Data Visualization

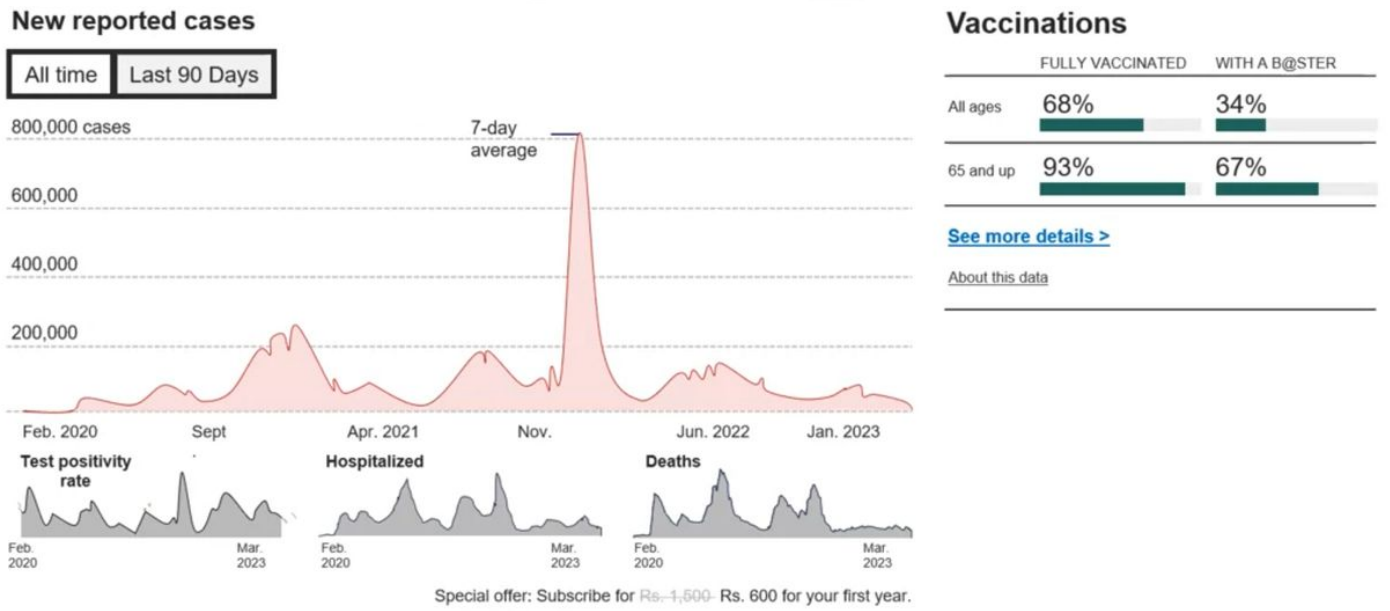
Agenda:

1	Introduction to Matplotlib
2	Introduction to Seaborn
3	Practical Visualization Examples

Understanding Data Visualization

- What is Data Visualization?
  - Graphical representation of data and information.
- Importance of Data Visualization
  - Helps understand complex datasets easily.
  - Highlights patterns, trends, and relationships.
- Uses for Data Visualization
  - Communicates insights to stakeholders effectively.
  - Identifies trends and data patterns.

Leveraging the power of data visualization



SOURCE: https://www.nytimes.com/interactive/2021/us/covid-cases.html

Best Practices in Data Visualization

- Choosing the Right Visualization
  - Different types for different data types.
- Keeping Visualizations Simple
  - Clear labels, titles, and legends.
- Audience Consideration
  - Tailoring visualizations to audience needs.
- Examples of Effective and Ineffective Visualizations
  - Example of bad data visualization with clutter.

Darkhorse Analytics' effective approach

When creating a visual, always remember:

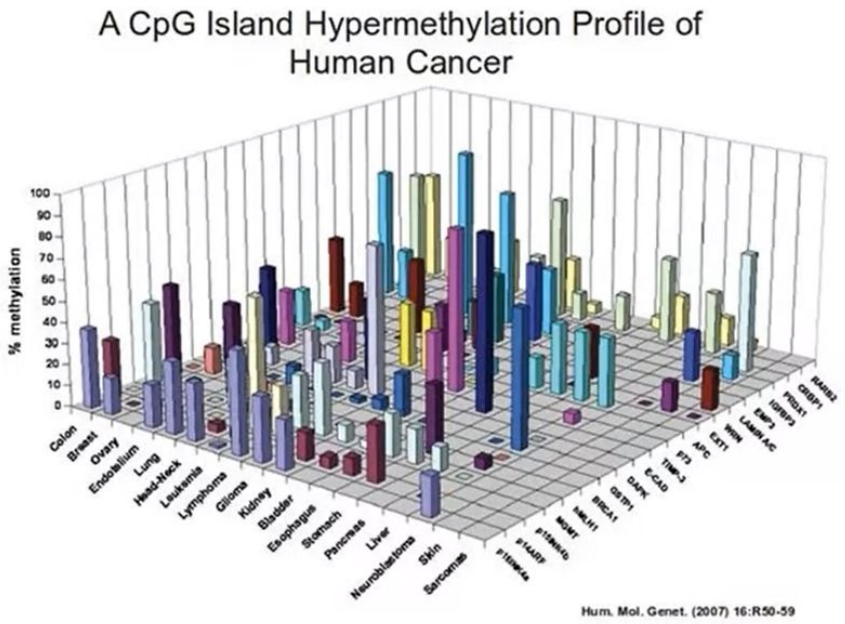
- Less is more effective
- Less is more attractive
- Less is more impactful

- Cause & Effect, but Hidden
- Data Over Opinions
- Know the System Before You Move

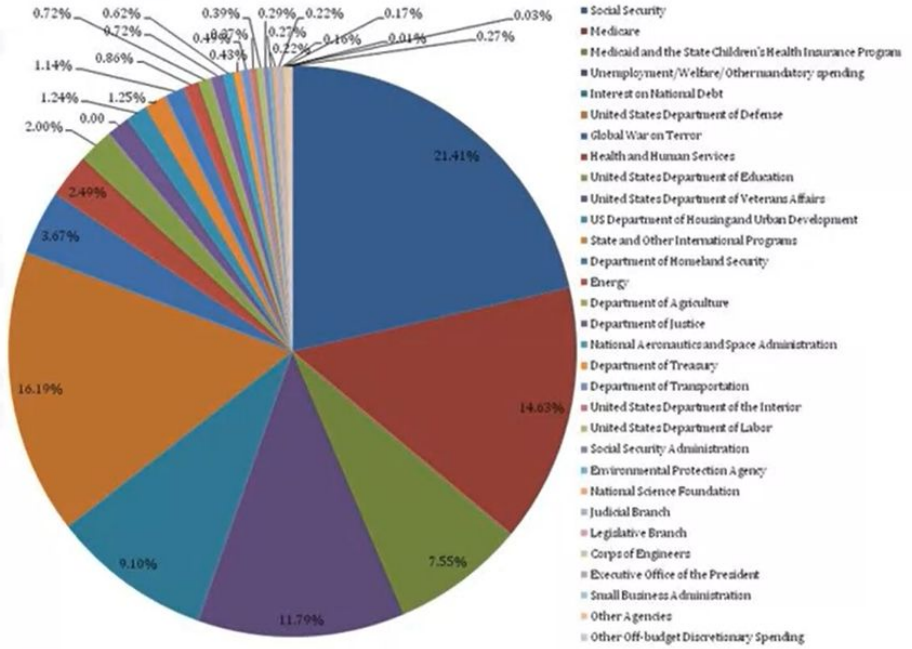
A chart is not art,
a chart is a message.
If the message isn't
clear, the chart
has failed.

– Baraa's Visualization Rule

Best practices (bad example)

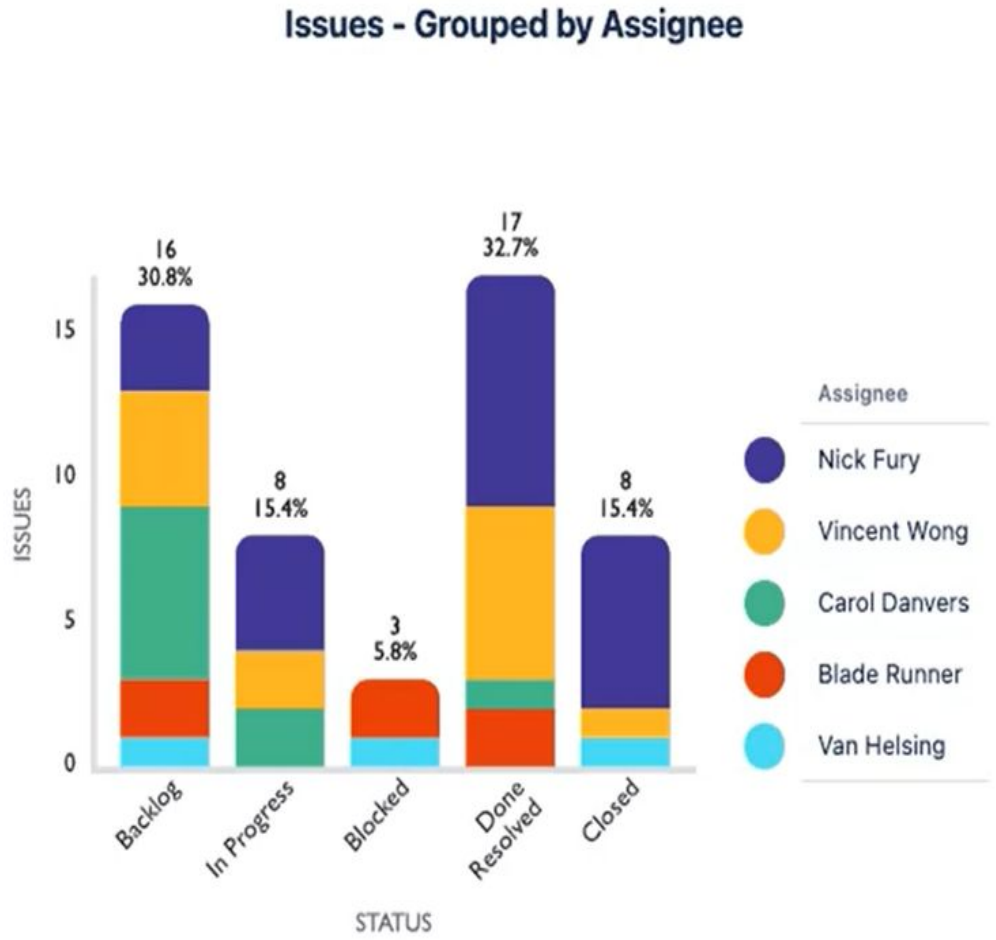


SOURCE: https://www.livestories.com/blog/five-ways-to-fail-data-visualization



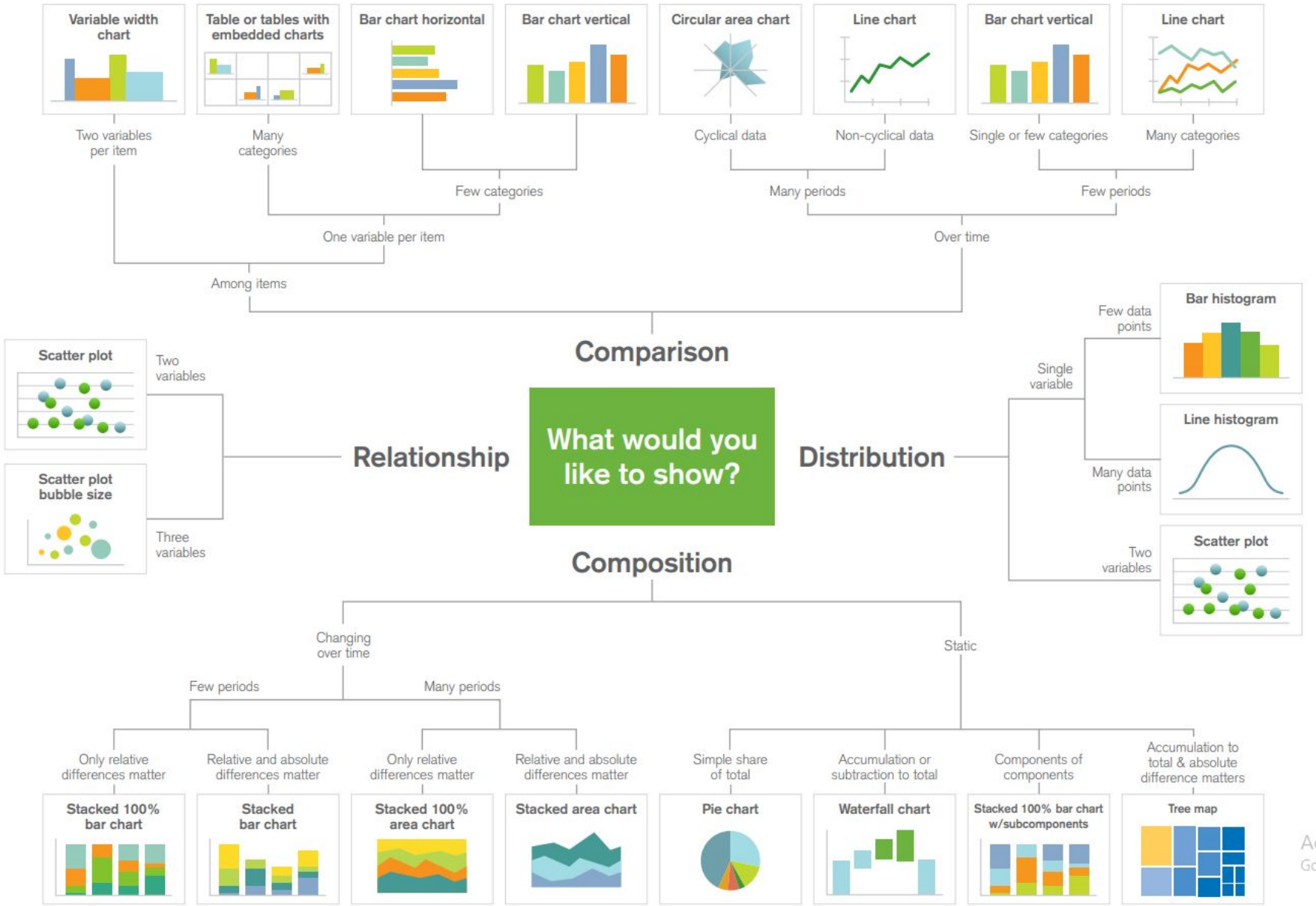
SOURCE: https://en.wikipedia.org/wiki/2007_United_States_federal_budget#.

Best practices (good example)



SOURCE: https://www.oldstreetsolutions.com/good-and-bad-data-visualization

Visualizations

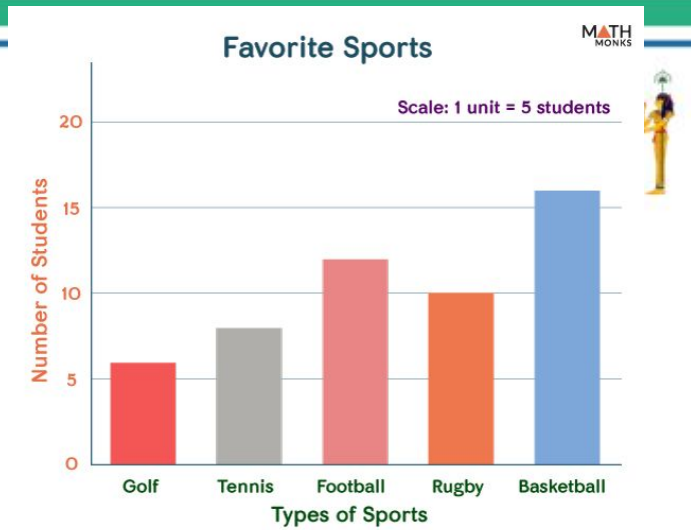


Source: ©A. Abela, 2010. www.ExtremePresentation.com

Added
By Eng. Baraa

Data Visualization Cheat Sheet

What do you want to show?

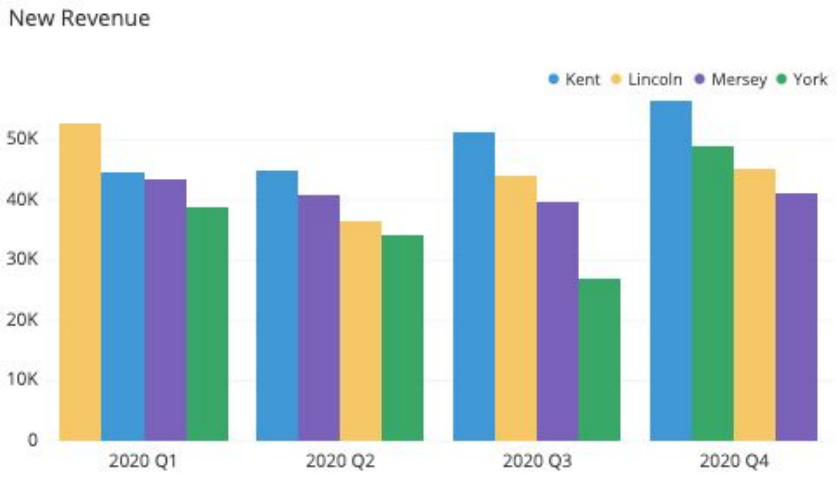


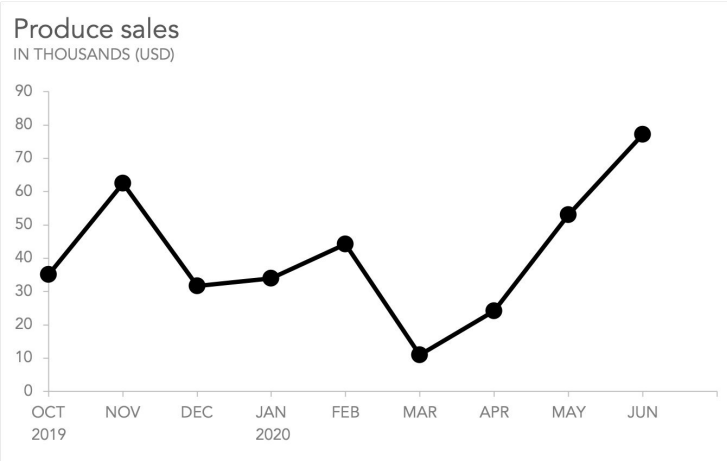
Comparison

👉 When you want to show **differences between values or categories**.

✅ Best Visuals:

- 📊 **Bar Chart** → Compare bran
- 📈 **Line Chart** → Trends over ti
- 🟦 **Grouped Bar** → Two groups



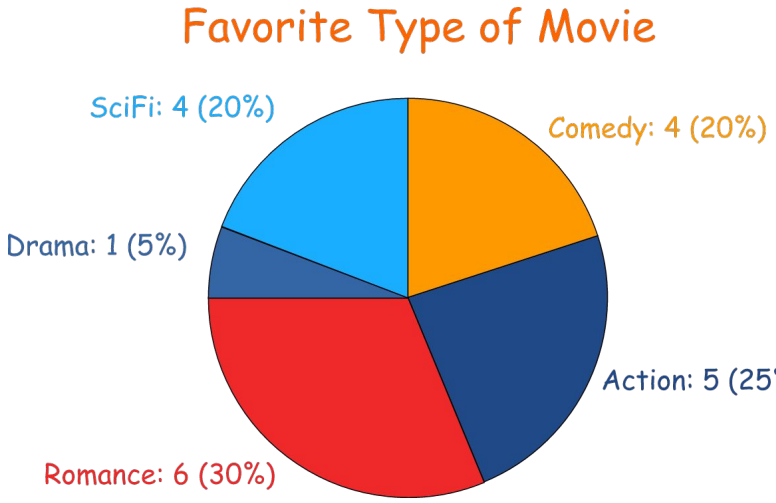


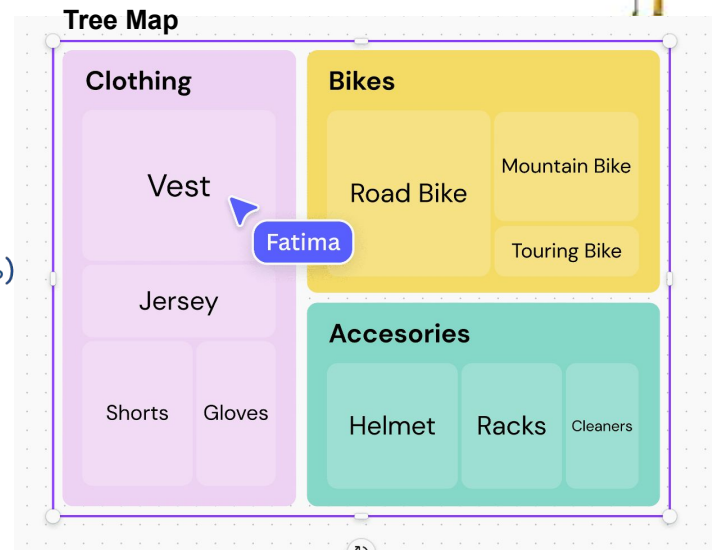
رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Data Visualization Cheat Sheet

What do you want to show?



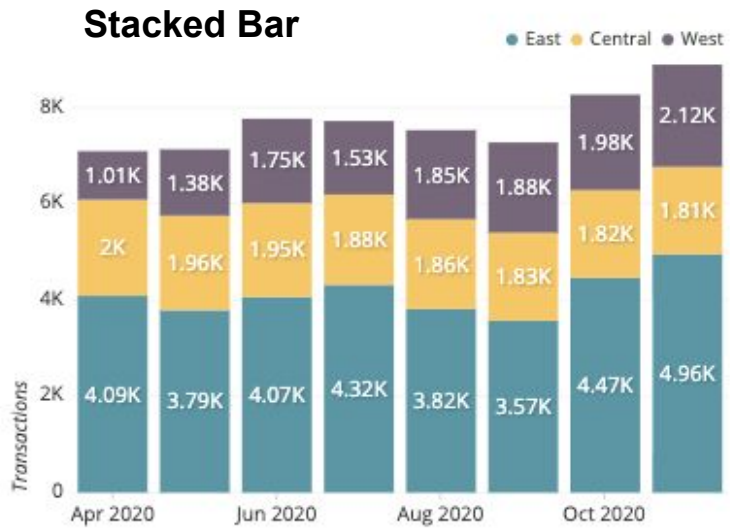


Composition

👉 When you want to show **how the whole is divided into parts**.

✅ Best Visuals:

- 🥧 **Pie Chart** → Percentages
- 📊 **Stacked Bar** → Parts of a bar
- 🟩 **Tree Map** → Area for each part
- 📈 **Stacked Area** → Parts changing over time

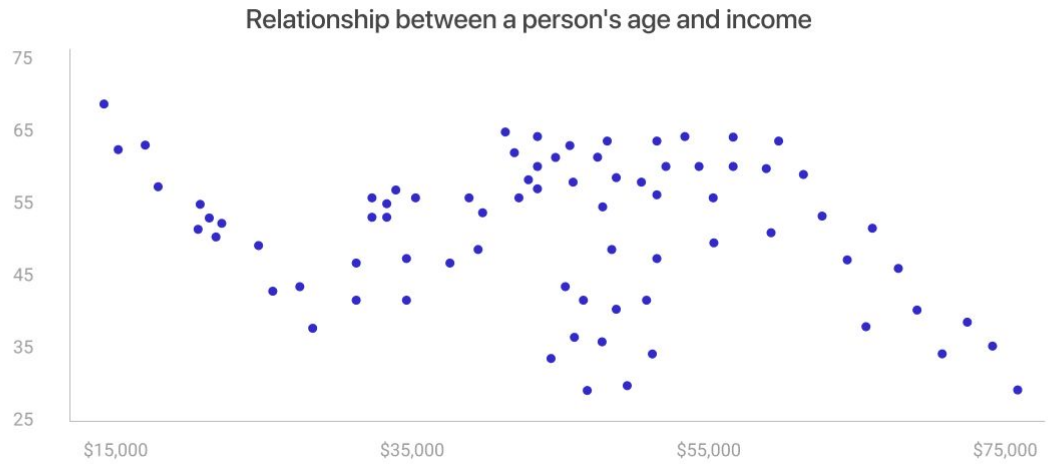


رواد مصر الرقمية

Data Visualization Cheat Sheet

What do you want to show?

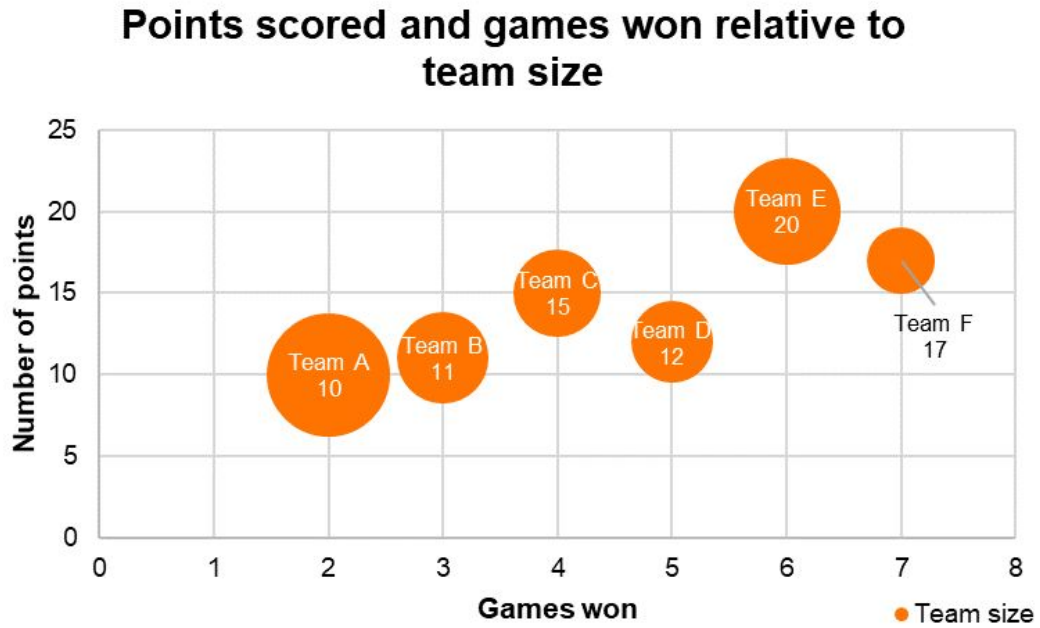
Relationship

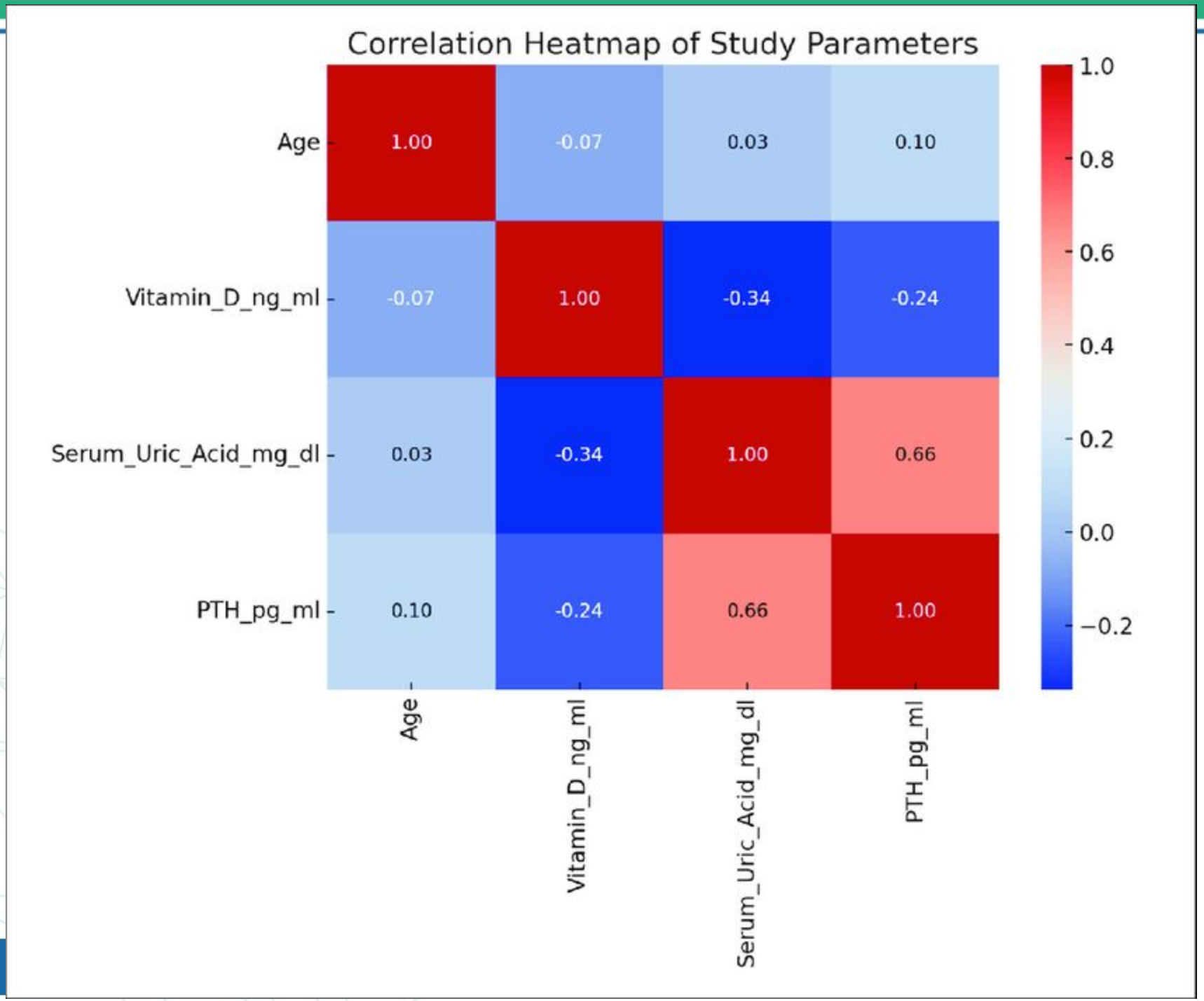


👉 When you want to show **connections or correlations between variables**.

✅ Best Visuals:

- 🔵 **Scatter Plot** → Variable × Variable
- 🔵⚪ **Bubble Chart** → Add a third variable (size)
- 🌡️ **Heatmap** → Correlation strength
- 📉 **Trend Line** → Direction of relationship



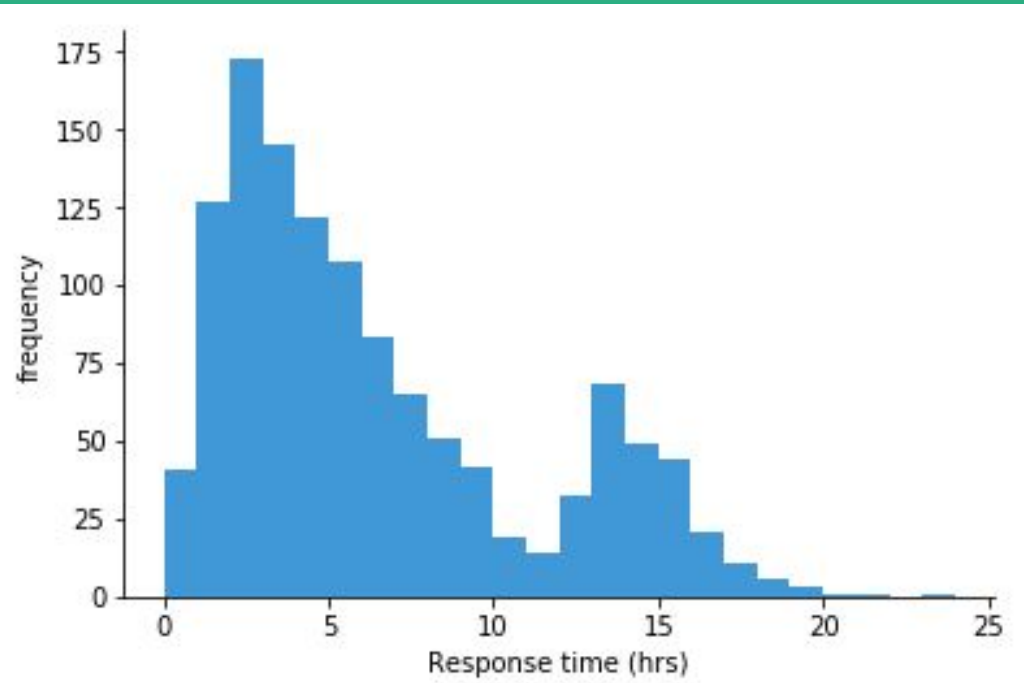

Correlation Heatmap of Study Parameters
Age
Vitamin_D_ng_ml
Serum_Uric_Acid_mg_dl
PTH_pg_ml
1.00
-0.07
0.03
0.10
-0.07
1.00
-0.34
-0.24
0.03
-0.34
1.00
0.66
0.10
-0.24
0.66
1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Age
Vitamin_D_ng_ml
Serum_Uric_Acid_mg_dl
PTH_pg_ml

رواد مصر الرقمية

Data Visualization Cheat Sheet

What do you want to show?

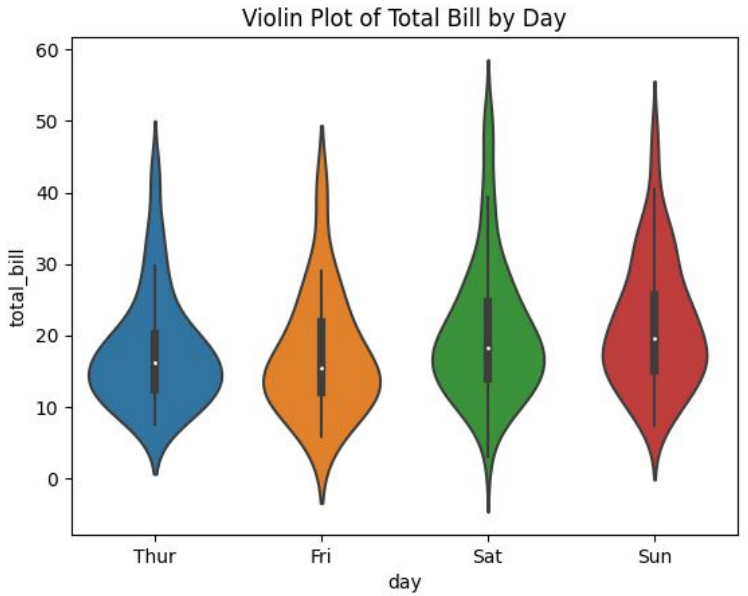
Distribution



👉 When you want to show **the shape of data and outliers**.

✅ Best Visuals:

- **Bar Chart → used for discrete data (Branchs: Gaza, Egypt , Dubai)**
- **Histogram → used for continuous data (Age, Salary , Grades)**
- 📦 **Boxplot** → Median + Outliers
- 🌊 **Density Plot (KDE)** → Smooth curve
- 🎻 **Violin Plot** → Box + Density combined



Quick Rules

- **Few categories?** → Bar / Pie
- **Over time?** → Line / Area
- **Relationship?** → Scatter / Heatmap
- **Distribution?** → Histogram / Boxplot

📌 Rule: **Always ask: What do I want to show?**

Hints

🟦 Comparison

- If you want to **focus on fine differences between categories** → Bar Chart.
- If you want to **track changes over time** → Line Chart.

🟩 Composition

- If you want **clear and simple percentages** → Pie Chart.
- If you have **time series data and want to see how the whole and its parts change over time** → Stacked Area Chart.
- If you want to **compare categories across multiple groups without a time dimension** → Stacked Bar Chart.

🟧 Distribution

- If your goal is to **see the overall data pattern** → Histogram.
- If your goal is to **detect outliers (unusual values)** → Box Plot.

🟥 Relationship

- If you have **two variables only** → Scatter Plot.
- If you have **three variables** → Bubble Chart.
- If you want to **see clusters in the data** → Scatter Clusters.

Choose the Right Chart – Real-World Scenarios

Module I

Introduction Data visualization tools

Plot Libraries

Popular plot libraries:

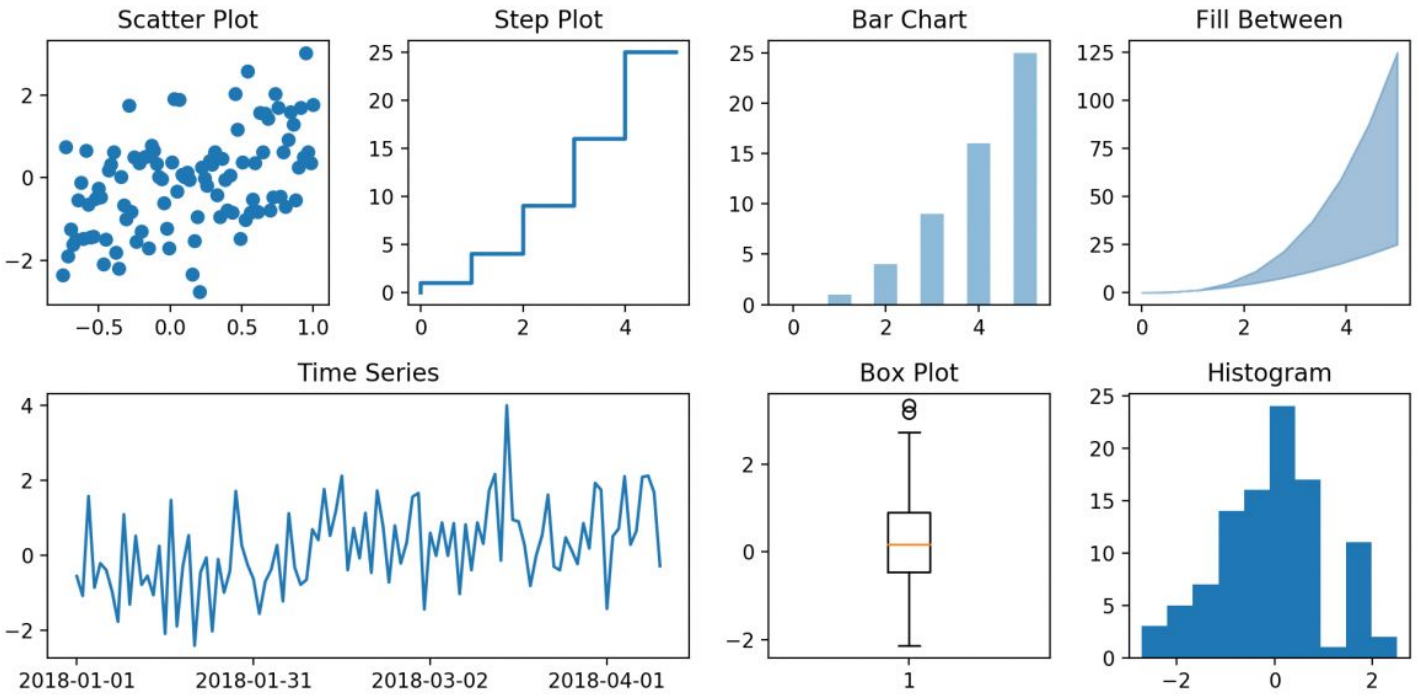
Plot libraries in Python:

- Matplotlib
- Pandas
- Seaborn
- Folium
- Plotly
- PyWaffle

Matplotlib:

- General purpose plotting library
- Integrates well with libraries and frameworks

matplotlib



Matplotlib:

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

Pandas:

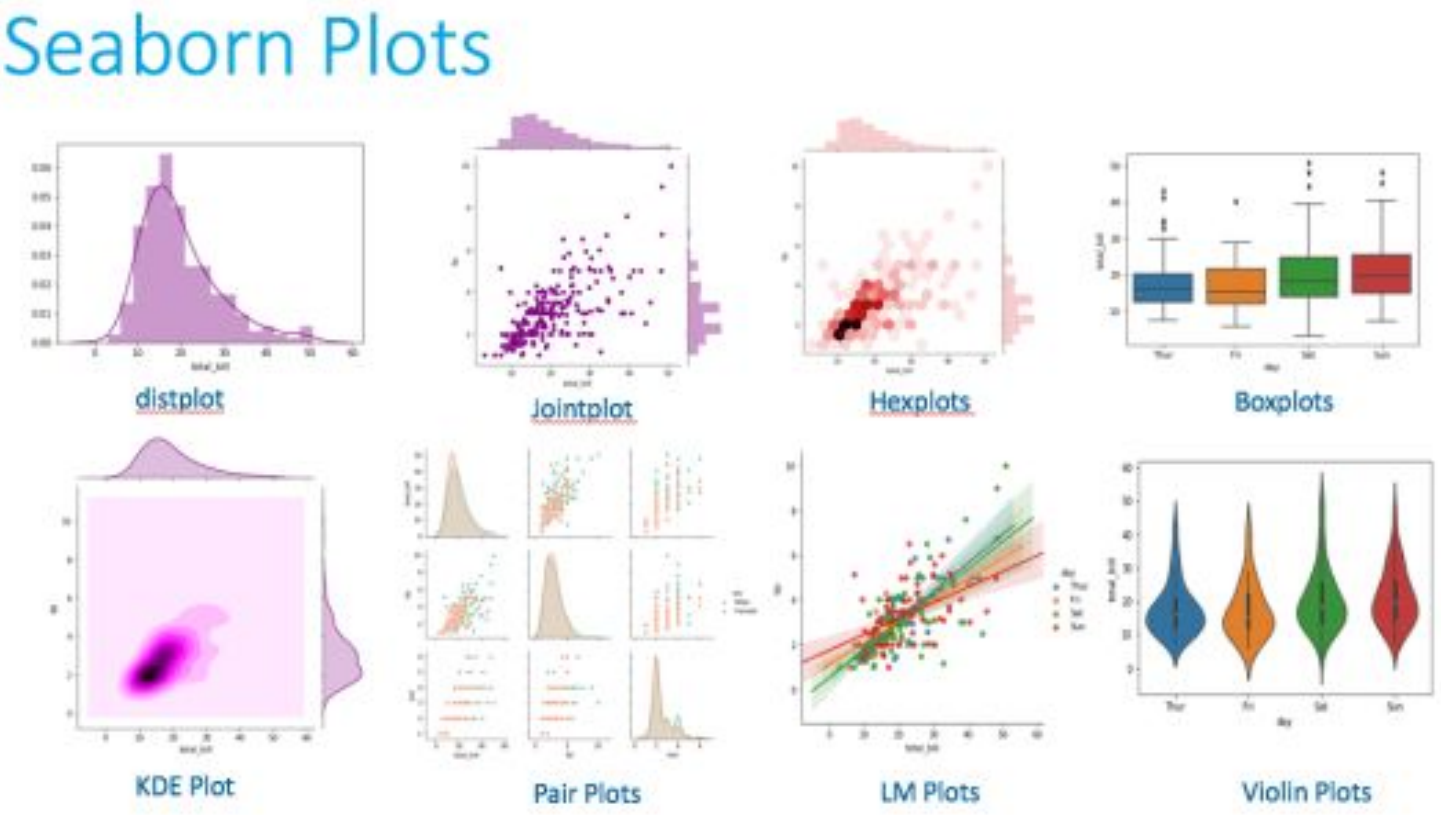
Features

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



Seaborn:

Features

Offers various color palettes and styles

Offers functions to combine multiple plots in a grid layout

Integrates with Pandas

رواد مصر الرقمية

وزارة الاتصالات وتكنولوجيا

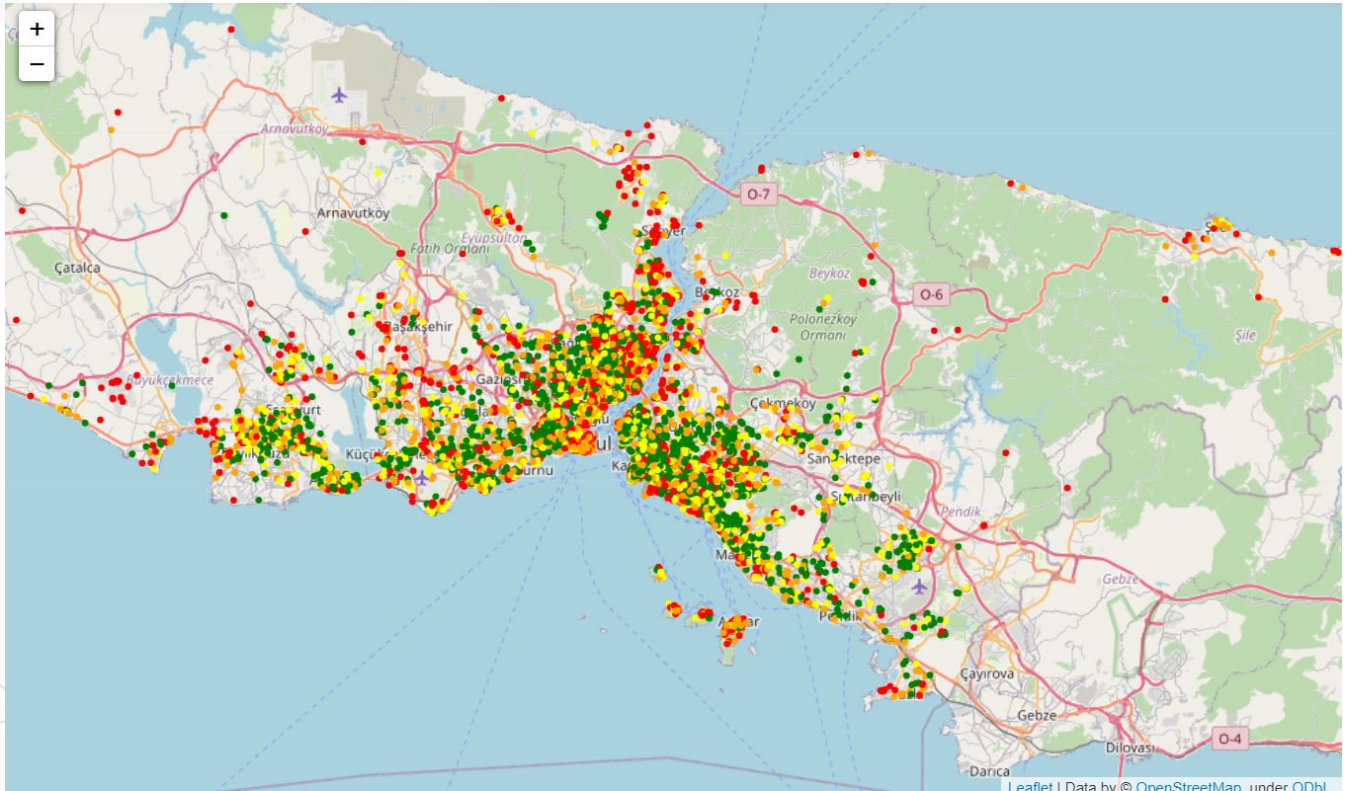
Folium:

Folium

- Excellent option for geospatial data visualization
- Builds interactive and customizable maps

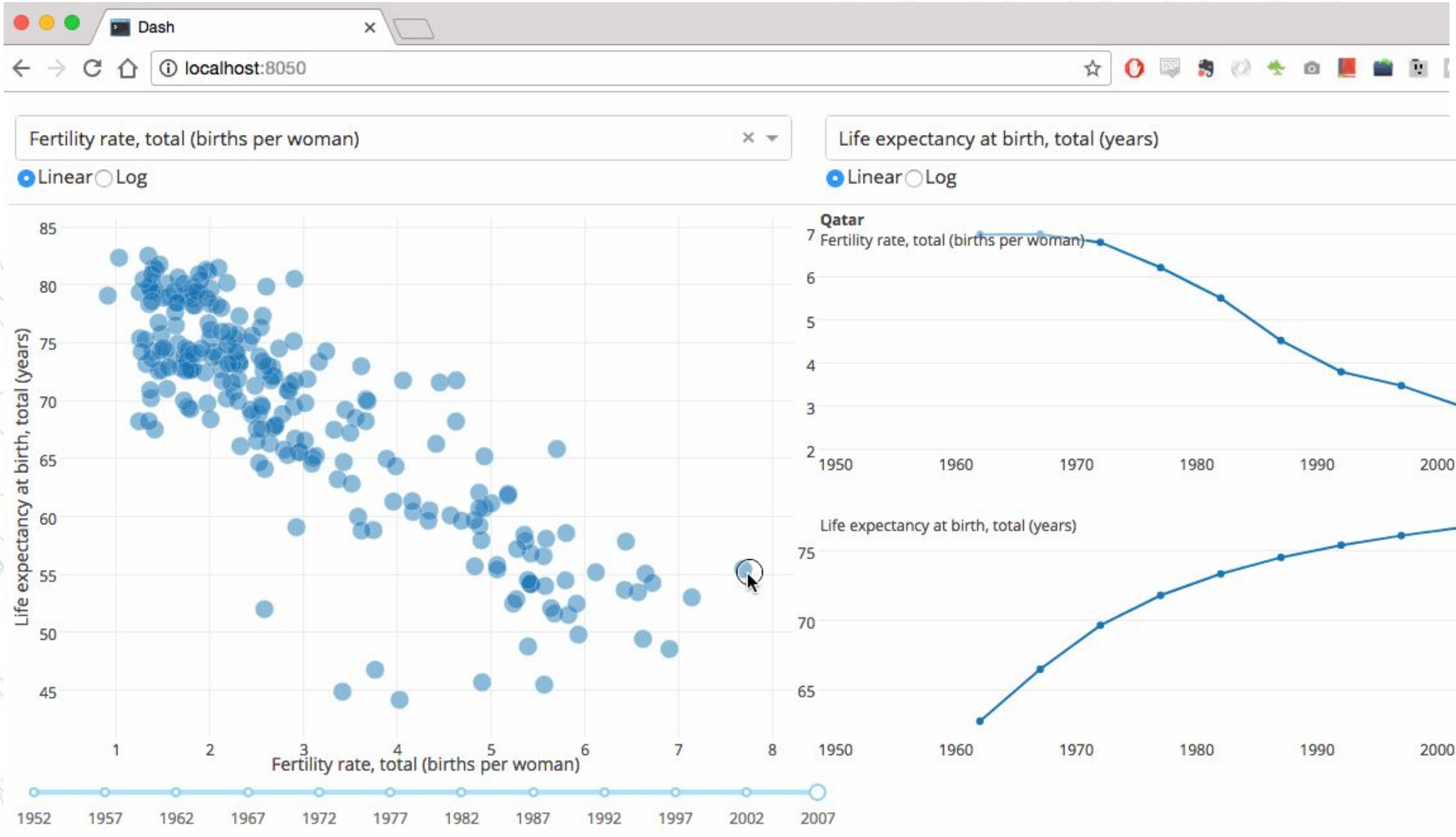
Feature

Integrates with popular data analysis libraries



Plotly & Dash:

- Has highly interactive plots and dashboards
- Can create a variety of plots



Plotly:

Builds interactive dashboards

Enables plotting in a web browser

Q&A

Questions and answers

Plotting in Pandas

- We will start by seeing how to make basic plots in pandas from data stored in a pandas dataframe.

- We will then see how to add the following features to our graphs:

    - Change titles/axis titles
    - Annotate the graph
    - Change the window of the graph
    - Customizing axis labels
    - Create multiple plots

- In a future lecture, we will then investigate more advanced plotting tools.

Added
By Eng. Baraa

Sleep Data

We have the following data on how hours of sleep is related to GPA.

```
df_sleep = pd.read_csv("Data/Sleep.csv")

df_sleep.head()
```

	Gender	Age	Year in College	Hours of Sleep	GPA
0	Female	22	4	7.0	3.80
1	Male	18	1	4.0	3.60
2	Male	19	2	9.0	3.50
3	Female	27	3	7.0	3.00
4	Female	37	3	5.0	3.61

Added
By Eng. Baraa

Basic Scatterplot

Makes plot show in Jupyter notebook

Type of plot given as string

```
%matplotlib inline

df_sleep.plot(kind = "scatter", x = "Hours of Sleep",\
              y = "GPA")
```

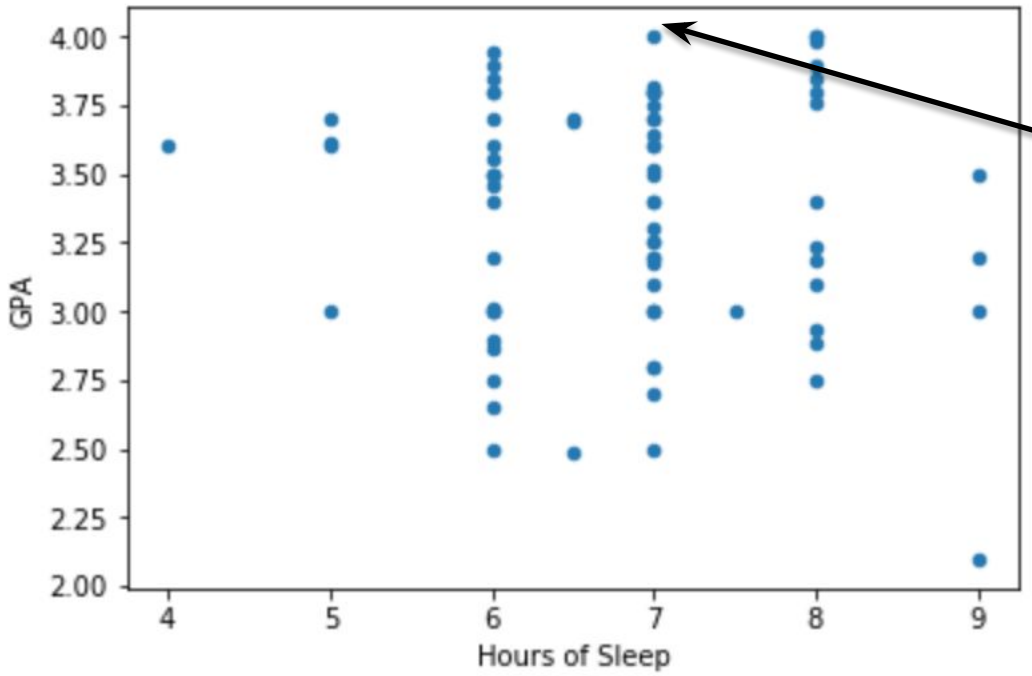
Given x,y axis

Added
By Eng. Baraa

Basic Scatterplot

```
%matplotlib inline

df_sleep.plot(kind = "scatter", x = "Hours of Sleep",\
                        y = "GPA")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c30f080>
```



No title

Axis labels are column names

Added
By Eng. Baraa

Other Types of Plots

pandas.DataFrame.plot

`DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False, layout=None, figsize=None, use_index=True, title=None, grid=None, legend=True, style=None, logx=False, logy=False, loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None, table=False, yerr=None, xerr=None, secondary_y=False, sort_columns=False, **kwds)` [source]

Make plots of DataFrame using matplotlib / pylab.

New in version 0.17.0: Each plot kind has a corresponding method on the `DataFrame.plot` accessor: `df.plot(kind='line')` is equivalent to `df.plot.line()`.

data : *DataFrame*

x : *label or position, default None*

y : *label or position, default None*

Allows plotting of one column versus another

kind : *str*

- 'line' : line plot (default)
- 'bar' : vertical bar plot
- 'barh' : horizontal bar plot
- 'hist' : histogram
- 'box' : boxplot
- 'kde' : Kernel Density Estimation plot
- 'density' : same as 'kde'
- 'area' : area plot
- 'pie' : pie plot
- 'scatter' : scatter plot
- 'hexbin' : hexbin plot

Added
By Eng. Baraa

Basic Barplot

I have computed the average GPA for each. I will show you later on how to do this.

```
df_gpa_gender
```

	Gender	Avg_GPA
0	Female	3.427000
1	Male	3.285175

Added
By Eng. Baraa

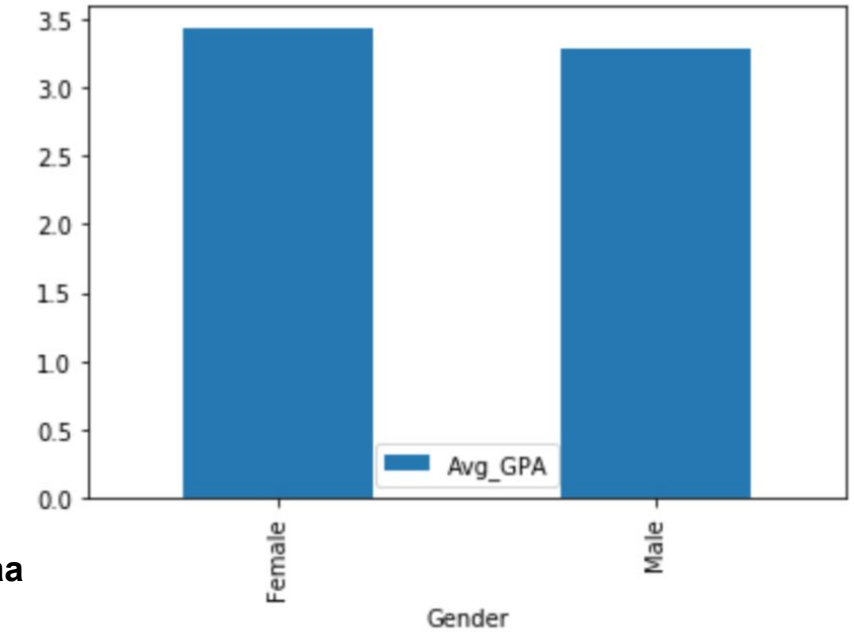
Basic Barplot

```
df_gpa_gender
```

	Gender	Avg_GPA
0	Female	3.427000
1	Male	3.285175

```
df_gpa_gender.plot(kind="bar" , x = "Gender", y = "Avg_GPA")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a6bdfd0>
```

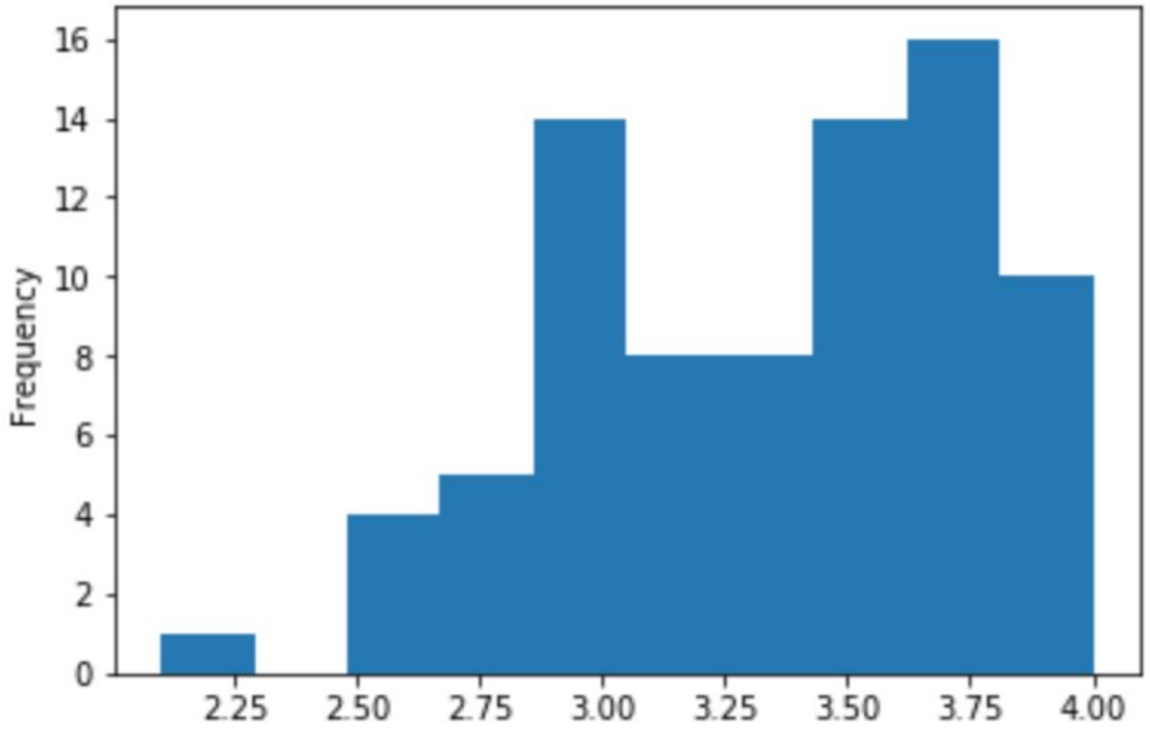


Added
By Eng. Baraa

Basic Histogram

```
df_sleep.GPA.plot(kind="hist")
```

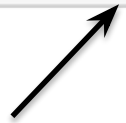
```
<matplotlib.axes._subplots.AxesSubplot at 0x11a7ed748>
```



Added
By Eng. Baraa

Basic Histogram

```
df_sleep.GPA.plot(kind="hist", bins = 20)
```



Number of bins to use

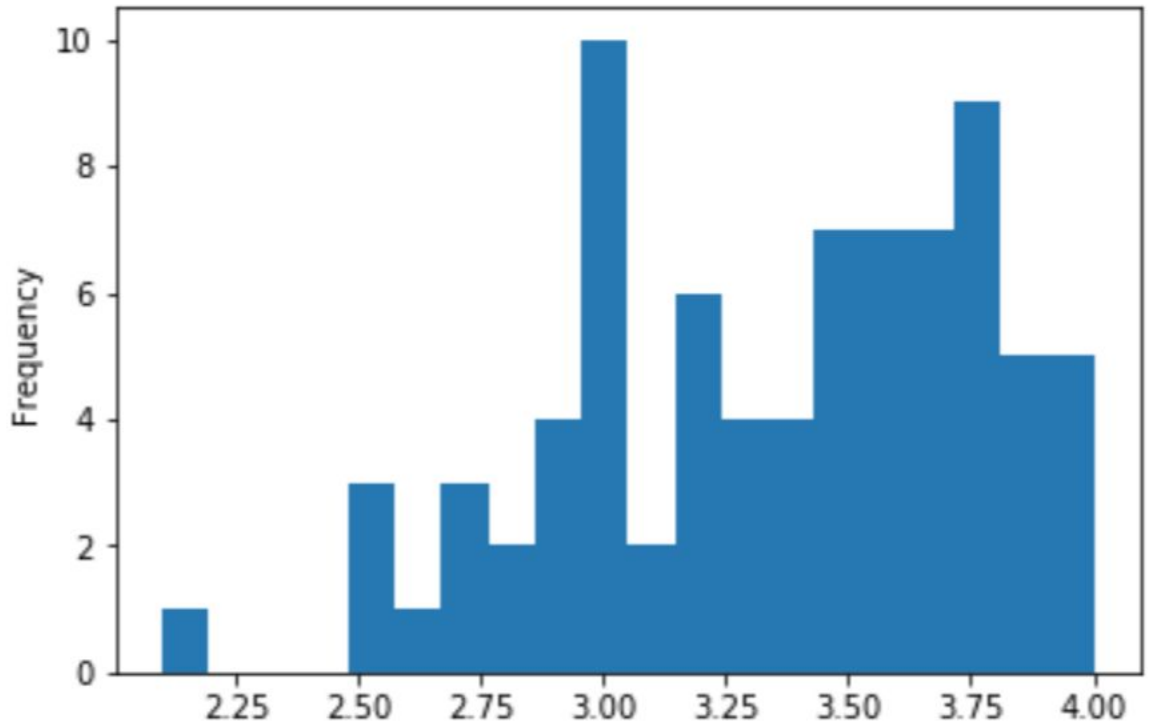
Think of bins in a histogram as baskets at a market – each one catching data points that fall within a certain range.

The better you choose your baskets, the clearer the story your data tells.

Added
By Eng. Baraa

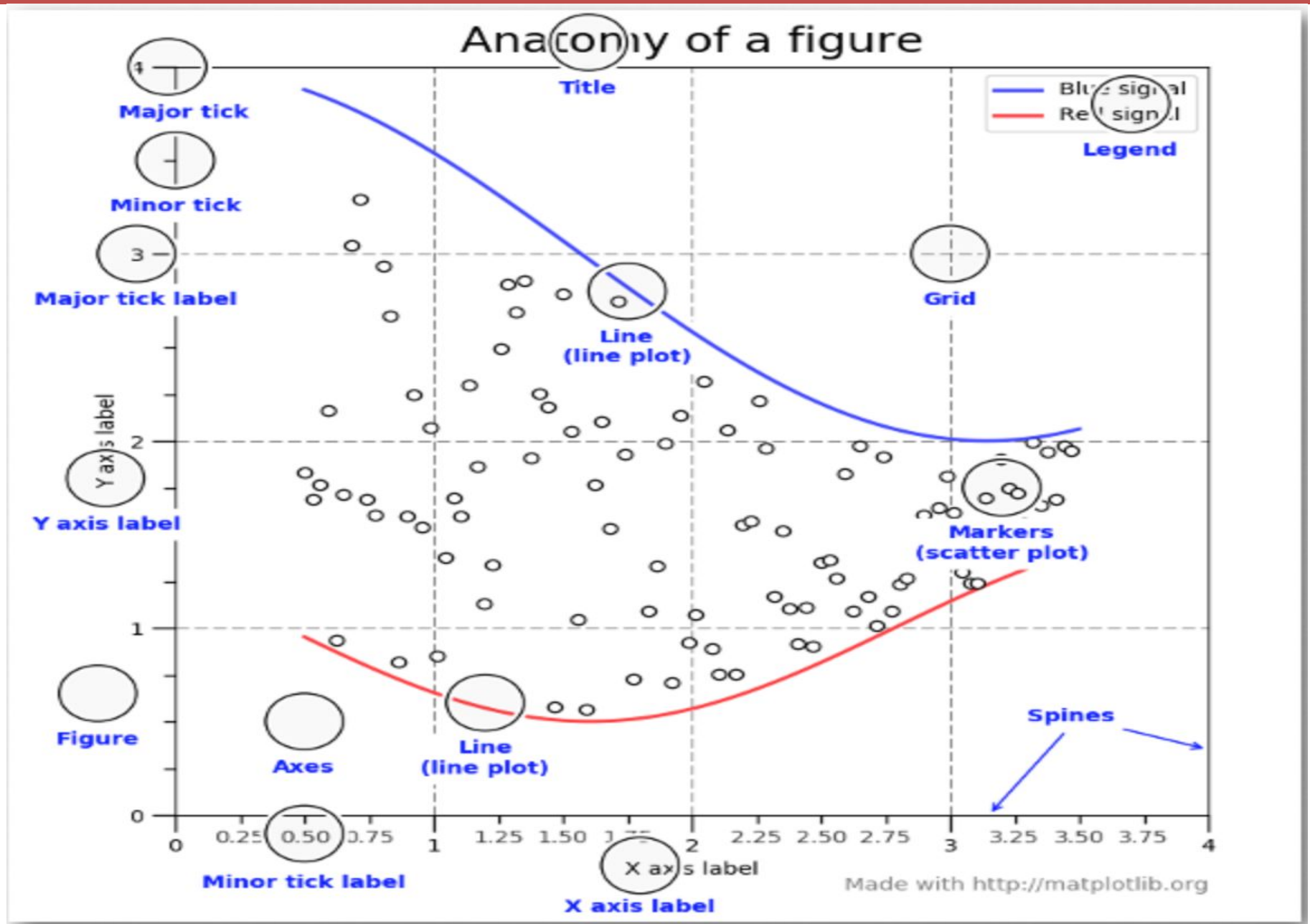
Basic Histogram

```
df_sleep.GPA.plot(kind="hist", bins = 20)
```



Added
By Eng. Baraa

Customizing the Plots



Added
By Eng. Baraa

Sales Data

Consider the data from the top 10 customers:

```
df_top_ten
```

	Name	Sales	Purchases
0	Kulas Inc	137351.96	94
1	White-Trantow	135841.99	86
2	Trantow-Barrows	123381.38	94
3	Jerde-Hilpert	112591.43	89
4	Fritsch, Russel and Anderson	112214.71	81
5	Barton LLC	109438.50	82
6	Will LLC	104437.60	74
7	Koepp Ltd	103660.54	82
8	Frami, Hills and Schmidt	103569.59	72
9	Keeling LLC	100934.30	74

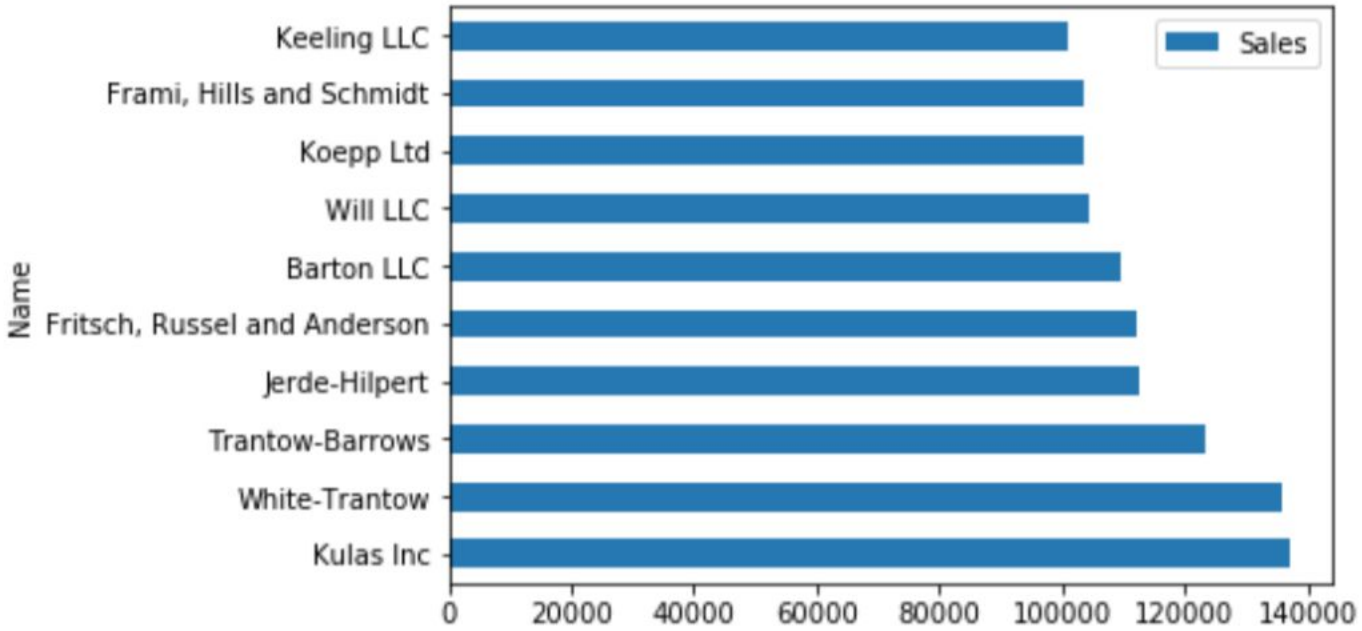
Added
By Eng. Baraa

Sales Data

Let’s use pandas to create a basic bar plot and then see how we can manipulate it with matplotlib.

```
df_top_ten.plot(kind="barh", y = "Sales", x = "Name")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b269cf8>
```



Added
By Eng. Baraa

Introduction to Matplotlib

Introduction to Matplotlib

What is Matplotlib?

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is highly customizable and is widely used for producing high-quality 2D graphics.

It provides a MATLAB-like interface via the pyplot module, making it easy to create plots by calling a series of simple functions.

Key Features of Matplotlib

- **Line Plots**: Basic line plots for showing trends over time.
- **Scatter Plots**: For visualizing the relationship between two numerical variables.
- **Bar Charts**: Displaying categorical data with rectangular bars.
- **Histograms**: Showing the distribution of numerical data.
- **Subplots**: Creating multiple plots within a single figure.
- **Customization**: Highly customizable with titles, labels, legends, colors, and more

Matplotlib

We need the following import statement when we want to use matplotlib

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

We can use matplotlib to change the styling of our plots:

```
#View available styles
print(plt.style.available)
```

```
['bmh', 'classic', 'dark_background', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-bright', 'seaborn-colorblin
d', 'seaborn-dark-palette', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook',
 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whit
egrid', 'seaborn']
```

We can change the style as follows:

```
plt.style.use("ggplot")
```

Added
By Eng. Baraa

Creating Figure + Axis

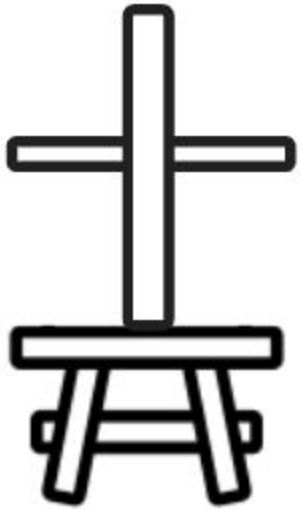
You will place something like this line of code before every plot that you create. Creates the following to elements of the visualization you want to produce:

- The variable fig will be the figure, which represents the entire visualization. A figure could contain multiple plots.
    - Adjust size
    - Number of plots
    - Boarders
- The variable ax represents the axes of a single plot
    - Change axis limit/labels/ticks
    - Add multiple plots
    - Change titles

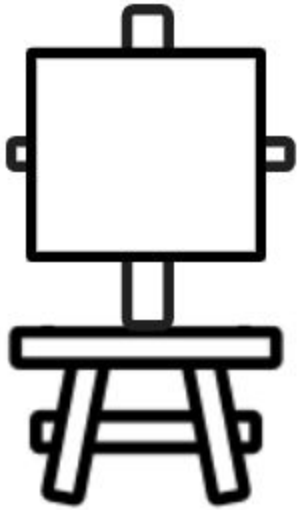
```
#Create figure and axes
fig,ax = plt.subplots()
```

Added
By Eng. Baraa

The stages of `plt.plot()`



*Creates a **figure**: Object that houses one or more plot (axes)*



*Creates an **axes**: Object that houses plot*

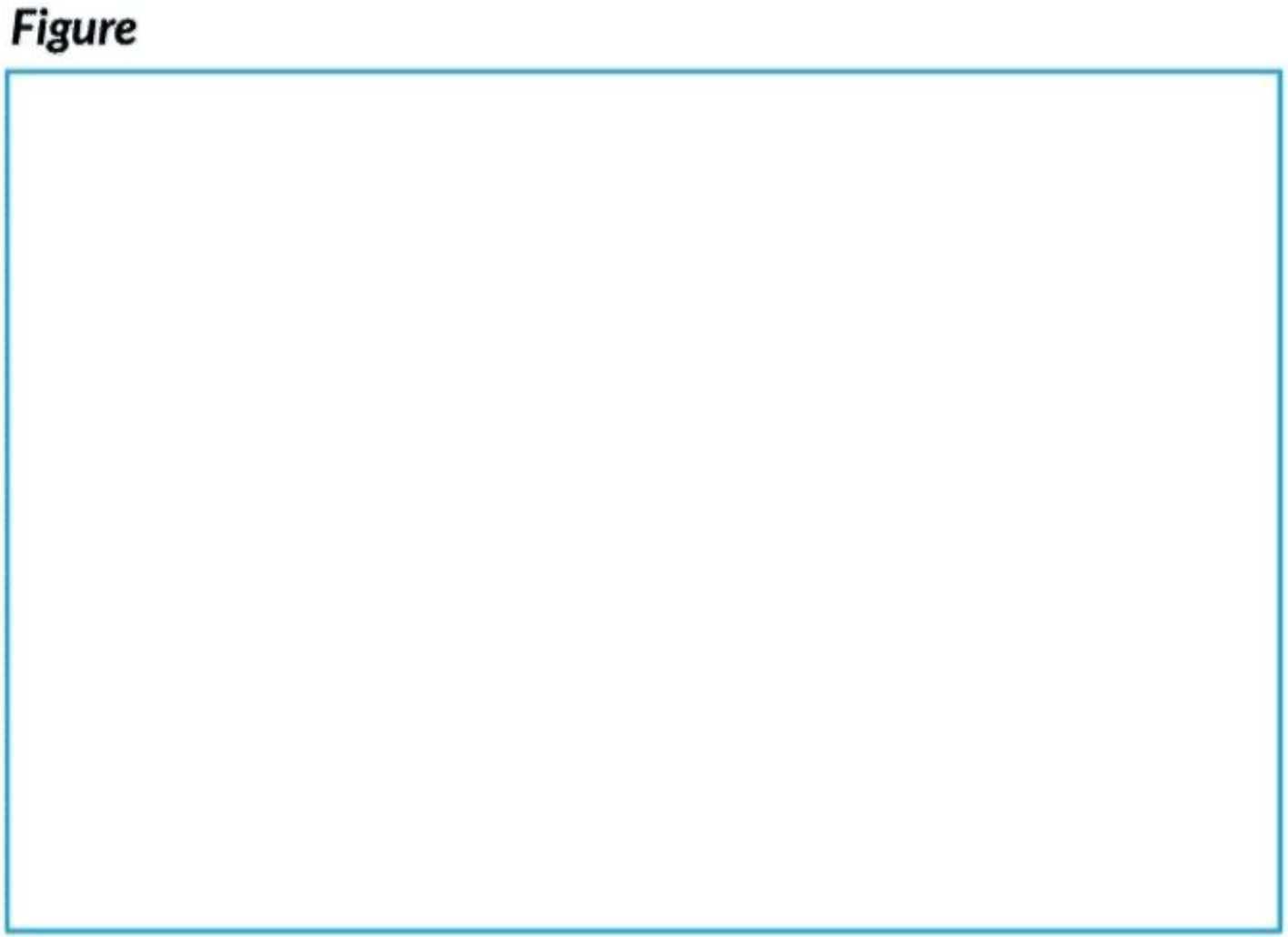
*Creates an **everything** inside of axes (plot)*

Added
By Eng. Baraa

Data Visualization Basics I: *The anatomy of a* `matplotlib` *figure*

When using a simple `matplotlib` or `seaborn` function like `plt.plot(my_data)`, what `matplotlib` is doing is creating three nested objects in the background.

- The **figure** object, which could be considered as the canvas-holder, or an object containing all possible axes (plots).
  - The **axes** object(s), which could be considered as the canvas(es), or the plot where we will be adding our visualizations.
    - Everything that makes up the plot.



Added
By Eng. Baraa

Data Visualization Basics II: *Subplots and multiple axes*

One of the defining features of creating visualizations with `matplotlib`, is the great level of control offered to create multiple plots at once. As discussed earlier, generating a plot neccessitates the creation of 3 hierarchical objects:

- The **figure** object, which could be considered as the canvas-holder, or an object containing all possible axes (plots).
    - The **axes** object(s), which could be considered as the canvas(es), or the plot where we will be adding our visualizations.
        - Everything that makes up the plot.

Creating a figure with multiple subplots (axes) is easy, and can be referenced with the following:

```
figure, axes = plt.subplot(nrows = , ncols = , figsize = ...)
```

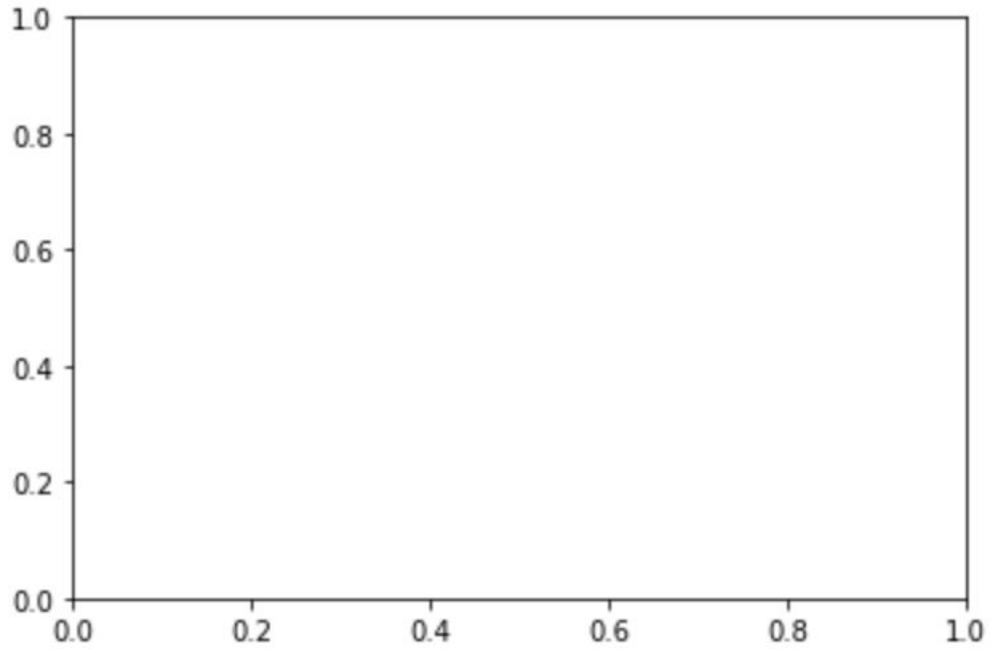
Where we generate a `figure` object and its `axes`. A `figure` can be divided into a grid of rows and columns, where each "cell" in our rows and columns is an axes and we can plot something inside of it.

Figure

Added
By Eng. Baraa

Creating Figure + Axis

```
#Create fig with blank axis
fig, ax = plt.subplots()
```

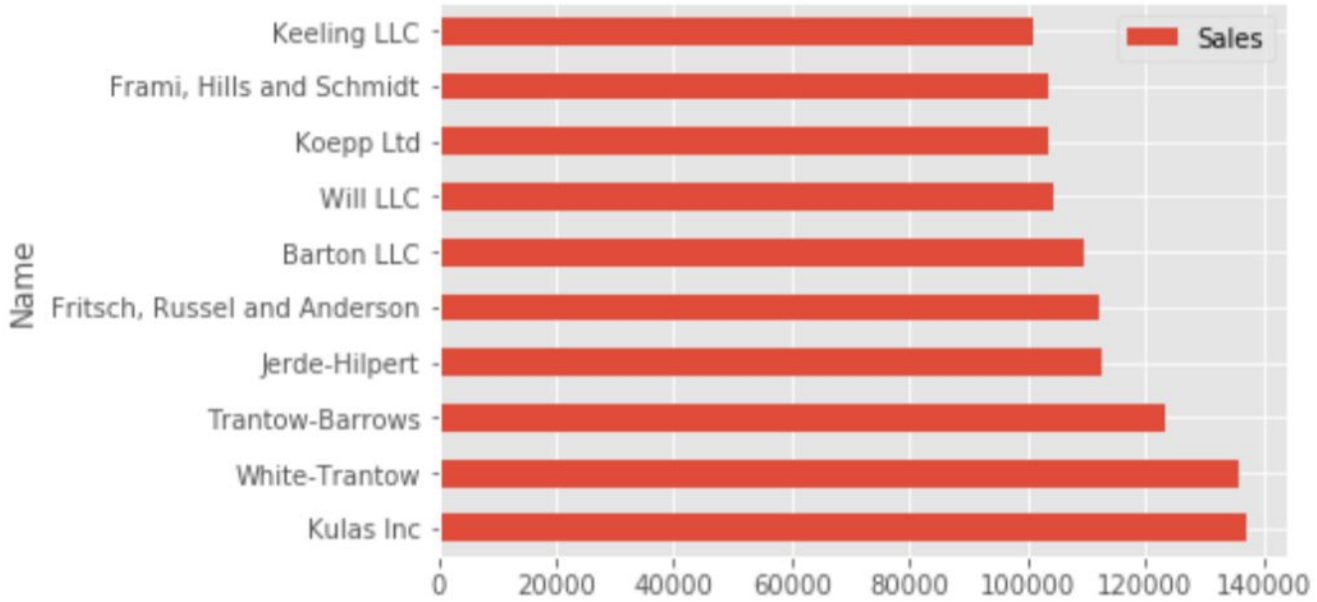


Added
By Eng. Baraa

Adding Barplot to Figure

```
#Create figure and axes
fig,ax = plt.subplots()
#Add plot to figure
df_top_ten.plot(kind="barh", y = "Sales", x = "Name", ax=ax )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b607b00>
```

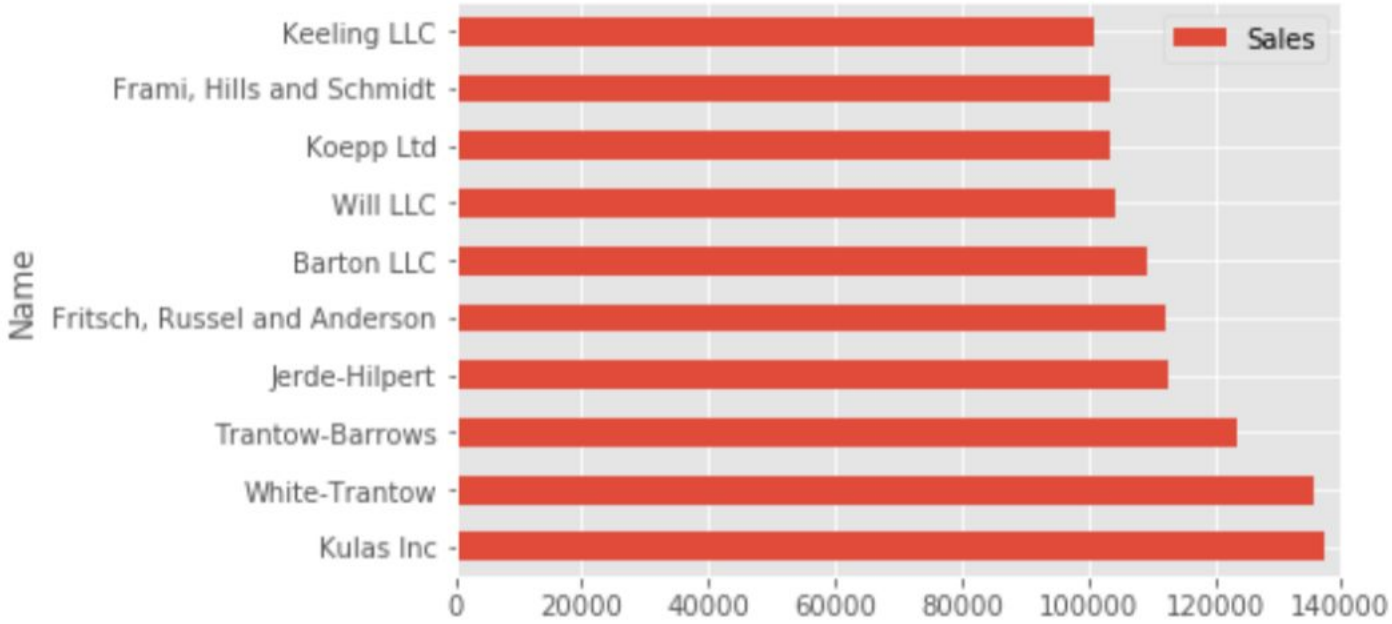


Same plot, but now we can change the plot using attributes of the variables fig and ax.

Added
By Eng. Baraa

Changing Features of Axes

```
#Change x limit
ax.set_xlim([0,140000])

#View updated plot
fig
```

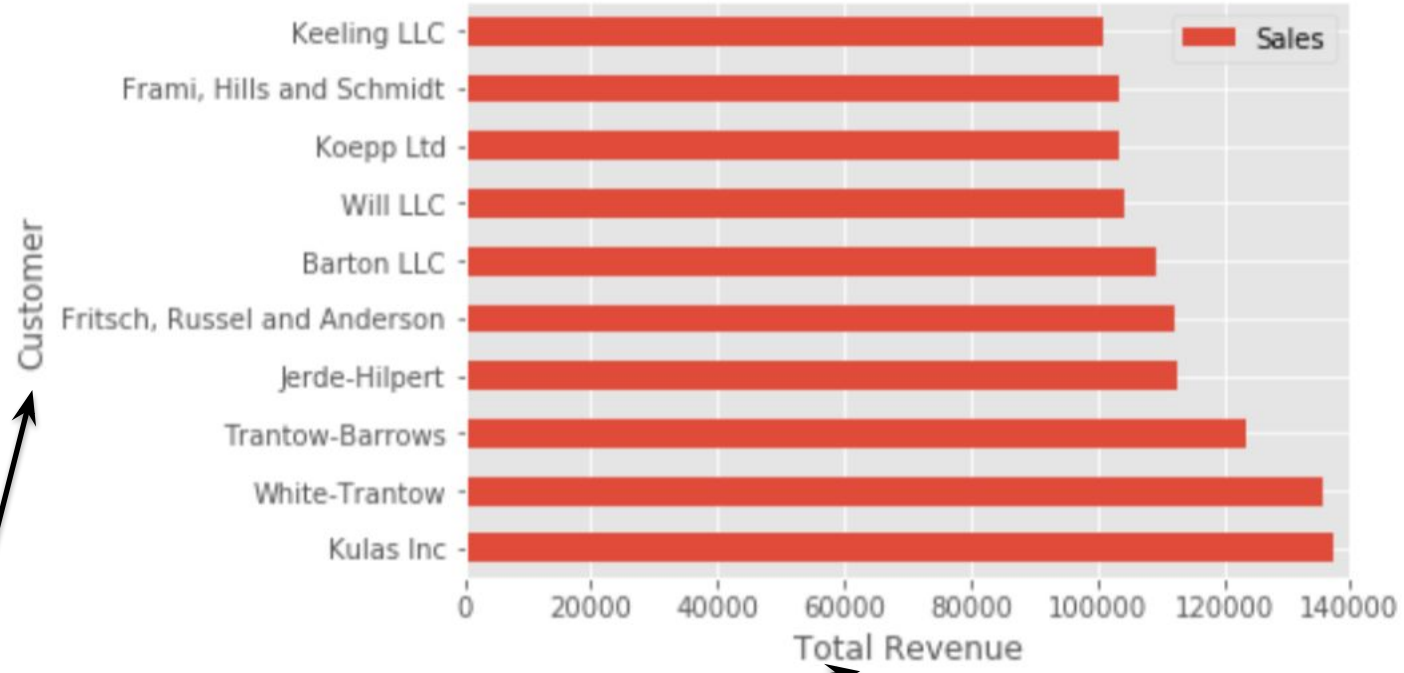


x-axis only goes up to 140000

Added
By Eng. Baraa

Changing Features of Axes

```
#Set axis labels
ax.set_xlabel("Total Revenue")
ax.set_ylabel("Customer")

fig
```



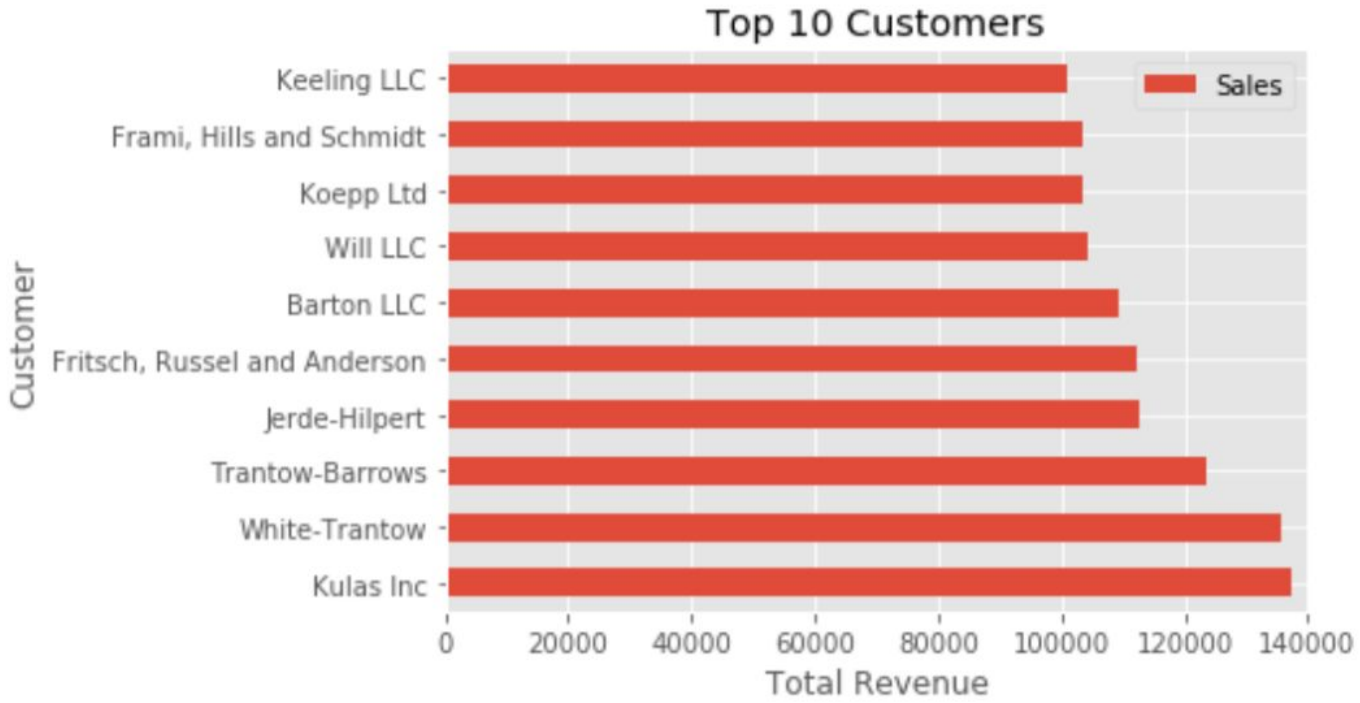
New axis labels

Added
By Eng. Baraa

Changing Features of Axes

```
#Adding labels all at once
ax.set(title = "Top 10 Customers", xlabel="Total Revenue",\
       ylabel = "Customer")

fig
```

Title



Added
By Eng. Baraa

Change Figure Size

Give list of length of length and width in inches

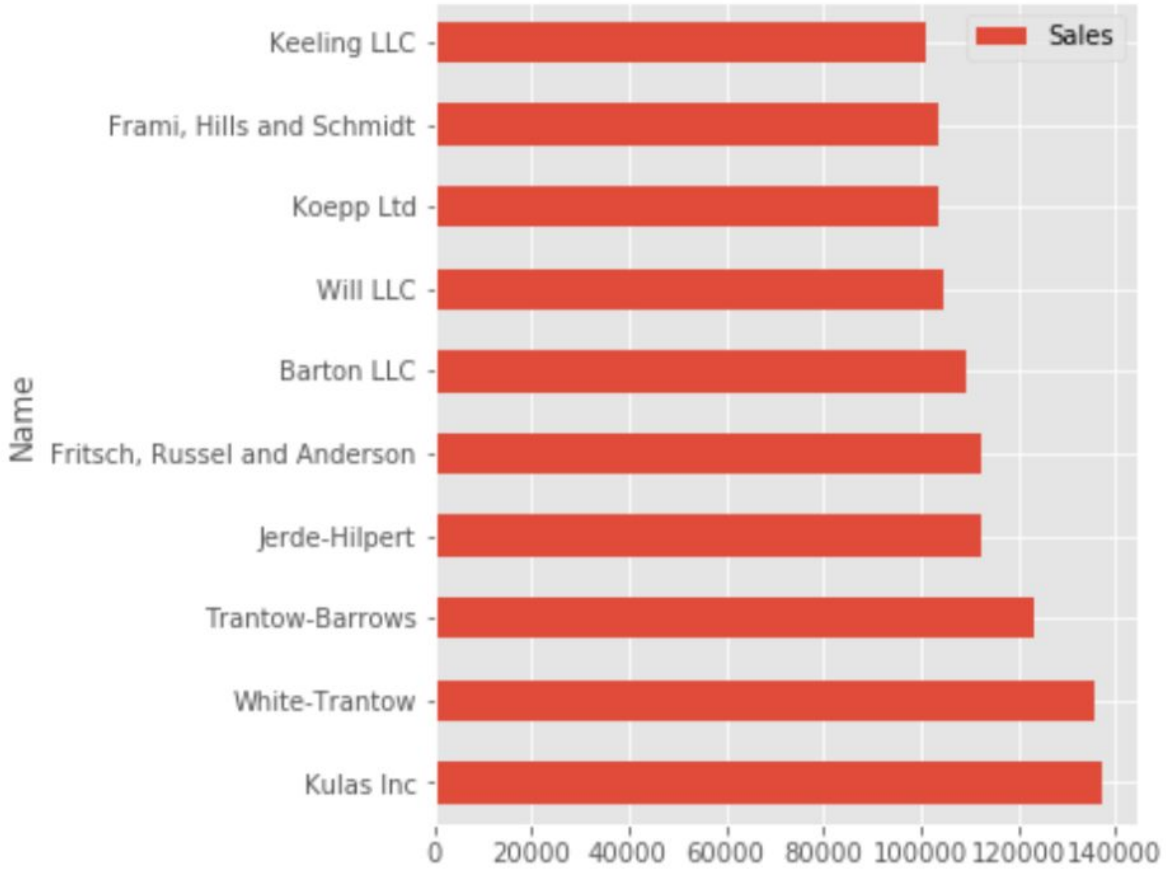
```
#Change figure size
fig,ax = plt.subplots(figsize = [5,6])
#Add plot to figure
df_top_ten.plot(kind="barh", y = "Sales", x = "Name", ax=ax )
```

Added
By Eng. Baraa

Change Figure Size

```
#Change figure size
fig,ax = plt.subplots(figsize = [5,6])
#Add plot to figure
df_top_ten.plot(kind="barh", y = "Sales", x = "Name", ax=ax )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11bc2b748>
```



Added
By Eng. Baraa

Move Legend

```
#Move Legend
ax.legend(loc = 5)
#View updated figure
fig
```

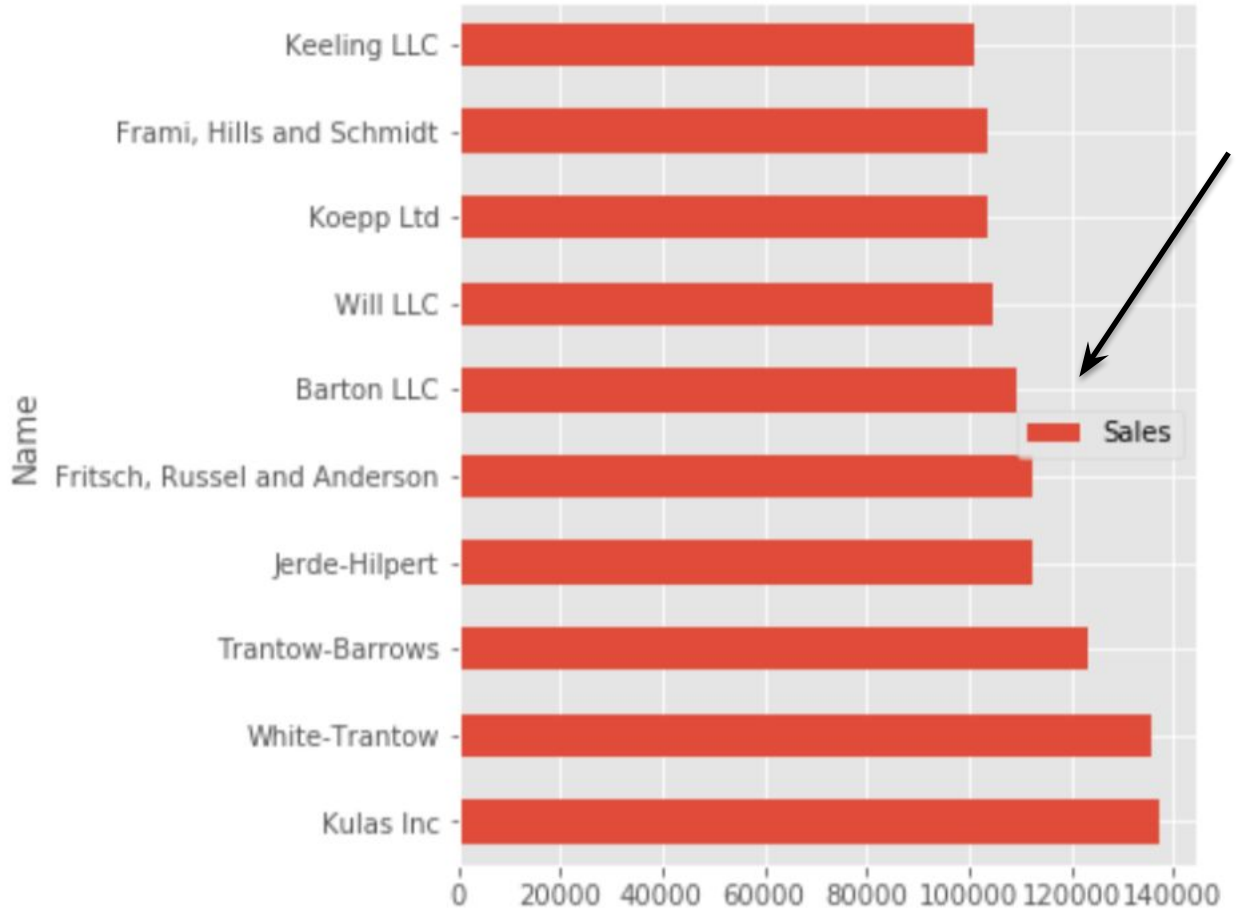
loc parameter moves legend

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Added
By Eng. Baraa

Move Legend

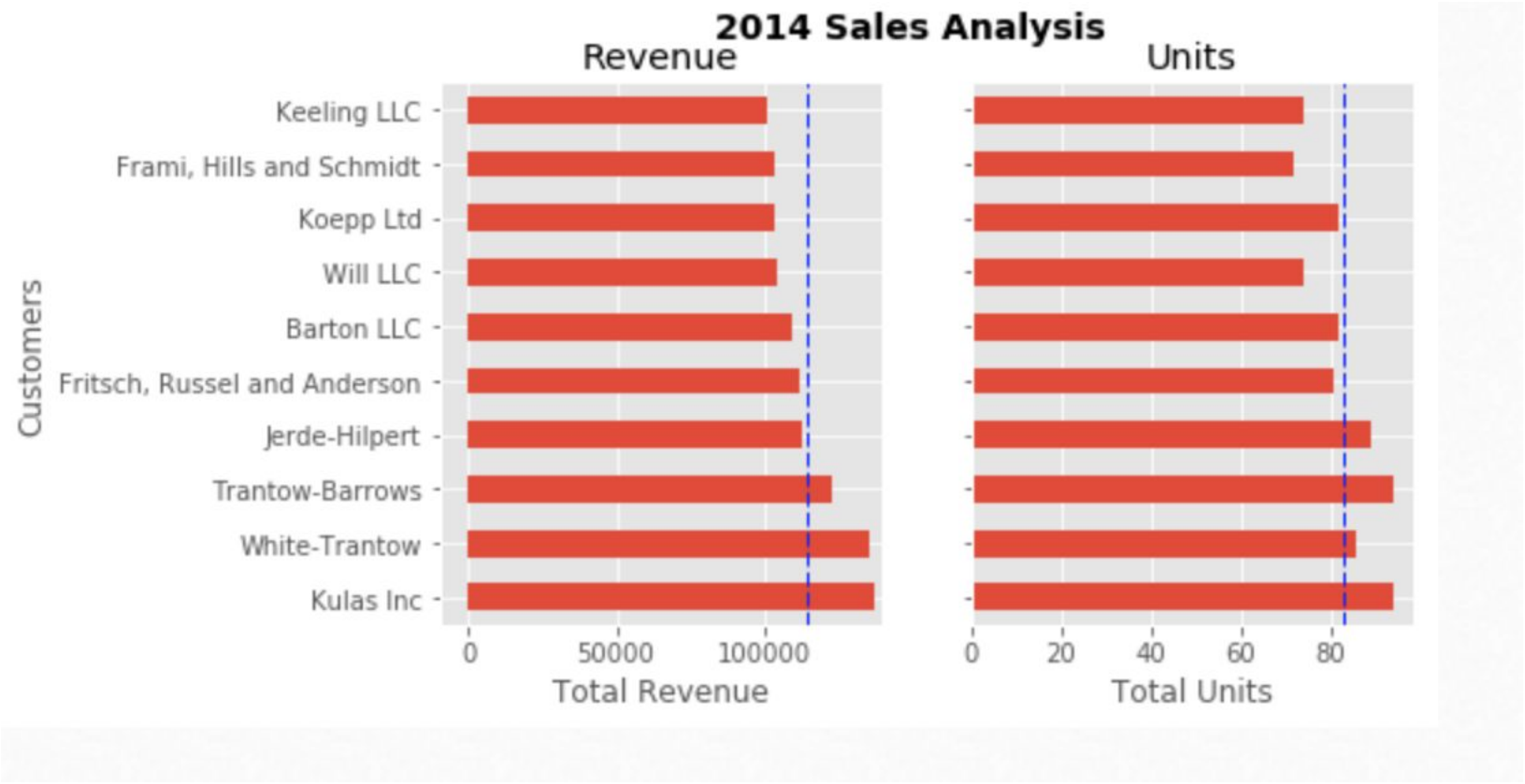
```
#Move Legend
ax.legend(loc = 5)
#View updated figure
fig
```



New legend placement

Added
By Eng. Baraa

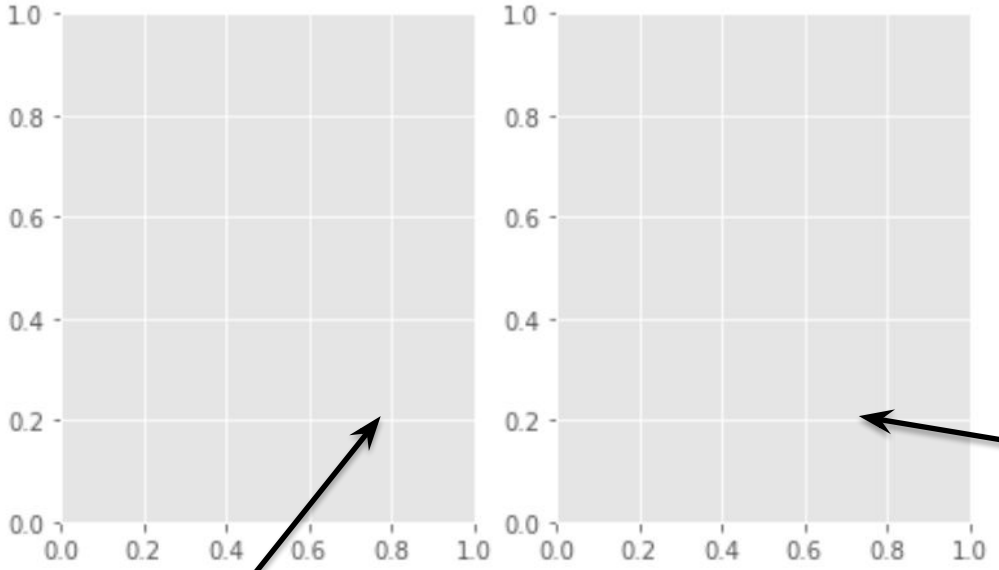
Creating Multiple Plots



Added
By Eng. Baraa

Creating Multiple Plots

```
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, figsize=(7, 4))
```



7x4 Figure, with 1 row and 2 columns of subplots

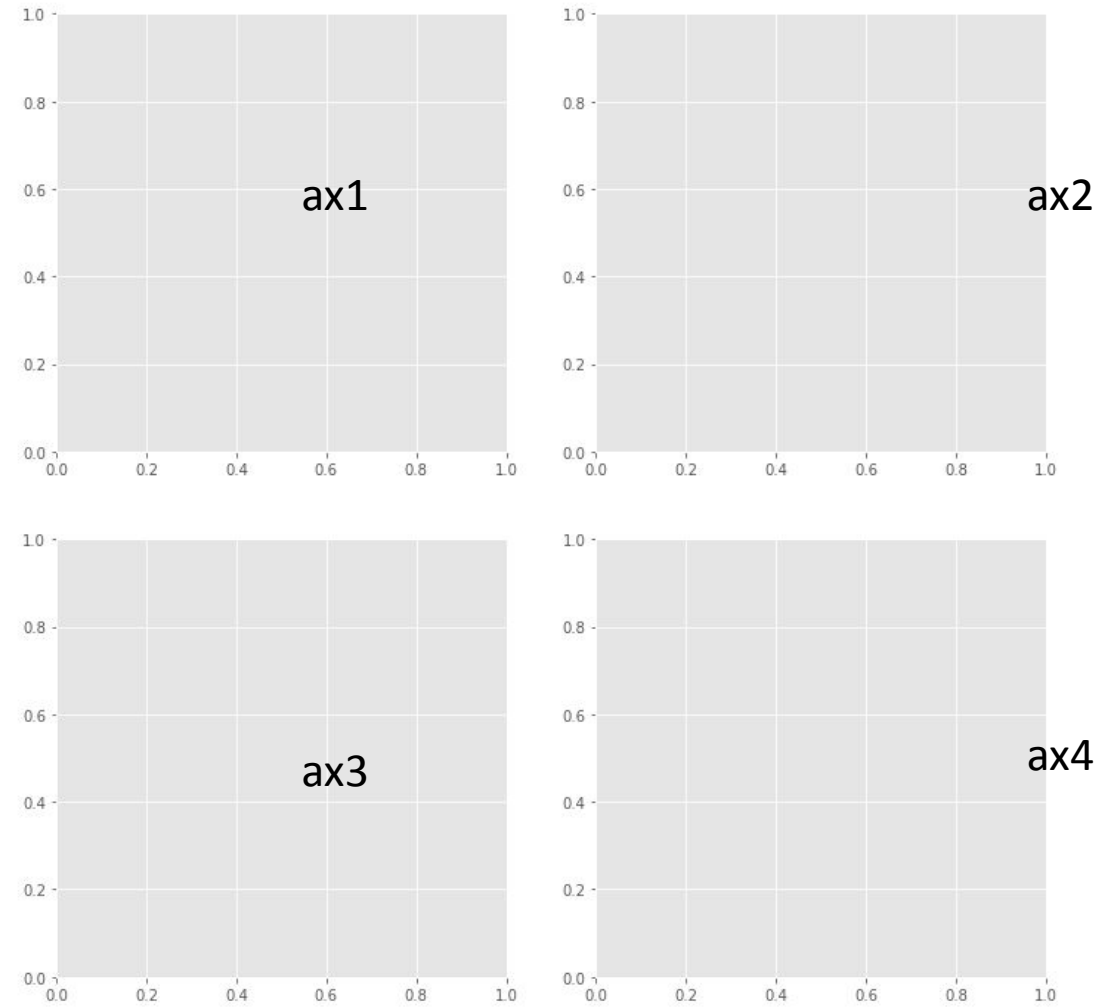
ax1

ax0

Creating Multiple Plots

List of lists

```
fig,[[ax1,ax2],[ax3,ax4]] = plt.subplots(nrows=2, ncols=2, figsize = (12,12))
```



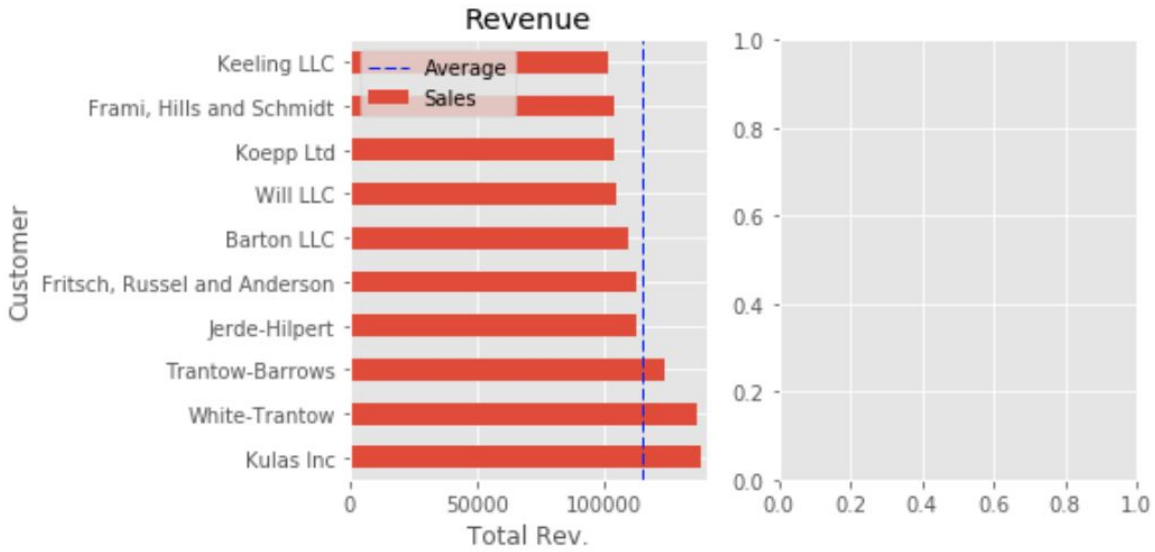
12x12 Figure, with 2 rows and 2 columns of subplots

Added By Eng. Baraa

Creating Multiple Plots

```
#Make first bar plot
df_top_ten.plot(kind="barh", x = "Name", y = "Sales", ax = ax0)
ax0.set(title = "Revenue", xlabel="Total Rev.", ylabel="Customer")
ax0.set_xlim([0,140000])
avg=df_top_ten.Sales.mean()
ax0.axvline(x = avg, color='b', label="Average", linestyle='--', linewidth=1)
ax0.legend(loc=0)
```

```
<matplotlib.legend.Legend at 0x118d6e438>
```

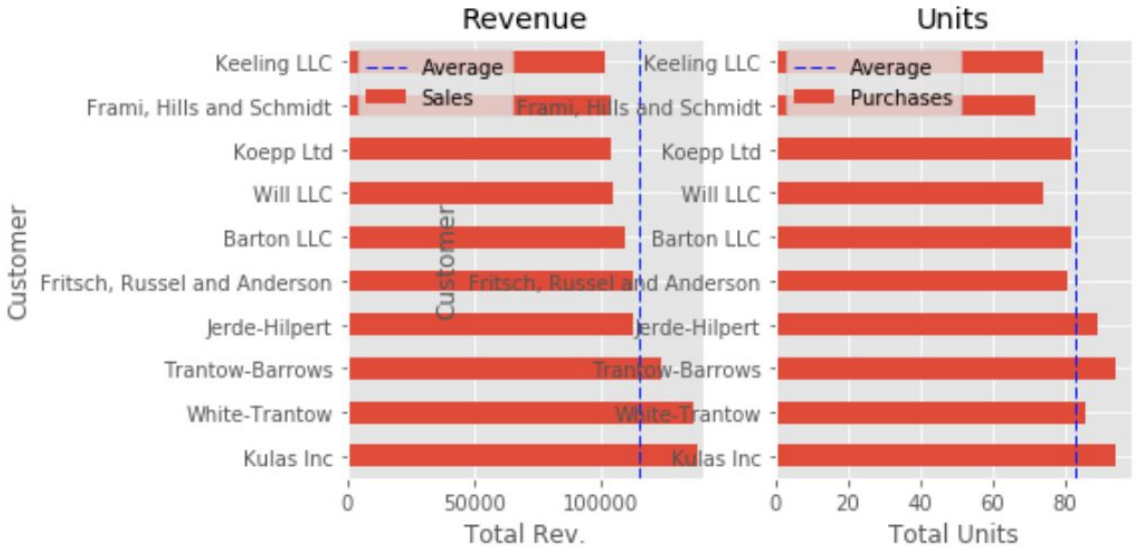


Notice that all references are made to ax0.

Added By Eng. Baraa

Creating Multiple Plots

```
#Make second bar plot
df_top_ten.plot(kind="barh", x = "Name", y = "Purchases", ax = ax1)
ax1.set(title = "Units", xlabel="Total Units", ylabel="Customer")
avg=df_top_ten.Purchases.mean()
ax1.axvline(x = avg, color='b', label="Average", linestyle='--', linewidth=1)
ax1.legend(loc=0)
fig
```



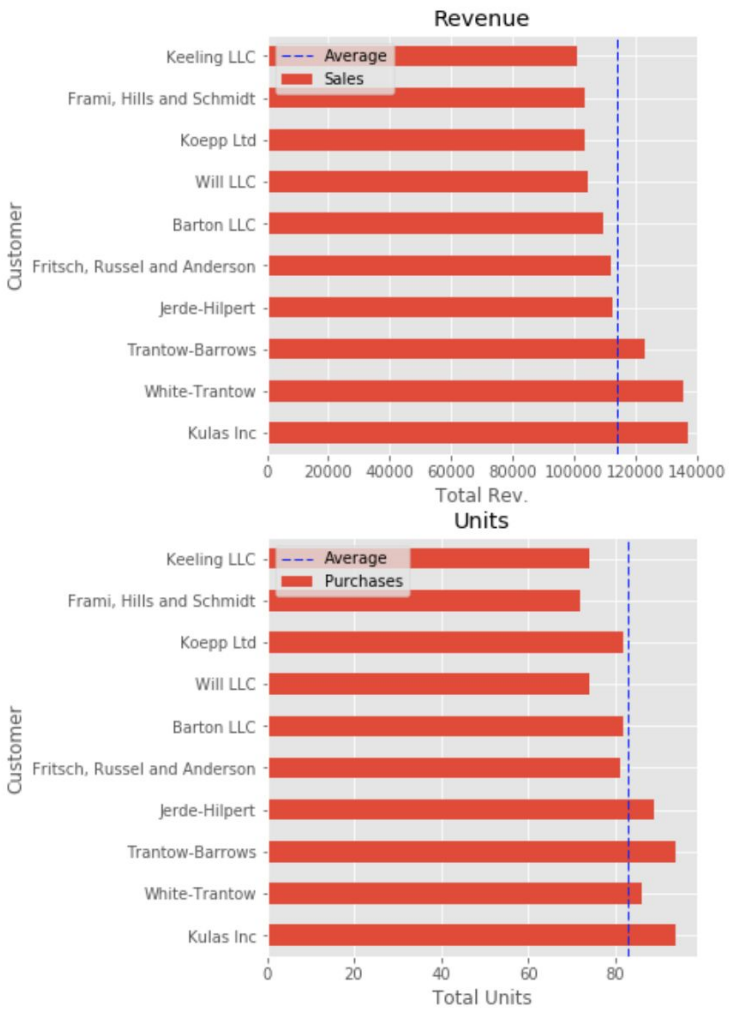
Uh oh! Plots are overlapping. Let's two options for fixing this.

Added By Eng. Baraa

Creating Multiple Plots

```
#Option 1 - Stack them
fig, (ax0, ax1) = plt.subplots(nrows=2, ncols =1, figsize=(5, 11))
```

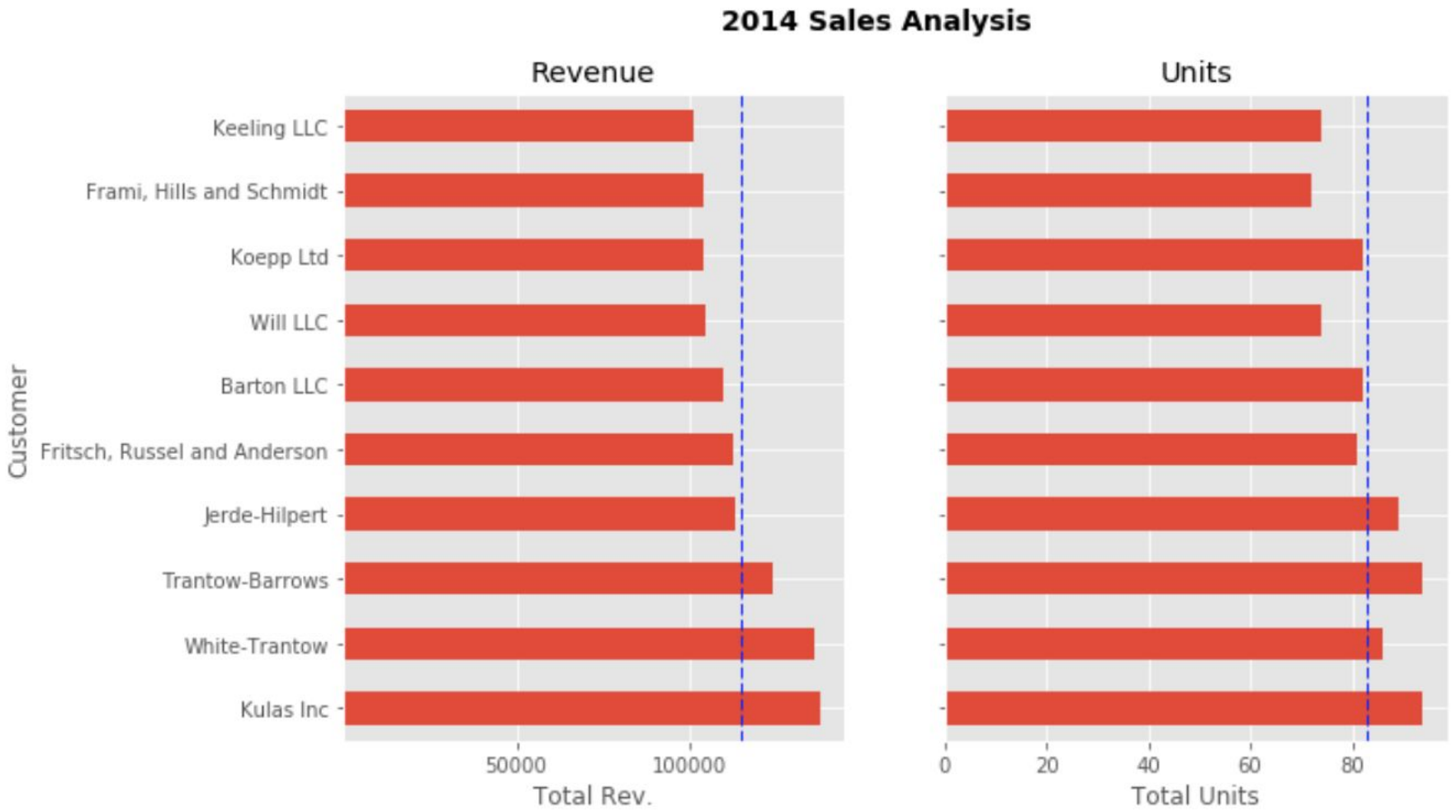
updated



Added
By Eng. Baraa

Creating Multiple Plots

```
#Option 2  - Have them share y axis
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(10, 6))
```



Added
By Eng. Baraa

```
#Option 2  - Have them share y axis
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(10, 6))

#Make first bar plot
df_top_ten.plot(kind="barh", x = "Name", y = "Sales", ax = ax0)
ax0.set(title = "Revenue", xlabel="Total Rev.", ylabel="Customer")
ax0.set_xticks([50000,100000])
avg=df_top_ten.Sales.mean()
ax0.axvline(x = avg, color='b', label="Average",\
            linestyle='--', linewidth=1)
ax0.legend(loc=0)

#Make second bar plot
df_top_ten.plot(kind="barh", x = "Name", y = "Purchases", ax = ax1)
ax1.set(title = "Units", xlabel="Total Units", ylabel="Customer")
avg=df_top_ten.Purchases.mean()
ax1.axvline(x = avg, color='b', label="Average", linestyle='--', linewidth=1)
ax1.legend(loc=0)

# Title the figure
fig.suptitle('2014 Sales Analysis', fontsize=14, fontweight='bold');

# Hide the legends
ax1.legend().set_visible(False)
ax0.legend().set_visible(False)
```

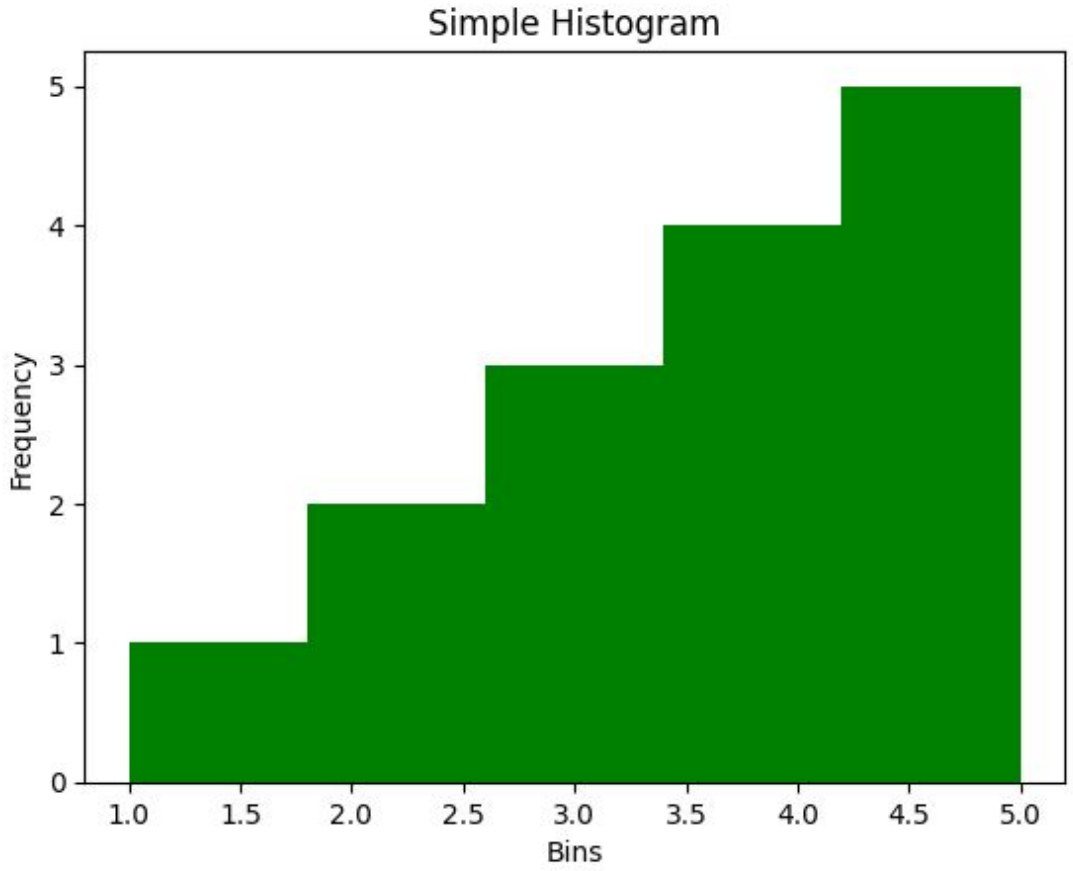
Added
By Eng. Baraa

Basic Plotting with Matplotlib

Histogram

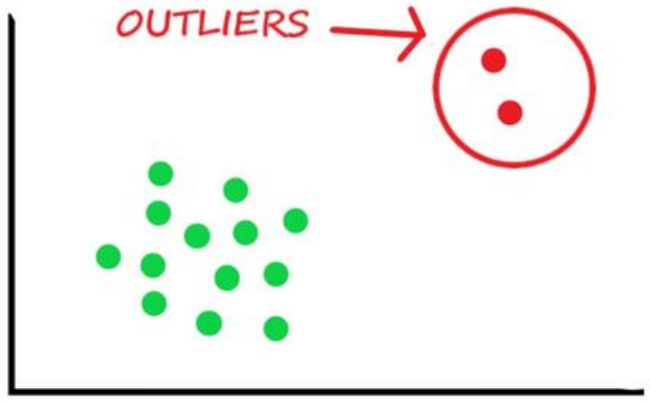
- A histogram is used to show the distribution of a numerical variable. plt.hist() creates a histogram with a specified number of bins.

```
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

plt.hist(data, bins=5, color='green')
plt.title("Simple Histogram")
plt.xlabel("Bins")
plt.ylabel("Frequency")
plt.show()
```



Outliers

Outliers are data points that significantly deviate from the rest of the data in a dataset. These anomalies can occur due to inconsistencies in measurement, errors during data collection, or actual variations. Outliers have the potential to distort statistical results and negatively influence machine learning models. Hence, it's crucial to detect and manage them effectively.



Identifying Outliers

- **Visualization Methods**: Techniques such as box plots, scatter plots, histograms, and density plots are valuable tools for detecting outliers visually.
- **Box Plot**: Displays data distribution, highlighting outliers as individual points outside the whiskers.
- **Scatter Plot**: Shows relationships between variables, where outliers can be seen as points far from the main cluster.
- **Histogram**: Illustrates the frequency of data points, with outliers appearing as bars that are distant from the rest.
- **Density Plot**: Provides a smooth curve of the data distribution, with outliers showing up as irregular bumps outside the main peak.

Introduction to Seaborn

Introduction to Seaborn

What is Seaborn?

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn makes it easy to create complex visualizations, including relationships, categorical data, and statistical plots.

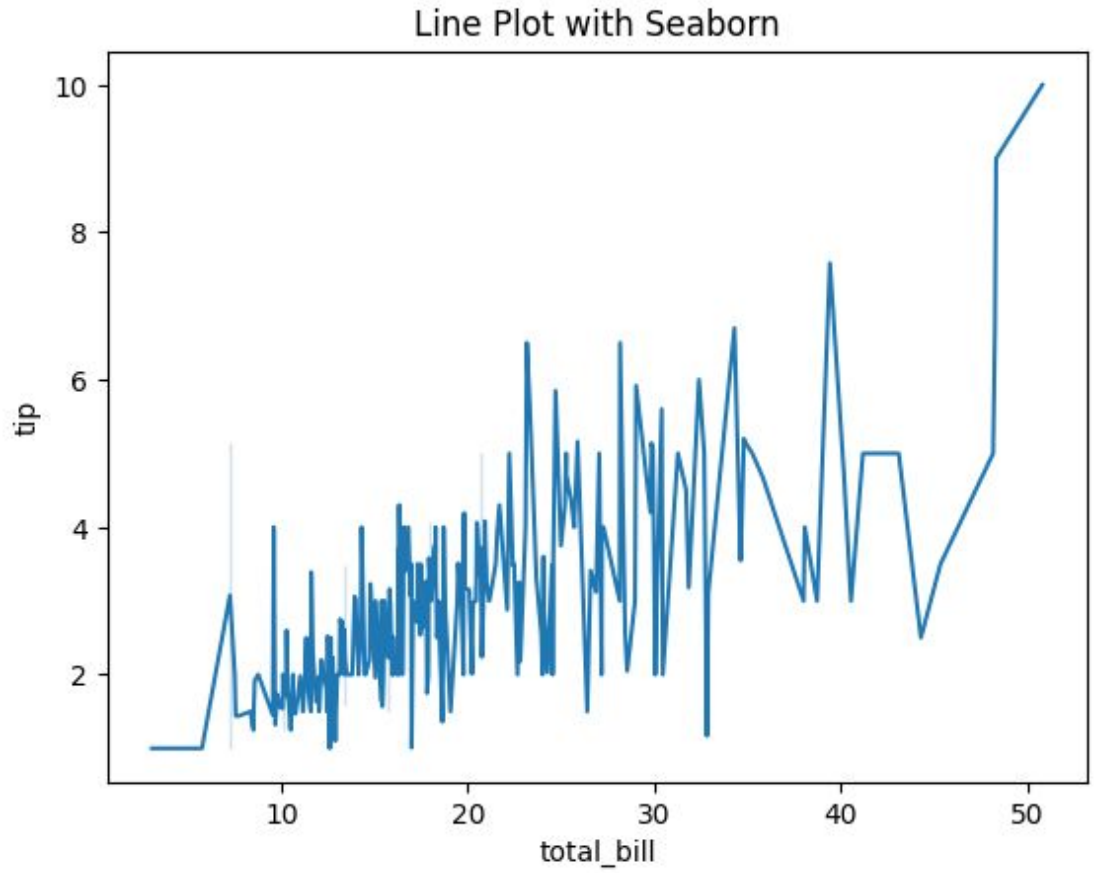
Key Features of Seaborn

- **Visualizing Distributions**: Histograms, KDE plots, and rug plots.
- **Visualizing Categorical Data**: Bar plots, count plots, box plots, and violin plots.
- **Visualizing Relationships**: Scatter plots, line plots, and pair plots.
- **Heatmaps**: For showing data correlations.
- **Themes**: Built-in themes for better-looking plots.

Basic Plotting with Seaborn

Line Plot

```
import seaborn as sns
import matplotlib.pyplot as plt

data = sns.load_dataset('tips')
sns.lineplot(x='total_bill', y='tip', data=data)
plt.title("Line Plot with Seaborn")
plt.show()
```



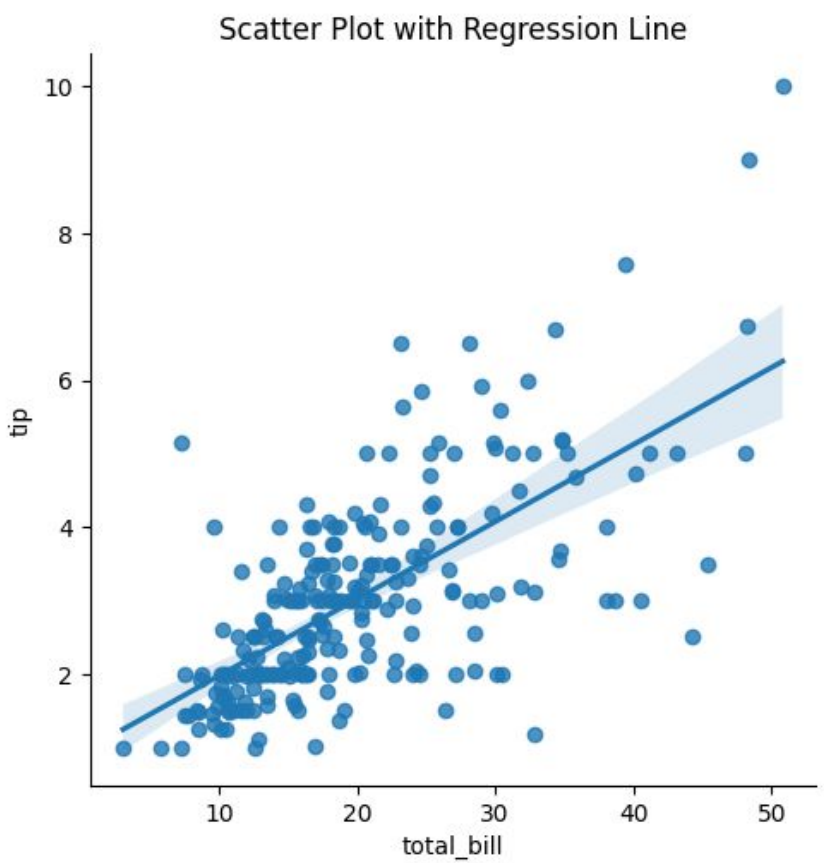
This uses Seaborn's lineplot() to create a line plot using the 'tips' dataset, showing the relationship between total bill and tip.

Basic Plotting with Seaborn

Scatter Plot with Regression Line

- Seaborn's lmplot() creates a scatter plot with a linear regression line, illustrating the trend between total bill and tip.

```
sns.lmplot(x='total_bill', y='tip', data=data)
plt.title("Scatter Plot with Regression Line")
plt.show()
```

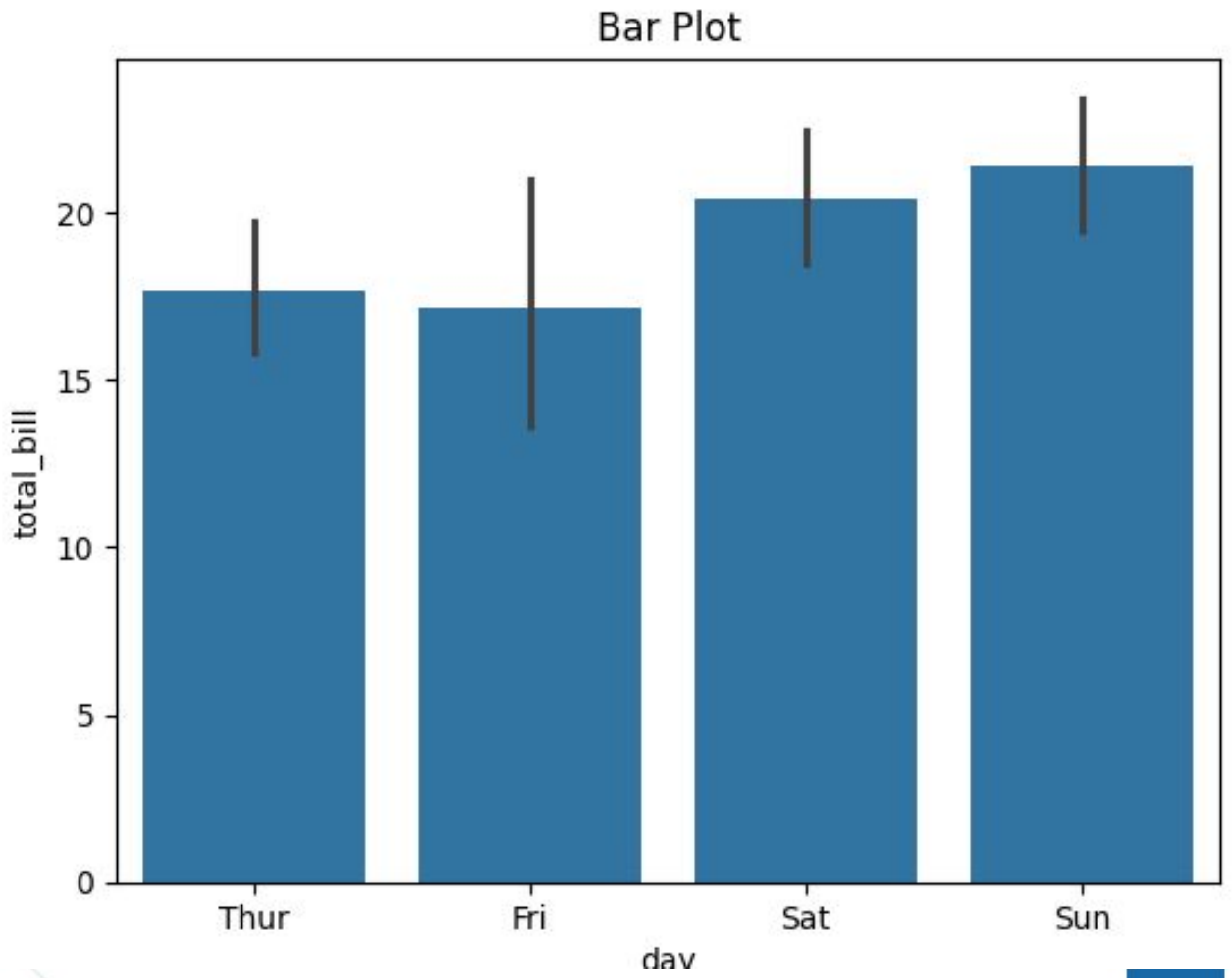


Basic Plotting with Seaborn

Bar Plot

- barplot() creates a bar chart showing the average total bill for each day of the week. It aggregates the data automatically.

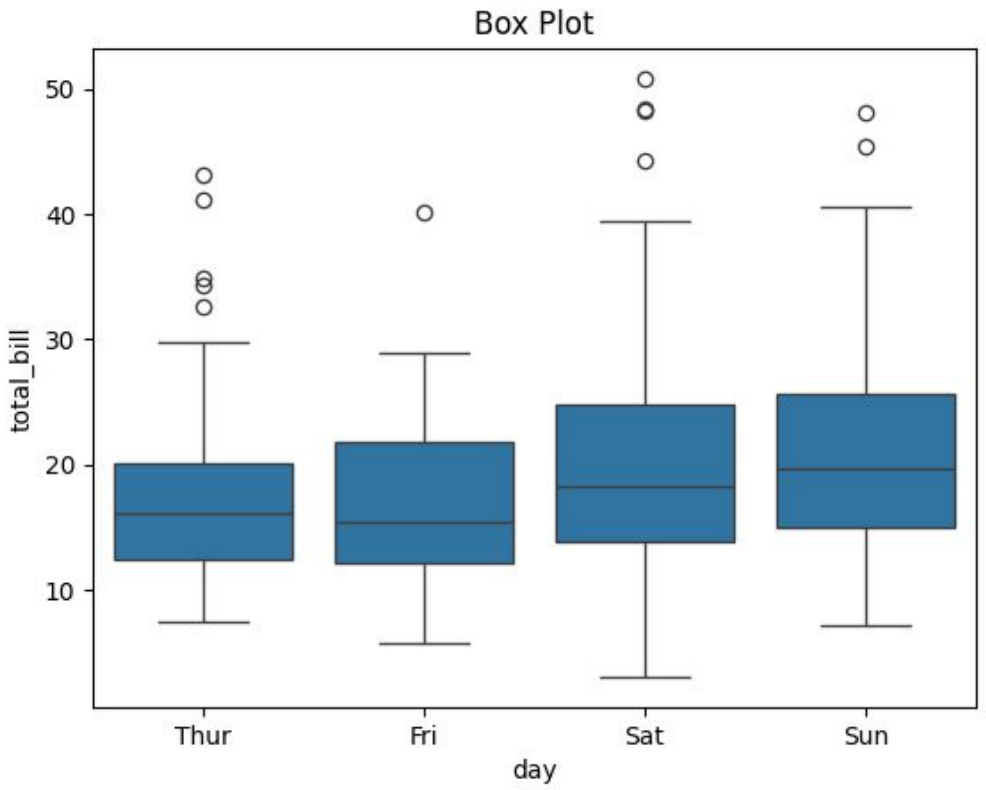
```
sns.barplot(x='day', y='total_bill', data=data)
plt.title("Bar Plot")
plt.show()
```



Basic Plotting with Seaborn

Box Plot

```
sns.boxplot(x='day', y='total_bill', data=data)
plt.title("Box Plot")
plt.show()
```

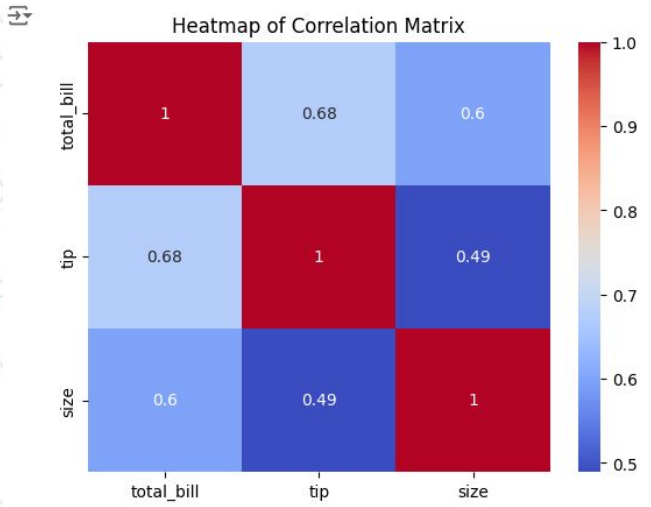


- boxplot() is used to show the distribution of total bills across different days, highlighting the median, quartiles, and potential outliers.

Basic Plotting with Seaborn

Heatmap

```
# Load a dataset using Seaborn
data = sns.load_dataset('tips')  # 'tips' is a built-in dataset in Seaborn

# Select only the numerical columns
numerical_data = data.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
corr = numerical_data.corr()

# Plot the heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Heatmap of Correlation Matrix")
plt.show()
```



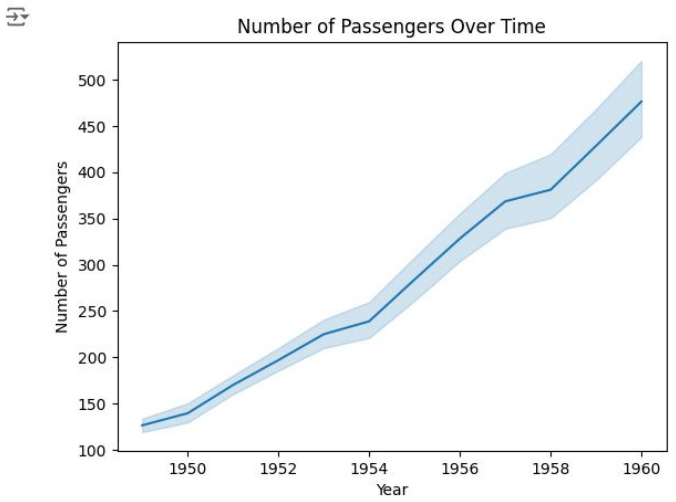
- heatmap() visualizes the correlation matrix of the dataset, showing relationships between variables using color gradients.

Practical Visualization Examples

Practical Visualization Examples

Line Plot to Show Trends Over Time

```
[14] import seaborn as sns
     import matplotlib.pyplot as plt

     # Load the 'flights' dataset
     data = sns.load_dataset('flights')

     # Line plot showing the number of passengers over time
     sns.lineplot(x='year', y='passengers', data=data)
     plt.title("Number of Passengers Over Time")
     plt.xlabel("Year")
     plt.ylabel("Number of Passengers")
     plt.show()
```

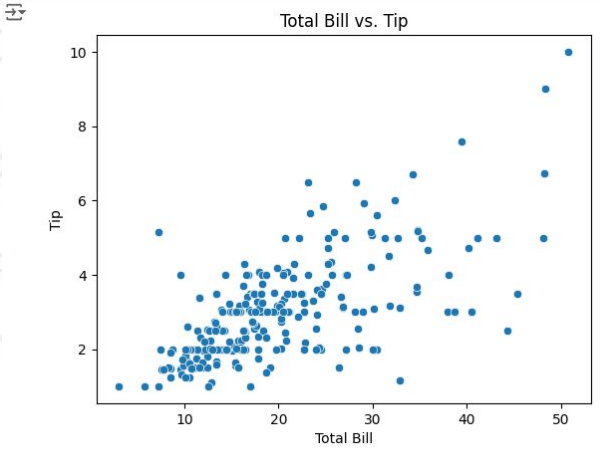


- This line plot shows the trend in the number of passengers over time using the 'flights' dataset. It helps to identify overall growth or decline trends.

Practical Visualization Examples

Scatter Plot to Visualize Relationships Between Two Variables

```
# Load the 'tips' dataset
data = sns.load_dataset('tips')

# Scatter plot showing the relationship between total bill and tip
sns.scatterplot(x='total_bill', y='tip', data=data)
plt.title("Total Bill vs. Tip")
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.show()
```



- A scatter plot shows the relationship between the total bill and the tip amount in the 'tips' dataset. It helps in identifying patterns or correlations.

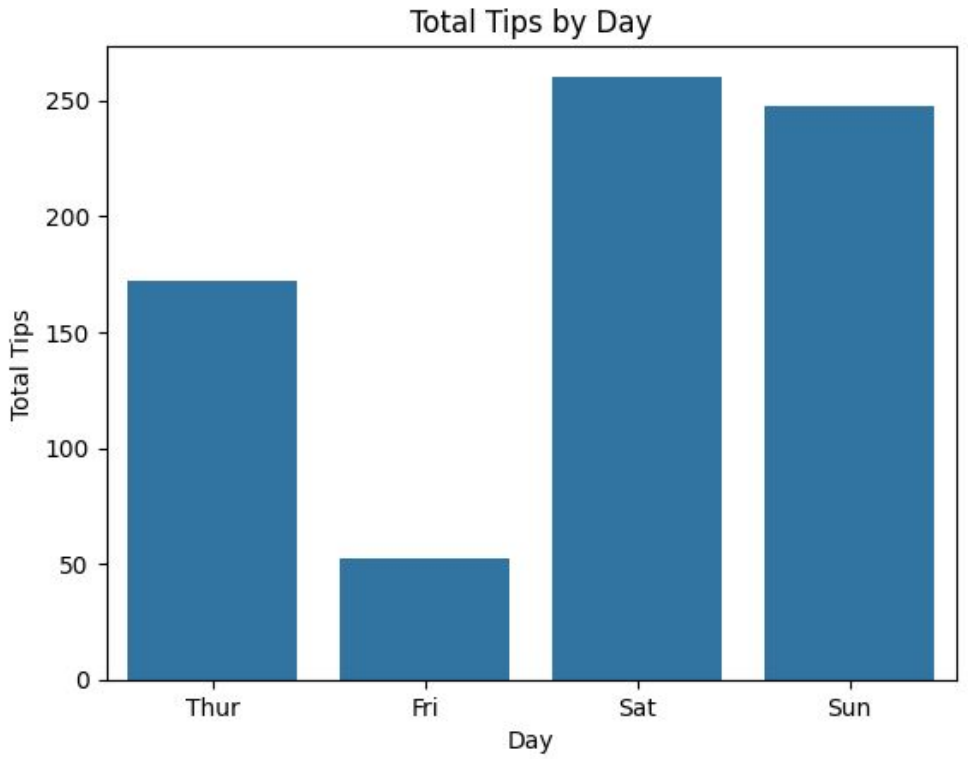
Practical Visualization Examples

Bar Plot to Show Categorical Data

- This bar plot shows the sum of tips collected for each day. It's useful for comparing categorical data and understanding how different days perform in terms of tip collection.

```
# Bar plot showing the average tip by day
sns.barplot(x='day', y='tip', data=data, estimator=sum, ci=None)
plt.title("Total Tips by Day")
plt.xlabel("Day")
plt.ylabel("Total Tips")
plt.show()
```

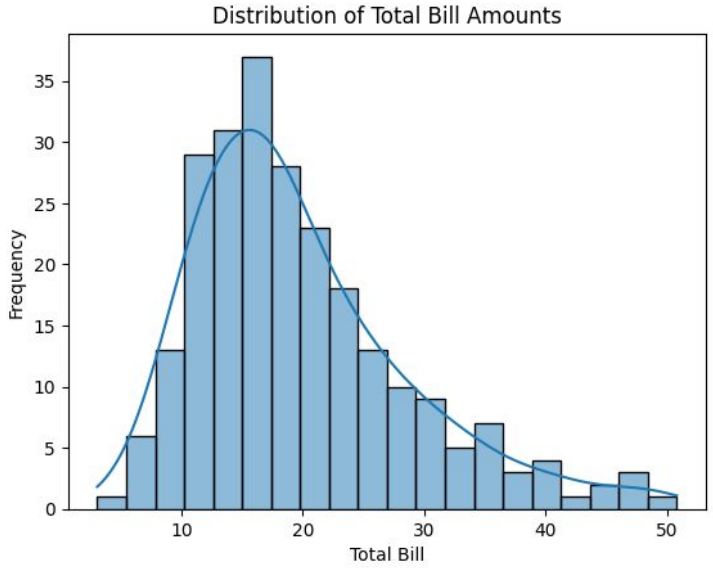
```
<ipython-input-16-e892f68ee143>:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='day', y='tip', data=data, estimator=sum, ci=None)
```



Practical Visualization Examples

Histogram to Show Data Distribution

```
# Histogram of total bill amounts
sns.histplot(data['total_bill'], bins=20, kde=True)
plt.title("Distribution of Total Bill Amounts")
plt.xlabel("Total Bill")
plt.ylabel("Frequency")
plt.show()
```

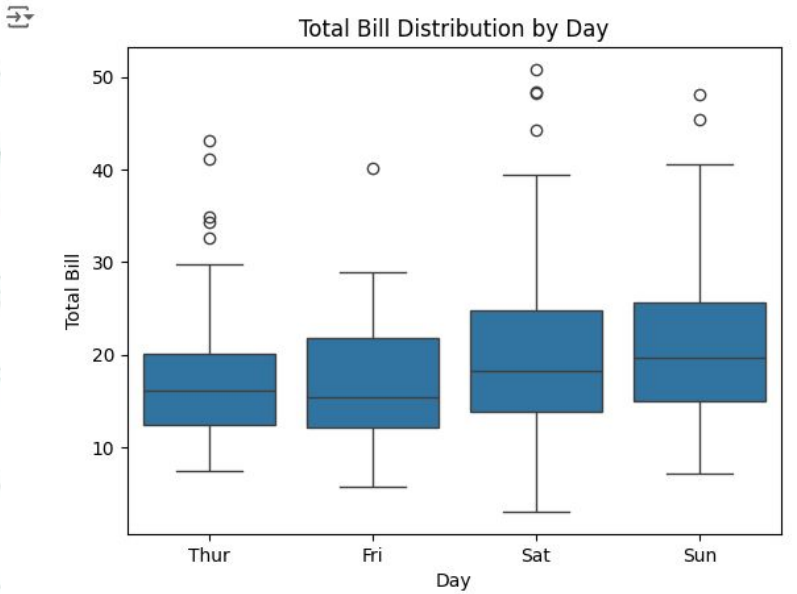


- This histogram displays the distribution of total bill amounts. The KDE (Kernel Density Estimate) line provides a smoothed version of the distribution, making it easier to see the underlying pattern.

Practical Visualization Examples

Box Plot to Visualize Distribution and Outliers

```
# Box plot showing the distribution of total bill amounts by day
sns.boxplot(x='day', y='total_bill', data=data)
plt.title("Total Bill Distribution by Day")
plt.xlabel("Day")
plt.ylabel("Total Bill")
plt.show()
```

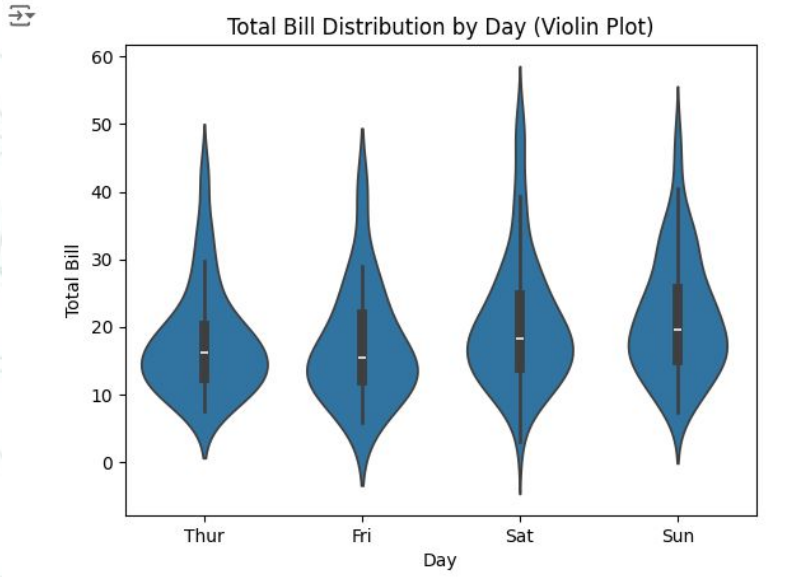


- Box plots provide a summary of the distribution, including median, quartiles, and potential outliers, for the total bill amounts categorized by day.

Practical Visualization Examples

Violin Plot to Show Density and Distribution

```
# Violin plot showing the distribution of total bill amounts by day
sns.violinplot(x='day', y='total_bill', data=data)
plt.title("Total Bill Distribution by Day (Violin Plot)")
plt.xlabel("Day")
plt.ylabel("Total Bill")
plt.show()
```



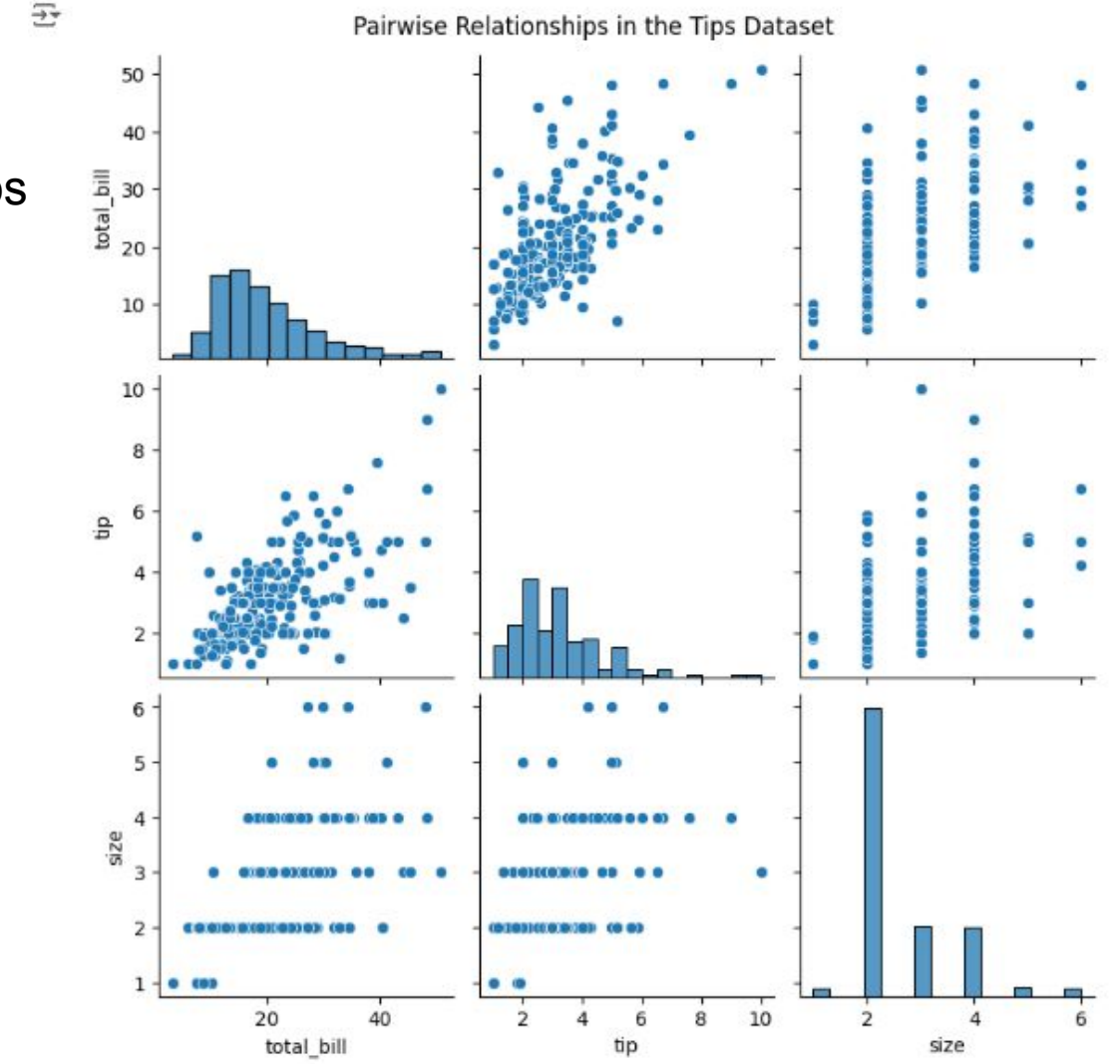
- Violin plots combine the features of box plots and KDE, showing the distribution and density of total bill amounts by day.

Practical Visualization Examples

Pair Plot to Show Pairwise Relationships

- Pair plots display pairwise relationships between different numerical variables in the dataset, helping to identify potential correlations and distributions.

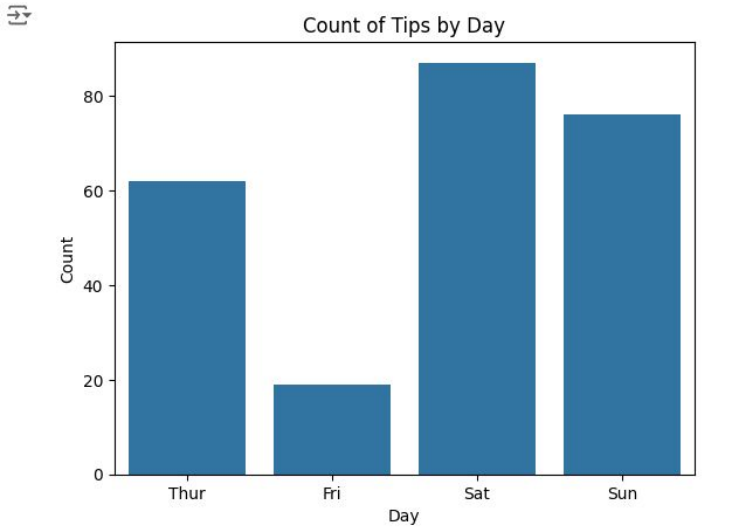
```
# Pair plot showing pairwise relationships in the 'tips' dataset
sns.pairplot(data)
plt.suptitle("Pairwise Relationships in the Tips Dataset", y=1.02)
plt.show()
```



Practical Visualization Examples

Count Plot to Show the Frequency of Categorical Values

```
# Count plot showing the count of tips by day
sns.countplot(x='day', data=data)
plt.title("Count of Tips by Day")
plt.xlabel("Day")
plt.ylabel("Count")
plt.show()
```



- Count plots visualize the frequency of occurrences for categorical data. This example shows how many entries exist for each day in the 'tips' dataset.

Practical Visualization Examples

Joint Plot to Show Distribution and Relationshi

- A joint plot combines a scatter plot and histograms, showing the relationship between total bill and tip amounts, along with their individual distributions.

```
# Joint plot showing the distribution of total bill and tip amounts
sns.jointplot(x='total_bill', y='tip', data=data, kind='scatter')
plt.suptitle("Joint Distribution of Total Bill and Tip", y=1.02)
plt.show()
```