UNIT 3
Visual Analytics

By

Mrs. Vaishali Baviskar

Contents

Spatial data

- **Spatial data**, also known as **geospatial data, GIS data, or geodata,** is a type of numeric data that defines the geographic location of a physical object, such as a building, a street, a town, a city, a country, or other physical objects, using a geographic coordinate system. You may determine not just the position of an object, but also its length, size, area, and shape using spatial data.

Spatial data

- To work with geospatial data in python we need the **GeoPandas & GeoPlot library**
- GeoPandas is an open-source project to make working with geospatial data in python easier. GeoPandas extends the data types used by pandas to allow spatial operations on geometric types. Geometric operations are performed shapely. Geopandas further depends on fiona for file access and matplotlib for plotting. GeoPandas depends on its spatial functionality on a large geospatial, open-source stack of libraries (GEOS, GDAL, and PROJ).

Spatial data

- Geoplot is a geospatial data visualization library for data scientists and geospatial analysts that want to get things done quickly. Below we'll cover the basics of Geoplot and explore how it's applied. Geoplot is for Python 3.6+ versions only.

Spatial data- Reading Shapefile

- First, we will import the geopandas library and then read our shapefile using the variable “world_data”. Geopandas can read almost any vector-based spatial data format including ESRI shapefile, GeoJSON files and more using the command:

Spatial data- Reading Shapefile

- ***Syntax:*** *geopandas.read_file()*
- ***Parameters***
- *filename: str, path object, or file-like object. Either the absolute or relative path to the file or URL to be opened or any object with a read() method (such as an open file or StringIO)*
- *bbox: tuple | GeoDataFrame or GeoSeries | shapely Geometry, default None. Filter features by given bounding box, GeoSeries, GeoDataFrame or a shapely geometry. CRS mis-matches are resolved if given a GeoSeries or GeoDataFrame. Cannot be used with mask.*

Spatial data- Reading Shapefile

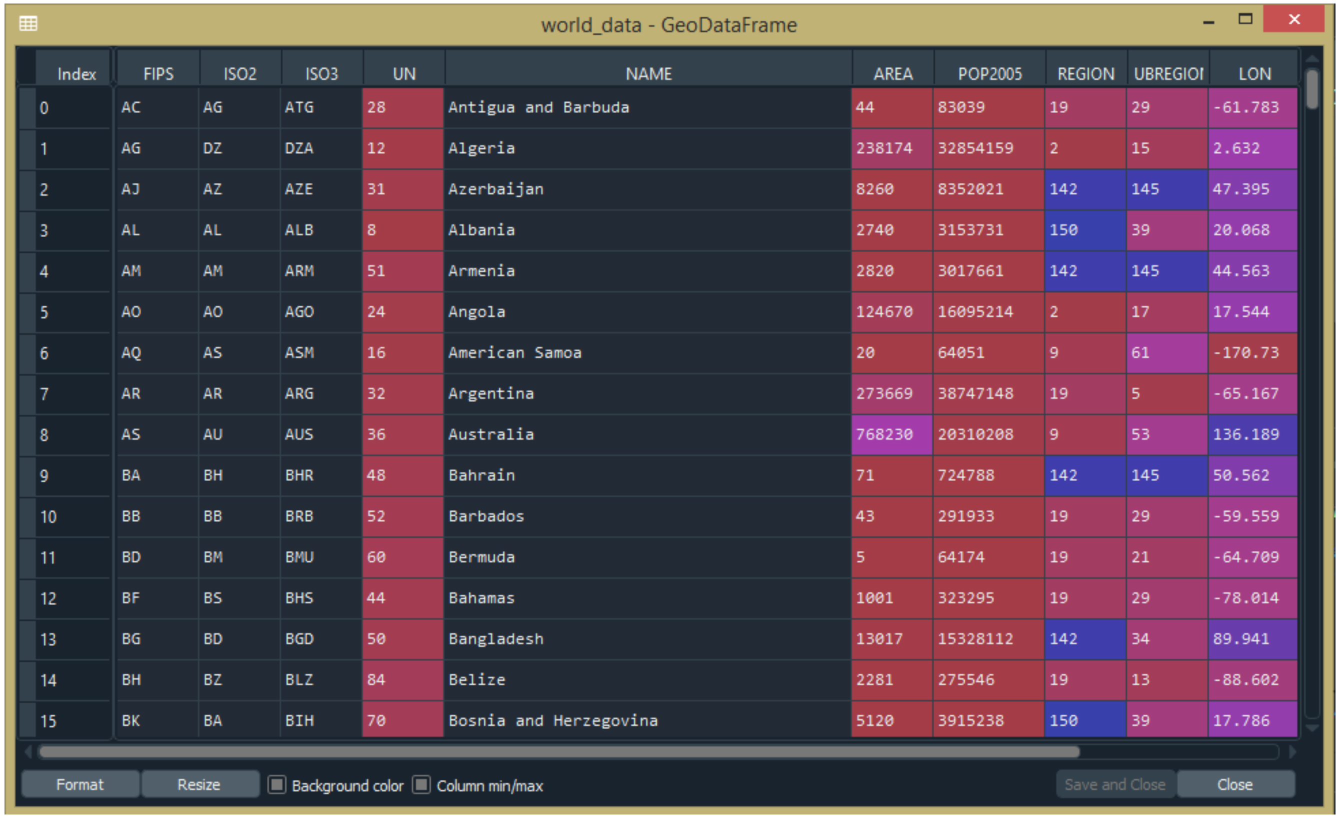
- *mask: dict | GeoDataFrame or GeoSeries | shapely Geometry, default None. Filter for features that intersect with the given dict-like geojson geometry, GeoSeries, GeoDataFrame or shapely geometry. CRS mis-matches are resolved if given a GeoSeries or GeoDataFrame. Cannot be used with bbox.*
- *rows: int or slice, default None. Load in specific rows by passing an integer (first n rows) or a slice() object.*
- ***kwargs : Keyword args to be passed to the open or BytesCollection method in the fiona library when opening the file. For more information on possible keywords, type: import fiona; help(fiona.open)*

Spatial data- Reading Shapefile

- import geopandas as gpd

- # Reading the world shapefile
- world_data = gpd.read_file(r'world.shp')

- world_data



Index	FIPS	ISO2	ISO3	UN	NAME	AREA	POP2005	REGION	UBREGIOI	LON
0	AC	AG	ATG	28	Antigua and Barbuda	44	83039	19	29	-61.783
1	AG	DZ	DZA	12	Algeria	238174	32854159	2	15	2.632
2	AJ	AZ	AZE	31	Azerbaijan	8260	8352021	142	145	47.395
3	AL	AL	ALB	8	Albania	2740	3153731	150	39	20.068
4	AM	AM	ARM	51	Armenia	2820	3017661	142	145	44.563
5	AO	AO	AGO	24	Angola	124670	16095214	2	17	17.544
6	AQ	AS	ASM	16	American Samoa	20	64051	9	61	-170.73
7	AR	AR	ARG	32	Argentina	273669	38747148	19	5	-65.167
8	AS	AU	AUS	36	Australia	768230	20310208	9	53	136.189
9	BA	BH	BHR	48	Bahrain	71	724788	142	145	50.562
10	BB	BB	BRB	52	Barbados	43	291933	19	29	-59.559
11	BD	BM	BMU	60	Bermuda	5	64174	19	21	-64.709
12	BF	BS	BHS	44	Bahamas	1001	323295	19	29	-78.014
13	BG	BD	BGD	50	Bangladesh	13017	15328112	142	34	89.941
14	BH	BZ	BLZ	84	Belize	2281	275546	19	13	-88.602
15	BK	BA	BIH	70	Bosnia and Herzegovina	5120	3915238	150	39	17.786

Spatial data- Plotting

- If we want to check which type of data you are using then go to the console and type “type(world_data)” which tells you that it’s not pandas data, it’s a geopandas geodata. Next, we are going to plot those GeoDataFrames using plot() method.

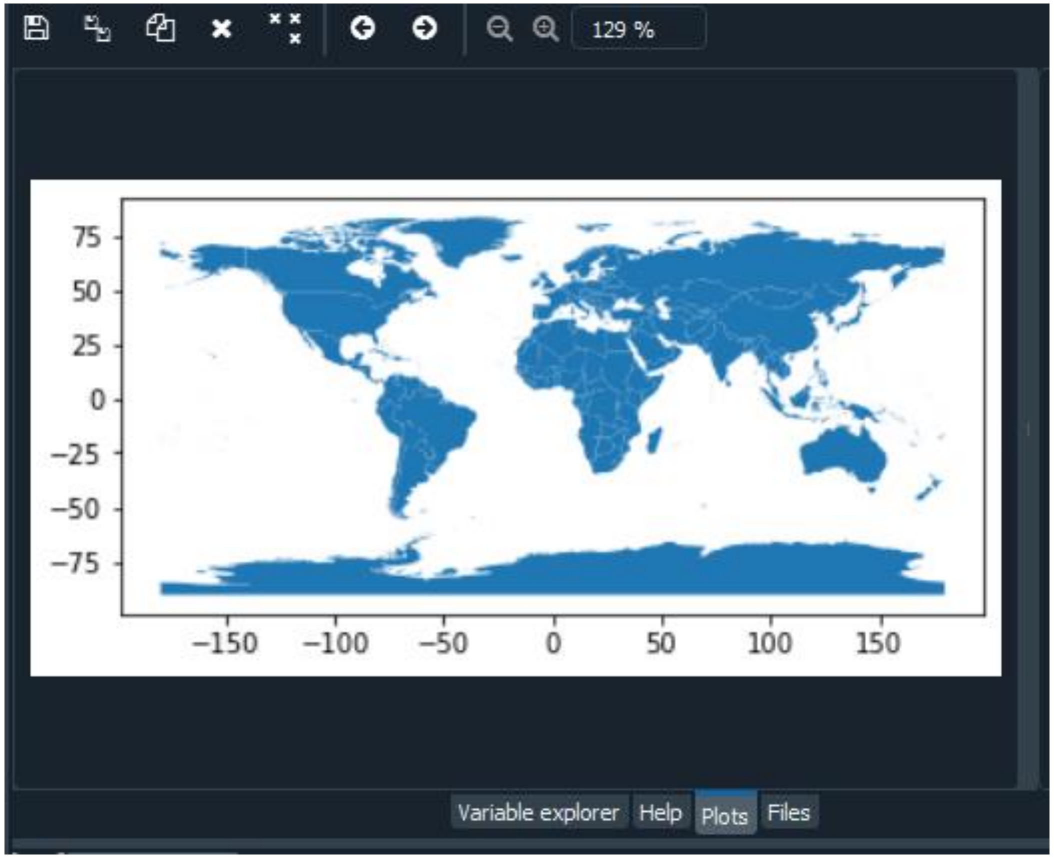
Spatial data- Reading Shapefile

- ***Syntax:*** *GeoDataFrame.plot()*

import geopandas as gpd

- # Reading the world shapefile
- world_data = gpd.read_file(r'world.shp')

world_data.plot()



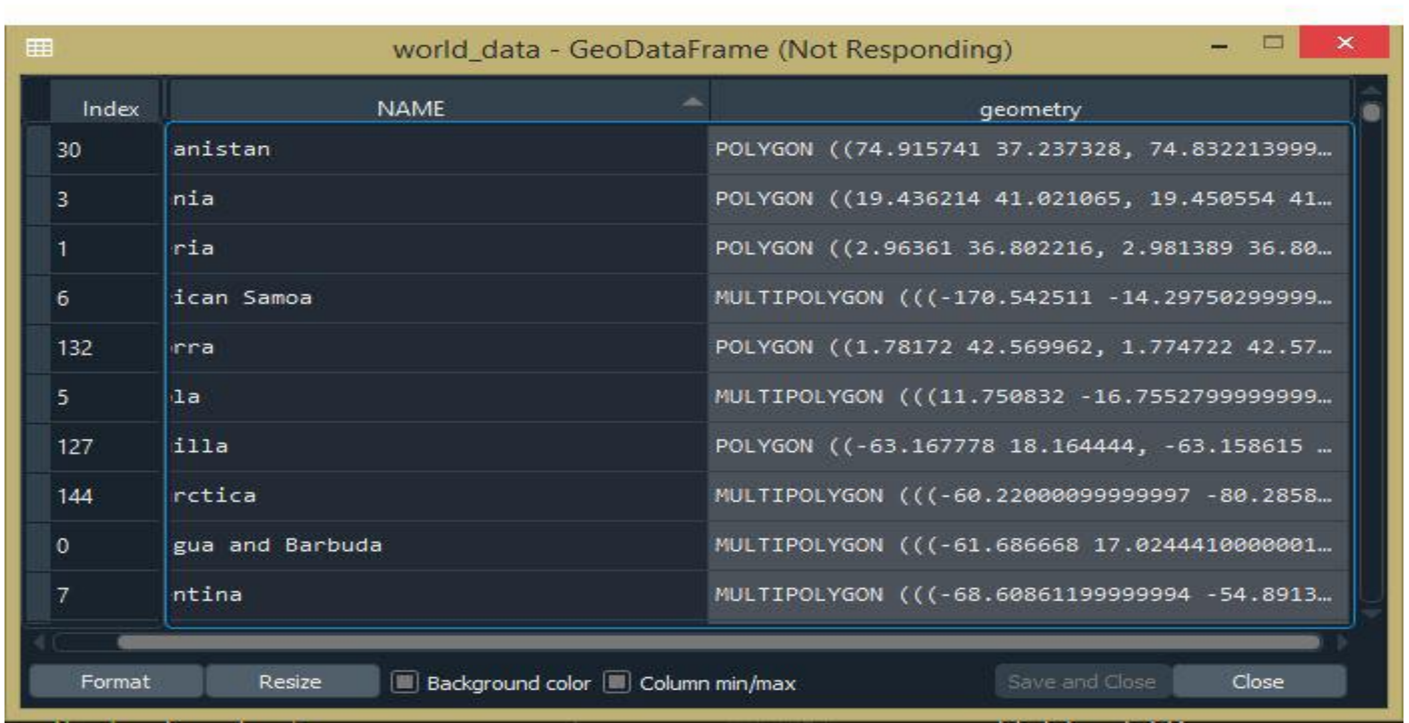
Spatial data- Selecting Columns

- If we see the “world_data” GeoDataFrame there are many columns(Geoseries) shown, you can choose specific Geoseries by:
- **Syntax:**

data[[‘attribute 1’, ‘attribute 2’]]

Spatial data- Selecting Columns

- import geopandas as gpd
- # Reading the world shapefile
- world_data = gpd.read_file(r'world.shp')
- world_data = world_data[['NAME', 'geometry']]



Index	NAME	geometry
30	anistan	POLYGON ((74.915741 37.237328, 74.832213999…
3	nia	POLYGON ((19.436214 41.021065, 19.450554 41…
1	ria	POLYGON ((2.96361 36.802216, 2.981389 36.80…
6	ican Samoa	MULTIPOLYGON (((-170.542511 -14.29750299999…
132	rra	POLYGON ((1.78172 42.569962, 1.774722 42.57…
5	la	MULTIPOLYGON (((11.750832 -16.7552799999999…
127	illa	POLYGON ((-63.167778 18.164444, -63.158615 …
144	rctica	MULTIPOLYGON (((-60.22000099999997 -80.2858…
0	gua and Barbuda	MULTIPOLYGON (((-61.686668 17.0244410000001…
7	ntina	MULTIPOLYGON (((-68.60861199999994 -54.8913…

Spatial data- Calculating Area

- We can calculate the area of each country using geopandas by creating a new column “area” and using the area property.

- **Syntax:**

GeoSeries.area

Returns a Series containing the area of each geometry in the GeoSeries expressed in the units of the CRS.

Spatial data- Calculating Area

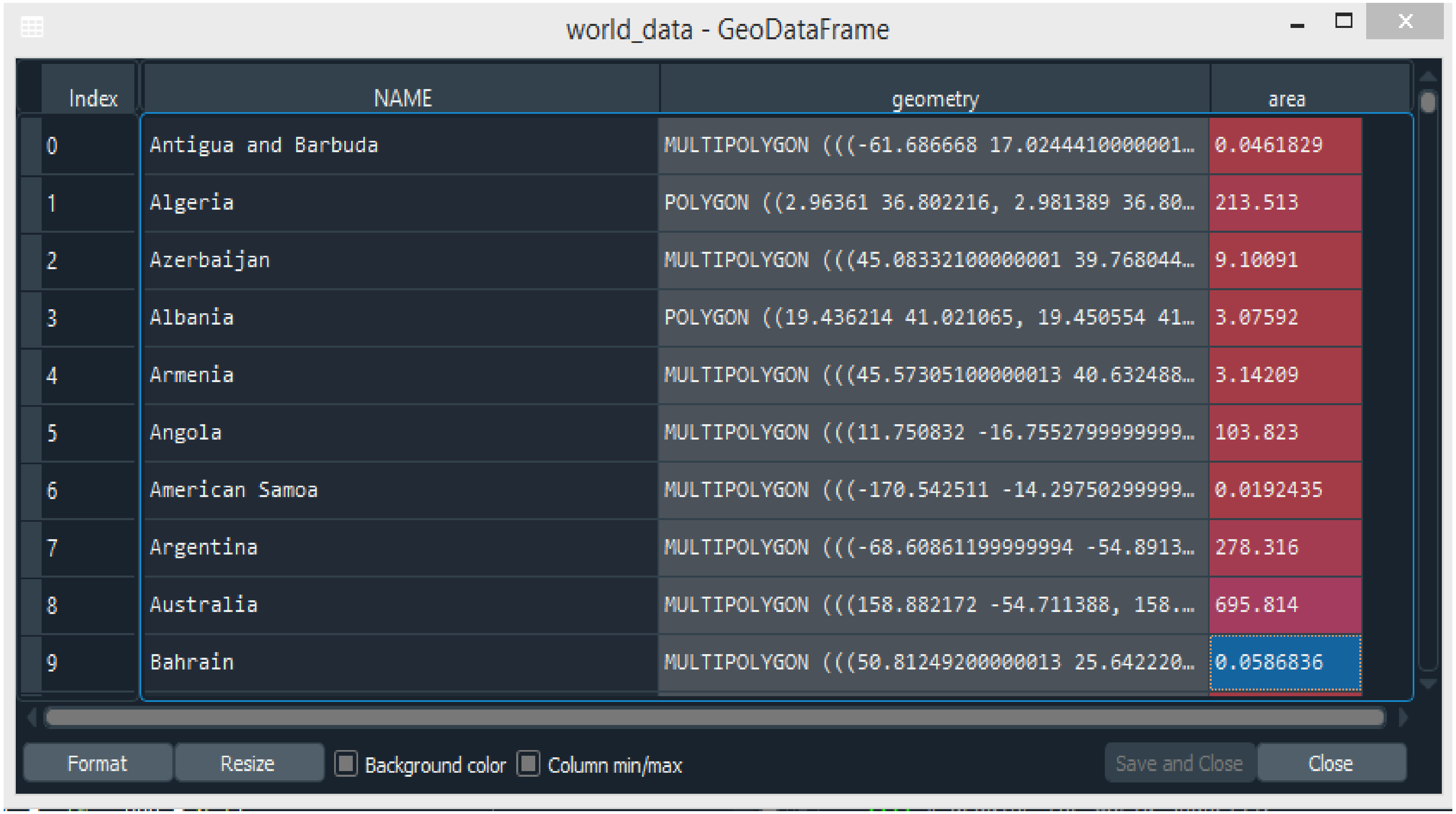
- import geopandas as gpd

- # Reading the world shapefile
- world_data = gpd.read_file(r'world.shp')

- world_data = world_data[['NAME', 'geometry']]

- # Calculating the area of each country
- world_data['area'] = world_data.area

Spatial data- Calculating Area



Index	NAME	geometry	area
0	Antigua and Barbuda	MULTIPOLYGON (((-61.686668 17.0244410000001…	0.0461829
1	Algeria	POLYGON ((2.96361 36.802216, 2.981389 36.80…	213.513
2	Azerbaijan	MULTIPOLYGON (((45.08332100000001 39.768044…	9.10091
3	Albania	POLYGON ((19.436214 41.021065, 19.450554 41…	3.07592
4	Armenia	MULTIPOLYGON (((45.57305100000013 40.632488…	3.14209
5	Angola	MULTIPOLYGON (((11.750832 -16.7552799999999…	103.823
6	American Samoa	MULTIPOLYGON (((-170.542511 -14.29750299999…	0.0192435
7	Argentina	MULTIPOLYGON (((-68.60861199999994 -54.8913…	278.316
8	Australia	MULTIPOLYGON (((158.882172 -54.711388, 158.…	695.814
9	Bahrain	MULTIPOLYGON (((50.81249200000013 25.642220…	0.0586836

Spatial data- Removing a Continent

We can remove a specific element from the Geoseries. Here we are removing the continent named “Antarctica” from the “Name” Geoseries.

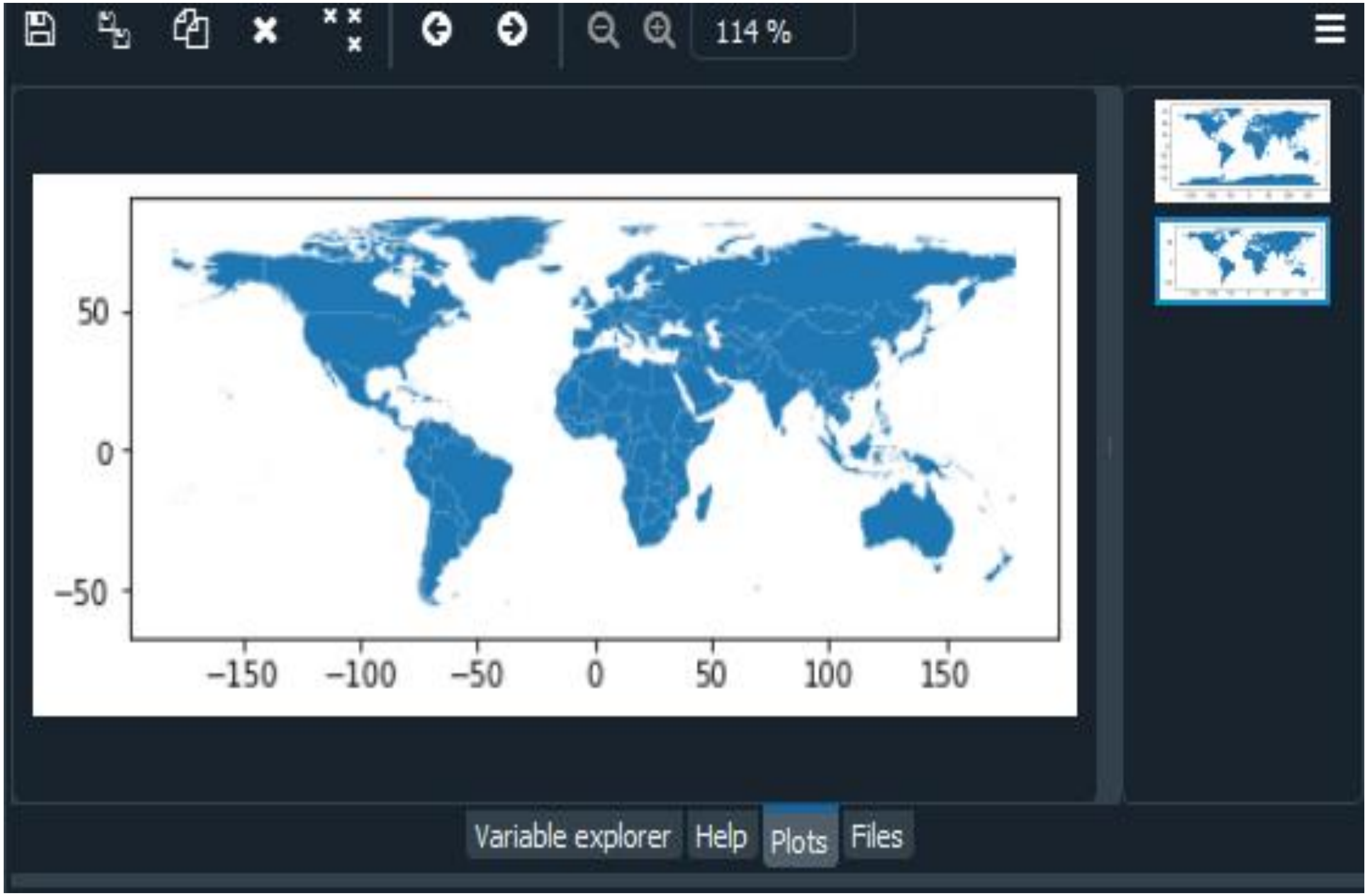
Syntax:

data[data[‘attribute’] != ‘element’]

Spatial data- Removing a Continent

```
import geopandas as gpd

# Reading the world shapefile
world_data = gpd.read_file(r'world.shp')

world_data = world_data[['NAME', 'geometry']]

# Calculating the area of each country
world_data['area'] = world_data.area

# Removing Antarctica from GeoPandas GeoDataframe
world_data = world_data[world_data['NAME'] != 'Antarctica']
world_data.plot()
```

Spatial data- Removing a Continent



Spatial data- Visualizing a specific country

We can visualize/plot a specific country by selecting it.
In the below example, we are selecting “India” from the “NAME” column.

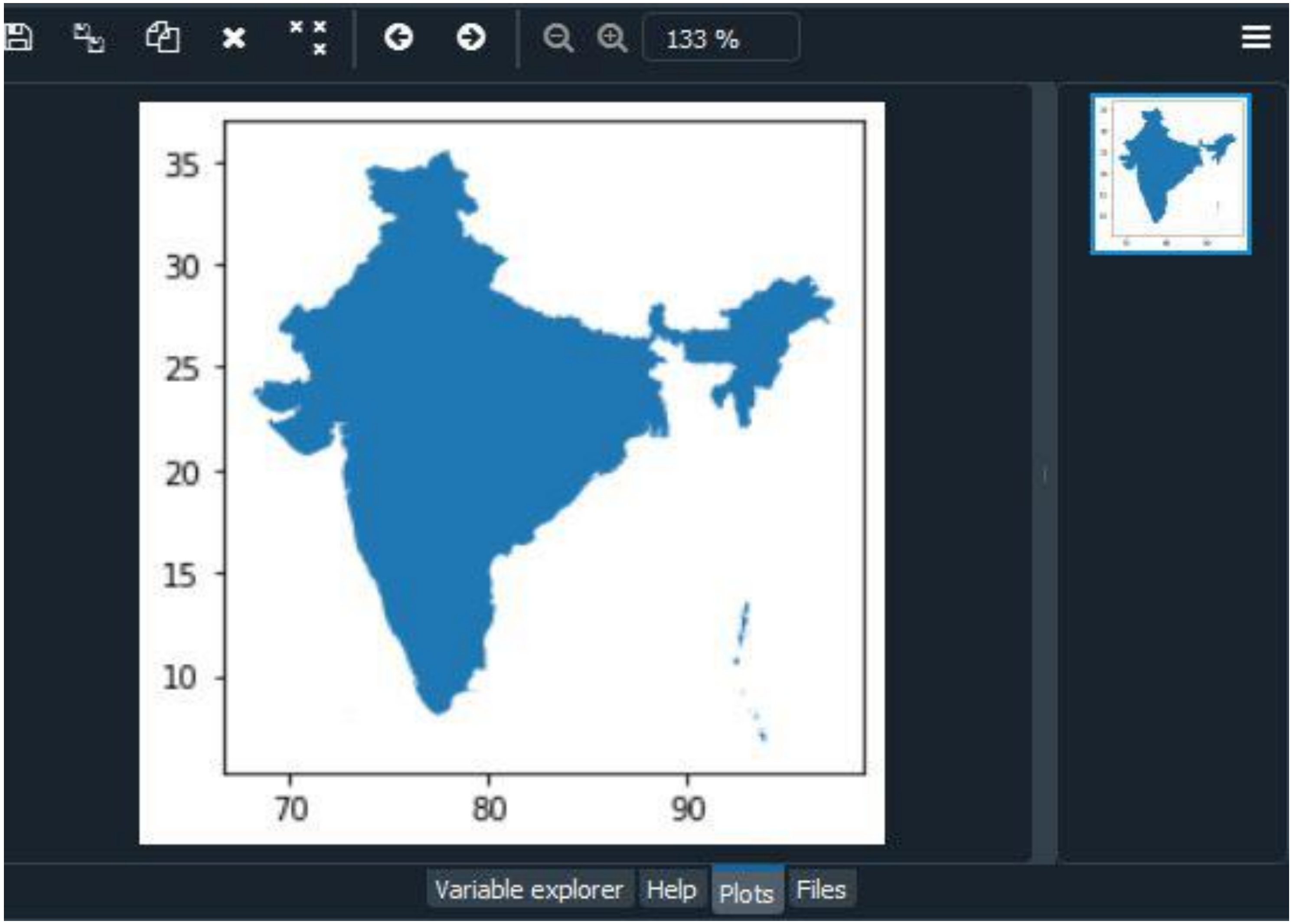
Syntax:

```
data[data.attribute==”element”].plot()
```

Spatial data- Visualizing a specific country

```
import geopandas as gpd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

# Reading the world shapefile
world_data = gpd.read_file(r'world.shp')
world_data = world_data[['NAME', 'geometry']]

# Calculating the area of each country
world_data['area'] = world_data.area

# Removing Antarctica from GeoPandas GeoDataframe
world_data = world_data[world_data['NAME'] != 'Antarctica']
world_data[world_data.NAME=="India"].plot()
```

Spatial data- Visualizing a specific country



References

- https://www.geeksforgeeks.org/working-with-geospatial-data-in-python/
- https://medium.com/@nelsonjoseph123/graph-visualization-using-python-bbd9a593c533

Mapcolor

- Data science is a multidisciplinary field that uses machine learning algorithms to analyze and interpret vast amounts of data.
- The combination of data science and machine learning has revolutionized how organizations make decisions and improve their operations.
- Matplotlib is a popular library in the Python ecosystem for visualizing the results of machine learning algorithms in a visually appealing way.
- It is a multi-platform library that can play with many operating systems and was built in 2002 by John Hunter.
- Colormaps or Cmap in Python are generated using Matplotlib, which we will discuss further.

Objectives

- Get introduced to Colormaps (Cmap) in Python.
- Familiarize yourself with the existing Colormaps in Matplotlib.
- Learn how to create and modify new and custom Cmaps in Python using Matplotlib.

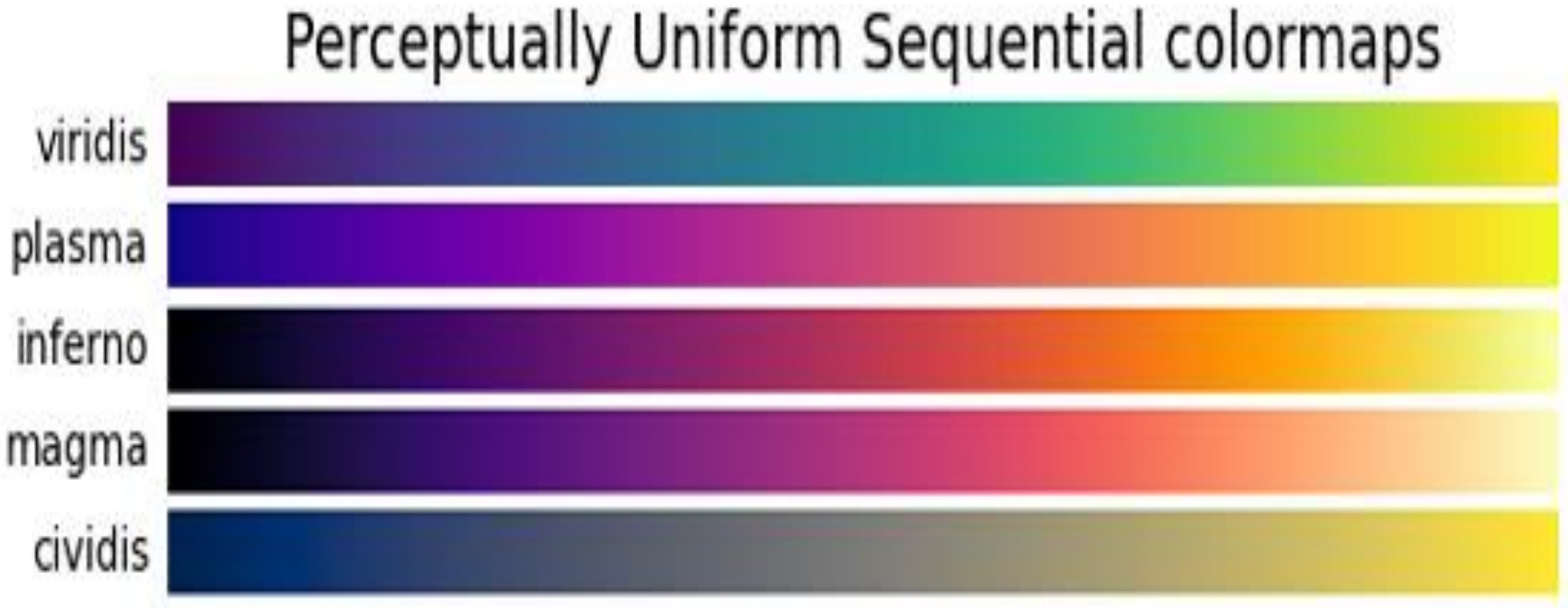
What Are Colormaps (Cmaps) in Matplotlib?

- In visualizing the 3D plot, we need colormaps to differ and make some intuitions in 3D parameters. Scientifically, the human brain perceives various intuitions based on the different colors they see.
- Nowadays, people have started to develop new Python packages with simpler and more modern styles than in Matplotlib, like Seaborn, Plotly, and even Pandas, using Matplotlib's API wrappers.
- But, Matplotlib is still in many programmers' hearts. Matplotlib is a popular data visualization library that provides several built-in colormaps and also allows users to create custom colormaps.

What Are Colormaps (Cmaps) in Matplotlib?

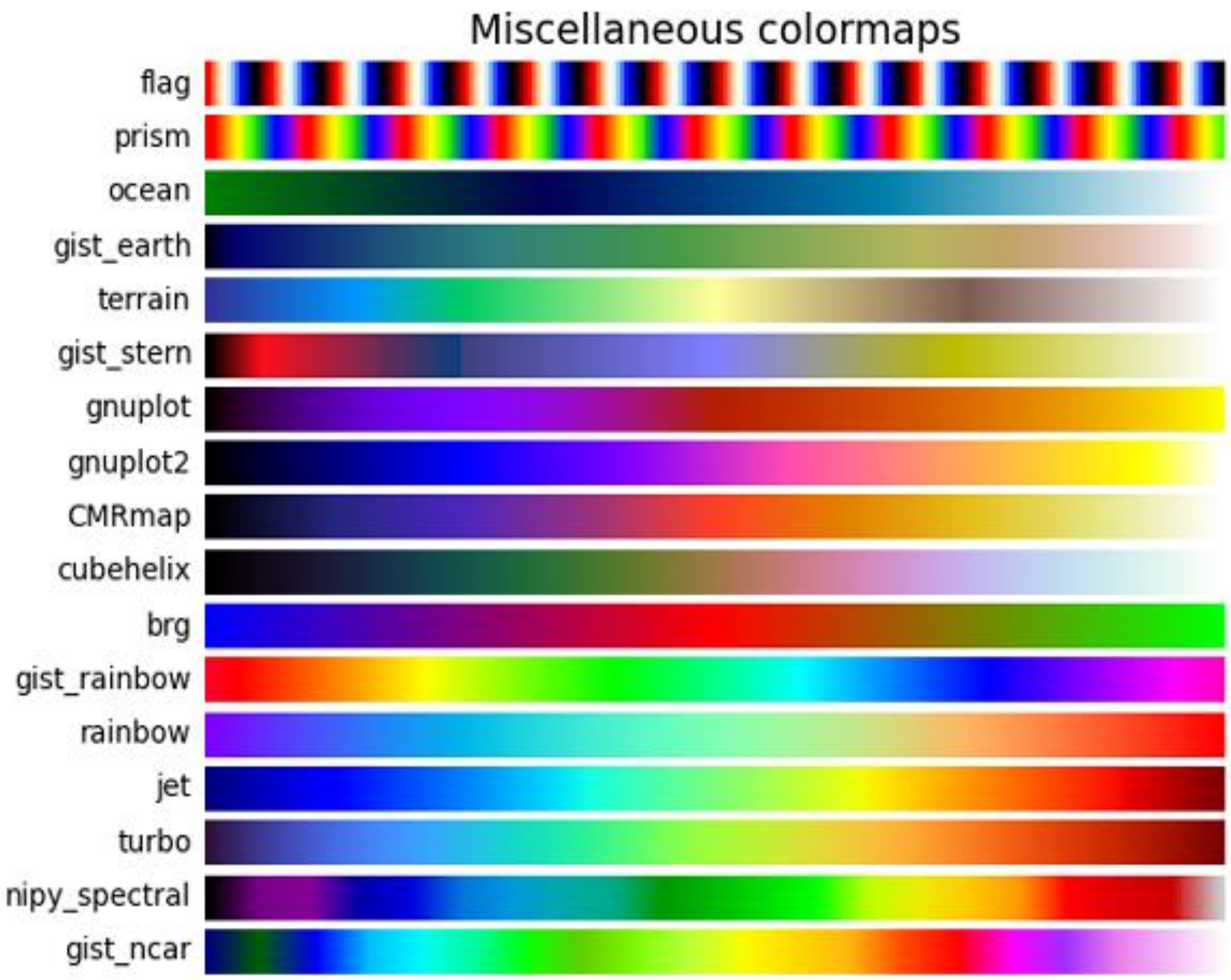
- Matplotlib allows for greater control and customization of the colors used in their visualizations.
- Python matplotlib provides some nice colormaps you can use, such as Sequential colormaps, Diverging colormaps, Cyclic colormaps, and Qualitative colormaps.

Sequential colormaps.

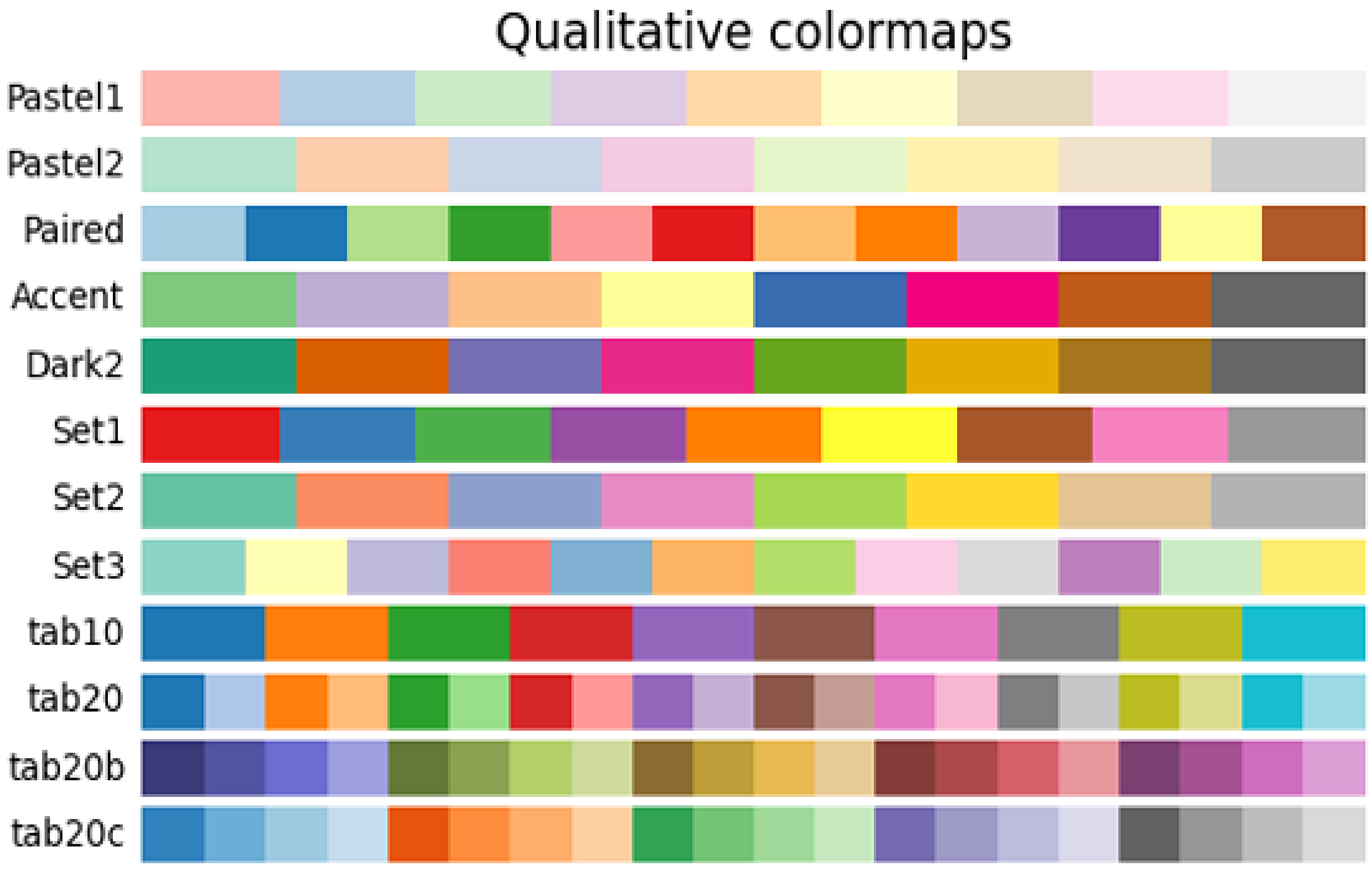


Matplotlib will give you **viridis** as a default colormaps.

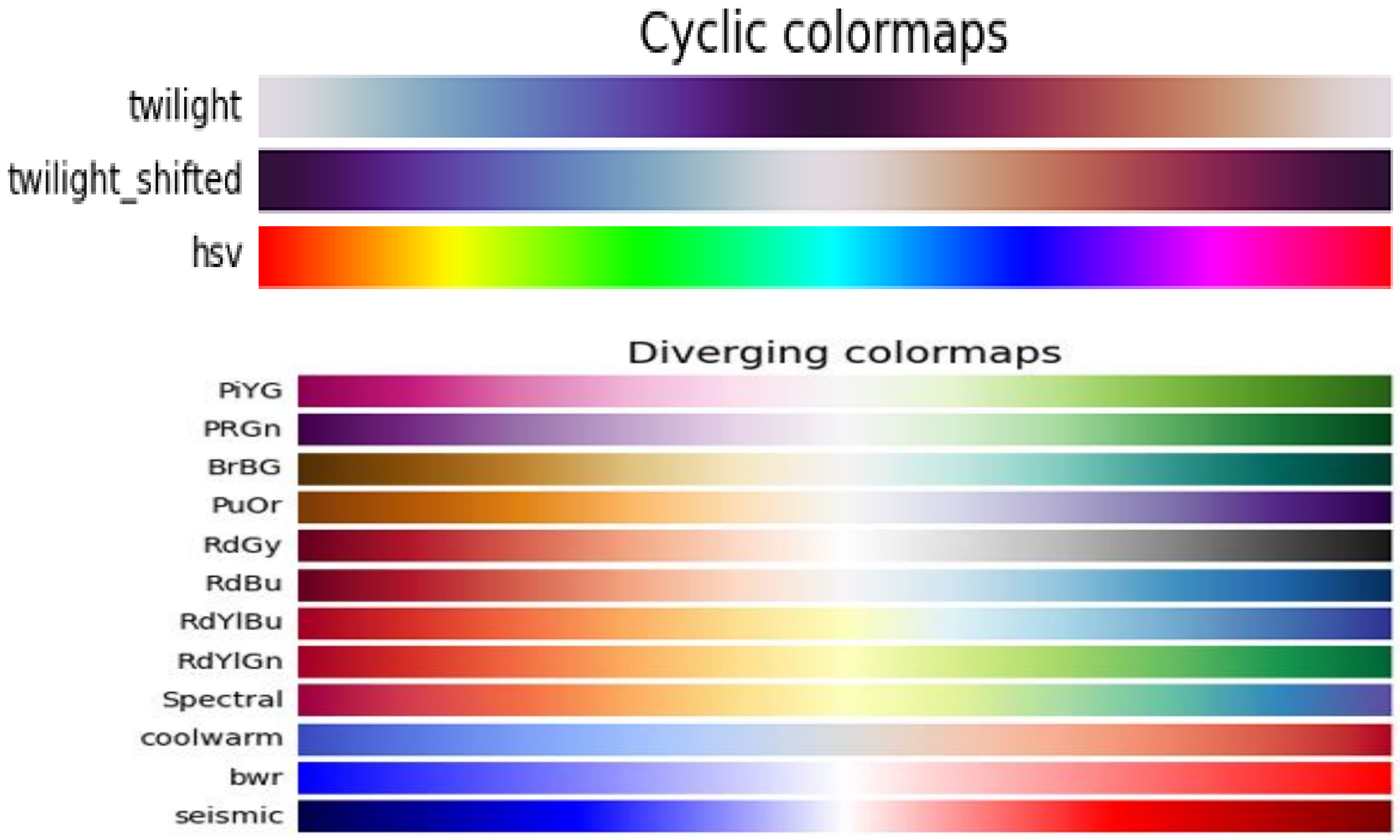
Diverging, Cyclic, Qualitative, and Misc colormaps in Matplotlib.



Diverging, Cyclic, Qualitative, and Misc colormaps in Matplotlib.



Diverging, Cyclic, Qualitative, and Misc colormaps in Matplotlib.



How to Create Subplots in Matplotlib and Apply Cmaps?

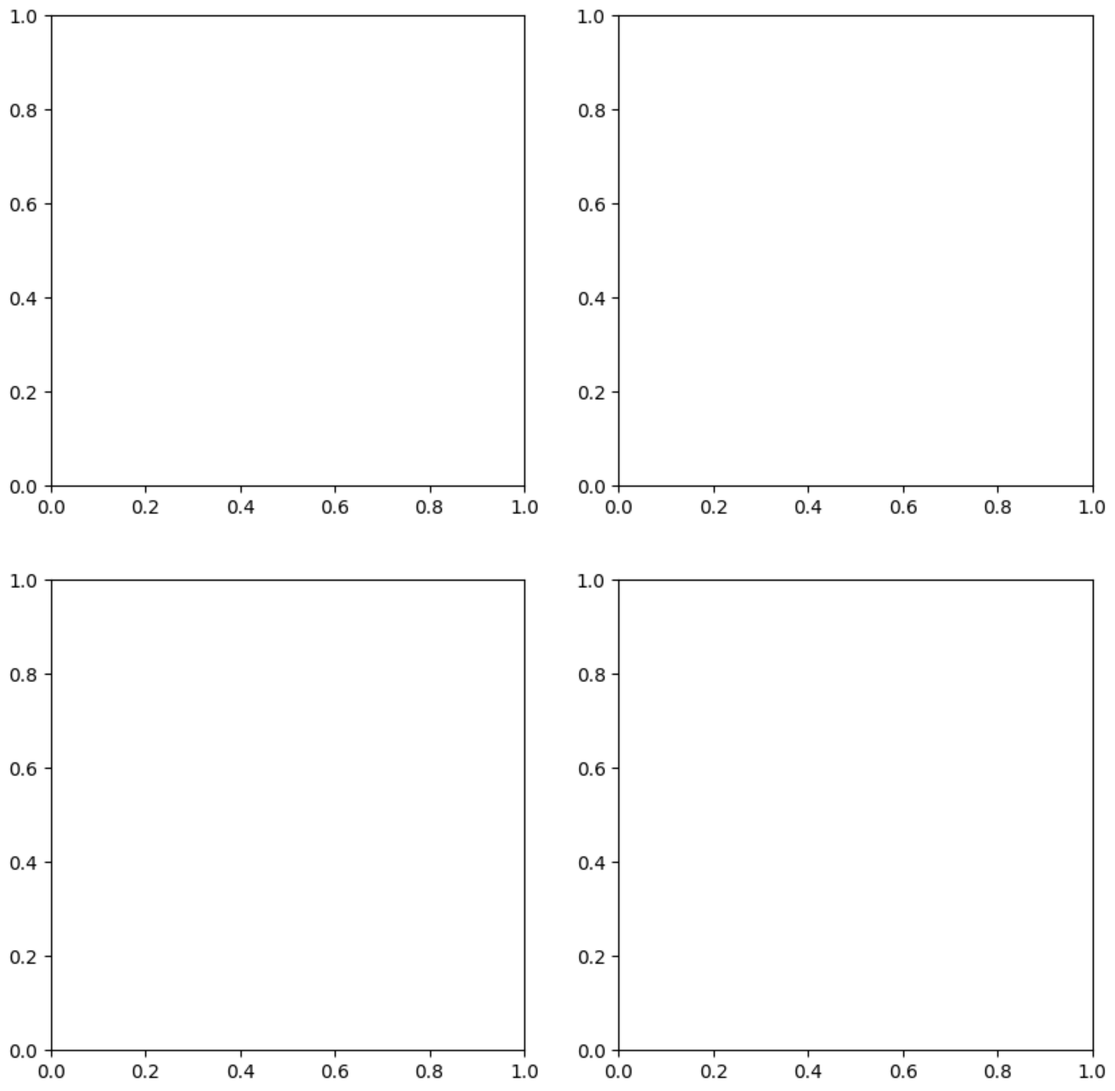
```
import matplotlib.pyplot as plt
import numpy as np
# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2,figsize=(10,10))
# Generate random dataset for each subplot
 for i in range(2):
           for j in range(2):
                      data = np.random.randn(100)
                    axs[i, j].hist(data, color='red', alpha=0.5) # Apply
a fancy colormap to the figure
 cmap = plt.get_cmap('hot')
 plt.set_cmap(cmap)
# Show the figure
plt.show()
```

How to Create Subplots in Matplotlib and Apply Cmaps?

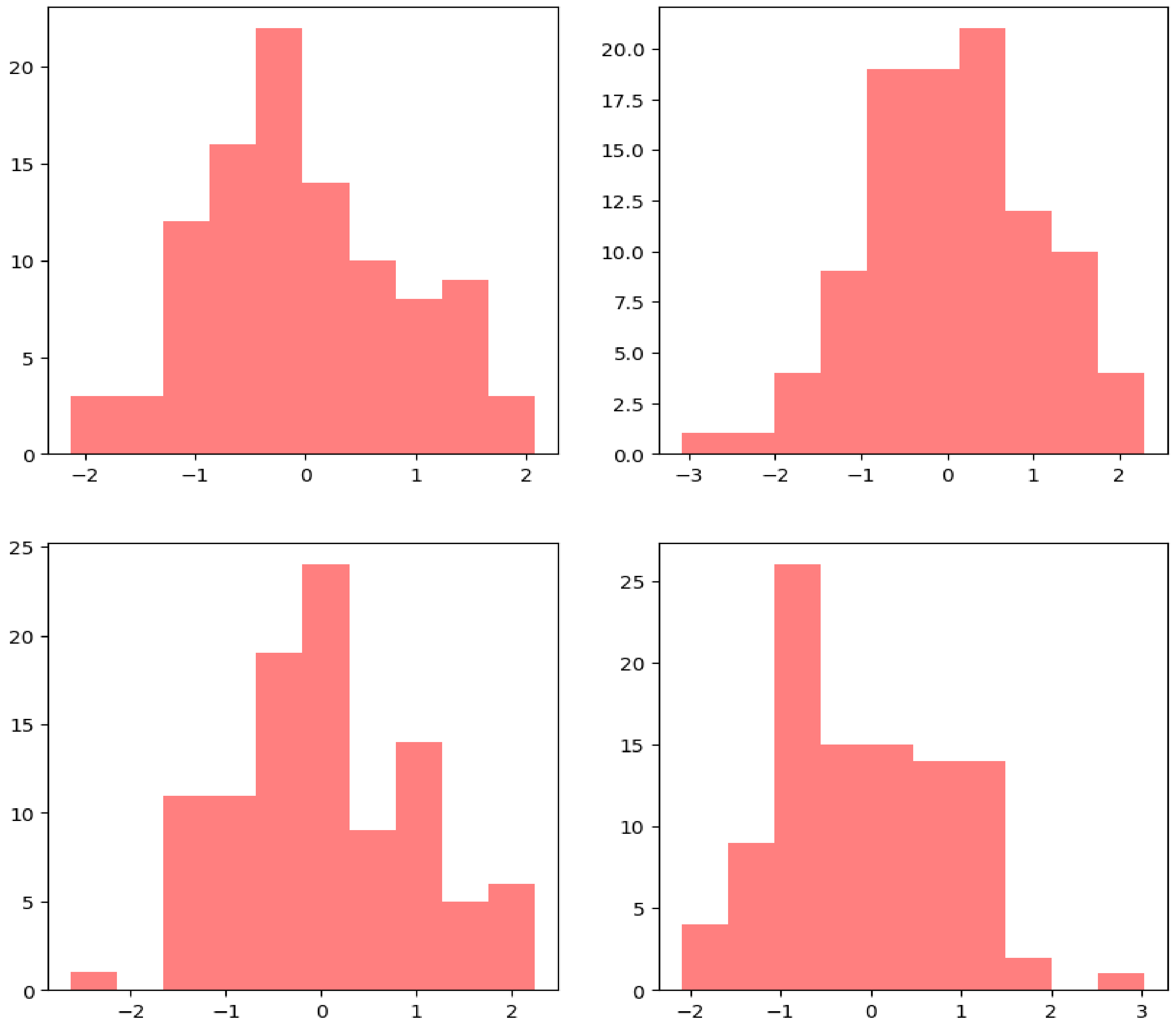
- This code creates a 2×2 grid of subplots, generates random data for each subplot
- plots a histogram of the data using the hist function.
- The subplots are then colored using a fancy colormap from the matplotlib library.
- In this example, the hot colormap is applied to the figure using the get_cmap and set_cmap functions.

How to Create Subplots in Matplotlib and Apply Cmaps?

```
# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2,figsize=(10,10))
```



How to Create Subplots in Matplotlib and Apply Cmaps?



How to Create Subplots in Matplotlib and Apply Cmaps?

uses the rdbu and rgba colormaps in matplotlib:

```
import matplotlib.pyplot as plt
 import numpy as np
# Generate dataset for the plot
x = np.linspace(-10, 10, 1000)
 y = np.sin(x)
# Plot the data
plt.plot(x, y, color='red')
# Use the rdbu colormap to color the background
plt.imshow(np.outer(np.ones(10), np.arange(100)),
 cmap='RdBu',
extent=(-10, 10, -1, 1), alpha=0.2)
 # Add text to the plot using an rgba color
plt.text(5, 0.5, 'Text in RGBA color', color=(0, 1, 0, 0.5), fontsize=16)
# Show the plot
plt.show()
```

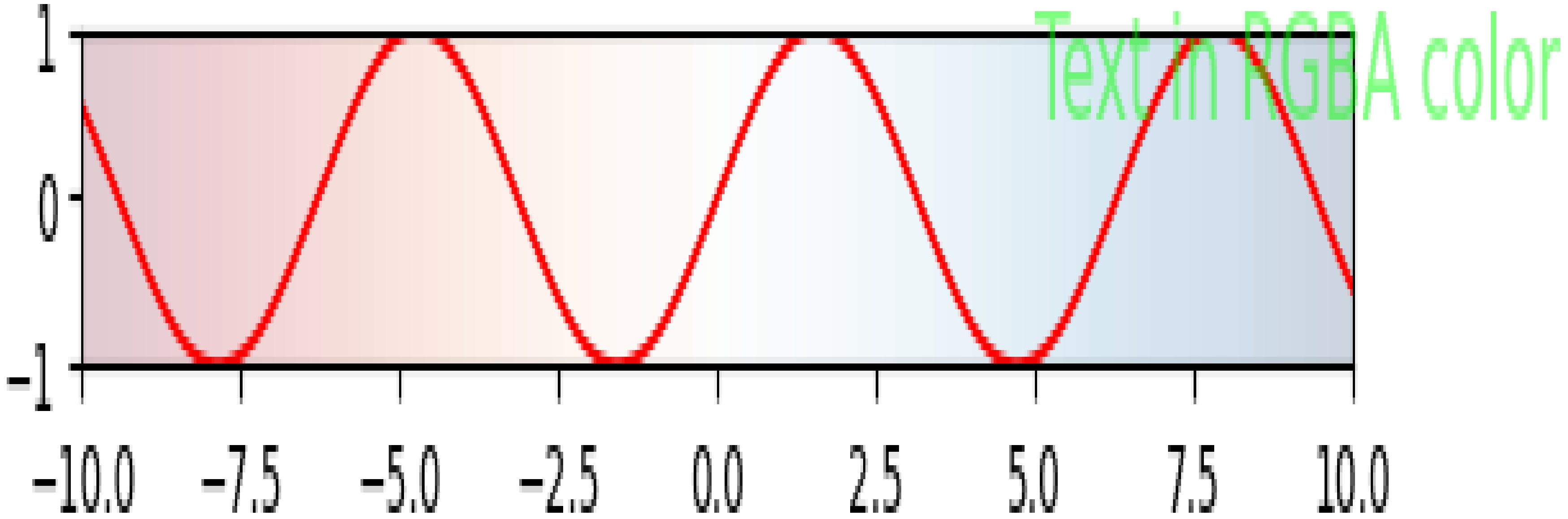
How to Create Subplots in Matplotlib and Apply Cmaps?

uses the rdbu and rgba colormaps in matplotlib:

- The above code generates data for a plot and plots it using the plot function.
- The background of the plot is then colored using the rdbu color map and the imshow function.
- The text is added to the plot using the text function and an rgba color, which allows you to specify the opacity of the color.
- Finally, the plot is displayed using the show function.
- **The** rgba **is used to define the colors using the Red-Green-Blue-Alpha (RGBA) model**.
- It is an extension of rgb() color values in CSS containing an alpha channel that specifies the transparency of color.

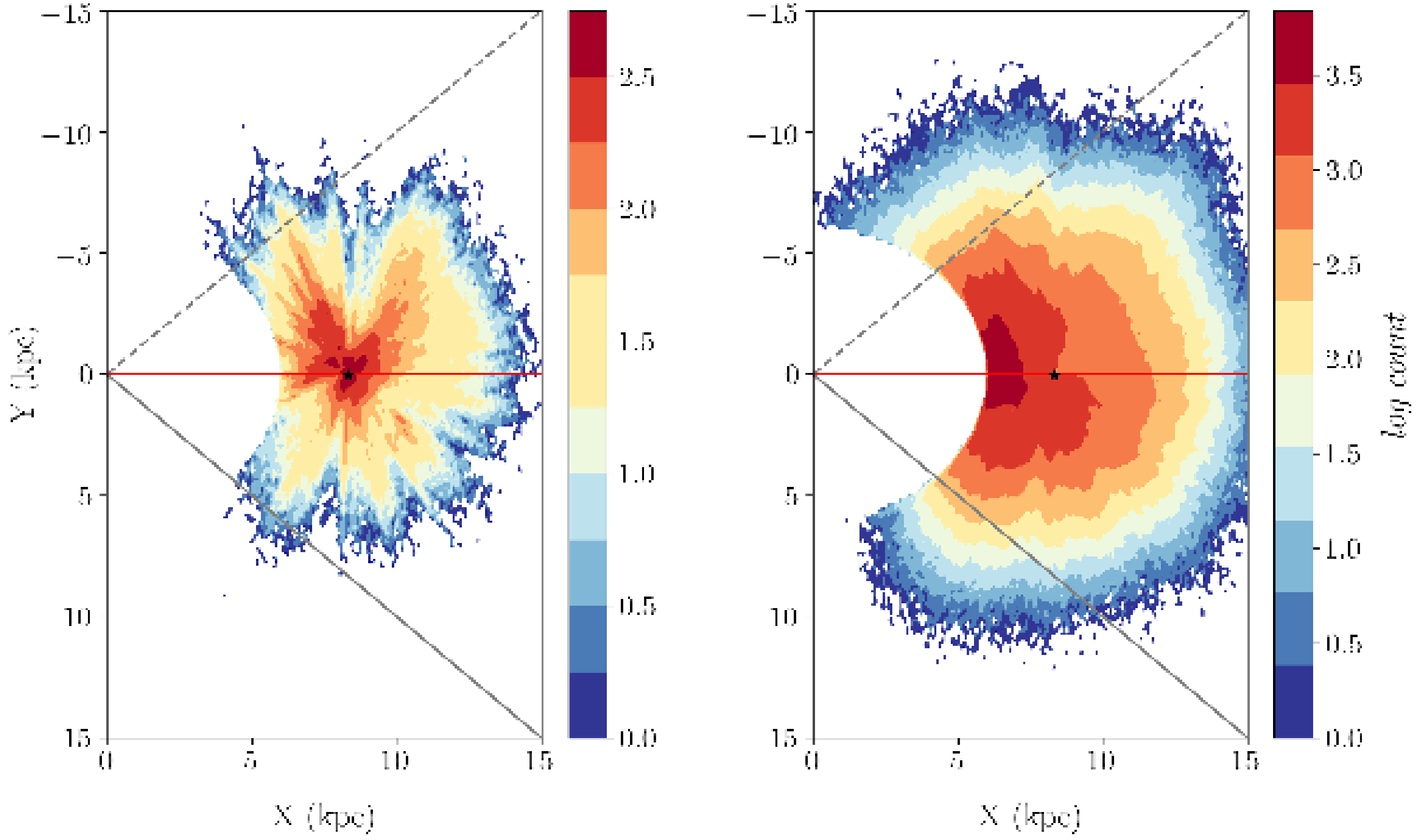
How to Create Subplots in Matplotlib and Apply Cmaps?

uses the rdbu and rgba colormaps in matplotlib:



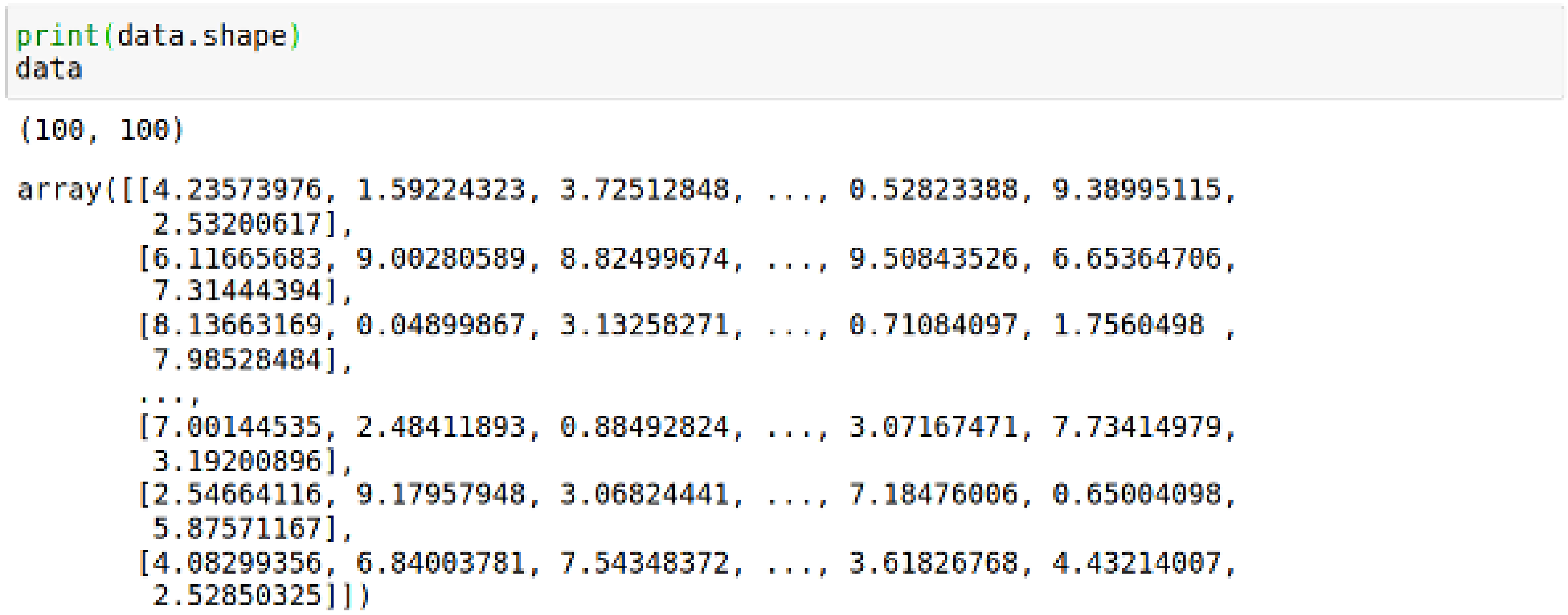
How to Create New Colormaps (Cmap) in Python

- an example of using colormaps. used the ‘**RdYlBu_r**’ colormaps to visualize my data.



How to Create New Colormaps (Cmap) in Python

- Let’s modify own colormaps.
- Firstly, we must create the mock data that will be visualized using this code.
- The data variable is an array that consists of 100 x 100 random numbers from 0–10. You can check it by writing this code



```
print(data.shape)
data
```

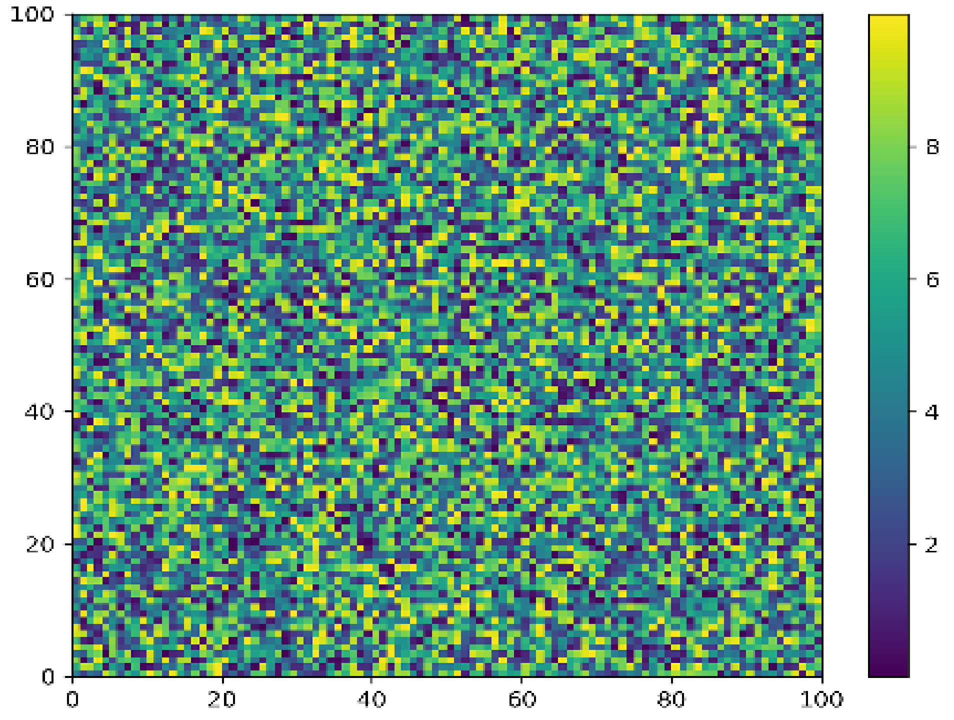
```
(100, 100)

array([[4.23573976, 1.59224323, 3.72512848, ..., 0.52823388, 9.38995115,
        2.53200617],
       [6.11665683, 9.00280589, 8.82499674, ..., 9.50843526, 6.65364706,
        7.31444394],
       [8.13663169, 0.04899867, 3.13258271, ..., 0.71084097, 1.7560498 ,
        7.98528484],
       ...,
       [7.00144535, 2.48411893, 0.88492824, ..., 3.07167471, 7.73414979,
        3.19200896],
       [2.54664116, 9.17957948, 3.06824441, ..., 7.18476006, 0.65004098,
        5.87571167],
       [4.08299356, 6.84003781, 7.54348372, ..., 3.61826768, 4.43214007,
        2.52850325]])
```

How to Create New Colormaps (Cmap) in Python

- Show the mock data with a default colormap using the simple code below.

plt.figure(figsize=(7, 6))

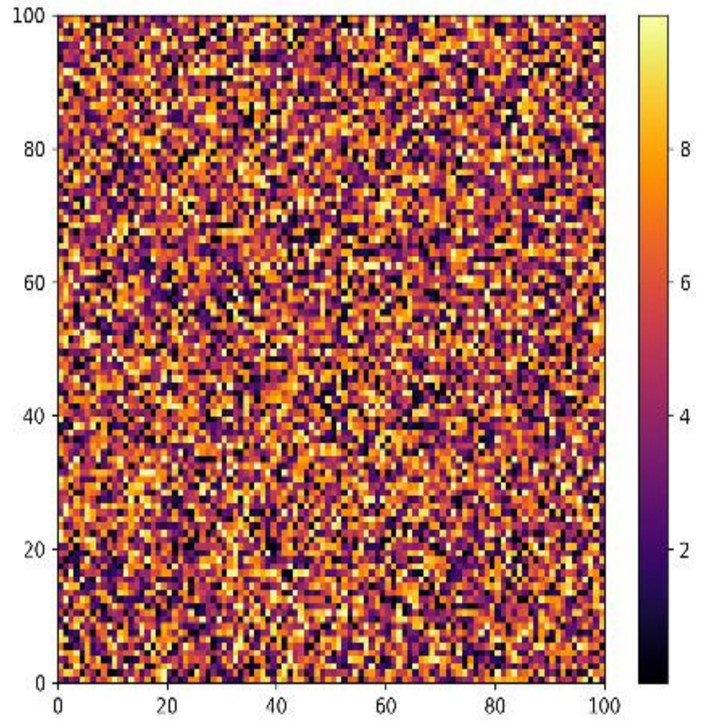
plt.pcolormesh(data) plt.colorbar()



How to Create New Colormaps (Cmap) in Python

- As mentioned, if you didn’t define the colormaps you used, you will get the default matplotlib colormaps. The default colormap name is ‘**viridis**’.
- change the colormaps from ‘**viridis**’ to ‘inferno’ colormaps with this code.

plt.pcolormesh(data, cmap='inferno')



How to Modify Colormaps (Cmap) in Python?

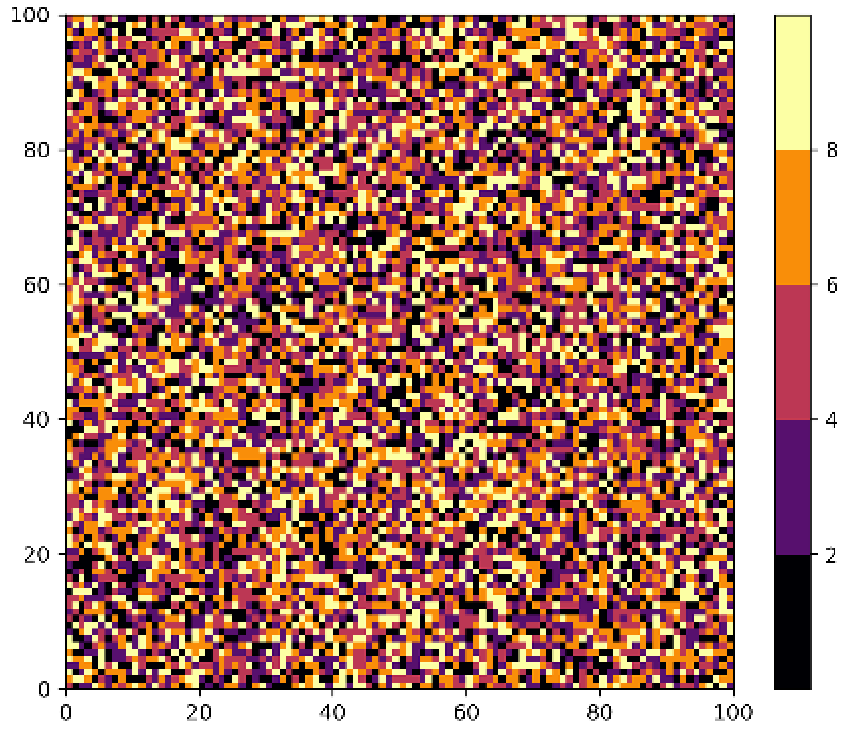
- To modify the colormaps, you need to import the following sublibraries in Matplotlib.

```
from matplotlib import cm
from matplotlib.colors import
ListedColormap,LinearSegmentedColormap
```

How to Modify Colormaps (Cmap) in Python?

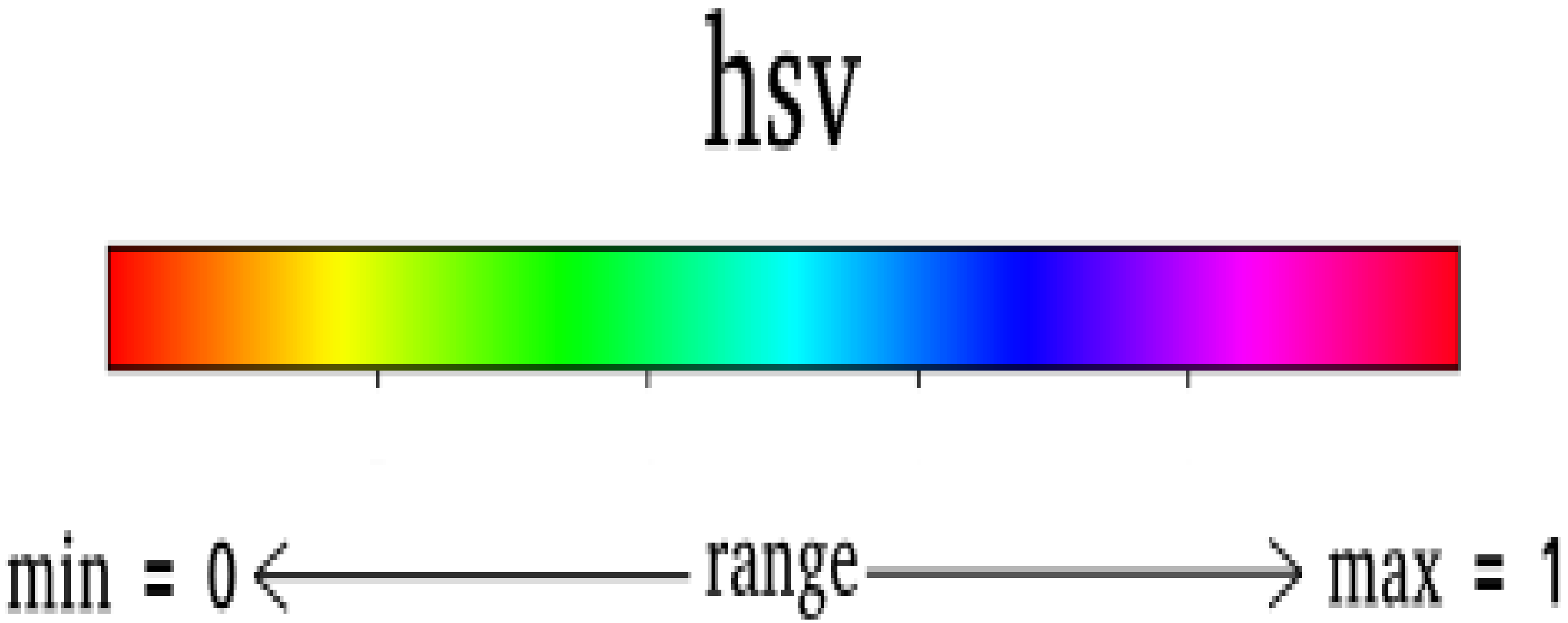
- To modify the number of color classes in your colormaps, we can use this code

- **new_inferno = cm.get_cmap('inferno', 5)# visualize with the new_inferno colormaps plt.pcolormesh(data, cmap = new_inferno) plt.colorbar()**



Modify Range of Color in Colormaps

- Modifying the range of color in a colormap. 'hsv' colormaps- need to understand the range of colors using the figure:



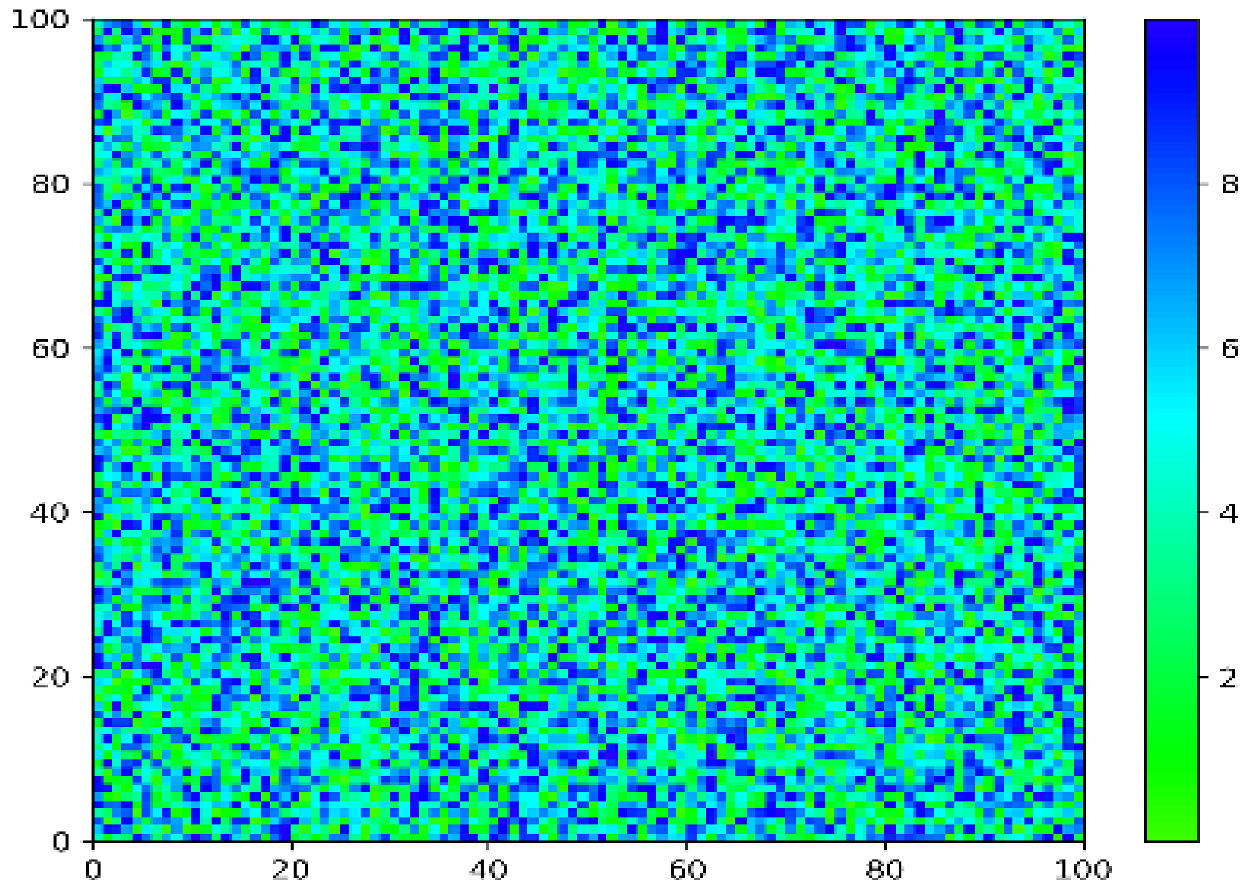
Modify Range of Color in Colormaps

- If we want to use only green color (about 0.3) to blue color (0.7), we can use the following code.

```
# modified hsv in 256 color class
 hsv_modified = cm.get_cmap('hsv', 256)# create new hsv colormaps
in range of 0.3 (green) to 0.7 (blue)
newcmp = ListedColormap(hsv_modified(np.linspace(0.3, 0.7, 256)))
# show figure
plt.figure(figsize=(7, 6))
plt.pcolormesh(data, cmap = newcmp)
 plt.colorbar()
```

Modify Range of Color in Colormaps

- It will give a figure like this:



List of Colors in Matplotlib

- List of colors in matplotlib

```
import matplotlib.pyplot as plt
colors = plt.cm.colors.CSS4_COLORS
print(colors)
```

List of Colors in Matplotlib

- List of colors in matplotlib

- {'aliceblue': '#F0F8FF', 'antiquewhite': '#FAEBD7', 'aqua': '#00FFFF', 'aquamarine': '#7FFFD4', 'azure': '#F0FFFF', 'beige': '#F5F5DC', 'bisque': '#FFE4C4', 'black': '#000000', 'blanchedalmond': '#FFEBCD', 'blue': '#0000FF', 'blueviolet': '#8A2BE2', 'brown': '#A52A2A', 'burlywood': '#DEB887', 'cadetblue': '#5F9EA0', 'chartreuse': '#7FFF00', 'chocolate': '#D2691E', 'coral': '#FF7F50', 'cornflowerblue': '#6495ED', 'cornsilk': '#FFF8DC', 'crimson': '#DC143C', 'cyan': '#00FFFF', 'darkblue': '#00008B', 'darkcyan': '#008B8B', 'darkgoldenrod': '#B8860B', 'darkgray': '#A9A9A9', 'darkgreen': '#006400', 'darkgrey': '#A9A9A9', 'darkkhaki': '#BDB76B', 'darkmagenta': '#8B008B', 'darkolivegreen': '#556B2F', 'darkorange': '#FF8C00', 'darkorchid': '#9932CC', 'darkred': '#8B0000', 'darksalmon': '#E9967A', 'darkseagreen': '#8FBC8F', 'darkslateblue': '#483D8B', 'darkslategray': '#2F4F4F', 'darkslategrey': '#2F4F4F', 'darkturquoise': '#00CED1', 'darkviolet': '#9400D3', 'deeppink': '#FF1493', 'deepskyblue': '#00BFFF', 'dimgray': '#696969', 'dimgrey': '#696969', 'dodgerblue': '#1E90FF', 'firebrick': '#B22222', 'floralwhite': '#FFFAF0', 'forestgreen': '#228B22', 'fuchsia': '#FF00FF', 'gainsboro': '#DCDCDC', 'ghostwhite': '#F8F8FF', 'gold': '#FFD700', 'goldenrod': '#DAA520', 'gray': '#808080', 'green': '#008000', 'greenyellow': '#ADFF2F', 'grey': '#808080', 'honeydew': '#F0FFF0', 'hotpink': '#FF69B4', 'indianred': '#CD5C5C', 'indigo': '#4B0082', 'ivory': '#FFFFF0', 'khaki': '#F0E68C', 'lavender': '#E6E6FA', 'lavenderblush': '#FFF0F5', 'lawngreen': '#7CFC00', 'lemonchiffon': '#FFFACD', 'lightblue': '#ADD8E6', 'lightcoral': '#F08080', 'lightcyan': '#E0FFFF', 'lightgoldenrodyellow': '#FAFAD2', 'lightgray': '#D3D3D3', 'lightgreen': '#90EE90', 'lightgrey': '#D3D3D3', 'lightpink': '#FFB6C1', 'lightsalmon': '#FFA07A', 'lightseagreen': '#20B2AA', 'lightskyblue': '#87CEFA', 'lightslategray': '#778899', 'lightslategrey': '#778899', 'lightsteelblue': '#B0C4DE', 'lightyellow': '#FFFFE0', 'lime': '#00FF00', 'limegreen': '#32CD32', 'linen': '#FAF0E6', 'magenta': '#FF00FF', 'maroon': '#800000', 'mediumaquamarine': '#66CDAA', 'mediumblue': '#0000CD', 'mediumorchid': '#BA55D3', 'mediumpurple': '#9370DB', 'mediumseagreen': '#3CB371', 'mediumslateblue': '#7B68EE', 'mediumspringgreen': '#00FA9A', 'mediumturquoise': '#48D1CC', 'mediumvioletred': '#C71585', 'midnightblue': '#191970', 'mintcream': '#F5FFFA', 'mistyrose': '#FFE4E1', 'moccasin': '#FFE4B5', 'navajowhite': '#FFDEAD', 'navy': '#000080', 'oldlace': '#FDF5E6', 'olive': '#808000', 'olivedrab': '#6B8E23', 'orange': '#FFA500', 'orangered': '#FF4500', 'orchid': '#DA70D6', 'palegoldenrod': '#EEE8AA', 'palegreen': '#98FB98', 'paleturquoise': '#AFEEEE', 'palevioletred': '#DB7093', 'papayawhip': '#FFEFD5', 'peachpuff': '#FFDAB9', 'peru': '#CD853F', 'pink': '#FFC0CB', 'plum': '#DDA0DD', 'powderblue': '#B0E0E6', 'purple': '#800080', 'rebeccapurple': '#663399', 'red': '#FF0000', 'rosybrown': '#BC8F8F', 'royalblue': '#4169E1', 'saddlebrown': '#8B4513', 'salmon': '#FA8072', 'sandybrown': '#F4A460', 'seagreen': '#2E8B57', 'seashell': '#FFF5EE', 'sienna': '#A0522D', 'silver': '#C0C0C0', 'skyblue': '#87CEEB', 'slateblue': '#6A5ACD', 'slategray': '#708090', 'slategrey': '#708090', 'snow': '#FFFAFA', 'springgreen': '#00FF7F', 'steelblue': '#4682B4', 'tan': '#D2B48C', 'teal': '#008080', 'thistle': '#D8BFD8', 'tomato': '#FF6347', 'turquoise': '#40E0D0', 'violet': '#EE82EE', 'wheat': '#F5DEB3', 'white': '#FFFFFF', 'whitesmoke': '#F5F5F5', 'yellow': '#FFFF00', 'yellowgreen': '#9ACD32'}

-

List of Colors in Matplotlib

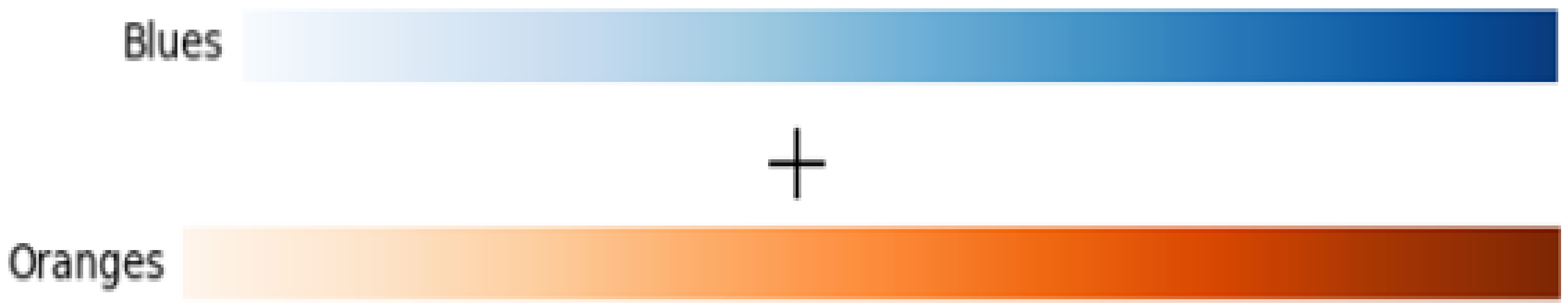
- This code will give you a list of all the colors available in CSS4 – a set of standardized color names used in web design.
- You can also access other color maps in matplotlib, such as plt.cm.Reds, or plt.cm.Blues by specifying the desired color map when calling plt.cm.colors.

Create Custom Colormaps (Cmap) in Python

- Custom colormaps can be created and used in Matplotlib to visualize data in a more informative and appealing way. Matplotlib provides various functions to create and modify colormaps, such as LinearSegmentedColormap and ListedColormap, making it a flexible library for creating colormaps in a more customized way.
- To create your own colormaps, there are at least two methods. First, you can combine two Sequential colormaps in Matplotlib. Second, you can choose and combine your favorite color in RGB to create colormaps.

Create Custom Colormaps (Cmap) in Python

- Combining two Sequential colormaps to create a new colormap. We want to combine ‘Oranges’ and ‘Blues.’

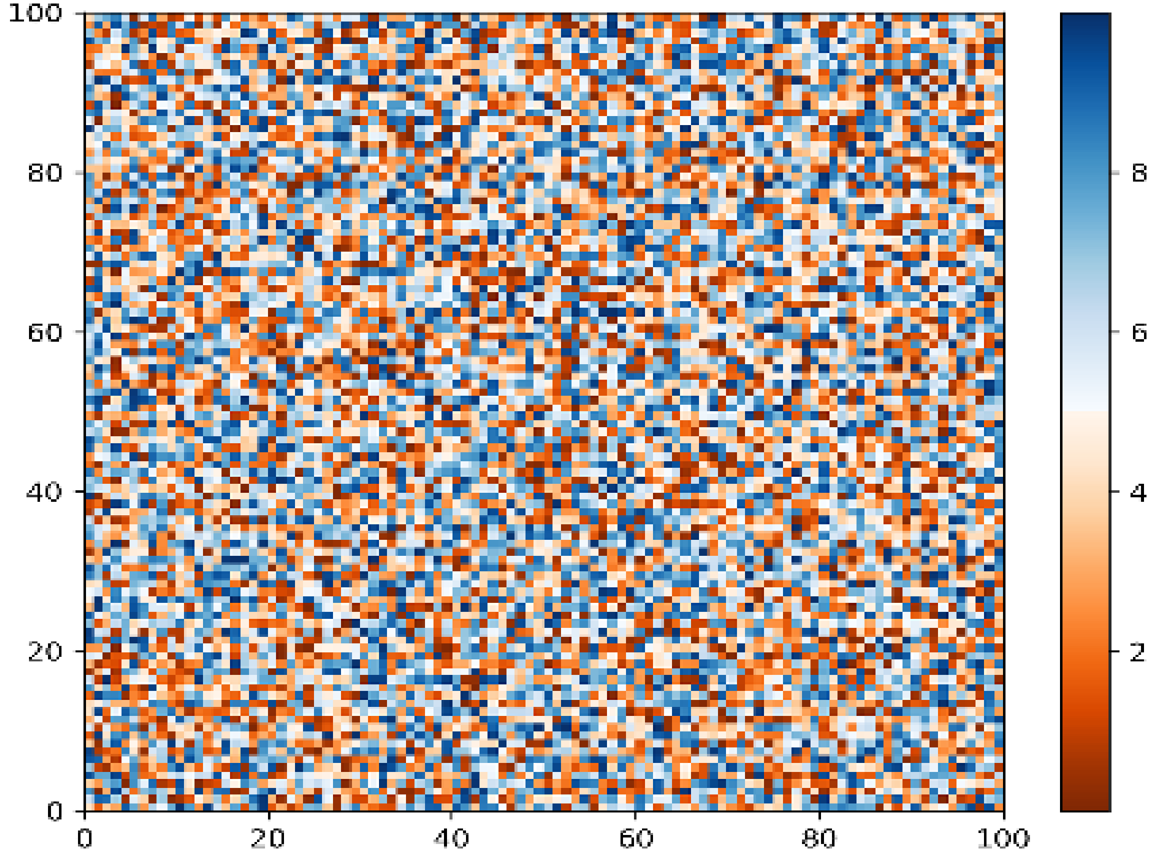


Create Custom Colormaps (Cmap) in Python

```
# define top and bottom colormaps
top = cm.get_cmap('Oranges_r', 128)
 # r means reversed version
bottom = cm.get_cmap('Blues', 128)
# combine it all
 newcolors = np.vstack((top(np.linspace(0, 1, 128)),
bottom(np.linspace(0, 1, 128))))
# create a new colormaps with a name of OrangeBlue
 orange_blue = ListedColormap(newcolors, name='OrangeBlue')
```

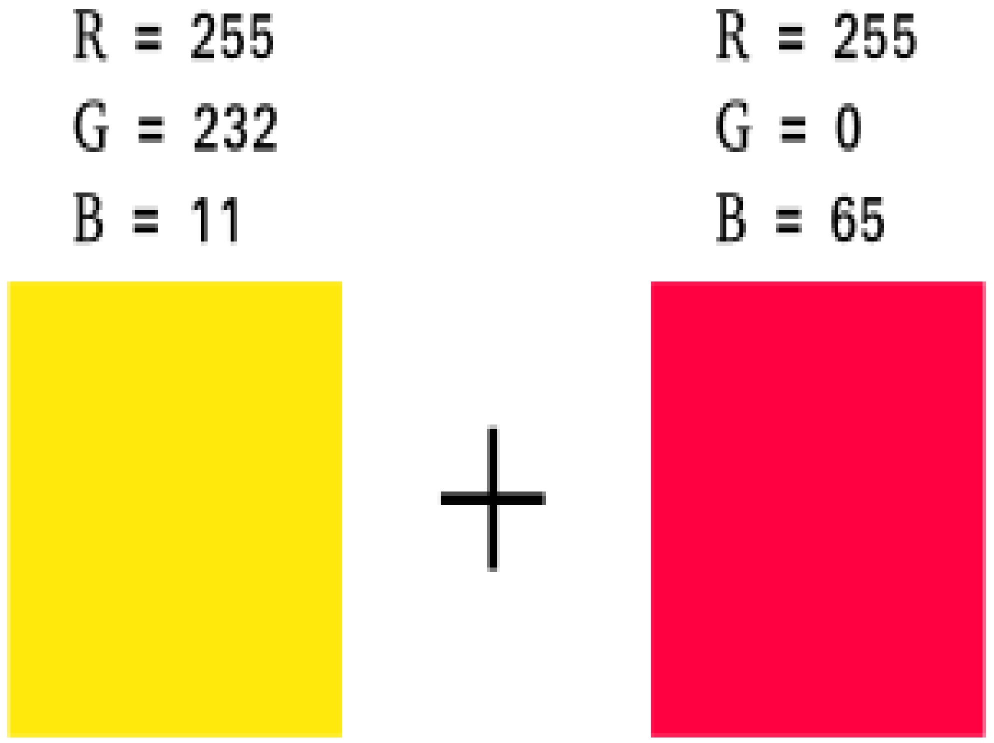
Create Custom Colormaps (Cmap) in Python

If we visualize the mock data using ‘OrangeBlue’ colormaps, we will get a figure like this.



Creating a colormap from two different colors we like

In this case, we will try to create it from yellow and red, as shown in the following picture.



Creating a colormap from two different colors we like

First, you need to create yellow colormaps

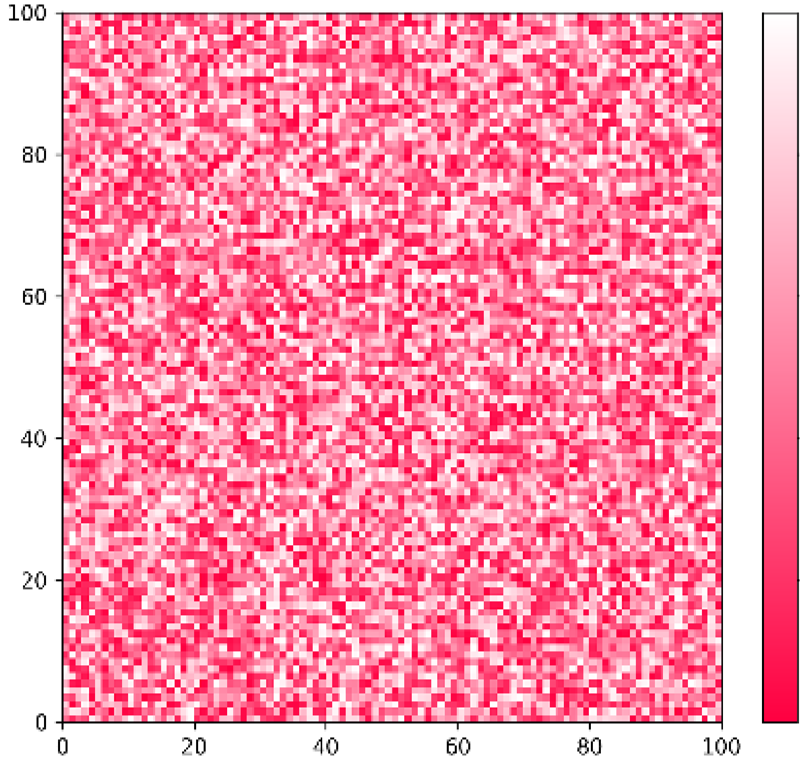
```
# create yellow colormaps
N = 256
yellow = np.ones((N, 4))yellow[:, 0] = np.linspace(255/256, 1, N)
# R
= 255 yellow[:, 1] = np.linspace(232/256, 1, N)
# G
= 232 yellow[:, 2] = np.linspace(11/256, 1, N)
# B
= 11yellow_cmp = ListedColormap(yellow)
```

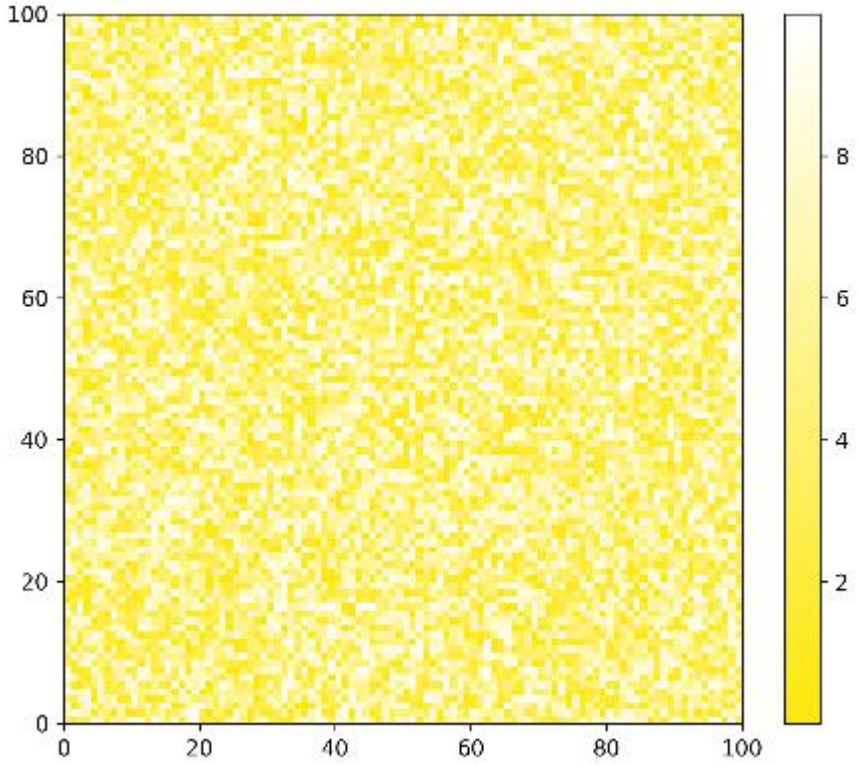
Creating a colormap from two different colors we like

and red colormaps

```
red = np.ones((N, 4))red[:, 0] = np.linspace(255/256, 1, N)
 red[:, 1] = np.linspace(0/256, 1, N)
red[:, 2] = np.linspace(65/256, 1, N)
red_cmp = ListedColormap(red)
```

The visualization of the yellow and red colormaps created is shown in the following picture.





Colormaps - Conclusion

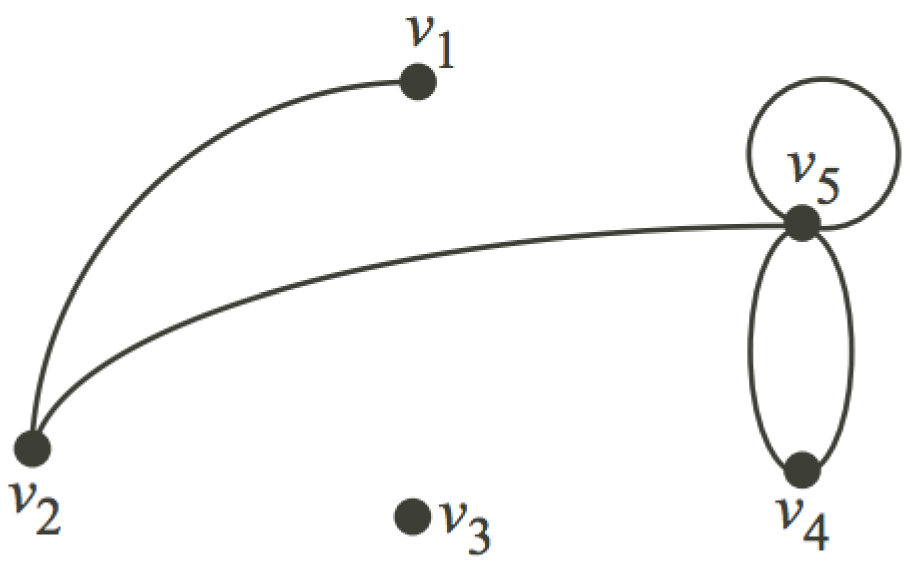
- Colormaps or Cmap in Python is a very useful tool for data visualization.
- Matlibpro comes with a number of built-in colormaps, such as sequential, diverging, cyclic, qualitative, miscellaneous, etc.
- We can modify these default Cmaps or create our own custom ones using Python.

References

- https://www.analyticsvidhya.com/blog/2020/09/colormaps-matplotlib/

Graphs

- Let us look at a simple graph to understand the concept. Look at the image below –



Graphs

- Consider that this graph represents the places in a city that people generally visit, and the path that was followed by a visitor of that city. Let us consider V as the places and E as the path to travel from one place to another.

V = {v1, v2, v3, v4, v5}

E = {(v1,v2), (v2,v5), (v5, v5), (v4,v5), (v4,v4)}

The edge (u,v) is the same as the edge (v,u) – They are unordered pairs.

Graphs

- The Data Science and Analytics field has also used Graphs to model various structures and problems. As a Data Scientist, we should be able to solve problems in an efficient manner and Graphs provide a mechanism to do that in cases where the data is arranged in a specific way.

Graphs

- Formally,
- A **Graph** is a pair of sets. G = (V,E). V is the set of vertices. E is a set of edges. E is made up of pairs of elements from V (unordered pair)

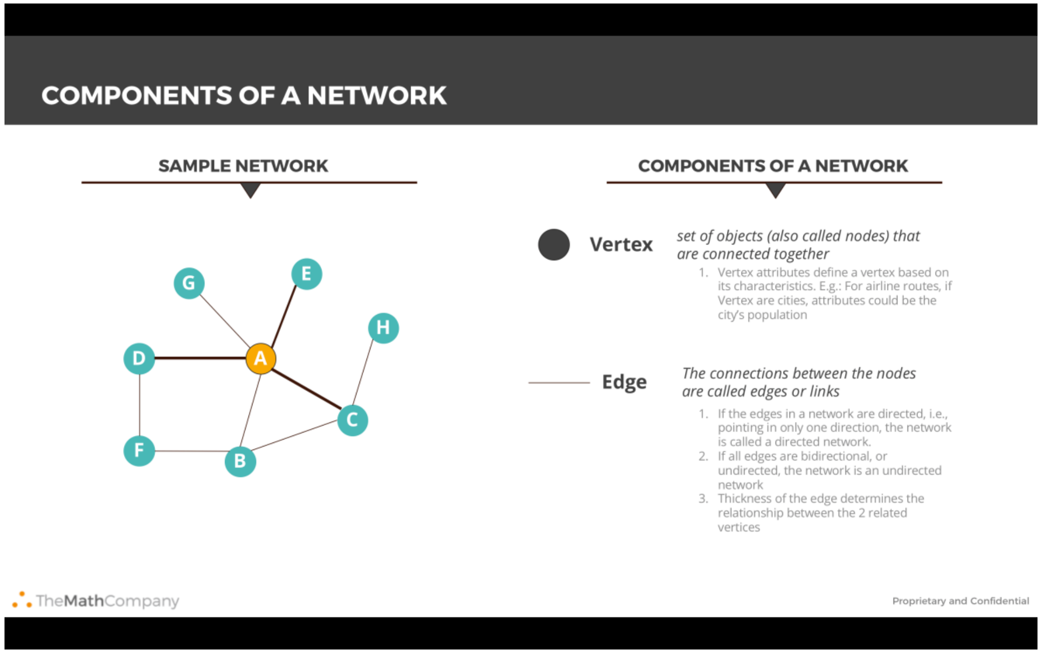
- A **DiGraph** is also a pair of sets. D = (V,A). V is the set of vertices. A is the set of arcs. A is made up of pairs of elements from V (ordered pair)

- In the case of digraphs, there is a distinction between \`(u,v)\` and \`(v,u)\`. Usually the edges are called arcs in such cases to indicate a notion of direction.

Graphs

- There are packages that exist in R and Python to analyze data using Graph theory concepts. In this article we will be briefly looking at some of the concepts and analyze a dataset using Networkx Python package.

from IPython.display import Image
Image('images/network.PNG')



COMPONENTS OF A NETWORK

SAMPLE NETWORK

COMPONENTS OF A NETWORK

● **Vertex** *set of objects (also called nodes) that are connected together*

1. Vertex attributes define a vertex based on its characteristics. E.g.: For airline routes, if Vertex are cities, attributes could be the city's population

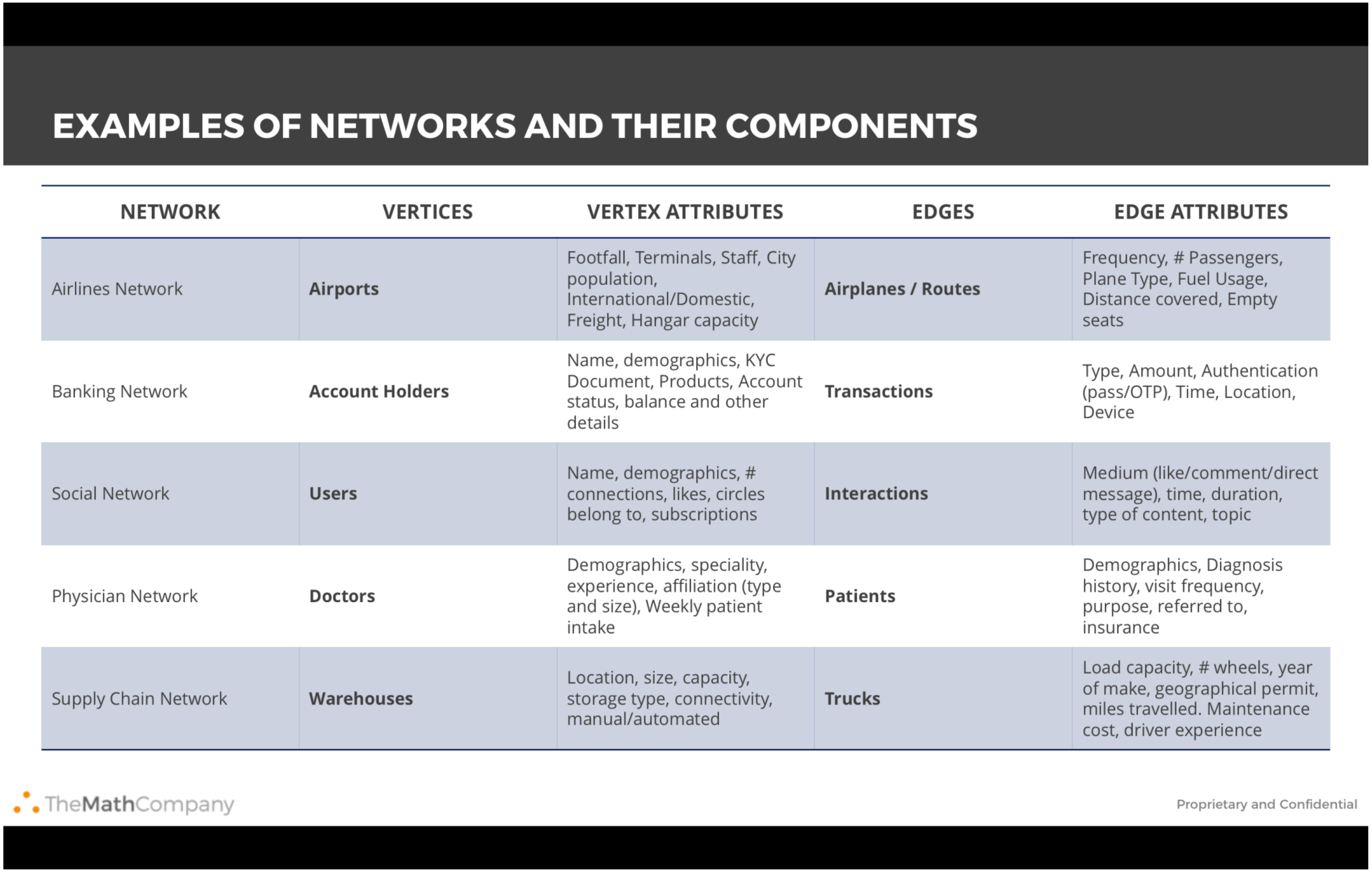
Edge *The connections between the nodes are called edges or links*

1. If the edges in a network are directed, i.e., pointing in only one direction, the network is called a directed network.
2. If all edges are bidirectional, or undirected, the network is an undirected network
3. Thickness of the edge determines the relationship between the 2 related vertices

TheMathCompany

Proprietary and Confidential

Examples of Networks and Their Components



EXAMPLES OF NETWORKS AND THEIR COMPONENTS

NETWORK	VERTICES	VERTEX ATTRIBUTES	EDGES	EDGE ATTRIBUTES
Airlines Network	**Airports**	Footfall, Terminals, Staff, City population, International/Domestic, Freight, Hangar capacity	**Airplanes / Routes**	Frequency, # Passengers, Plane Type, Fuel Usage, Distance covered, Empty seats
Banking Network	**Account Holders**	Name, demographics, KYC Document, Products, Account status, balance and other details	**Transactions**	Type, Amount, Authentication (pass/OTP), Time, Location, Device
Social Network	**Users**	Name, demographics, # connections, likes, circles belong to, subscriptions	**Interactions**	Medium (like/comment/direct message), time, duration, type of content, topic
Physician Network	**Doctors**	Demographics, speciality, experience, affiliation (type and size), Weekly patient intake	**Patients**	Demographics, Diagnosis history, visit frequency, purpose, referred to, insurance
Supply Chain Network	**Warehouses**	Location, size, capacity, storage type, connectivity, manual/automated	**Trucks**	Load capacity, # wheels, year of make, geographical permit, miles travelled. Maintenance cost, driver experience

TheMathCompany

Proprietary and Confidential

Applications of Graphs in Data Analytics

Marketing Analytics – Graphs can be used to figure out the most influential people in a Social Network. Advertisers and Marketers can estimate the biggest bang for the marketing buck by routing their message through the most influential people in a Social Network

Banking Transactions – Graphs can be used to find unusual patterns helping in mitigating Fraudulent transactions. There have been examples where Terrorist activity has been detected by analyzing the flow of money across interconnected Banking networks

Applications of Graphs in Data Analytics

Supply Chain – Graphs help in identifying optimum routes for your delivery trucks and in identifying locations for warehouses and delivery centres

Pharma – Pharma companies can optimize the routes of the salesman using Graph theory. This helps in cutting costs and reducing the travel time for salesman

Telecom – Telecom companies typically use Graphs (Voronoi diagrams) to understand the quantity and location of Cell towers to ensure maximum coverage

Why Graphs

- Motivate to use graphs in our day-to-day data science problems

1. Graphs provide a better way of dealing with abstract concepts like relationships and interactions. They also offer an intuitively visual way of thinking about these concepts. Graphs also form a natural basis for analyzing relationships in a Social context
2. Graph Databases have become common computational tools and alternatives to SQL and NoSQL databases
3. Graphs are used to model analytics workflows in the form of DAGs (Directed acyclic graphs)

Why Graphs

4. Some Neural Network Frameworks also use DAGs to model the various operations in different layers.
5. Graph Theory concepts are used to study and model Social Networks, Fraud patterns, Power consumption patterns, Virality and Influence in Social Media. Social Network Analysis (SNA) is probably the best known application of Graph Theory for Data Science
6. It is used in Clustering algorithms – Specifically K-Means

Why Graphs

7. System Dynamics also uses some Graph Theory concepts – Specifically loops
8. Path Optimization is a subset of the Optimization problem that also uses Graph concepts
9. From a Computer Science perspective – Graphs offer computational efficiency. The Big O complexity for some algorithms is better for data arranged in the form of Graphs (compared to tabular data)

Getting Familiar with Graphs in python

- We will be using the networkx package in Python.
- It can be installed in the Root environment of Anaconda (if you are using the Anaconda distribution of Python).
- We can also use **pip install it**.
- Let us look at some common things that can be done with the Networkx package. These include importing and creating a Graph and ways to visualize it.

Graph Creation

```
import networkx as nx

# Creating a Graph
G = nx.Graph() # Right now G is empty

# Add a node
G.add_node(1)
G.add_nodes_from([2,3]) # You can also add a list of nodes by passing a
list argument

# Add edges
G.add_edge(1,2)

e = (2,3)
G.add_edge(*e) # * unpacks the tuple
G.add_edges_from([(1,2), (1,3)]) # Just like nodes we can add edges from
 a list
```

Graph Creation

```
print('\n\nNodes of the Graph\n')
print(G.nodes())
```

Output :

```
Nodes of the Graph [1, 2, 3]
```

```
print('\n\n Graph Edges \n')
print(G.edges())
```

Output :

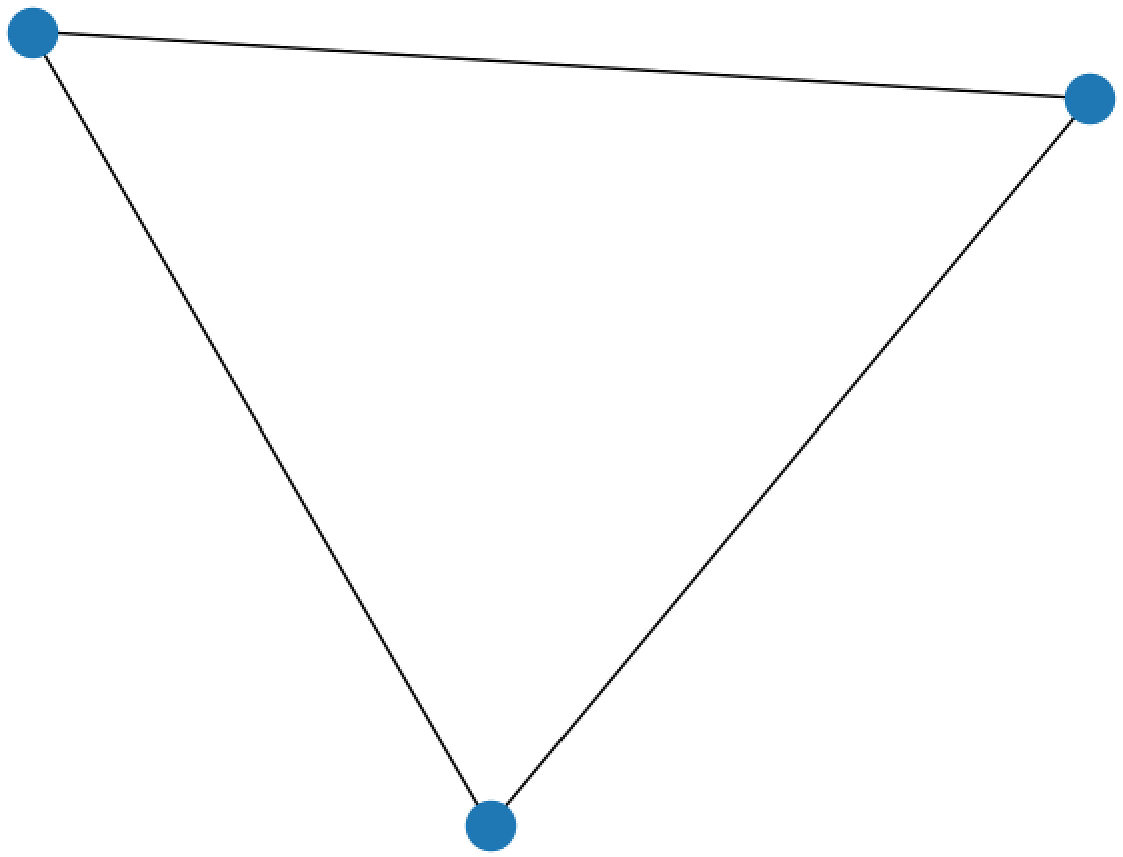
```
Graph Edges [(1, 2), (1, 3), (2, 3)]
```

Graph Visualization

- Networkx provides basic functionality for visualizing graphs, but its main goal is to enable graph analysis rather than perform graph visualization.
- Graph visualization is hard and we will have to use specific tools dedicated for this task.
- Matplotlib offers some convenience functions.
- But GraphViz is probably the best tool for us as it offers a Python interface in the form of PyGraphViz

Graph Visualization

```
import matplotlib.pyplot as plt
nx.draw(G)
```



Graph Visualization

- We first have to Install Graphviz from the website (link below).
- And then pip install pygraphviz --install-option=" <>.
- In the install options you will have to provide the path to the Graphviz lib and include folders.

Graph Visualization

```
import pygraphviz as pgv
d={'1': {'2': None}, '2': {'1': None, '3': None}, '3': {'1': None}}
A = pgv.AGraph(data=d)
print(A)
# This is the 'string' or simple representation of the Graph
```

Output:

```
strict graph "" {
        1 -- 2;
        2 -- 3;
        3 -- 1;
}
```

Graph Visualization

- PyGraphviz provides great control over the individual attributes of the edges and nodes.
- We can get very beautiful visualizations using it.

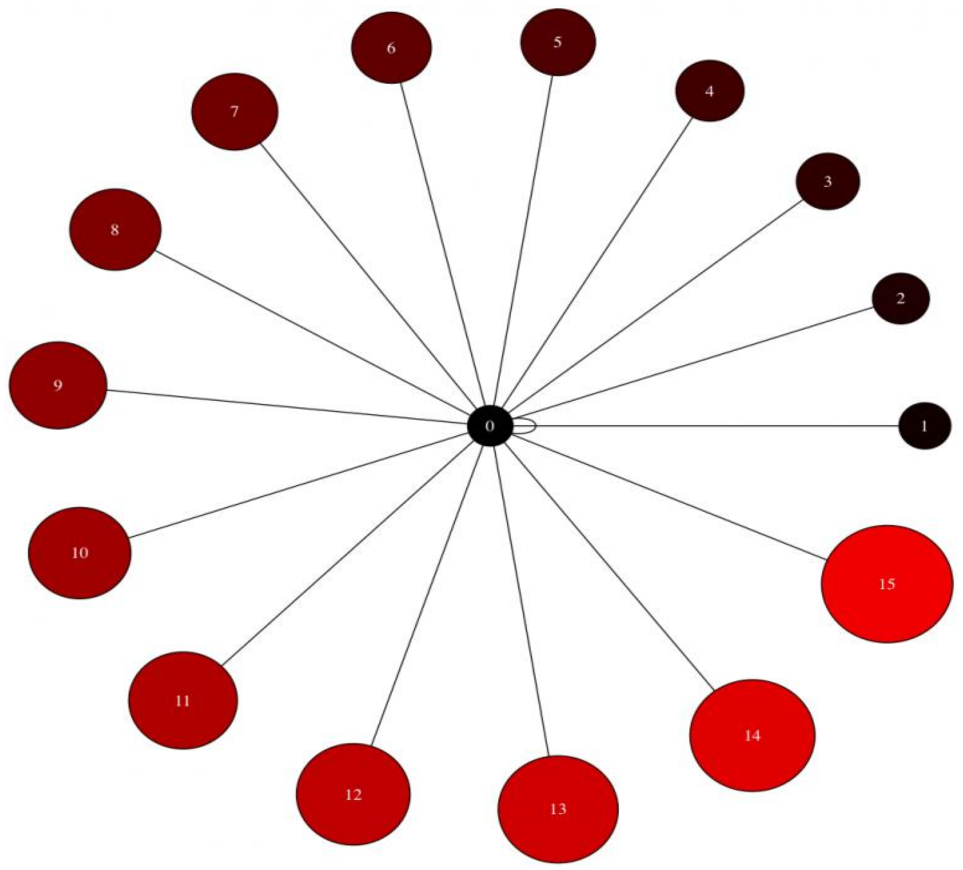
Graph Visualization

```
# Let us create another Graph where we can
individually control the colour of each node
B = pgv.AGraph()

# Setting node attributes that are common for all
nodes
B.node_attr['style']='filled'
B.node_attr['shape']='circle'
B.node_attr['fixedsize']='true'
B.node_attr['fontcolor']='#FFFFFF'
```

Graph Visualization

```
# Creating and setting node attributes that vary for
each node (using a for loop)
for i in range(16):
 B.add_edge(0,i)
 n=B.get_node(i)
 n.attr['fillcolor']="#%2x0000"%(i*16)
 n.attr['height']="%s"%(i/16.0+0.5)
 n.attr['width']="%s"%(i/16.0+0.5)
B.draw('star.png',prog="circo") # This creates a
.png file in the local directory. Displayed below.
Image('images/star.png', width=650) # The Graph
visualization we created above.
```

Graph Visualization



Usually, visualization is thought of as a separate task from Graph analysis. A graph once analyzed is exported as a Dotfile. This Dotfile is then visualized separately to illustrate a specific point we are trying to make.

Analysis on a Dataset

- The dataset we will be looking at comes from the Airlines Industry.
- It has some basic information on the Airline routes.
- There is a Source of a journey and a destination. There are also a few columns indicating arrival and departure times for each journey.
- As you can imagine this dataset lends itself beautifully to be analysed as a Graph.
- Imagine a few cities (nodes) connected by airline routes (edges).
- If you are an airline carrier, you can then proceed to ask a few questions like:

1. What is the shortest way to get from A to B? In terms of distance and in terms of time
2. Is there a way to go from C to D?
3. Which airports have the heaviest traffic?
4. Which airport in “in between” most other airports? So that it can be converted into a local hub

Analysis on a Dataset

```
import pandas as pd
import numpy as np

data = pd.read_csv('data/Airlines.csv')

data.shape
(100, 16)
```

Analysis on a Dataset

data.dtypes

```
year              int64
month               int64
day               int64
dep_time          float64
sched_dep_time    int64
dep_delay         float64
arr_time          float64
sched_arr_time    int64
arr_delay         float64
carrier           object
flight            int64
tailnum             object
origin            object
dest              object
air_time          float64
distance            int64
dtype: object
```

Analysis on a Dataset

1. We notice that origin and destination look like good choices for Nodes. Everything can then be imagined as either node or edge attributes. A single edge can be thought of as a journey. And such a journey will have various times, a flight number, an airplane tail number etc associated with it
2. We notice that the year, month, day and time information is spread over many columns. We want to create one datetime column containing all of this information. We also need to keep scheduled and actual time of arrival and departure separate. So we should finally have 4 datetime columns (Scheduled and actual times of arrival and departure)

Analysis on a Dataset

3. Additionally, the time columns are not in a proper format. 4:30 pm is represented as 1630 instead of 16:30. There is no delimiter to split that column. One approach is to use pandas string methods and regular expressions
4. We should also note that sched_dep_time and sched_arr_time are int64 dtype and dep_time and arr_time are float64 dtype
5. An additional complication is NaN values

Analysis on a Dataset

```
# converting sched_dep_time to 'std‘
```

Scheduled time of departure

```
data['std'] =
data.sched_dep_time.astype(str).str.replace('(\d{2}$)',
'') + ':' +
data.sched_dep_time.astype(str).str.extract('(\d{2}$)',
expand=False) + ':00'
```

Analysis on a Dataset

# converting sched_arr_time to 'sta' - Scheduled time of arrival

```
data['sta'] =
data.sched_arr_time.astype(str).str.replace('(\d{2}$)',
'') + ':' +
data.sched_arr_time.astype(str).str.extract('(\d{2}$)',
expand=False) + ':00'
```

Analysis on a Dataset

# converting dep_time to 'atd' - Actual time of departure

```
data['atd'] = data.dep_time.fillna(0).astype(np.int64).astype(str).str.replace('(\d{2}$)', '') + ':' + data.dep_time.fillna(0).astype(np.int64).astype(str).str.extract('(\d{2}$)', expand=False) + ':00'
```

Analysis on a Dataset

# converting arr_time to 'ata' - Actual time of arrival

```
data['ata'] =
data.arr_time.fillna(0).astype(np.int64).astype(str).str.r
eplace('(\d{2}$)', '') + ':' +
data.arr_time.fillna(0).astype(np.int64).astype(str).str.e
xtract('(\d{2}$)', expand=False) + ':00'
```

Analysis on a Dataset

We now have time columns in the format we wanted. Finally we may want to combine the year, month and day columns into a date column. This is not an absolutely necessary step. But we can easily obtain the year, month and day (and other) information once it is converted into datetime format.

```
data['date'] = pd.to_datetime(data[['year', 'month',
'day']])

# finally we drop the columns we don't need
data = data.drop(columns = ['year', 'month', 'day'])
```

Analysis on a Dataset

Now import the dataset using the networkx function that ingests a pandas dataframe directly. Just like Graph creation there are multiple ways Data can be ingested into a Graph from multiple formats.

```
import networkx as nx
FG = nx.from_pandas_edgelist(data, source='origin', target='dest',
edge_attr=True,)
FG.nodes()
```

Output:

```
NodeView(('EWR', 'MEM', 'LGA', 'FLL', 'SEA', 'JFK', 'DEN', 'ORD', 'MIA', 'PBI', 'MCO', 'CMH',
'MSP', 'IAD', 'CLT', 'TPA', 'DCA', 'SJU', 'ATL', 'BHM','SRQ', 'MSY', 'DTW', 'LAX', 'JAX', 'RDU',
'MDW', 'DFW', 'IAH', 'SFO', 'STL', 'CVG', 'IND', 'RSW', 'BOS', 'CLE'))
```

Analysis on a Dataset

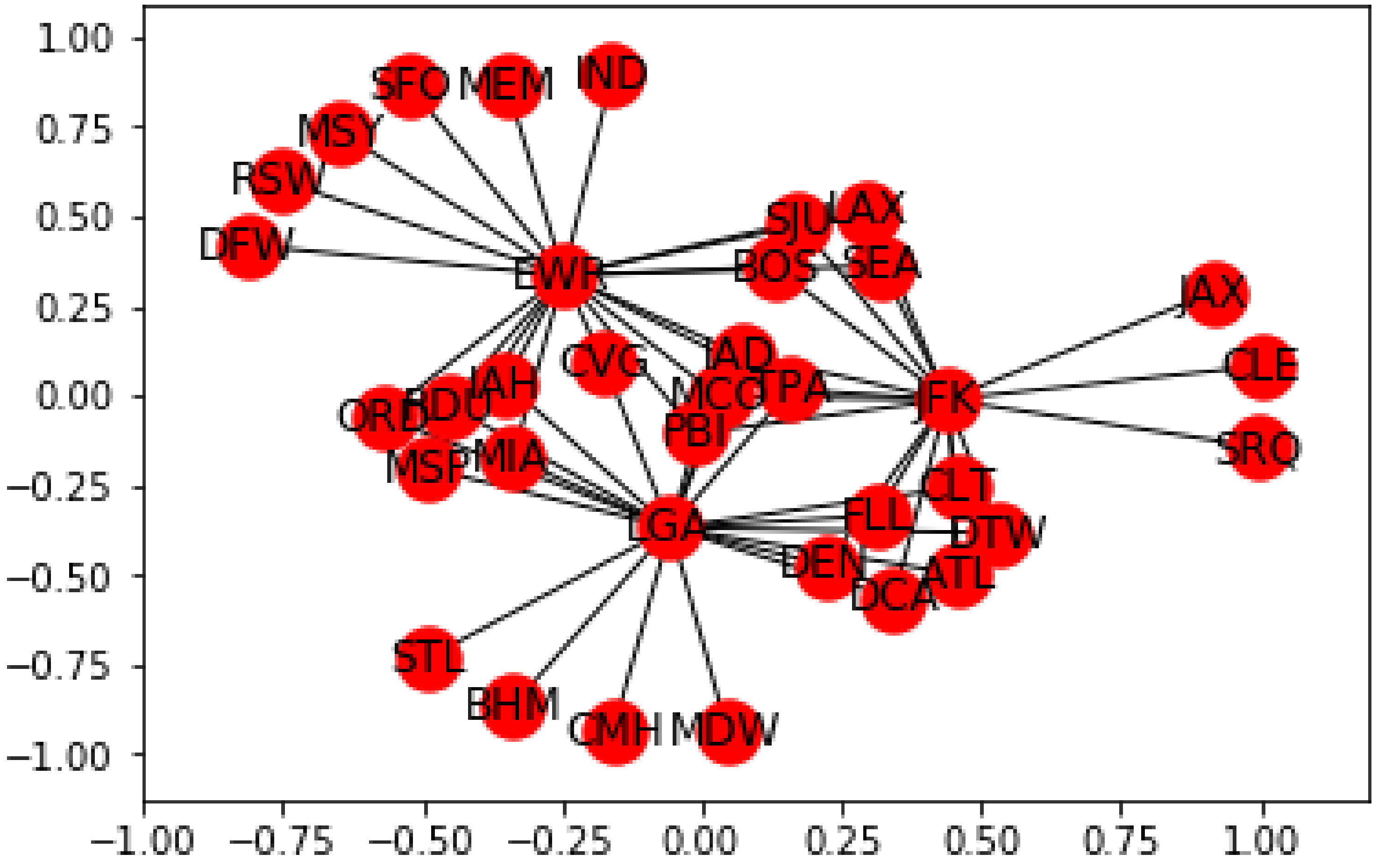
```
FG.edges()
```

Output:

```
EdgeView([('EWR', 'MEM'), ('EWR', 'SEA'), ('EWR', 'MIA'),
('EWR', 'ORD'), ('EWR', 'MSP'), ('EWR', 'TPA'), ('EWR', 'MSY'),
 ('EWR', 'DFW'), ('EWR', 'IAH'), ('EWR', 'SFO'), ('EWR', 'CVG'),
 ('EWR', 'IND'), ('EWR', 'RDU'), ('EWR', 'IAD'), ('EWR', 'RSW'),
('EWR', 'BOS'), ('EWR', 'PBI'), ('EWR', 'LAX'), ('EWR', 'MCO'),
 ('EWR', 'SJU'), ('LGA', 'FLL'), ('LGA', 'ORD'), ('LGA', 'PBI'),
 ('LGA', 'CMH'), ('LGA', 'IAD'), ('LGA', 'CLT'), ('LGA', 'MIA'),
('LGA', 'DCA'), ('LGA', 'BHM'), ('LGA', 'RDU'), ('LGA', 'ATL'),
('LGA', 'TPA'), ('LGA', 'MDW'), ('LGA', 'DEN'), ('LGA', 'MSP'),
 ('LGA', 'DTW'), ('LGA', 'STL'), ('LGA', 'MCO'), ('LGA', 'CVG'),
('LGA', 'IAH'), ('FLL', 'JFK'), ('SEA', 'JFK'), ('JFK', 'DEN'),
('JFK', 'MCO'), ('JFK', 'TPA'), ('JFK', 'SJU'), ('JFK', 'ATL'),
('JFK', 'SRQ'), ('JFK', 'DCA'), ('JFK', 'DTW'), ('JFK', 'LAX'),
 ('JFK', 'JAX'), ('JFK', 'CLT'), ('JFK', 'PBI'), ('JFK', 'CLE'),
 ('JFK', 'IAD'), ('JFK', 'BOS')]
```

Analysis on a Dataset

nx.draw_networkx(FG, with_labels=True)
# Quick view of the Graph. As expected we see 3 very busy airports



Analysis on a Dataset

```
nx.algorithms.degree_centrality(FG) # Notice the 3 airports
from which all of our 100 rows of data originates

nx.density(FG) # Average edge density of the Graphs
```

Output:

```
0.09047619047619047
```

Analysis on a Dataset

```
nx.average_shortest_path_length(FG)
# Average shortest path length for ALL paths
in the Graph
```

Output:

```
2.36984126984127
```

Analysis on a Dataset

```
nx.average_degree_connectivity(FG) # For a node of degree k - What is the average of its neighbours' degree?
```

Output:

```
{1: 19.307692307692307, 2: 19.0625, 3: 19.0,
17: 2.0588235294117645, 20: 1.95}
```

Analysis on a Dataset

As is obvious from looking at the Graph visualization (way above) – There are multiple paths from some airports to others. Let us say we want to calculate the shortest possible route between 2 such airports. Right off the bat we can think of a couple of ways of doing it

There is the shortest path by distance

There is the shortest path by flight time

Analysis on a Dataset

What we can do is to calculate the shortest path algorithm by weighing the paths with either the distance or airtime. Please note that this is an approximate solution – The actual problem to solve is to calculate the shortest path factoring in the availability of a flight when you reach your transfer airport + wait time for the transfer. This is a more complete approach and this is how humans normally plan their travel. For the purposes of this article we will just assume that is flight is readily available when you reach an airport and calculate the shortest path using the airtime as the weight

Analysis on a Dataset

Let us take the example of JAX and DFW airports:

```
# Let us find all the paths available
for path in nx.all_simple_paths(FG, source='JAX', target='DFW'):
 print(path)

# Let us find the dijkstra path from JAX to DFW.

# You can read more in-depth on how dijkstra works from this resource –

https://courses.csail.mit.edu/6.006/fall11/lectures/lecture16.pdf
dijpath = nx.dijkstra_path(FG, source='JAX', target='DFW')
dijpath
```

Analysis on a Dataset

Output:

```
['JAX', 'JFK', 'SEA', 'EWR', 'DFW']
```

```
# Let us try to find the dijkstra path weighted by airtime
(approximate case)
shortpath = nx.dijkstra_path(FG, source='JAX', target='DFW',
weight='air_time')
Shortpath
```

Output:

```
['JAX', 'JFK', 'BOS', 'EWR', 'DFW']
```

References

- https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/

Manipulate View

- Creating Visuals & Charts using Python
- Pandas in Python
- Matplotlib in Python
- Getting data imported to Python
- Indexing in dataframes
- Pandas Advanced and Matplotlib
- Data cleaning : Empty value treatment in dataframes

Manipulate View

- Creating Visuals & Charts using Python
- Pandas in Python
- Matplotlib in Python
- Getting data imported to Python
- Indexing in dataframes
- Pandas Advanced and Matplotlib
- Data cleaning : Empty value treatment in dataframes

Creating Visuals & Charts using Python

- The topic is focused around the use of **Pandas** and **Matplotlib** libraries of **Python** for data manipulation and visualization.

Pandas in Python

- Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- When the values are in a table, the table in Python programming is called a **dataframe**. Basically, dataframes are nothing but variables with values in structured data format.
- Pandas library is used to view, manipulate (modify) and analyse dataframes which are 2-D or 3-D in nature.

Matplotlib in Python

- Matplotlib is a **plotting library** in Python. It is used for creating static, animated, and interactive visualizations in Python.
- In this topic, we will use some modules of matplotlib library to create line, bar, column and pie charts with customized formatting to visualize certain data.

Getting data imported to Python

- Pandas library supporting loading data from various online and offline sources to Python. It can fetch data from CSV, EXCEL, TEXT or from any website (URL) or servers as well.
- **pd** is an alias name for pandas.

Importing pandas library

- There are different pandas functions used to import different file types. To use any of those functions, the first thing we need to do is to import pandas library.

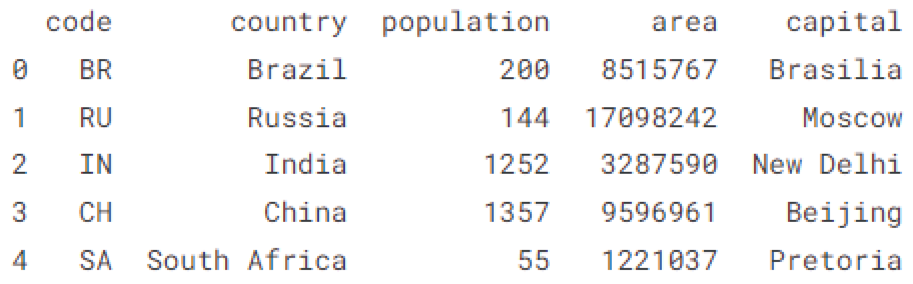
```
import pandas as pd
```

Dataset is Used: brics.csv

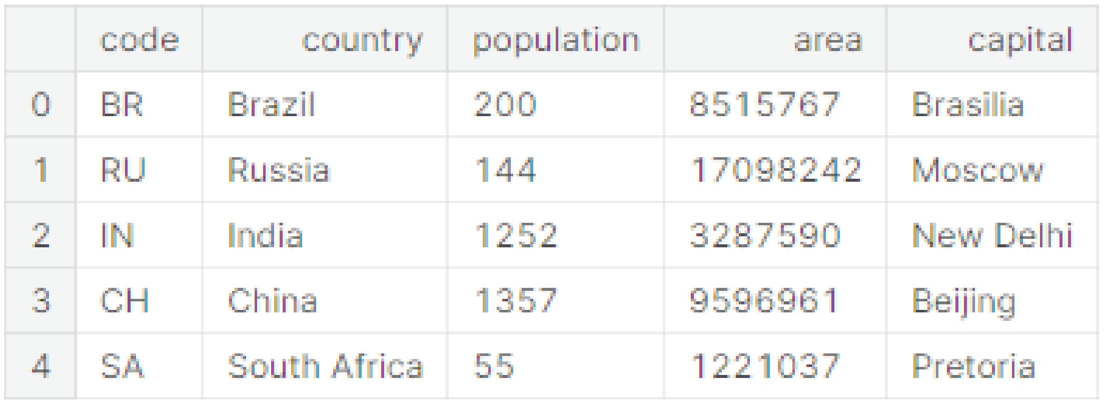
brics = pd.read_csv("../input/simple-data-manipulation-and-visualization/brics.csv") # importing brics.csv dataset and converting the table into a dataframe (pandas table).

print(brics) #print() dataframe using print()

brics # Printing dataset without print() Funtion.



```
  code       country  population      area    capital
0   BR        Brazil         200   8515767   Brasilia
1   RU        Russia         144  17098242     Moscow
2   IN         India        1252   3287590  New Delhi
3   CH         China        1357   9596961    Beijing
4   SA  South Africa          55   1221037   Pretoria
```



	code	country	population	area	capital
0	BR	Brazil	200	8515767	Brasilia
1	RU	Russia	144	17098242	Moscow
2	IN	India	1252	3287590	New Delhi
3	CH	China	1357	9596961	Beijing
4	SA	South Africa	55	1221037	Pretoria

Dataset is Used: brics.csv

- There is a difference between print() dataframe using print() and without print().
- Without print() function is just displayed the dataset.

```
# Data type of any variable can be seen using the function type()

print(type(5))
print(type('a text'))
print(type(True))
print(type(brics))

# datatype of the dataframe created above
<class 'int'>
<class 'str'>
<class 'bool'>
<class 'pandas.core.frame.DataFrame'>
```

Explore the dataframe

- Initial exploration should be done on any dataset to understand the variables/columns, values, data types and the structure of the data.

brics.info() # Object dtype can store variables with any or mixed data types

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
0    code        5 non-null      object
 1   country     5 non-null      object
 2   population  5 non-null      int64
 3   area        5 non-null      int64
 4   capital     5 non-null      object
dtypes: int64(2), object(3)
memory usage: 328.0+ bytes
```

Explore the dataframe

- The info() function is used to print a concise summary of a DataFrame.
- This method prints information about a DataFrame including the index dtype and column dtypes,non-null values and memory usage.

Explore the dataframe

brics.value_counts()

```
code  country       population   area           capital
SA    South Africa  55           1221037        Pretoria       1
RU    Russia        144          17098242       Moscow         1
IN    India         1252         3287590        New Delhi      1
CH    China         1357         9596961        Beijing        1
BR    Brazil        200          8515767        Brasilia       1
dtype: int64
```

value_counts() function returns object containing counts of unique values.
The resulting object will be in descending order so that the first element is the most frequently-occurring element.
Excludes NA values by default.

Explore the dataframe

# When the dataset is large and we want to sort the data by ascending or descending order to look at largest or smallest values.

# Ascending order by column code.
brics.sort_values(by=['code']) # Ascending by default.
or
brics.sort_values(by=['code'], ascending = True) # No need to mention

```
code  country                population      area       capital
0     BR      Brazil            200         8515767     Brasilia
3     CH      China             1357        9596961     Beijing
2     IN      India             1252        3287590     New Delhi
1     RU      Russia            144         17098242    Moscow
4     SA      South Africa  55  1221037  Pretoria
```

Explore the dataframe

```
# Sorting in descending order.
brics.sort_values(by=['code'], ascending = False)
```

```
        code    country         Population          area        capital
4       SA      South Africa       55               1221037     Pretoria
1       RU      Russia            144               17098242    Moscow
2       IN      India             1252              3287590     New Delhi
3       CH      China             1357              9596961     Beijing
0       BR      Brazil             200              8515767     Brasilia
```

- The sorting the data does not change the data in original dataframe.
- Order of the data has been changed with respect to the variable.
- It is sorted by the Code which is in alphabetical order.

Explore the dataframe

```
print(brics) # printing the dataframe to observe output

   code    country       population    area        capital
0  BR      Brazil         200          8515767     Brasilia
1  RU      Russia         144          17098242    Moscow
2  IN      India          1252         3287590     New Delhi
3  CH      China          1357         9596961     Beijing
4  SA      South Africa   55           1221037     Pretoria
```

Explore the dataframe

Sort the data to see which Country has Largest Population
(Population is in millions.)

```
brics.sort_values(by= 'population', ascending= False)
```

```
     code   country        population    area        capital
3    CH     China            1357       9596961      Beijing
2    IN     India            1252       3287590     New Delhi
0    BR     Brazil            200       8515767      Brasilia
1    RU     Russia            144      17098242      Moscow
4    SA     South Africa       55       1221037      Pretoria
```

Result : China has largest population.

Explore the dataframe

Sort the data to see which country is smallest in size.

```
brics.sort_values(by = 'area')
```

```
    code   country        population   area        capital
4   SA     South Africa   55           1221037     Pretoria
2   IN     India          1252         3287590     New Delhi
0   BR     Brazil         200          8515767     Brasilia
3   CH     China          1357         9596961     Beijing
1   RU     Russia         144          17098242    Moscow
```

Result: South Africa is smallest country in size.

Explore the dataframe

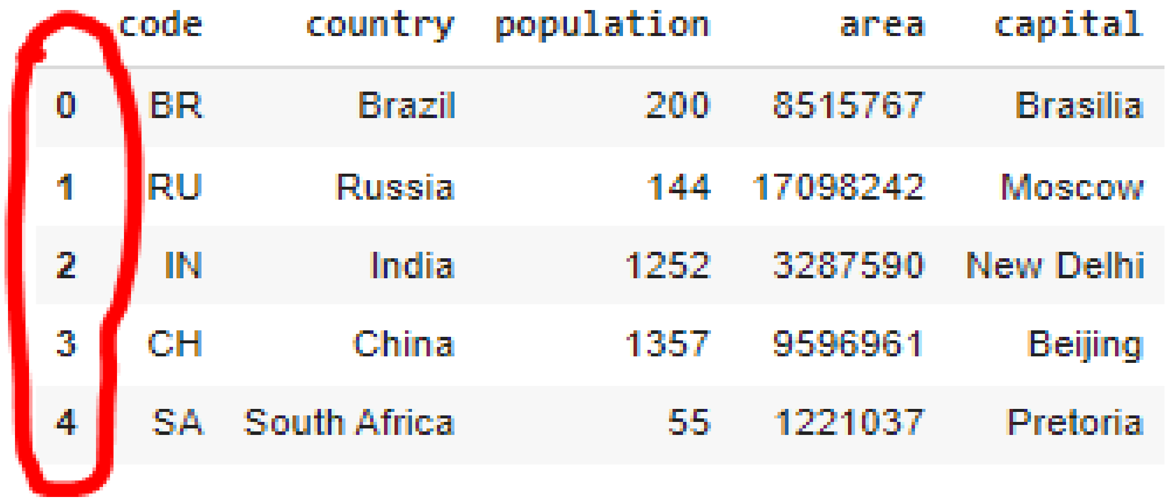
```
# Sorting and printing just one column.
brics.country.sort_values(ascending=False)
4    South Africa
1          Russia
2           India
3           China
0          Brazil
Name: country, dtype: object
```

Q. Sort & print area column on from largest to smallest.

```
brics.area.sort_values(ascending=False)
1    17098242
3     9596961
0     8515767
2     3287590
4     1221037
Name: area, dtype: int64
```

Indexing in dataframes

When we print a dataframe, a column with numbers 0, 1, 2... shows up at the left side. This column is called Index column. An index column indicates the position/row of records.



	code	country	population	area	capital
0	BR	Brazil	200	8515767	Brasilia
1	RU	Russia	144	17098242	Moscow
2	IN	India	1252	3287590	New Delhi
3	CH	China	1357	9596961	Beijing
4	SA	South Africa	55	1221037	Pretoria

If our data contains a column with unique values such as StudentID, EmployeeID, OrderID etc., we can make that specific column (with unique values) as our index column.

In this case, we can make code column or even country column as index column

Indexing in dataframes

```
# To fetch data from the 2nd row, we can use index 1 (one)

brics.country[1]

'Russia‘
```

Indices can be used to fetch data from a particular row.

```
# Getting single column from a dataframe

print(brics["country"])
0          Brazil
1          Russia
2           India
3           China
4    South Africa
Name: country, dtype: object
```

Indexing in dataframes

```
# Print area column from brics dataframe

print(brics["area"])
0     8515767
1    17098242
2     3287590
3     9596961
4     1221037
Name: area, dtype: int64
# Other way to get a single column from a dataframe - use dot

brics.country
0          Brazil
1          Russia
2           India
3           China
4    South Africa
Name: country, dtype: object
```

Indexing in dataframes

```
# Print code column from the dataframe using dot

brics.code
0    BR
1    RU
2    IN
3    CH
4    SA
Name: code, dtype: object
# Printing multiple columns

brics[['country', 'capital']]
          country          capital
0          Brazil         Brasilia
1          Russia          Moscow
2          India          New Delhi
3          China            Beijing
4          South Africa     Pretoria
```

Indexing in dataframes

```
# Print country, capital and population

brics[['country','capital','population']]

        Country          Capital    Population
0        Brazil         Brasilia       200
1        Russia           Moscow      144
2         India        New Delhi     1252
3         China          Beijing      1357
4  South Africa         Pretoria        55

##### It is not advisable to use dot everywhere to print column.
##### Using square brackets to print one column is also correct
way.
```

Indexing in dataframes

```
# use col 0 as index.

brics = pd.read_csv("../input/simple-data-manipulation-and-visualization/brics.csv",index_col=0)
brics
```

```
      country       population    area        capital
code
BR    Brazil        200           8515767     Brasilia
RU    Russia        144           17098242    Moscow
IN    India         1252          3287590     New Delhi
CH    China         1357          9596961     Beijing
SA    South Africa  55            1221037     Pretoria
```

Col 0 as index will remove the 1st column containing the index or sl.no. from the above output.
And make the 1st column from actual data as the index.

Transforming the dataframe

```
#Adding a new column to the existing Data Frame

brics["on_earth"] = [True, True, True, True, True]
Brics

        country       population     area       capital      on_earth
code
BR      Brazil           200        8515767     Brasilia      True
RU      Russia          144        17098242     Moscow        True
IN      India           1252        3287590     New Delhi     True
CH      China           1357       9596961       Beijing      True
SA      South Africa     55         1221037      Pretoria     True
```

Transforming the dataframe

```
# adding a calculated column

brics["density"] = brics["population"] / brics["area"] * 1000000   # per km sq.
brics
      country       population  area       capita     on_earth  density
code
BR    Brazil        200         8515767    Brasilia   True      23.485847
RU    Russia        144         17098242   Moscow     True      8.421918
IN    India         1252        3287590    New Delhi  True      380.826076
CH    China         1357        9596961    Beijing    True      141.398928
SA    South Africa   55         1221037    Pretoria   True      45.043680
```

Transforming the dataframe

Removing the additional decimal points or setting the precision can be done on the entire dataframe.

brics.style.set_precision(2) # Upto 2 decimal points for all float columns in the dataframe.

```
      country          population     area       capital          on_earth   density
code
BR        Brazil            200          8515767     BrasiliaTrue       True        23.49
RU        Russia            144          17098242    Moscow             True        8.42
IN        India            1252          3287590     New Delhi          True        380.83
CH        China            1357            9596961    Beijing           True        141.40
SA        South Africa          55         1221037    Pretoria          True        45.04
```

This will not change the original dataframe i.e. the original dataframe will still have multiple digits after decimals.

Transforming the dataframe

brics["density"].round(2) # Rounding of individual column to fixed decimal points.

```
code
BR     23.49
RU      8.42
IN    380.83
CH    141.40
SA     45.04
Name: density, dtype: float64
```

Fetching data using indices and location functions (.loc and .iloc)

- .loc is used with the actual value/label of the index
- .iloc is used with the index position/reference - which are numbers (0, 1, 2, 3, 4......)

```
brics
      country      population       area       capital     on_earth      density
code
BR    Brazil          200         8515767     Brasilia       True       23.485847
RU    Russia          144        17098242     Moscow         True        8.421918
IN    India          1252         3287590     New Delhi      True      380.826076
CH    China           1357         9596961    Beijing        True      141.398928
SA    South Africa  55             1221037    Pretoria       True       45.043680
```

Notice that the indices are BR, RU, IN, CH and SA. However, their position(numerical index) are 0,1,2,3, and 4

Fetching data using indices and location functions (.loc and .iloc)

```
brics.loc["SA"]   # .loc is used with actual value of the index.

country            South Africa
population              55
area               1221037
capital            Pretoria
on_earth             True
density            45.0437
Name: SA, dtype: object
```

The value mentioned in the argument must be there in the dataframe's index list.

Fetching data using indices and location functions (.loc and .iloc)

```
brics.iloc[4]  # .loc is used when we are giving the index reference as number, not the actual value of the index.
```

```
country       South Africa
population              55
area             1221037
capital          Pretoria
on_earth            True
density          45.0437
Name: SA, dtype: object
```

Fetching data using indices and location functions (.loc and .iloc)

Use .loc to print the values from all columns for IN.

```
brics.loc['IN']
```

```
country           India
population         1252
area            3287590
capital       New Delhi
on_earth           True
density         380.826
Name: IN, dtype: object
```

Fetching data using indices and location functions (.loc and .iloc)

Use .iloc to print the values from all columns from 3rd row (index number will be 1 less than the row number).

```
brics.iloc[2]
```

```
country            India
population          1252
area             3287590
capital        New Delhi
on_earth            True
density          380.826
Name: IN, dtype: object
```

Adding rows in the dataframe

```
#adding a new ROW using append

newrow = {'code':'WK', 'country':'Wakanda',
'population':5,'area':1000000,'capital':'Wakanda
City','on_earth':False,'density':5}


brics1 = brics.append(newrow,ignore_index= True)   # Ignoring the
existing index.

# Ignoring the existing index will result into new dataframe index
0,1,2,....


brics1   # append will not result into changing the actual dataframe.
brics will still be the same.
```

Adding rows in the dataframe

```
      country      population  area      capital       on_earth   density      code
0     Brazil          200      8515767   Brasilia       True      23.485847    NaN
1     Russia          144     17098242   Moscow        True       8.421918     NaN
2     India          1252     3287590    New Delhi     True       380.826076   NaN
3     China          1357    9596961     Beijing        True      141.398928   NaN
4     South Africa   55      1221037      Pretoria     True       45.043680    NaN
5    Wakanda          5      1000000  Wakanda City  False       5.000000     WK
```

- There is NaN for code in other rows, Because the previous 'code' was not a part of data table. It was an index.

- With 'code' in 'newrow', we are adding a new column called 'code'.

- ignore_index=False will result into error because 'newrow' is a dictionary and appending a dictionary into a dataframe will not work if we do not ignore the index.

Adding a new row using .loc

```
brics.loc['WK'] = ['Wakanda', 5, 1000000, 'Wakanda City', False, 5]

# Sequence is very important. Data types may get changed due to wrong
sequence.

Brics

country  population          area      capital    on_earth             density
code
BR       Brazil      200     8515767Brasilia   True       23.485847
RU       Russia      144     17098242          Moscow True       8.421918
IN       India       1252    3287590New Delhi          True       380.826076
CH       China       1357    9596961Beijing    True       141.398928
SA       South Africa        55         1221037Pretoria  True       45.043680
WK       Wakanda             5          1000000Wakanda City        False      5.000000
```

Add a new row in the dataframe using .loc any other country with some values in all columns

```
brics.loc['GD'] = ['Gondor', 3, 1855000 ,'Minas Tirith',False,2.214022]
```

Fetching the value in 2-D way

```
brics.loc["IN","capital"]
```

```
'New Delhi‘
```

```
brics["capital"].loc["IN"]
```

```
'New Delhi‘
```

```
brics.loc["IN"]['capital']
```

```
'New Delhi'
```

Deleting rows and columns from a dataframe using drop() function.

```
# create a new dataframe so that brics is not affected.

brics1 = brics
Brics1

        country       population  area       capital        on_earth      density
code
BR      Brazil          200       8515767    Brasilia         True        23.485847
RU      Russia          144       17098242   Moscow           True         8.421918
IN      India           1252      3287590    New Delhi        True       380.826076
CH      China           1357      9596961    Beijing          True       141.398928
SA      South Africa    55        1221037    Pretoria         True        45.043680
WK      Wakanda         5         1000000    Wakanda City     False        5.000000
GD      Gondor          3         1855000    Minas Tirith     False        2.214022


brics1.drop(['area'], axis=1)  # axis = 0 by default for rows. Axis = 1 for column.
```

For multiple columns, mention the columns separated by comma

```
df.drop(['name','max'], axis=1)
```

```
      country      population  capital       on_earth      density
code
BR    Brazil       200         Brasilia      True    23.485847
RU    Russia       144         Moscow        True    8.421918
IN    India        1252        New Delhi     True    380.826076
CH    China        1357        Beijing       True    141.398928
SA    South Africa 55          Pretoria      True    45.043680
WK    Wakanda      5           Wakanda City          False    5.000000
GD    Gondor       3           Minas Tirith  False   2.214022
```

Deleting area and density columns from brics1 dataframe.

```
brics1 = brics

brics1.drop(['area','density'], axis = 1)
```

```
        country       population   capital          on_earth
code
BR      Brazil          200         Brasilia          True
RU      Russia         144         Moscow            True
IN      India          1252        New Delhi         True
CH      China          1357        Beijing           True
SA      South Africa  55           Pretoria          True
WK      Wakanda         5          Wakanda City      False
GD      Gondor          3          Minas Tirith      False
```

Delete the record for Wakanda by using its index WK.

```
brics1.drop(['WK'], axis = 0)

          country       population   area        capital        on_earth  density
code
BR        Brazil           200       8515767    Brasilia         True     23.485847
RU        Russia           144       17098242   Moscow           True     8.421918
IN        India            1252      3287590    New Delhi        True     380.826076
CH        China            1357      9596961    Beijing          True     141.398928
SA        South Africa     55        1221037    Pretoria         True     45.043680
GD        Gondor           3         1855000    Minas Tirith     False    2.214022
```

Delete the record for Wakanda by using its index WK.

```
brics1 = brics

# Another way to drop columns

brics1.drop(columns=['area', 'density'])
```

```
        country       population   capital         on_earth
code
BR      Brazil        200          Brasilia        True
RU      Russia        144          Moscow          True
IN      India         1252         New Delhi       True
CH      China         1357         Beijing         True
SA      South Africa  55           Pretoria   True
WK      Wakanda       5            Wakanda City    False
GD      Gondor        3            Minas Tirith    False
```

Drop Rows by Index

```
brics1 = brics

# Drop rows by index

brics1.drop(['IN', 'CH'])
```

```
        country       population  area       capital        on_earth    density
code
BR      Brazil         200       8515767    Brasilia         True       23.485847
RU      Russia         144      17098242    Moscow           True        8.421918
SA      South Africa    55       1221037    Pretoria         True       45.043680
WK      Wakanda          5       1000000    Wakanda City     False       5.000000
GD      Gondor           3       1855000    Minas Tirith     False       2.214022
```

Drop Rows by Index

- The shape function() helps us to find the shape or size of an array or matrix. In Excel - A1:D10.
- shape[0] means we are working along the first dimension of an array.
- If Y has n rows and m columns, then Y.shape is (n,m). So Y.shape[0] is n.

```
for index in range(brics.shape[0]):
   countryName = brics.iloc[index,0]  # row - index, column - 0
   cityName = brics.iloc[index, 3]  # row - index, column - 3
   print('The Capital City of', countryName, 'is', cityName)
```

```
The Capital City of Brazil is Brasilia
The Capital City of Russia is Moscow
The Capital City of India is New Delhi
The Capital City of China is Beijing
The Capital City of South Africa is Pretoria
The Capital City of Wakanda is Wakanda City
The Capital City of Gondor is Minas Tirith
```

Drop Rows by Index

```
#Another solution

for index in range(brics.shape[0]):

    print('The Capital City of', brics.iloc[index, 0], 'is', 
brics.iloc[index, 3])
```

```
The Capital City of Brazil is Brasilia
The Capital City of Russia is Moscow
The Capital City of India is New Delhi
The Capital City of China is Beijing
The Capital City of South Africa is Pretoria
The Capital City of Wakanda is Wakanda City
The Capital City of Gondor is Minas Tirith
```

Drop Rows by Index

```
#Another solution
# iterrows() is a generator that iterates over the rows of the
dataframe and returns the index of each row,
# in addition to an object containing the row itself.

for index, row in brics.iterrows():
    print("The Capital City of",row['country'],"is", row['capital'])
```

```
The Capital City of Brazil is Brasilia
The Capital City of Russia is Moscow
The Capital City of India is New Delhi
The Capital City of China is Beijing
The Capital City of South Africa is Pretoria
The Capital City of Wakanda is Wakanda City
The Capital City of Gondor is Minas Tirith
```

Pandas Advanced and Matplotlib

TO DO: Show the shape of the dataframe

```
marks.shape
(20, 7)
```

Get the information and structure of the data with non-null value counts

```
marks.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Student_ID      20 non-null     object 
 1   Student_Name    20 non-null     object 
 2   English         20 non-null     float64
 3   Maths           20 non-null     float64
 4   Science         20 non-null     float64
 5   History         20 non-null     float64
 6   Social_Studies  20 non-null     float64
dtypes: float64(5), object(2)
memory usage: 1.2+ KB
```

Print datatypes of the each column

```
marks.dtypes
```

```
Student_ID        object
Student_Name      object
English          float64
Maths            float64
Science          float64
History          float64
Social_Studies   float64
dtype: object
```

A column of a dataframe is called a SERIES. The type of a series is Series.

```
 print(type(marks))
print(type(marks.English))  # Column's dtype is Series, the dtype
of the values is Object.
print(type(5))
print(type(5.5))
print(type("Python"))
print(type(True))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
```

display() Vs print()

- display() is similar to print() with some difference.
- Useful in advanced Python when we can 'display' images, but not 'print' them.
- When working on a dataframe, prefer to use display()

```
# Using display()
display(marks)
Student_ID	Student_Name	English	Maths	Science	History	Social_Studies
0	S01	Alice	96.874050	15.367999	58.682033
	59.690510	61.373070
1	S02	Bob	5.462131	81.032680	83.171957
	45.664012	80.805730
2	S03	Charlie	60.753099	65.219548	99.323628
	24.281512	58.057612
3	S04	David	21.160844	95.052971	35.860852
	62.865185	95.264572
```

References- Manipulate view

- https://www.kaggle.com/code/blossome568/data-manipulation-visualization-using-python

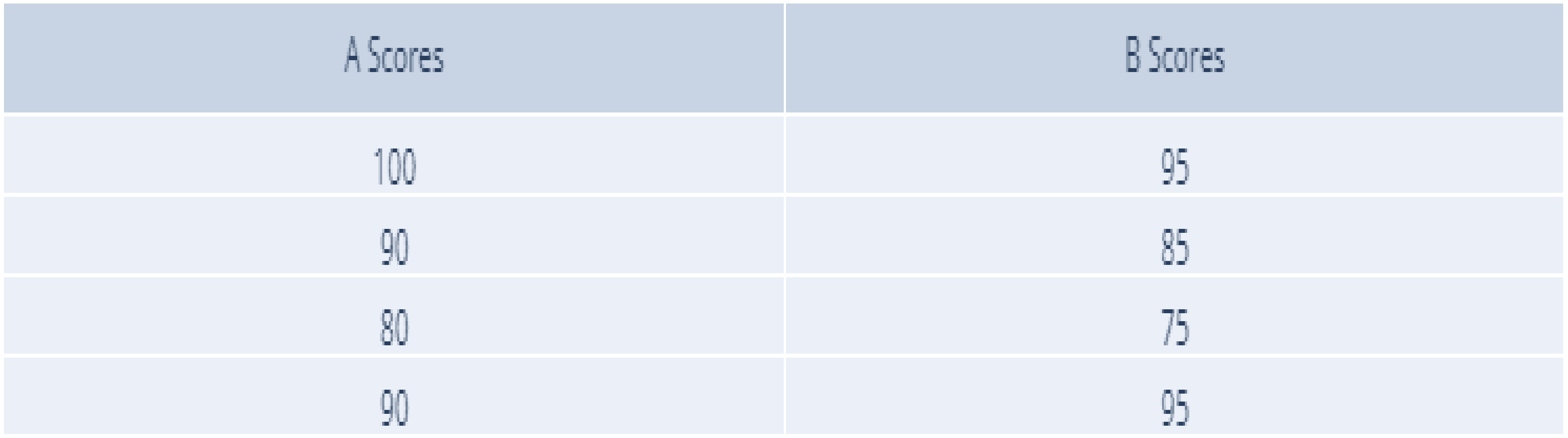
Tables in Python

- How to make tables in Python with Plotly.
- Plotly is a free and open-source graphing library for Python. We recommend you read our Getting Started guide for the latest installation or upgrade instructions, then move on to our Plotly Fundamentals tutorials or dive straight in to some Basic Charts tutorials.
- go.Table provides a Table object for detailed data viewing. The data are arranged in a grid of rows and columns. Most styling can be specified for header, columns, rows or individual cells. Table is using a column-major order, ie. the grid is represented as a vector of column vectors.

Basic Table

```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Table(header=dict(values=['A
Scores', 'B Scores']),

cells=dict(values=[[100, 90, 80, 90], [95, 85, 75, 95]]))

])
fig.show()
```

Basic Table

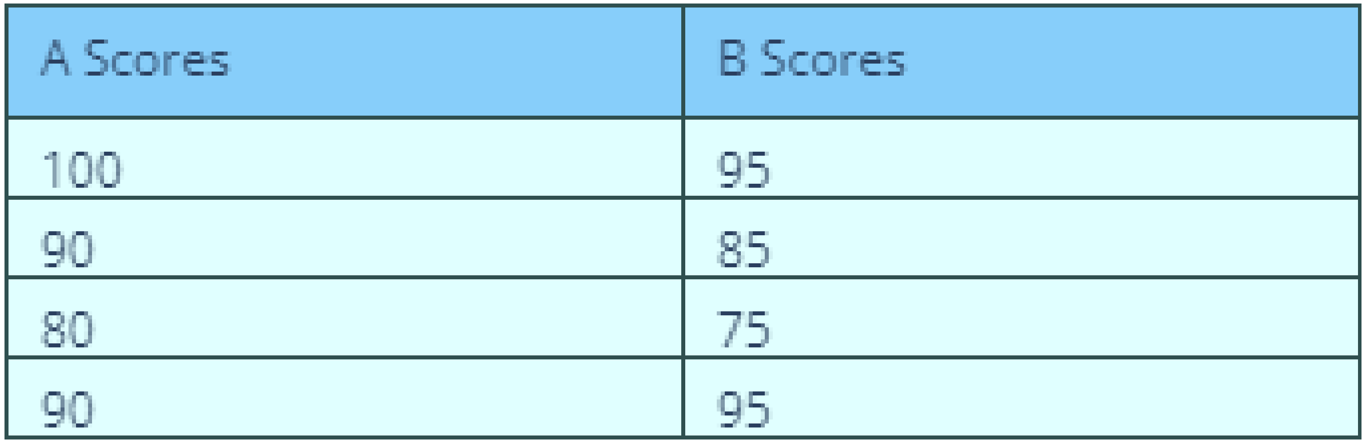


A Scores	B Scores
100	95
90	85
80	75
90	95

Styled Table

```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Table(
    header=dict(values=['A Scores', 'B Scores'],
                line_color='darkslategray',
                fill_color='lightskyblue',
                align='left'),
    cells=dict(values=[[100, 90, 80, 90], # 1st column
                       [95, 85, 75, 95]], # 2nd column
               line_color='darkslategray',
               fill_color='lightcyan',
               align='left'))
])

fig.update_layout(width=500, height=300)
fig.show()
```

Styled Table

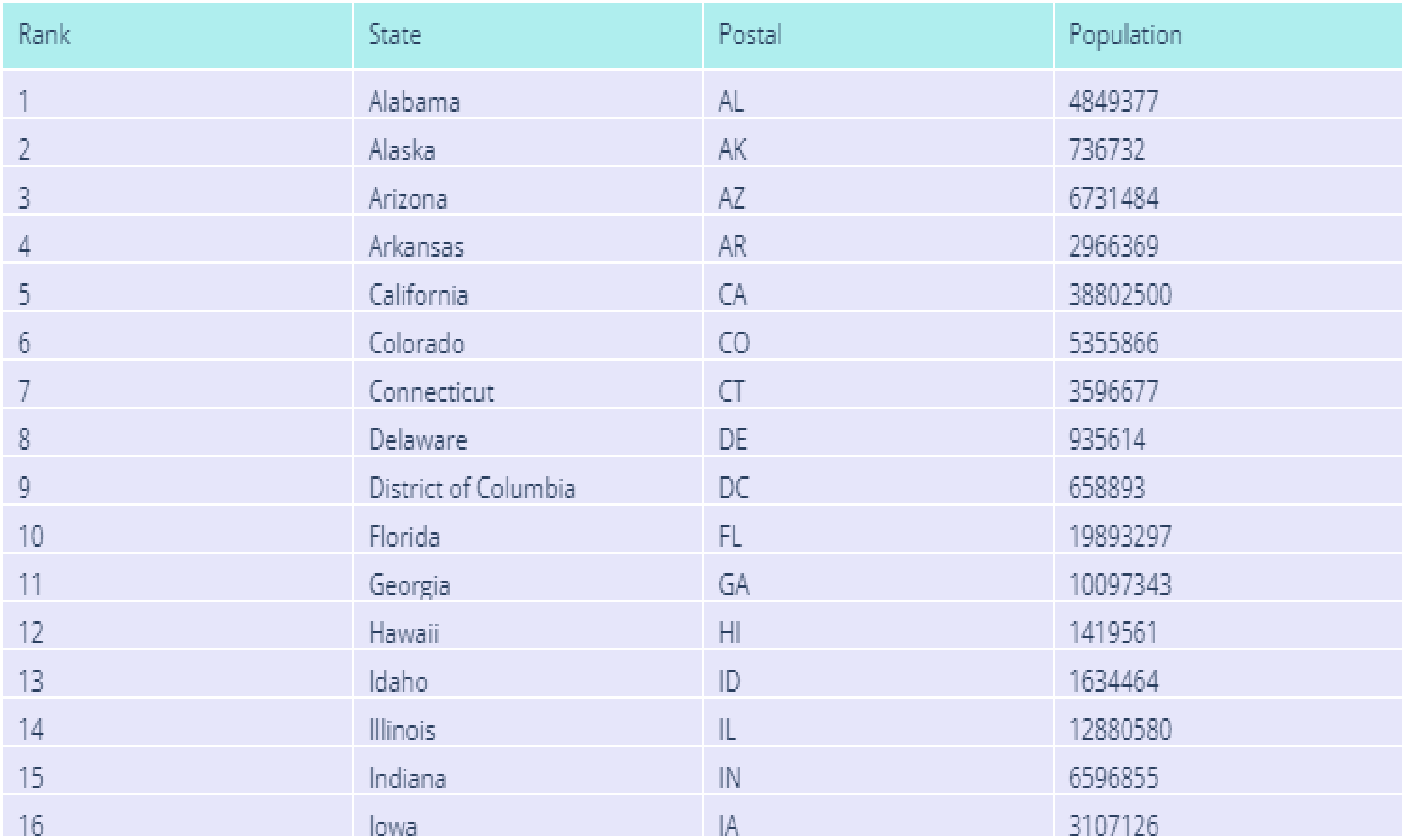


A Scores	B Scores
100	95
90	85
80	75
90	95

Use a Pandas Dataframe

```
import plotly.graph_objects as go
import pandas as pd

df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/m
aster/2014_usa_states.csv')

fig = go.Figure(data=[go.Table(
    header=dict(values=list(df.columns),
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[df.Rank, df.State, df.Postal, df.Population],
               fill_color='lavender',
               align='left'))
])

fig.show()
```

Use a Pandas Dataframe



Rank	State	Postal	Population
1	Alabama	AL	4849377
2	Alaska	AK	736732
3	Arizona	AZ	6731484
4	Arkansas	AR	2966369
5	California	CA	38802500
6	Colorado	CO	5355866
7	Connecticut	CT	3596677
8	Delaware	DE	935614
9	District of Columbia	DC	658893
10	Florida	FL	19893297
11	Georgia	GA	10097343
12	Hawaii	HI	1419561
13	Idaho	ID	1634464
14	Illinois	IL	12880580
15	Indiana	IN	6596855
16	Iowa	IA	3107126

References- Arrange Tables

- https://plotly.com/python/table/

References- Reduce items and attributes

- https://builtin.com/machine-learning/how-to-preprocess-data-python
- https://www.kdnuggets.com/2020/07/easy-guide-data-preprocessing-python.html
- https://www.kaggle.com/code/ajay1216/practical-guide-on-data-preprocessing-in-python
- https://blog.quantinsti.com/data-preprocessing/
- https://medium.datadriveninvestor.com/data-preprocessing-3cd01eefd438
- https://iwconnect.com/data-preprocessing-for-machine-learning-in-python/