

---






# **Proceedings of the REU2022**



Cybertraining

Gregor von Laszewski, laszewski@gmail.com


04 June, 2022

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
1.0.1	Contribute 	5
1.0.2	REU 2022 	5
1.0.2.1	Books	5
1.0.2.2	The repository for our report(s)	6
1.0.2.3	The many UVA technical help ticket systems	6
1.0.2.4	Our technical slack	6
1.0.2.5	The administrative slack	6
1.0.2.6	Rivanna Information	6
1.0.2.7	GitBash (Windows only)	6
1.0.2.8	Pycharm	6
1.0.2.9	Bibliography Management	7
1.0.2.10	GitHub repo	7
1.0.2.11	Only important for Gregor (do not use)	7
1.0.3	Communications with C4GC Students 	7
<b>2</b>	<b>INSTALL</b>	<b>8</b>
2.0.1	Install 	8
2.0.1.1	Windows	8
2.0.1.1.1	Git Bash install	8
2.0.1.1.2	Python 3.10 install	9
2.0.1.1.3	Uninstall	10
2.0.1.2	Choco install	10
2.0.1.3	Choco install pycharm	10
2.0.1.4	Installing Pycharm	11
2.0.1.5	Linux	11
2.0.1.5.1	Install Python 3.10.5	11
2.0.1.5.2	Setting up the a venv	12
2.0.1.5.3	Uninstall	12
2.0.1.5.4	Update	12
2.0.1.6	macOS	13
2.0.1.6.1	Cloudmesh	13
2.0.1.6.2	Updating Python	13
2.0.1.6.3	Homebrew install	14
2.0.2	Ramdisk 	15
2.0.2.1	Ubuntu	15

2.0.2.2	windows . . . . .	16
<b>3</b>	<b>GRAPH VIZUALIZATION</b>	<b>16</b>
3.0.1	Python Data Management  . . . . .	16
3.0.1.1	Lists . . . . .	16
3.0.1.2	Dictionaries . . . . .	16
3.0.1.3	CSV Files . . . . .	16
3.0.2	Python Graphics  . . . . .	16
3.0.2.1	Matplotlib . . . . .	16
3.0.2.1.1	Installation . . . . .	16
3.0.2.1.2	Import Statements . . . . .	17
3.0.2.1.3	Bar Chart . . . . .	17
3.0.2.1.4	Line Chart . . . . .	18
3.0.2.1.5	Pie Chart . . . . .	20
3.0.2.1.6	Contour Plot . . . . .	21
3.0.2.1.7	Titles, Labels, and Legends . . . . .	23
3.0.2.1.8	Rotating Ticks . . . . .	23
3.0.2.1.9	Exporting . . . . .	25
3.0.2.2	Bokeh . . . . .	25
3.0.2.2.1	Installation . . . . .	26
3.0.2.2.2	Import Statements . . . . .	26
3.0.2.2.3	Bokeh Plotting Interface . . . . .	26
3.0.2.2.4	Figure Parameters Example . . . . .	27
3.0.2.2.5	Saving Figures . . . . .	27
3.0.2.2.6	Scatter Plot . . . . .	29
3.0.2.2.7	Line Plots . . . . .	30
3.0.2.2.8	Bar Chart . . . . .	32
3.0.2.3	Seaborn . . . . .	34
3.0.2.3.1	Installation . . . . .	35
3.0.2.3.2	Import Statements . . . . .	35
3.0.2.3.3	Relational Plots . . . . .	35
3.0.2.3.4	Distribution Plots . . . . .	36
3.0.2.3.5	Categorical Plots . . . . .	38
3.0.2.3.6	Regression Plots . . . . .	39
3.0.2.3.7	Saving Figures . . . . .	40
3.0.2.4	Sources . . . . .	40
3.0.2.4.1	Matplotlib . . . . .	40
3.0.2.4.2	Seaborn . . . . .	40

3.0.2.4.3	Bokeh	40
3.0.3	Pandas Graphics	41
3.0.3.1	Installation	41
3.0.3.2	Import Statements	41
3.0.3.3	Bar Chart	41
3.0.3.4	Line Chart	43
3.0.3.5	Pie Chart	45
3.0.3.6	Exporting	47
<b>4</b>	<b>PYTHON</b>	<b>47</b>
4.0.1	Cloudmesh StopWatch	47
4.0.2	cms sys command generate	47
4.0.3	Linux	48
4.0.4	pandas	48
4.0.5	Python	48
4.0.5.1	Queue	48
4.0.5.1.1	FIFO Queue	48
4.0.5.1.2	LIFO Queue	48
4.0.5.1.3	Priority Queue	48
4.0.6	FastAPI	49
4.0.6.1	FastAPI Install	49
4.0.6.2	FastAPI Example	49
4.0.6.2.1	Step 2: create a FastAPI instance	51
4.0.6.2.2	Step 3: create a path operation	52
4.0.6.2.3	Step 4: define the path operation function	52
4.0.6.2.4	Step 5: return the content	53
<b>5</b>	<b>RIVANNA</b>	<b>53</b>
5.0.1	Rivanna	53
5.0.1.1	Notes: superpod	55
5.0.1.2	Special DGX Nodes on Rivanna	55
5.0.1.2.1	Starting interactive job on special partition	56
5.0.1.3	SSH Config	58
5.0.2	Run Python MPI programs on Rivanna	58
<b>6</b>	<b>AI</b>	<b>58</b>
6.0.1	DL Timeseries	58

<b>7 BIOS</b>	<b>59</b>
7.0.1 Bios 	59
7.0.1.1 Paul Kiattikhunphan	59
7.0.1.2 Alex Beck	59
7.0.1.3 Alison Lu	59
7.0.1.4 Jackson Miskill	60
7.0.1.5 Jacques Fleischer	60
7.0.1.6 Eric He	61
7.0.1.7 Abdulbaqiy Diyaolu	61
<b>8 REFERENCES</b>	<b>61</b>

## 1 INTRODUCTION

### 1.0.1 Contribute

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 will be considered. For example let us assume you spent significant time on a section that is a duplication of what others do, we will then delete that section 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.

### 1.0.2 REU 2022

#### 1.0.2.1 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>

#### **1.0.2.2 The repository for our report(s)**

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

#### **1.0.2.3 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>

#### **1.0.2.4 Our technical slack**

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

#### **1.0.2.5 The administrative slack**

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

#### **1.0.2.6 Rivanna Information**

- <https://www.rc.virginia.edu/>
- <https://www.rc.virginia.edu/userinfo/rivanna/overview/>
- 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>

#### **1.0.2.7 GitBash (Windows only)**

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

#### **1.0.2.8 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.

### 1.0.2.9 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.

### 1.0.2.10 GitHub repo

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

### 1.0.2.11 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/>

## 1.0.3 Communications with C4GC Students

- weekly meetings with Ben Hurt every Tuesday and Wednesday from 11-11:45 am on Zoom and in room 4402.
- weekly presentations during these Tuesday and Wednesday 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 INSTALL

### 2.0.1 Install

TODO: Abdul

Improve the installation instructions for python in the book.

#### 2.0.1.1 Windows

##### 2.0.1.1.1 Git Bash install

- Install gitbash from <https://git-scm.com/downloads>
- Click **Download** for Windows. The download will commence. Please open the file once it is finished downloading.
- The UAC Prompt will appear. Click **Yes** because Git is a safe program. It will show you Git's license: a GNU General Public License. Click **Next**.
- Click **Next** to confirm that `C:\Program Files\Git` is the directory where you want Git to be installed.
- Click **Next** unless you would like an icon for Git on the desktop (in which case you can check the box and then click **Next**).
- Click **Next** to accept the text editor,
- Click **Next** again to Let Git decide the default branch name
- Click **Next** again to run Git from the command line and 3rd party software,
- Click **Next** again to use the OpenSSL library
- Click **Next** again to checkout Windows-style,
- Click **Next** again to use MinTTY,



- Click **Next** again to use the default git pull,
- Click **Next** again to use the Git Credential Manager Core,
- Click **Next** again to enable file system caching, and then
- 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/HCoTx\\_xCfA](https://youtu.be/HCoTx_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/>

### 2.0.1.1.2 Python 3.10 install

- Install python from <https://python.org>
- Click **Download** . The download will commence. Please open the file once it is finished downloading
- Click **Add python 3.10 to path**
- Click **Install now**

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>

Document the options, e.g. switch on path, icon on desktop, allow path longer then 256 chars

Start gitbash

```
$ 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
```

start new gitbash and remove the first gitbash window, see if you see (ENV3) and continue. Git bash will initialize the environment

start now again gitbash and remove the second gitbash you created. Now gitbash is properly created.

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

#### 2.0.1.1.3 Uninstall

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

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

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

**2.0.1.2 Choco install** There are a number of usefull packages that you can install via choco this includes visual code, pycharm, emacs, and make

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

Even python could be installed with it however we have not tested, if it adds python to the path or sets the maxmunm oath to greated then 256. For that reason we recommend to install python the regular way as documented in the video ... jps video

#### 2.0.1.3 Choco install pycharm

- 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 by launching PowerShell as Administrator or gitbash.

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

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

**2.0.1.4 Installing Pycharm** PyCharm can be installed in gitbash with choco while typing

```
$ choco install pycharm -y
```

Once the install completes PyCharm will be ready for you to use. You can install many programs this way, and the

**2.0.1.5 Linux** .

**2.0.1.5.1 Install Python 3.10.5** The installation from source can be done easily as shown next

```
$ 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
```

#### 2.0.1.5.2 Setting up the a venv We assume you use bash

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

#### 2.0.1.5.3 Uninstall

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

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

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

**2.0.1.5.4 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
$ cloudmesh-installer new ~/ENV3 cmd5 --python=/usr/local/bin/python3
  ↪ .10
```

```
$ source ~/ENV3/bin/activate
$ python -V
$ which python
```

**2.0.1.6 macOS** We assume you use zsh which is the default on macOS

#### **2.0.1.6.1 Cloudmesh** 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 ~/ENV3/bin/activate
$ 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
```

In a short summary, this essentially creates the virtual environment, creates another 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`.

Uninstall

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

**2.0.1.6.2 Updating Python** Before starting this process, ensure that python is in the correct path. This can be checked by following the scripting below:

```
$ which python # should print user/ENV3/bin/python
```

```
$ python3.10 --version # should print the current version of python
↪ in the venv
```

Then, follow the directions below:

- First, download the latest version of `python`
- Second, follow the download instructions from the python launcher that is created.
- Third, navigate into the virtual environment directory: `ENV3`
- Fourth, execute the following command:

```
$ python -m venv --upgrade ENV3
```

This will properly update the virtual environment python to the correct python version. Run the `which python` and `python --version` commands once again to ensure that the correct version has been installed.

**2.0.1.6.3 Homebrew install** Homebrew (`brew`) like `choco` is a package management software. Unlike `choco`, it is used by macOS devices rather than Windows devices. `brew` is used to more easily download packages to an operating system; simply put, it eliminates the need for the user to search for and download the desired package.

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]
```

## 2.0.2 Ramdisk

how to set up and manage a ramdisk on linux

develop cms program

cms ramdisk -... -size=SIZE

use humanize so we can use 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) backup 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}`')
```

**2.0.2.1 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/](https://wiki.ubuntuusers.de/RAM-Disk_erstellen/)

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

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

## 3 GRAPH VIZUALIZATION

### 3.0.1 Python Data Management

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 manage them. They are lists, dictionaries and CSV files.

#### 3.0.1.1 Lists

#### 3.0.1.2 Dictionaries

#### 3.0.1.3 CSV Files

### 3.0.2 Python Graphics

In Python, data and equations can be visually represented using graphs and plots. Whereby showcasing how to use different plotting libraries, this includes Matplotlib, Bokeh, and Seaborn.

**3.0.2.1 Matplotlib** Matplotlib is a library that allows the user to visualize data. The library can create pie charts, bar charts, line plots, and other graphs specifically for data visualization. Matplotlib creates figures that can be manipulated and transformed. This includes manipulations of axes, labels, fonts, and the size of the images.

**3.0.2.1.1 Installation** To install matplotlib, please use the command:

```
$ pip install matplotlib
```



**3.0.2.1.2 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
```

**3.0.2.1.3 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

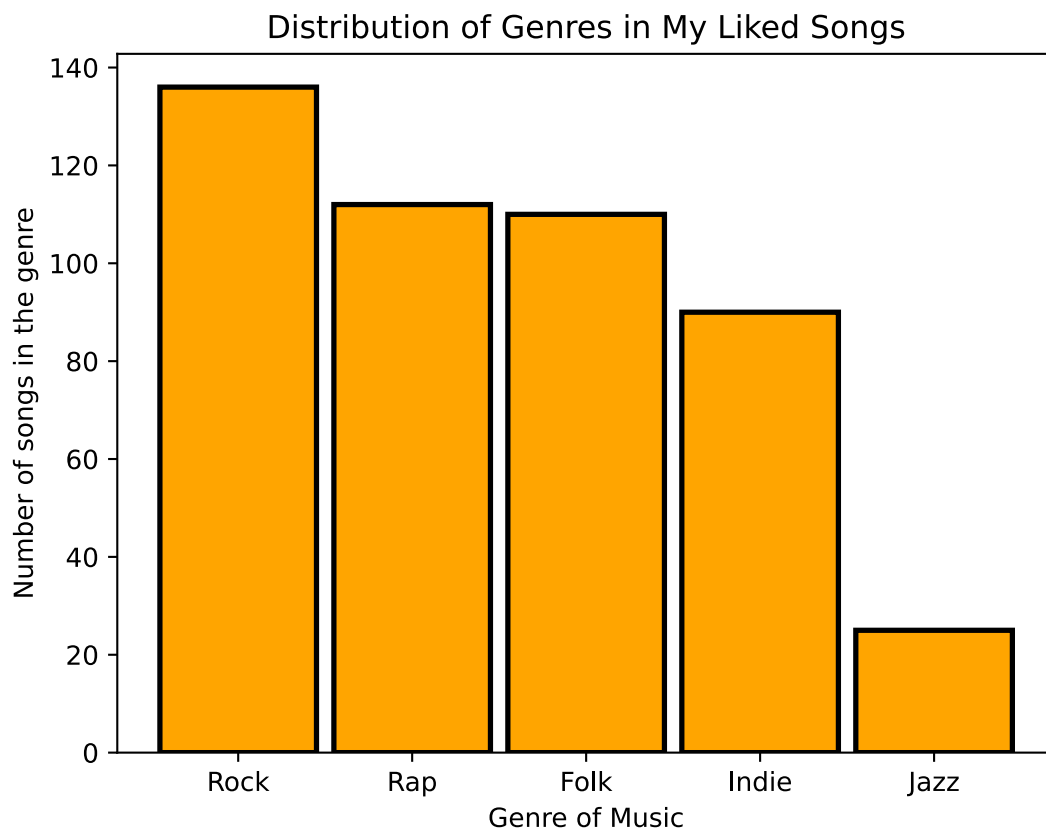
data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz':
    ↪ 25}
categories = data.keys()
count = data.values()

# Creating the bar chart
plt.bar(categories,
        count,
        align='edge',
        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 Figure *barchart*.



**Figure 1:** barchart

Figure *barchart*: Barchart created from data from Spotify

**3.0.2.1.4 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

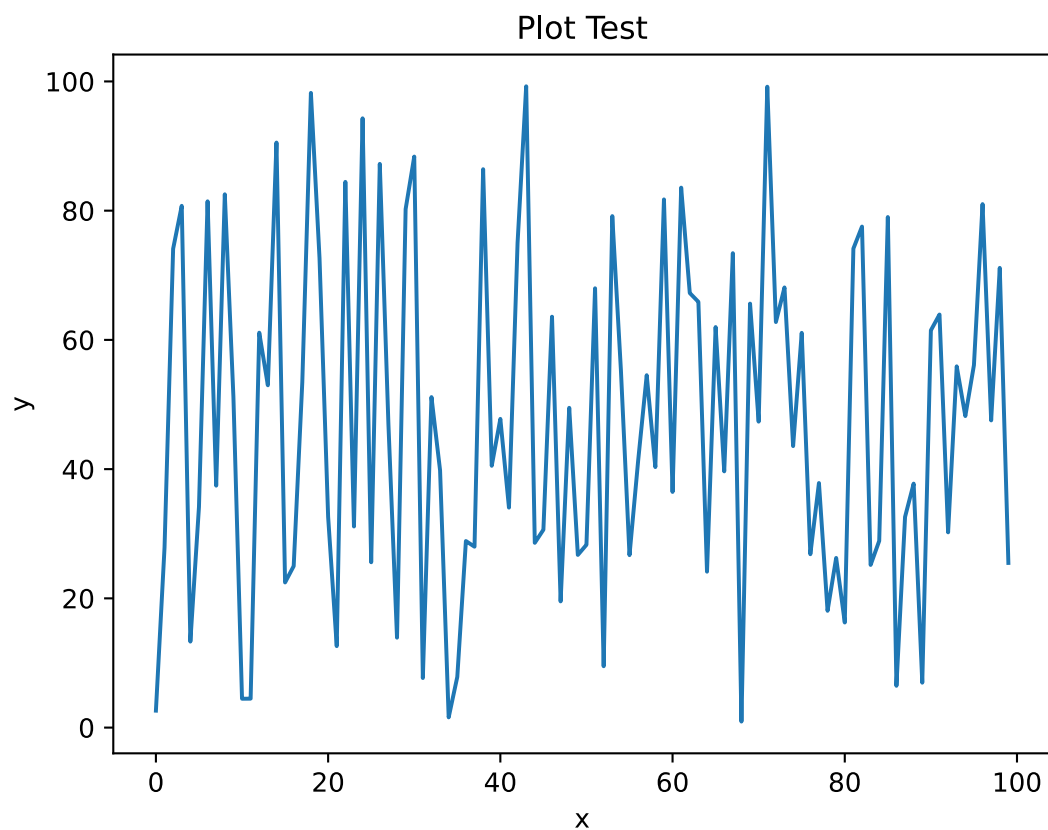
x = []
y = []
for i in range(0, 100):
    x.append(i)
    value = random.random() * 100
```

```
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 Figure *linechart*.



**Figure 2:** linechart

Figure *linechart*: Linechart created from random variables

**3.0.2.1.5 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

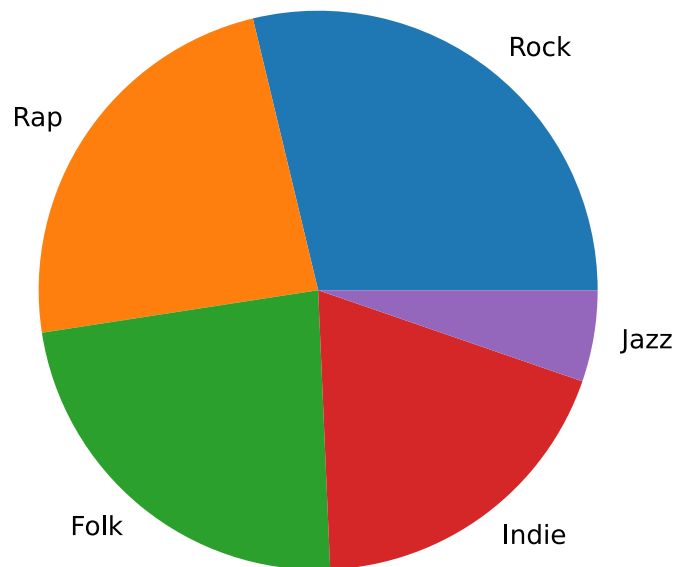
data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz':
    ↪ 25}
categories = data.keys()
count = data.values()

# 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 Figure *piechart*.



**Figure 3:** piechart

Figure *piechart*: Barchart created from data from Spotify

**3.0.2.1.6 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

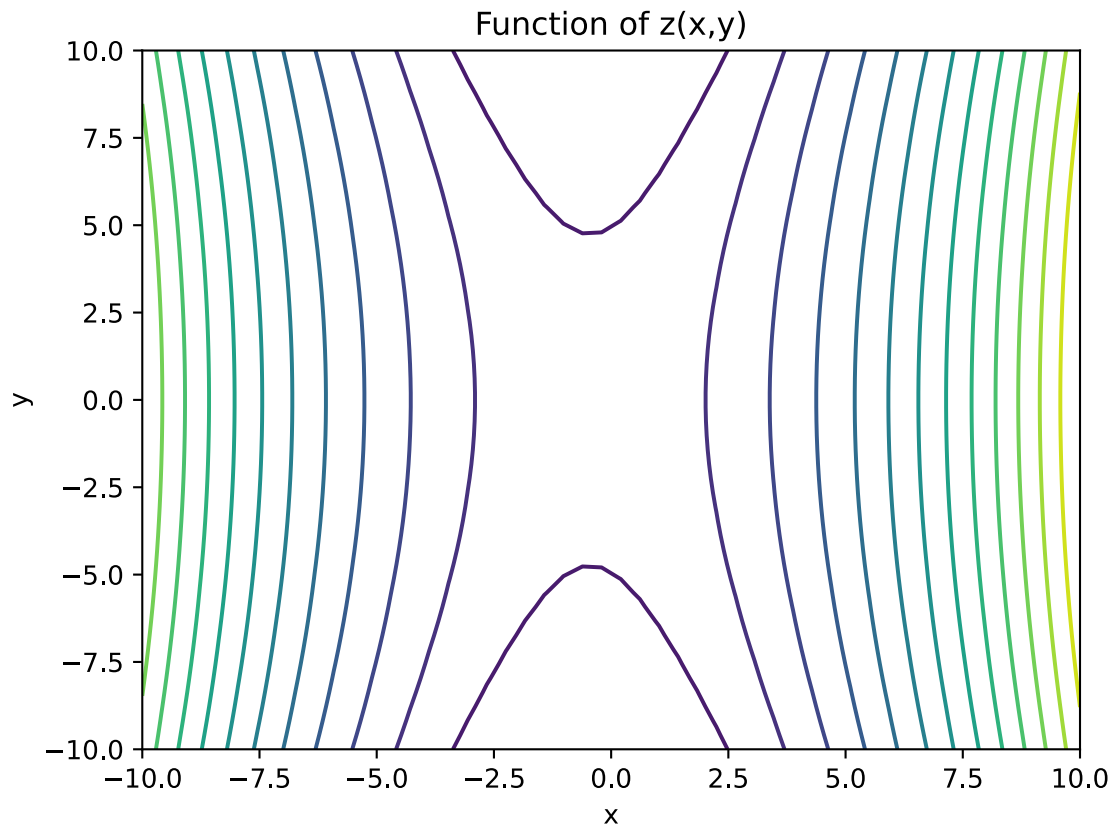
#creating an equation for z based off of variables x,y
x, y = np.meshgrid(np.linspace(-10, 10), np.linspace(-10, 10))
z = 9*(x**2+1)+8*x-(y**2)
levels = np.linspace(np.min(z), np.max(z), 15)
```

```
#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 Figure *contourplot*.



**Figure 4:** contour

Figure *contourplot*: Multivariable (x, y, z) Equation Plotted

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.

### 3.0.2.1.7 Titles, Labels, and Legends 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").
```

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")
```

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()
```

**3.0.2.1.8 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`. This can be shown by this simple [example](#).

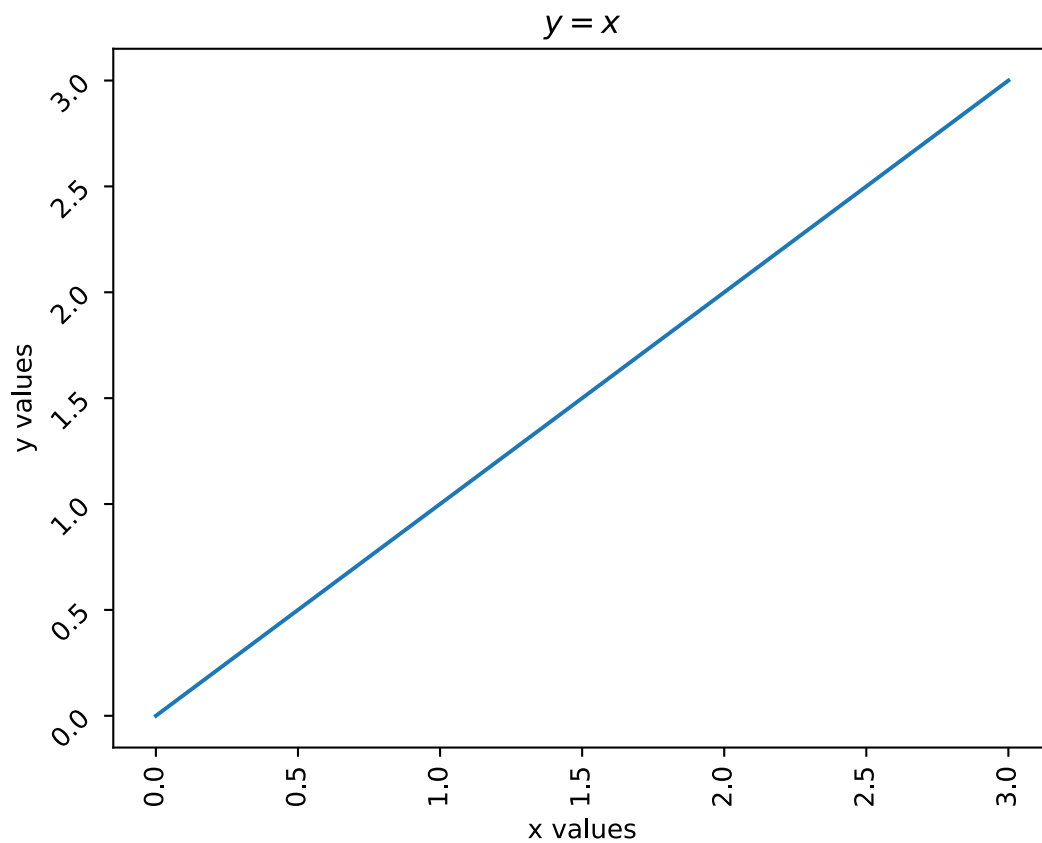
```
import matplotlib.pyplot as plt

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()
```



**Figure 5:** ticks

Figure *ticks* x-axis ticks rotated by 90° and y-axis ticks rotated by 45°



### 3.0.2.1.9 Exporting 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(p):
    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(p)
```

This code can be accessed on [GitHub](#)

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()
```

**3.0.2.2 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.

**3.0.2.2.1 Installation** To install Bokeh, please use the command:

```
$ pip install bokeh
```

**3.0.2.2.2 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
```

**3.0.2.2.3 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

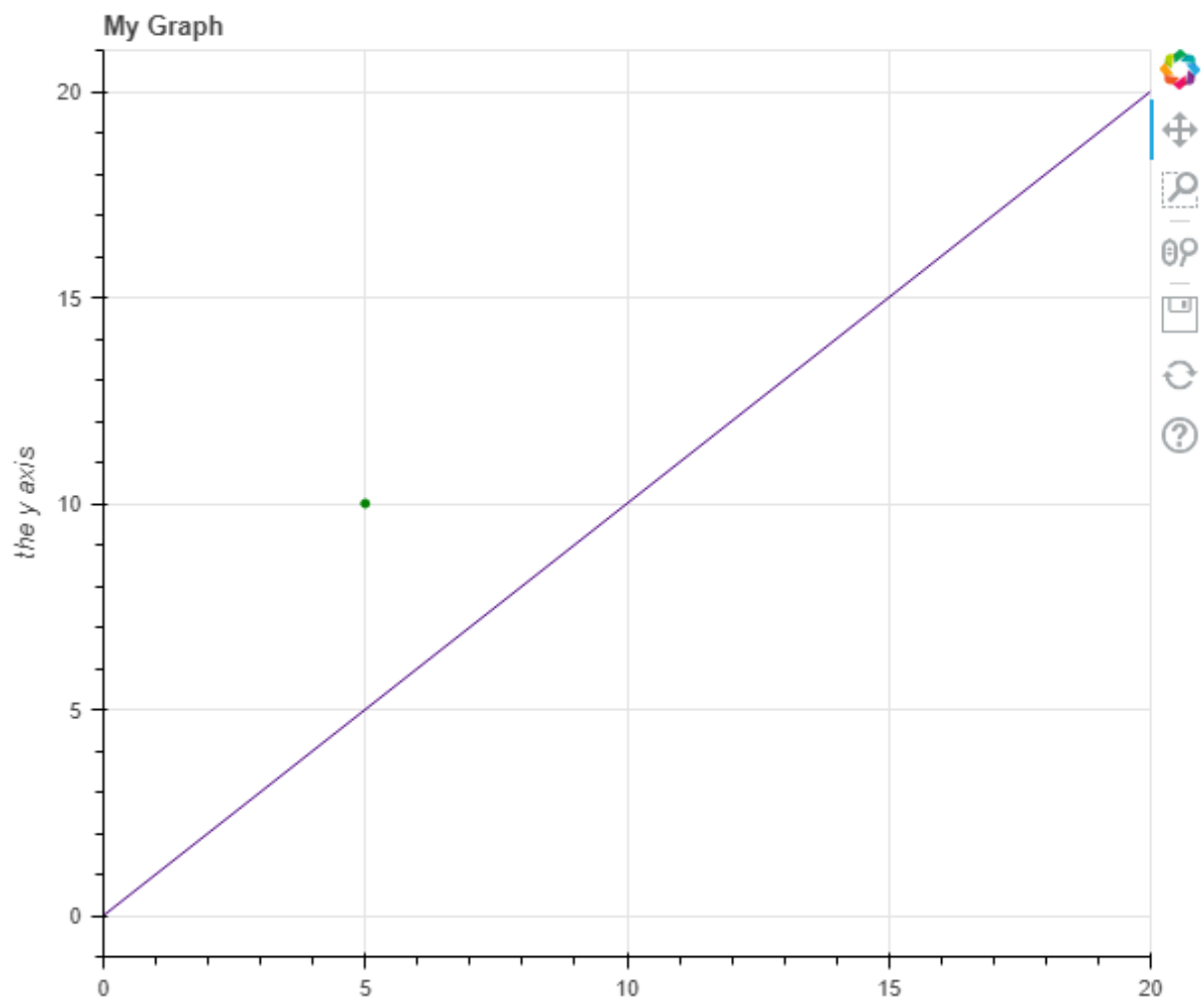
# labeling the title, specifying the range of the x-axis, labeling
# the y-axis, specifying the height to be 500 pxls
p = figure(title = "My Graph", x_range = [0,20], y_axis_label = "the
# y axis", height = 500)

# 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](#)



**Figure 6:** figure

Figure *lineplot*: Figure created with Bokeh.

#### 3.0.2.2.4 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)

**3.0.2.2.5 Saving Figures** Bokeh also 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.

#### Saving Figures as PNG and SVG

In order to save figures as a PNG or a SVG, 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 command:

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

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

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

In addition, the `export_png()` and `export_svg()` functions 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")
```

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](#).

**3.0.2.2.6 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

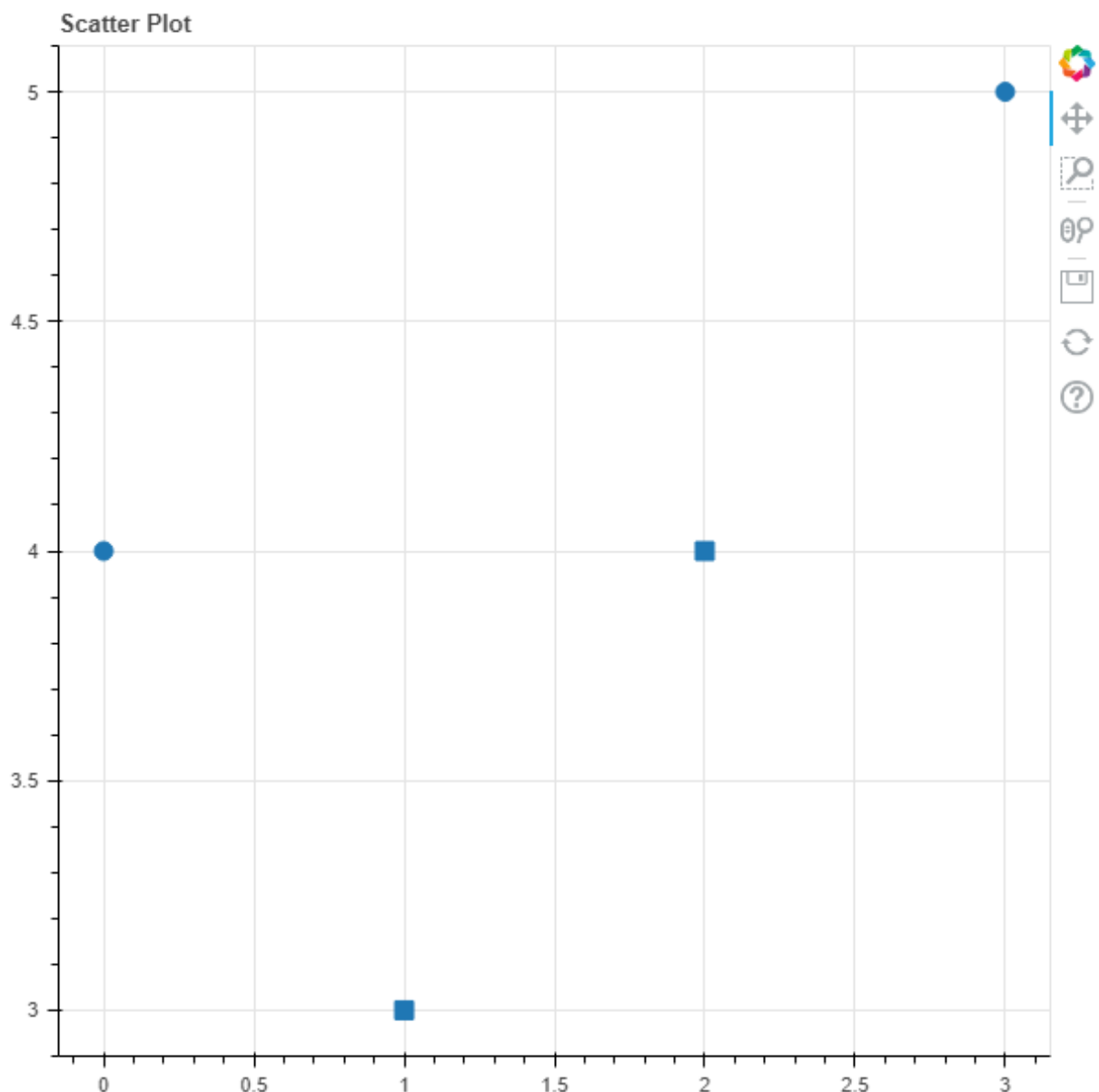
p = figure(title="Scatter Plot")

# 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](#)



**Figure 7:** Scatter Plot

Figure *Scatter Plot*: Scatter Plot created with user Spotify data.

The list of all possible marker types and the functions used to create them can be found [here](#)

**3.0.2.2.7 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.

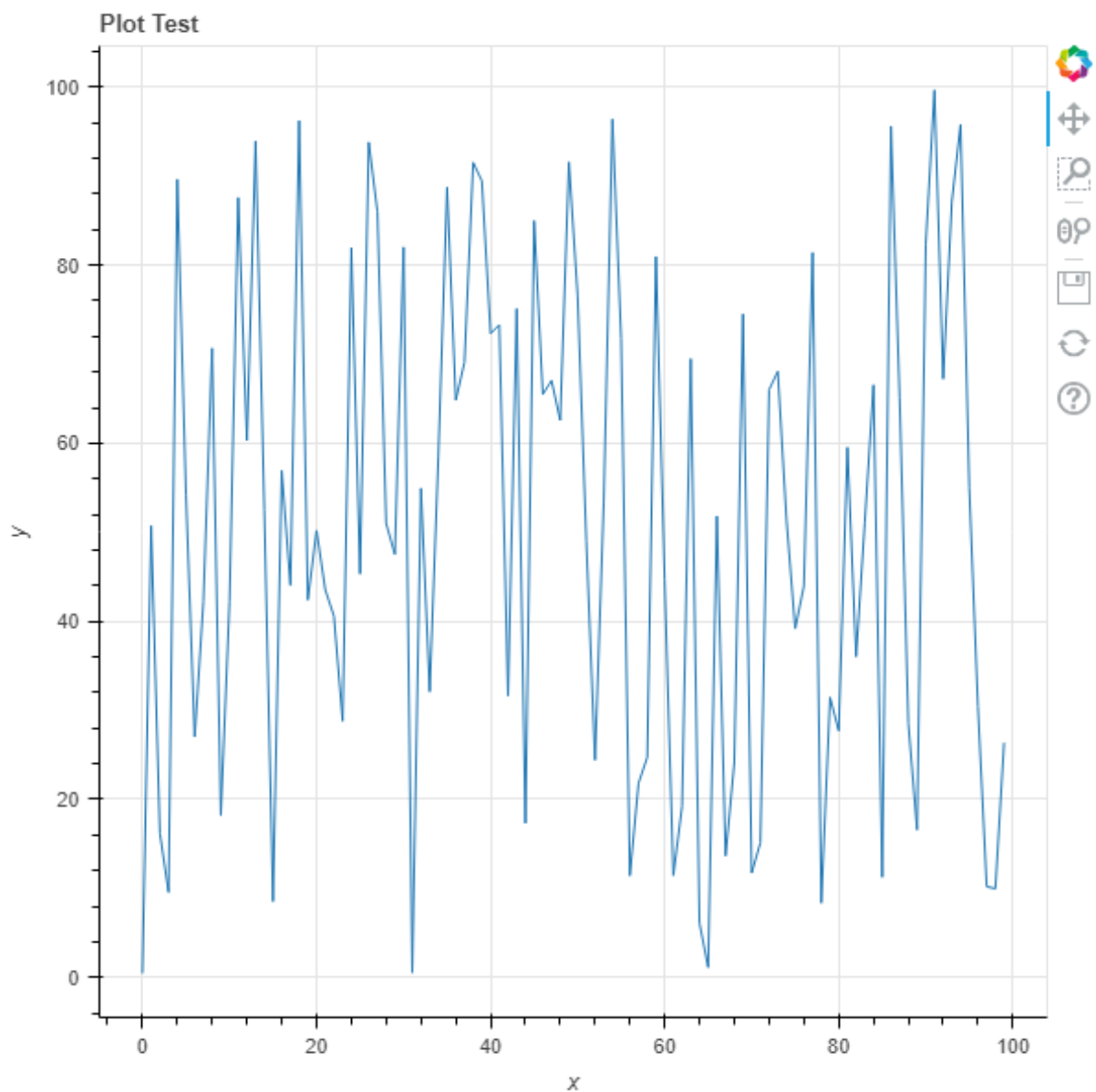
```
from bokeh.io import show, export_png, export_svg
from bokeh.plotting import figure
import random

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](#)



**Figure 8:** Line Plot

Figure *Line Plot*: Line Plot created with user Spotify data.

You can find the source code for other types of line plots here: [http://docs.bokeh.org/en/latest/docs/user\\_guide/plotting.html](http://docs.bokeh.org/en/latest/docs/user_guide/plotting.html)

**3.0.2.2.8 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

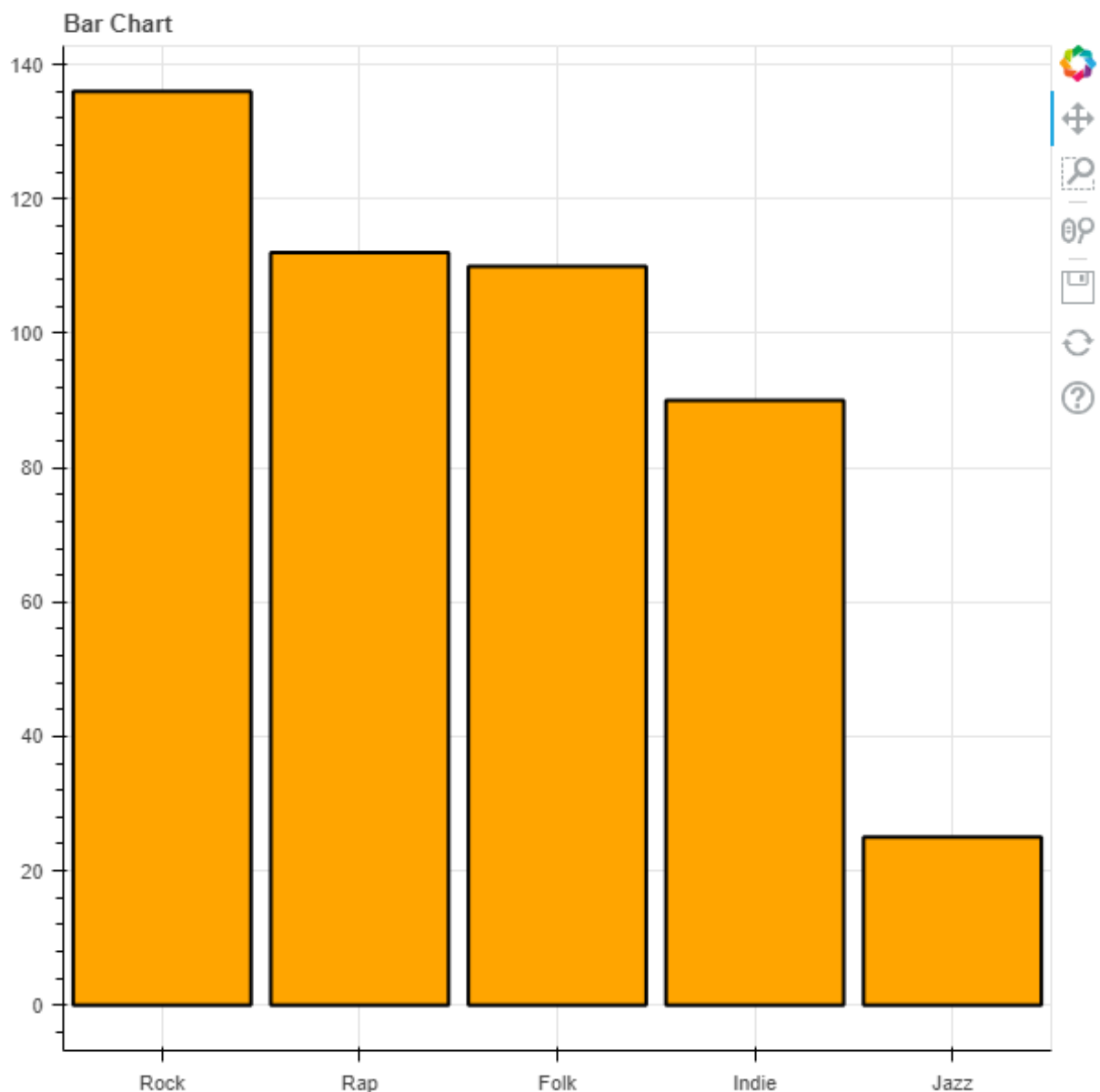
data = {'Rock': 136, 'Rap': 112, 'Folk': 110, 'Indie': 90, 'Jazz':
    ↪ 25}
x = list(data.keys())
y = list(data.values())

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](#)



**Figure 9:** Bar Chart

Figure *Bar Chart*: Bar Chart created with user Spotify data.

**3.0.2.3 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.

**3.0.2.3.1 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
```

**3.0.2.3.2 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]
```

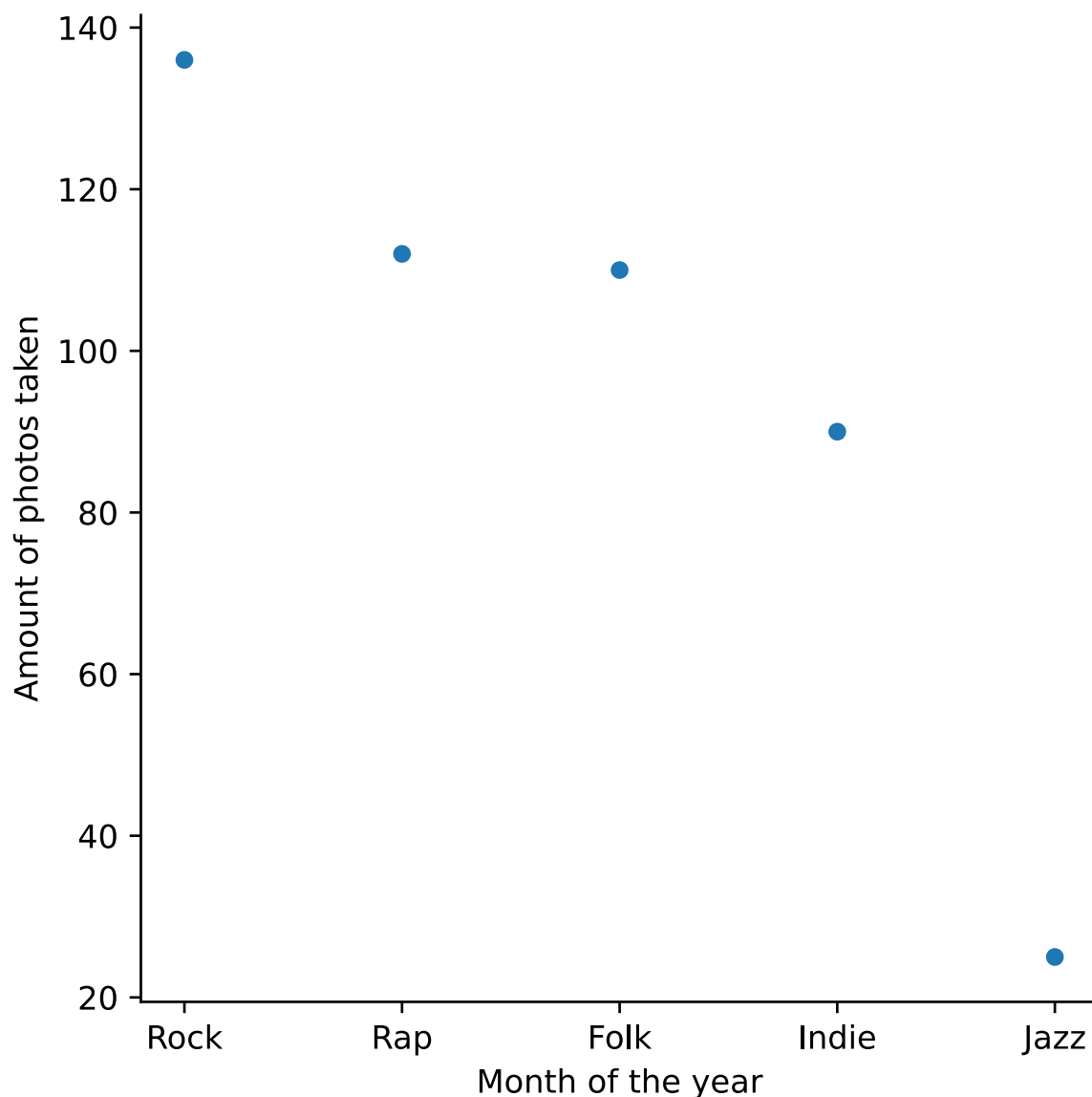
**3.0.2.3.3 Relational Plots** Relational plots showcase the relationship between variables in a visual format. It is a broad term for data representation. Examples of relational plots in Seaborn are `relplot`, `lineplot` and `scatterplot`.

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 Figure *lineplot*



**Figure 10:** lineplot

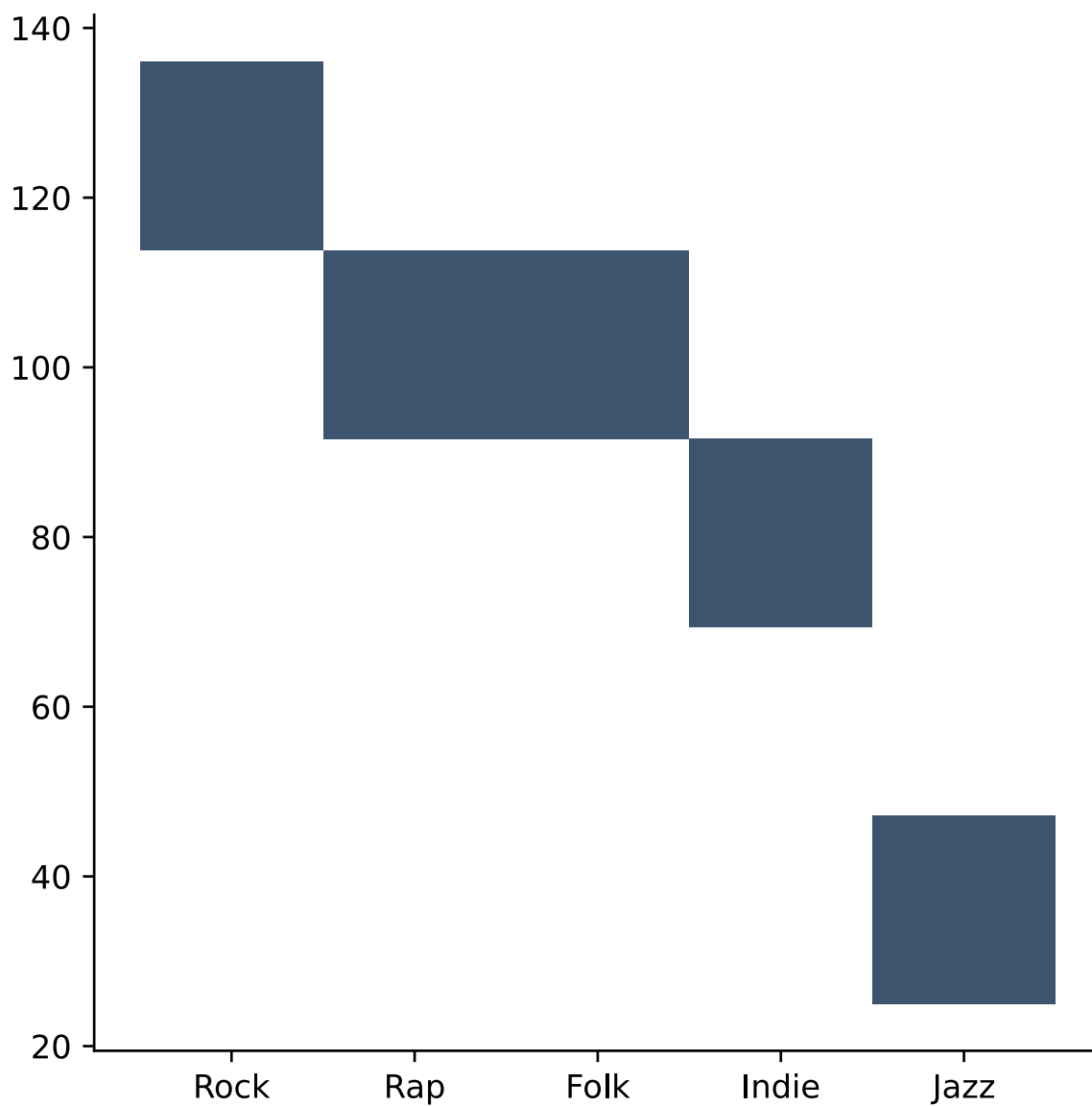
Figure *lineplot*: Lineplot created from user Spotify data.

**3.0.2.3.4 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 Figure *displot*



**Figure 11:** *displot*

Figure *displot*: Displot created from user Spotify data.

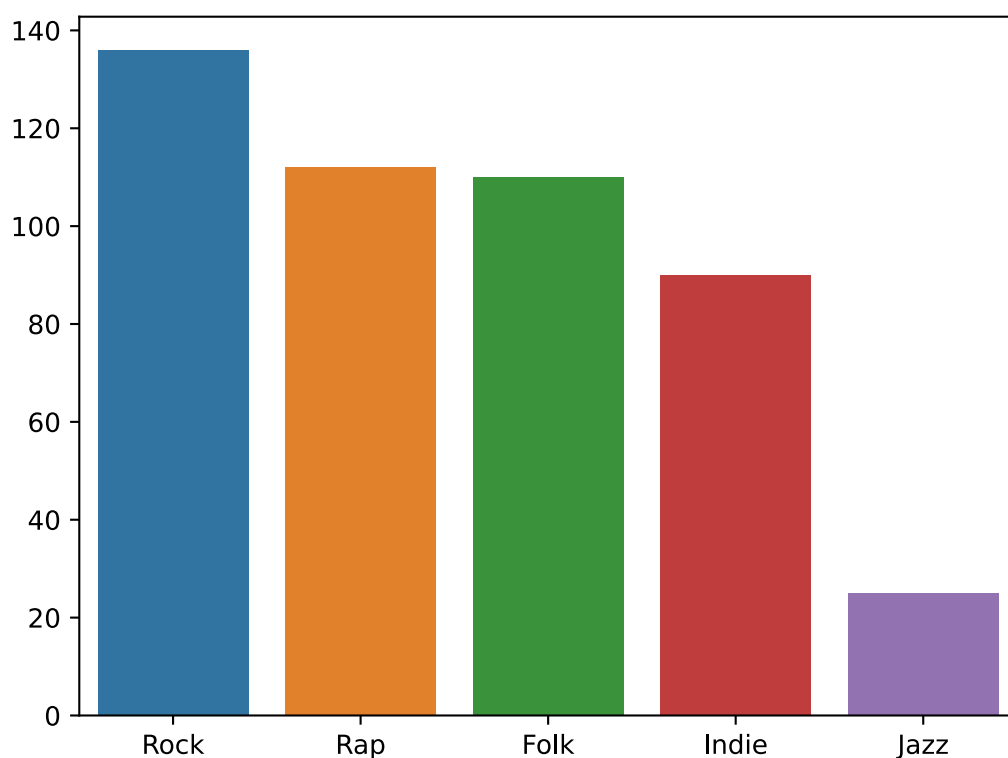
**3.0.2.3.5 Categorical Plots** Categorical plots are statistical graphs that help visualize the magnitudes of different variables in a dataset. A type of categorical plot is a bar chart, exactly like the example produced in the Matplotlib section. The categorical plots are `catplot` `striplot` `swarmplot` `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 Figure *catplot*



**Figure 12:** *catplot*

Figure *catplot*: Created from user Spotify data.

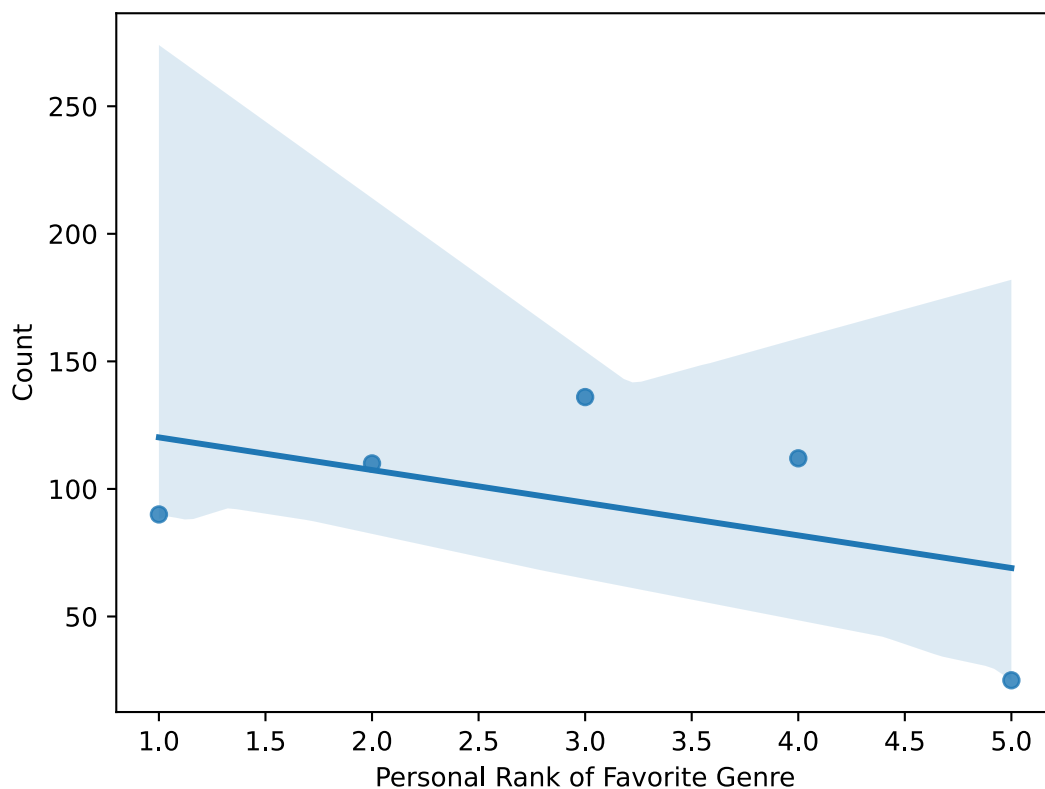
**3.0.2.3.6 Regression Plots** Regression plots are like relational plots in the way that they help visualize the relationship between two variables. Regression plots, however, show the linear correlation that may or may not exist in a scatter plot of data. Their regression plots are `lmpplot`, `regplot`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



**Figure 13:** *regplot*

Figure *regplot*: 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.

**3.0.2.3.7 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](#)

### 3.0.2.4 Sources

#### 3.0.2.4.1 Matplotlib

- <https://matplotlib.org/>
- [https://matplotlib.org/stable/api/pyplot\\_summary.html](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/>

#### 3.0.2.4.2 Seaborn

- <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>

#### 3.0.2.4.3 Bokeh

- [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/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>



### 3.0.3 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.

**3.0.3.1 Installation** To install Pandas, please use the command:

```
$ pip install pandas
```

**3.0.3.2 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
```

**3.0.3.3 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()

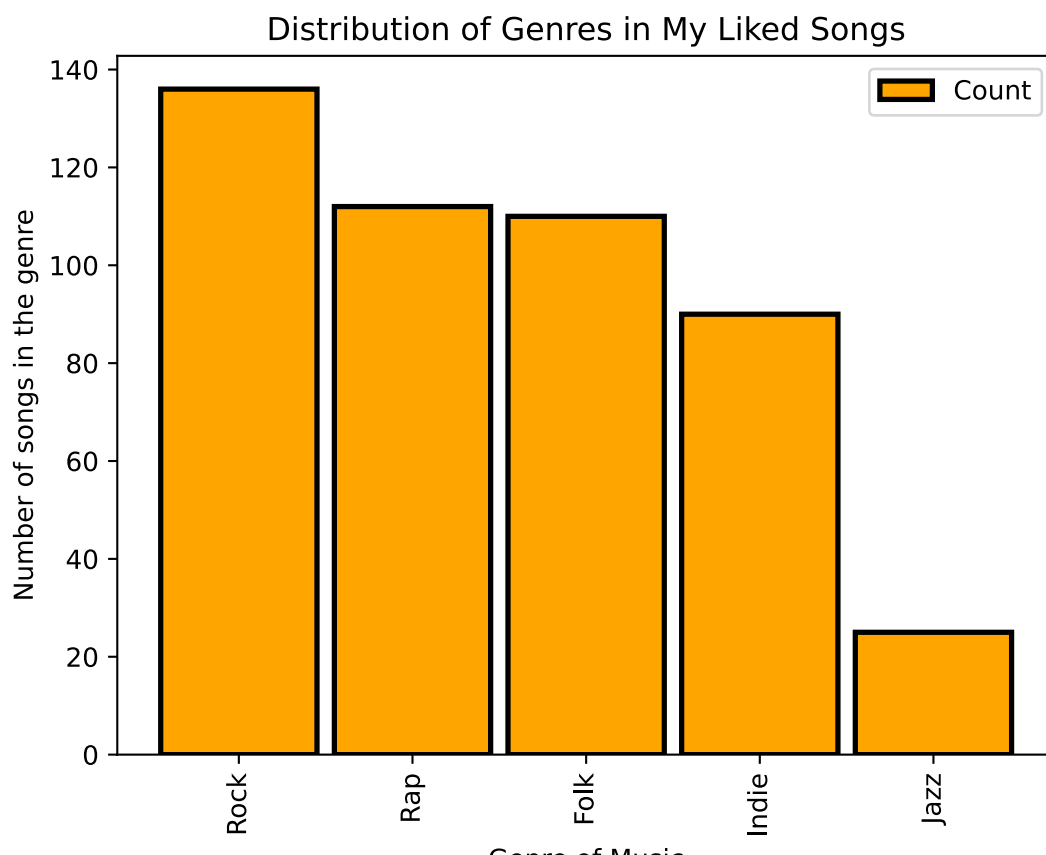
df = pd.DataFrame({'Count':count, 'Categories':categories})

# 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 Figure *barchart*.



**Figure 14:** barchart

Figure *barchart*: Barchart created from data from Spotify.

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.

**3.0.3.4 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)

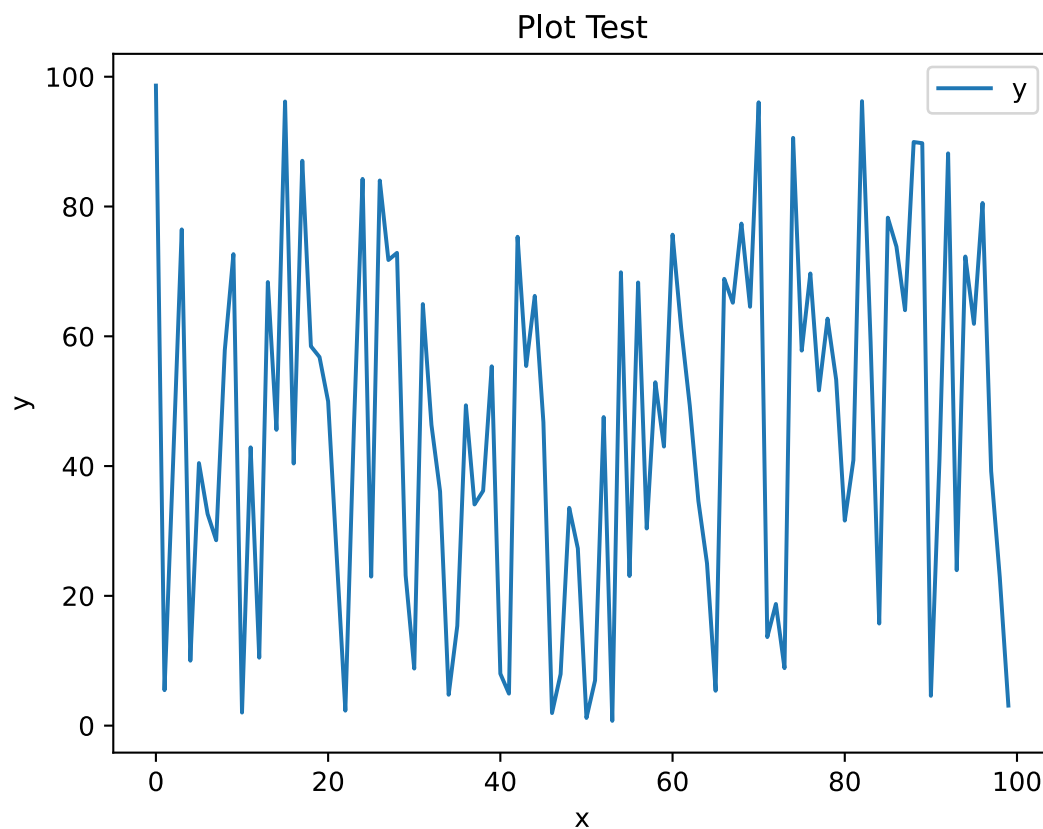
df = pd.DataFrame({'x':x, 'y':y})

# 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 Figure *linechart*.



**Figure 15:** linechart

Figure *linechart*: Barchart created from random number generation.

**3.0.3.5 Pie Chart** A pie chart is useful for showing a division 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.

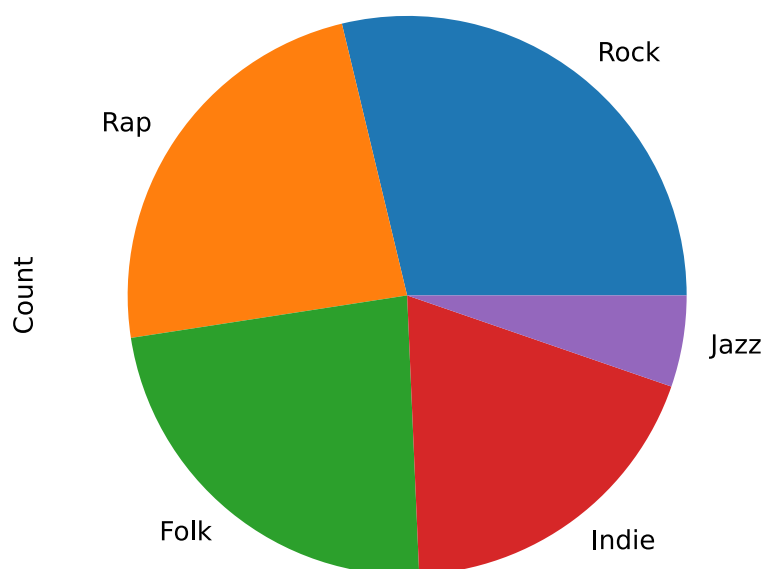
```
import matplotlib.pyplot as plt
import pandas as pd

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 Figure *piechart*.



**Figure 16:** piechart

Figure *piechart*: Barchart created from data from Spotify.

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()`.

**3.0.3.6 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(p):
    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(p)
```

This code can be accessed on [GitHub](#)

## 4 PYTHON

### 4.0.1 Cloudmesh 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 from the docstring. Then we can save it to a file, so we could autogenerate the md file from the docstring.

### 4.0.2 cms sys command generate

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.

### 4.0.3 Linux

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

### 4.0.4 pandas

See if we have already a chapter Pandas in the book and learn about it. Improve or add a new features that you found.

### 4.0.5 Python

find a topic that is not yet in the book and create a description for others. Do not duplicate efforts. Create a pull request with your contribution.

**4.0.5.1 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. The first is FIFO, which stands for first in, first out. This means that the first element that is added to the data structure is the first element that is pulled from the data structure. The second type is LIFO, which stands for Last in, First out. This means that the element that was last added to the queue is the element that will be removed first. Finally, there is a priority queue. In a priority queue, the element that is removed first is the element with the lowest value for a priority (meaning the highest priority).

Fortunately, python has a built-in queue module. To access, simply type:

```
import queue
```

Then, the data structures can be built as follows:

#### 4.0.5.1.1 FIFO Queue

#### 4.0.5.1.2 LIFO Queue

#### 4.0.5.1.3 Priority Queue



#### 4.0.6 FastAPI

FastAPI is a framework in python that allows developers to use restinterface to call function that implement application, it uses RestAPI to call the common building block of an application.

**4.0.6.1 FastAPI Install** To install the FastAPI they are two steps one might need to install it fully with the `uvicorn` or install both the `FastAPI` and the `uvicorn` by part

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

**4.0.6.2 FastAPI Example** The simplest FastAPI file could look like this:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Copy that to a file `main.py`

Run the live server:

```
$ uvicorn main:app --reload

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

In the output, there's a line with something like:

```
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to
    ↪ quit)
```

That line shows the URL where your app is being served, in your local machine.

Check it

Open your browser at <http://127.0.0.1:8000>

You will see the JSON response as:

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

Interactive API docs

- Go to <http://127.0.0.1:8000/docs>.
- Then go to <http://127.0.0.1:8000/redoc>.
- OpenAPI:

FastAPI generates a `schema` with all your API using the `OpenAPI` standard for defining APIs.

- OpenAPI and JSON Schema

OpenAPI defines an API schema for your API. And that schema includes definitions (or “schemas”) of the data sent and received by your API using JSON Schema, the standard for JSON data schemas.

You can see it directly at: <http://127.0.0.1:8000/openapi.json>.

It will show a JSON starting with something like:

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "FastAPI",
    "version": "0.1.0"
  },
  "paths": {
    "/items/": {
      "get": {
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": {
              "application/json": {
```

#### 4.0.6.2.1 Step 2: create a FastAPI instance

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Here the `app` variable will be an “instance” of the class `FastAPI`.

This will be the main point of interaction to create all your API.

This `app` is the same one referred by `uvicorn` in the command:

```
$ uvicorn main:app --reload

INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to
    ↪ quit)
```

If you create your app like:

```
from fastapi import FastAPI

my_awesome_api = FastAPI()

@my_awesome_api.get("/")
async def root():
    return {"message": "Hello World"}
```

And put it in a file `main.py` then you would call `uvicorn` like:

```
$ uvicorn main:my_awesome_api --reload

INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to
    ↪ quit)
```

**4.0.6.2.2 Step 3: create a path operation** Path here refers to the last part of the URL starting from the first .

So, in a URL like: `https://example.com/items/foo` ...the path would be:

`/items/foo`

” While building an API, the “path” is the main way to separate “concerns” and “resources”.

Operation “Operation” here refers to one of the HTTP “methods”.

When building APIs, you normally use these specific HTTP methods to perform a specific action. Normally you use: \* POST: to create data. \* GET: to read data. \* PUT: to update data. \* DELETE: to delete data.

In the HTTP protocol, you can communicate to each path using one (or more) of these “methods”.

So, in OpenAPI, each of the HTTP methods is called an “operation”.

Define a path operation decorator

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

**4.0.6.2.3 Step 4: define the path operation function** This is our “path operation function”:

path: is /. operation: is get. function: is the function below the “decorator” (below **app.get?**(“/”)).

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

This is a Python function.

It will be called by FastAPI whenever it receives a request to the URL “/” using a GET operation.

In this case, it is an async function.

You could also define it as a normal function instead of async def:

```
from fastapi import FastAPI

app = FastAPI()

Reference: <https://fastapi.tiangolo.com/tutorial/first-steps/>
@app.get("/")
def root():
    return {"message": "Hello World"}
```

#### 4.0.6.2.4 Step 5: return the content

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

You can return a dict, list, singular values as str, int, etc.

You can also return Pydantic models .

There are many other objects and models that will be automatically converted to JSON (including ORMs, etc). Try using your favorite ones, it's highly probable that they are already supported.

References: <https://fastapi.tiangolo.com/tutorial/first-steps/>

## 5 RIVANNA

### 5.0.1 Rivanna

Logging in to Rivanna via web interface

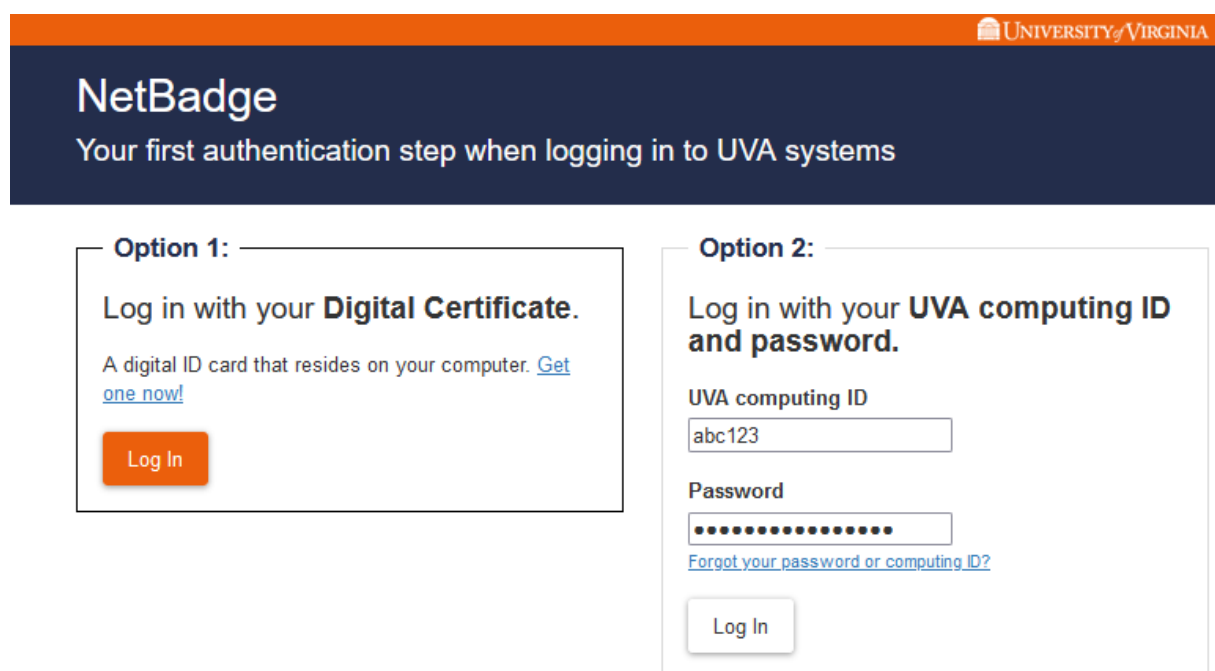
Documentation: <https://www.rc.virginia.edu/userinfo/rivanna/login/#web-based-access>

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



**NetBadge**  
Your first authentication step when logging in to UVA systems

**Option 1:**

Log in with your **Digital Certificate**.

A digital ID card that resides on your computer. [Get one now!](#)

Log In

**Option 2:**

Log in with your **UVA computing ID and password**.

UVA computing ID  
abc123

Password  
.....

[Forgot your password or computing ID?](#)

Log In

**Figure 17:** 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$
```

**5.0.1.1 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.

**5.0.1.2 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)
```

#### 5.0.1.2.1 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
```

```
Wed Jun 1 17:15:49 2022
```

```
+-----+
| 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		
+-----+					
↪					

### 5.0.1.3 SSH Config

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

## 5.0.2 Run Python MPI programs on Rivanna

see the book Python MPI

add chapter if not there

## 6 AI

### 6.0.1 DL Timeseries

create and document a 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.

## 7 BIOS

### 7.0.1 Bios

Please add here a 2-3 paragraph professional Bio. Look up who a professional bio is in 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).

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

**7.0.1.2 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.

**7.0.1.3 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.

**7.0.1.4 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.



**Figure 18:** Jacques's Picture

**7.0.1.5 Jacques Fleischer** 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.

**7.0.1.6 Eric He** Junyang (Eric) He is a rising second year majoring in Computer Science at the University of Virginia. He will 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.

**7.0.1.7 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.

## 8 REFERENCES

