Assignment-1

Focus: Understanding the Basic Data Visualizations

Exercises: Matplotlib , Chart types , Basic Image data Visualizations

Matplotlib is the most widely used library in python for plotting of the commercial, scientific, public data . It comes with an interactive environment across platforms. It can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and GUI toolkits. With this library, with just a few lines of code, one can generate plots, bar charts, histograms, power spectra, stemplots, scatterplots, error charts, pie charts, and many other types.

- **Line Plots:** it is a graph that displays the occurrence of data along a number line.
- **Bar charts:** it is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent.
- **Histograms:** it is a graphical presentation of data using bars of different heights. It is similar to a Bar Chart, but histogram groups numbers into ranges. The height of each bar shows how many fall into each range.
- **Stemplots:** it is a special table where each data value is split into a "stem" (the first digit or digits) and a "leaf" (usually the last digit).
- **Scatterplots:** it is a type of data display that is the illustrations of the relationship between two numerical variables. Each member of the dataset gets plotted as a point whose x-y coordinates relate to its values for the two variables.
- **Error charts:** it is used to exhibit the extent of uncertainty in information relative to an average value.
- **Pie charts:** it is a circular statistical graphic, which is separated into slices to elucidate numerical proportion.

Matplotlib Architecture:

Matplotlib has three main layers: the backend layer, the artist layer, and the scripting layer.

- **Backend layer:** it has three interface classes:
  - figure canvas ➔ defines the area of the plot
  - renderer ➔ knows how to draw on figure canvas
  - event ➔ handles the user inputs such as clicks
- **Artist layer:** this layer knows how to use the Renderer and draw on the canvas. Everything on a Matplotlib plot is an instance of an artist layer. The ticks, title, labels the plot itself everything is an individual artist.

Scripting layer: is a lighter interface and very useful for everyday purposes

Data Preparation

- It is a communal task before any data visualization or data analysis project. As data never comes in the way data is needed. Here are some of the some of the steps to prepare "Canada.xlsx" (contains Canadian Immigration information, available in assignment-1) to observe some of the analytical information and trends.

- Keep the data 'Canada.xlsx' in the local work directory.

- Dataset is consisting of Immigration to Canada from 1980 to 2013.
    - International migration flows to and from particular countries according to the United Nations website.
    - The dataset comprises annual data on the flows of international migrants as recorded by the countries of destination (in this case it is Canada).
    - The data presents both inflows and outflows according to the place of birth, citizenship, or place of previous/next residence both for foreigners and nationals.

    - We are going to skip the first 20 rows and last 2 rows because it doesn't contain tabulated data.

Step-1: Skip first 20 rows from the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
df = pd.read_excel( 'Canada.xlsx' ,
                    sheet_name = 'Canada by Citizenship' ,
                    skiprows = range(20),
                    skipfooter =  2 )
print(df.head())
```

Output:

	Type	Coverage	OdName	AREA	...	2010	2011	2012	2013
0	Immigrants	Foreigners	Afghanistan	935	...	1758	2203	2635	2004
1	Immigrants	Foreigners	Albania	908	...	561	539	620	603
2	Immigrants	Foreigners	Algeria	903	...	4752	4325	3774	4331
3	Immigrants	Foreigners	American Samoa	909	...	0	0	0	0
4	Immigrants	Foreigners	Andorra	908	...	0	0	1	1

Step-2: Check the columns of the dataset to get an idea about the dataset

```
# see the column names to get the idea about the dataset
print(df.columns)
```

Output:

```
[5 rows x 43 columns]
Index([    'Type', 'Coverage',   'OdName',     'AREA', 'AreaName',      'REG',
       'RegName',      'DEV',  'DevName',       1980,       1981,       1982,
            1983,       1984,       1985,       1986,       1987,       1988,
            1989,       1990,       1991,       1992,       1993,       1994,
            1995,       1996,       1997,       1998,       1999,       2000,
            2001,       2002,       2003,       2004,       2005,       2006,
            2007,       2008,       2009,       2010,       2011,       2012,
            2013],
      dtype='object')
```

Step-3: We are going to get rid of some columns that we are not using to make the dataset smaller and more manageable.

```
# get rid of some columns that we are not using to make the dataset smaller and more manageable
df.drop(['AREA', 'REG', 'DEV', 'Type', 'Coverage'], axis=1, inplace=True)
print(df.head())
```

Output:

```
           OdName AreaName          RegName  ...  2011  2012  2013
0     Afghanistan     Asia    Southern Asia  ...  2203  2635  2004
1         Albania   Europe  Southern Europe  ...   539   620   603
2         Algeria   Africa  Northern Africa  ...  4325  3774  4331
3  American Samoa  Oceania        Polynesia  ...     0     0     0
4         Andorra   Europe  Southern Europe  ...     0     1     1
```

Step-4: We are going to change the column names to something more understandable

```
# Change the column names to something more understandable
df.rename(columns={'OdName': 'Country', 'AreaName': 'Continent', 'RegName': 'Region'}, inplace=True)
print(df.columns)
```

Step-5: Then add a 'total' column which will show the total immigrants that came into Canada from 1980 to 2013 from each country

```
# add a 'total' column which will show the total immigrants that came into Canada from 1980 to 2013 from each country
df['Total'] = df.sum(axis=1)
print(df.head())
```

Output:

```
          Country Continent          Region  ...  2012  2013  Total
0     Afghanistan      Asia    Southern Asia  ...  2635  2004  58639
1         Albania    Europe  Southern Europe  ...   620   603  15699
2         Algeria    Africa  Northern Africa  ...  3774  4331  69439
3  American Samoa   Oceania        Polynesia  ...     0     0      6
4         Andorra    Europe  Southern Europe  ...     1     1     15
```

Step-6 :Then check if there are any null values in any of the columns.

```
# Check if there are any null values in any of the columns
print(df.isnull().sum())
```

Step-7 :Then set the 'Country' column as the index

```
# Set the 'Country' column as the index
df = df.set_index('Country')
print(df.head)
```

Step-8: There are several styles of plots available from which we can choose accordingly.

```
print(plt.style.available)
mpl.style.use(['ggplot'])
```

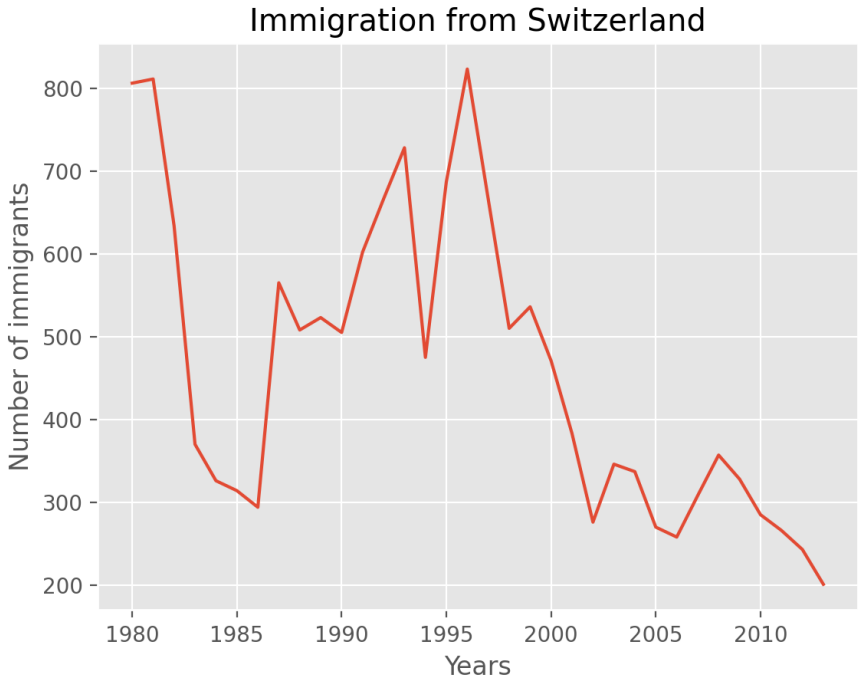
```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright',
```

Basic Data Visualization types

Line Plot:

1. It will be useful to see any country's immigration trend to Canada over the years. Here we take Switzerland as an example and plot the data as a line plot.

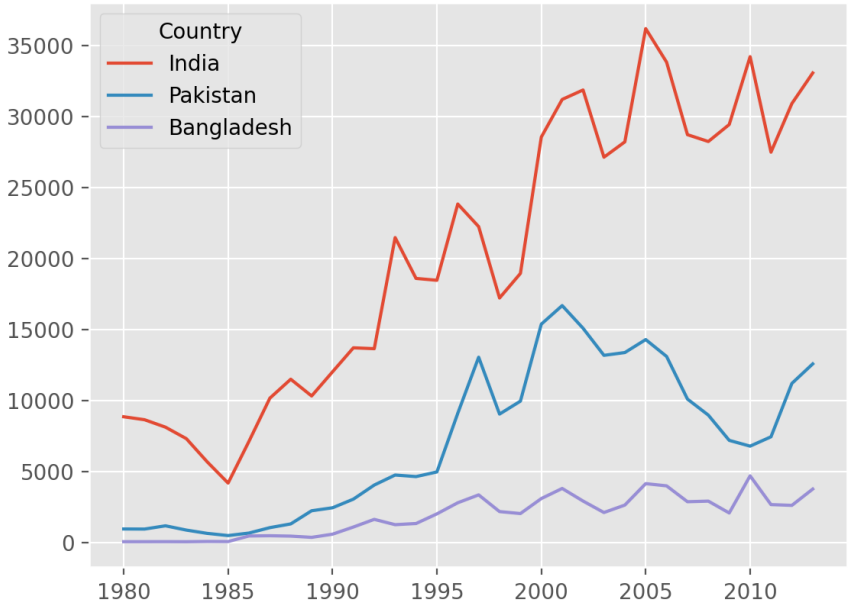
```
years = list(map(int, range(1980, 2014)))

# immigration data of Switzerland and the years.
df.loc['Switzerland', years]

df.loc['Switzerland', years].plot()
plt.title('Immigration from Switzerland')
plt.ylabel('Number of immigrants')
plt.xlabel('Years')
plt.show()
```



---

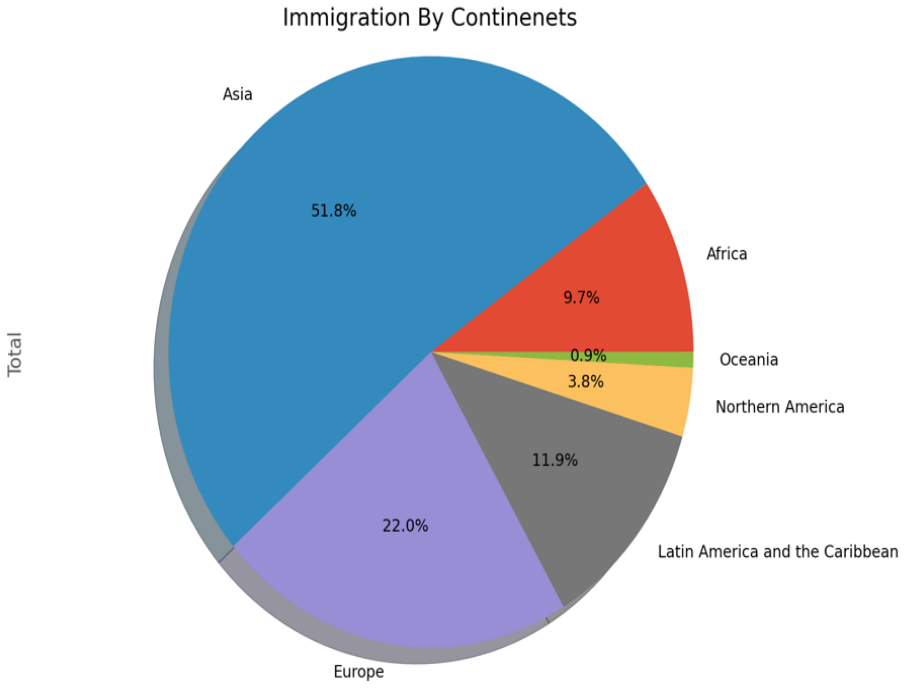
2. Another example is to observe the immigration trend over the years for several countries to compare those countries' immigration trends to Canada. Plot the number of immigrants of three subcontinent countries i.e. India, Pakistan, and Bangladesh vs the years.

```
ind_pak_ban = df.loc[['India', 'Pakistan', 'Bangladesh'], years]
ind_pak_ban.head()
ind_pak_ban.T

ind_pak_ban.T.plot()
plt.show()
```



3. Pie plot

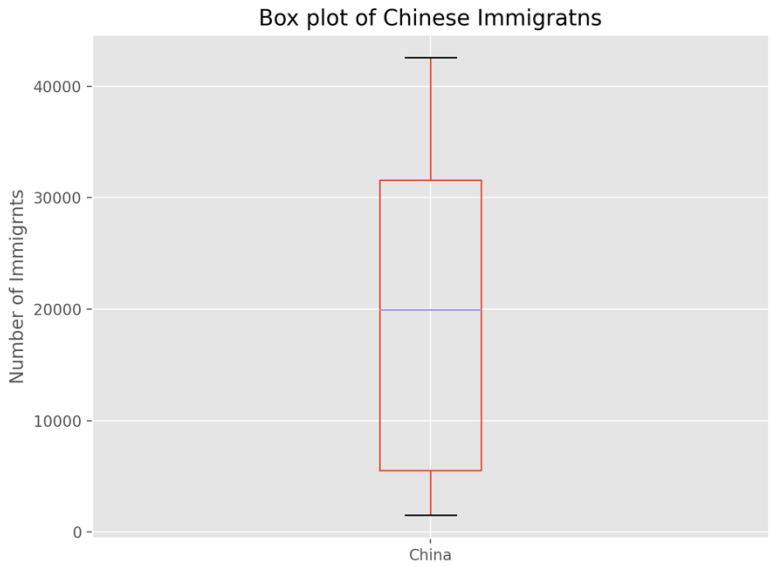
```
# group the number of immigrants, to sum up, the total number of immigrants for each continent
cont = df.groupby('Continent', axis=0).sum()

cont['Total'].plot(kind='pie', figsize=(7, 7),
                   autopct='%1.1f%%',
                   shadow=True)
plt.title('Immigration By Continents')
plt.axis('equal')
plt.show()
```



4. Box Plot

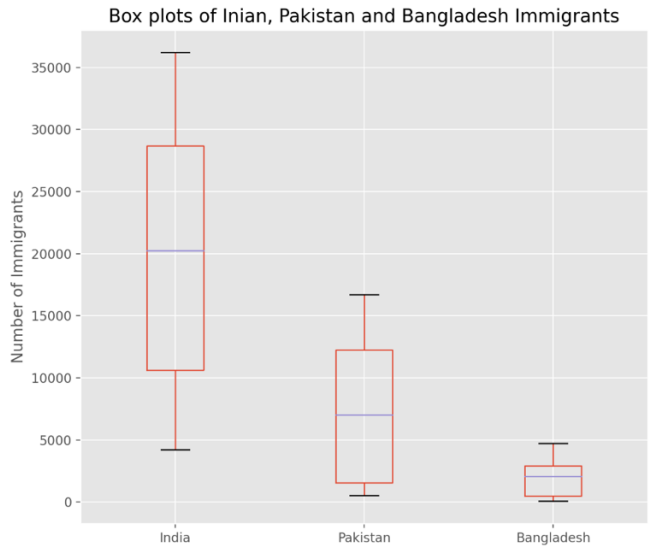
a. To demonstrate the box plot, we will plot the total number of immigrants of China to Canada over the years.

```
china = df.loc[['China'], years].T

china.plot(kind='box', figsize=(8, 6))
plt.title('Box plot of Chinese Immigrants')
plt.ylabel('Number of Immigrants')
plt.show()
```



b. We can also plot numerous boxplots in the same plot. Use the DataFrame 'ind_pak_ban' and make box plots of the number of immigrants of India, Pakistan, and Bangladesh.

```
# box plots of the number of immigrants of India, Pakistan, and Bangladesh
ind_pak_ban.T.plot(kind='box', figsize=(8, 7))
plt.title('Box plots of Indian, Pakistan and Bangladesh Immigrants')
plt.ylabel('Number of Immigrants')
plt.show()
```



4.Scatter Plot

A Scatter plot is the finest to understand the correlation amid variables. Therefore, we will plot a scatter plot to see the trend of the number of immigrants to Canada over the years in challenging questions i.e. Q4. In order to accomplish that, we will make a new Data Frame of the years as an index and the number of total immigrants each year. Afterward, we will convert the years into integers to make the Data Frame more presentable (Note that we will use this DataFrame to produce a scatter plot in the challenging questions section).

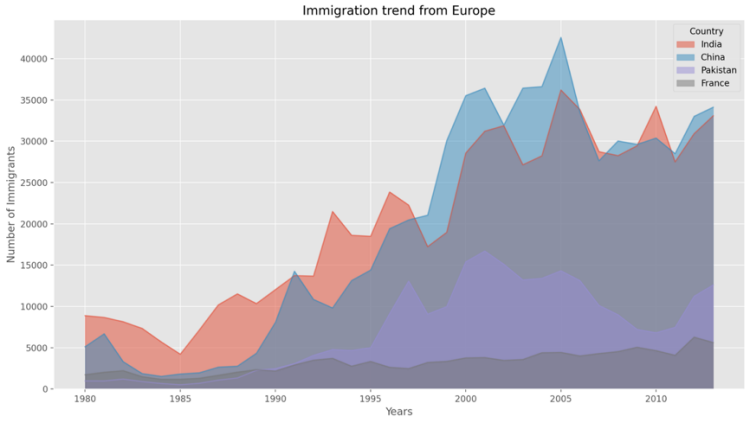
```
# make a new DataFrame that will contain the years as an index and the total number of immigrants each year
totalPerYear = pd.DataFrame(df[years].sum(axis=0))
totalPerYear.head()
print(totalPerYear.head())

# need to convert the years to integers and polish the DataFrame to make it presentable
totalPerYear.index = map(int, totalPerYear.index)
totalPerYear.reset_index(inplace=True)
totalPerYear.head()
print(totalPerYear.head())
```

Output: _____ (Try It yourself)

5. Area Plot

The area plot shows the area covered under a line plot. For this plot, we are going to make DataFrame including the information of India, China, Pakistan, and France.

```
# make DataFrame including the information of India, China, Pakistan, and France.
top = df.loc[['India', 'China', 'Pakistan', 'France'], years]
top = top.T
print(top)

colors = ['Black', 'Green', 'Blue', 'Red']
top.plot(kind='area', stacked=False, figsize=(20, 10))
plt.title('Immigration trend from Europe')
plt.ylabel('Number of Immigrants')
plt.xlabel('Years')
plt.show()
```

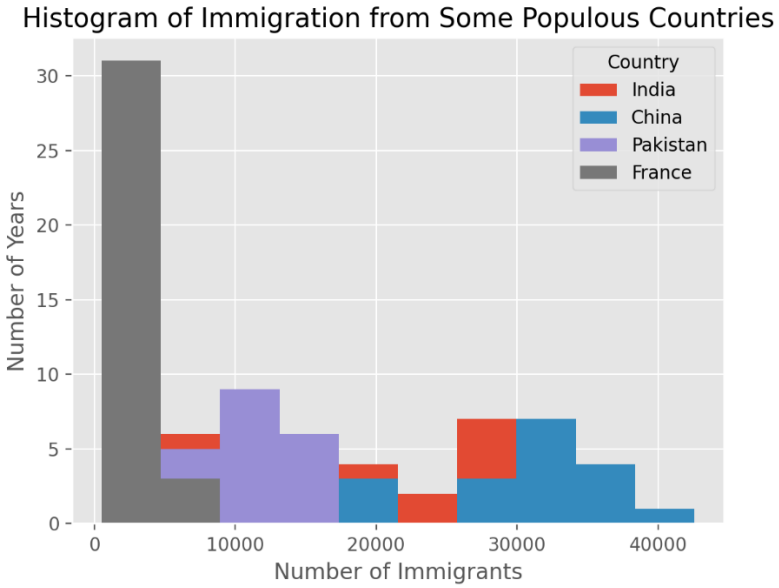


6. Histogram

The histogram shows the distribution of a variable. For this, we are going to use the 'top' Data Frame from the scatter plot example and plot each country's distribution of the number of immigrants in the same plot.

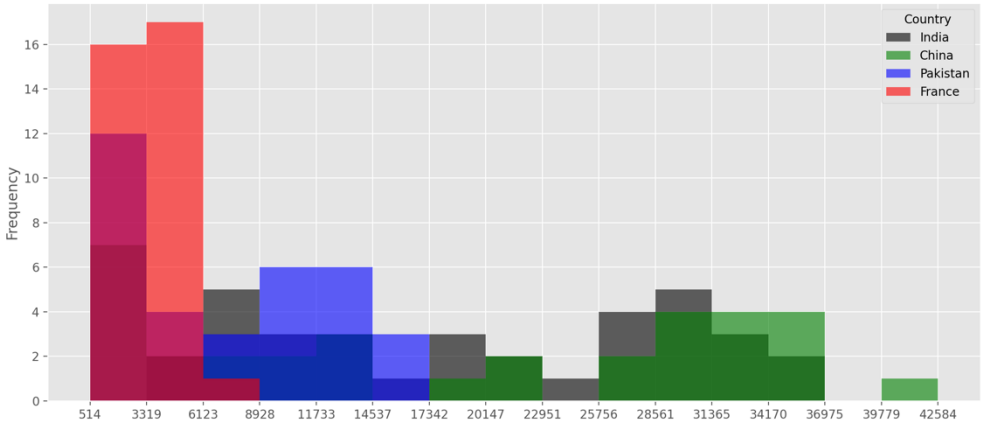
```
top.plot.hist()
plt.title('Histogram of Immigration from Some Populous Countries')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')
```

In this histogram, we can see that Canada had 20000 and 40000 immigrants from a few countries. Looks like China and India are amongst those few countries.



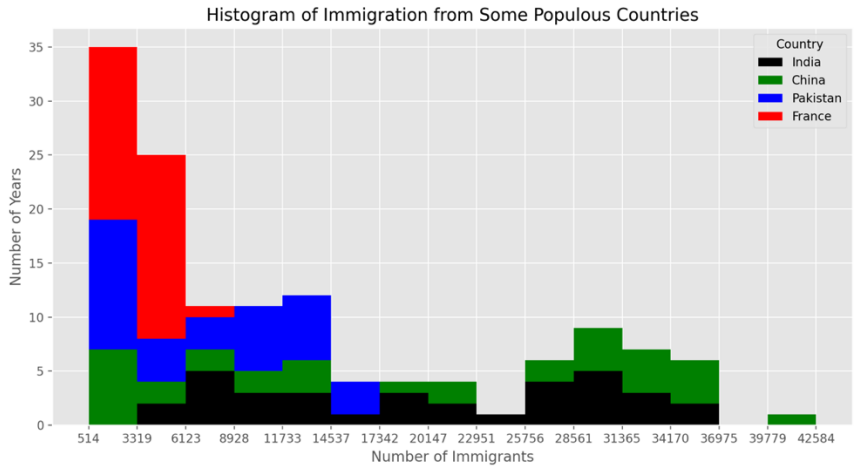
- In the above plot, we do not see the bin edges clearly. Let's improve this plot by specifying the number of bins as 15 and introduce a new parameter here called 'alpha' that controls the transparency of the colors. For such types of overlapping plots, transparency is important to see the shape of each distribution.

```
# Specify the number of bins and find out the bin edges
count, bin_edges = np.histogram(top, 15)
top.plot(kind='hist', figsize=(14, 6), bins=15, alpha=0.6,
         xticks=bin_edges, color=colors)
plt.show()
```



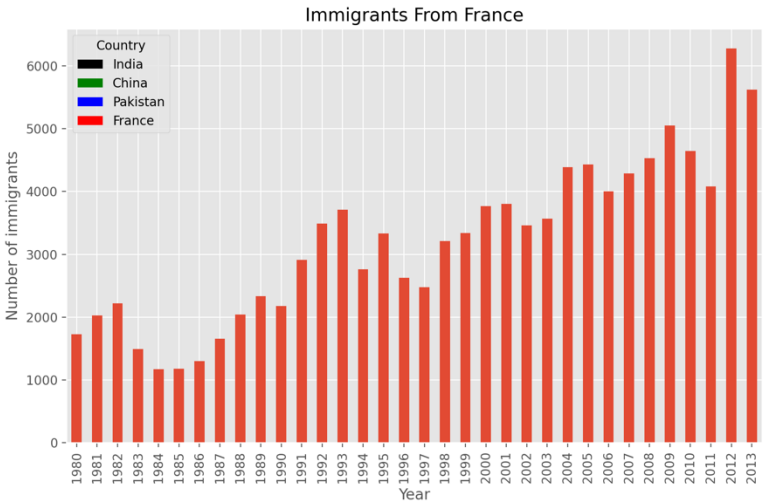
-Similar to the area plot, you can make a stacked plot of the histogram as well by setting the "stacked" feature as true.

```
top.plot(kind='hist',
         figsize=(12, 6),
         bins=15,
         xticks=bin_edges,
         color=colors,
         stacked=True,
         )
plt.title('Histogram of Immigration from Some Populous Countries')
plt.ylabel('Number of Years')
plt.xlabel('Number of Immigrants')
plt.show()
```



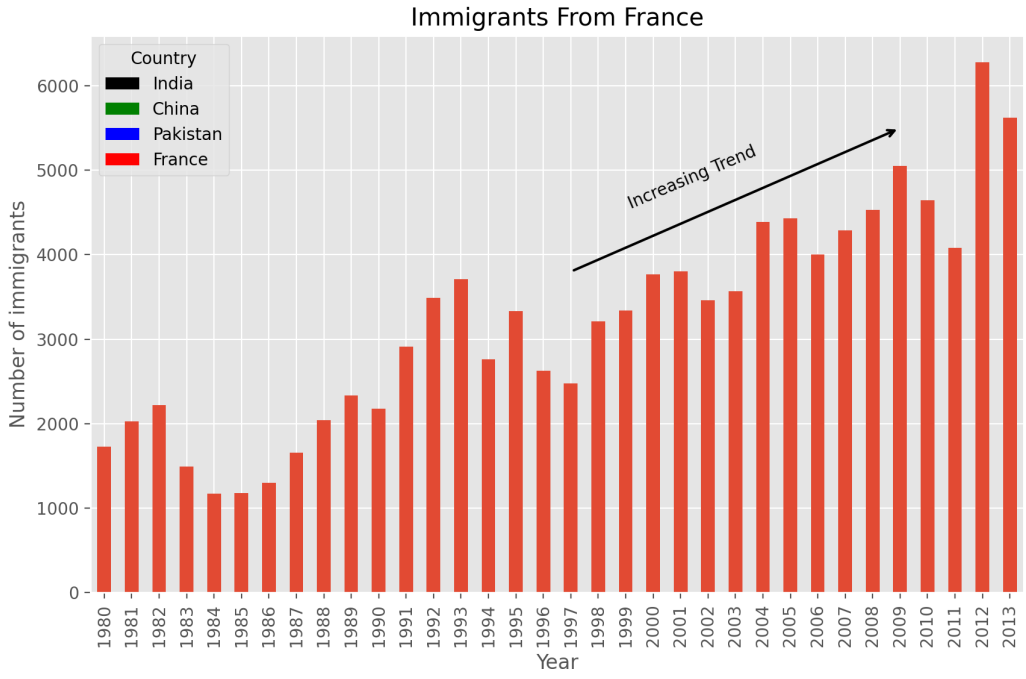
7. Bar Plot:

```
# bar plot of the number of immigrants from France per year
france = df.loc['France', years]
france.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Year')
plt.ylabel('Number of immigrants')
plt.title('Immigrants From France')
```



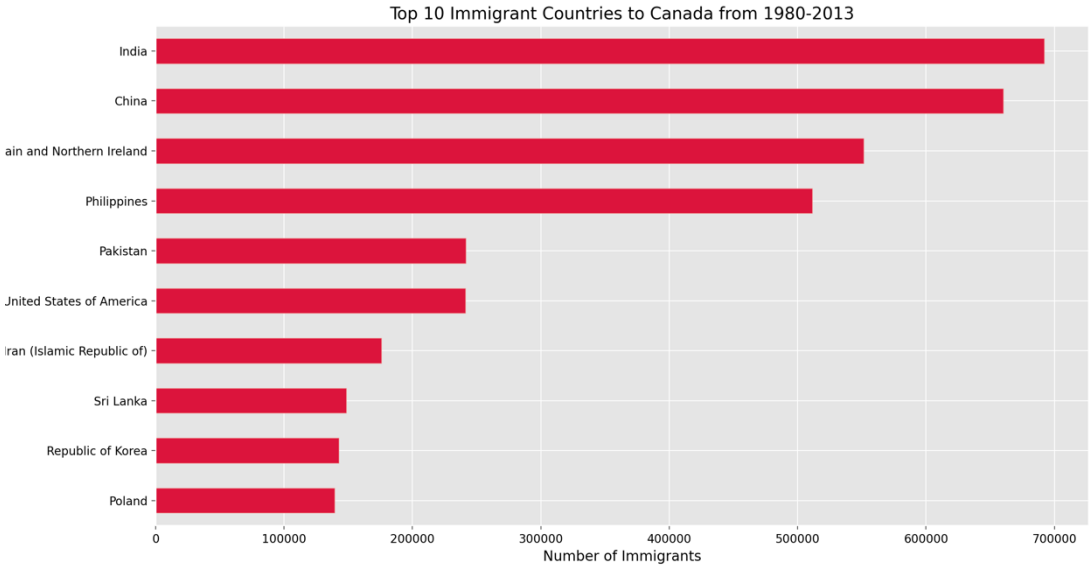
- We can add extra information to the bar plot. This plot shows an increasing trend since 1997 for over a decade. It could be worth mentioning. It can be done using an annotate function.

```
# increasing trend since 1997 for over a decade using an annotate function
france.plot(kind='bar', figsize=(10, 6))
plt.xlabel('Year')
plt.ylabel('Number of immigrants')
plt.title('Immigrants From France')
plt.annotate('Increasing Trend',
             xy=(19, 4500),
             rotation=23,
             va='bottom',
             ha='left')
plt.annotate('',
             xy=(29, 5500),
             xytext=(17, 3800),
             xycoords='data',
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='black', lw=1.5))
```



-Sometimes, showing the bars horizontally makes it more understandable. Showing a label on the bars can be even better.

```
df_top10 = pd.DataFrame(df.nlargest(10, 'Total')['Total'].sort_values(ascending=True))

df_top10.plot.barh(legend=False, color='crimson', edgecolor='LightCoral')
plt.title('Top 10 Immigrant Countries to Canada from 1980-2013', color='black')
plt.xlabel('Number of Immigrants', color='black')
plt.ylabel('Country', color='black')
plt.xticks(color='black')
plt.yticks(color='black')
```



--

Visualizing Image Data:

Mesh grid and RGB:

First, we are going to create a Gaussian disc's array with values somewhere in the range of 0 and 8 with 10 rows and 15 columns.

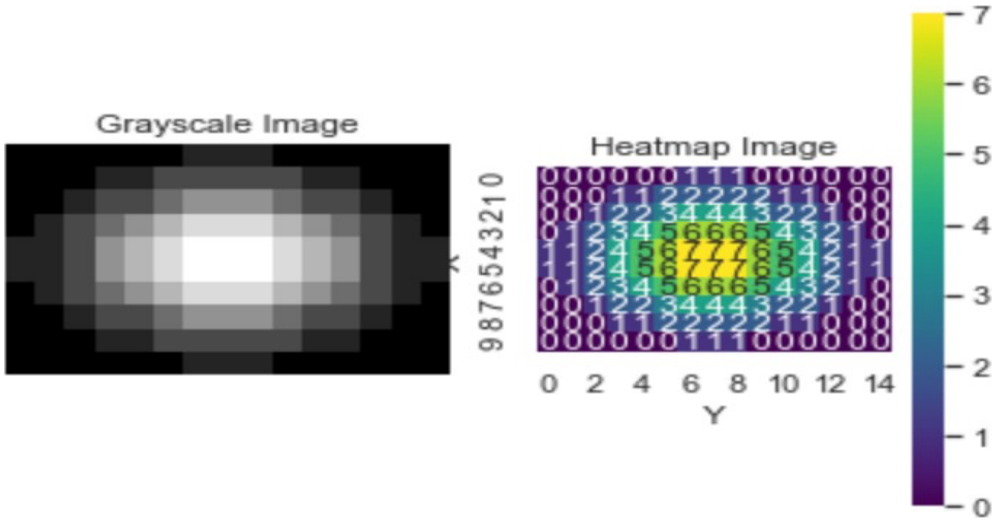
1.

```
import numpy as np

rows = 10
columns = 15
x, y = np.meshgrid(np.linspace(-1,1,columns), np.linspace(-1,1,rows))
d = np.sqrt(x*x+y*y)
sigma = 0.5
myGaussianDisc = (8*np.exp(-( (d)**2 / ( 2.0 * sigma**2 ) ) )).astype('uint8')

display(myGaussianDisc)
```

```
array([[0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 2, 2, 2, 2, 1, 1, 0, 0, 0, 0],
       [0, 0, 1, 2, 2, 3, 4, 4, 4, 3, 2, 2, 1, 0, 0],
       [0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0],
       [1, 1, 2, 4, 5, 6, 7, 7, 7, 6, 5, 4, 2, 1, 1],
       [1, 1, 2, 4, 5, 6, 7, 7, 7, 6, 5, 4, 2, 1, 1],
       [0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0],
       [0, 0, 1, 2, 2, 3, 4, 4, 4, 3, 2, 2, 1, 0, 0],
       [0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]], dtype=uint8)
```

2a.

```
f,axes = plt.subplots(1,2)
(aGrayscale, aHeatmap) = axes.flatten()


aGrayscale.imshow(myGaussianDisc, cmap="gray")
aGrayscale.set_axis_off()
aGrayscale.set_title("Grayscale Image")


sns.heatmap(myGaussianDisc,cmap="viridis",square=True,annot=True,ax=aHeatmap)
aHeatmap.set_ylim(0,10)
aHeatmap.invert_yaxis()
aHeatmap.set_ylabel('X')
aHeatmap.set_xlabel('Y')
aHeatmap.set_title("Heatmap Image")

plt.show()
```



2b. For color images, we have to create a 3-channel Gaussian disc's array with values somewhere in the range of 0 and 8 with 10 rows and 15 columns.

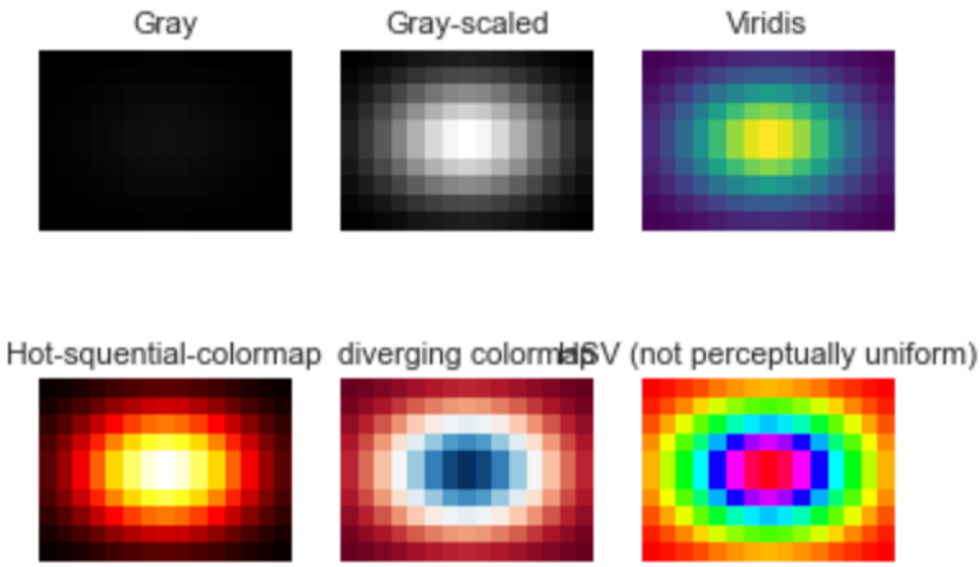
```
rows = 10
columns = 15
x, y = np.meshgrid(np.linspace(-1,1,columns), np.linspace(-1,1,rows))
d = np.sqrt(x*x+y*y)
sigma = 0.5
disc = (8*np.exp(-( (d)**2 / ( 2.0 * sigma**2 ) ))).astype('uint')
myRGBColorArray = np.stack([disc,np.roll(disc,2,axis=0),np.roll(disc,2,axis=1)],axis=2)

print("Red:")
display(myRGBColorArray[:,:,1])
```

```
array([[0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 1, 1, 0, 0, 0],
       [0, 0, 1, 2, 2, 3, 4, 4, 4, 3, 2, 2, 1, 0, 0],
       [0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0],
       [1, 1, 2, 4, 5, 6, 7, 7, 7, 6, 5, 4, 2, 1, 1],
       [1, 1, 2, 4, 5, 6, 7, 7, 7, 6, 5, 4, 2, 1, 1],
       [0, 1, 2, 3, 4, 5, 6, 6, 6, 5, 4, 3, 2, 1, 0],
       [0, 0, 1, 2, 2, 3, 4, 4, 4, 3, 2, 2, 1, 0, 0]], dtype=uint64)
```

3. Create gray, gray-scaled, Viridis, Hot, diverging colormap, and HSV not perceptually uniform plot accordingly.

```
rows = 10
columns = 15
x, y = np.meshgrid(np.linspace(-1,1,columns), np.linspace(-1,1,rows))
d = np.sqrt(x*x+y*y)
sigma = 0.5
my12BitArray = ((2**12-1)*np.exp(-( (d)**2 / ( 2.0 * sigma**2 ) ) )).astype('uint16')

f, axes = plt.subplots(2,3)
(aG, aS, aV, aH, aRB, aHSV) = axes.flatten()

aG.imshow(my12BitArray, cmap="gray", vmin=0, vmax=(2**16)-1)
aG.set_axis_off()
aG.set_title("Gray")

aS.imshow(my12BitArray, cmap="gray")
aS.set_axis_off()
aS.set_title("Gray-scaled")

aV.imshow(my12BitArray, cmap="viridis")
aV.set_axis_off()
aV.set_title("Viridis")

aH.imshow(my12BitArray, cmap="hot")
aH.set_axis_off()
aH.set_title("Hot-squential-colormap")

aRB.imshow(my12BitArray, cmap="RdBu")
aRB.set_axis_off()
aRB.set_title("diverging colormap")

aHSV.imshow(my12BitArray, cmap="hsv")
aHSV.set_axis_off()
aHSV.set_title("HSV (not perceptually uniform)")

plt.show()
```

Output:



4. Reading and storing image data, we are starting it with the image '1.png'

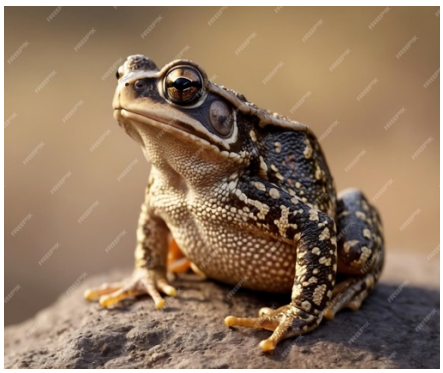
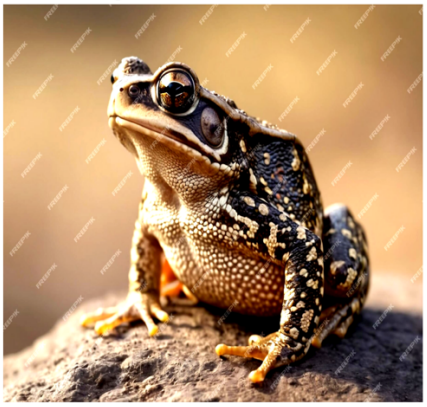


Fig: 1.png

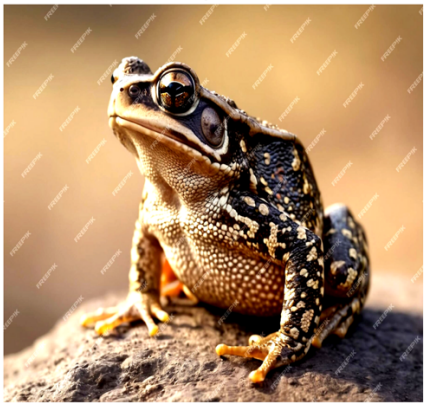
We are applying the basic image enhancement and filtering on '1.png' as

```python
from PIL import Image, ImageEnhance, ImageFilter, ImageOps
import matplotlib.pyplot as plt

# Load the image
img = Image.open("/mnt/data/1.jpg")

# 1. Brightness and contrast adjustment
enhancer_brightness = ImageEnhance.Brightness(img)
bright_img = enhancer_brightness.enhance(1.2)  # increase brightness by 20%

enhancer_contrast = ImageEnhance.Contrast(bright_img)
contrast_img = enhancer_contrast.enhance(1.3)  # increase contrast by 30%

# 2. Sharpening
sharpened_img = contrast_img.filter(ImageFilter.SHARPEN)

# 3. Blur (Gaussian blur for softening)
blurred_img = sharpened_img.filter(ImageFilter.GaussianBlur(radius=1))

# 4. Grayscale conversion
gray_img = ImageOps.grayscale(blurred_img)

# 5. Edge detection
edges_img = gray_img.filter(ImageFilter.FIND_EDGES)

# 6. Resize (reduce image size by 50%)
resized_img = blurred_img.resize((img.width // 2, img.height // 2))

# Display all processed images in a grid
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
axes = axes.ravel()

titles = [
    "Brightness + Contrast", "Sharpened", "Blurred",
    "Grayscale", "Edge Detection", "Resized"
]
images = [contrast_img, sharpened_img, blurred_img, gray_img, edges_img, resized_img]

for ax, im, title in zip(axes, images, titles):
    ax.imshow(im, cmap='gray' if im.mode == 'L' else None)
    ax.set_title(title)
    ax.axis("off")

plt.tight_layout()
plt.show()
```

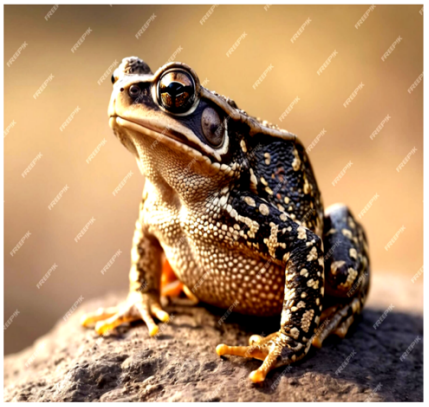
Brightness + Contrast



Sharpened



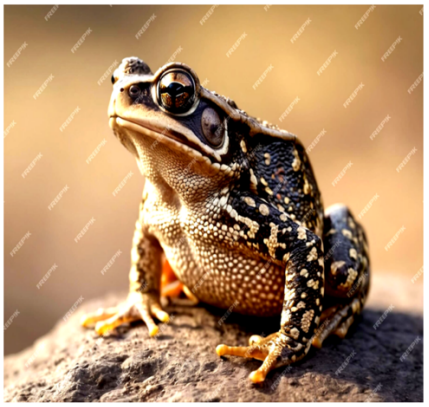
Blurred



Grayscale

Edge Detection

Resized



Assignment Questions:

Answer these questions through visualization (only):

Data : Use the '**wti-daily.csv**' and '**brent-daily.csv**' files into a data frame in which the Date column is treated as a datetime value and is set to be the index.

Q1. Plot the average price trend of oil from 1992-2002 from wti-daily.csv?

Q2. Plot the average price trend of oil from 1992 -2002

Q3. Compare the both the average prices of a barrel of oil from 1992-2002, indicate significant differences though markers.

Data: Use the ' **sales.xlsx** ' to answer the questions Q4 and Q5.

Q4. Plot the city wise distribution of sales, which city has contributed maximum in the sales.

Q5. Does payment methods have impact on sales, which payment method is contributing to the sales.

Data: Use ' **2.jpg** ' file to answer the questions Q6 and Q7.

Q6. Read the image '2.jpg' into a NumPy array, apply six different types of 'cmaps', and put these images six subplots.

Q7. Apply six basic filters on image '2.jpg'. Put these six images into two subplots of the following format as:

- total number of the rows are 3

- 3 subplots in the 1st row, one sublot in the 2nd row and two subplots in the 3rd row.

Useful links:

1. https://matplotlib.org/stable/users/explain/quick_start.html
2. https://seaborn.pydata.org/tutorial/introduction.html
3. https://github.com/datasets/oil-prices/tree/main/data
4. https://s3.amazonaws.com/tripdata/index.html
