Data visualization is a key part of any data science workflow, Data visualization should really be part of your workflow from the very beginning, as there is a lot of value and insight to be gained from just looking at your data.
In data visualization, there are three main types of variables:

1. **Quantitative**: These are numerical data and represent a measurement. Quantitative variables can be discrete (e.g., units sold in 2016) or continuous (e.g., average units sold per person).
2. **Categorical**: The values of these variables are names or labels. There is no inherent ordering to the labels. Examples of such variables are countries, counties in a sales database and the names of products.
3. **Ordinal**: Variables that can take on values that are ranked on an arbitrary numerical scale. The numerical index associated with each value has no meaning except to rank the values relative to each other. Examples include days of the week, levels of satisfaction (not satisfied, satisfied, very satisfied), and customer value (low, medium, high, yes, no).

When visualizing data, the most important factor to keep in mind is the purpose of the visualization. This is what will guide you in choosing the best plot type. It could be that you are trying to compare two quantitative variables to each other. Maybe you want to check for differences between groups
Each of these goals is best served by different plots and using the wrong one could distort your interpretation of the data or the message that you are trying to convey

Introduction to Matplotlib

Matplotlib is the leading visualization library in Python. It is powerful, flexible, and has a dizzying array of chart types for you to choose from. For new users, matplotlib often feels overwhelming. You could spend a long time tinkering with all of the options available, even if all you want to do is create a simple scatter plot.

In your python(.py) file import

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

Importing Data with Pandas

The first step is to read the data. The data is stored as a comma-separated values, or csv, file, where each row is separated by a new line, and each column by a comma (,). In order to be able to work with the data in Python, it is needed to read the csv file into a Pandas DataFrame. A DataFrame is a way to represent and work with tabular data. Tabular data has rows and columns, just like this csv file (Click and Download it, save it inside your python project)..

```
# Import the pandas library, renamed as pd
import pandas as pd

# Read data.csv into a DataFrame, assigned to df
df = pd.read_csv("school.csv")
# Prints the first 5 rows of a DataFrame as default
print(df.head())
# Prints no. of rows and columns of a DataFrame
print(df.shape)
```

Class Distribution

Let's now take a look at the number of instances (rows) that belong to each class. We can view this as an absolute count.

```
# class distribution
print(df.groupby('Smoking').size())
print(df.groupby('Gender').size())
print(df.groupby('State').size())
```

Output

Smoking

0.0 304

1.0 37

2.0 70

dtype: int64

Gender

0.0 204

1.0 222

dtype: int64

State

In state 314

Out of state 94

dtype: int64

Explore Relationships Between Quantitative Variables

A common step in data analysis projects is to visually inspect and compare different quantitative variables in your dataset. This can quickly reveal relationships between your variables. For example, you may find that two independent variables are correlated and that you will need to account for that correlation in downstream analysis steps. Alternatively, your analysis might show a spurious relationship between variables that is only revealed through visual inspection.

group by Gender, Rank by Math performance. Math can be done in mean, sum, count..

```
z = df.groupby(['Gender','Rank'])['Math'].mean()
print(z)

z = df.groupby(['Gender','Rank'])['Math'].sum()
print(z)

z = df.groupby(['Gender','Rank'])['Math'].count()
print(z)

z = df.groupby(['Gender','Rank'])['Math'].std()  # standard deviation
print(z)

z = df.groupby(['Gender','Rank','Smoking'])['Math'].mean()  # standard deviation
print(z)
```

Run each at time and see the the output

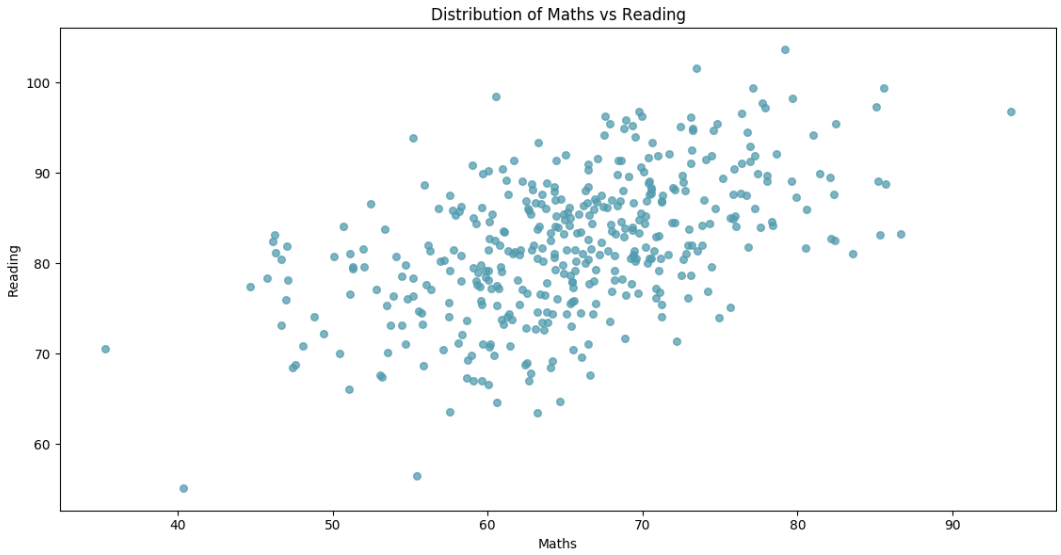
1. **Scatter Plot**

The first plot to consider in these situations is the scatter plot. In many cases this is the least aggregated representation of your data. We will plot the **Math** scores against the **Reading**:

```
# Code

import matplotlib.pyplot as plt

# Create the plot object
_, ax = plt.subplots()
# Plot the data, set the size (s), color and transparency (alpha)
# of the points
ax.scatter(df['Math'], df['Reading'], s = 30, color = '#539caf', alpha = 0.75)
    # Label the axes and provide a title
ax.set_title("Distribution of Maths vs english")
ax.set_xlabel("Maths")
ax.set_ylabel("Reading")
plt.show()
```



Interpretation

We can see that the scatter tends to move upwards to y axis and not much on x-axis, this shows Reading scored higher than Maths but not with a big marging...at least by 15 marks higher.

Examine Distributions

We will now switch gears and look at the family of plots for visualizing distributions. These plots can provide instant insights and guide further analysis. Is it uniform (equal frequency over all observed values)? Are there peaks at particular values? If so, which ones? You might find that a variable is extremely skewed and will need to be transformed.

1**a**. **Heatmaps**

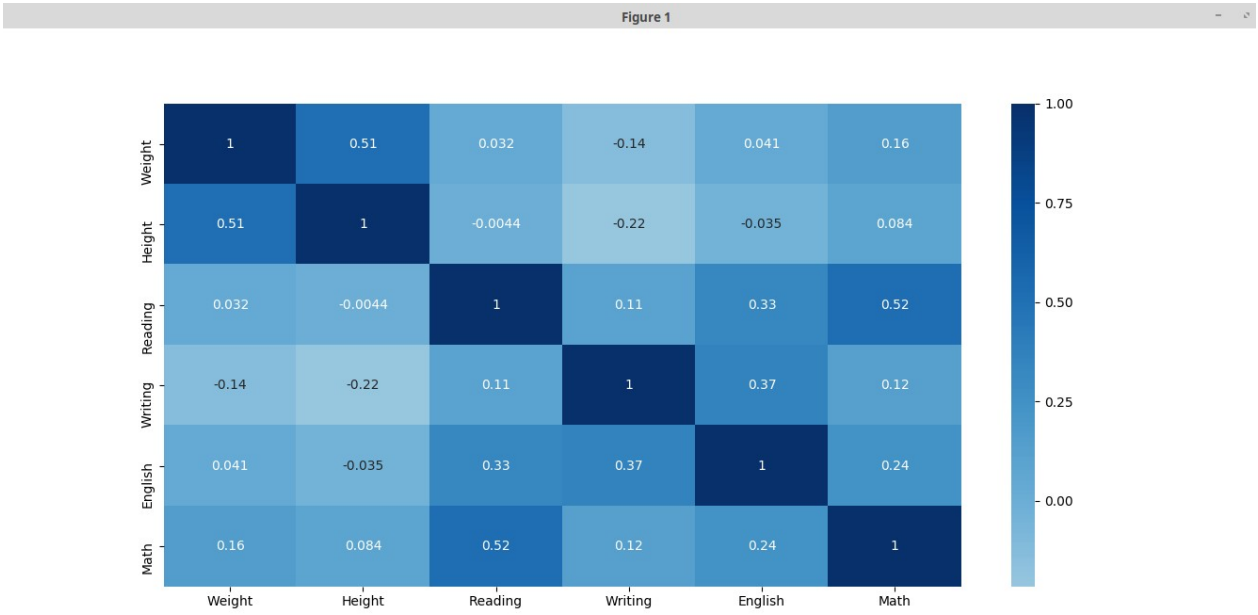
A *heat map* (or *heatmap*) is a graphical representation of data where the individual values contained in a matrix are represented as colors

The heatmap above represents the collinearity of the multiple variables in the dataset.**data**.**corr**() was used in the code to show the correlation between the values.

0 – represent no correlation, 1 represent strong positive correlation, -1 represent Strong Negative correlation.

```
df = pd.read_csv("school.csv")
import seaborn as sns
fig, ax = plt.subplots(figsize=(10,6))
sns.heatmap(df.corr(), center=0, cmap='Blues', annot=True)
plt.show()
```

cmap parameter can have other values such as **Greens and Reds**, see next heatmaps using blues

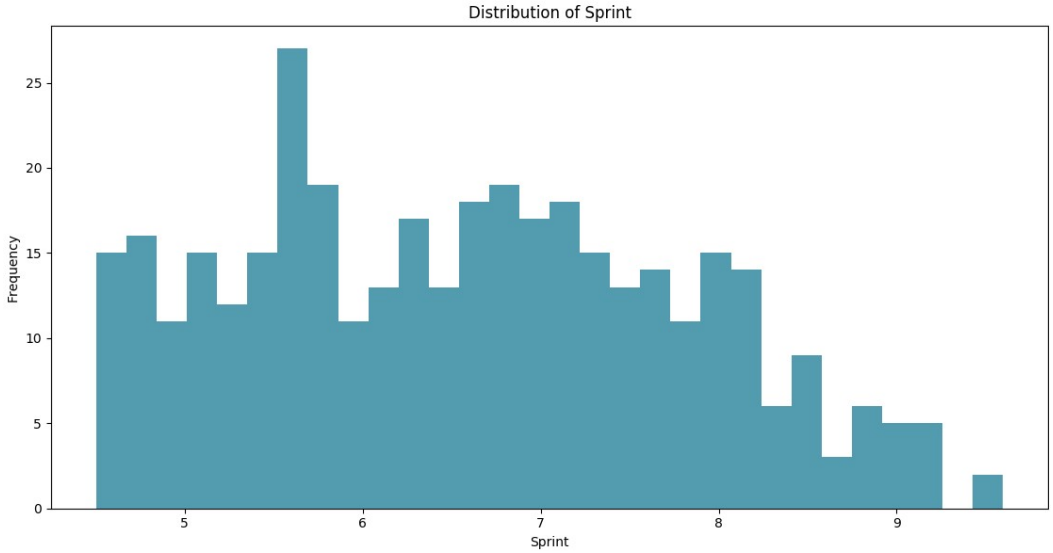


2. **Histogram**

Histograms are used to get a rough idea of how a quantitative variable is distributed. The observed values are placed into different bins and the frequency of observations in each of those bins is calculated.

The histogram represents the frequency of occurrence of specific phenomena which lie within a specific range of values and arranged in consecutive and fixed intervals.

For this example, let's examine the distribution of registered **Sprint time**.




```
# code
_, ax = plt.subplots()
ax.hist(df['Sprint'], color='#539caf', bins=30)
    # Label the axes and provide a title
ax.set_title("Distribution of Sprint")
ax.set_xlabel("Sprint")
ax.set_ylabel("Frequency")
plt.show()
```

Above graph shows that 26 students sprinted with 5.5 seconds , also with a good number of students sprinted with 6 to 8 seconds, very few students who sprinted with above 9 seconds..

Your Turn.

Draw a scatter plot for H**eight** and W**eight**

Draw a histogram of **Writing**

What conclusions do you make? Draw a scatter plot for height and weight

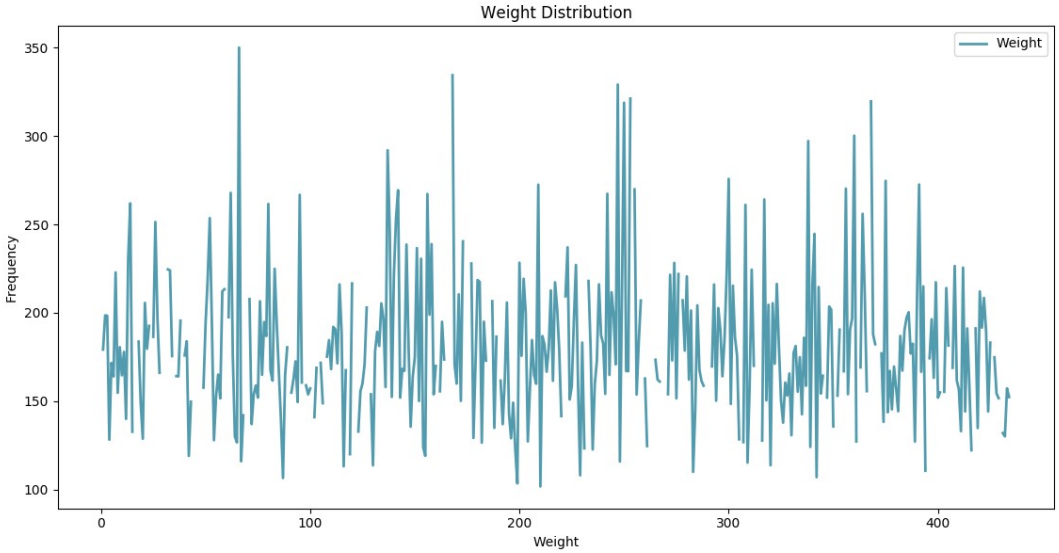
3. **Overlaid Histogram**

If you are looking to compare two (or more) distributions, use an overlaid histogram. Some additional care needs to be taken with these plots to ensure that they remain clear and easy to read, especially when more than two distributions are visualized. In this example, we will compare the distributions **Reading** and **Math**.

```
import matplotlib.pyplot as plt
# Create the plot object
# Create the plot
_, ax = plt.subplots()
ax.hist(df['Reading'], bins=20, color='red', alpha=1, label="Reading")
ax.hist(df['Math'], bins=20, color='green', alpha=0.75, label="Math")
ax.set_ylabel("Frequency")
ax.set_xlabel("Reading vs Math")
ax.set_title("Reading vs Math")
ax.legend(loc='best')
plt.show()
```

4. **Density Plot**

Although histograms are intuitive and easily digested, the apparent shape of the distribution can be strongly affected by the number of bins chosen. Using a density plot is a more rigorous method to determine the shape of a distribution. This constructs an estimate of the underlying probability density function of the data. In the example below, we will use **Weight**



Above plot shows distribution of weight and shows high frequency and lows frequency

```
import matplotlib.pyplot as plt
# Create the plot object
# Create the plot
# Create the plot object
_, ax = plt.subplots()
ax.plot(df['Weight'], color='#539caf', lw=2)
ax.set_ylabel("Frequency")
ax.set_xlabel("Weight")
ax.set_title("Weight Distribution")
ax.legend(loc='best')
plt.show()
```

Your turn!

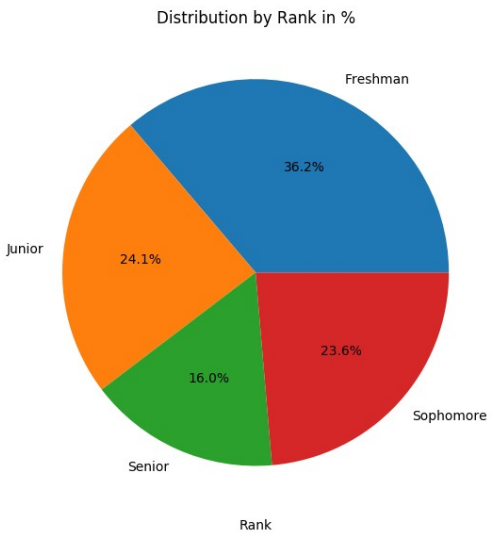
Create an Overlaid histogram for S**leepTime** and S**tudyTime**

Compare Groups or Categories

The final family of plots that we will cover are used to compare quantitative variables between different groups or categories. Arguably, this group of plots have the highest number of factors to take into consideration during creation. For example, is a stacked or grouped bar chart more appropriate? If you decide on the grouped version, which level of grouping will you use? How many distinct groups should be displayed and which, if any, should be grouped together into an "other" category? These are likely to be among the plots that you will use the most. As such, it will really pay off to consider these details when making your design choices.

5. **Pie charts in Matplotlib**

Pie charts are good to show proportional data of different categories and figures are usually in percentages here below pie chart shows distribution of **Rank** in our data set.




```
import matplotlib.pyplot as plt
_, ax = plt.subplots()
df.groupby('Rank').size().plot(kind='pie',autopct='%1.1f%%')
ax.set_xlabel("Rank")
ax.set_ylabel("")
ax.set_title("Distribution by Rank in %")
plt.show()
```

Your turn! .. Create a pie chart of HowCommute

Adding explode

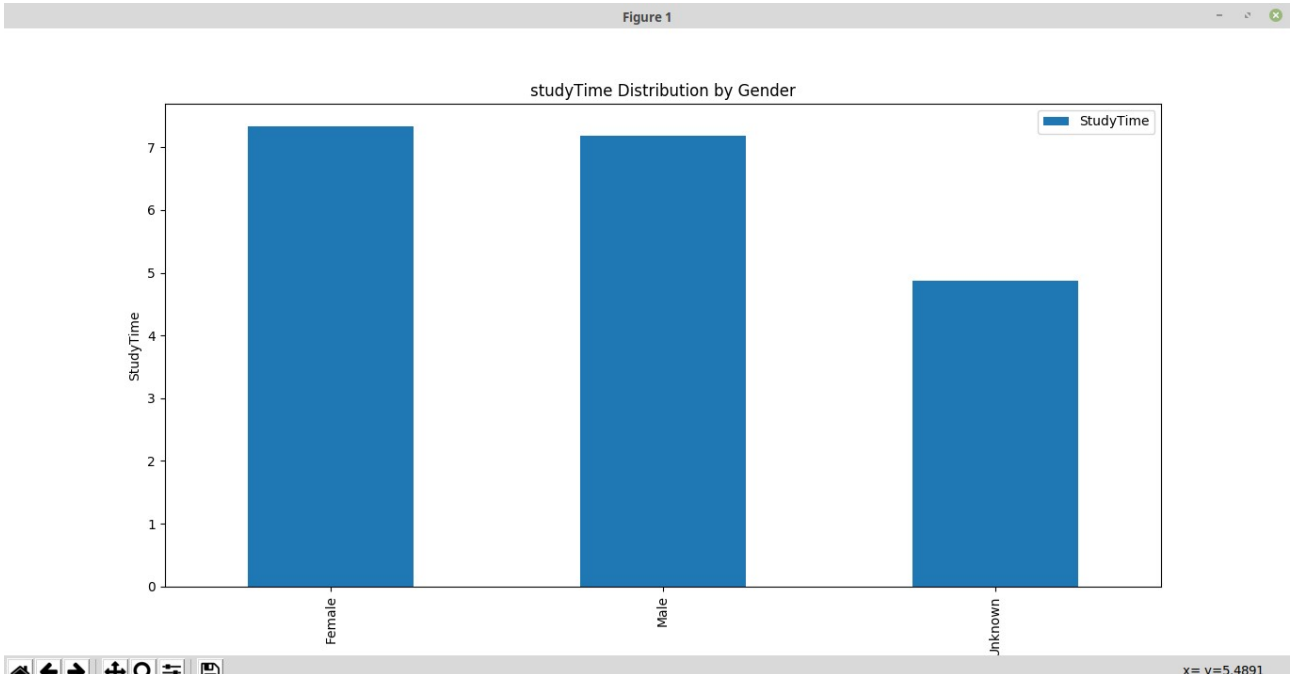
```
explode = (0, 0.1, 0, 0.3) # here we set explode for all Categories

# then add explode parameter as shown below..

df.groupby('Rank').size().plot(kind='pie',autopct='%1.1f%%', explode=explode)
```

6. **Bar Plot**

The simple bar plot is best used when there is just one level of grouping to your variable. Let's take a look at what mean of S**tudyTime** by Gender, this will help understand the learning culture of both Male and Female

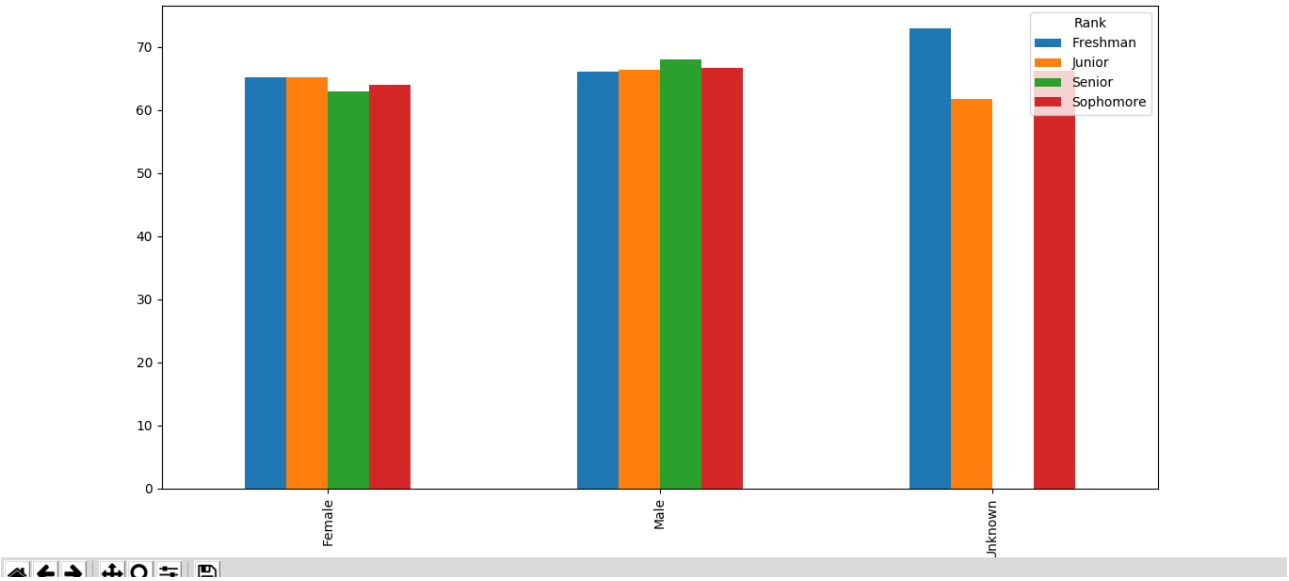



```
import matplotlib.pyplot as plt
_, ax = plt.subplots()
df.groupby('Gender')['StudyTime'].mean().plot(kind='bar', color = "#ffcccc") # use barh for horizontal bar chart
ax.set_ylabel("StudyTime")
ax.set_xlabel("Gender")
ax.set_title("studyTime Distribution by Gender")
ax.legend(loc='best')
plt.show()
```

7. **Stacked Bar Plot**

Stacked bar plots are best used to compare proportions between categories (proportion of Gender vs. Rank by their Math score).

```
import matplotlib.pyplot as plt
# code
_, ax = plt.subplots()
df.groupby(['Gender', 'Rank'])['Math'].mean().unstack().plot(kind='bar', stacked=False)
ax.set_ylabel("Math")
ax.set_xlabel("Gender")
ax.set_title("Math Distribution by Gender and Rank")
plt.show()
```



Your turn!

Draw a stacked graph to compare proportions between categories (proportion of **LiveOnCampus** vs. **Gender** by their **SleepTime**).

Read more

https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/

https://swcarpentry.github.io/python-novice-gapminder/09-plotting/

https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/

https://pbpython.com/simple-graphing-pandas.html

https://www.geeksforgeeks.org/data-visualization-different-charts-python/

https://www.geeksforgeeks.org/data-analysis-visualization-python/