School of Information Studies

ADVANCED TOPIC PRESENTATION

IST – 652

SCRIPTING FOR DATA ANALYSIS

FALL 2022

PREPARED BY : **HENDI KUSHTA**

Table of Contents

Table of Figures

Matplotlib Introduction

Matplotlib is the most widely used Python library for creating plots and other two-dimensional data visualizations. It was initially developed by John D. Hunter, and a group of developers now looks after it. It is intended for writing stories that are suitable for publishing. Although Python programmers have access to several visualization tools, matplotlib is the most used and, as a result, has generally good ecosystem integration.

If you do not have matplotlib installed, you can install as mentioned below:

- using pip	- using conda
pip install matplotlib	conda install matplotlib

Our project has been divided into two parts. The first half of this article introduces matplotlib and gives several examples of how to make straightforward visualizations, and the second part gives a practical application of matplotlib.

First Graph

First, we have imported the necessary libraries for our project like matplotlib.pyplot, numpy and pandas. We have used numpy and pandas in small fractions in our project. We used abbreviations for each library imported. Plt for matplotlib, np for numpy and pd for pandas.

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

Figure 1: Import libraries

We began our project by creating our first graph. First, we made two lists, x and y, which contain the values for the x- and y-axes, respectively. There are 2 ways how we can plot a graph: plot the graph using arguments or plot the graph using shorthand notation.

```
# Plot our graph using arguments
plt.plot(x,y, label='2x', color='blue', linewidth=2, marker='.',
                        markersize=10, linestyle='--', markeredgecolor='blue')
```

```
# Use shorthand notation to create a graph
plt.plot(x,y,'bs--',label='2x')
```

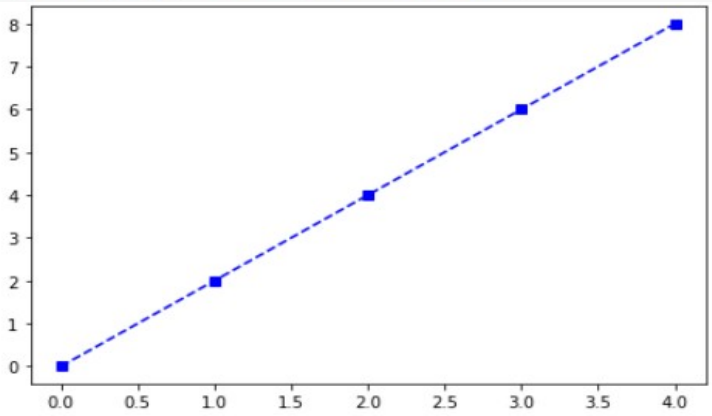

Figure 2: Simple graph

As seen below, we add a title, label the x- and y-axes, and plot a legend to make our graph more illustrative. Font dictionary is used to format the titles fonts.
Values on the x-axis are decimal numbers, as can be seen in the first figure. In place of decimal values, we use xticks to add integer values, and yticks to display only even numbers on the y-axis.

```
# add a title to our graph
plt.title("Our first graph", fontdict = {'fontname':'Noto Mono','fontsize':15})

# name our x and y axis.
plt.xlabel("X - Axis")
plt.ylabel("Y - Axis")

# plot the legend
plt.legend()

# if we want int numbers
plt.xticks([0,1,2,3,4])
plt.yticks([0,2,4,6,8,10])
```

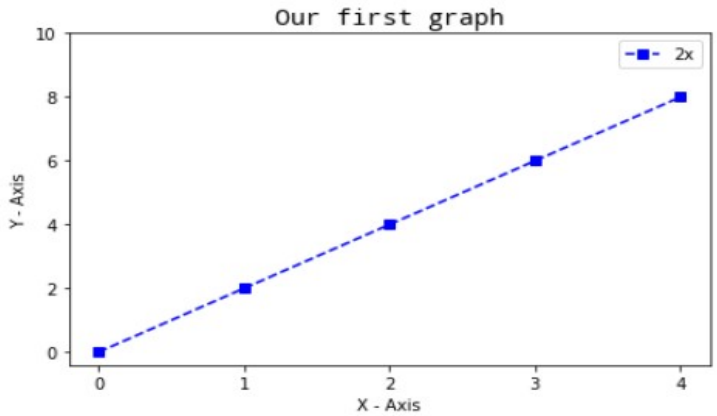

Figure 3: Add title, legend and x and y axis labels

We have added a 2nd line in our graph which takes as parameters x2 which returns evenly spaced values from 0 to 4.5 increasing by 0.5 using numpy and x2 square as y-axis. The second line is represented in red color.

```
### Line 2

# Select interval we want to plot points at
x2 = np.arange(0,4.5,0.5)

# Plot second line named X^2
plt.plot(x2, x2**2, 'r^--', label='X^2')
```

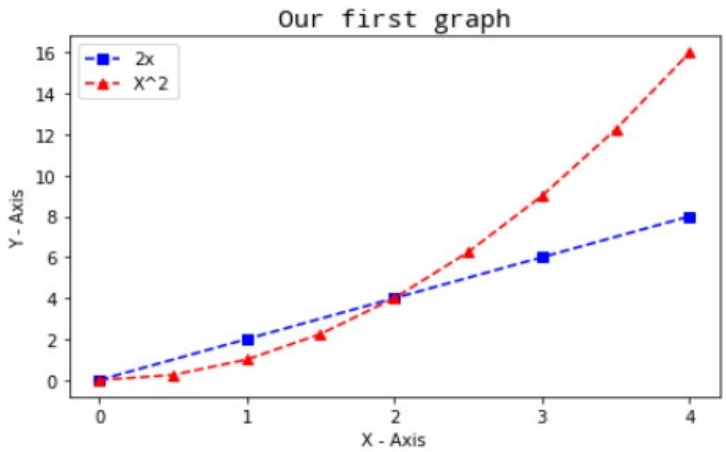

Figure 4: Add a second line to our simple graph

To resize the graph, we use figsize and choose the measure in inches the figure will be created. We have chosen 5 inches by 3 with resolution in dots equal to 300. The total size of the figure will be 1500 x 1300 pixels.
To save the graph in the working directory, we use savefig and give a name to the graph.

```
# Resize our graph
plt.figure(figsize=(5,3), dpi=300) # pixels per inch 300 and figsize will be 5 inches by 3
                                   # so the total figure size will be 1500 pixels
                                   # by 900 pixels


# to save the graph
plt.savefig('mygraph.png', dpi=300)
```

Figure 5: Save and resize the figure

Bar Chart

A bar chart, also known as a bar graph, is a type of chart or graph that displays categorical data using rectangular bars with heights or lengths proportional to the values they represent. The bars can be plotted horizontally or vertically.
To plot a bar chart using matplotlib we just write plt.bar and assign the values that will be used in it.

```
# create 2 lists, one for the x-axis, another for y-axis.
labels = ['A', 'B' , 'C']
values = [1,4,2]

# plot bar chart
plt.bar(labels, values)
```

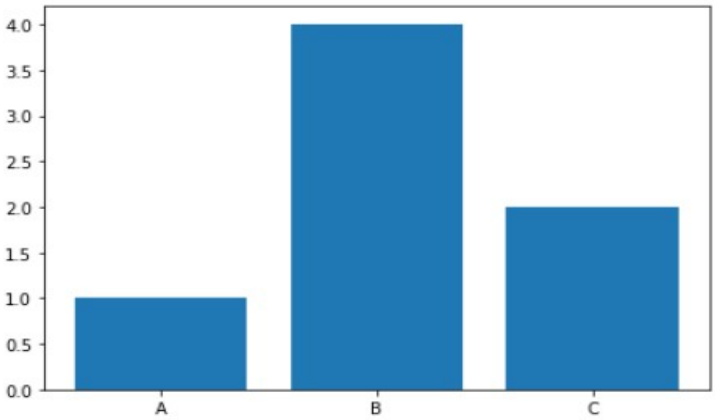

Figure 6: Plot a bar chart

To make our chart more visible, we can assign colors to our bars, patterns etc.
We have assign red, blue and green colors to the bars. After that we have created a list of patterns that include /, * and o and assigned these patterns to each bar.

```
# create 2 lists, one for the x-axis, another for y-axis.
labels = ['A', 'B' , 'C']
values = [1,4,2]

# plot bar chart and assign to bars variable
bars = plt.bar(labels, values, color=['r', 'g', 'b'])

patterns = ['/', '*', 'o'] # create a list of patterns
for bar in bars: # for each bar in the created bars,
    bar.set_hatch(patterns.pop(0)) # pop to assign each pattern element to each of the bars

# bars[0].set_hatch('/')
# bars[1].set_hatch('o')
# bars[2].set_hatch('*')

# add a title to our graph
plt.title("Bar Chart", fontdict = {'fontname':'Noto Mono','fontsize':15})

# name our x and y axis.
plt.xlabel("Labels")
plt.ylabel("Values")

# if we want int numbers
plt.yticks([0,1,2,3,4])


# plt.figure(figsize = (6,4), dpi=300)

plt.show()
```

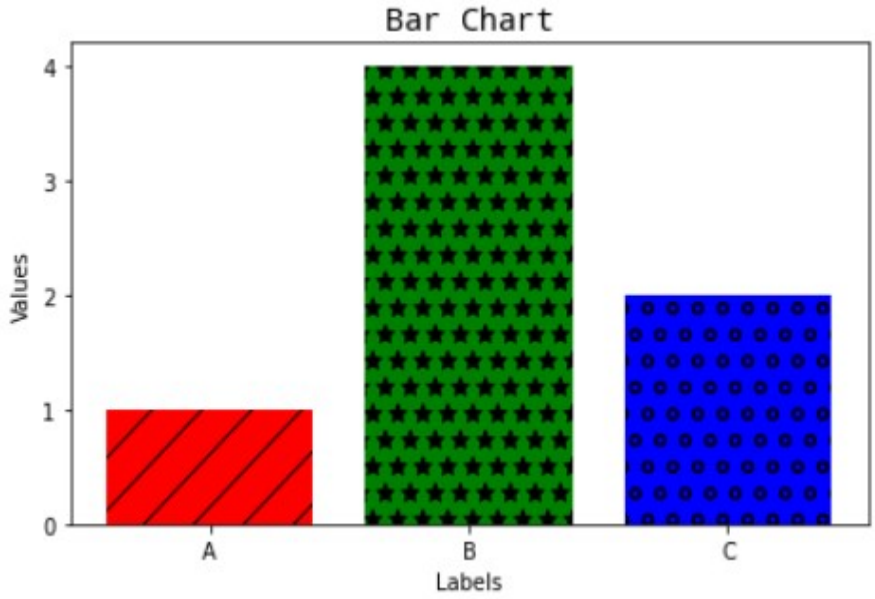

Figure 7: Add colors, paterns, labels, title to bar chart

Subplots

Subplots are collections of axes that can be found in a single matplotlib figure. The matplotlib library's subplots() function aids in the creation of multiple subplot layouts. It gives you control over all of the individual plots you create.

plt.subplots() is a function that returns a tuple containing a figure and axes object(s). Thus when using fig, ax = plt.subplots() you unpack this tuple into the variables fig and ax.
Fig is like the canvas where we want to draw and ax are the plots built in it.
Having fig is useful if you want to change figure-level attributes or save the figure as an image file later (e.g. with fig.savefig('yourfilename.png'))

Subplots can be performed in a variety of ways. We begin by separating the axes from one another. We have created two plots in one figure in one line, as seen in the code below.
To demonstrate how to manipulate and work with each of the subplots, we colored the canvas background red, the first graph blue, and the second graph orange.

```
fig, (ax1, ax2) = plt.subplots(1,2) # 2 plots in 1 line
fig.set_facecolor("red") # red color to the canvas/ figure
ax1.set_facecolor("lightblue") # lightblue color to background of first axis
ax1.bar(labels, values) # plot the bar chart in the first axis
ax2.set_facecolor("orange") # orange color to background of second axis
ax2.plot(labels, values) # plot the line graph to the second
plt.show()
```

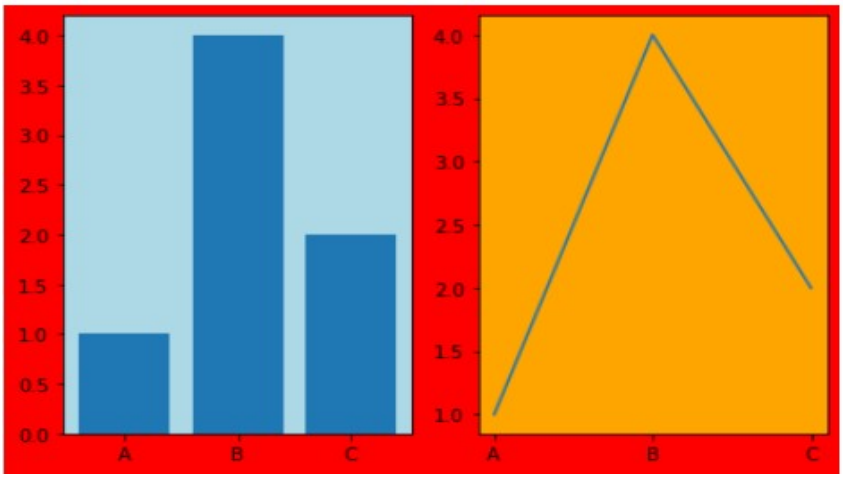

Figure 8: First way of sub-plotting

The second method for working with subplots is to assign only ax when creating the subplot and access them via their coordinates.
In the figure below, we have created a 2 x 2 subplot. That is, there are two rows and two columns.

```
fig, ax = plt.subplots(2,2)  # 2 plots in each of 2 line 2x2
ax[0,0].set_facecolor("green") # access the first subplot in the first row
ax[0,1].set_facecolor("blue") # access the second subplot in the first row
ax[1,0].set_facecolor("yellow") # access the first subplot in the second row
ax[1,1].set_facecolor("red") # access the second subplot in the second row
fig.set_facecolor("grey")
```

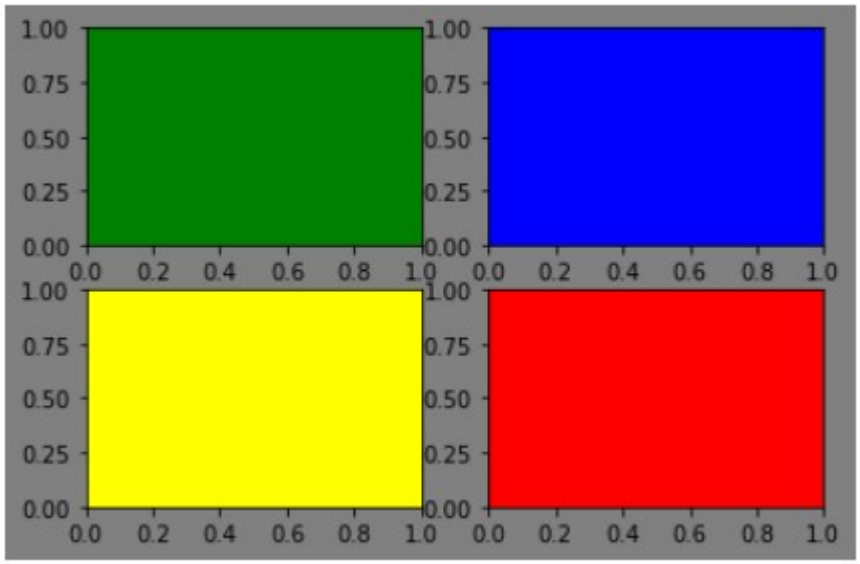

Figure 9: Second way of sub-plotting

Plt.subplot(x,y,n) is another way to perform subplots in matplotlib, where x is the number of rows, y is the number of columns, and n is the subplot we want to work with.

```
# create 2 lists, names used in x- axis and values used in y-axis
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(12, 3)) # choose the figure size

plt.subplot(131) # access the first plot from 3 columns in the first line
plt.bar(names, values) # build a bar chart
plt.subplot(132) # access the second plot from 3 columns in the first line
plt.scatter(names, values) # build a scatter plot
plt.subplot(133)  # access the third plot from 3 columns in the first line
plt.plot(names, values) # build a line graph
plt.suptitle('Categorical Plotting') # give a name for the figure
plt.show()
```

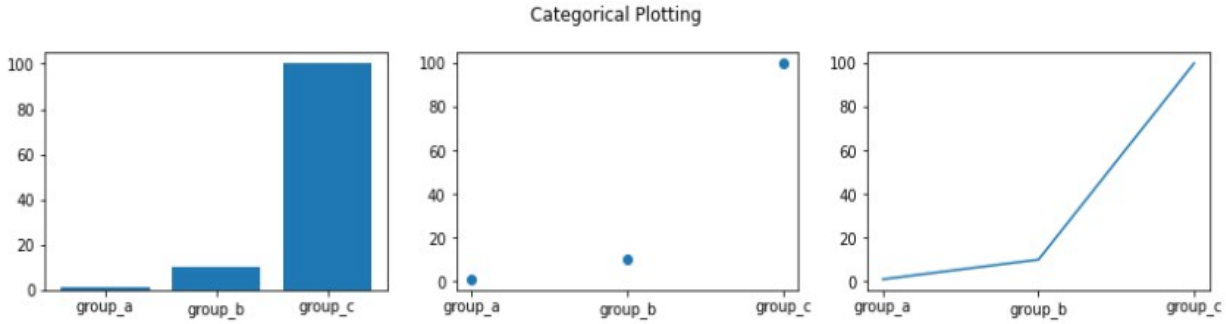

Figure 10: Third way of sub-plotting

Examples

The second part of our project consists of 3 different examples.

Cost of Living USA

On our first example we have made an analysis about the living cost in the United States of America. The data covers second quarter of 2022 and was taken from https://meric.mo.gov/data/cost-living-data-series. For our project we have chosen only the name of states and Index. Indexes are designed to compare the costs that an average person can expect to incur in different regions for food, shelter, transportation, energy, clothing, education, healthcare, childcare, and entertainment.
Our data is shown in a dictionary named data. The first thing we do is to create 2 lists; 1 for the dictionary keys named group_names and one for the values named group_data.
Since we will separate the countries with the highest index of cost of living from those with the lowest, we find also the average named group_mean of group_data by using numpy.

```
data = {'Missisipi': 84.9,
        'Oklahoma': 85.7,
        'Kansas': 86.4,
        'Alabama': 87.5,
        'Georgia': 87.8,
        'West Virginia': 88.5,
        'Missouri': 88.6,
        'Indiana': 88.9,
        *
        *
        *
        'California': 139.8,
        'Massachusetts': 147.9,
        'District of Colombia': 154.5,
        'Hawaii':  189.9
       }
group_data = list(data.values()) # from our data, which are a dictionary in this case,
                                 # take the values of dictionary and create a list named
                                 # group_data with the indexes of cost of living
group_names = list(data.keys())  # take the keys of dictionary and create a list named
                                 # group_names with the country names

group_mean = np.mean(group_data) # find the mean of indexes
```

Figure 11: Read the data, crate lists of keys and values and find the mean of values

To plott our data in a horizontal bar chart, where y-axis are the states or group_names and on the x-axis, the index of cost of living. We have separated the bars in two different colors green and red. Green bars are the countries whose index of cost of living is lower than the average, and red the ones whose index is higher. We also plotted a vertical line in the graph to show where the mean lies.

```
fig, ax = plt.subplots(figsize=(20,10)) # plt.subplots() is a function that returns a tuple
                        # containing a figure and axes object(s). Thus when using fig, ax = plt.subplots()
                        # you unpack this tuple into the variables fig and ax.

colors = [] # create a colors list
for value in group_data:
    if value < group_mean: # check if an index is lower than the average of indexes
        colors.append('g') # adds green color to colors list
    else:
        colors.append('r') # adds red color to colors list

plt.barh(group_names, group_data, color=colors) # plot a horizontal bar chart with 2 colors

labels = ax.get_xticklabels()  # To gain access on the x-axis
plt.setp(labels, rotation=0, horizontalalignment='right') # set the property of many items at once

labels = ax.get_yticklabels() # To gain access on the y-axis
plt.setp(labels, rotation=0, horizontalalignment='right') # set the property of many items at once

# add a title to our graph
plt.title("Cost of Living Index", fontdict = {'fontname':'Noto Mono','fontsize':15})

# name our x and y axis.
plt.xlabel("Cost of Living Index", fontdict = {'fontname':'Noto Mono','fontsize':10})
plt.ylabel("State", fontdict = {'fontname':'Noto Mono','fontsize':10})

# Add a vertical line, here we set the style in the function call
ax.axvline(group_mean, ls='--', color='b')

plt.savefig('All_states_COL.png', dpi=300)

plt.show()
```

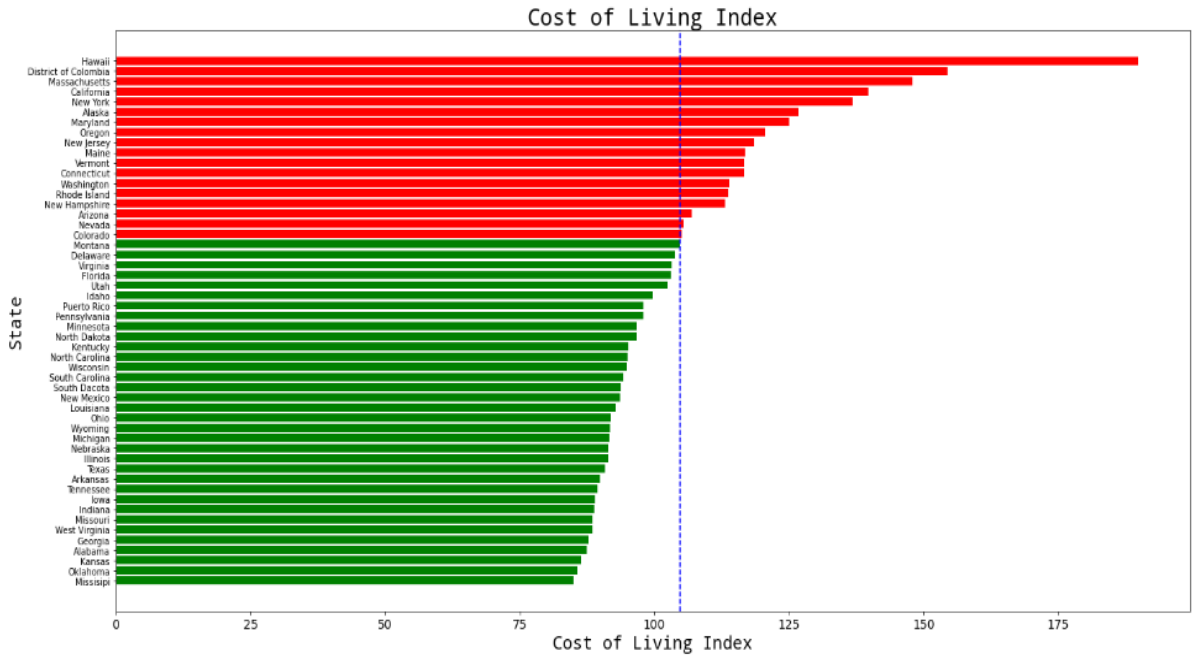

Figure 12: Representation of index of cost of living in USA in a horizontal bar chart. Bars are in green where the index is lower than the average cost of living, and red color when index is higher.

To better check which countries have index of cost of living higher than the average, we built a new dictionary with only countries whose indexes are higher than average.
Create again 2 lists one for the names and one for indexes of countries with expensive cost of living and build a bar chart.

```
fig, ax = plt.subplots(figsize=(20,10)) # plt.subplots() is a function that returns a tuple
                        # containing a figure and axes object(s). Thus when using fig, ax = plt.subplots()
                        # you unpack this tuple into the variables fig and ax.
ax.bar(expensive_Living_Cost_Country_Name, expensive_Living_Cost_Index)

labels = ax.get_xticklabels() # To gain access on the x-axis
plt.setp(labels, rotation=90, horizontalalignment='right',
            fontsize = 14) # set the property of many items at once

labels = ax.get_yticklabels() # To gain access on the y-axis
plt.setp(labels, fontsize = 14) # set the property of many items at once


# add a title to our graph
plt.title("States with Index Higher than Average", fontdict = {'fontname':'Noto Mono','fontsize':25})

# name our x and y axis.
plt.xlabel("Cost of Living Index", fontdict = {'fontname':'Noto Mono','fontsize':20})
plt.ylabel("State", fontdict = {'fontname':'Noto Mono','fontsize':20})

plt.savefig('Expesive_states_COL.png', dpi=300)

plt.show()
```

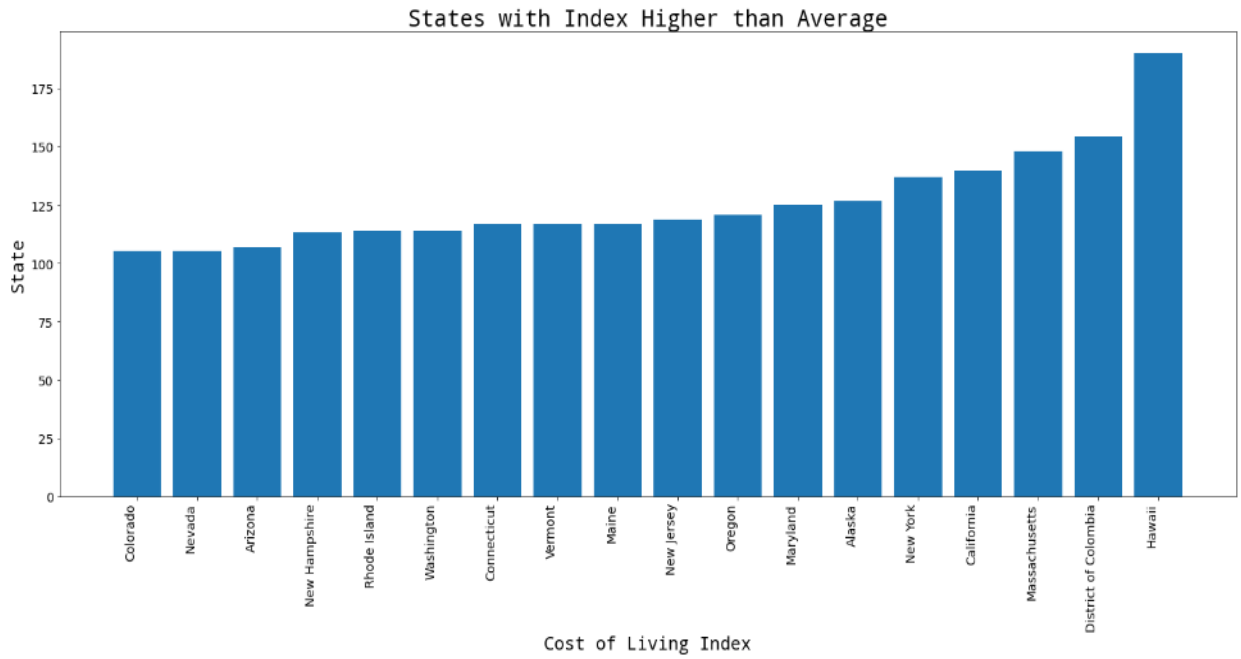

Figure 13: Representation of expensive countries in a bar chart

Gas Price

Our second example is about gas prices over years in some countries. The gas price is in US Dollars and the years range between 1990 and 2008. We took the dataset from https://www.kaggle.com/datasets/anuragshakya2005/gas-prices.
We read the dataset using pandas and choose to plot the gas prices for United States, Canada, South Korea and Australia, so the graph would not look overwhelmed. X-axis has year data increasing by 2, and y-axis has price per gallon in US Dollars.

```
plt.figure(figsize=(8,5))

plt.title('Gas Prices over Time(in USD)', fontdict = {'fontweight': 'bold', 'fontsize': 18})

plt.plot(gas.Year, gas.USA,'b.-',label='United States') # Choose column year from our data frame and USA
plt.plot(gas['Year'], gas.Canada,'r.-', label='Canada') # There is another way of selecting the columns we want
                                                        # from the data frame

plt.plot(gas.Year, gas['South Korea'], 'g.-',label='South Korea') # Used mostly when there are colum names with
                                                                  # more than 1 word and no space.
plt.plot(gas.Year, gas['Australia'], 'y.-',label='Australia')

# print(gas.Year)
# print(gas.Year[::2])
gas.Year[:2]
plt.xticks(gas.Year[::2]) # print every 2 years in x axis.

# Add labels to axis
plt.xlabel('Year')
plt.ylabel('US Dollars')

plt.legend()

plt.savefig('Gas Prices.png', dpi=300) # dpi for higher resolution will make in this case 2400:1500 image

plt.show()
```

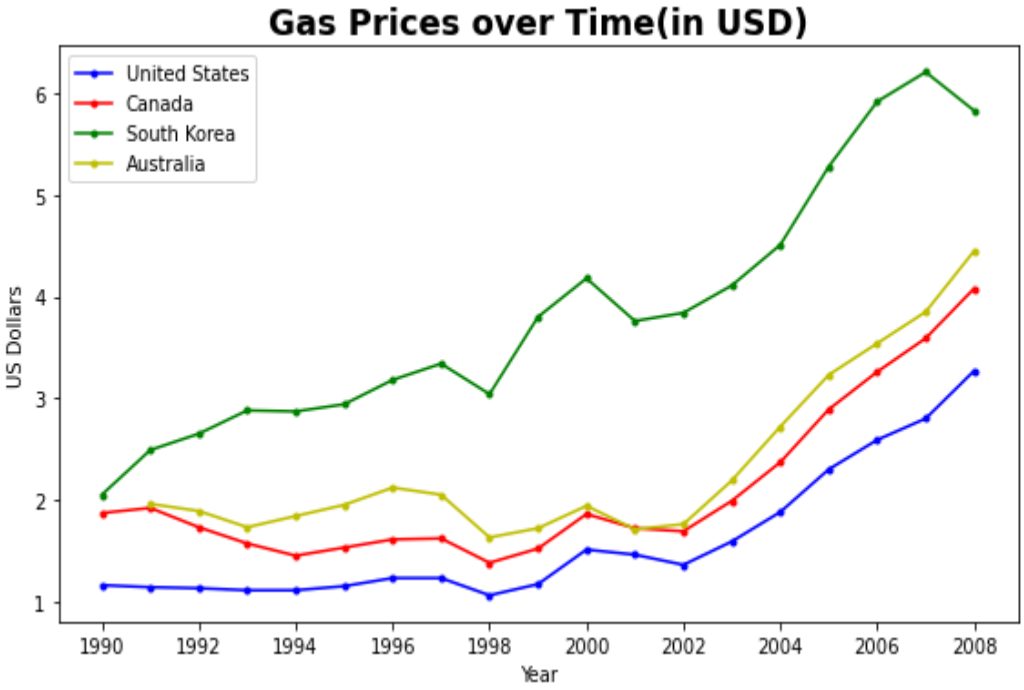

Figure 14: Gas prices over years in United States of America, Canada, Australia and South Korea

As we see from the graph, there is almost no difference in gas prices between US, Canada and Austria, but if we see South Korean line, seems that price has been twice as much expensive as in the other countries. Some reasons might be that South Korea implies high tax amounts on petrol and gas, and they have low gas reserves, so they have to import it.

Fifa

Our third and last example is a Fifa dataset. Data was taken from https://www.kaggle.com/code/sagnik1511/fifa-19-player-nationality/data. In this example we built a Histogram about the players overall rating and a pie chart about players preferred foot.
Our histogram has bins from 40 – 100 because there are no players with lower rankings.
To get the data from the histogram, we assigned it to arr. When we print arr, it gives

```
(array([  89., 2815., 9665., 5083.,  541.,   14.]), array([ 40,  50,  60,  70,  80,  90, 100]), <BarContainer obje
ct of 6 artists>)
[  89. 2815. 9665. 5083.  541.   14.]
```

The first element of the array shows the number players in each category. The second element shows the values of the x-axis or the bins, and the third element shows that there are 6 bar containers created.
To make the histogram more descriptive we have added the total number of players in each category at the top of each bar.

```
bins = [40,50,60,70,80,90,100] # set the bins from 40 - 100 since there are no players rating from 0 to 30

data_points = fifa.Overall.tolist() # get all the values from column Overall and assign to a list named
                                    # data_points

fig, ax = plt.subplots()

arr = ax.hist(data_points, ec='white' ,bins = bins) # create an object arr with 3 elements, data_points which
                                        # will have the numbers of each group,
                                        # values of bins and total number of bins.
                                        # ec - edgecolor is the border line between the bins.
print(arr)

labels = arr[0] #[arr[0][i] for i in range(len(arr[0]))]
                # take the first element from arr object and assign to labels
print(labels)

rects = arr[2] # assign the bins to rects

for rect, label in zip(rects, labels): # returns a zip object, which is an iterator of tuples
                                    # where the first item in each passed iterator is paired together, and so on
    height = rect.get_height() # gets the high of the bin in each tupple.
    ax.text(                                    # text position and type
        rect.get_x() + rect.get_width()/2,      # get the x cordinate where we want to insert the text
                                                # add middle of the width of bin
        height + 0.1,                           # height of text from the bin serves as y cordinate
        int(label),                             # to make an integer label
        ha='center',                            # horizontal alignment
        va='bottom'                             # vertical alignment
    )
# plt.hist(fifa.Overall, bins = bins)
# plt.xticks(bins)

plt.xlabel('Overall Rating')
plt.ylabel('Number of Players')

plt.title("Fifa Overall Rating")

plt.show()
```

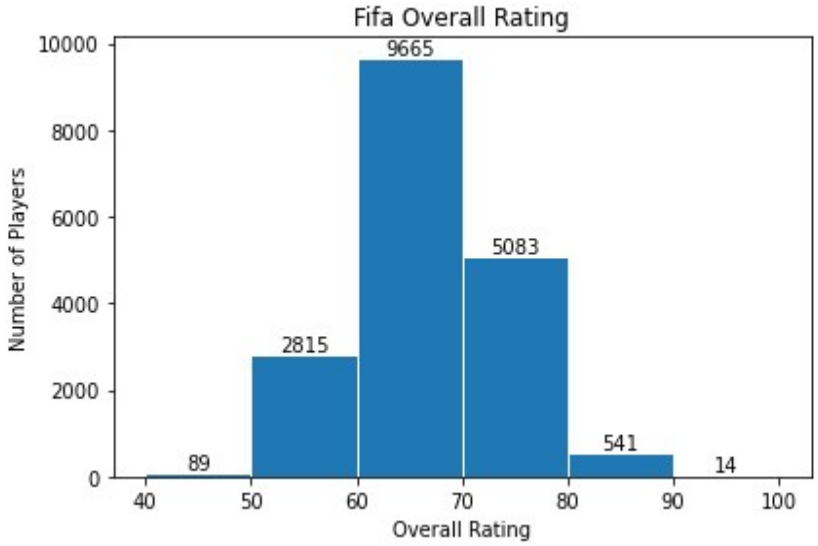

Figure 15: Soccer players overall ratings

In our next chart we find the percentage of players playing with right or left foot.
Firstly we find the total number of players playing with the left or right foot. We access the column "Preferred Foot" using .loc[]. Since a dataframe is returned, we just add count[0] to count the rows of one of the columns.

```
# Access a group of rows and columns by label(s) using .loc[]
left = fifa.loc[fifa['Preferred Foot'] == 'Left'].count()[0] # since it returns a dataframe, we use [0] to take the
                                                             # first element
                                                             # find number of players playing with the left foot
right = fifa.loc[fifa['Preferred Foot'] == 'Right'].count()[0] # find number of players playing with the right foot
```

Next we build the pie chart using the information found above.

```
labels = ['Left Foot', 'Right Foot']
colors = ['orange', 'blue']

plt.pie([left, right], labels = labels, colors = colors, autopct = '%.2f %%')# plot a pie chart with labels, colors
                                                                              # and percentage up to decimal numbers.

plt.title('Foot Preference')


plt.show()
```

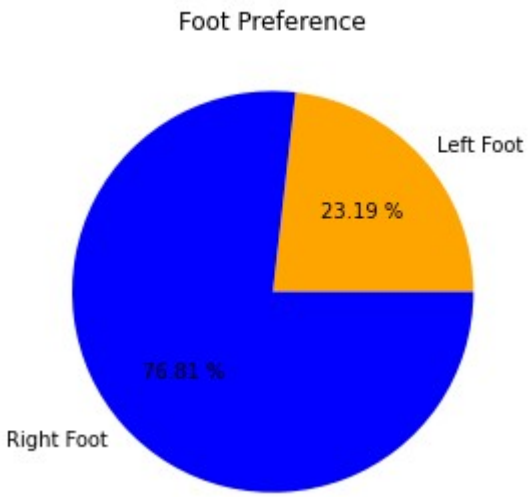

Figure 16: Preferred foot pie chart

Matplotlib Alternatives

There are numerous Python visualization libraries available today that we can use to illustrate our data. We have listed some Python libraries that may be matplolib alternatives below.

- Seaborn - uses matplotlib's power to generate beautiful charts in just a few lines of code. The main distinction is that Seaborn's default styles and color palettes are more aesthetically pleasing and modern. Because Seaborn is built on matplotlib, you'll need to know matplotlib to change the defaults.
- Plotly - Plotly's specialty is creating interactive plots, but it also includes charts that you won't find in most libraries, such as contour plots, dendrograms, and 3D charts.
- Geoplotlib - is a map-making and data-plotting toolkit. It can be used to generate a variety of map types, such as choropleths, heatmaps, and dot density maps.
- Gleam - allows you to convert analyses into interactive web apps using only Python scripts, eliminating the need to know other languages such as HTML, CSS, or JavaScript. Gleam is compatible with all Python data visualization libraries.
- Missingno - Dealing with missing data is inconvenient. Instead of trudging through a table, missingno lets you quickly assess the completeness of a dataset with a visual summary. You can use a heatmap or a dendrogram to filter and sort data based on completion or to identify correlations.

- Leather - Leather is a Python charting library for people who need charts right away and don't care if they're perfect. It's intended to work with any data type and generates charts as SVGs, allowing you to scale them without losing image quality. Because this library is still in its early stages, some documentation is still in the works.
- Altair - Altair, like Seaborn, is a declarative visualization library that allows you to create aesthetically pleasing graphs and charts; however, unlike Seaborn, Atair is based on Vega and Vega-Lite.
- Folium - enables the visualization of geospatial data You can create interactive maps such as choropleth maps, scatter maps, bubble maps, heatmaps, and so on. Folium's various plugins, such as Markercluser, ScrollZoomToggler, and DualMap, allow you to wrap leaflet maps and extend their functionality.