10 GitHub repo

- https://github.com/cybertraining-dsc/reu2022
- Sandra, JP: https://github.com/cloudmesh/cloudmesh-mpi

1.1.11 Only important for Gregor (do not use)

- request storage: https://www.rc.virginia.edu/form/storage/
- rivanna partition/account: https://www.rc.virginia.edu/userinfo/rivanna/allocations/
- managing groups: https://mygroups.virginia.edu/

2 CONTRIBUTION TO THE PROJECT

2.1 Contribute

Learning Objectives

- Learn how to contribute
- Identify who contributes what
- Install and use `cms git` command

Before you do anything, check first if it is already in one of the books. Evaluate if it needs to be expanded in the book, or if a new section is needed. If a new section is needed, please consult with Gregor and ask for approval. We will then also decide if the chapter will be located in reu2022 or in books (both are repos).

Contribution will be determined based on review of commits, lines in such a fashion that the final lines willbe considered. For example let us assume you spent significant time on a section that ia a duplication of what others do, we will then delete that sections and you have not achieved a contribution.

So please make sure that you contribute valuable things.

Ask if you have an idea and want confirmation.

2.1.1 GitHub Insights

- Project: https://github.com/cybertraining-dsc/reu2022/projects/2
- Issues: https://github.com/cybertraining-dsc/reu2022/issues
- Pulse: https://github.com/cybertraining-dsc/reu2022/pulse
- Contributors: https://github.com/cybertraining-dsc/reu2022/graphs/contributors
- Traffic: https://github.com/cybertraining-dsc/reu2022/graphs/traffic

2.2 Calender

- dynamic calendar
- recorded meetings

2.2.1 Team presentations

- June 2, Program start
- June 13, JP: title TODO
- June 15, Alison
- June 22, Jackson
- June 28, Abdul, Alex
- June 29, (Paul)
- July 6, JP
- July 13,Jackson, Alision
- July 26 Abdul, Alex
- July 27 (Paul)
- July 28, Program end

2.3 Lines contributed

```
ERROR: this command does not exist: 'git'
```

2.4 Issues

Cloudmesh contains a conveient program to list all issues of reporitories downloaded in the current directory. The command can be installed with

```
$ pip install cloudmesh-git
```

alternatively you can simply install it from source with

```
$ cd cm
$ git clone git@github.com:cloudmesh/cloudmesh-git.git
$ cd cloudmesh-git
$ pip install -e .
$ cd ..
```

To run it cd to the directory where all your git repos are located and say

```
$ cms git issues --repo=. --refresh
```

This will open a Web page that list all issues across all repos in that directory.

2.4.1 Issues for the REU2022

In case you like to see all issues only for the REU2022, please check out the following

```
$ cd cm
$ cloudmesh-installer get cmd5
$ git clone git@github.com:cloudmesh/yamldb.git
$ git clone git@github.com:cloudmesh/cloudmesh-git.git
$ git clone git@github.com:cloudmesh/cloudmesh-catalog.git
$ git clone git@github.com:cloudmesh/cloudmesh-data.git
$ git clone git@github.com:cloudmesh/cloudmesh-sbatch.git
$ git clone git@github.com:cloudmesh/cloudmesh-mpi.git
$ git clone git@github.com:cloudmesh/cloudmesh-slurm.git
$ git clone git@github.com:cloudmesh/cloudmesh-pi-burn.git
$ git clone git@github.com:cloudmesh/cloudmesh-pi-cluster.git
$ git clone git@github.com:cloudmesh-community/book.git
$ git clone git@github.com:cybertraining-dsc/reu.git
$ git clone git@github.com:cyberaide/bookmanager.git
```

To get the list use

```
$ cms git issues --repo=reu --refresh
```

2.5 Communications with C4GC Students

- There are weekly meetings with Ben Hurt every Tuesday and Wednesday from 11-11:45 am on Zoom and in room 4402.
- Around 4 students present each at the meetings. Every student presents twice and the link to the schedule can be found at this link.
  - first presentation is about the student's goals and the introduction to the project:
    * How you define success for this summer
    * Who you're working with
    * What your project is
    * What you've done so far
    * [optional] What challenges you've faced so far
  - It should be around 5-7 minutes long.
- In order to access the Biocomplexity Institute building, you must alert your mentor and sign up on the BII app. If coming for the first time you need to request access with Ben S.
- C4GC Links and Recordings
- Rivanna Orientation (C4GC specific) on Thursday June 9, 10 am - 11 am.

2.6 Teamwork

Learning Objectives

- Principals of Teamwork

---

1. Team works together
2. Team always communicates with each other
3. A canceled meeting for a dasy does not mean that the teamwork stops
4. If you need help you must ask the team
5. If a team member does not want to help you that is not team work
6. If you can not keep up with the team memberes you need to step up
7. If you are assigned a task and you can not do it you need to ask for help
8. If you spend hours or days on a task that you can not do you need to ask for help
9. In case you know you have an issue with a task you need to communicate that early
10. In case you need to go on vacation 2 weeks before the REU is over, this request will be denied
11. In case you do need to go on vacation, you need to provied a concrete plan on (a) how does this work effect the team and how will your work be distributed to the team, (b) what are concrete workhours that you propose to make up the work (c) be aware that any vacation time not only negatively impacts the team time planing, but also your supervisors work

    (d) plan any additional impact on your work, for example you could have other obligations in unrelated projects.

Always ask yourself everyday:

- What have you done to increase teamwork?
- What is a concrete contribution you made for the team?
- Did you help enyone?
- Were you helped by anyone?
- Did you communicate your progress?

3 INSTALL

3.1 Install

Learning Objectives

- Learn how to install python from python.org
- Learn how to use a python venv
- Learn how to install cloudmesh Shell

In this section, we present an easy-to-follow installation guide for a recent version of python and cloudmesh.

Before you start, please read the entire section and develop a plan for installation.

3.1.1 Windows

The installation of cloudmesh benefits from using Git Bash as it allows us to have a terminal that is similar to that of macOS and Linux.

Hence, before we install python and cloudmesh we install Git Bash.

Furthermore, we provide the option to use chocolatey to install packages in similar fashion as on linus.

3.1.1.1 Git Bash install First, make sure you completly uninstall previous versions of gitbash. If you do not have it installed, you can skip this step.

```
Go to Start Menu > Add Or Remove Programs
```

In the search bar that is located on that page, type Git. Then just click on Uninstall and follow the instructions.

To install Git Bash, pleas download it first from

- https://git-scm.com/downloads

Click `Download` for Windows. The download will commence. Please open the file once it is finished downloading. Next, please start the downloaded program and follow the instructions carefully.

- The administration window (UAC Prompt) will appear. Click `Yes`. It will show you Git's license: a GNU General Public License. Read it and Click `Next`. To ensure security of the operating system, a UAC prompt allows operating systems, particularly Windows, to prompt for consent or credentials from local administrators before starting a program.
- Click `Next` to confirm that `C:\Program Files\Git` is the directory where you want Git to be installed.
- Select the box to create a shortcut icon on the desktop. Click `Next` to continue with the install.
- Click `Next` to accept the default text editor which is vim,
- Replace the default branch name (`master`) with `main`
- Click `Next` and check on to run Git from the command line and 3rd party software,
- Click `Next` and check on to use the OpenSSL library
- Click `Next` and check on to check out Windows-style,
- Click `Next` and check on to use MinTTY,
- Click `Next` and check on to use the default git pull,
- Click `Next` and check on to use the Git Credential Manager Core,
- Click `Next` and check on enable file system caching, and then
- Click 'Next' and check on Enble experiment support for pseudo consoles
- Click `Install` because we do not need experimental features.

A video tutorial on how to install Git and Git Bash on Windows 10 is located at https://youtu.be/HCotEx_xCfA

A written tutorial on how to install Git and Git Bash on Windows 10 is located at https://cybertraining-dsc.github.io/docs/tutorial/reu/github/git/

3.1.1.2 Python 3.10 install

To install Python 3.10 please go to

- https://python.org

and download the latest version.

- Click `Download`. The download will commence. Please open the file once it is finished downloading
- Click the checkbox `Add Python 3.10 to PATH`
- Click `Install Now`
- At the end of the installation click the option to `Disable path length limit`

A video tutorial on how to install Professional PyCharm is located at https://youtu.be/QPESX-VBnEU

A video on how to configure PyCharm with cloudmesh is located at https://youtu.be/eb1IQBx0D50

3.1.1.3 Installing cloudmesh Cloudmesh can be installed in any shell that has python and `git` access. However, it is convenient to use Git Bash as it simplifies the documentation and allows us to interact with Linux commands with the Windows file system. The installation is done with a Python virtual environment `ENV3` using command `venv` so you do not affect your computer's current python configurations or settings. Here are the steps:

```
$ python -m venv ~/ENV3
$ source ~/ENV3/Scripts/activate
$ cd
$ mkdir cm
$ cd cm
$ pip install pip -U
$ pip install cloudmesh-installer
$ cloudmesh-installer get cmd5
$ cms help
$ touch .bashrc
$ echo "source ~/ENV3/Scripts/activate" >> .bashrc
$ echo "cd ~/cm" >> .bashrc
```

To activate it, start new Git Bash and terminate the first Git Bash window. If you see in the new window `(ENV3)`, continue. Git Bash will initialize the environment.

If you do not want to always start in the directory `cm` do replace the line in your `.bashrc` `cd cm` with `cd`

3.1.1.4 Uninstall To remove the virtual environment `ENV3`, use the following command:

```
$ rm -f ~/ENV3
```

Next, edit the `.bashrc` and `.bash_profile` file and delete the lines:

```
$ source ~/ENV3/Scripts/activate
$ cd cm
```

3.1.2 Choco install

There are a number of useful packages that you can install via choco. This includes Visual Code, Pycharm, Emacs, and make. Even Python could be installed with it; however, we have not tested the installation of python via choco, while we have tested the installation of Emacs, Pycharm, and make.

3.1.3 Install Chocolatey

To install, pleas start a Git Bash terminal as administrator: To do so press the `Windows` key and type powershell. Click `Run as Administrator`. Click `Yes`.

2. In PowerShell execute the following command:

```
PS C:\Windows\system32> Set-ExecutionPolicy AllSigned
```

   Then type `y`.

3. Next type in the command (copy and paste to not make a mistake)

```
PS C:\Windows\system32> Set-ExecutionPolicy Bypass -Scope Process -Force; [
    ↪ System.Net.ServicePointManager]::SecurityProtocol = [System.Net.
    ↪ ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
    ↪ System.Net.WebClient).DownloadString('https://community.chocolatey.org/
    ↪ install.ps1'))
```

4. Wait for the installation to complete; once you see

```
PS C:\Windows\system32>
```

with a blinking cursor again, and lines have stopped appearing, then the Chocolatey installation has finished. Type `choco` and you should see Chocolatey in green text.

Now you can install many programs with choco by launching PowerShell as Administrator or Git Bash.

A list of programs that you can install with `choco` can be found at:

- https://community.chocolatey.org/packages/

3.1.4 Installing Useful Developer Programs

The following useful developer programs can be installed. Select the once you like and install them with the appropriate command

```
$ choco install make -y
$ choco install emacs -y
$ choco install pycharm -y
$ choco install firefox -y
$ choco install vscode -y
$ choco install zoom -y
```

Once the installation completes, your program will be ready for you to use.

3.1.5 Linux

3.1.5.1 Install Python 3.10.5 The installation from source can be done easily as shown:

```
$ mkdir -p ~/tmp
$ cd ~/tmp
$ wget https://www.python.org/ftp/python/3.10.5/Python-3.10.5.tar.xz
$ tar xvf Python-3.10.5.tar.xz
$ cd Python-3.10.5/
$ ./configure --enable-optimizations
$ make -j $(nproc)
$ sudo make altinstall
$ pip install pip -U
$ python3.10 -V
```

3.1.5.2 Setting up the venv We assume you use bash

```
$ python3.10 -m venv ~/ENV3
$ source ~/ENV/bin/activate
$ cd
$ mkdir cm
$ cd cm
$ pip install cloudmesh-installer
$ cloudmesh-installer get cmd5
$ cms help
$ touch .bashrc
$ echo "source ~/ENV3/bin/activate" >> .bashrc
$ echo "cd cm" >> .bashrc
$ echo "source ~/ENV3/bin/activate" >> .bash_profile
$ echo "cd cm" >> .bash_profile
```

3.1.5.3 Uninstall

```
$ rm -f ~/ENV3
```

Edit the `.zshrc` and `.zprofile` file and delete the lines

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

3.1.5.4 Update In case you need to update the Python version it is sufficient to follow the instructions provided in the section `Install Python 3.10.5`, while replacing the version number with the current python release number.

In case you need to create a new virtual ENV3. You can first uninstall it and then reinstall it.

An easy way to do all of this with a command is the following:

```
$ cd ~/cm
$ pip install cloudmesh-installer -U
$ pip install --upgrade pip
$ cloudmesh-installer new ~/ENV3 cmd5 --python=/usr/local/bin/python3.10
$ source ~/ENV3/bin/activate
$ python -V
$ which python
```

3.1.6 macOS

We assume you use `~/zsh` which is the default on macOS

3.1.6.1 Xcode Install There are a number of digital tools that are needed before proceeding further. These tools include git, make, and a c compiler. All of these tools can be downloaded at Xcode, which is an IDE App on the Apple App Store that includes all of these necessary elements.

Once installed, there is one simple command line command to run:

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

This will install all the necessary command line tools. Xcode can be used as an IDE, but for the most part will not be used outside the command line tools it provides.

3.1.6.2 Cloudmesh

3.1.6.2.1 Install Before any of the following, make sure to download the current version of python. At the time of this writing, it is python 3.10.5.

Second, execute the following commands in your terminal. Make sure to do this in order.

```
$ cd
$ python3.10 -m venv ~/ENV3
$ source ~/ENV/bin/activate
$ pip install pip -U
$ mkdir cm
$ cd cm
$ pip install cloudmesh-installer
$ cloudmesh-installer get cmd5
```

```
$ cms help
$ echo "source ~/ENV3/bin/activate" >> .zshrc
$ echo "cd cm" >> .zshrc
$ echo "source ~/ENV3/bin/activate" >> .zprofile
$ echo "cd cm" >> .zprofile
```

It creates the virtual environment, a directory called `~/cm`, then installs `cloudmesh`. Following this, it sets the macOS startup commands `.zshrc` and `.zprofile` to start up in the virtual environment `~/ENV3`.

3.1.6.2.2 Uninstall

```
$ rm -rf ~/ENV3
```

You may need to enter your system password.

3.1.6.3 Updating Python Before starting this process, ensure that python is in the correct path. Test in the terminal.

To do so remove the existing ENV3 first and start a new terminal in which you will be working.

```
$ rm -rf ~/ENV3
```

Start the new terminal and execute the commands to verify if you have the right updated version of python

```
$ which python3.10
$ where python3.10
$ python3.10 -V
```

Now execute:

```
$ cd ~/cm
$ python3.10 -m venv ~/ENV3
$ source ~/ENV3/bin/activate
$ pip install pip -U
$ pip install cloudmesh-installer
$ cloudmesh-installer get cmd5
$ cms help
```

As `~/zsh` is already configured previously, we do not have to set it up again.

3.1.6.4 Homebrew install Homebrew is a package management software. The Homebrew command is called `brew`. It is used to install Linux packages on macOS. Installing `brew` is simple.

- First, make sure the computer that is downloading Homebrew is up-to-date with the latest software for its OS.
- Second, ensure that `xcode` has been installed. `xcode` can be installed from the Apple App Store.
- Third, in the terminal, write out:

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

  This installs `xcode` command line tools.

- Fourth, run the following command in the terminal:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install
    ↪ /HEAD/install.sh)"
```

- Fifth, enter the administrator password into the desired location.
- It may take a moment for the software to install, but it will eventually say **"Installation successful!"** in the terminal. After that, Homebrew is installed onto the device.

After the user has correctly installed Homebrew, it is simple to install packages directly to the operating system:

```
$ brew install [package name]
```

3.2 Ramdisk

Learning Objectives

- Learn how to set up a RAM disk
- Learn how to use a DAM disk

how to set up aand manage a ramdisk on lunux

develop cms program

cms ramdisk –… –size=SIZE

use humanize so we can us 1GB for size …

showcase

a) dynamic ramdisk no reboot needed, but if reboot, ramdisk needs to be set up new
b) ramdisk integrated in fstab with reboot
c) backu and load ramdisk

On macOS a RAM disk with 512MB space can be created with the following command:

```
n = 512 * 2048
os.system('diskutil eraseVolume HFS+ "RAMDisk" `hdiutil attach -nomount ram://{n}`')
```

3.2.1 Ubuntu

On Ubuntu, a RAM disk and its read-only shadow can be created by:

```
mount -t tmpfs -o size=512m tmpfs /mnt/ramdisk
mount -t aufs -o br:/mnt/ramdisk=ro none /mnt/readonly

# mkdir /tmp/ramdisk; chmod 777 /tmp/ramdisk
# mount -t tmpfs -o size=256M tmpfs /tmp/ramdisk/
```

using /dev/shm:

http://ubuntuguide.net/ubuntu-using-ramdisk-for-better-performance-and-fast-response

Various methods: (in german): https://wiki.ubuntuusers.de/RAM-Disk_erstellen/

ramfs is an older file system type and is replaced in mostly by tmpfs.

3.2.2 windows

https://forums.guru3d.com/threads/guide-using-imdisk-to-set-up-ram-disk-s-in-windows-with-no-limit-on-disk-size.356046/

4 GRAPH VIZUALIZATION

	matlotlib	seaborn	bokeh	pandas
charts				
barchart	plt.bar	sns.barplot	p.vbar	df.plot.bar
grouped bar-chart	plt.bar	sns.catplot	-	df.plot.bar
stacked bar-chart	plt.bar	-	p.vbar_stack	df.plot.bar(stacked=True)
spline-chart	-	-	-	-
multiline chart	plt.plot	sns.lineplot	p.multi_line	df.plot.line
compound line chart		sns.lineplot	-	df.plot.area
histogram	plt.hist	sns.histplot	-	df.plot.hist
linechart	plt.plot	sns.lineplot	p.line	df.plot.line
piechart	plt.pie	-	p.wedge	df.plot.pie
exploded piechart	plt.pie(explode=)	-	-	-
donutchart	plt.pie(wedgeprops=)	-	Donut	-
countourplot	plt.contour	sns.kdeplot	-	-
distributionplot	plt.hist	sns.displot	Histogram	-
point-chart		sns.pointplt	-	df.plot.scatter
scatterplot	plt.scatter	sns.scatterplot	Scatter	df.plot.scatter
bubblechart	plt.scatter	-	-	df.plot.scatter(s=…, c=…)

	matlotlib	seaborn	bokeh	pandas
radar chart	plt.plot	-	-	-
boxplot	plt.boxplot	sns.boxplot	Boxplot	df.plot.boxplot
features				
pdf export	+		-	
png export	+	via matplotlib	export_png	via matplotlib
svg export	+	via matplotlib	export_svg	via matplotlib
color palettes	-	sns.color_palette	Color Palettes	-
interactive graph		-	+	-
data frame	via Pandas	via Pandas	via Pandas	link missing

4.1 Python Data Management for Visualizations

In python, there are several ways that data can be interpreted in order to generate the graphics for data visualization.

Through several examples, we will show how to manage different types of data and how to best interact with them. They are lists, dictionaries and CSV files.

4.1.1 Lists

Lists are clumps of continuous values that are put directly next to each other in the computer memory system.

4.1.1.1 Construction

In python, lists can be constructed like this:

```
example_list = ['example', 2, 'b', 45, False]
```

Lists have order and can hold many types (bool, int, String, etc.) of values.

4.1.1.2 Accessing Values

In python, it is easy to access values within a list. The following code provides an example of how to access certain values within a list:

```
index_4 = example_list[4]
print(index_4)
# output is False
```

It is important to note that lists have indices, which range from 0 to the length of the list - 1. The indices provide access to values within the list.

4.1.1.3 Updating Values

To update a value within a list, simply utilize same brackets:

```
example_list[3] = 'bear'
print(example_list[3])
# output is 'bear' instead of 45
```

This will change the value at the third index from 45 to bear.

4.1.1.4 Python Built-In Methods

Python has several built-in methods for lists. These include `append()`, `clear()` `copy()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove(`, `reverse()`, and `sort()`.

A user can utilize these methods to make changes in the necessary ways to the list.

4.1.2 Dictionaries

Dictionaries are like specialized lists. They hold a key-value pair that allows for a user to look up a key and find the associated value. Dictionaries are useful for storing values in a way that is more organized than a linear list. Furthermore, dictionaries make it easy for users to look up the necessary information.

4.1.2.1 Construction

In python, dictionaries are constructed as follows:

```
example_dictionary = {'motorcycles': 2, 'autocycles': 3,
                      'cars': 4, 'small_trucks': 6
                      'large_trucks': '18'}
```

The string values are the keys, which provide access to the values within the dictionary. The colon provides the computer with the command for assigning key-value pair.

4.1.2.2 Accessing Values

There are several built-in commands to access both the keys and values in a dictionary. They are as follows:

```
example_dictionary.get()
example_dicionary.keys()
example_dictionary.values()
```

The `.get()` method returns the value that is associated with the given key, the `.keys()` method returns a list of the keys alone, and the `.values()` method returns a list of the values alone. There are more methods (see the Python Built-In Methods section)

4.1.2.3 Updating Values

To update the dictionary, there is one method that can be used:

```
example_dictionary.update({'motorcycles': 20})

# or to add a new key-value pair to the dictionary:

example_dictionary['New Key'] = 'New Value'
```

This will update the value associated with this particular key-value pair.

4.1.2.4 Python Built-In Methods

There are several built-in methods that allow for more dictionary manipulation: They are `clear()`, `copy()`, `fromkeys()`, `items()`, `pop()`, `popitem()`, and `setdefault()`.

4.1.3 CSV Files

CSV stands for *comma-separated-values* and is a data structure that is incredibly common for data management and analysis. There are many ways to access CSV files. The most common way is to use pandas, a python library. However, python also has a built-in CSV module that can be used. We will show examples of using both below.

4.1.3.1 Installing and Importing

Before beginning with any of these CSV manipulation tasks, it is necessary to install and import the correct modules and libraries. The following showcases this:

```
$ pip install pandas
```

```
import pandas as pd
import csv
```

4.1.3.2 Construction

CSV files are files that exist elsewhere and have already been created. Therefore, for creation, we are not creating a CSV files, but rather deconstructing it into something that is usable.

In pandas, this means creating a dataframe, which is essentially and indexed table. In the python `csv` module, this means using the `csvreader` object within the module. Following are two examples showcasing how exactly to do this.

For the `csv` module:

```
# open the csv and creates the reader object for it
file = open('/Users/jacksonmiskill/Downloads/biostats.csv')
reader = csv.reader(file)
```

The `reader` is an object that was created by python developers to help parse through the `csv` files.

For the `pandas` module:

```
file = pd.read_csv("/Users/jacksonmiskill/Downloads/biostats.csv")
df = pd.DataFrame(file)
```

4.1.3.3 Accessing Values

For the `csv` module:

It is more challenging to access files using the `csv` module as opposed to the `pandas` library. To make it more simple, it is necessary to convert the values that lie within the `csv` into a list in order to access.

```
data = []
for each_row in reader:
    data.append(each_row[2])

print(data)  # output is the whole list
```

This code is available on GitHub

For the `pandas` module, it is less complicated to access that values. This is because the module includes a function that converts the data into a dataframe. After converting, you can use the various methods that are within the dataframe to essentially pass in the correct values.

4.1.3.4 Examples Once the `csv` values have been accessed, creation of the graphics can begin. Starting with the `csv` module and then moving into `pandas` the following will demonstrate this action.

The `csv` file that will be utilized for the following examples can be found here. It represents made up data on a group of made up people such as age, height, and weight.

It is slow and complicated to create graphs with the `csv` module. We have implemented a separate module called `ConvertCSV` which provides the user with the ability to convert the data received from the `csv` module into doubly nested lists, lists, and dictionaries, depending on necessity.

```
# reading in the data!
filename = path_expand("./biostats")
table = ConvertCSV.create_table(filename)
print(table)

list_names = []
list_heights = []

list_names = ConvertCSV.csv_read_to_list(table=table, index=0, convert=False)
list_heights = ConvertCSV.csv_read_to_list(table=table, index=3, convert=True)

dict_names = []
dict_heights = []

idict = ConvertCSV.csv_read_to_dict(table=table, index=3)
dict_names = list(idict.keys())
dict_heights = list(idict.values())

# now just plot these values. You can use whichever library you desire
# for the example, we use matplotlib, because it is simple

# list
plt.plot(list_names, list_heights)
plt.xlabel("Names")
plt.ylabel("Heights")
plt.xticks(rotation=90)
plt.savefig('images/csv-list-lineplot.png')
plt.savefig('images/csv-list-lineplot.svg')
```

```
plt.savefig('images/csv-list-lineplot.pdf')
plt.title("Names and Corresponding Height")
plt.show()

# dictionary
plt.plot(dict_names, dict_heights)
plt.xlabel("Names")
plt.ylabel("Heights")
plt.xticks(rotation=90)
plt.savefig('images/csv-dict-lineplot.png')
plt.savefig('images/csv-dict-lineplot.svg')
plt.savefig('images/csv-dict-lineplot.pdf')
plt.title("Names and Corresponding Height")
plt.show()
```

This code can be access from GitHub

This code produces Figure **??**. and Figure Figure **??**.:

{#fig:csv-list-lineplot width=50%}

Figure **??**: created using the data available here.

{#fig:csv-dict-lineplot width=50%}

Figure **??**: created using the data available here.

However, it is so much more simple to accomplish this with the `pandas` library:

```
file = pd.read_csv("/Users/jacksonmiskill/Downloads/biostats.csv")
biostats = pd.DataFrame(file)

plt.plot(biostats['Name'], biostats[' "Height (in)"'])
plt.xlabel("Name")
plt.ylabel("Height")
plt.xticks(rotation=90)
plt.savefig('images/pandas-lineplot.png')
plt.savefig('images/pandas-lineplot.svg')
plt.savefig('images/pandas-lineplot.pdf')
plt.title("Names and Corresponding Height")
plt.show()
```

This code can be accessed from GitHub

This code produces the Figure **??**:

{#fig:pandas-lineplot width=50%}

Figure **??**: created using the data available here.

4.1.4 Links

4.1.4.1 Lists

- https://towardsdatascience.com/python-basics-6-lists-and-list-manipulation-a56be62b1f95
- https://www.w3schools.com/python/python_ref_list.asp

4.1.4.2 Dictionaries

- https://www.pythonforbeginners.com/dictionary/dictionary-manipulation-in-python
- https://www.w3schools.com/python/python_ref_dictionary.asp
- https://www.w3schools.com/python/ref_dictionary_update.asp

4.1.4.3 CSV Files

- https://www.geeksforgeeks.org/creating-a-dataframe-using-csv-files/
- https://docs.python.org/3/library/csv.html#examples
- https://people.sc.fsu.edu/~jburkardt/data/csv/csv.html
- https://docs.python.org/3/library/functions.html#open
- https://www.protechtraining.com/blog/post/python-for-beginners-reading-manipulating-csv-files-737#extracting-information-from-a-csv-file
- https://stackoverflow.com/questions/13039392/csv-list-index-out-of-range
- https://www.geeksforgeeks.org/visualize-data-from-csv-file-in-python/

4.2 Python Graphics

Learning Objectives

- Introduction to plotting libraries
  - Introduction to matplotlib
  - Introduction to seaborn
  - Introduction to bokeh
  - Introduction to pandas plots
- Introduction to graph plotting libraries
  - Introduction to networkX

  - Introduction to garphvis
  - Introduction to mermaid
  - Introduction to rackdiag

- Identify which library to chose

In Python, data and equations can be visually represented using graphs and plots. Here we showcase how to use the different plotting libraries Matplotlib, Bokeh, and Seaborn.

For each of these frameworks exist an extensive example set as part of Galleries included with the original documentation. We encourage to browse through these examples to identfy plots that you may want to generate.

- matplotlib gallery
- seaborn gallery
- bokeh gallery
- pandas gallery

Another combined gallery is available as

- (interactive selection)[https://www.python-graph-gallery.com/]

And provides you with choices based on your selection.

4.2.1 Matplotlib

Matplotlib is the main plotting library that allows the user to visualize data. Matplotlib creates figures that can be manipulated and transformed. This includes manipulations of axes, labels, fonts, and the size of the images.

4.2.1.1 Installation To install matplotlib, please use the command:

```
$ pip install matplotlib
```

4.2.1.2 Import Statements The user will need to supply these import statements at the top of their code in order for Matplotlib to be imported.

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

4.2.1.3 Bar Chart In Matplotlib, it is easy to create bar charts. For example, this is a demonstration of a simple bar chart using data from a user using Spotify.

```
import matplotlib.pyplot as plt

# you can also do this: from matplotlib import pyplot as plt

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
categories = data.keys()
count = data.values()

# Creating the bar chart
plt.bar(categories,
        count,
        align='center',
        color='darkorange',
        width=0.4,
        edgecolor="royalblue",
        linewidth=4)

# Editing the bar chart's title, x, and y axes
plt.xlabel("Genre of Music")
plt.ylabel("Number of songs in the genre")
plt.title("Distribution of Genres in My Liked Songs")
plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 1.

Fig. 1. Spotify Data Shown in Bar Chart

4.2.1.4 Line Chart The Matplotlib library in python allows for comprehensive line plots to be created. Here a line chart was created using a for loop to generate random numbers in a range and plot it against the `x` and `y` axis to display the changes between two variables/data sets.

```
import matplotlib.pyplot as plt
import random

x = []
y = []
for i in range(0, 100):
  x.append(i)
  value = random.random() * 100
  y.append(value)

# creating the plot and labeling axes and title
```

```
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot Test")
plt.show()
```

This program can be downloaded from GitHub . The output of this program is showcased in Fig. 2.

Fig. 2. Sample Randomized Line Plot

4.2.1.5 Pie Chart A pie chart is most commonly used when representing the division of components that form a whole thing e.g. showing how a budget is broken down into separate spending categories. In Matplotlib, the function `pie()` creates a pie chart. In the following code example, a user's Spotify data will be displayed as a pie chart.

```
import matplotlib.pyplot as plt

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
categories = data.keys()
count = data.values()

# Creating the pie chart
plt.pie(count, labels=categories)

plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 3.

Fig. 3. Genres of Music from Spotify Distributed on a Pie Chart

4.2.1.6 Contour Plot Unlike the previous types of plots shown, contour plots allow data involving three variables to be plotted on a 2D surface. In this example, an equation of a hyperbolic paraboloid is graphed on a contour plot.

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

# creating an equation for z based off of variables x,y
x, y = np.meshgrid(np.linspace(-10, 10), np.linspace(-10, 10))
z = 9 * (x ** 2 + 1) + 8 * x - (y ** 2)
levels = np.linspace(np.min(z), np.max(z), 15)

# creating a contour graph based off the equation of z
plt.contour(x, y, z, levels=levels)
```

```
plt.xlabel("x")
plt.ylabel("y")
plt.title("Function of z(x,y)")
plt.show()
```

This program can be downloaded from GitHub . The output of this program is showcased in Fig. 4.

Fig. 4. Multivariable Math Equation Shown on a Contour Plot

A contour plot allows data and equations consisting of three variables to be plotted through plotting 3D surfaces as 2D slices on a `xy` plane. Matplotlib can display data and equations through contour graphs after they are inputted. Shown below are the parameters for `plt.contour` .

```
plt.contour([x, y], z, levels)
```

The independent variables `x` and `y` must be defined so the dependent variable `z` can be defined. The variables can come in the form of a list or dictionary or as an equation. The `levels` parameter determines the number of contour lines that can be drawn.

4.2.1.7 Titles, Labels, and Legends

4.2.1.7.1 Titles Titles are necessary to let the reader know about your graph or plot is exactly about. To give a title to your whole graph in matplotlib, simply type:

```
plt.title("Title you want to set").
```

4.2.1.7.2 x-axis labels and y-axis labels Within the Matplotlib library are the functions `plt.xlabel()` and `plt.ylabel()` . All these functions do is set a string to the two axes. To use these functions, simply type:

```
plt.xlabel("Label you want to set")
plt.ylabel("Label you want to set")
```

4.2.1.7.3 Legend Sometimes, a legend may be necessary to let the reader know which part of the graph/plot corresponds to each part of the data shown. To show a legend, use the command:

```
plt.legend()
```

4.2.1.8 Rotating Ticks When a chart is created, ticks are automatically created on the axes. By default, they are set horizontally; however, they can be rotated using `plt.xticks(degrees)` for the `x-axis` or `plt.yticks(degrees)` for the `y-axis`.

```
import matplotlib.pyplot as plt

x = range(0, 4)
y = x
plt.plot(x, y)

# Rotating Ticks
plt.xticks(rotation=90)
plt.yticks(rotation=45)

plt.xlabel('x values')
plt.ylabel('y values')
plt.title(r'$y=x$')
plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 5.

Fig. 5. Line Plot with `x-axis` Ticks Rotated by 90° & `y-axis` Ticks Rotated by 45°

4.2.1.9 Exporting

4.2.1.9.1 Saving Chart as Files After a chart is created and displayed, it can be exported as a file outside the code using this command:

```
plt.savefig("fname", dpi='figure')
```

The name and format of the file are set as a string using `fname`. Make sure to specify the format of the file by using a `.` after the file name and specify the type after such as `.pdf`, `.png`, `svg`, etc.

The parameter `dpi` sets the DPI (Dots per Inch) of the image being saved. Specify this number in the form of a float. For example, set `dpi=300`.

Additionally, there is another way to save files that may be faster than calling a specific method for each file. The following code showcases this:

```
import matplotlib.pyplot as plt
import os
from matplotlib import pyplot
```

```
def save():
  name = os.path.basename(__file__).replace(".py", "")
  plt.savefig(f'/filepath/{name}.png')
  plt.savefig(f'filepath/{name}.pdf')
  plt.savefig(f'filepath/{name}.svg')
  plt.show()
```

This code can be accessed on GitHub

4.2.1.9.2 Display The very last command that should be written is `plt.show()`, as this command displays the graph that you made. To show, simply type:

```
plt.show()
```

4.2.1.10 Links

- https://matplotlib.org/
- https://matplotlib.org/stable/api/pyplot_summary.html
- https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/
- https://www.geeksforgeeks.org/bar-plot-in-matplotlib/

4.3 Seaborn

Seaborn, like Matplotlib, is a data visualization tool. However, the graphs and charts that Seaborn can create are more complex than Matplotlib. The graphs that are created in Seaborn are more statistically detailed. Unlike matplotlib, Seaborn draws upon other imported libraries such as Matplotlib, Numpy, and Pandas. This is because Seaborn relies on more complex math (Numpy) and data frames (generated from Pandas) that are passed into its functions as the data.

Several types of plots can be made from Seaborn; they are relational, distributional, categorical, regression, and matrix plots.

We have created examples to demonstrate the abilities of Seaborn.

4.3.1 Installation

Seaborn can be installed in the same way as the other libraries installed earlier. The user who is installing the library should make sure that it is being installed in the correct environment.

```
$ pip install seaborn
```

4.3.2 Import Statements

The user will need to supply these import statements at the top of their code in order for Seaborn to be imported. Additionally, the data created for the examples represents a user's Liked songs from Spotify.

```
import seaborn as sns
import matplotlib.pyplot as plt

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
categories = list(data.keys())
count = list(data.values())
personal_rank = [3, 4, 2, 1, 5]
```

4.3.3 Relational Plots

Relational plots showcase the relationship between variables in a visual format. It is a broad term for data representation. Examples of relational plots in Seaborn are `relplot` `lineplot` and `scatterplot`.

It is simple to create a relational plot with Seaborn:

```
sns.relplot(x=months, y=photos)
plt.xlabel("Month of the year")
plt.ylabel("Amount of photos taken")
plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 1.

Fig. 1. Line Plot Created from User Spotify Data.

4.3.4 Distribution Plots

A distribution plot shows how the data is concentrated in a range of values. The graph that appears looks similar to a bar graph in that there are bars. However, these bars show the concentration of a variable across a range of values rather than the quantity possessed by a singular variable. The distributional plots in Seaborn are `displot` `histplot` `kdeplot` `ecdfplot` and `rugplot`.

```
sns.displot(x=source, y=value)
plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 2.

Fig. 2. Distribution Plot Created from User Spotify Data

4.3.5 Categorical Plots

Categorical plots are statistical graphs that help visualize the magnitudes of different variables in a dataset. A type of categorical plot is a bar chart, exactly like the example produced in the Matplotlib section. The categorical plots are `catplot` `stripplot` `swarmplot` `boxplot` `violinplot` `boxenplot` `pointplot` `barplot` and `countplot`.

Categorical plots are relatively simple to implement. If using the `catplot` method, it is necessary to include the `kind` parameter.

```
sns.barplot(x=source, y=value)
plt.show()
```

This program can be downloaded from GitHub

The output from the program is showcased in Fig. 3.

Fig. 3. Categorical Plot Created from User Spotify Data.

4.3.6 Regression Plots

Regression plots are like relational plots in the way that they help visualize the relationship between two variables. Regression plots, however, show the linear correlation that may or may not exist in a scatter plot of data. Their regression plots are `lmplot` `regplot` and `residplot`.

Regression plots are simple to implement:

```
sns.regplot(x=months, y=photos)
plt.show()
```

This program can be downloaded from GitHub . The output of this program is showcased in Fig. 4.

Fig. 4. Regression Plot Created from User Spotify Data

Each of these plots can be manipulated to the users needs via the API that is listed in the sources section.

4.3.7 Saving Figures

Saving figures created by Seaborn is quite simple. This is because it is the exact same as in Matplotlib.

To save a figure:

```
plt.savefig('figure_path/figure_name')
```

This program can be downloaded from GitHub

4.3.8 Links

- https://seaborn.pydata.org/api.html
- https://www.geeksforgeeks.org/python-seaborn-tutorial/
- https://www.geeksforgeeks.org/introduction-to-seaborn-python/
- https://www.geeksforgeeks.org/plotting-graph-using-seaborn-python/
- https://stackoverflow.com/questions/30336324/seaborn-load-dataset
- https://github.com/mwaskom/seaborn-data/blob/master/planets.csv

4.4 Bokeh

Bokeh is a Python library useful for generating visualizations for web browsers. It generates graphics for all types of plots and dashboards powered by JavaScript without the user's need to write any JavaScript code. The guide below will walk you through useful Bokeh commands and features.

4.4.1 Installation

To install Bokeh, please use the command:

```
$ pip install bokeh
```

4.4.2 Import Statements

To plot figures, we import the `show` and `figure` functions from the Bokeh libraries.

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

4.4.3 Bokeh Plotting Interface

`bokeh.plotting` is the library's main interface. It gives the ability to generate plots easily by providing parameters such as axes, grids, and labels. The following code shows some of the simplest examples of plotting a line and a point on a chart.

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

# labeling the title, specifying the range of the x-axis, labeling the
# y-axis, specifying the height to be 500 pxls

p = figure(title="My Graph", x_range=[0, 20], y_axis_label="the y axis",
           height=500)

# plotting a line from (0,0) to (20,20); any of the CSS colors can be
# used

p.line([0, 20], [0, 20], color='indigo')

# plotting a point (circle) at (5,10)
p.circle(5, 10, color='green')

show(p)
```

This program can be downloaded from GitHub . The output is shown in Fig. 1.

Fig. 1. Line and Point Plotted on a Chart

4.4.4 Figure Parameters Example

- **x_axis_label** and **y_axis_label**: labels for the x and y axis
- **x_range** and **y_range**: specifications for the range of the x
- **and y axis title**: text title for your graph width** and
- ****height**: width and height of your graph in pixels
- **background_fill_color**: the background of the figure (takes any
- **CSS colors)

4.4.5 Scatter Plot

The Bokeh library provides various marker shapes for marking points on the scatter plot. The example below demonstrates how to create a scatter plot with two points at locations (1,3) and (2,4) respectively with circular and square marker shapes. The size parameter controls the size of the marker.

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

p = figure(title="Scatter Plot")

# Circle
p.circle([0, 3], [4, 5], size=10)

# Square
p.square([1, 2], [3, 4], size=10)

show(p)
```

This program can be downloaded from GitHub. The output is shown in Fig. 2.

Fig. 2. Sample Scatter Plot with Various Point Shapes

The list possible marker types and the functions used to create them can be found here.

4.4.6 Line Plots

The library provides a series of functions for creating various types of line graphs ranging from a single line graph, step line graph, stacked line graph, multiple line graph, and so on.

```
from bokeh.io import show, export_png, export_svg
from bokeh.plotting import figure
import random

x = []
y = []
for i in range(0, 100):
  x.append(i)
  value = random.random() * 100
  y.append(value)

p = figure(title="Plot Test", x_axis_label="x", y_axis_label="y")
p.line(x, y)

show(p)
```

This program can be downloaded from GitHub The output is shown in Fig. 3.

Fig. 3. Sample Randomized Line Plot

You can find the source code for other types of line plots here: http://docs.bokeh.org/en/latest/docs/user_guide/plotting.html

4.4.7 Bar Chart

Similarly, the `hbar()` and `vbar()` functions can be used to display horizontal and vertical bar graphs, respectively.

```
from bokeh.io import show, export_png, export_svg
from bokeh.plotting import figure

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
x = list(data.keys())
y = list(data.values())

p = figure(x_range=x, title="Bar Chart")

p.vbar(x=x, top=y, line_color='black', color='orange', width=0.9, line_width=2)

show(p)
```

This program can be downloaded from GitHub . The output is shown in Fig. 4.

Fig. 4. Spotify Data Shown in Bar Chart

4.4.8 Saving Figures

Bokeh supports outputs to a static HTML file with a specific name.

```
from bokeh.plotting import output_file

output_file("name.html")
```

After importing the Bokeh plotting interface, it is possible to be able to create different types of plots utilizing the figure created with the figure function.

4.4.8.1 Saving Figures as PNG As the purpose of Bokeh is to create interactive `.html` visualizations, it's recommended to keep your visualizations in this format. However, it may sometime be necessary to save as an image file.

In order to save figures as a PNG, both Selenium and a web driver will need to be installed. We will use Chromium here for our web driver. To install both at once, use the commands:

(Windows)

```
$ pip install selenium chromedriver-binary
$ pip install chromedriver-binary-auto
```

There seems to be issues installing `chromedriver-binary` on Mac computers due to the built-in security, so it is recommended to simply save Bokeh figures as a `.html` file.

When writing a program, Chromium must be added to the PATH through these import statements:

```
from selenium import webdriver
import chromedriver_binary
```

Bokeh appears to support saving files as a `.svg` but it seems to have bugs and is not recommended. To use the functions, `export_png()` and `export_svg()` must be imported, and can be used as follows:

```
from bokeh.io import export_png, export_svg

export_png(fig, filename="file-name.png")
export_svg(fig, filename="file-name.svg")
```

Note that Chromium is slow and this process may take delay the execution and performance of the program.

Similarly to Matplotlib, Bokeh can utilize a function to save all created images.

```
from matplotlib import pyplot as plt
from bokeh.io import export_png, export_svg
import os


def save(p):
  name = os.path.basename(__file__).replace(".py", "")
  export_png(p, filename=f"images/{name}.png")
  export_svg(p, filename=f"images/{name}.svg")
  plt.show(p)
```

This code can be accessed on GitHub .

4.4.9 Links

- http://docs.bokeh.org/en/latest/docs/user_guide/plotting.html
- http://docs.bokeh.org/en/latest/docs/user_guide/plotting.html
- http://docs.bokeh.org/en/latest/
- https://docs.bokeh.org/en/latest/docs/reference/plotting/figure.html
- https://docs.bokeh.org/en/latest/docs/user_guide/export.html

4.5 Pandas Graphics

Pandas is a useful library for working with data. It relies on storing data in data frames. Instead of using a set of ordered pairs, a data frame in Pandas is similar to an Excel Spreadsheet or an SQL data frame and supports multiple rows and columns in a tabular fashion.

Visualization for Panda's data frames are an important tool for understanding the data set. Panda's visualization tools are based off of Matplotlib, so many of the plotting functions are the same.

4.5.1 Installation

To install Pandas, please use the command:

```
$ pip install pandas
```

4.5.2 Import Statements

The user will need to import both Pandas and Matplotlib in order to create and visualize data frames. In addition, Python's `numpy` may be a useful library for mathematical procedures on the data.

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

4.5.3 Bar Chart

Creating a bar chart with data frames is similar to creating bar charts with Matplotlib, with a couple differences in how you manipulate the data. In the following program, we use the same data and modifications as the Matplotlib bar chart example that can be found on Github.

```
import matplotlib.pyplot as plt
import pandas as pd

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
categories = data.keys()
count = data.values()

df = pd.DataFrame({'Count':count, 'Categories':categories})

# Creating the bar chart
df.plot.bar(
```

```
        x='Categories',
        y='Count',
        align='center',
        color='orange',
        width=0.9,
        edgecolor="black",
        linewidth=2)

# Editing the bar chart's title, x, and y axes
plt.xlabel("Genre of Music")
plt.ylabel("Number of songs in the genre")
plt.title("Distribution of Genres in My Liked Songs")
plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 1.

Fig. 1. Spotify Data Shown in Bar Chart

Note the differences in creating the chart. Since data frames support multiple dimensions of data, the x and y we want to graph must be specified in `df.plot.bar()`. However, editing the title and axes are the same as in Matplotlib.

4.5.4 Line Chart

A line chart is typically used for time series data and non-categorical data. Pandas supports line chart visualization with `plot.line()`. We use the same data and modifications as the Matplotlib line chart example that can be found on Github. Note that since this data relies on random number generation the graphs will look slightly different each time.

```
import matplotlib.pyplot as plt
import pandas as pd
import random

x = []
y = []
for i in range(0, 100):
    x.append(i)
    value = random.random() * 100
    y.append(value)


df = pd.DataFrame({'x':x, 'y':y})

# creating the plot and labeling axes and title
df.plot.line(x='x', y='y')
```

```
plt.ylabel("y")
plt.title("Plot Test")

plt.show()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 2.

Fig. 2. Sample Randomized Line Plot

4.5.5 Pie Chart

A pie chart is useful for showing a divison of a whole. Data that can be represented by a pie chart can also be used to make a bar chart, since both typically use categorical counts. Pandas uses `plot.pie()` to make a pie chart, in a manner similar to `plot.bar()`. We use the same data used to create the line chart for this visualization.

```
import matplotlib.pyplot as plt
import pandas as pd

data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz': 25}
categories = data.keys()
count = data.values()

df = pd.DataFrame({'Count':count},index=categories)
plot = df.plot.pie(y='Count',legend=None)
save()
```

This program can be downloaded from GitHub. The output of this program is showcased in Fig. 3.

Fig. 3. Genres of Music from Spotify Distributed on a Pie Chart

Note that instead of listing both the Categories and the Count as data, we use the categories as index. This gets the proper labeling for our pie chart. In addition, Pandas automatically adds a legend, but this is unnecessary so we can remove the legend by setting the parameter `legend=None` in `plot.pie()`.

4.5.6 Exporting

Note that this is the same as the Matplotlib tutorial found here on Github.

To export your graph as an image file, you can use the Matplotlib function `savefig("fname.x")`. You can specify the file type by filling in `.x` with `.pdf`, `.png`, `svg`, etc.

The parameter `dpi` sets the DPI (Dots per Inch) of the image being saved. Specify this number in the form of a float. For example, set `dpi=300`.

Additionally, there is another way to save files that may be faster than calling a specific method for each file. The following code showcases this:

```
import matplotlib.pyplot as plt
import os
from matplotlib import pyplot


def save():
    name = os.path.basename(__file__).replace(".py", "")
    plt.savefig(f'/filepath/{name}.png')
    plt.savefig(f'filepath/{name}.pdf')
    plt.savefig(f'filepath/{name}.svg')
    plt.show()
```

This code can be accessed on GitHub

The very last command is `plt.show()`, as this command displays the graph that you made. To show, simply type:

```
plt.show()
```

4.5.7 Links

- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.bar.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.line.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.pie.html

5 PYTHON

5.1 Cloudmesh StopWatch

Learning Objectives

- Using timers in python

- Using cloudmesh.common.StopWatch
- Create benchmarkes from StopWatch

Improve the documentation of Cloudmesh StopWatch if needed

https://github.com/cloudmesh/cloudmesh-common/blob/main/cloudmesh/common/StopWatch.py

Please be reminded that we could develop a program that read the documentation form the docstring Then we can safe it to a file, so we could autogenerate the md file from the docstring.

5.2 cms sys command generate

Learning Objectives

- Learn how to create your own cloudmesh commands

---

locate in the book how to use cms sys command generate. Generate a command with your username. No commit of this is necessary, but we need to make sure you understand how to create a command.

5.2.1 Creating CLI commands using the cloudmesh command

Command line interface commands essentially allow a user to execute commands that have been programmed in `python` from the command line (i.e. on a `macOS` terminal). In the `cloudmesh` project, it is easy to do this, as there is a built-in module that allows a user to develop this command. We assume that the user has properly set up cloudmesh on their own device. The following showcases the `sys command generate`:

- Execute `cms sys command generate` in the home directory of the device. This essentially calls a function within cloudmesh cmd5 that allows

- Once you have done this, it is necessary to `cd` to that directory. For instance, if you typed: `cms sys command generate apples`, then that would have created a command within the directory you were in. Thus, you need to change directory and navigate to that new directory. Then, it is necessary to run `python setup.py` in your command line. Then execute `pip install .` This will configure your device to have the proper requirements for the command that you are generating.
- After this, you can create python scripts within the directory that can be called as an actual command. You will notice a directory called `cloudmesh` –> `"your command"` –> `command` –> and then finally, `"yourcommand".py`. In this final `python` file, you can edit and create what it is you were intending on creating.
- When you add new arguments to the command, you have to add those arguments to the `arguments parser` function. This ensures that the parser correctly separates all arguments of the program.
- Finally, in the `if` statements at the end of the program, it is necessary to indicate what arguments/options are specified and what should be done if these commands + arguments + options are called.

Important Sidebar

The generate command function relies on an API base called `docopts` which is a framework for creating a command line interface. What this does in essence is it creates command line commands based on the framework of a python script.

In this framework there are a few parts to understand: * Usage- indicate how the commands would be called, specifically. * Arguments- indicates what can be passed into the commands to make them work in a specific way. * Options- indicates the options that can be specified. These are optional.

It is necessary to update the arguments in the corresponding if statement.

5.2.1.1 Example of the CLI Creation

```
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 ""
```

This example can be found here in section **6.0.7.1.4 Create your own Extension**.

Again, all of this relies on having cloudmesh properly downloaded and installed.

5.2.1.2 Example of CLI meshing with a python script

CLI Implementation:

```
    def do_cc(self, args, arguments):
        """
        ::

          Usage:
                cc upload --data=FILENAME
                cc update --data=FILENAME
                cc delete --data=FILENAME
                cc create --queues=QUEUES --database=DATABASE
                cc add --queue=QUEUE --job=JOB --command=COMMAND
                cc run --queue=QUEUE --scheduler=SCHEDULER
                cc remove --queue=QUEUE --job=JOB
                cc remove --queue=QUEUE
                cc list --queue=QUEUE
                cc start
                cc stop
                cc doc
                cc test
                cc temperature

. . .
```

This code can be found here

Part of the code implementation of the above CLI:

```
class Queues:
    """
    The Queues data structure is a structure that holds all of the queues
    with their corresponding names. It is a meta-queue, essentially. The queues
    class will be a dictionary of dictionaries of jobs, which are
    job names and commands.
```

```
    An example command would likely look like:
        cms cc queues list --queues=ab
    """

    def __init__(self, database='yamldb'):
        """
        Initializes the giant queue structure.
        Default database is yamldb
        :param name: name of the structure
        :return: creates the queues structure
        """
        if database.lower() == 'yamldb':
            from cloudmesh.cc.db.yamldb.database import Database as QueueDB
            self.filename = path_expand("~/.cloudmesh/queue/queue")

        elif database.lower() == 'shelve':
            from cloudmesh.cc.db.shelve.database import Database as QueueDB
            self.filename = path_expand("~/.cloudmesh/queue/queue")

        else:
            raise ValueError("This database is not supported for Queues, please fix.
                ↪ ")
```

This code can be found on GitHub

5.3 Linux

Learning Objectives

- Learn how to use the most basic Linux commands

The book has some introduction material to linux, please contribute to the book. Make sure you do not duplicate wht others have done or are doing, coordinate. Create a pull riquest with your contribution.

5.3.1 Introdutyion to Linux

Please read the folloing book and contribute to it.

- https://cloudmesh-community.github.io/pub/vonLaszewski-linux.pdf

5.3.2 Github

PLease read teh folloing tutorials and improve if needed

- https://cybertraining-dsc.github.io/docs/tutorial/git/git-pull-request/
- https://cybertraining-dsc.github.io/docs/tutorial/git/git-ssh/
- https://cybertraining-dsc.github.io/docs/tutorial/git/git-gh/

5.4 Python

5.4.1 Queue

Learning Objectives

- Learn how to use the build in Python queue

A queue is a data structure. Essentially, it is a list that has order, meaning that the queue has an object to be removed first and an object to be removed last.

There are three types of queues: FIFO (first in first out), LIFO (last in first out), and Priority Queue (order is based on the data's individual priority score).

```
import queue
```

Then, the data structures can be built as follows:

5.4.1.1 FIFO Queue A FIFO queue is the most basic type of queue. Simply put, the values that go into the data structure first are the values that comes out of the data structure first.

FIFO queues are easy to implement:

```
# this queue is a FIFO queue
fq = queue.Queue()

for x in range(0, 10):
    fq.put(x)
```

```
while not fq.empty():
    print(fq.get())
```

This code can be accessed on GitHub

5.4.1.2 LIFO Queue A LIFO queue is another very basic type of queue. Simply put, it is a queue that operates the last item inserted being the first item that is removed.

LIFO queues are simple to implement:

```
# this queue is a LIFO queue

lq = queue.LifoQueue()

for x in range(0, 10):
    lq.put(x)
```

This code can be accesed on GitHub

5.4.1.3 Priority Queue Finally, priority queues are a more complex queue type. Priority queues work by inserting elements with a sorted priority. For example, if an 8 is inserted before a 2, the 2 will be removed first.

Priority queues are easily implemented:

```
# this queue is a Priority queue

pq = queue.PriorityQueue()

for x in range(0, 10):
    value = int(random.random() * 100)
    print('number being inserted into the priority queue', value)
    pq.put(value)

print()

while not pq.empty():
    print('number being removed from the pq', pq.get())
```

This code can be accessed on GitHub

All specifics on queues can be researched at the below links:

- https://pymotw.com/3/queue/index.html

- https://www.geeksforgeeks.org/priority-queue-in-python/
- https://docs.python.org/3/library/queue.html

5.5 glob

`glob` is a small module that searches for files by reading the patterns of filenames. However, `glob` doesn't work in the same way regular expressions do as they follow standard Unix path expansion rules.

5.5.1 Glob with asterisk

The example showcases different states under a single directory and counties under subdirectories. The files consist of `.txt` files consisting of different versions of a program under a single directory. Let us first create some file sin a temporary directory:

```
$ cd
$ mkdir tmp/subdir
$ touch tmp/a.txt
$ touch tmp/a-1.txt
$ touch tmp/b.txt
$ touch tmp/b+.txt
$ touch tmp/subdir/c.txt
```

Now let us create the following file file in the home directory. To list all the file in the tmp directory you can use the asterisk '*':

```
import glob

for name in sorted(glob.glob('tmp/*')):
    print(name)
```

It will list the files in the directory tmp in alphabetical order.

```
tmp/a.txt
tmp/a-1.txt
tmp/b.txt
tmp/b+.txt
```

5.5.2 Single Character Wildcard (?)

A question mark (?) can be used to search for files with the same pattern of names through singling out one character as a wildcard. This can be shown in this example.

```
import glob

for name in sorted(glob.glob('tmp/a-?.txt')):
    print(name)
```

This program lists the files starting with `a-` , a single character and as prefix `.txt`

```
tmp/a-1.txt
```

5.5.3 Escape Characters

`glob` can also search for files that contain a special characters through using the command `glob.escape(char)` . This can be shown in this example.

```
import glob

specials = '!+('

for char in specials:
    pattern = 'tmp/*' + glob.escape(char) + '.txt'
    for name in sorted(glob.glob(pattern)):
        print(name)
```

The output shown here is a every file that specifically contains the characters `!` , `+` , or `(` . There is only file that does so which is:

```
tm/b+.txt
```

5.5.4 Subdirectories

Not only `glob` can search for files recursively in directries with the `**` query

```
import glob


for name in sorted(glob.glob('tmp/**')):
    print(name)
```

In this example, will produce

```
tmp/subdir
tmp/a.txt
tmp/a-1.txt
tmp/b.txt
tmp/b+.txt
tmp/subdir/c.txt
```

5.5.5 Lins

- https://pymotw.com/3/glob/index.html

5.6 Mmap

`mmap` standards for memory-map files. Memory-mapping a file involves accessing files directly without the use of traditional I/O functions.

5.6.1 Reading

The example shown here is a short, simple story that is contained in a `.txt` file. The file can be accessed here as `example.txt`.

```
Let's play with the dog, it's really nice out today!
```

First, the actual file should be opened using the `open` command with the parameter `'r'` for reading and is indicated with the header `f`.

After that, memory-map file can be created using the command `mmap.mmap()` and can be indicated with the header `m`. Within the parentheses `()`, various arguments should be made.

The first argument should be `f.fileno()`, a file descriptor that opens and closes the `mmap` file.

The second argument is the size in bytes, in the form of a float, of the portion of the file to map. If it's `0`, like in this example, the entire file is mapped.

The third argument, which is optional, is the accessibility settings. In this example, it's set to read-only through `access=mmap.ACCESS_READ`.

The code in this example will read various parts of `story.txt` both progressively and through slices.

```
import mmap
```

```
with open('example.txt', 'r') as f:
    with mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ) as m:
        # Reads the first ten characters
        print('Char. 1-10 (Read) :', m.read(10))

        # Reads a slice of characters
        print('Char. 1-10 (Slice):', m[5:14])

        # Reads the next ten characters
        print('Char. 11-20 (Read) :', m.read(10))
```

The following output is produced:

```
Char. 1-10 (Read) : b"Let's play"
Char. 1-10 (Slice): b' play wit'
Char. 11-20 (Read) : b' with the '
```

5.6.2 Writing

In order to modify files, do the same procedure as reading files by opening the actual file with the `open` command; however, this time, use the parameter `r+` instead of `r`.

Then, create the `mmap` file with `mmap.mmap` with the same required arguments. The optional argument set to the default access mode of `access=mmap.ACCESS_WRITE` in this case.

After those commands, edits can then be made to the `mmap` file as shown in the following example.

```
import mmap
import shutil

# Copy the example file
shutil.copyfile('example.txt', 'example_copy.txt')

word = b'dog'

with open('example_copy.txt', 'r+') as f:
    with mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_WRITE) as m:

        # Memory-map file before change

        print('Memory Before:\n{}'.format(m.readline().rstrip()))
        m.seek(0)  # rewind

        loc = m.find(word)
        m[loc:loc + len(word)] = b'cat'
```

```
        m.flush()

        # Memory-map file after change

        m.seek(0)  # rewind
        print('Memory After:\n{}'.format(m.readline().rstrip()))

        # Actual file after change

        f.seek(0)  # rewind
        print('File After:\n{}'.format(f.readline().rstrip()))
```

The following output shows that this access mode allows the modification of the actual file.

```
Memory Before:
b"Let's play with the dog, it's really nice out today!"
Memory After:
b"Let's play with the cat, it's really nice out today!"
File After:
Let's play with the cat, it's really nice out today!
```

This can be changed by setting the access mode to `access=mmap.ACCESS_COPY` as shown in this example.

```
import mmap
import shutil

# Copy the example file
shutil.copyfile('example.txt', 'example_copy.txt')

word = b'dog'

# Changing access settings

with open('example_copy.txt', 'r+') as f:
    with mmap.mmap(f.fileno(), 0,
                   access=mmap.ACCESS_COPY) as m:

        # Memory-map file before change

        print('Memory Before:\n{}'.format(m.readline().rstrip()))

        # Actual file before change

        print('File Before:\n{}'.format(f.readline().rstrip()))
```

```
        m.seek(0)  # rewind
        loc = m.find(word)
        m[loc:loc + len(word)] = b'cat'

        # Memory-map file after change

        m.seek(0)  # rewind
        print('Memory After:\n{}'.format(m.readline().rstrip()))

        # Actual file after change

        f.seek(0)
        print('File After:\n{}'.format(f.readline().rstrip()))
```

The following output shows that only the `mmap` file and not the actual file was modified in the end.

```
Memory Before:
b"Let's play with the dog, it's really nice out today!"
File Before:
Let's play with the dog, it's really nice out today!
Memory After:
b"Let's play with the cat, it's really nice out today!"
File After:
Let's play with the dog, it's really nice out today!
```

5.6.3 Links

- https://pymotw.com/3/mmap/index.html

5.7 Pickle

`pickle` is a module that turns Python objects into series of bytes that can be transmitted, stored, or reconstructed.

5.7.1 Encoding Data

A data structure can be encoded into a string by using the command `pickle.dumps(data)`. In this example, a dictionary is being encoded.

```
import pickle

# Creating dictionary of data
```

```
temperatures = {
    'red': 50,
    'blue': 30,
    'yellow': 20,
}
print('Temperatures:', temperatures)

# Pickling the data
pickle_temperatures = pickle.dumps(temperatures)
print('Pickle:', pickle_temperatures)
```

This following output is produced:

```
temperatures: [{'Red': 50, 'Blue': 30, 'Yellow': 20}]
Pickle: b' ... A binary string that we ommitted here ... .'
```

5.7.2 Decoding Data

The encoded data can then be decoded using the command `pickle.loads(data)`:

```
import pickle

# Creating dictionary of data
temperatures_0 = {'red': 50, 'blue': 30, 'yellow': 20}]
print('Initial temperatures:', temperatures_0)

# Encoding the data
pickle_temperatures_0 = pickle.dumps(temperatures_0)

# Decoding the data
temperatures_1 = pickle.loads(pickle_temperatures_0)
print('From pickle database:', temperatures_1)

# Checking authenticity
print("Same:", temperatures1 is temperatures2)
print("Equal:", temperatures1 == temperatures2)
```

This can be shown in the following output:

```
Initial temperatures: [{'Red': 5, 'Blue': 3, 'Yellow': 2}]
From pickle database: [{'Red': 5, 'Blue': 3, 'Yellow': 2}]
```

This command will produce data that is equal to the original data, but it's not the same as shown by the following output:

```
Same: False
Equal: True
```

5.7.3 Links

- https://pymotw.com/3/pickle/index.html

5.8 Shelve

`shelve` is a library that gives users the ability to create, store, modify, and control the accessibility of data without a relational database. A shelf has a string key and data that can be near anything as long as it is supported by `pickle`. This includes integers, dictionaries, and Objects.

5.8.1 Creating a New Shelf

A new shelf can be created using the command: `d = shelve.open(filename)`. Shelve can store objects; you can insert a dictionary, for example, as shown next:

```
import shelve

computers = shelve.open('computers.db')
computers['temperature'] = {
    'red': 80,
    'blue': 40,
    'yellow': 50,
}
```

On a Windows: This creates three files `computers.db.bak`, `computers.db.dat`, and `computers.db.dir`. These `.bak`, `.dat`, and `.dir` files hold the shelf information that can be accessed for future use.

For other computers, this creates the file 'computers.db'.

5.8.2 Accessing a Shelf

After it's been created, it can be accessed while reading objects into variables.

```
import shelve

with shelve.open('computers.db') as computers:
```

```
    t = computers['temperature']

print(t)
```

This produces the following output:

```
{'red': 80, 'blue': 40, 'yellow': 50}
```

5.8.3 Making Shelf Read-Only

The user can also make their data read-only by adding the `flag` parameter as shown:

```
shelve.open('computers.db', flag='r')
```

That way, when a user tries to modify it, it produces an error:

```
import dbm
import shelve

computers = shelve.open('computers.db', flag='r')
print('Temperature:', computers['temperature'])
try:
    computers['temperature']['green'] = 100
except dbm.error as err:
    print('ERROR:', err)
```

The following output is produced:

```
Temperature: {'red': 80, 'blue': 40, 'yellow': 50}
ERROR: The database is opened for reading only
```

5.8.4 Modifying Shelves

In order to modify a shelf, simply open up the shelf again as shown:

```
shelve.open('shelf_name')
```

Make sure to use this before making the modification, or else it won't work. For simple assignments, this will suffice.

However, in most cases, you use the `writeback=True` parameter. This caches the modifications and slows down the saving process. As seen, however, you can now directly access and modify the shelf entries as it was a two-dimensional dictionary.

You can also delete shelf items using their keys with the `del` method.

```
import shelve
from pprint import pprint

computers = shelve.open("computers.db",writeback=True)
print('Initial temperature:')
pprint(computers['temperature'])

computers['temperature']['green'] = 101
del computers['temperature']['yellow']
print()
print('Modified temperature:')
pprint(computers['temperature'])
```

Modifications are preserved as shown in this output:

```
Initial temperature:
{'red': 80, 'blue': 40, 'yellow': 50}

Modified temperature:
{'blue': 40, 'green': 101, 'red': 80}
```

5.8.5 Closing Shelves

Lastly, in order to save the shelf so that it may be accessed next time, use the command '.close()'

```
import shelve

computers = shelve.open('computers.db')
# do your shelf operations here
computers.close()
```

5.8.6 Links

- https://pymotw.com/3/shelve/index.html
- https://docs.python.org/3/library/shelve.html

5.9 Yaml Database (yamldb)

Yamldb is a python package that allows a user to store variables on a computer's hard-drive, as opposed to having variables stored in memory.

Yamldb makes use of the yaml data file, and it stores values in the yaml files so that they can be easy to use. The yaml files can be accessed like python dictionaries, which makes the process significantly easier for the user.

The yamldb package is used in the `cloudmesh-cc` section of the `cloudmesh` repository and can be accessed here.

5.9.1 Installing and importing

Yamldb is easy to install. Simply execute the following command:

```
pip install yamldb
```

Then, to import yamldb to a python file, simply execute the following code:

```
from yamldb import YamlDB
```

Now, following these commands, the computer is set up to run and access yamldb.

5.9.2 Using yamldb

Yamldb has several methods within itself that allow the user easy access to creating a yaml file and storing things within it. These things can be anything, really. The methods include: `Yamldb()`, `.get()`, `.load()`, `.save()`, and `.search()`. All of these functions can be used. The following is example code of how someone might use yamldb for their own code.

```
from yamldb import YamlDB

filename = 'PATH'
database = YamlDB(filename=filename)

database['queue1'] = 'job1'
database['queue2.job1'] = 'echo hello world'

d = database.get('queue1', default=3)
```

This code can be accessed from GitHub.

Which creates:

```
queue1: job1
queue2:
  job1: echo hello world
```

5.9.3 Accessing Values

As one can see, these values are easy to access. Let's say, for instance, that we were trying to access the `queue2:job1` string `"echo hello world"`. To do this, we would execute the following:

```
job = database.get('queue2')
print(job)

command = database['queue2'].get('job1')
print(command)
```

This code can be accessed from GitHub.

Which returns:

```
{'job1': 'echo hello world'}
echo hello world
```

5.9.4 Save, Load, and Search

With the `yamldb` implementation, it is simple to save, load, and search. The following code shows this output.

```
from yamldb import YamlDB

filename = 'PATH'
database = YamlDB(filename=filename)

database['queue1'] = 'job1'
database['queue2.job1'] = 'echo hello world'

d = database.get('queue1', default=3)

database.load(filename)  # loads the file
database.save(filename)  # saves the file
database.search('queue1') # searches the file for the query
```

This code can be accessed from GitHub.

The `.load()` function loads in the current `.yaml` file that was created originally. The `.save()` function saves the any updates that were made to the `.yaml` file.

Outside of that, this should be the fully implementation of the `yamldb` python package.

The overview of this python package was accessed from here.

5.10 Requests with Python

The `requests` library is a python library that allows the user to interact with the internet via HTTP very easily. Typically, users specify the type of request they are making via their programming, but with the `requests` library, it is much easier to program for user enhancement.

5.10.1 Installing and Importing

The `requests` library can be easily installed from the command line and imported at the top of any python file.

The following script can be used to install `requests`:

```
$ python -m pip install requests
```

Once this has been done, it is simple to import the library into a python file. Simply execute the following:

```
import requests
```

5.10.2 Using Requests

The `requests` library has many functions that allow it to be utilized in the way it is. Typical functions are `requests.put()`, `requests.head()`, `requests.delete()`, `requests.get()`, and `requests.options()`. What the `requests` library does is it properly encodes everything that is programmed by the user.

A basic example of how requests could be utilized within a program:

```
import requests
r = requests.get("https://api.github.com/events")
r = requests.post('https://httpbin.org/post', data={'key': 'value'})
r = requests.put('https://httpbin.org/put', data={'key': 'value'})
r = requests.delete('https://httpbin.org/delete')
r = requests.head('https://httpbin.org/get')
r = requests.head('https://httpbin.org/get')
```

This code can be accessed from GitHub

For requests, an `r` value of 200 means that the method was a success whereas a 100 means not a success.

In addition to these functions, the `requests` library has a plethora of other functionalities. It can send more data, convert to `json` files, execute `patches`, etc. The documentation for the `requests` library can be found here. This link is also where the examples are drawn from.

5.11 FastAPI

Learning Objectives

- Learn how to use fastAPI

FastAPI is a Python framework that allows developers set up a REST service and define its functionality with an easy-to-use API.

5.11.1 FastAPI Install

As FastAPI will need a web server, we will use `uvicron` for development purposes. In a production environment other, more mature Web services are recommended. To install FastAPI and uvicorn simply use the command:

```
$ pip install "fastapi[all]"
```

TODO: THERE ARE TOO MANY DIFFERENT EXAMPLES, PLEASE CREATE ONE THAT BUILDS ON TOP OF EACH OTHER, USE COMPUTERS WITH TEMPERATURES

5.11.2 FastAPI Quickstart

One of the simplest FastAPI file looks like this, which we assume is placed in a file called `main.py`:

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"processor": "5950X"}
```

Start the live FastAPI `app` in the `uvicorn` server use the command:

```
$ uvicorn main:app --reload
```

This will yield the output

```
1 INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
2 INFO: Started reloader process [28720]
3 INFO: Started server process [28722]
4 INFO: Waiting for application startup.
4 INFO: Application startup complete.
```

The first line includes information about which URL is used to contact it to obtain a response. An easy way to view it is to enter http://127.0.0.1:8000 in your browser. The JSON response will appear as:

```
{"message": "Hello World"}
```

One of the embedded features of FastAPI is its build in documentation framework based on OpenAPI schema but also alternative formats such as redoc. You can look at it while going with your browser to the URL

- OpenAPI: http://127.0.0.1:8000/docs.
- OpenAPI json: http://127.0.0.1:8000/openapi.json
- Redoc: http://127.0.0.1:8000/redoc

5.11.2.1 Path One of the mechanisms FastAPI provides it to easily specify the URL that is needed to trigger the functionality of the defined function after its definition.

We have seen such an example in `@app.get("/")` which activates the `root` function when the URL of the server is specified followed by "/"

You can add other path's and functions. Let us assume you add to our initial program the function

```
@app.get("/temperature")
async def temperature():
    return {"temperature": 0}
```

The if you use the URL http://127.0.0.1:8000/temperature, we will see

```
{"temperature": 0}
```

5.11.3 Query Parameters

When you declare other function parameters that are not part of the path parameters, they are automatically interpreted as URL "query" parameters.

```
from fastapi import FastAPI

app = FastAPI()
jobs = [{"name": "Foo"},
        {"name": "Bar"},
        {"name": "Baz"}]

@app.get("/jobs/")
async def get_job(skip: int = 0, limit: int = 10):
    return jobs[skip : skip + limit]
```

The query is the set of key-value pairs that go after the `?` in a URL, separated by & characters. For example, in the URL:

```
http://127.0.0.1:8000/jobs/?skip=0&limit=10
```

In this case the query parameters are:

- skip: with a value of 0
- limit: with a value of 10

5.11.3.1 Searching in the fastapi

```
from fastapi import FastAPI

app = FastAPI()
jobs = [{"name": "Foo"},
        {"name": "Bar"},
        {"name": "Baz"}]

@app.get("/job/")
async def search_job(name:str):
    result = None
    for item in jobs:
        if item['name'] == name:
            result = name
    return result
```

For example, in the URL

```
http://127.0.0.1:8000/job/?name=Foo')
```

Output

```
"Foo"
```

5.11.3.2 Running Through Git bash

```
import requests

result = requests.get('http://127.0.0.1:8000/job/?name=Foo')

print(result.text)

print(result.status_code)
print(result.headers['content-type'])
print(result.encoding)
print(result.text)
print(result.json())
```

Run python code on Git bash

```
$ python r.py
```

where r.py is the file name, yeilding in the output

```
"Foo"
200
application/json
utf-8
"Foo"
Foo
```

5.11.4 Running the cc FastAPI service

The cloudmesh cc FastAPI service can be started with

```
$ cms cc start
```

To see the documentation you do not have to type in the URL in the browser, but instead you can use the command

```
$ cms cc doc
```

which will open the url `http://127.0.0.1:8000/docs` in the browser.

To stop the server, use the command

```
$ cms cc stop
```

5.11.5 Testing

In order to test the FastAPI code, we want to create a new test file. In this file, we import `TestClient from fastapi.testclient`. TestClient creates a test object that follows pytest conventions. In addition, you must import the `app` FastAPI object from the main module.

You can then create the test object by passing this FastAPI object into the TestClient. You use assert statements to test for validity.

```
from fastapi.testclient import TestClient
from .main import app

client = TestClient(app)

def test_temperature():
    response = client.get("/temperature")
    assert response.status_code == 200
    assert response.json() == {"temperature": 0}
```

You can create several test functions and run them with `pytest`. Lastly, there are other parameters for `client.get()` that may be further explored in the requests documentation.

5.11.5.1 Asynchronous Testing To test asynchronous functions, for example async queries, we can no longer use TestClient due to Pytest's inherent sync nature. Instead, we use a very similar client called `HTTPX` that can make both synchronous and asynchronous requests.

In addition, we must mark these async test functions with the `pytest.mark.aniyo` flag.

```
import pytest
from httpx import AsyncClient

from .main import app


@pytest.mark.anyio
async def test_temperature():
    async with AsyncClient(app=app, base_url="http://test") as ac:
        response = await ac.get("/temperature")
    assert response.status_code == 200
    assert response.json() == {"temperature": 0}
```

The `await` functions sends the asynchronous request.

This can once again be run with pytest, and additional async or sync test functions can be added as well.

5.11.6 Links

- https://fastapi.tiangolo.com/tutorial/first-steps/
- https://fastapi.tiangolo.com/tutorial/testing/#using-testclient
- https://fastapi.tiangolo.com/advanced/async-tests/
- cloudmesh cc TODO

6 GRAPHS WITH PYTHON

6.1 Networkx

`networkx` is a python library that allows a user to create graphs. The library allows the user to create different kinds of graphs, included directed and undirected. With this in mind, the following is a basic overview of the library.

6.1.1 Installing and Importing

Thankfully, `networkx` can be super easily downloaded. Simply execute the following:

```
$ pip install networkx
```

6.1.2 Creating a graph

It is super simple to create a graph, add nodes to the graph, and add edges to the graph. For the purposes of this tutorial, we will be working with a synthetic job queueing service. Each node represents a job and the edges are the edges that connect each job.

The following is the code that was created:

```
import networkx as nx

# creating a list of nodes and edges- you have to be extremely careful not to typo
nodes = ['job-1', 'job-2', 'job-3','job-4', 'job-5']
```

```
edges = [('job-1', 'job-2'), ('job-2', 'job-3'), ('job-3', 'job-4'),
         ('job-4', 'job-5')]
```

This code can be accessed from GitHub

To create the graph and add the above nodes and edges, `networkx` provides super simple methods.

```
1 import networkx as nx
2
3 graph = nx.Graph()  # creates the graph that we will use
4
5 graph.add_nodes_from(nodes)  # adds the nodes from the dict we created
6 graph.add_edges_from(edges)  # adds the edges from the dict we created
7
8 print(graph)  # prints: "Graph with 5 nodes and 4 edges"
```

This code can be accessed from GitHub

Thus, we have created the graph.

6.1.3 Accessing

It is necessary to be able to access the elements within the graph. There is a slightly strange way of going about doing this with `networkx`. Rather than being able to directly access the element you input, it will return the adjacent element. Thus, it seems to be necessary to have a dummy head, which makes it so that the first node inserted into the graph can be accessed.

The following looks into this:

```
9 print(graph['job-1'])  # will return {'job-2' : {}}
```

This code can be accessed from GitHub

6.1.4 Removing nodes

It is very easy to remove nodes and edges using `networkx`. This can be accomplished by executing the following:

```
import networkx as nx

G = nx.Graph()

# creating a list of nodes and edges- you have to be extremely careful not to typo
nodes = ['job-1', 'job-2', 'job-3', 'job-4', 'job-5']
```

```
edges = [('job-1', 'job-2'), ('job-2', 'job-3'), ('job-3', 'job-4'),
         ('job-4', 'job-5')]

print(G)  # will return "Graph with 5 nodes and 4 edges".

G.remove('job-1')

print(G)  # will return "Graph with 4 nodes and 3 edges"
```

This code can be accessed on GitHub

6.1.5 Colors and Labels

It is super simple to add colors and labels, as well as display the networks in a graphic notation. To do so, execute the following code:

```
color_map = []
for n in nodes:
    color_map.append('lightblue')

nx.draw(G, node_color=color_map, with_labels=True)
plt.show()
```

This code can be accessed on GitHub

The `color_map` was created as a list based on the number of nodes in our list of nodes we created before. Thus, it makes it very easy to implement as it allows the user to update individual nodes, if needed.

The code produces the following image:

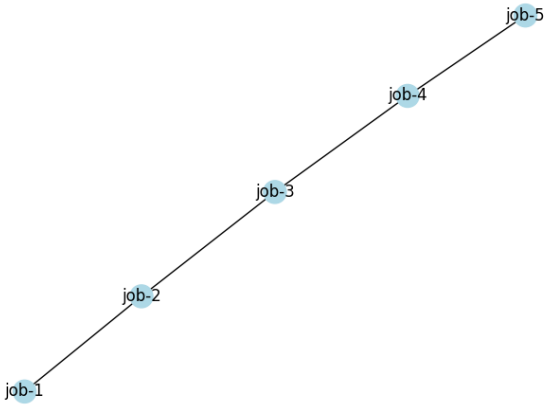


Figure 1: network-image created with networkx python package

There are many other ways to manipulate the nodes and edges. These ways can be accessed from the official documentation

6.2 Graphviz

`graphviz` is a library that can be installed on Python which renders graphs from the DOT languages, allowing for visualizations of data structures using undirected or directed graphs consisting of nodes and edges of various shapes and colors.

6.2.1 Installation and Importing

In order to install `graphviz` on Windows fully, first, have Chocolatey installed. Next, run Command Prompt as an administrator and type in the following:

```
choco install graphviz
```

Next, go onto GitBash and type in the following:

```
$ pip install graphviz
```

6.2.2 Creating the graph, adding nodes, and adding edges

Creating a basic graph with nodes and edges is very simple using `graphviz`. The following example is a synthetic job queueing service. Each node represents a job and the edges connect the jobs.

There are two types that can be made using `graphviz`. Regular graphs can be made using `Graph()`. They don't have arrows. Directed graphs can be made using `Digraph()`. They do have arrows.

Nodes can be created using `node()` where the variable of the job can be defined and labeled.

Edges can be created either using `edge()` or `edges()`, as used in this example. The commands take in the start and end node variables which will create either one or multiple edges, respectively. Edges can be labeled too.

The following is the code that was created:

```
import graphviz

f = graphviz.Graph('jobs in queues', filename='examples/basic-graphviz.gv')

f.node('job-1', 'ls')
f.node('job-2', 'echo hello world')
f.node('job-3', 'cd ~')
f.node('job-4', 'cd cm/cloudmesh-cc')
f.node('job-5', 'pytest tests')
f.edges([('job-1', 'job-2'), ('job-2', 'job-3'), ('job-3', 'job-4'),
         ('job-4', 'job-5')], )

f.view()
```

This code can be accessed via Github.

Shown here in Fig. 1. is the graph produced from the code:

Fig. 1. Sequence of Bash Command Path

6.2.3 Subgraphs

Subgraphs are clusters of nodes and edges that can be created using the `subgraph()` command. However, it's required to have the prefix `'cluster'` in the name of it.

The following code shows the usage of subgraphs by expanding on the last example.

```
import graphviz

g = graphviz.Digraph('jobs in queues', filename='subgraph-graphviz.gv')

with g.subgraph(name='cluster_1') as s:
    s.node('job-1', 'ls')
    s.node('job-2', 'echo hello world')
    s.node('job-3', 'cd')
```

```
    s.node('job-4', 'cd cm/cloudmesh-cc')
    s.node('job-5', 'pytest tests')
    s.edges([('job-1', 'job-2'), ('job-2', 'job-3'), ('job-3', 'job-4'),
             ('job-4', 'job-5')], )

with g.subgraph(name='cluster_2') as s:
    s.node('job-6', 'cd')
    s.node('job-7', 'cd cm')
    s.node('job-8', 'cd cm/cloudmesh-alex')
    s.node('job-9', 'git status')
    s.node('job-10', 'git pull')
    s.edges([('job-6', 'job-7'), ('job-7', 'job-8'), ('job-8', 'job-9'),
             ('job-9', 'job-10')], )

g.edge('start', 'job-1')
g.edge('start', 'job-6')
g.edge('job-5', 'end')
g.edge('job-10', 'end')
g.edge('job-3', 'job-7')
g.edge('job-4', 'job-9')

g.node('start', shape='square')
g.node('end', shape='square')

g.view()
```

This code can be accessed via Github.

Shown here in Fig. 2. are the subgraphs produced by the code.

Fig. 2. Sequence of Two Bash Different Command Paths

6.2.4 Data Structures

Rectangular data structures can be created in `graphviz` when the shape of the nodes is set to `'record'`. This specific type of data structure allows for nodes to be clustered together in the same rectangle. The following code shows a diagram of different files and directories.

```
import graphviz

s = graphviz.Digraph('files in directories', filename='structure-graphviz.gv')
s.node_attr={'shape' : 'record'}

s.node('s1', '<d1> cloudmesh-cc | <d2> workflow')
s.node('s2', '{<d1> cloudmesh | <d2> tests}')
```

```
s.node('s3', '<d1> contribute | {<d2> graphs |{<d3> graphviz | <d4> networkx}}')

s.edges([('s1:d1', 's2:d2'), ('s1:d2', 's3:d4')])

s.view()
```

This code can be accessed via Github.

Shown here in Fig. 3. are the data structures produced by the code.

Fig. 3. Data Structure of Various Directories and Files

6.2.5 Colors and Labels

It is super simple to add colors and labels to graphs, nodes, and edges in `graphviz`.

In terms of labels, just add the parameter `label=''` inside the `attr()`, `node()`, or `edge()` commands. The font color of the label can be set using `fontcolor=`.

In terms of color, just add the parameter `color=''` inside the `attr()` or `node()` commands. This will change the color of the perimeter. In order to fill, use the parameter `style='filled'` or set the fill color using `fillcolor=''`.

Gradients can also be added by setting a colon `:` between two different colors. Furthermore, the angle of the gradient can be set using the parameter `gradientangle=''`.

The following code demonstrates many of the features explained.

```
import graphviz

g = graphviz.Digraph('Colors', filename='color-graphviz.gv')
g.attr(bgcolor='red:pink', label='Red Graph', fontcolor='white')

with g.subgraph(name='cluster') as c:
    c.attr(color='cyan', style='filled', label='Cyan Cluster',
           fontcolor='white')
    c.node('n1', 'Orange Node', shape='circle', fillcolor='red:yellow',
           style='filled', gradientangle='90')
    c.node('n2', 'Yellow Node', shape='diamond', color='yellow', style='filled')
    c.edge('n2', 'n1', label='Edge 1')

g.view()
```

This code can be accessed via GitHub.

Shown here in Fig. 4. is what the code produced.

Fig. 4. Label Directed Graph with Various Colored Nodes

6.2.6 Sources

- https://pypi.org/project/graphviz/
- https://graphviz.readthedocs.io/en/stable/examples.html
- https://graphviz.readthedocs.io/en/stable/api.html

7 RIVANNA

7.1 Rivanna

Learning Objectives

- Learn how to use Rivanna
- Learn how to set up VPN to access Rivanna from commandline
- Learn how to access Rivanna from a terminal including Windows

---

Presentation:

- Knuuti
- UVA Rivanna presentation, June 8th, 2022

Logging in to Rivanna via web interface

Documentation: https://www.rc.virginia.edu/userinfo/rivanna/login/#web-based-access

Login: https://rivanna-portal.hpc.virginia.edu/

UVA VPN: https://in.virginia.edu/vpn

Shell access: https://rivanna-portal.hpc.virginia.edu/pun/sys/shell/ssh/rivanna.hpc.virginia.edu

JupyterLab: https://rivanna-portal.hpc.virginia.edu/pun/sys/dashboard/batch_connect/sys/jupyter_lab/session_contexts/new

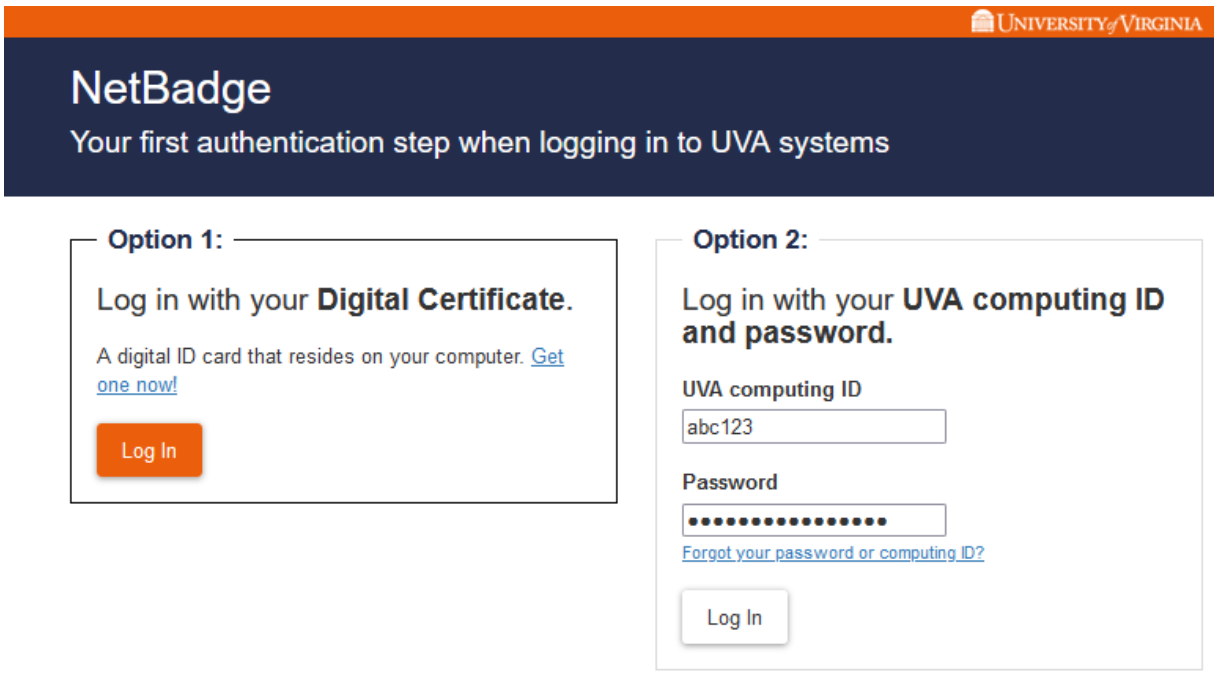


Figure 2: UVA Login

Figure: UVA Login

The user must install Duo Mobile on smartphone to use as an authentication service to approve logins.

For security reasons we suggest never saving the password within the browser autofill.

After logging in, you will receive an email through your UVA email inbox to create an account on Rivanna. Once completing the sign-up process, it will take around 1 hour for your account creation to be finalized.

If connecting through SSH, then a VPN is required. Follow the instructions to download UVA Anywhere at the following link: https://in.virginia.edu/vpn

To log in to Rivanna, ensure you are connected to UVA Anywhere and issue the following (make sure you replace `abc123` with your UVA id):

```
you@yourcomputer$ ssh-copy-id abc123@rivanna.hpc.virginia.edu
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
    ↪ any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
    ↪ now it is to install the new keys
abc123@rivanna.hpc.virginia.edu's password:
```

```
Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'abc123@rivanna.hpc.virginia.edu'"
and check to make sure that only the key(s) you wanted were added.

you@yourcomputer$ ssh abc123@rivanna.hpc.virginia.edu
Last login: Tue May 31 11:55:43 2022
Authorized Use Only!
-bash-4.2$
```

7.1.1 Notes: superpod

Estimated deployment for testing by the end of this summer.

Hardware Components:

- 10 DGX-A100 (80GB) Servers (8 GPUs)
- 2 DGX-A100 (40GB) Servers (16 GPUs)
- HDR Infiniband (200GB/s) IB network fabric for GPU-to-GPU direct communication
- 500T ESS3200 pure SSD SpectrumScale (aka GPFS) direct-to-GPU storage array

The SuperPod is a collection of GPU servers (Nvidia DGX-A100) integrated into the Rivanna Cluster (on the GPU partition) with an 200Gb/s IB fabric interconnecting the GPUs with each other and with dedicated temporary storage for Nvidia GPUDirect features. The GPU Direct features allow for very fast transfers between the GPUs, storage and also for larger distributed GPU models.

7.1.2 Special DGX Nodes on Rivanna

DGX A100 (udc-an36-1) is now available for your bii_dsc and bii_dsc_community members to test.

Here is the current status:

- The server is NOT YET integrated into the NVIDIA SuperPod because we are still awaiting networking equipment for implementing the SuperPod. We will be in touch if there is a need for a maintenance outage to integrate the server into the SuperPod.
- There is a RAID0 array of NVMe disks mounted locally at /localscratch. The capacity is 27TB. Please keep in mind that /localscratch is not backed up.
- The server is named udc-an36-1 and is currently in the bii-gpu partition with a permanent reservation named bi_fox_dgx for only bii_dsc and bii_dsc_community allocation members to use. To use this reservation for the A100 node, your researchers and students will have to use the following slurm flags:

```
#SBATCH --reservation=bi_fox_dgx
#SBATCH --account=<enter relevant allocation here>
#SBATCH --partition=bii-gpu
#SBATCH --gres=gpu:<number of GPUs to request>
```

For -account, users will enter either bii_dsc or bii_dsc_community depending on which group they belong to. You can find this by running the allocations utility at the commandline. For -gres=gpu:, users should enter the number of GPUs requested.

The full details of the reservation are below. I named the Slurm reservation “bi_fox_dgx”. It’s not a typo. To change the name of the reservation, I would have to delete the reservation and re-create it and the actual name of the reservation does not affect the reservation’s usability. I’ve successfully tested the ability to use this reservation for all the current bii_dsc and bii_dsc_community members using the Slurm parameters I sent previously.

```
ReservationName=bi_fox_dgx StartTime=2022-06-01T08:37:38 EndTime=2022-06-02T08:37:38
    ↪  Duration=1-00:00:00
   Nodes=udc-an36-1 NodeCnt=1 CoreCnt=256 Features=(null) PartitionName=bii-gpu
       ↪ Flags=DAILY,SPEC_NODES
   TRES=cpu=256
   Users=(null) Groups=(null) Accounts=bii_dsc,bii_dsc_community Licenses=(null)
       ↪ State=ACTIVE BurstBuffer=(null) Watts=n/a
   MaxStartDelay=(null)
```

7.1.2.1 Starting interactive job on special partition

```
ssh $USERNAME@rivanna.hpc.virginia.edu

$ ijob --reservation=bi_fox_dgx --account bii_dsc --partition=bii-gpu --gres=gpu:1
salloc: Pending job allocation 39263336
salloc: job 39263336 queued and waiting for resources
salloc: job 39263336 has been allocated resources
salloc: Granted job allocation 39263336
salloc: Waiting for resource configuration
salloc: Nodes udc-an36-1 are ready for job

$ nvidia-smi
```

which will result in

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.103.01   Driver Version: 470.103.01   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
```

```
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM...  Off  | 00000000:07:00.0 Off |                    0 |
| N/A   29C    P0    54W / 400W |     85MiB / 81251MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     13486      G   /usr/bin/X                        63MiB |
|    0   N/A  N/A     13639      G   /usr/bin/gnome-shell              21MiB |
+-----------------------------------------------------------------------------+
```

7.1.3 SSH Config

```
$ cat ~/.ssh/config
host rivanna
        User <USERNAME>
        HostName rivanna.hpc.virginia.edu
        IdentityFile ~/.ssh/id_rsa
```

7.2 Run Python MPI programs on Rivanna

see the book Python MPI

add chapter if not there

8 NLP

8.1 Documentation to get started with AWS Translate Service

AWS offers there a Text-translation service. A comorehensive manual on how to use it and sign up for this service is located at

- https://docs.aws.amazon.com/translate/latest/dg/setting-up.html

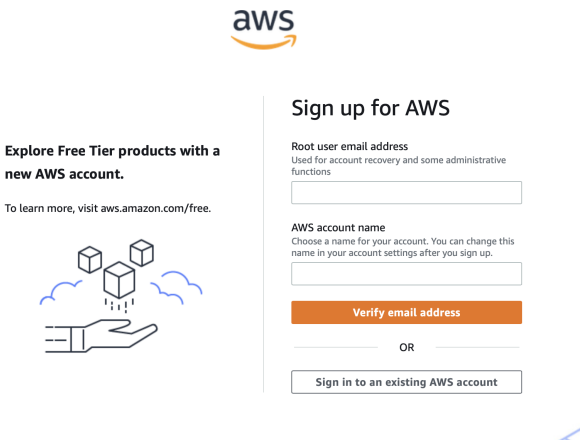
In this documentation we provide a summary of steps that give you quickly access to the service.

8.1.1 Step 1: Creating a new iam user account on aws.

To use AWS-translate you need to create an account via the IAM user account application form:

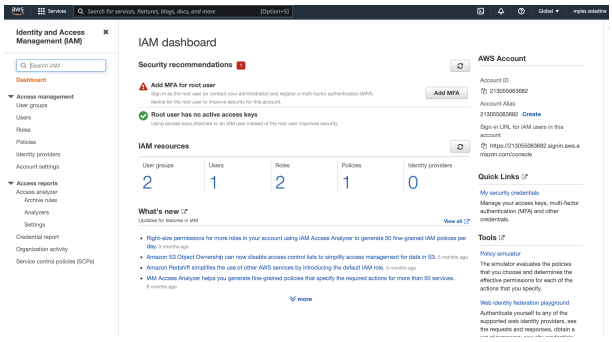
- https://portal.aws.amazon.com/billing/signup#/start/email

Like many other cloud services you will have to enable billing in order to use this service. Sign up for an iam user account and make the username name `adminuser`:



8.1.1.1 Step 2: Creating a access key ID and secret ID Once you have signed up with an IAM user account and have implemented billing you can navigate to the aws console:

- https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1#/home



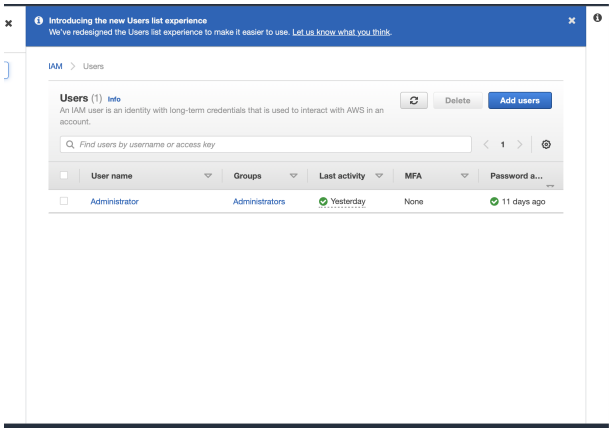
Here you can access secret keys and set permissions. At the top of this page search for text translate.

From here we are going to create an access key ID and a secret key id. This step is trivial to the success of a translation example

From the IAM dashboard page select the ‘Users’ option under IAM Resources

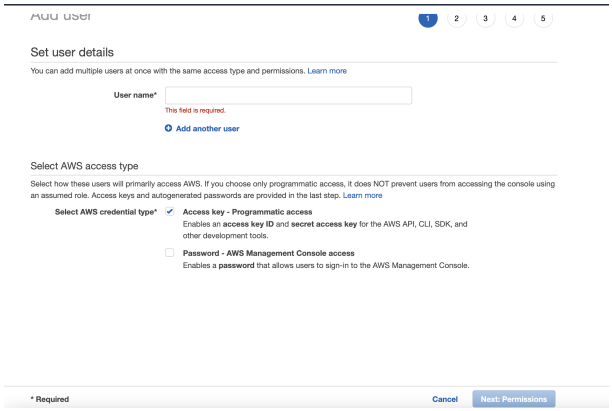
After clicking users you can create a user to run credentials.

From here click add user.



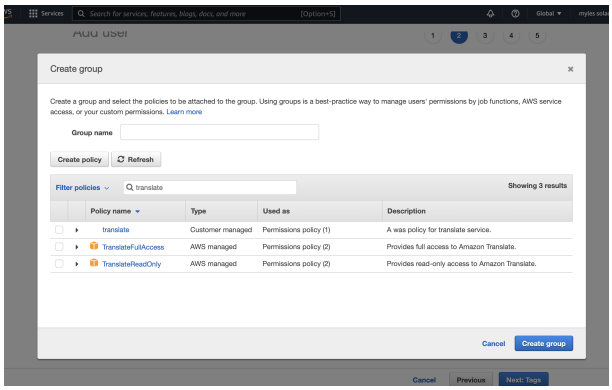
You will be required to add a name.

Make sure you check the box for Access Key and go to the next step.



Here is a section to add permissions for your IAM account User Group. From here search translate and check the boxes for `Translate` , and `TranslateFullAccess` .

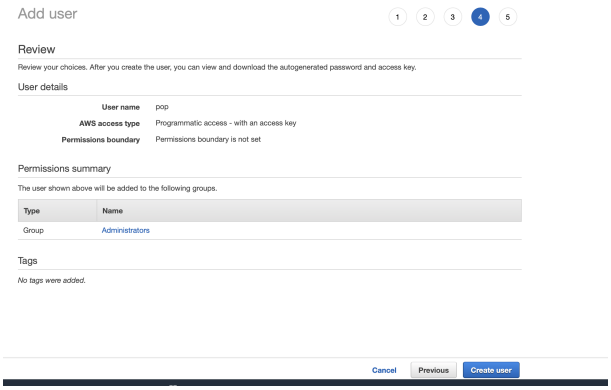
You can then click create group.



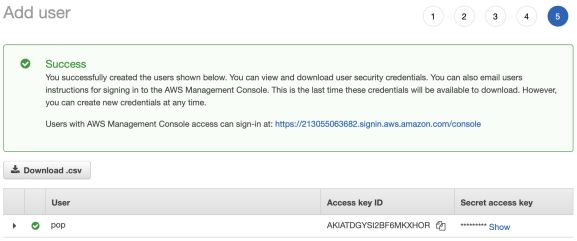
Next step is tags. this step is meant for users that want to give optional tags to their project.

You can skip this as it is irrelevant to this project.

After reaching this page you will want to confirm the creation of your user group.



From here you will be sent to a screen where you can download credentials



download these credentials as a csv file for later use.

Open up a terminal window to install aws on the command line.

Run these commands:

Start of a virtual enviorment

```
$ python3.10 -m venv ~/ENV3
$ source ~/ENV3/bin/activate
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
$ sudo installer -pkg ./AWSCLIV2.pkg -target /
$ which aws
$ aws --version
$ aws configure
```

Here they will give you an output like

```
AWS Access Key ID [****************4FCA]:
```

You will open the file you downloaded from the user creation earlier. You will see your personalized access key in that file. You will copy that key and paste it into the terminal.

You then will add your secret key which is also in the csv you downloaded.

```
AWS Secret Access Key [****************/kIg]:
```

You then will add the region

```
Default region name [eu-west-1]:
```

The region depends on which one is closest to you. For me, it is `eu-west-1`.

Then you will insert json

```
Default output format [json]:
```

Here It is recommended to insert `json`:

From here you can get started working in the command line found here: CLI Start

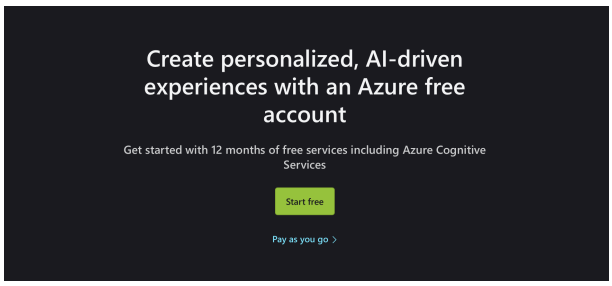
8.2 Documentation to get started with Azure Translate.

8.2.1 Step 1: account creation

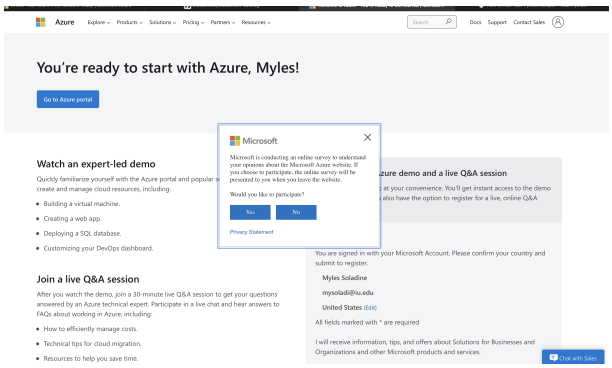
Here are steps to get started with the azure translate example in cloudmesh nlp.

navigate tot his link to sign uo with an azure account: https://azure.microsoft.com/en-us/free/cognitive-services/

After selecting this link you will follow the isntructions to set up a microsoft azure account. This requires billing.

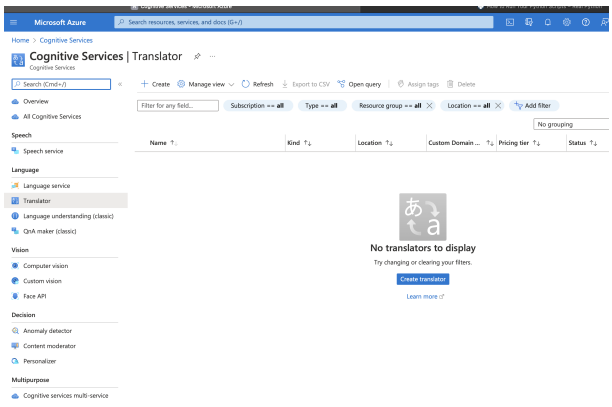


furthermore, after setting up an account with an email and billing you will be prompted to the console screen.



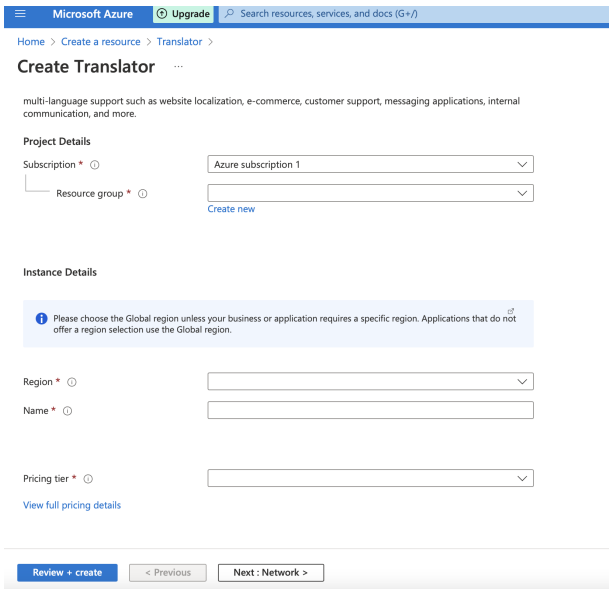
from here, you will click the top left link to go to the console.

You can either type translators in the search bar or follow this link: https://portal.azure.com/#blade/Microsoft_Azure_Pro

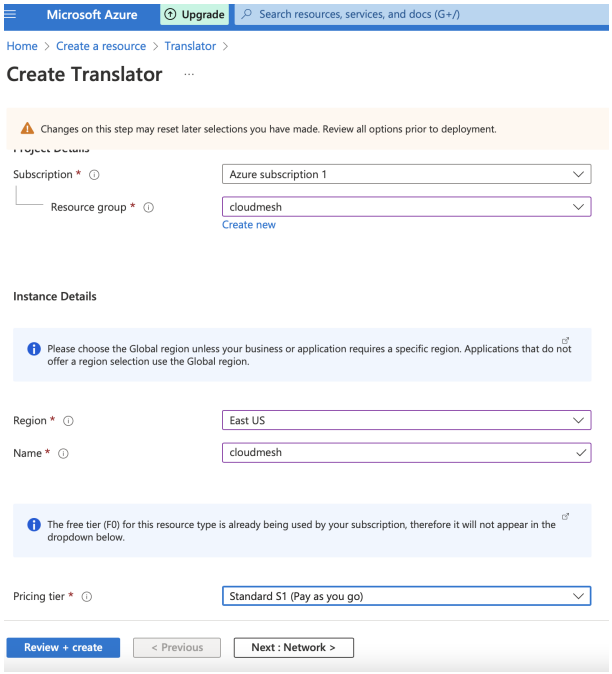


From here you will click the create translator button.

This is what the Create Translator console will look like. Here there is a form for creating an endpoint for this translate service

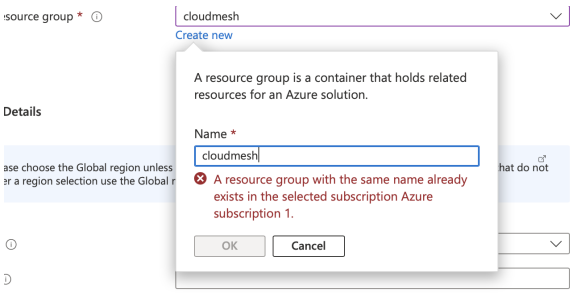


This what the form should look like filled out with the proper applied information.

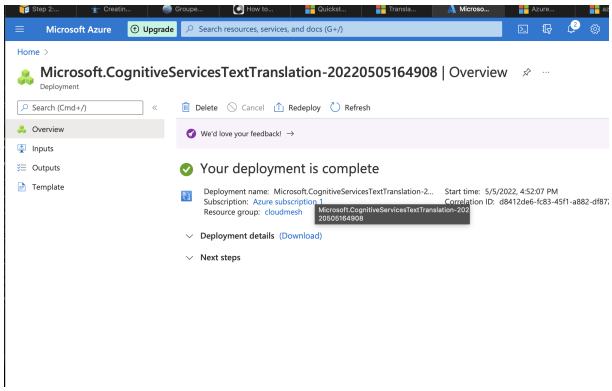


Resource groups are mostly irrelevant they are for bigger scaled projects where multiple people are working. Title this something for the project since it is required for an endpoint.

After this scroll down and hit ‘Create’



This is what the screen looks like after deployment. Download the deployment details and click next steps.



8.2.2 Installing and Starting Azure Translate through the command line

8.2.2.1 Step 2: installing using Homebrew Homebrew is by far the easiest way of installing Microsoft Azure.

On the command line use the command

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD
    ↪ /install.sh)"
```

This will install Homebrew and all of its packages. Now run the command below to install azure cli.

```
$ brew update && brew install azure-cli
```

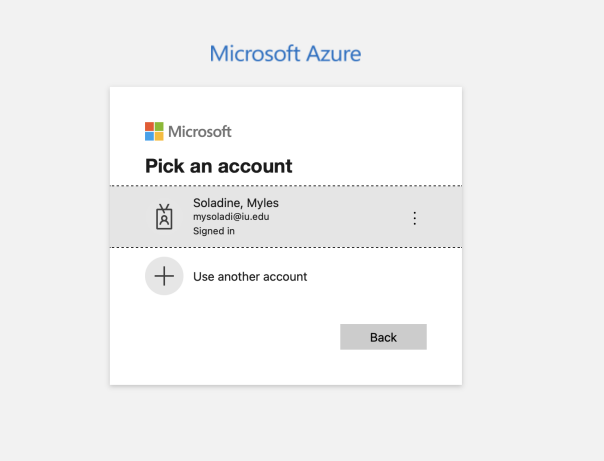
This installs the azure command line interface in the current directory.

From here we need to login to azure and its console that we created above.

start with

```
$ az login
```

This command will prompt you with a new window page with a microsoft login.



From here choose your proper login that was created with azure translate.

you will see a success in login.

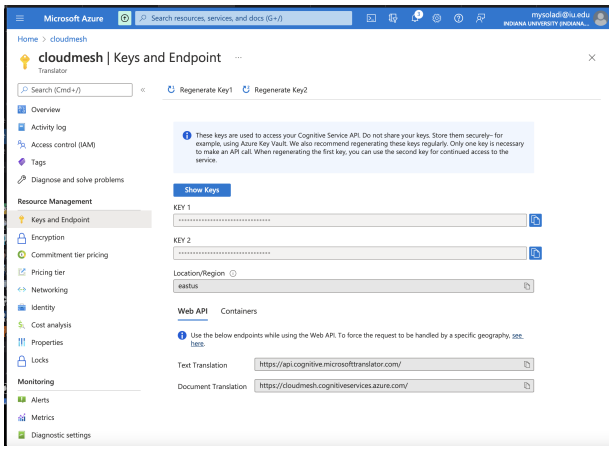
information about your account will be displayed on the command line.

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "1113be34-aed1-4d00-ab4b-cdd02510be91",
    "id": "d0ff5454-d152-4d11-8fe8-58a0c08581f1",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure subscription 1",
    "state": "Enabled",
    "tenantId": "1113be34-aed1-4d00-ab4b-cdd02510be91",
    "user": {
      "name": "mysoladi@iu.edu",
      "type": "user"
    }
  }
]
```

Use the command line to create a translate example using azure:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version
    ↪ =3.0&to=es" \
     -H "Ocp-Apim-Subscription-Key:<secret key>" \
     -H "Ocp-Apim-Subscription-Region:<region>" \
     -H "Content-Type: application/json" \
     -d "[{'Text':'Hello, what is your father?'}]"
```

For `secret key` you must insert the endpoint key that was generated in the previous account creation (same as key 1 in the figure below).



Region is also highlighted in this form.

This will return

```
[{"detectedLanguage":{"language":"en","score":1.0},"translations":[{"text":"Hallo
    ↪ Welt","to":"de"}]}]%
```

Using azure in a program sample found here you can use the endpoints, region, and secret key found in account creation to return a text translation.

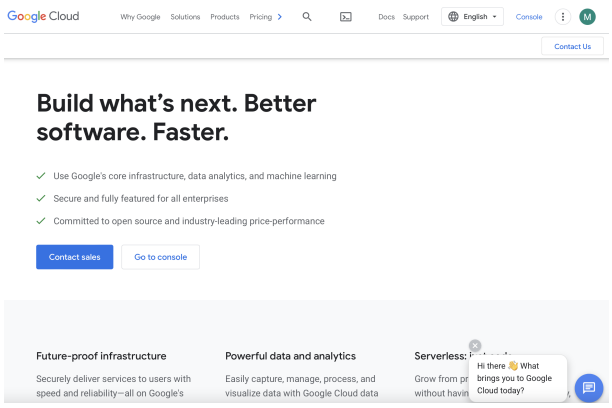
8.3 Natural Language Translation Example using Google Service

In order to get started using google translate there are steps for setup.

8.3.1 How to get started using this api.

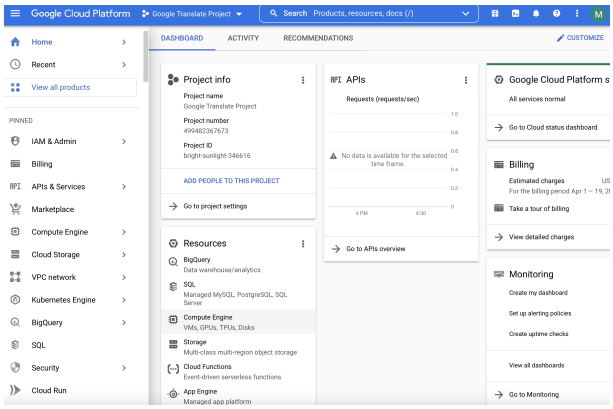
Navigate to the link

- https://cloud.google.com



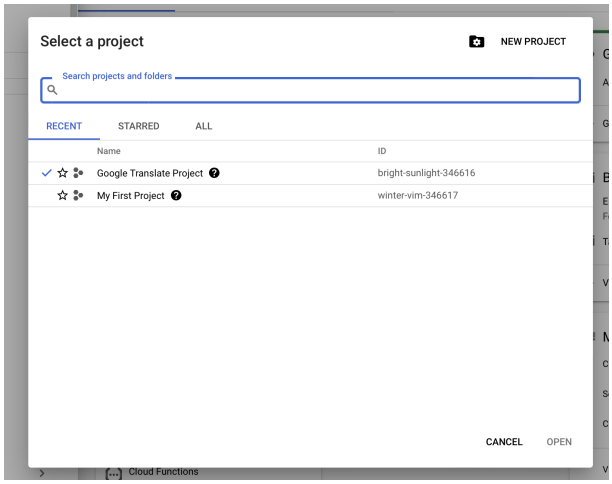
This is the homepage for Google cloud. Will need to activate your console with a Google account and billing.

Google offers a free trial of up to $300 of language translation to test. After activation of account, you will want to click console in the top right.

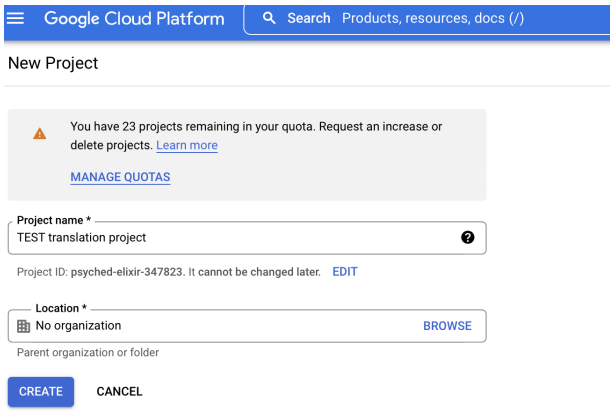


Here is the console for all google projects. The next step is to make a new project by selecting the dropdown at the top left.

The project creation page will look like this:



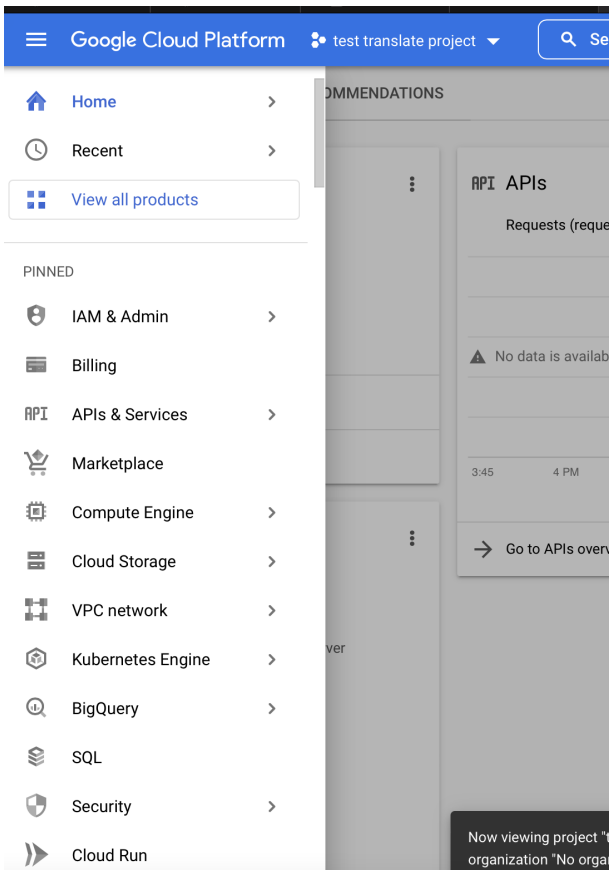
Then, click create new project.



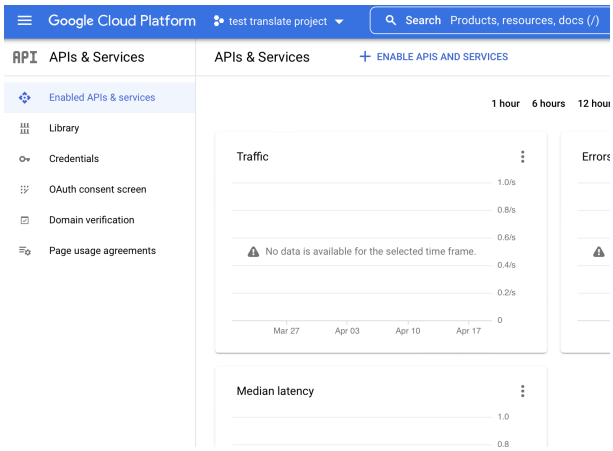
Give your project a title.

The dashboard will automatically update to use your new project, If not, select your project at the top left.

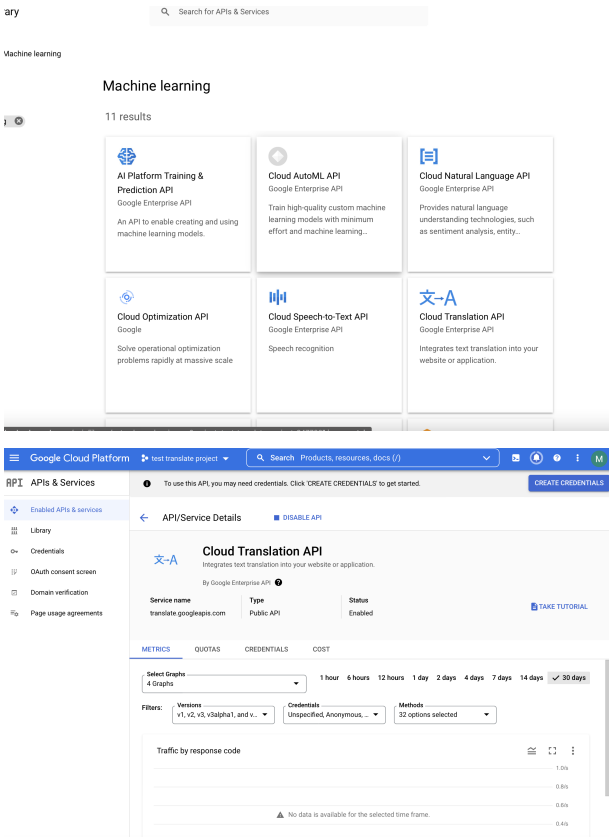
From here we need to activate the api for language translation. Select the api tab in the sidebar to the left.



From here you can click the enable apis and services tab at the top



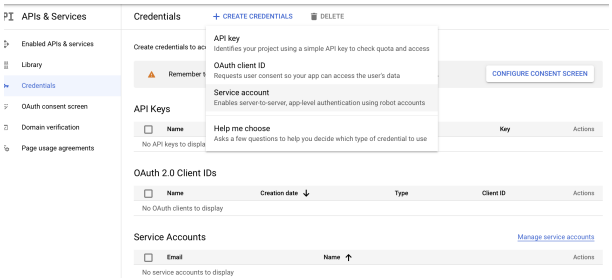
Scroll down to machine learning. Here you will see a cloud translation api. Click the api and enable it.



The api is now enabled, and you will see it on your dashboard.

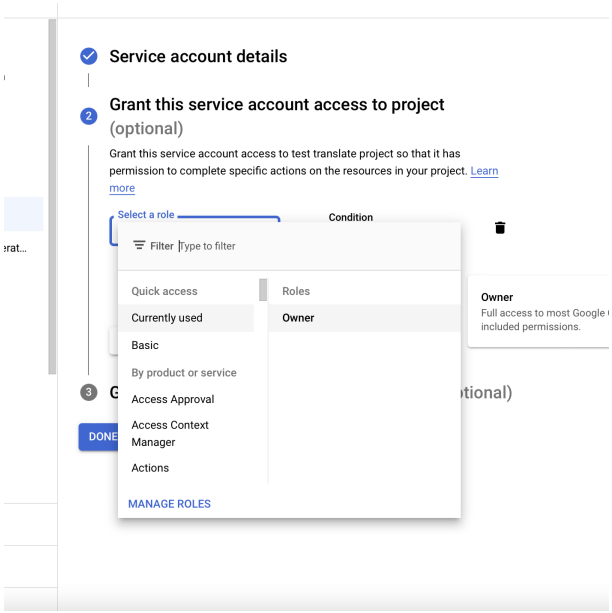
For each project you will have to enable credentials. this is a vital part to the continuation of the project.

on the tab to the left you will see credentials. here we are going to click create service account.

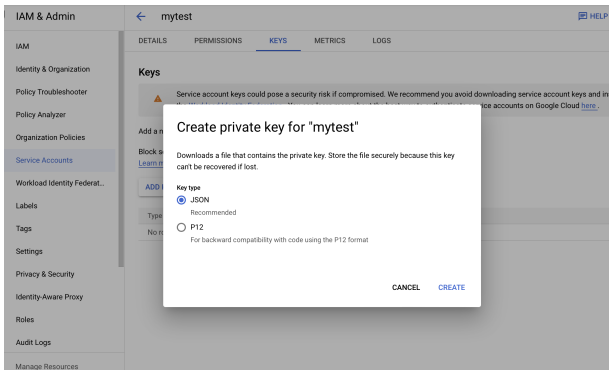


The next step is to title your service account. You will proceed and click the role owner.

this step is to decide what role each user has in the terms of the project. owners typically have access to most resources so we are going to select that one.



After the creation of the service account you will be prompted to download the json private key.



This private key will be placed in your Downloads folder. it is a very important piece of information. When creating this private key make sure to download as a json format.

In the command line you must create a virtual enviroment for the location of this project

```
$ python3.10 -m venv ~/ENV3
$ source ~/ENV3/bin/activate
```

You will need to give this private key a GOOGLE_CREDENTIALS_PATH. This is a very important step.

```
$ export GOOGLE_APPLICATION_CREDENTIALS="KEY_PATH"
```

Example:

```
export GOOGLE_APPLICATION_CREDENTIALS="/home/user/Downloads/service-account-file.
    ↪ json"
```

In order to use the client library for Natural language translation you will need to install some packages

```
pip install google-cloud-translate==2.0.1
```

or

```
pip install -r requirments.txt
```

From here you should have a linked api-key with a service account, and will be ready to use some examples of natural language programming.

An example for a small natural language program is showcased in snippets.py.

Documentation for getting started in the command line can be found in CLI Start.

8.4 Implementation of A Hybrid Cloud natural Language Example

8.4.1 How to Implement using Command Line Interface and The Cloudmesh Catalog (AWS Provider)

Step 1: Use the cloudmesh catalog to start the natural language example. AWS Natural Language Example

Here you will install the catalog first.

On mac

```
python3.10 -m venv ~/ENV3
source ~/ENV3/bin/activate
mkdir cm
cd cm
pip install cloudmesh-installer
cloudmesh-installer -ssh install catalog
```

From here you will install the source packages for this command line reference:

```
pip install -r requirements.txt
```

```
cms help
cms nlp translate --provider=aws --from=en --to=de --region=eu-west-1 hello world
```

The output to this command should look like:

```
{'date': '05/02/2022 14:45:45',
 'input': 'hello world',
 'input_language': 'en',
 'output': 'hallo welt',
 'output_language': 'de',
 'provider': 'aws',
 'time': 0.2641}
Timer: 0.3864s Load: 0.0004s nlp translate --provider=aws --from=en --to=de --region
    ↪ =eu-west-1 hello world
```

8.4.2 How to Implement using Command Line Interface and The Cloudmesh Catalog (Google Provider)

Step 1: Use the cloudmesh catalog to start the Natural Language example. Google Natural Language Example

Here you will install the catalog first.

On mac

```
python3.10 -m venv ~/ENV3
source ~/ENV3/bin/activate
mkdir cm
cd cm
pip install cloudmesh-installer
cloudmesh-installer -ssh install catalog
```

From here you will install the source packages for this command line reference:

```
pip install -r requirements.txt
```

```
cms help
cms nlp translate --provider=google --from=en --to=de --region=eu-west-1 hello world
```

The output to this command should look like:

```
{'date': '05/02/2022 14:45:45',
 'input': 'hello world',
 'input_language': 'en',
 'output': 'Hallo Welt',
 'output_language': 'de',
```

```
 'provider': 'aws',
 'time': 0.2641}
Timer: 0.3864s Load: 0.0004s nlp translate --provider=aws --from=en --to=de --region
    ↪ =eu-west-1 hello world
```

8.4.3 heterogenous cloudmesh nlp service

In this documentation we have an example of using a natural language operator from different providers. This is a service that is started with cloudmesh catalog. After installation of the catalog there are a list of services that can be used. Using `cms help` on the command line will give that list of services this output will look like:

```
Documented commands (type help <topic>):
========================================
EOF      commands  dryrun  host       nlp    quit   stopwatch  var
banner   config    echo    info       pause  set    sys        version
catalog  debug     gui     inventory  py     shell  sysinfo
clear    default   help    man        q      sleep  term
```

Here there is a newly implemented `nlp` command. This can be accessed by `cms nlp` in the command line. the source code can be found here

In order to start the translate service from CMS there are a few arguments we will be using. cms nlp translate takes the arguments:

```
--provider=google
--from=en
--to=de
--region=eu-west-1
```

The provider is interchangeable from the implemented services 'aws', 'google' and 'azure'. The argument `from` takes an interchangeable language code offered from the cloud-providers. This is the language your initial text is decoded in. The argument `to` is the target language the text will be translated to. This argument also takes a language code offered from the cloud providers. A list of Language codes are found here:

Provider	Google	aws
Language	Supported Language Codes	Supported Language Codes
Afrikaans	af-ZA	af-ZA

Provider	Google	aws
Arabic, Gulf	ar-AE	ar-AE
Arabic, Modern Standard	ar-SA	ar-SA
Chinese, Simplified	zh-CN	zh-CN
Chinese, Traditional	zh-TW	zh-TW
Danish	da-DK	da-DK
Dutch	nl-NL	nl-NL
English, Australian	en-AU	en-AU
English, British	en-GB	en-GB
English, India	en-IN	en-IN
English, Irish	en-IE	en-IE
English, New Zealand	en-NZ	en-NZ
English, Scottish	en-AB	en-US
English, South African	en-ZA	en-ZA
English, US	en-US	en-US
English, Welsh	en-WL	en-WL
French	fr-FR	fr-FR
French, Canadian	fr-CA	fr-CA
Farsi	fa-IR	fa-IR
German	de-DE	de-DE
German, Swiss	de-CH	de-CH
Hebrew	he-IL	he-IL
Hindi, Indian	hi-IN	hi-IN
Indonesian	id-ID	id-ID
Italian	it-IT	it-IT
Japanese	ja-JP	ja-JP
Korean	ko-KR	ko-KR
Malay	ms-MY	ms-MY

Provider	Google	aws
Portuguese	pt-PT	pt-PT
Portuguese, Brazilian	pt-BR	pt-BR
Russian	ru-RU	ru-RU
Spanish	es-ES	es-ES
Spanish, US	es-US	es-US
Tamil	ta-IN	ta-IN
Telugu	te-IN	te-IN
Thai	th-TH	th-TH
Turkish	tr-TR	tr-TR

When selecting a `region` parameter it is recommended to use `eu-west-1` for best success.

FAST API REDOCS DOCS * how to stop it * how to use it * how to see the documentation with docs and redoc

How to enable the service for google and aws given the previous readmes

9 AI

9.1 DL Timeseries

Learning Objectives

- Learn how to use deep learning for time series analysis

craete and documenta simple time series command

use cloudmesh sys command generate to create an extension so we can do a commandline tool

```
cms timeseries --config=CONFIG
```

where the config file is a yaml file.

Work with Gregor as he will create a separate github repo for this and create the command template so you can fill it out with content.

10 OTHER

10.1 Online book contribution

See if you can understand how to contribute to the online books. Demonstarte a contribution and if needed improve the documentation. You do not have to create the book from scratch ... we will do that another time. It is more important to work on git pull requests.

10.2 How to Set Up Browser Defaults on Windows

10.2.1 Changing Default Browser

1. Press the Windows key and type `Default apps`, then press `Enter`.
2. Change `Web browser` to your desired browser.

10.2.2 Changing Default Search Engine

Changing your search engine depends on the browser you are using. If you are using a fresh install of Windows, then the default browser is Microsoft Edge.

10.2.2.1 Microsoft Edge

Edge's default search engine is Bing.

1. In Microsoft Edge, click the three dots in the top-right corner and then click `Settings` at the bottom.
2. On the left-hand side, click `Privacy, search, and services`. Scroll all the way to the bottom and click `Address bar and search`.
3. Lastly, change `Search engine used in the address bar` from Bing to Google (or whichever engine you desire).

10.2.2.2 Google Chrome

Chrome's default search engine is Google.

1. In Chrome, click the three dots in the top-right corner and then click on `Settings`.
2. On the left-hand side, click `Search engine`.
3. Change `Google` to your desired search engine.

10.2.2.3 Firefox

Firefox's default search engine is Google.

1. In Firefox, click the horizontal three lines (hamburger menu) in the top-right corner.
2. Click on `Search` on the left-hand side.
3. Change `Default Search Engine` from Google to your desired search engine.

10.2.2.4 Opera

Opera's default search engine is Google.

1. In Opera, click the horizontal three lines with sliders (hamburger menu).
2. Scroll down and click on `Go to full browser settings`.
3. Scroll down to `Search engine` and change `Google Search` to your desired search engine.

10.3 Docker

there is loots of info in the book.

here we focus only on the installation and a simple example with fastapi make sure to check if this is not already documented.

10.3.1 Installation Windows

TBD * https://docs.docker.com/desktop/windows/install/

10.3.2 Installation Ubuntu

TBD * https://docs.docker.com/engine/install/ubuntu/
* https://docs.docker.com/desktop/linux/install/ubuntu/

10.3.3 Installation macOS

TBD * https://docs.docker.com/desktop/mac/install/

10.4 How to Set Up Git Bash with Pseudo Console Support on Windows

If you already have Git Bash installed, then uninstall it by hitting the Windows key, searching for `Add or remove programs`, searching for `Git`, clicking on it, then clicking `Uninstall` and completing the uninstallation wizard.

Then, reinstall (or install for the first time) with chocolatey in an instance of Powershell ran as administrator with

```
choco install git.install --params "/GitAndUnixToolsOnPath /Editor:Nano /
↪ PseudoConsoleSupport" -y
```

If you do not have chocolatey then follow the tutorial at https://chocolatey.org/install, or use the standard Git installer and check the box that reads `Enable experiment support for pseudo consoles`.

Now you can use `wsl` and other commands that would otherwise require `winpty` prepended to the command.

10.5 How to Auto-launch SSH Agent on Windows

Copy the following code:

```
env=~/.ssh/agent.env

agent_load_env () { test -f "$env" && . "$env" >| /dev/null ; }

agent_start () {
    (umask 077; ssh-agent >| "$env")
    . "$env" >| /dev/null ; }

agent_load_env

# agent_run_state: 0=agent running w/ key; 1=agent w/o key; 2=agent not running
agent_run_state=$(ssh-add -l >| /dev/null 2>&1; echo $?)

if [ ! "$SSH_AUTH_SOCK" ] || [ $agent_run_state = 2 ]; then
    agent_start
    ssh-add
elif [ "$SSH_AUTH_SOCK" ] && [ $agent_run_state = 1 ]; then
    ssh-add
fi

unset env

alias emacs="C:/ProgramData/chocolatey/bin/emacs.exe"
alias tree="cmd //c tree.com //a //f"
source ~/ENV3/Scripts/activate
```

```
cd ~/cm
```

Then, using Git Bash, run `nano ~/.bashrc`, use the down arrow key to ensure you are at the bottom of the file on a new line, and then paste the contents using `Shift + Insert`. Consider, beforehand, even deleting the `.bashrc` if you have a preexisting script that may conflict with the new additions and then creating a new `.bashrc` with the above contents.

Keep in mind that if emacs is not installed using choco, the emacs alias will not function. You can install emacs using choco by running `choco install emacs`. This will not work if you have not installed choco, which you can do by following https://chocolatey.org/install

Additionally, the source command will not work if you have not created a virtual Python environment named `ENV3` in your home dir.

The `cd` will also not function if `cm` dir does not exist in the home dir. If so, then execute `mkdir ~/cm`.

10.6 How to Install WSL Ubuntu 22.04 on Command Line for Windows

Execute each command on Git Bash:

```
curl -o ~/ubuntu-base-22.04-base-amd64.tar.gz https://cdimage.ubuntu.com/ubuntu-base
    ↪ /releases/22.04/release/ubuntu-base-22.04-base-amd64.tar.gz
mkdir -p ~/wsl/ubuntu-22.04/instances
wsl --import ubuntu-22.04 ~/wsl/ubuntu-22.04/instances ~/ubuntu-base-22.04-base-
    ↪ amd64.tar.gz
rm ~/ubuntu-base-22.04-base-amd64.tar.gz
```

Then you can run Ubuntu 22.04 by executing `wsl -d ubuntu-22.04`

Git Bash may freeze if pseudo console support is not enabled. In such a case, execute the `wsl -d ubuntu-22.04` command on Powershell.

10.7 University of Virginia

Create a nice section with real paragraphs and words from this list and some of the discussions in the slack

https://cloudmesh-reu2022.slack.com/archives/C03F89TMFQ9/p1653399309252089

11 BIOS

11.1 Bios

Please add here a 2-3 paragrapgh professional Bio. Look up who a professional bio is isn IEEE papers. Write in 3rd person. TOD: provide link example.

Review other peoples bios and improve or give improvement tips where needed.

If it turns out you never contributed to anything, your bio will be removed (as well as your name in this proceedings).

11.1.1 Paul Kiattikhunphan

Paul Kiattikhunphan is a second year at the University of Virginia majoring in computer science.

11.1.2 Alex Beck

Alex Beck has completed his first year at the University of Virginia where he is majoring in electrical engineering. He is set to receive his Bachelor of Science degree in the Spring of 2025. He currently maintains a 3.9 GPA.

Alex is currently working in research this summer at the UVA Biocomplexity Institute's Computing for Global Challenges program under Dr. Gregor Von Laszewski and Dr. Geoffrey Fox where he is planning to gain experience in programming and data science. Prior to that, he has previous experience in sales from working in retail.

At UVA, Alex is involved in a few extracurricular organizations. He is currently active in the Virginia Eta Chapter of Sigma Phi Epsilon, the UVA Climbing Team, and the UVA Chapter of the QuestBridge Scholars Network.

11.1.3 Alison Lu

Alison Lu has completed her second year (Class of 2024) at the University of Virginia pursuing a double major in CS and Chemistry with a minor in Japanese. She is conducting research with the physics department studying quantum computing and photon resolution using machine learning. In addition, she works with UVA's Repair Lab to study gentrification in Norfolk, VA.

She is currently participating in the Biocomplexity Institute's C4GC REU program. Her interests include computer architecture, machine learning, and quantum computing alongside quantum mechanics.

11.1.4 Jackson Miskill

Jackson Miskill has completed his second year at the University of Virginia where he is studying Computer Science and Cognitive Science. He will receive a Bachelor of Arts degree from UVa in Spring of 2024. Jackson has studied python and java in his courses, delving into concepts from basic syntax to data structures and algorithms.

Jackson is currently working at the UVa Biocomplexity Institute under Dr. Gregor von Laszewski as a part of the Computing for Global Challenges program. He is studying the intersection between python and cloud computing. In the future, Jackson plans to continue research.

11.1.5 Jacques Fleischer

Figure 3: Jacques's Picture

Jacques Fleischer is a sophomore at the Miami Dade Honors College. He is set to receive his associate degree in computer science in summer 2022. He received the Miami Dade Honors College Fellows Award and currently maintains a 4.0 GPA on the Dean's List.

In the summer of 2021, he participated in the Florida-Georgia Louis Stokes Alliance for Minority Participation REU Data Science and AI Research Program; his research focused on predicting the price of cryptocurrency using artificial intelligence. This was done in conjunction with faculty from Florida A&M University and Indiana University. He presented his findings at the Miami Dade College School of Science Symposium in October 2021. Jacques was accepted to the 2022 Emerging Researchers National (ERN) Conference in STEM after applying with his abstract on cryptocurrency time-series. Additionally, he was one of four Miami Dade College students to be nominated for the Barry Goldwater Scholarship due to his research findings.

Jacques is active in extracurriculars; for instance, he is the current Vice President of the MDC Computer Club. There, he hosts virtual workshops on how to use computer software, including Adobe Premiere

Pro and PyCharm. He is also a member of Phi Theta Kappa. Furthermore, he is an active contributor to Cloudmesh: an open-source, all-in-one grid-computing solution written in Python. He presently participates in the University of Virginia's Computing for Global Challenges program with Dr. Gregor von Laszewski and Dr. Geoffrey C. Fox to find high performance computing solutions using Raspberry Pis.

11.1.6 Eric He

Junyang (Eric) He completed his first year of study at the University of Virginia majoring in Computer Science. He anticipates to receive his B.S. in Computer Science in 2025. He is currently a member of the Engineering Student Council and the Chinese Student and Scholars Society at UVA.

In the summer of 2021, Eric worked as a Data Analyst intern at Nint (Shanghai) Co., Ltd, a company that provides market data analysis products for E-commerce His work included time series data cleaning and natural language processing.

Eric is currently conducting research with Prof. Geoffrey C. Fox on a Deep Learning model based on LSTM networks trained to predict hydrological features like streamflow, precipitation, and temperature at different locations in the US. He focused primarily on the possibilities of extending the model to countries outside of the US such as Chile and UK.

11.1.7 Abdulbaqiy Diyaolu

AbdulBaqiy Diyaolu is a Computer science and Mathematics Major from Mississippi Valley State University. He will be receiving his bachelor's degrees in both Computer science and Mathematics in the year 2025. AbdulBaqiy is currently a presidential scholar at Mississippi Valley State University and he maintains a 4.0 GPA.

AbdulBaqiy currently works at Fedex Logistics at MVSU. He helps in data entry and data Analysis. He is hoping to polish his data analysis skills with this opportunity. In the summer of 2022, he joins the Bio complexity research program at UVA where he will be able to use his skills in support of different researches, and also learn more research skills along the way.

Abdulbaqiy participates in several extracurricular activities in MVSU he is a member of African Student Union(ASU), National Society of Black engineers (NSBE), and the google developer's club. He is also a Strada scholar at MVSU where he participates in several leadership development activities.

11.1.8 Thomas Butler

Thomas Butler is a Graduate Student at University of Virginia's School of Data Science. His undergraduate degree is in Biomedical engineering. He has over eight years of experience in the Infertility field helping patients, jointly running a Andrology lab, and contributing research to advance the field through joint research on how AMH effects pregnancy outcomes and sperm antibodies effect PSA. He has a certificate in Data Analytics from Georgia Institute of Technology.

11.1.9 Robert Knuuti

Robert Knuuti is a Graduate Student at University of Virginia's School of Data Science. He has over 10 years experience in system architecture and software engineering, and specializes in Development Operations and Cloud Computing. He has constructed air gapped Continuous Integration and Continuous Delivery systems for multiple organizations each supporting more than 100 developers and has facilitated the construction of repeatable, tractable builds for users of these systems.

11.1.10 Jake Kolessar

Jake Kolessar is a Graduate Student at the University of Virginia's School of Data Science. He has a background in mechanical engineering and 2 years of experience as a Modeling, Simulation and Analysis Engineer. He has supported the software design and development of modeling capabilities for event simulation products as well as the integration of models into the simulation framework.

12 REFERENCES