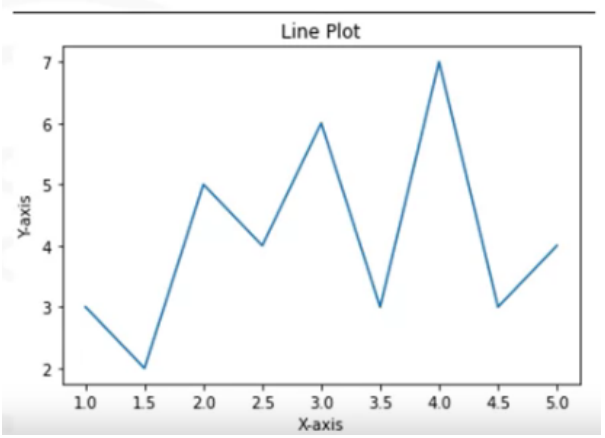
- MODULE #1: Introduction to Data Visualization Tools
- MODULE #2: Basic and Specialized Visualization Tools
- MODULE #3: Advanced Visualizations and Geospatial Data
- MODULE #4: Creating Dashboards with Plotly and Dash

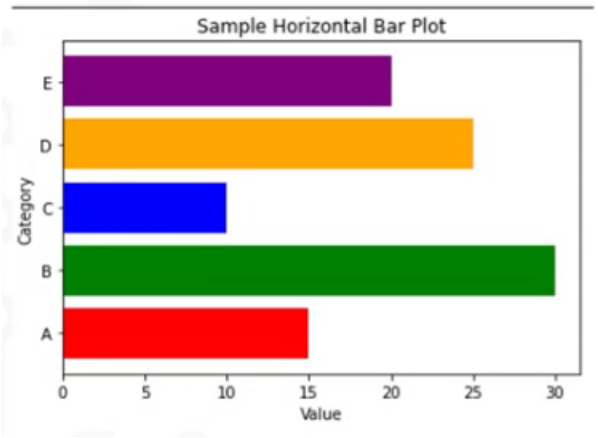
❖ MODULE #1: Introduction to Data Visualization Tools

Overview of Data Visualization

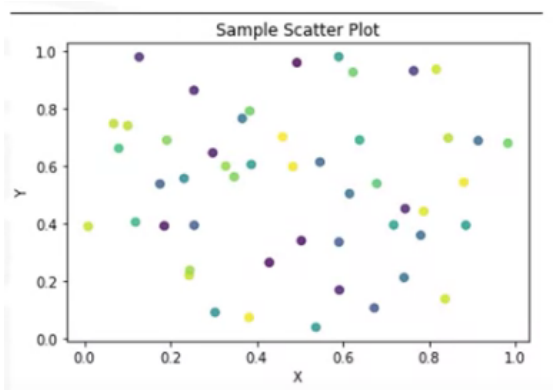
- What is Data Visualization?
    - Graphical representation of data and information
    - Can take a variety of forms
    - Basic charts and graphs
    - Interactive dashboards and maps
- Why build visuals?
    - Easily Understand Data
        - Highlight patterns, trends, and relationships
    - Communicate insights
        - Simplifies the data for stakeholders
    - Identify trends and data patterns
        - Help gain new insights
    - Present data for better understanding and interpretation
- Leveraging the power of data visualization
    - Business and finance
    - Healthcare
    - Education
    - Government
    - Research and science
    - Entertainment
- Best practices
    - Data is accurately represented
    - Message is clearly communicated
    - Choose the right type of Visualization
        - Varied visualization techniques
    - Keep it simple
        - Use of line chart and bar chart
    - Use clear Labeling and Formatting
        - Including a clear title and legend
    - Keep visualization focused
        - Use necessary data and labels only
    - Consider the audience
        - Visualization should be tailored
- When creating a visual
    - Less is more effective
    - Less is more attractive
    - Less is more impactful

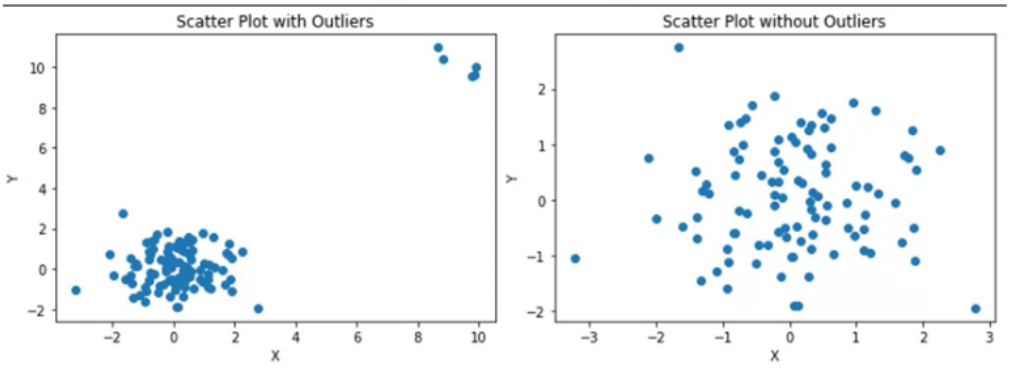
Types of plots

- Different types of plots
    - Line plot
    - Bar plot
    - Scatter plot
    - Box plot
    - Histogram
- Line plot
    - Displays data as a series of data points connected by straight lines
    - 

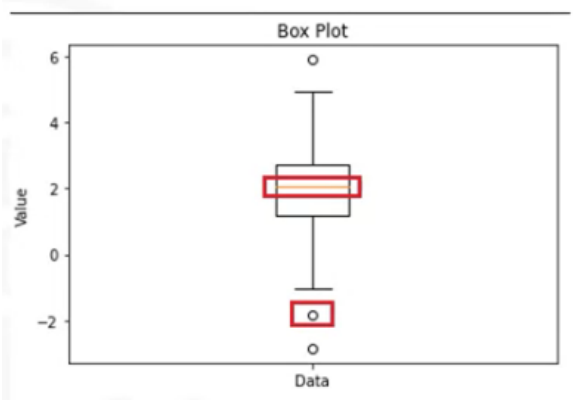
    - Use case
        - Display trends over time
        - Compare data sets with a continuous independent variable
        - Illustrate cause-and-effect relationships
        - Visualize continuous data
- Bar plot
    - Displays data using rectangular bars
    - The height or length of the bars represents the magnitude of the data
    - Displays bars either vertically or horizontally
    - 

    - Use cases
        - Compare different categories or groups
        - Display discrete data that has distinct categories
        - Show how different categories contribute to a whole

        - Visualize data that you can easily ranked or ordered
- Scatter plot
    - Uses cartesian coordinates to display values for two variables
    - Value of one variable determines the position on the horizontal axis, and the value of the other variable determines the position on the vertical axis
    - 

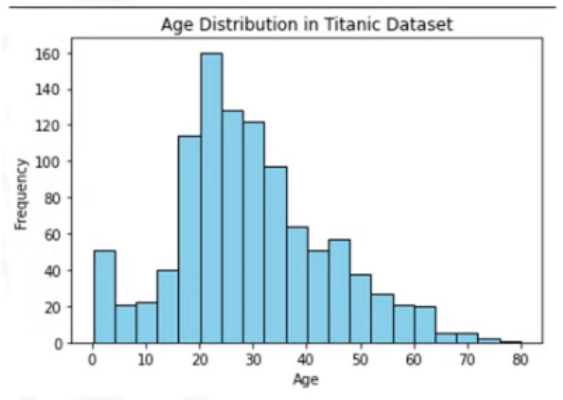
    - Use case
        - Examine the relationship between two continuous variables
        - Investigate patterns or trends in data
        - Detect outliers or unusual observations
        - Identify clusters or groups in the data
    - 

        - With outliers, it shows two clusters, but removing outliers makes the remaining data more visible. Proper outlier handling, enhances accuracy and meaningful insights in scatter plots.
- Box plot
    - Displays the distribution of a dataset along with key statistical measures
    - Represents the interquartile range as a box, a median line, and whiskers indicating range
    - Represents outliers as individual data points beyond whiskers

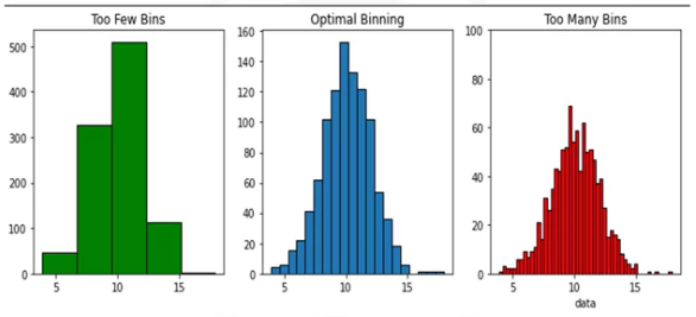


- Use case
  - Compare the distribution of a continuous variable across different categories or groups
  - Examine spread and skewness of a dataset, visualizing quartiles
  - Identify and analyze potential outliers
  - Visualize summary statistics
  - Compare distributions of multiple variables in datasets

- Histogram
  - Graphical representation of dataset distribution
  - Shows frequency or relative frequency within intervals
  - Bars represent the data count in each interval



  - Use case
    - Understand data distribution
    - Visually depict the shape of the data
    - Assess skewness in the data
    - Showcase data variability and spread



Plot Libraries

- Popular plot libraries in Python:
  - Matplotlib
  - Pandas
  - Seaborn
  - Folium
  - Plotly
  - PyWaffle
- Matplotlib
  - General purpose plotting library
  - Integrates well with libraries and frameworks
  - Features:
    - Creates a wide variety of plots
    - Customizes various elements
    - Empowers data visualization tasks
- Pandas
  - Users employ it for data manipulation
  - Its functions are built on Matplotlib
  - Features:
    - Integrates seamlessly with Pandas data structure
    - Analyzes exploratory data using visualization capabilities
- Seaborn
  - Great option for specialized statistical visualizations
  - It offers a variety of stylish plots
  - Features:
    - Offers various color palettes and styles
    - Offers functions to combine multiple plots in a grid layout
    - Integrates with Pandas
- Folium
  - Excellent option for geospatial data visualization
  - Builds interactive and customizable maps
  - Features:
    - Integrates with popular data analysis libraries
- Plotly
  - Has highly interactive plots and dashboards
  - Can create a variety of plots
  - Features:
    - Builds interactive dashboards
    - Enables plotting in a web browser
- PyWaffle
  - Visualizes categorical data using waffle charts
  - Features:
    - Provides a unique way to represent proportions

❖ **MODULE #2: Basic and Specialized Visualization Tools**

Area Plots

- Area plot
  - Also known as area chart or graph
  - Displays magnitude and proportion of multiple variables
  - Represents time or another ordered dimension
  - Is based on the line plot
- Generating area plots

```
df_canada.sort_values(['Total'], ascending = False, axis = 0, inplace = True)
```

	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Country																					
India	Asia	Southern Asia	Developing regions	8880	8670	8147	7338	5704	4211	7150	...	36210	33848	28742	28261	29456	34235	27509	30933	33087	691904
China	Asia	Eastern Asia	Developing regions	5123	6682	3308	1863	1527	1816	1960	...	42584	33518	27642	30037	29622	30391	28502	33024	34129	659962
United Kingdom of Great Britain and Northern Ireland	Europe	Northern Europe	Developed regions	22045	24796	20620	10015	10170	9564	9470	...	7258	7140	8216	8979	8876	8724	6204	6195	5827	551500
Philippines	Asia	South-Eastern Asia	Developing regions	6051	5921	5249	4562	3801	3150	4166	...	18139	18400	19837	24887	28573	38617	36765	34315	29544	511391
Pakistan	Asia	Southern Asia	Developing regions	978	972	1201	900	668	514	691	...	14314	13127	10124	8994	7217	6811	7468	11227	12603	241600

  - Inia China United Kingdom Philippines Pakistan

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

df_canada.sort_values(['Total'], ascending = False, axis = 0, inplace = True)

df_top5 = df_canada.head()
df_top5 = df_top5[years].transpose()
```

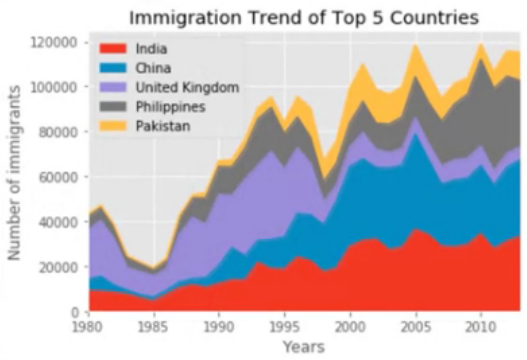
Country	India	China	United Kingdom of Great Britain and Northern Ireland	Philippines	Pakistan
1980	8880	5123	22045	6051	978
1981	8670	6682	24796	5921	972
1982	8147	3308	20620	5249	1201
1983	7338	1863	10015	4562	900
1984	5704	1527	10170	3801	668

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
df_top5.plot(kind='area')

plt.title('Immigration trend of top 5 countries')
plt.ylabel('Number of immigrants')
plt.xlabel('Years')

plt.show()
```



- Area plot: Uses

    - Tracking stock market performance
    - Visualizing population demographics
    - Displaying the distribution of resources

Histograms

- Histogram
    - Represents the frequency distribution of a dataset
        - Partitions numeric data into bins
        - Assigns each data point in the dataset
        - Counts the number of data points

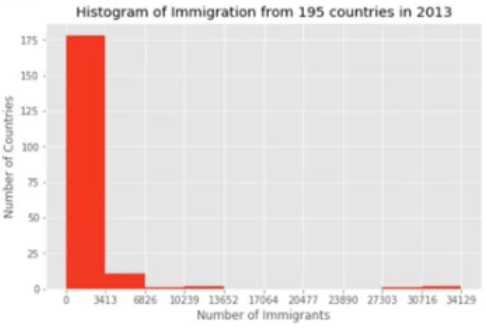
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

count, bin_edges = np.histogram(df_canada['2013'])

df_canada['2013'].plot(kind='hist', xticks = bin_edges)

plt.title('Histogram of Immigration from 195 countries in 2013')
plt.ylabel('Number of Countries')
plt.xlabel('Number of Immigrants')

plt.show()
```



Bar Charts

- Bar chart
    - Also known as a bar graph
    - It compares the values of a variable
- Generating a bar chart

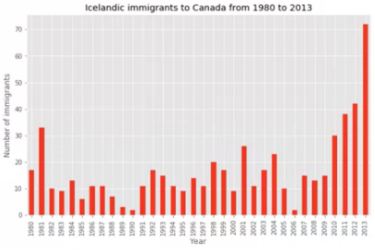
```
import matplotlib as mpl
import matplotlib.pyplot as plt

Years = list (map(str, range (1980, 2014)))
df_iceland = df_canada.loc[ 'Iceland',
years]

df_iceland.plot(kind='bar')

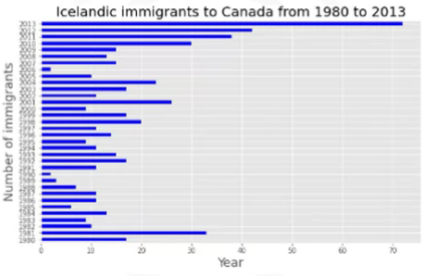
plt.title('Icelandic immigrants to
Canada from 1980 to 2013')
plt.xlabel('Year')
plt.ylabel('Number of immigrants')

plt.show()
```



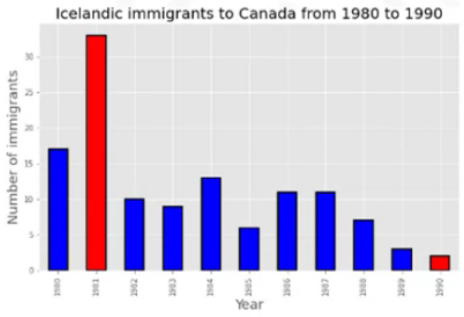
- Horizontal bar chart

```
df_iceland.plot(kind='barh', color ='red')
```



- Highlighted bar and edge color

```
c=['blue','red',"blue",'blue','blue','blue','blue','blue','blue','blue','red']
df_Iceland.plot(kind = 'bar', color = c, edgecolor = 'black')
```



Pie Charts

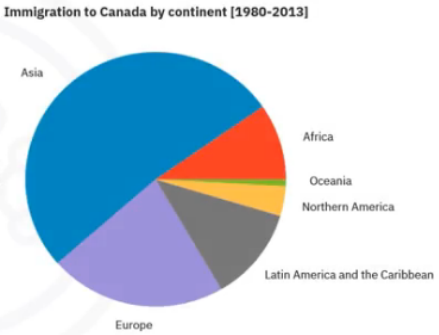
- Pie chart
  - A circular statistical graphic, divided into segments

```
import matplotlib as mpl
import matplotlib.pyplot as plt

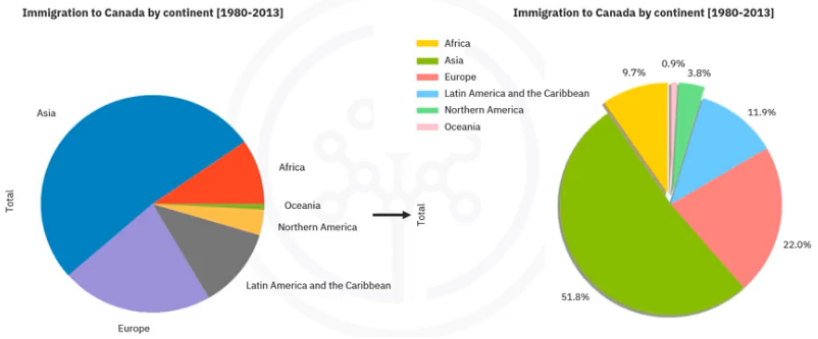
df_continents['Total'].plot(kind='pie')

plt.title('Immigration to Canada by Continent [1980-2013]')

plt.show()
```

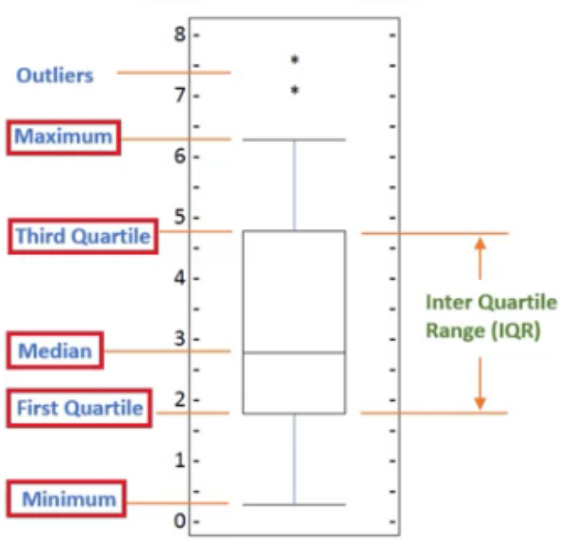


- Pie chart: Explode property



Box Plots

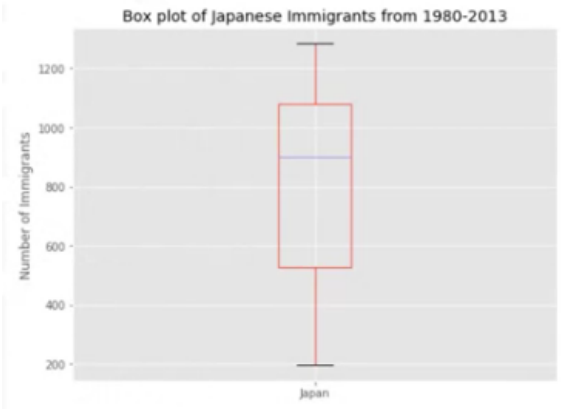
- Box plot

- 

- Generating a box plot
  - ```
    import matplotlib as mpl
    import matplotlib.pyplot as plt
    ```
  - ```
    df_japan = df_canada.loc[['Japan'], years].transpose()

    df_japan.plot(kind='box')

    plt.title('Box plot of Japanese Immigrants from 1980-2013')
    plt.ylabel("Number of Immigrants")

    plt.show()
    ```
  - 

Scatter Plots

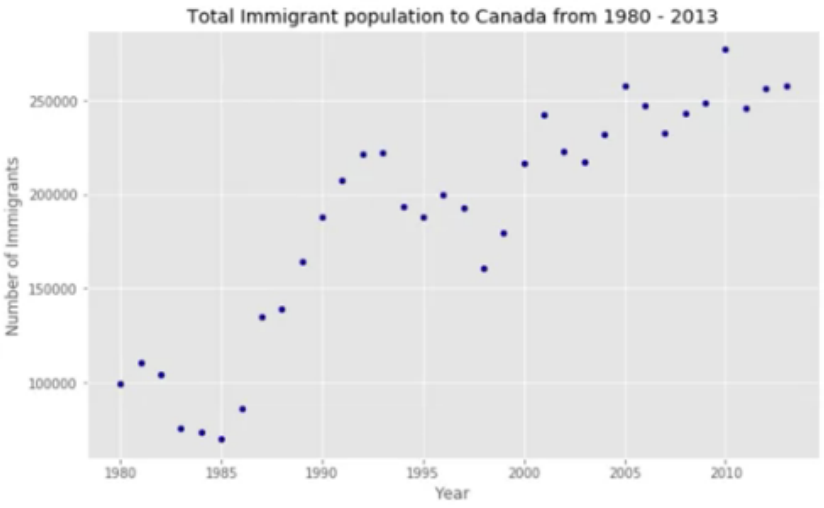
- Scatter plot
  - Displays values pertaining to two variables
  - Determines the correlation between the two variables
- Generating a scatter plot
  - ```
    import matplotlib as mpl
    import matplotlib.pyplot as plt
    ```
    ```

-
```
df_total.plot(
    kind='scatter',
    x='year',
    y='total',
)

plt.title('Total Immigrant population to Canada from 1980 - 2013')
plt.xlabel ('Year')
plt.ylabel('Number of Immigrants')

plt.show()
```
- df_total

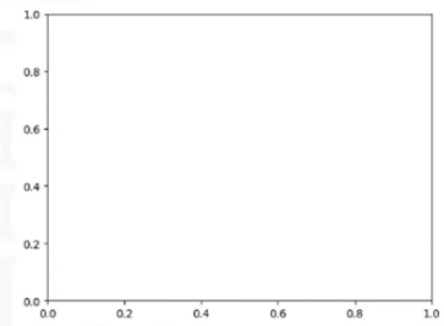
year	total
1980	99137
1981	110563
1982	104271
1983	75550
1984	73417
.	.
.	.

- 

Plotting Directly with Matplotlib

- Importing libraries, figures, and axes
-
```
import matplotlib.pyplot as plt
import numpy as np

import pandas as pd

# Create figure and axes
fig, ax = plt.subplots()
```


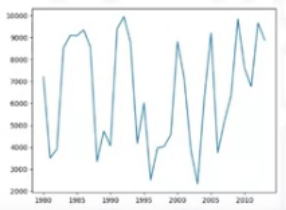
- Line plot and scatter plot

-
```
# Synthetic data
years = np.arange(1980, 2014)
immigrants = np.random.randint(2000, 10000, size = (34,))
```
- • Line Plot

```
#Create figure and axes
fig, ax = plt.subplots()

# Plot the line
ax.plot(years, immigrants)

# Display the plot
plt.show()
```

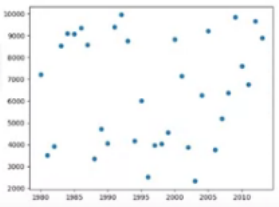


• Scatter Plot

```
#Create figure and axes
fig, ax = plt.subplots()

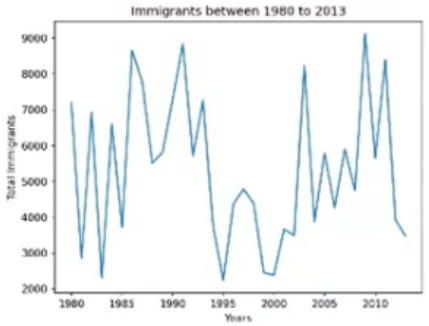
# Plot the line
ax.scatter(years, immigrants)

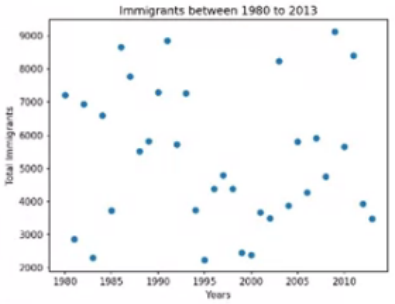
# Display the plot
plt.show()
```

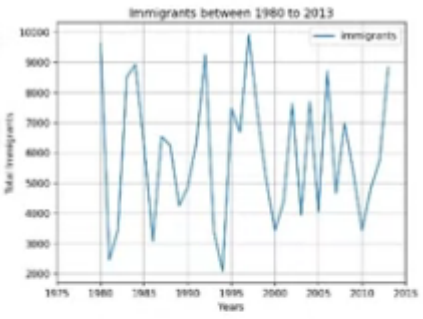


- Customizing attributes
  -
```
#add title
plt.title('Immigrants between 1980 to 2013')
#add labels
plt.xlabel('Years')
plt.ylabel('Total Immigrants')
```

```
ax.set_title('Immigrants between 1980 to 2013')
ax.set_xlabel('Years')
ax.set_ylabel('Total Immigrants')
```



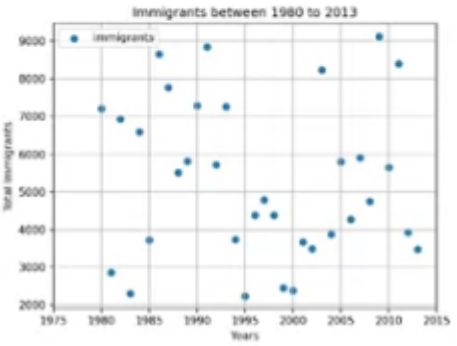




  -
```
#limits on x-axis
plt.xlim(1975, 2015)
#or ax.set_xlim()

#Enabling Grid
plt.grid(True)
#or ax.grid()

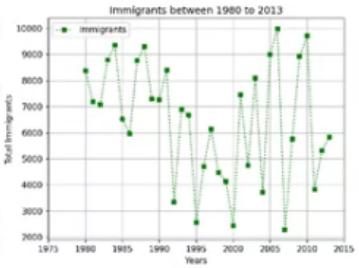
#Legend
plt.legend(["Immigrants"]) #or ax.legend()
```



- Marker styles, colors, and sizes

-

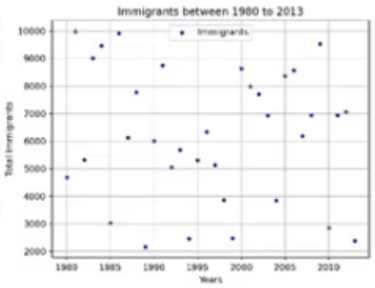
```
# Customizing the appearance of Plot
ax.plot(years, immigrants,
        marker='s',
        markersize=5,
        color='green',
        linestyle="dotted")
```



-

```
# Customizing Scatter Plot
ax.scatter(years, immigrants,
           marker='*',
           s = 20,
           color='darkblue')

#Legend at upper center of the figure
ax.legend(["Immigrants"], loc='upper center')
```

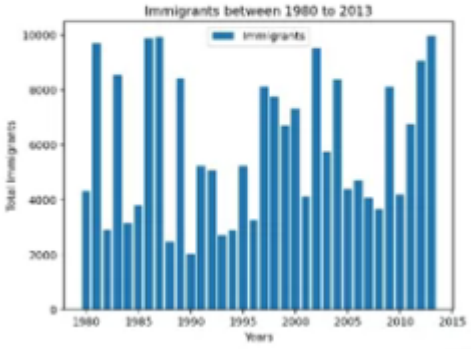


- IIs Network

- Bar plot, histogram, and Pie

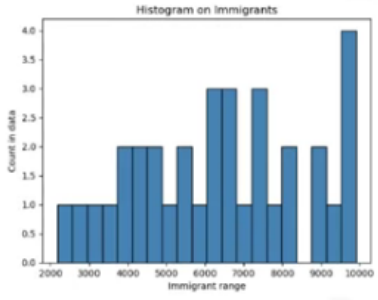
-

```
# Plot the bar
ax.bar(years, immigrants)
```

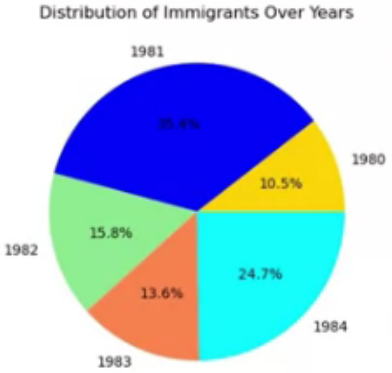


-

```
#histogram on immigrants
ax.hist(immigrants, bins=20, edgecolor='black',color='steelblue')
```



```
#Pie on immigrants
ax.pie(immigrants[0:5], labels=years[0:5],
       colors = ['gold','blue','lightgreen','coral','cyan'],
       autopct='%1.1f%%')
```

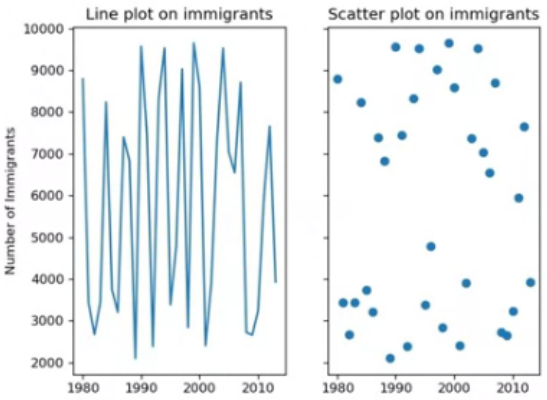


- Multiple plots and sub-plotting

```
# Create a figure with two axes in a row
fig, axs = plt.subplots(1, 2, sharey=True)

#Plotting in first axes - the left one
axs[0].plot(years, immigrants)
axs[0].set_title("Line plot on immigrants")

#Plotting in second axes - the right one
axs[1].scatter(years, immigrants)
axs[1].set_title("Scatter plot on immigrants")
```



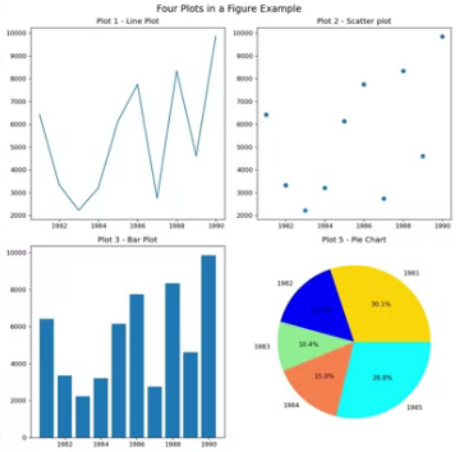
  - Three Arguments
    - Number of rows
    - Columns
    - Index of the subplot

```
# Add the first subplot (top-left)
axs1 = fig.add_subplot(2, 2, 1)
#Plotting in first axes - the left one
axs1.plot(total_immigrants)
axs1.set_title("Plot 1 - Line plot")
```

```
# Add the second subplot (top-right)
ax2 = fig.add_subplot(2, 2, 2)
ax2.scatter(years, immigrants)
ax2.set_title('Plot 2 - Scatter plot')

# Add the third subplot (bottom-left)
ax3 = fig.add_subplot(2, 2, 3)
ax3.bar(years, immigrants)
ax3.set_title('Plot 3 - Bar Plot')

# Add the fourth subplot (bottom-right)
ax4 = fig.add_subplot(2, 2, 4)
ax4.pie(immigrants[0:5], labels=years[0:5],
       colors = ['gold','blue','lightgreen','coral','cyan'],
       autopct='%1.1f%%')
ax4.set_aspect('equal')
ax4.set_title('Plot 5 - Pie Chart')
```



- Data storytelling and data visualization
    - Data storytelling
        - Is the art of storytelling
        - Creates a narrative around the data
    - Data visualization
        - Is an important aspect of data storytelling
        - Creates engaging visuals

❖ MODULE #3: Advanced Visualizations and Geospatial Data

Waffle Charts & Word Cloud

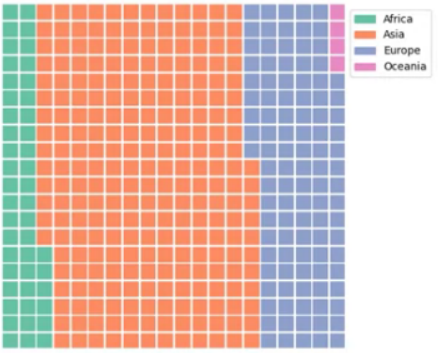
- Waffle charts
    - Represents categorical data in the form of:
        - Square tiles
        - Cells
    - Displays proportion or percentage of different categories
    - Simplifies data for all types of audiences
    - Use case
        - Market share analysis
        - Demographic representation
        - Budget allocation
        - Survey responses
        - Election results
        - Product sales analysis
- PyWaffle library
    -

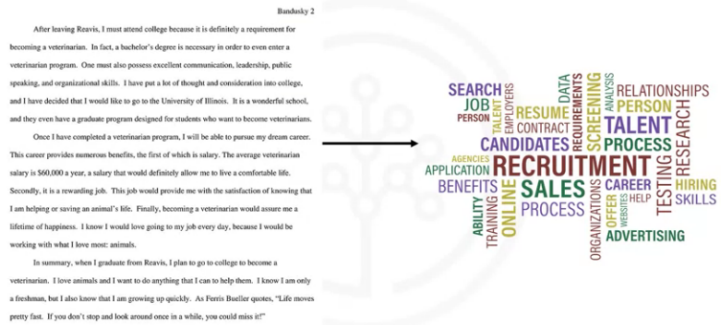
```
import matplotlib.pyplot as plt
from pywaffle import Waffle

# Create data for the waffle chart
data = df_dsn[['Africa', 'Asia','Europe','Oceania']]

# Set up the waffle chart figure
fig = plt.figure(
    FigureClass=Waffle,
    rows=20,
    columns=20,
    values=data,
    legend={'labels':['Africa', 'Asia','Europe','Oceania'],
            'loc': 'upper left', 'bbox_to_anchor': (1, 1)})

# Display the chart
plt.show()
```

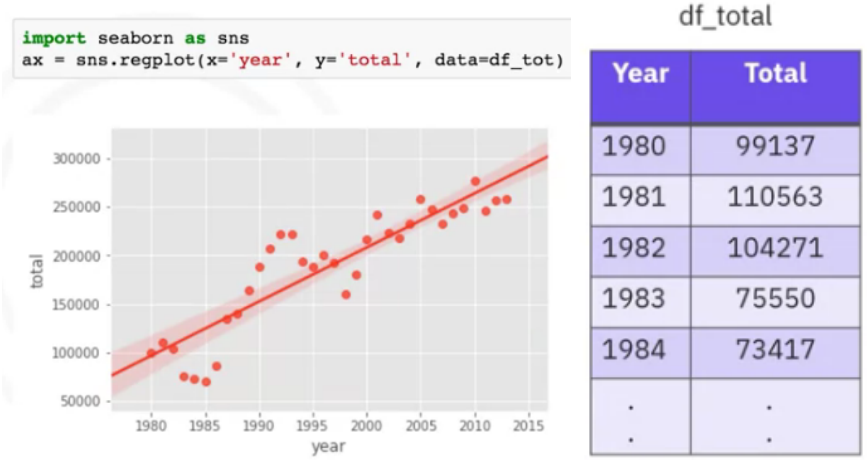


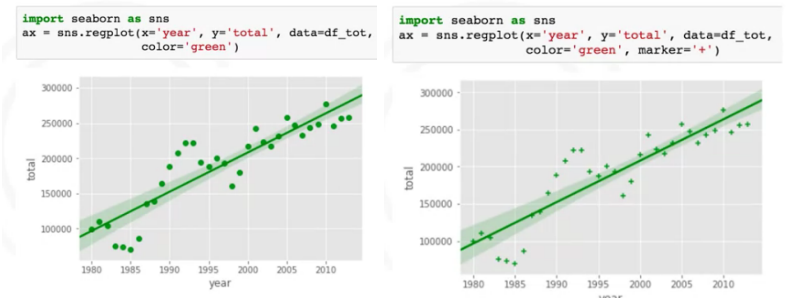
- Word cloud
    - Is a popular data visualization method
    - Presents a concise summary of textual content
    - Depicts the importance of different words
    - 

    - Use case
        - Social media analysis
        - Customer feedback analysis
        - Content analysis

        - Market research
        - Resume or job description analysis

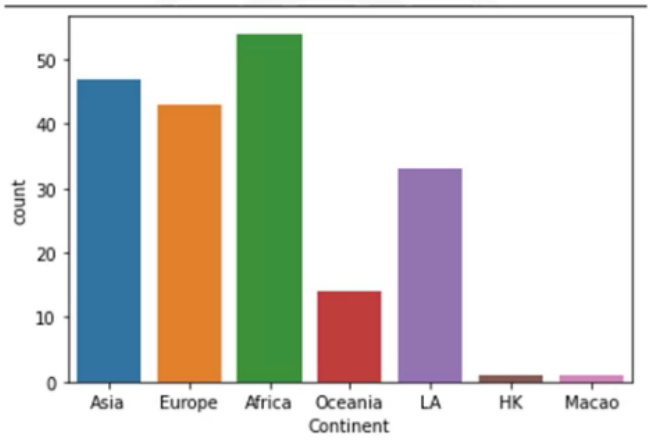
Seaborn and Regression Plots

- Seaborn
    - Seaborn is a Python visualization library based on Matplotlib
    - Seaborn offers built-in themes and color palettes to improve your plot visuals
    - Visuals that need ~20 lines of code using Matplotlib to be created, with Seaborn, the number of lines of code is reduced by 5-fold
    - Special Plots
        - Regression plot
        - Distribution plot
        - Categorical plot
        - Creating Scatter plots with Regression lines
- Regression Plots
    - 

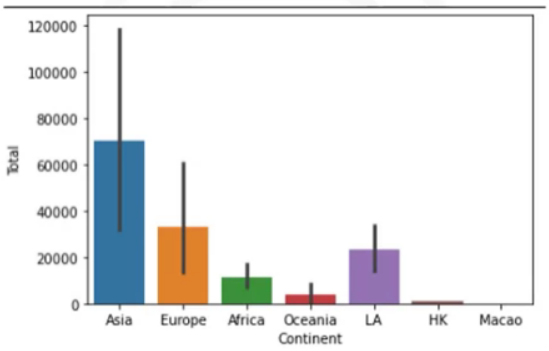
    - 

- Categorical Plots

- sns.countplot(x='Continent', data=df_canada)



- sns.barplot(x='Continent',y='Total',data=df_canada)



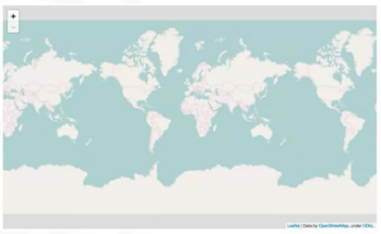
Introduction to Folium

- What is Folium?
  - Folium is a powerful data visualization library in Python that was built primarily to help people visualize geospatial data
  - With Folium, you can create a map of any location in the world using latitude and longitude values. You can also create a map and superimpose markers and clusters on top of the map for interesting visualizations.
  - You can also create maps of different styles, such as street-level maps, stamen maps, and a couple of others.
- Creating a world map
  -

```
#import library
import folium

# define the world map
world_map = folium.Map()

# display world map
world_map
```



- Creating a map of Canada

```
# define the world map centered around
# Canada with a low zoom level
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4
)

# display world map
world_map
```

- Map styles
  - Stamen Toner

```
# create a Stamen Toner map of
# the world centered around Canada
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4,
    tiles='Stamen Toner'
)

# display map
world_map
```

  - Stamen Terrain

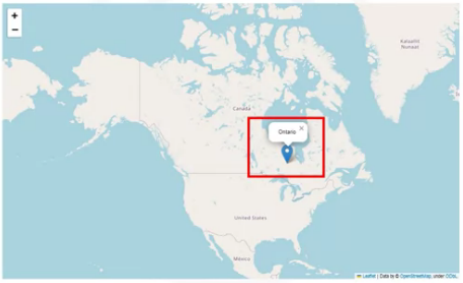
```
# create a Stamen Toner map of
# the world centered around Canada
world_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4,
    tiles='Stamen Terrain'
)

# display map
world_map
```

Maps with Markers

- Add marker and label

```
# Add a marker for Ontario province
folium.Marker(location=[51.2538, -85.3232], popup='Ontario').add_to(canada_map)
```



- Add marker with feature group

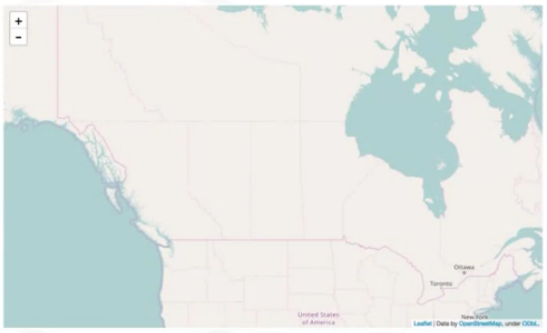
-
```
# generate map of Canada
canada_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4
)

## add a red marker to Ontario

# create a feature group
ontario = folium.map.FeatureGroup()

# style the feature group
ontario.add_child(
    folium.features.CircleMarker(
    [51.25, -85.32], radius = 5,
    color = "red", fill_color = "Red"
    )
)

# add the feature group to the map
canada_map.add_child(ontario)
```



```
# display map
canada_map
```

- Label the marker
-
```
# generate map of Canada
canada_map = folium.Map(
    location=[56.130, -106.35],
    zoom_start=4
)

## add a red marker to Ontario

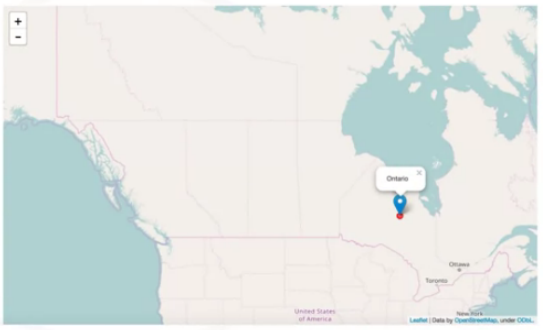
# create a feature group
ontario = folium.map.FeatureGroup()

# style the feature group
ontario.add_child(
    folium.features.CircleMarker(
    [51.25, -85.32], radius = 5,
    color = "red", fill_color = "Red"
    )
)

# add the feature group to the map
canada_map.add_child(ontario)

# label the marker
folium.Marker([51.25, -85.32],
    popup='Ontario').add_to(canada_map)

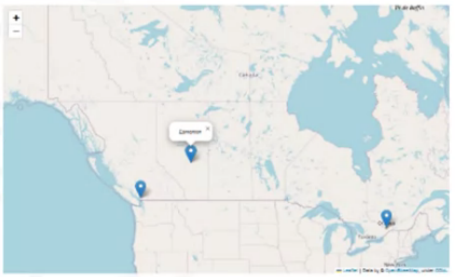
# display map
canada_map
```



- Multiple markers
-
```
# Define a list of locations and their corresponding popups
locations = [
    {"location": [45.4215, -75.6989], "popup": "Ottawa"},
    {"location": [53.5461, -113.4938], "popup": "Edmonton"},
    {"location": [49.2827, -123.1207], "popup": "Vancouver"},
    # Add more locations and their popups here
    ]

# Add markers for each location in the list
for loc in locations:
    folium.Marker(location=loc["location"],
                  popup=loc["popup"]).add_to(map)

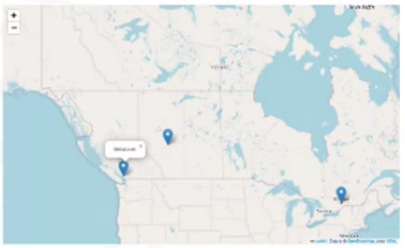
# Display the map with the markers
map
```



-
```
#import MarkerCluster
from folium.plugins import MarkerCluster

# Create a MarkerCluster object
marker_cluster = MarkerCluster().add_to(map)

# Add markers for each location in the list to the MarkerCluster
for loc in locations:
    folium.Marker(location=loc["location"],
                  popup=loc["popup"]).add_to(marker_cluster)
```



Choropleth Maps

- Choropleth map
    - A choropleth map is a thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable.
    - When creating a choropleth map, Folium requires a GeoJson file that includes geospatial data of the region.
    - The Mapbox Bright Tileset displays the name of every country when

used on a map.

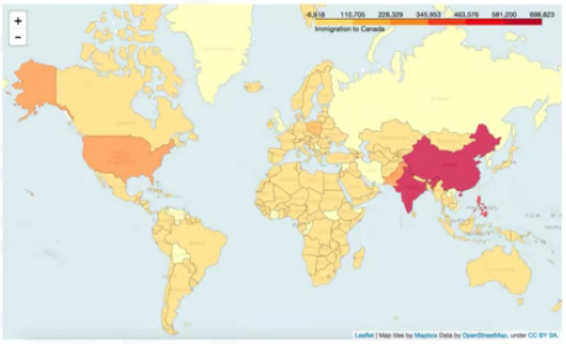
- Creating the Map
  -

```
# create a plain world map
world_map = folium.Map(
    zoom_start=2,
    tiles='Mapbox Bright'
)

## geojson file
world_geo = r'world_countries.json'

# generate choropleth map using the total
# population of each country to Canada from
# 1980 to 2013
world_map.choropleth(
    geo_path=world_geo,
    data=df_canada,
    columns=['Country', 'Total'],
    key_on='feature.properties.name',
    fill_color='YlOrRd',
    legend_name='Immigration to Canada'
)

# display map
world_map
```



❖ MODULE #4: Creating Dashboards with Plotly and Dash

Dashboarding Overview

- Dashboard
  - Real-time visuals simplify business moving parts
  - Display Key Performance Indicators (KPI) for analysis
  - Help businesses by providing the big picture
  - Best dashboards answer important business questions
- Scenario
  - Without dashboard

1.

Reporting_Airline	Flights
WN	13972
DL	10241
AA	9853
OO	8515
UA	6430
YX	3416
MQ	3368
B6	3170
OH	2969
AS	2783

2.

Month	Flights
1	6125
2	5578
3	6553
4	6282
5	6441
6	6583
7	6798
8	6753
9	6140
10	6533
11	6280
12	6550

3.

DestState	DistanceGroup	Flights
AK	10	4
AL	6	1
AR	7	1
AZ	2	361
AZ	3	220
AZ	4	9
CA	1	283
CA	2	2256
CA	3	36
CO	3	19
CO	4	401
CO	5	2
CT	11	2
FL	9	51
FL	10	90
FL	11	26
GA	8	114
GA	9	65
HI	10	168
HI	11	116
ID	2	7
ID	3	60
IL	7	169
IL	8	154
IN	8	16
KY	8	13
KY	9	5
LA	7	33
LA	8	14
MA	11	128

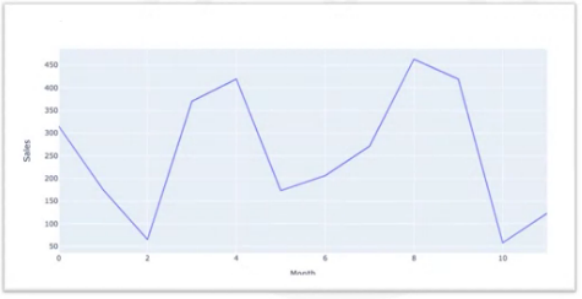
  - With dashboard

- Web-based dashboarding
  - Plotly Dash
  - Panel
  - Voila
  - Streamlit
- Dashboard tools
  - Matplotlib
  - Bokeh
  - Ipywidgets
  - Bowtie
  - Flask

Introduction to Plotly

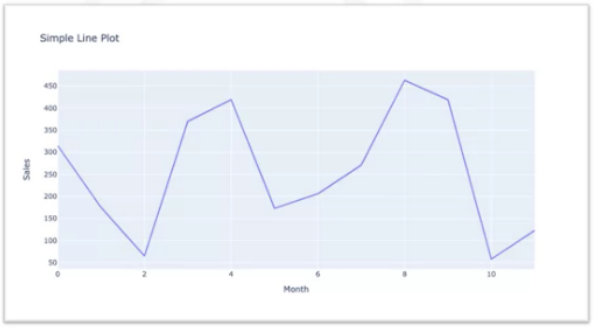
- An overview
  - Interactive, open-source plotting library
  - Supports over 40 unique chart types
  - Includes various types of charts
  - Visualization can be
    - Displayed in Jupyter notebook
    - Saved to HTML files
    - Used in developing Python-build web applications
- Plotly sub-modules
  - Plotly Graph Objects: Low-level interface to figures, traces, and layout
    - `plotly.graph_objects.Figure`
  - Plotly Express: High-level wrapper
- Using plotly.graph_objects
  -
```
# Import required packages
import plotly.graph_objects as go
import plotly.express as px
import numpy as np

# Set random seed for reproducibility
np.random.seed(10)
x = np.arange(12)
# Create random y values
y = np.random.randint(50, 500, size=12)
```
  - Plotly.graph contains a JSON object
  - Here, ‘go’ is the plotly JSON object
  - Chart is plotted by updating the values
  - Figure is created by adding a trace
    -
```
fig = go.Figure(data=go.Scatter(x=x, y=y))
```
    -
```
fig = go.Figure(data=go.Scatter(x=x, y=y))

fig.update_layout(title='Simple Line Plot', xaxis_title='Month', yaxis_title='Sales')
fig.show()
```


  - Using plotly.express

```
# Entire line chart can be created in a single command
fig = px.line(x=x, y=y, title='Simple Line Plot', labels=dict(x='Month', y='Sales'))
fig.show()
```



Introduction to Dash

- An overview
  - Open-source User Interface Python library from Plotly
  - Easy to build GUI
  - Declarative and Reactive
  - Rendered in a web browser and can be deployed to servers
  - Inherently cross-platform and mobile ready
- Dash Components
  - Core components
    - `import dash_core_components as dcc`
  - HTML Components
    - `import dash_html_components as html`
- Core Components
  - Higher-level components that are interactive and are generated with JavaScript, HTML, and CSS through the React.js library
  - Example: Creating a slider, input area, check items, datepicker, and other components
- HTML Components
  - Components for every HTML tag
  - Keyword arguments describe the HTML attributes like style, className, and id

Make Dashboards Interactive

- Dash: Callbacks
  - The Callback function is a python function
  - It is automatically called by Dash
    - Decorator
    - `@app.callback`

- Callback function

```
def callback_function:
    .....
    .....
    .....
    return some_result
```

  - @app.callback(Output, Input)
  - Two parameters:
    - Output: It sets results returned from the callback
    - Input: It is provided to the callback function
- Callback with one input
  -
```
# Import required packages
import pandas as pd
import plotly.express as px
import dash
import dash_html_components as html
import dash_core_components as dcc
from dash.dependencies import Input, Output
# Read the data
airline_data = pd.read_csv('airline_2m.csv',
                           encoding = "ISO-8859-1",
                           dtype={'Div1Airport': str,
                           'Div1TailNum': str,
                           'Div2Airport': str,
                           'Div2TailNum': str})
```
  -
```
app = dash.Dash()
# Design dash app layout
app.layout = html.Div(children=[ html.H1('Airline Dashboard',
                           style={'textAlign': 'center', 'color': colors['text'],
                           'font-size': 40}),
                           html.Div(["Input: ", dcc.Input(id='input-yr', value='2010',
                           type='number', style={'height':'50px', 'font-size': 35}),],
                           style={'font-size': 40}),
                           html.Br(),
                           html.Br(),
                           html.Div(dcc.Graph(id='bar-plot')),
                           ])
```
  -
```
@app.callback( Output(component_id='bar-plot', component_property='figure'),
               Input(component_id='input-yr', component_property='value'))
def get_graph(entered_year):
    # Select data
    df = airline_data[airline_data['Year']==int(entered_year)]
    # Top 10 airline carrier in terms of number of flights
    g1 = df.groupby(['Reporting_Airline'])['Flights'].sum().nlargest(10).reset_index()
    # Plot the graph
    fig1 = px.bar(g1, x='Reporting_Airline', y='Flights', title='Top 10 airline carrier in
                  year ' + str(entered_year) + ' in terms of number of flights')
    fig1.update_layout()
    return fig1
if __name__ == '__main__':
    app.run_server(port=8002, host='127.0.0.1', debug=True)
```
  - 

- Callback with two inputs

-
```
app = dash.Dash()
# Design dash app layout
app.layout = html.Div(children=[ html.H1('Airline Dashboard', style={'textAlign': 'center',
                                 'color': colors['text'], 'font-size': 40}),
                                 html.Div(["Year: ", dcc.Input(id='input-yr', value='2010',
                                 type='number', style={'height':'50px', 'font-size': 35}),
                                 ], style={'font-size': 40}),
                                 html.Div(["State Abbreviation: ", dcc.Input(id='input-ab',
                                 value='AL', type='text', style={'height':'50px',
                                 'font-size': 35})], style={'font-size': 40}),
                                 html.Br(),
                                 html.Br(),
                                 html.Div(dcc.Graph(id='bar-plot')),
                                 ])
```
-
```
@app.callback( Output(component_id='bar-plot', component_property='figure'),
               [ Input(component_id='input-yr', component_property='value'),
                 Input(component_id='input-ab', component_property='value')])

def get_graph(entered_year, entered_state):
    # Select data
    df = airline_data[ (airline_data['Year']==int(entered_year)) &
                       (airline_data['OriginState'] == entered_state)]
    # Top 10 airline carrier in terms of number of flights
    ....................
    fig1.update_layout()
    return fig1
if __name__ == '__main__':
app.run_server(port=8002, host='127.0.0.1', debug=True)
```
-