Data Visualization in a Jupyter Notebook

Didem B. Aykurt

Colorado State University Global

MIS542; Business Analytics

Dr.Emmanuel Tsukerman

June 25, 2023

Explore Business Analysis by Python Data Visualization

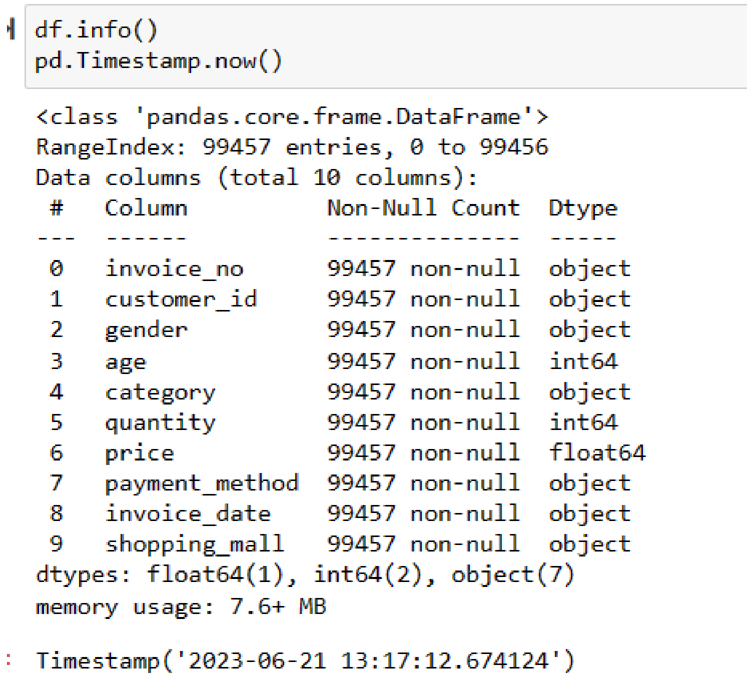
Data visualization is an excellent process for representing data, and it can also be a critical step in a data science process; it is easy to explain patterns and trends and communicate business needs and users as the most common visualization graphics: charts, plots, infographics, and animations. Harvard Business Review categorizes data visualization into four parts: idea generation is called design thinking before the start of a project by planning the action or process of different perspectives to make a visual to address critical stakeholders; the Idea illustration component transfers the primary purpose, tactic or strategy and primarily used in learning settings or organization structure or function or help communication between specific tasks such as Gantt chart and waterfall chart; visual discovery and everyday data viz help data analysts, and data competent explore patterns and trends with a dataset as data viz support every day dataset. Data visualization has a few steps: loading data, checking information, cleaning data, and exploring with visualization methods.

Data visualization can help change users' perspectives or behaviors; graphs or dashboards are most valuable and practical for tracking and visualizing data from various sources. And Python has a robust library to use to help create visualizations that are compelling and ready for publication that lists Matplotlib, Seaborn, Plotnine(ggplot), Bokeh, pygal, Plotly, geoplotlib, Gleam, missingno, Leather, Altair, and Folium. Let's look at the most common visualization methods and the most common case to prefer to use with the Python library:

Matplotlib

Matplotlib has the broadest range of graphs and is easy to use, so a low-level data visualization library is built with Numpy and other packages. It is best to use a case if it needs details about unique visualization problems and check quality or performance with a big community. The following visualization contains the customer shopping dataset with 99457 rows and 10 variables.

Figure 1: Customer shopping dataset information.



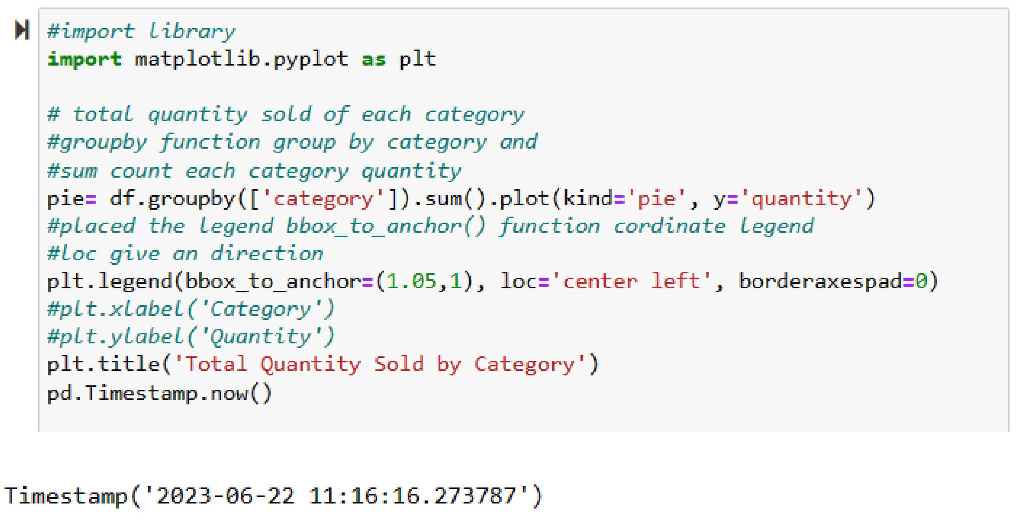
```
df.info()
pd.Timestamp.now()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   invoice_no      99457 non-null  object 
 1   customer_id     99457 non-null  object 
 2   gender          99457 non-null  object 
 3   age             99457 non-null  int64  
 4   category        99457 non-null  object 
 5   quantity        99457 non-null  int64  
 6   price           99457 non-null  float64
 7   payment_method  99457 non-null  object 
 8   invoice_date    99457 non-null  object 
 9   shopping_mall   99457 non-null  object 
dtypes: float64(1), int64(2), object(7)
memory usage: 7.6+ MB

Timestamp('2023-06-21 13:17:12.674124')
```

Pie chart

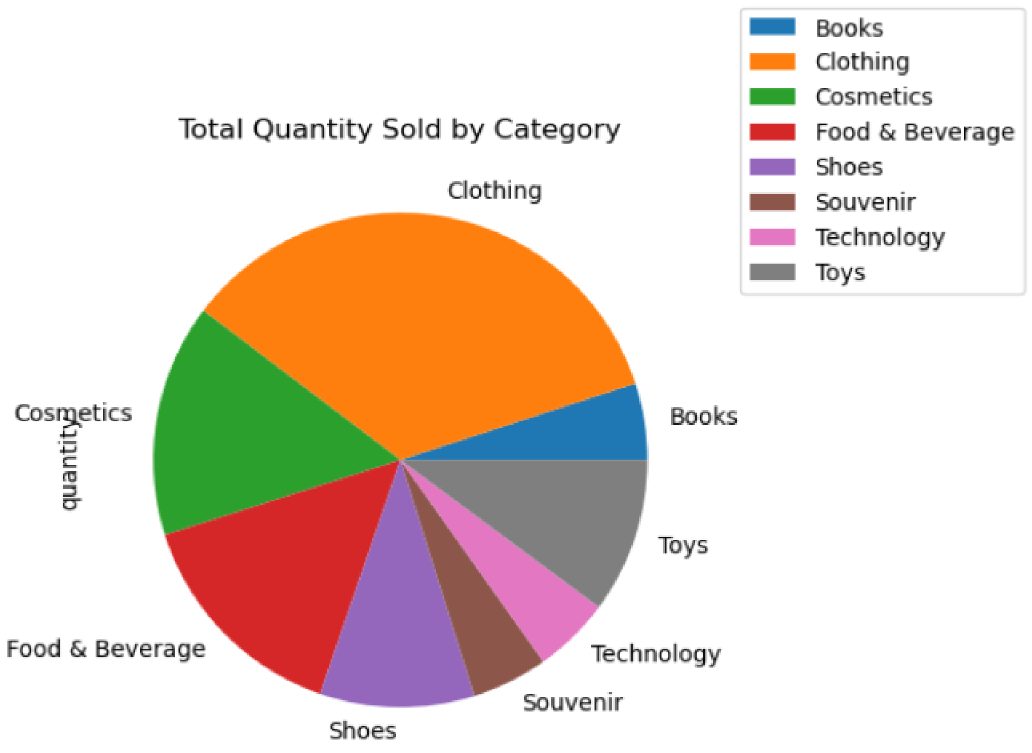
Pie charts are separated into parts of the whole and compare the size of each piece to another, such as finding answers to how many items are sold by category with a bar chart.

Figure 2: the total quantity sold by category with a pie chart.



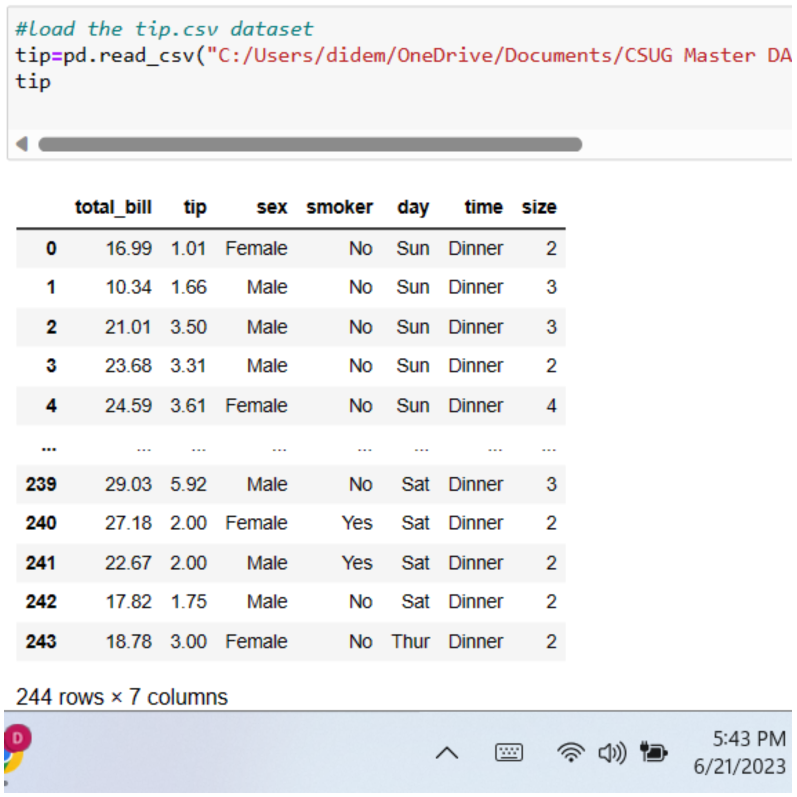
```python
#import library
import matplotlib.pyplot as plt

# total quantity sold of each category
#groupby function group by category and
#sum count each category quantity
pie= df.groupby(['category']).sum().plot(kind='pie', y='quantity')
#placed the legend bbox_to_anchor() function cordinate legend
#loc give an direction
plt.legend(bbox_to_anchor=(1.05,1), loc='center left', borderaxespad=0)
#plt.xlabel('Category')
#plt.ylabel('Quantity')
plt.title('Total Quantity Sold by Category')
pd.Timestamp.now()
```

```
Timestamp('2023-06-22 11:16:16.273787')
```



I will use the tip.csv dataset for the following visualization as the tip dataset contains customers in a restaurant during the two-and-a-half mounts with six columns.

Figure 3: The tip.csv dataset.



```
#load the tip.csv dataset
tip=pd.read_csv("C:/Users/didem/OneDrive/Documents/CSUG Master DA
tip
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

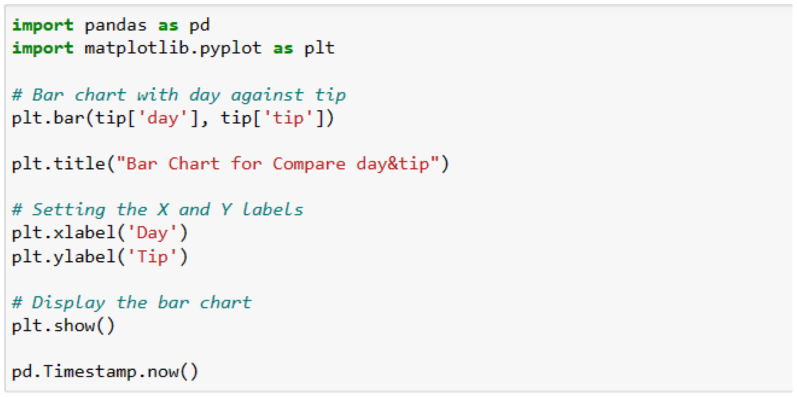
244 rows × 7 columns

5:43 PM
6/21/2023

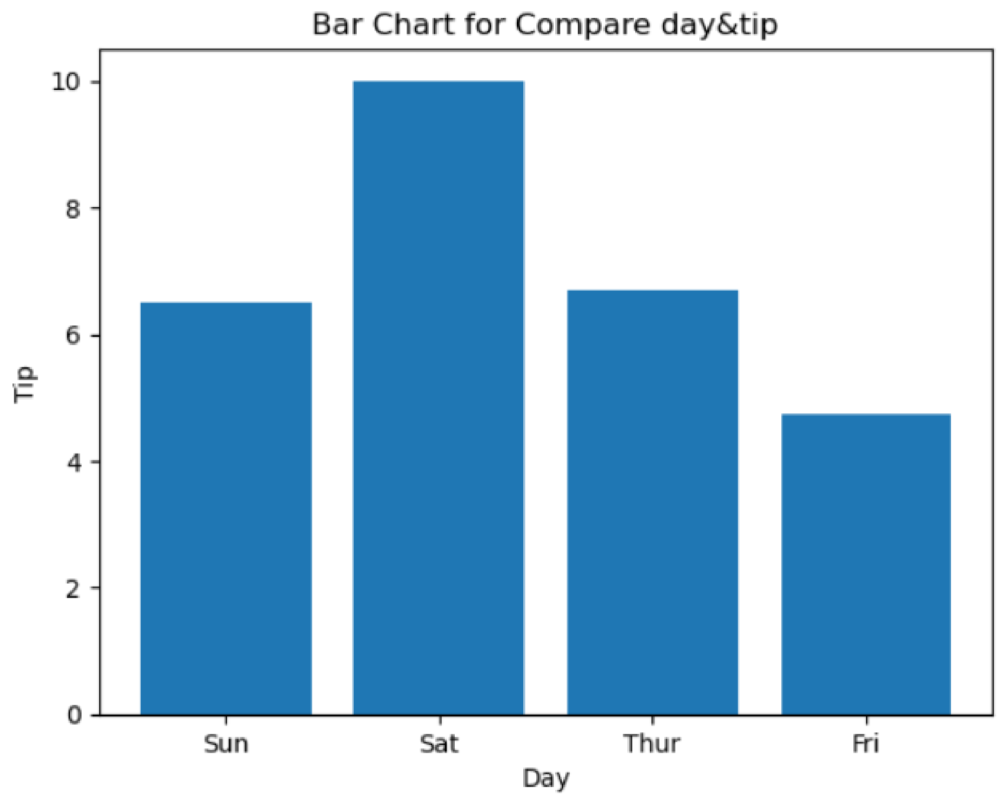
Bar chart

The bar chart explains the categorical data with rectangular bars whose lengths and heights show the group's values. The bar() function format X and y axes offer two variables.

Figure 4: Relation between day and tip with a bar chart.



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

# Bar chart with day against tip
plt.bar(tip['day'], tip['tip'])

plt.title("Bar Chart for Compare day&tip")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

# Display the bar chart
plt.show()

pd.Timestamp.now()
```

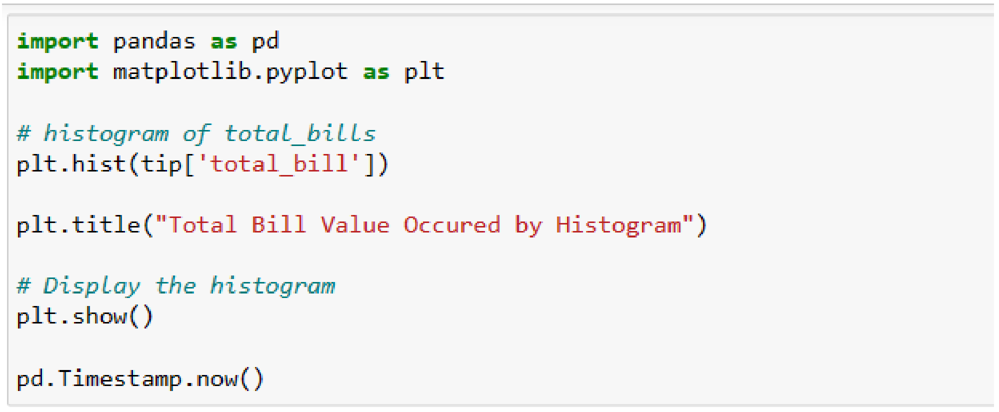


```
Timestamp('2023-06-22 11:49:43.543498')
```

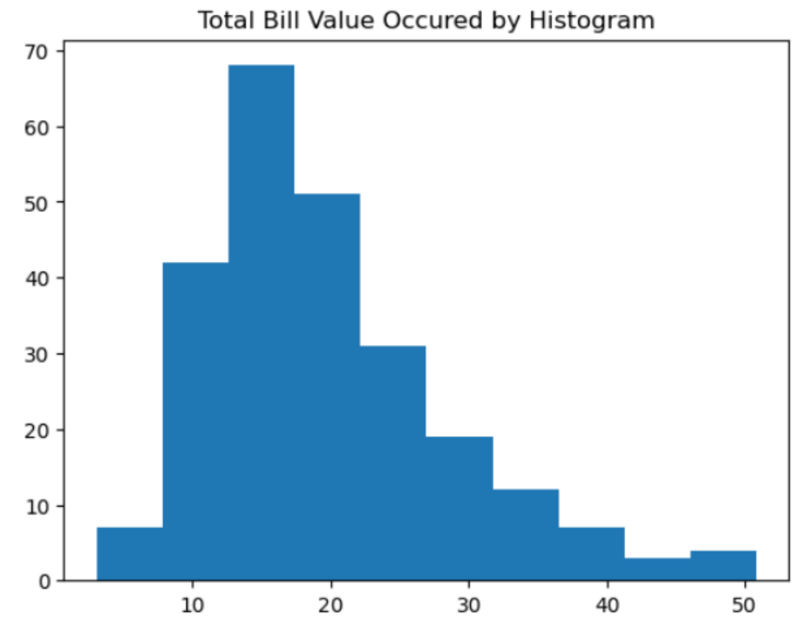
Histogram

A histogram represents a group of data like a bar plot format. The X-axis contains bin ranges, and the Y-axis shows information about occurrences with the hist() function, which automatically calculates the frequency of the data and creates a histogram.

Figure 5: Each value occurred off the total bill by histogram.



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

# histogram of total_bills
plt.hist(tip['total_bill'])

plt.title("Total Bill Value Occured by Histogram")

# Display the histogram
plt.show()

pd.Timestamp.now()
```



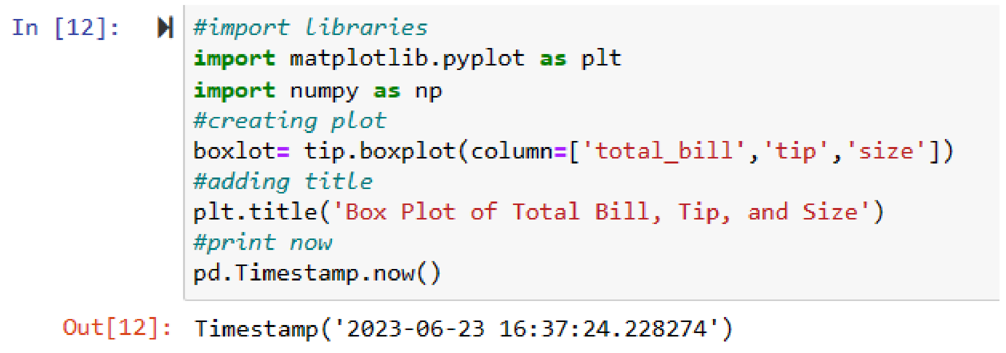
```
Timestamp('2023-06-22 12:21:39.426850')
```

Histogram can also show how data is distributed as there are three different skews: negative, average, and positively skewed; the histogram indicates that total bill data has a right-skewed.

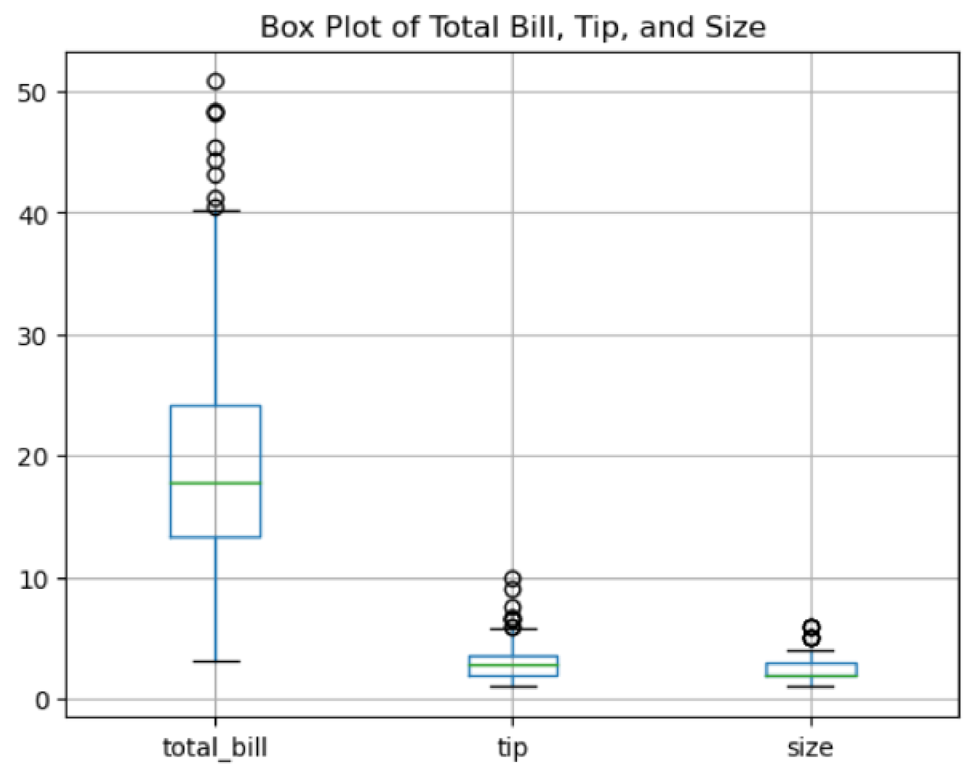
Box Plot

A box plot called a whisker plot helps create a summary of the dataset like minimum, first quarter, median, third quartile, and maximum in a box plot from the first to the third quartile, and the vertical line represents the median value.

Figure 6: Creating a box plot for the value of ‘total_bill,’ ‘tip,’ and ‘size.’



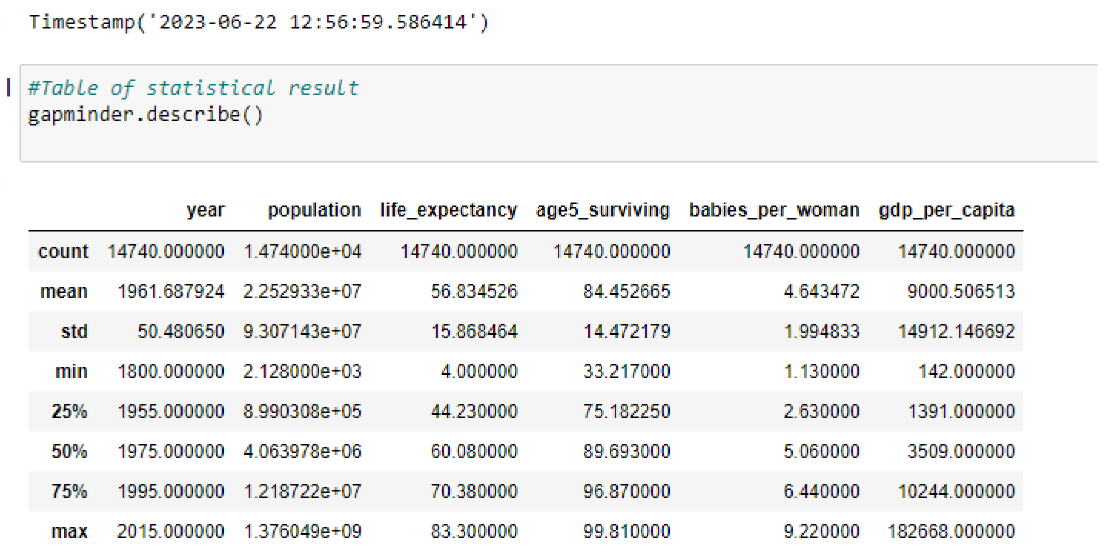
```
In [12]:  #import libraries
          import matplotlib.pyplot as plt
          import numpy as np
          #creating plot
          boxlot= tip.boxplot(column=['total_bill','tip','size'])
          #adding title
          plt.title('Box Plot of Total Bill, Tip, and Size')
          #print now
          pd.Timestamp.now()

Out[12]:  Timestamp('2023-06-23 16:37:24.228274')
```



For the following visualization, I will use the gapminder.csv file containing eight columns, including data about the global population health and wealth statistics in different countries from 1800 to 2015.

The table helps show helpful information, making it easy to interpret and understand statistical information data and check the validity. The table can be used to understand conditions, outcomes, number of worsen, and so on. Also, great to use when analyzing data or consuming date like a parameter.

Figure 7: Table of the global statistical result of the gapminder.csv dataset.



```
Timestamp('2023-06-22 12:56:59.586414')
```

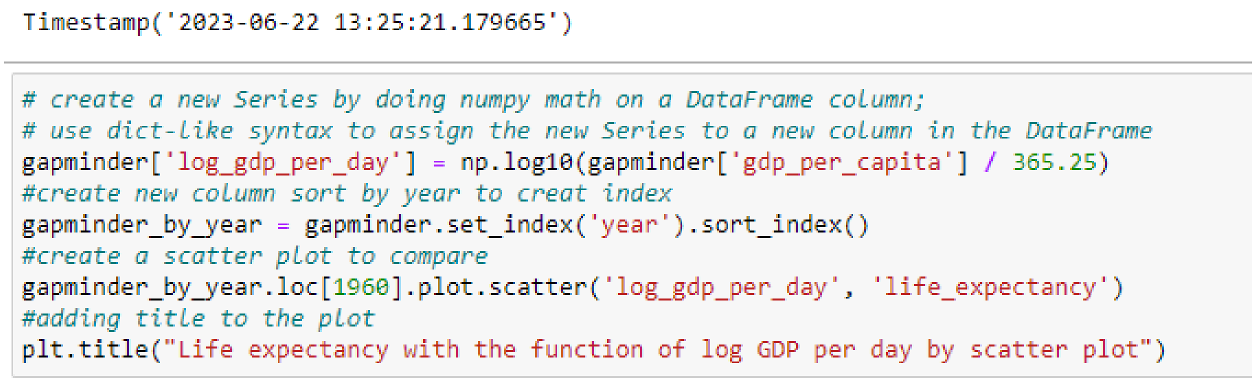
```
#Table of statistical result
gapminder.describe()
```

	year	population	life_expectancy	age5_surviving	babies_per_woman	gdp_per_capita
count	14740.000000	1.474000e+04	14740.000000	14740.000000	14740.000000	14740.000000
mean	1961.687924	2.252933e+07	56.834526	84.452665	4.643472	9000.506513
std	50.480650	9.307143e+07	15.868464	14.472179	1.994833	14912.146692
min	1800.000000	2.128000e+03	4.000000	33.217000	1.130000	142.000000
25%	1955.000000	8.990308e+05	44.230000	75.182250	2.630000	1391.000000
50%	1975.000000	4.063978e+06	60.080000	89.693000	5.060000	3509.000000
75%	1995.000000	1.218722e+07	70.380000	96.870000	6.440000	10244.000000
max	2015.000000	1.376049e+09	83.300000	99.810000	9.220000	182668.000000

Scatter Plot

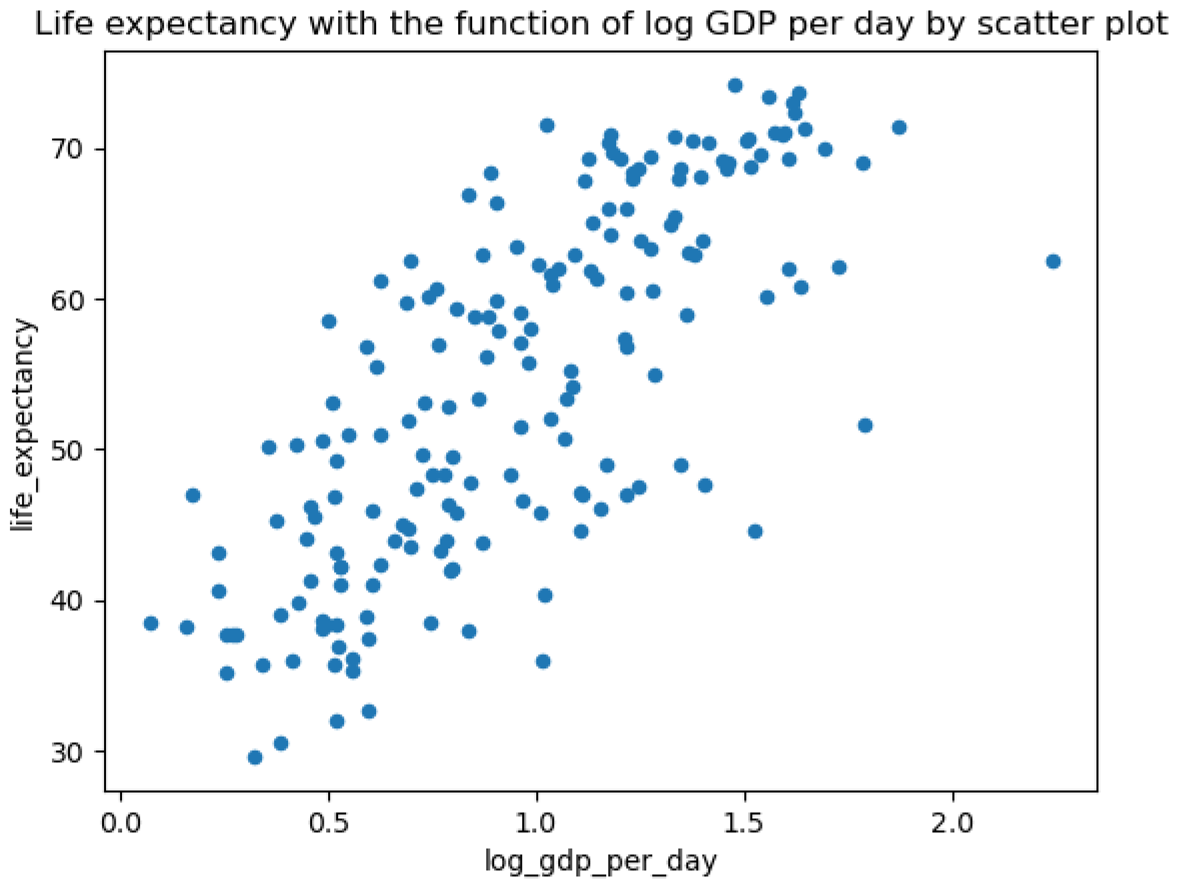
Scatter Plot helps to view relationships between variables with the scatter() method; draw dots to describe the relationship. The methods format axes x and y, s is size, c is color, vmin is the starting point, and vmax is the ending point. Let's look at the logarithm of the Gross Domestic Product (GDP) per person per day life expectancy years of 1960 with a scatter plot that can explain more clearly. First, create the new column as divided GDP per capita by the average number of days in a year, then apply a non-pi log 10 and generate the plot.

Figure 8: Life expectancy with the function of log GDP per day by scatter plot.



```
Timestamp('2023-06-22 13:25:21.179665')
```

```
# create a new Series by doing numpy math on a DataFrame column;
# use dict-like syntax to assign the new Series to a new column in the DataFrame
gapminder['log_gdp_per_day'] = np.log10(gapminder['gdp_per_capita'] / 365.25)
#create new column sort by year to creat index
gapminder_by_year = gapminder.set_index('year').sort_index()
#create a scatter plot to compare
gapminder_by_year.loc[1960].plot.scatter('log_gdp_per_day', 'life_expectancy')
#adding title to the plot
plt.title("Life expectancy with the function of log GDP per day by scatter plot")
```



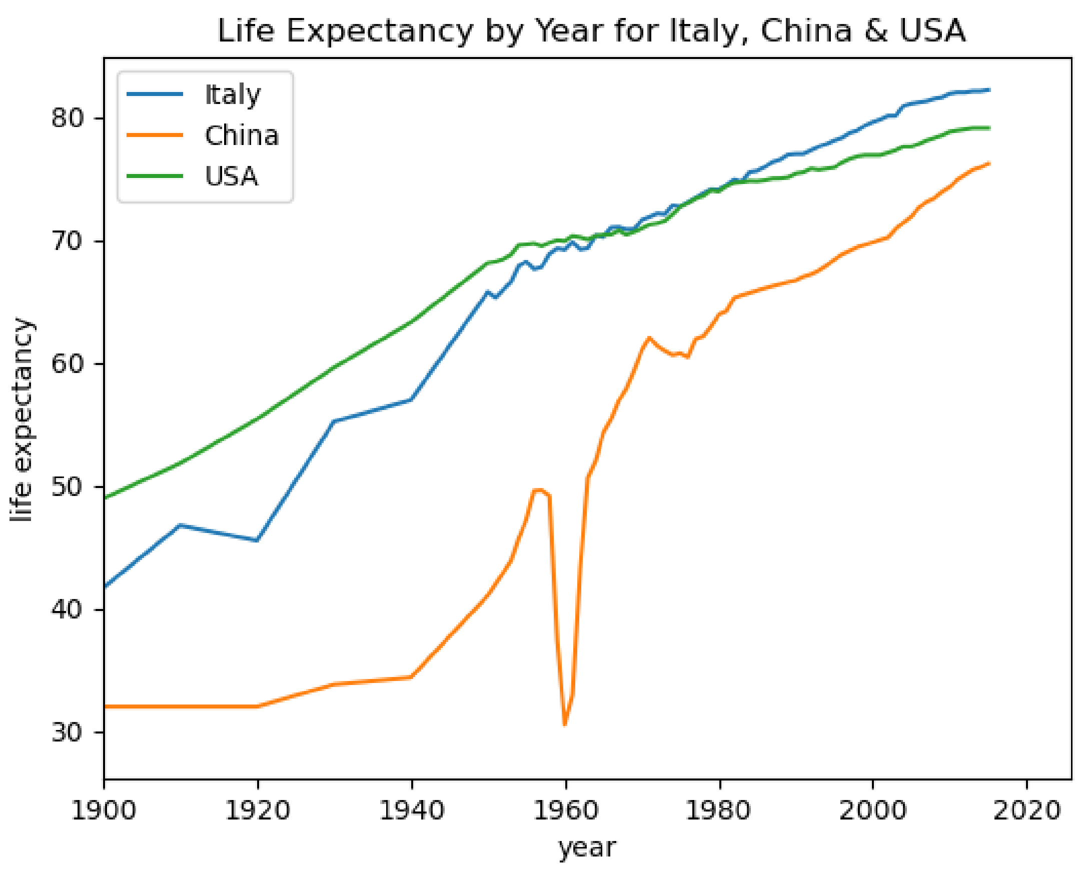
Line Chart

A line Chart helps to view two data, X and Y, on a different axis and see the other group of data's behavior with the plot() function format, such as life expectancy and year for Italy, China, and the USA. The loc[] method takes the specific group from the dataset, then the sort_values() function is sorted by year for each country, and the plot() process creates a line for each country.

Figure 9: compare Italy, China, and the USA's life expectancy by year with a line chart.

```
#create new column to sorth and set index
gapminder_by_country = gapminder.set_index('country').sort_index()
#filter the country and sorth by year and create line for each country
axes = gapminder_by_country.loc['Italy'].sort_values('year').plot('year', 'life_expectancy', label='Italy')
gapminder_by_country.loc['China'].sort_values('year').plot('year', 'life_expectancy', label='China', ax=axes)
gapminder_by_country.loc['United States'].sort_values('year').plot('year', 'life_expectancy', label='USA', ax=axes)
#set X and Y axis
plt.axis(xmin=1900)
plt.ylabel('life expectancy')
#adding title
plt.title(" Life Expectancy by Year for Italy, China & USA")
#print time now
pd.Timestamp.now()
```

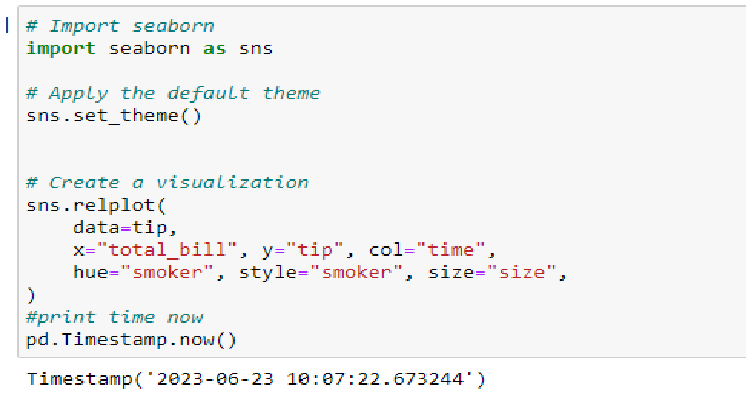
```
Timestamp('2023-06-22 13:51:48.992498')
```



Seaborn

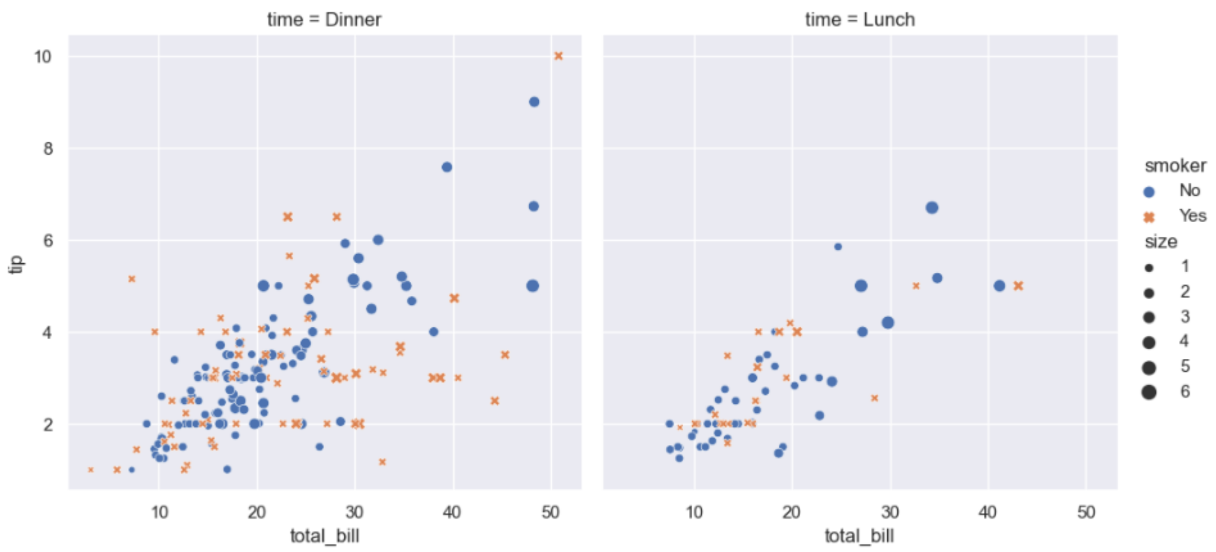
Seaborn built high-level statistical graphics in Python at the top of the Matplotlib and used it with a pandas data structure to help understand the dataset or arrays. Let’s use the tips.csv file to compare total_bill, tip, smoker, and size with the seaborn function. The relplot() method shows the relationship between five variables in the tips dataset.

Figure 10: Seaborn scatter plot to compare total bill, tip, smoker, time, and size.



```
# Import seaborn
import seaborn as sns

# Apply the default theme
sns.set_theme()


# Create a visualization
sns.relplot(
    data=tip,
    x="total_bill", y="tip", col="time",
    hue="smoker", style="smoker", size="size",
)
#print time now
pd.Timestamp.now()
```

```
Timestamp('2023-06-23 10:07:22.673244')
```



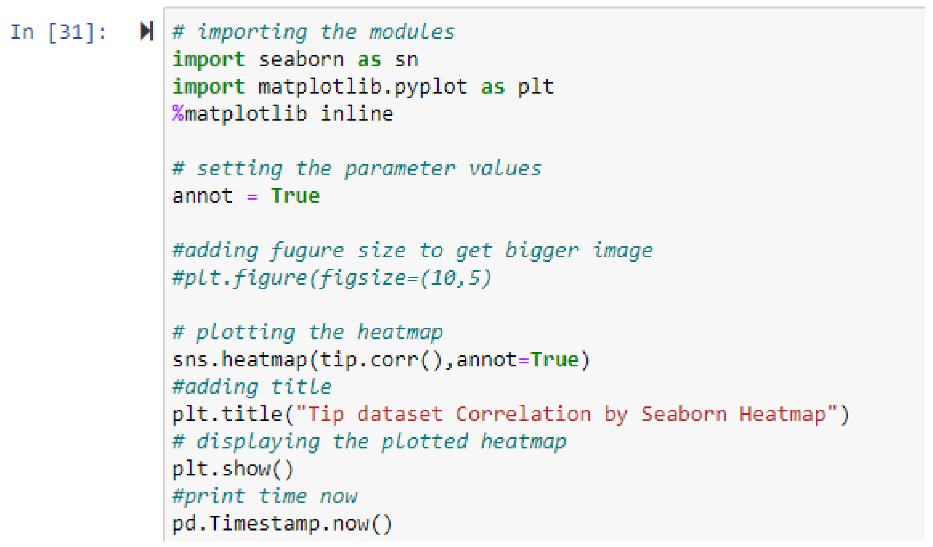
Heatmap

Heat maps create different shades of color, representing higher values in darker shades, and lower values in lighter shades can also create a different color, which means the data is different from other data values. That is a great graph to compare all the dataset variable relations between each other. Heatmaps used the seaborn package, and the syntax is :

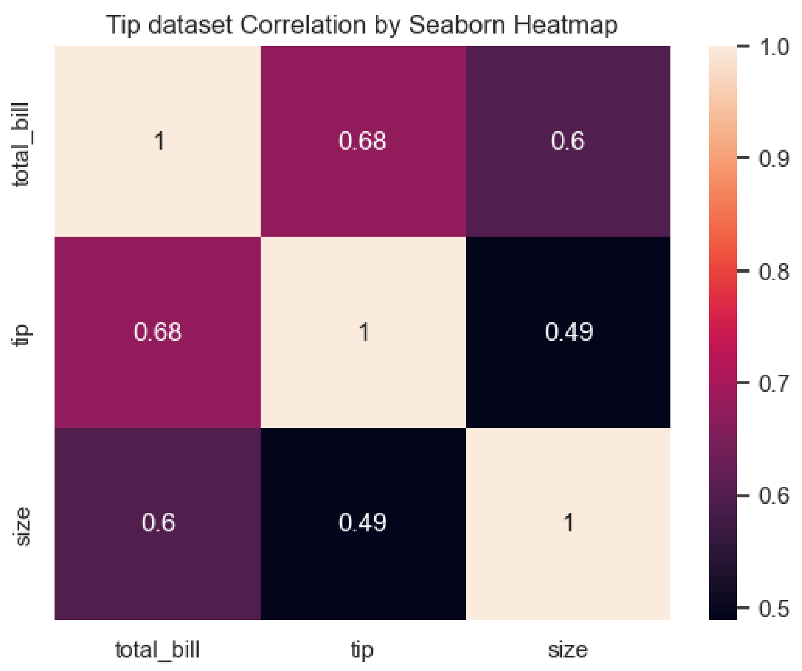
```
seaborn.heatmap(data, vmin=None, vmax=None, cmap=None, center=None, annot_kws=None,
linewidths=0, linecolor = 'black', cbar=True)
```

Note: From "Python Heat Maps" (2023). https://pythongeeks.org/python-heat-maps/

Figure 11: Heatmaps in the seaborn package for tip data set correlation.



```
In [31]:  # importing the modules
          import seaborn as sn
          import matplotlib.pyplot as plt
          %matplotlib inline

          # setting the parameter values
          annot = True

          #adding fugure size to get bigger image
          #plt.figure(figsize=(10,5)

          # plotting the heatmap
          sns.heatmap(tip.corr(),annot=True)
          #adding title
          plt.title("Tip dataset Correlation by Seaborn Heatmap")
          # displaying the plotted heatmap
          plt.show()
          #print time now
          pd.Timestamp.now()
```



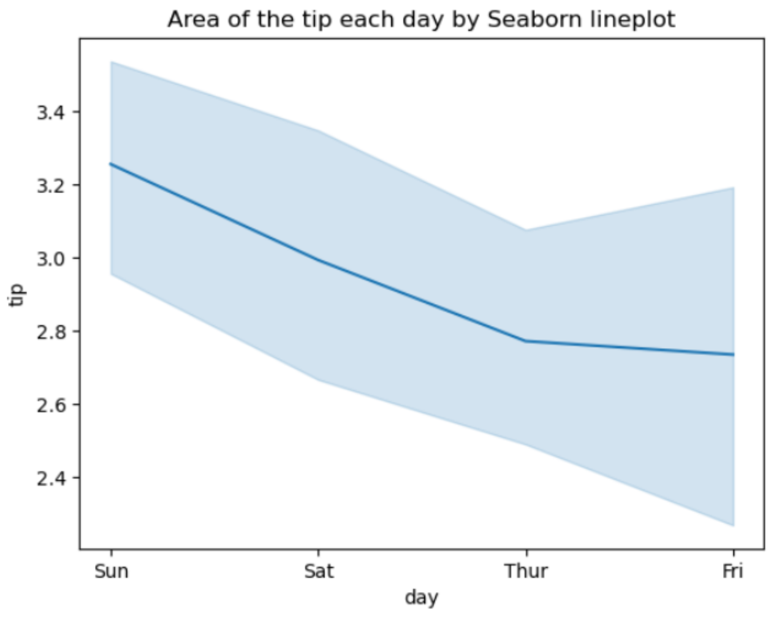
```
Out[31]:  Timestamp('2023-06-23 10:13:52.037404')
```

Seaborn line plot

A line plot draws a line with the possible area and creates three different dimensions separately, with hue, size, and style showing different subsets.

Figure 12: Seaborn line plot with probability area of the tip by day.

```
In [9]:  # importing packages
         import seaborn as sns
         import matplotlib.pyplot as plt
         import pandas as pd
         # draw lineplot
         sns.lineplot(x='day', y='tip', data=tip)
         # setting the title using Matplotlib
         plt.title('Area of the tip each day by Seaborn lineplot')
         plt.show()
         pd.Timestamp.now()
```



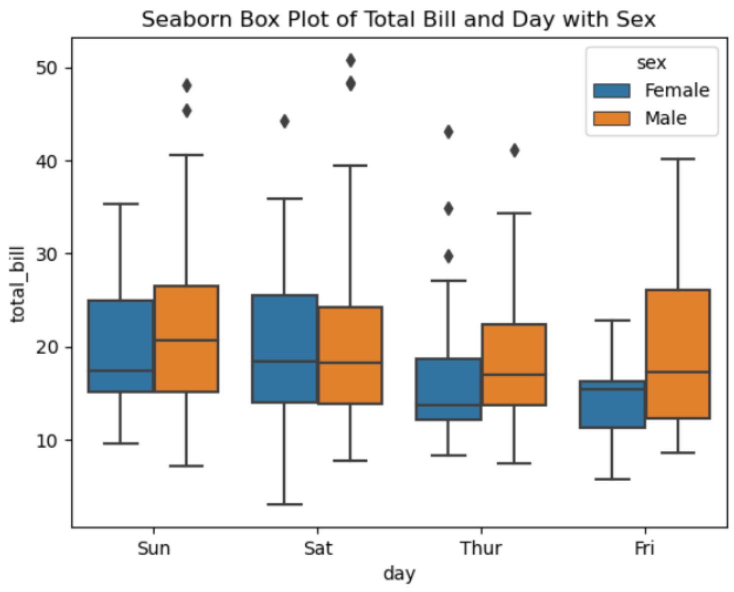
```
Out[9]: Timestamp('2023-06-23 16:14:44.275919')
```

Box Plot

Seaborn box plots are like matplotlib models, so they have higher performance and view than matplotlib box plots.

Figure 13: Seaborn box plot with tip dataset to view total bill by day and sex data.

```
In [22]:  import seaborn as sns

          #create box plot with tip dataset
          sns.boxplot(data=tip,x= "day", y="total_bill", hue= "sex")
          #adding title
          plt.title("Seaborn Box Plot of Total Bill and Day with Sex")
          #print now
          pd.Timestamp.now()

Out[22]:  Timestamp('2023-06-23 17:51:18.815916')
```



Conclusion

In this project, I used three data sets to help me create ten visualizations with two different plotting models, Matplotlib and Seaborn, as each library has its unique way and view. The customer shopping data set with pie charts and the tip dataset helped me create bar charts, histograms, and box plots, and the gapminder dataset helped me create statistical tables, scatter plots, and line charts with Matplotlib models. Additionally, the Seaborn model is excellent for deep diving into statistical views. I used the tip dataset to create a scatter plot, heatmap, box plot, and line plot showing the probability area.

References

McKinney, W. (n.d.). *Python for Data Analysis Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media. ISBN- 1491957638.

Yim, A., Chung, C., & Yu, A. (2018). Matplotlib for Python Developers *Effective Techniques for Data Visualization with Python.* 2nd Edition. ISBN 798-1-78862-517-3

Embarak, O. (2018). Data Analysis and Visualization Using Python *Analyze Data to Create Visualization for BI Systems. ISBN-13 (electronic): 978-1-4842-4109-7*