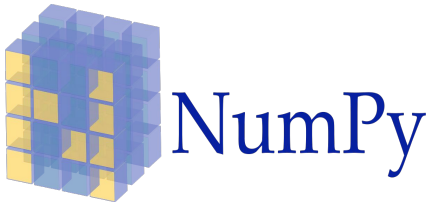
Machine Learning Training

Week 1

Salahadin S. Musa
salubinseid@gmail.com

Python and Popular libraries

- Popular Python Library
  - TensorFlow
  - Scikit-Learn
  - Numpy
  - Keras -
  - PyTorch
  - SciPy
  - Theano
  - Pandas
  - … more



Installing and Getting Started

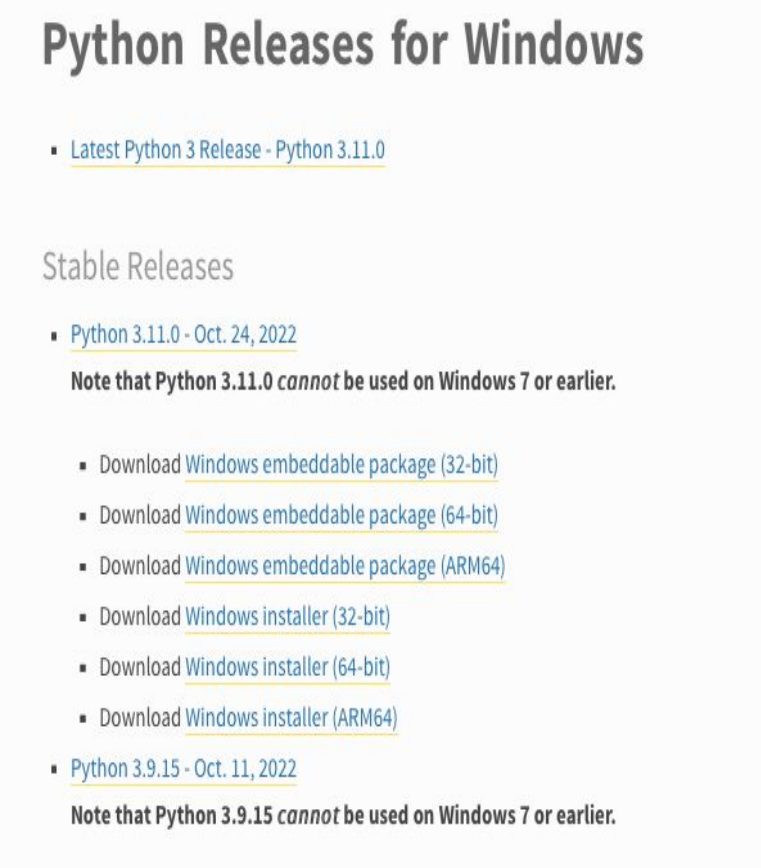
- Install - https://www.python.org
  - Many computers may have python installed already
  - Check if you have installed on a window PC: type the following on the command line (cmd.exe) on window

```
C:\Users\Your Name> python --version
```

  - Download anaconda / Miniconda - https://www.anaconda.com/products

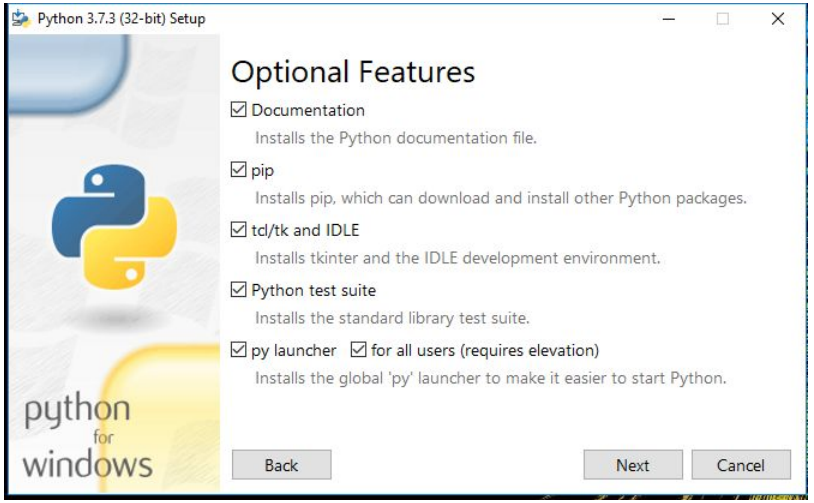
Installing and Getting Started

- Install - https://www.python.org
  - Step 1: Download the Python Installer binaries
    - Click on the link to download **Windows x86 executable installer** if you are using a 32-bit installer.
    - In case your Windows installation is a 64-bit system, then download **Windows x86-64 executable installer**.



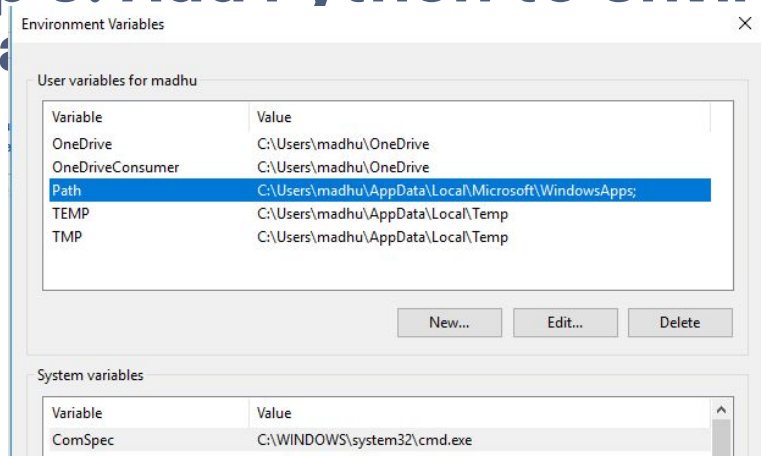
Installing and Getting Started

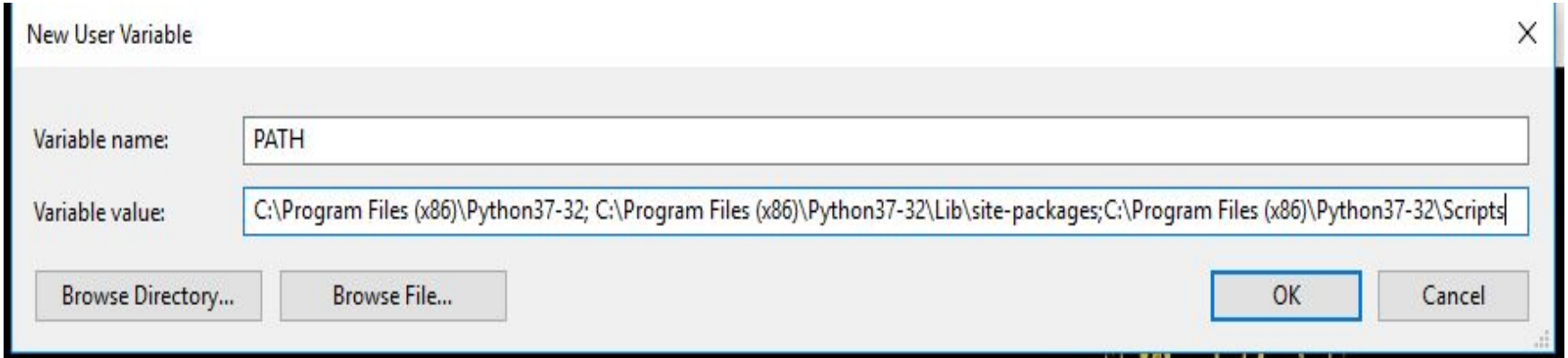
- Install - https://www.python.org
  - **Step 2: Run the Executable Installer**



Installing and Getting Started

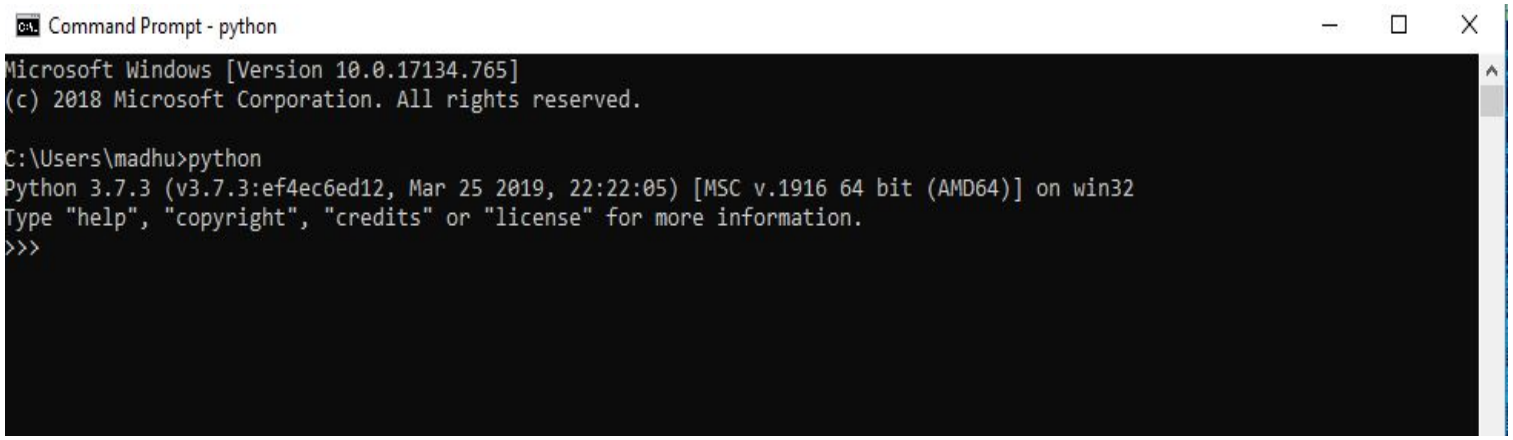
- Install - https://www.python.org
  - **Step 3: Add Python to environmental vari**





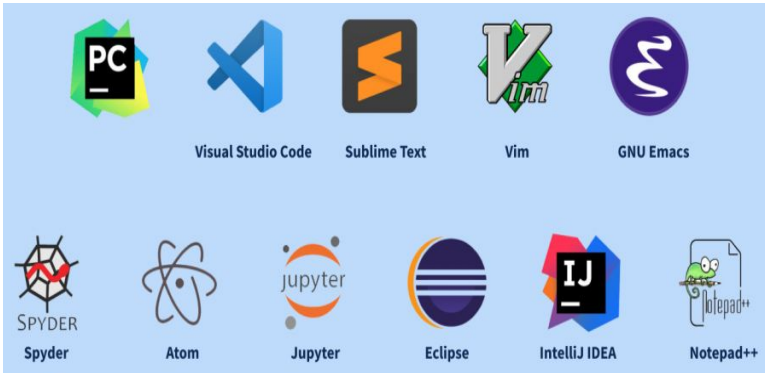
Installing and Getting Started

- Install - https://www.python.org
  - **Step 4: Verify the Python Installation**



Text Editor …

- Using your favorite text editor



Google Colab

- No Installation
  - https://colab.research.google.c

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

Running a Python File

- Using your favorite text editor
  - Write this python code print(“Hello, World”)

  and save with file name helloworld.py
  - Run it on the command line

```
C:\Users\Your Name> python helloworld.py
```

  - The output should read:

```
Hello, World
```

Congratulations,
you have written and executed your first Python program.

Install python libraries

pip install numpy pandas scikit-learn tensorflow keras opencv-python nltk

NumPy

- NumPy is a fundamental library for numerical computing in Python.
- It provides efficient **array operations** and mathematical functions, making it an essential library for machine learning tasks.

```
import numpy as np
```

```
# Creating an array
arr = np.array([1, 2, 3, 4, 5])
# Performing array operations
arr_sum = np.sum(arr)
arr_mean = np.mean(arr)
```

Pandas: Data Manipulation and Analysis

- Pandas is a versatile library for data manipulation and analysis.
- It provides powerful data structures, such as DataFrame and Series - data loading, preprocessing and analysis.

```
import pandas as pd
```

```
# Loading data from a CSV file
df = pd.read_csv('data.csv')
# Performing data analysis
df.head()  # Display the first few rows
df.describe()  # Statistical summary of the data
```

Scikit-learn: Machine Learning Made Easy

- Scikit-learn is a comprehensive library for machine learning tasks.
- It offers a wide range of algorithms for **classification, regression, clustering**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
# Loading the Iris dataset
data = load_iris()
X, y = data.data, data.target # Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Creating and training a Decision Tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)# Making predictions
predictions = clf.predict(X_test)
```

TensorFlow: Deep Learning at Scale

- **TensorFlow** is a powerful library for building and deploying **deep learning models.**
- It provides a flexible architecture for **creating neural networks** and supports distributed computing for large-scale applications.

```
import tensorflow as tf
```

```
# Creating a simple neural network
model = tf.keras.Sequential([
   tf.keras.layers.Dense(64, activation='relu', input_shape=(784,)),
   tf.keras.layers.Dense(10, activation='softmax')
])
# Compiling and training the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10)# Evaluating the model
loss, accuracy = model.evaluate(X_test, y_test)
```

Keras: High-Level Deep Learning Library

- Keras is a u**ser-friendly API** built on top of TensorFlow that simplifies the process of building deep learning models.
- It provides a high-level interface for **creating and training neural networks.**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
# Creating a simple neural network
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(784,)))
model.add(Dense(10, activation='softmax')) # Compiling and training the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10) # Evaluating the model
loss, accuracy = model.evaluate(X_test, y_test)
```

OpenCV: Computer Vision and Image Processing

- For computer vision tasks: image manipulation, object detection and recognition.

```
import cv2
```

```
image = cv2.imread('image.jpg') # Reading and displaying an image
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()# Performing object detection using Haar cascades
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5)
for (x, y, w, h) in faces:
   cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
cv2.imshow('Faces', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natural Language Toolkit (NLTK): Text Processing and NLP

- NLTK is a comprehensive library for **natural language processing (NLP) tasks**.
- It provides tools for text preprocessing, sentiment analysis, language modeling and more.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.sentiment import SentimentIntensityAnalyzer
```

```
# Tokenizing a sentence
sentence = "NLTK is a powerful library for NLP tasks."
tokens = word_tokenize(sentence)# Performing sentiment analysis
sia = SentimentIntensityAnalyzer()
polarity = sia.polarity_scores(sentence)['compound']
```

Data Analysis and Manipulation using Pandas

Topics

- What is Pandas?
- Data Loading Pandas
- Data Cleaning: Handling missing values, removing duplicates, and correcting inconsistencies.
- Data Transformation: Filtering, sorting, grouping, and reshaping data.
- Data Aggregation: Calculating summary statistics (mean, median, sum, etc.).
- Data Merging: Combining datasets.

Pandas

- Pandas - a powerful Python library for data analysis and manipulation.
  - It provides data structures like **DataFrames** that allow for efficient handling of structured data
  - Allows us to analyze big data and make conclusions based on statistical theories.
  - Can clean messy data sets, and make them readable
  - Pandas able to delete rows that are not relevant, or contains wrong values, like empty or NULL values.
    - **cleaning the data.**
- Pandas gives you answers about the data. Like:
  - Is there a correlation between two or more columns?
  - What is average value?
  - Max value? Min value?

Pandas Installation

If you have **Python** and **PIP** already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install pandas
```

Pandas analyzing Data

- Viewing the Data
  - The `head()` method returns the headers and a specified number of rows, starting from the top.
  - The `tail()` method returns the headers and a specified number of rows, starting from the bottom.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv')
print(df.head(10))
# print(df.tail())
```

Note: if the number of rows is not specified, the `head()` method will return the top 5 rows.

Pandas analyzing Data

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
```

Tip: use `to_string()` to print the entire DataFrame.

Note: If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:

```
import pandas as pd
print(pd.options.display.max_rows)
# pd.options.display.max_rows = 9999
```

Pandas analyzing Data

- Info About the Data
  - The DataFrames object has a method called `info()` - gives you more information about the data set.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10592 entries, 0 to 10591
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     10592 non-null  object
 1   Tita     10592 non-null  float64
 2   Kocha    10592 non-null  object
 3   Ancharo  10592 non-null  float64
 4   Chefa    10590 non-null  float64
 5   Kemesie  10590 non-null  float64
 6   Majetie  10592 non-null  float64
 7   Dessie   10592 non-null  float64
dtypes: float64(6), object(2)
memory usage: 662.1+ KB
None
```

Pandas analyzing Data

- Null Values
  - info() method also tells us how many Non-Null values there are present in each column.
    - Chefa - 10590
    - Tita - 10592

Empty values, or Null values, can be bad when analyzing data, and you should consider **removing** rows with empty values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10592 entries, 0 to 10591
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     10592 non-null  object
 1   Tita     10592 non-null  float64
 2   Kocha    10592 non-null  object
 3   Ancharo  10592 non-null  float64
 4   Chefa    10590 non-null  float64
 5   Kemesie  10590 non-null  float64
 6   Majetie  10592 non-null  float64
 7   Dessie   10592 non-null  float64
dtypes: float64(6), object(2)
memory usage: 662.1+ KB
None
```

Pandas cleaning Data

- Data cleaning - means fixing bad data in your data set.
- Bad data could be: Empty cells, Data in wrong format, Wrong data, Duplicates

Empty values, or Null values, can be bad when analyzing data, and you should consider **removing** rows with empty values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10592 entries, 0 to 10591
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     10592 non-null  object
 1   Tita     10592 non-null  float64
 2   Kocha    10592 non-null  object
 3   Ancharo  10592 non-null  float64
 4   Chefa    10590 non-null  float64
 5   Kemesie  10590 non-null  float64
 6   Majetie  10592 non-null  float64
 7   Dessie   10592 non-null  float64
dtypes: float64(6), object(2)
memory usage: 662.1+ KB
None
```

Pandas - Cleaning Empty Cells

- **Empty cells** - can potentially give you a wrong result when you analyze data.
- Solutions
  - Remove Rows
  - Replace Empty Values
  - Replace Using Mean, Median, or Mode

Pandas - Cleaning Empty Cells

- Other Solutions
  - **Imputation Using Predictive Modeling:** Use algorithms (e.g., **KNN**, **Random Forest**, **Linear Regression**) to predict missing values based on other features.
  - **Interpolation -** especially for time series data. Fill in missing values by **interpolating** (linear, spline, or polynomial interpolation).
  - **Forward Fill / Backward Fill -** Use the **previous (forward fill)** or **next (backward fill)** available value in the dataset (common in time-series).
  - **Multiple Imputation-** A more statistically robust method using **Monte Carlo techniques** to generate several possible values and average them.
  - **Use Domain-Specific Constants -** Fill missing values with a **domain-specific constant** (like 0, "Unknown", or -999) if it makes sense contextually.
  - **Leave as Missing -** Some models (like **decision trees or XGBoost**) can handle missing values internally.
  - **Use a Flag Variable -** Create a **new binary feature** indicating if a value was missing, then impute using another method. Helps retain information about missingnes

Pandas - Cleaning Empty Cells

- Remove Rows - One way to deal with empty cells is to remove rows that contain empty cells.
  - It's usually OK, since data sets can **be very big,** and removing a few rows will not have a big impact on the result.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
new_new = df.dropna()
print(new_df.to_string())
```

Note: By default, the `dropna()` method returns a *new* DataFrame, and will not change the original.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
new_new = df.dropna()
df.dropna(inplace = True)
print(new_df.to_string())
```

Note: Now, the `dropna(inplace = True)` will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

Pandas - Cleaning Empty Cells

- Replace Empty Values- We do not have to delete entire rows just because of some empty cells.
  - The fillna() method allows us to replace empty cells with a value:

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df.fillna(130, inplace = True)
print(new_df.to_string())
```

The example above replaces all empty cells in the whole Data Frame.

Pandas - Cleaning Empty Cells

- Replace Only For Specified Columns
  - To only replace empty values for **one column**, specify the ***column name*** for the DataFrame

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df["Kemese"].fillna(130, inplace = True)
#df["Calories"].fillna(130, inplace = True)
```

Pandas - Cleaning Empty Cells

- Replace Using Mean, Median, or Mode
  - A common way to replace empty cells, is to calculate the mean, median or mode value of the column.
  - Pandas uses the `mean()` `median()` and `mode()` methods to calculate the respective values for a specified column.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
x = df["Kemese"].mean()
#x = df["Kemese"].mode()
#x = df["Kemese"].mode()
df["Kemese"].fillna(x, inplace = True)
```

Mean = the average value (the sum of all values divided by number of values).

Median = the value in the middle, after you have sorted all values ascending.

Mode = the value that appears most frequently.

Pandas - Data of Wrong Format

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data. To fix it, you have two options:
  - remove the rows, or
  - convert all cells in the columns into the same format.

Convert Into a Correct Format

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

21	60	'2020/12/21'	108	131	364.2
22	45	NaN	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
28	60	'2020/12/28'	103	132	NaN

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df.dropna(subset=['Date'], inplace = True)
```

21	60	'2020/12/21'	108	131	364.2
22	45	NaT	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	'2020/12/26'	100	120	250.0
27	60	'2020/12/27'	92	118	241.0

Pandas - Fixing Wrong Data

- "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".

Replacing Values

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
# df.loc[7, Kemese] = 45
for x in df.index:
  if df.loc[x, "Kemese"] > 120:
    df.loc[x, "Kemese"] = 120
```

Removing Rows

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Empty Pandas - Removing Duplicates

- Duplicate rows - rows that have been registered more than one time.

Discovering Duplicates

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
print(df.duplicated())
```

Removing Duplicates

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df.drop_duplicates(inplace = True)
```

Pandas - Data Correlations

- The corr() method calculates the relationship between each column in your data set.

```
import pandas as pd
df = pd.read_csv('Python_Rainfall_Data.csv)
df.corr()
```

	Tita	Ancharo	Chefa	Kemesie	Majetie	Dessie
Tita	1.000000	0.302148	0.279438	0.186577	0.268659	0.483619
Ancharo	0.302148	1.000000	0.271477	0.194156	0.236401	0.315109
Chefa	0.279438	0.271477	1.000000	0.376196	0.426238	0.235934
Kemesie	0.186577	0.194156	0.376196	1.000000	0.749582	0.167392
Majetie	0.268659	0.236401	0.426238	0.749582	1.000000	0.216949
Dessie	0.483619	0.315109	0.235934	0.167392	0.216949	1.000000

The number varies from -1 to 1.

- 1 means - there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.
- 0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.
- -0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.
- 0.2 means - NOT a good relationship, meaning that if one value goes up does not mean that the other will.

Merge multiple Excel files into one

```
import pandas as pd
import os

folder_path = 'path/to/your/excels/'
all_data = pd.DataFrame()

for file in os.listdir(folder_path):
        if file.endswith('.xlsx'):
            df = pd.read_excel(os.path.join(folder_path, file))
            all_data = pd.concat([all_data, df], ignore_index=True)
 all_data.to_excel('MergedFile.xlsx', index=False)
```

Split Dataframe

- Split by Columns

```
import pandas as pd
data = {'col1': [1, 2, 3, 4], 'col2': [5, 6, 7, 8], 'col3': [9, 10, 11, 12]}
df = pd.DataFrame(data)
df1 = df[['col1', 'col2']]
df2 = df[['col3']]
```

Split Dataframe

- Split By Rows

```
import pandas as pd
df1 = df[0:2] #First two rows
df2 = df[2:] #Remaining rows

df1 = df.sample(frac=0.5) #Randomly sample 50% of rows
df2 = df.drop(df1.index) #Remaining rows
```

Split Dataframe

- Split By **Column Value**

```
df1 = df[df['col1'] > 2]
df2 = df[df['col1'] <= 2]
```

Split Dataframe

- Using `str.split()`

```
data = {'col1': ['a b c', 'd e f', 'g h i']}
df = pd.DataFrame(data)


df[['col2', 'col3', 'col4']] = df['col1'].str.split(' ', expand=True)
```

Split Dataframe

- **Train-test split**

```
from sklearn.model_selection import train_test_split


X = df.drop('target_column', axis=1)
y = df['target_column']
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp,
y_temp, test_size=0.5, random_state=42)
```

Pandas and Matplotlib Plotting

Installation of Matplotlib

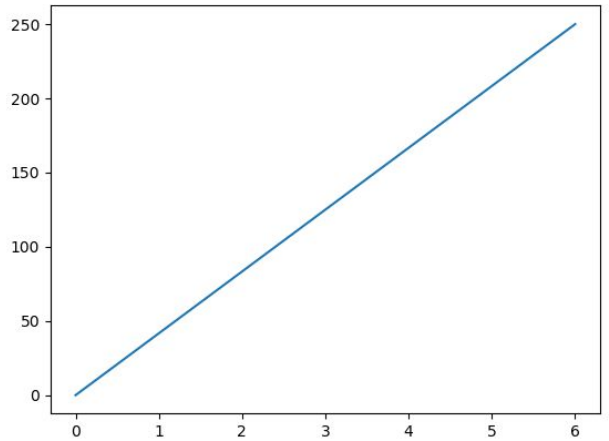
pip install matplotlib

```
import matplotlib
print(matplotlib.__version__)
```

Most of the Matplotlib utilities lies under the `pyplot` submodule, and are usually imported under the `plt` alias:

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

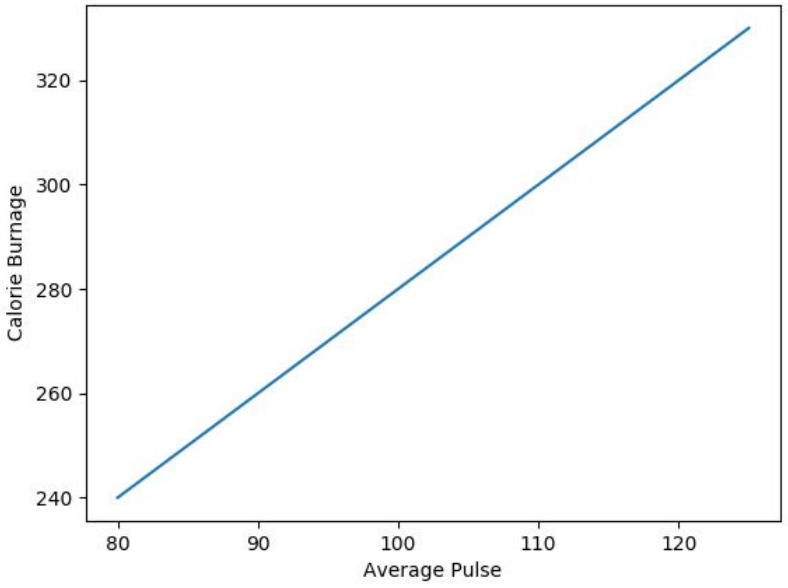
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



Pandas and Matplotlib Plotting

Label of Matplotlib

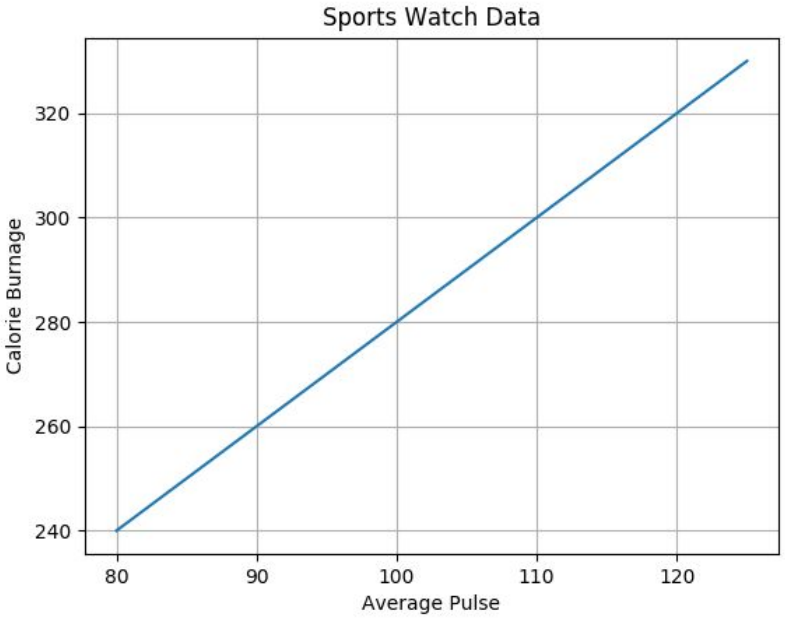
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100,
105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270,
280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



Pandas and Matplotlib Plotting

Grid of Matplotlib

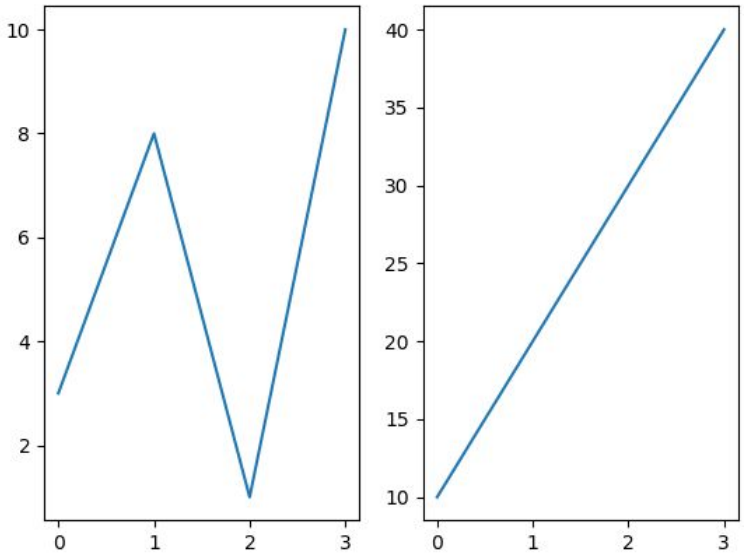
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110,
115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300,
310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```



Pandas and Matplotlib Plotting

Display Multiple Plots

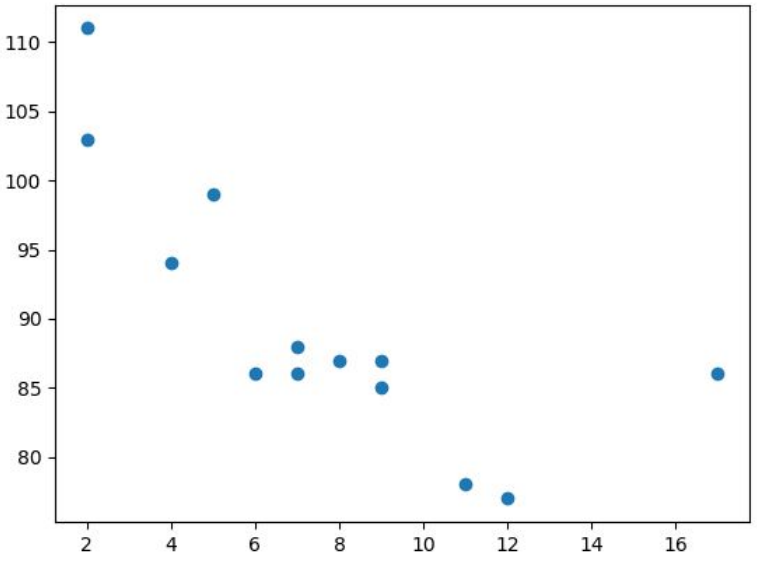
```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



Pandas and Matplotlib Plotting

Matplotlib Scatter

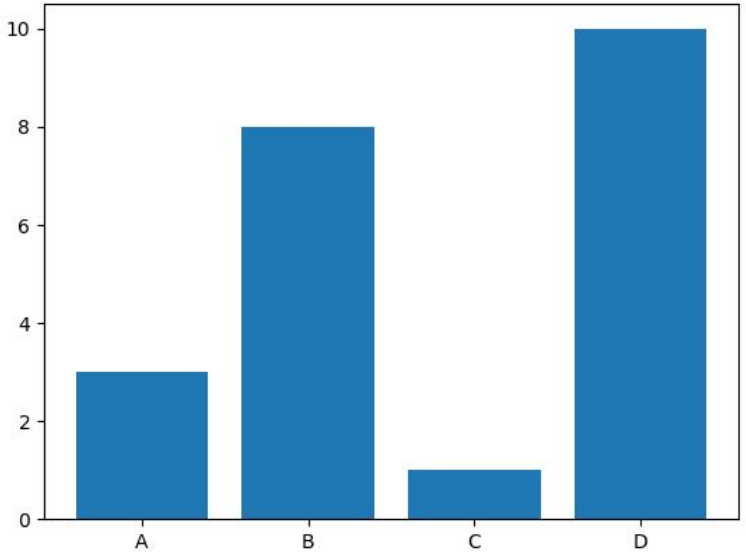
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,
86])

plt.scatter(x, y)
plt.show()
```



Pandas and Matplotlib Plotting

Matplotlib Bars

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

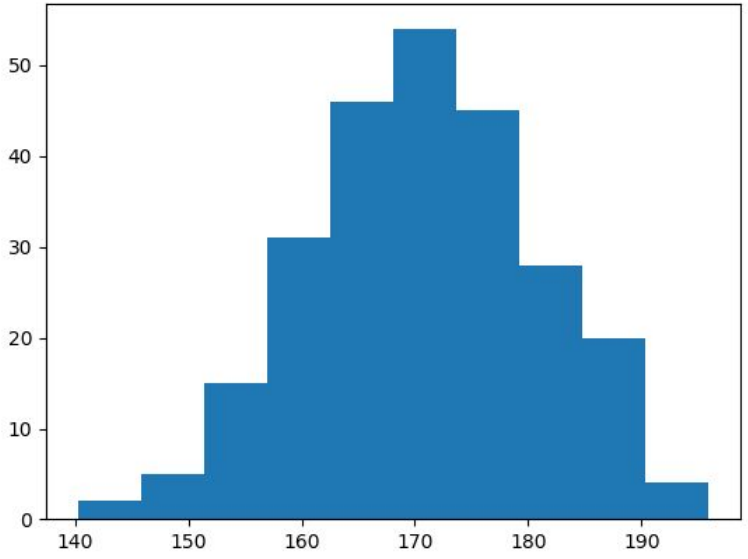


Pandas and Matplotlib Plotting

Matplotlib Histograms

A histogram is a graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.

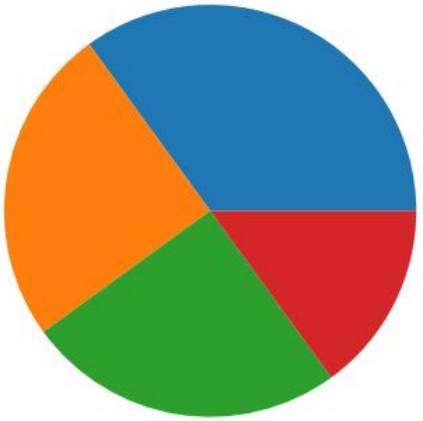



```
import matplotlib.pyplot as plt

import numpy as np


x = np.random.normal(170, 10, 250)


plt.hist(x)

plt.show()
```

Pandas and Matplotlib Plotting

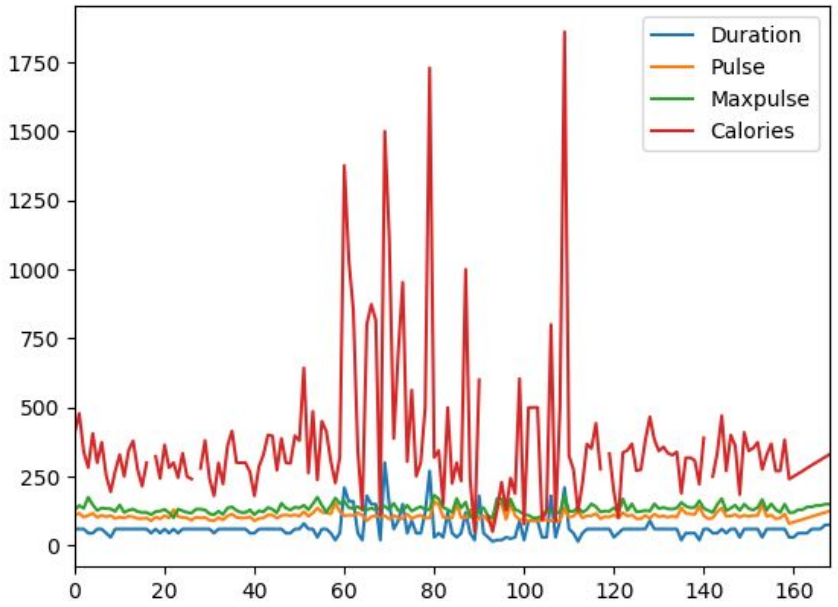
Pie Charts

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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



Pandas and Matplotlib Plotting

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
# df = pd.read_csv('Python_Rainfall_Data.csv)
df.plot()
plt.show()
```

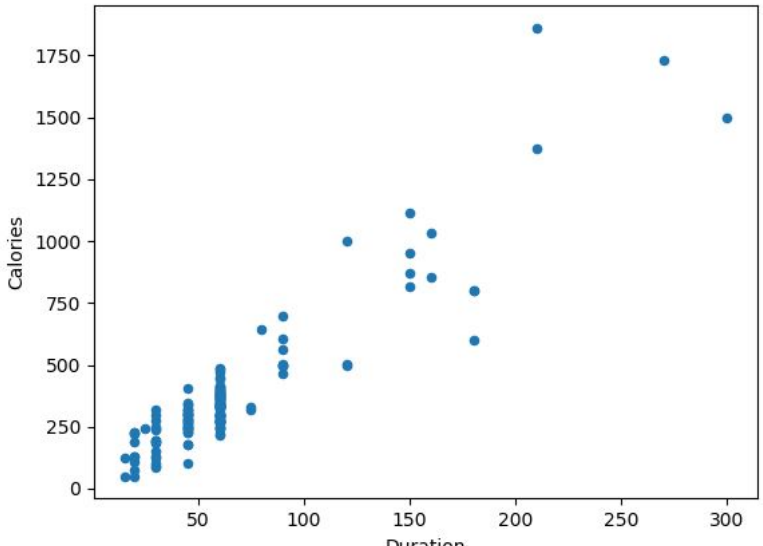


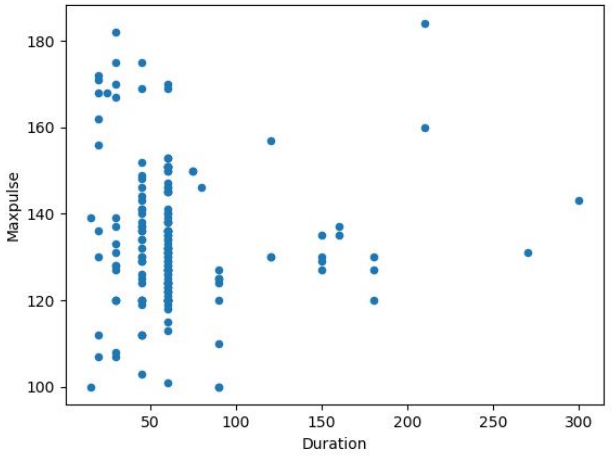
Pandas Plotting

Scatter Plot

Tell you want a scatter plot with the `kind` argument: `kind = 'scatter'` A scatter plot needs an x- and a y-axis.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
# df = pd.read_csv('Python_Rainfall_Data.csv)
df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')
# df.plot(kind = 'scatter', x = 'Duration', y = 'Maxpulse')
plt.show()
```





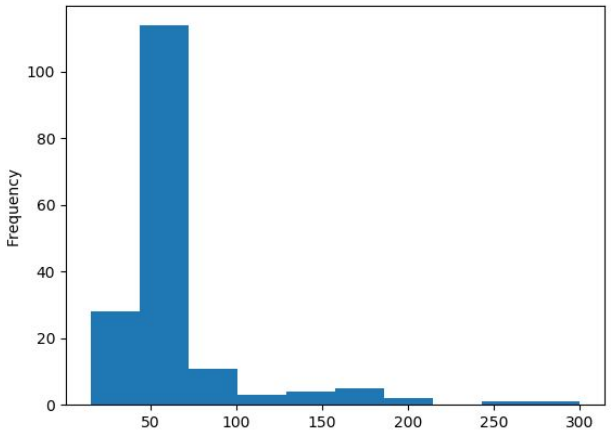
Pandas Plotting

Histogram Plot

Use the `kind` argument to specify that you want a histogram: `kind = 'hist'`

A histogram needs only one column.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
# df = pd.read_csv('Python_Rainfall_Data.csv)
df["Duration"].plot(kind = 'hist')
plt.show()
```



Example - Data Loading

```
#importing useful libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
%matplotlib inline
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Example - read csv

- ...

```
#importing useful libraries
#reading csv
health_camp_detail=pd.read_csv('../input/health-care-analytics/Health_Camp_Detail.csv')
patient_profile=pd.read_csv('../input/health-care-analytics/Patient_Profile.csv')
fhc=pd.read_csv('../input/health-care-analytics/First_Health_Camp_Attended.csv')
shc=pd.read_csv('../input/health-care-analytics/Second_Health_Camp_Attended.csv')
thc=pd.read_csv('../input/health-care-analytics/Third_Health_Camp_Attended.csv')
train=pd.read_csv('../input/health-care-analytics/Train.csv')
test=pd.read_csv('../input/health-care-analytics/test.csv')
```

Example - read csv

- **Health Camp Detail**

health_camp_detail.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Health_Camp_ID   65 non-null     int64
 1   Camp_Start_Date  65 non-null     object
 2   Camp_End_Date    65 non-null     object
 3   Category1        65 non-null     object
 4   Category2        65 non-null     object
 5   Category3        65 non-null     int64
dtypes: int64(2), object(4)
memory usage: 3.2+ KB
```

Example - read csv

- **Health Camp Detail**

health_camp_detail.head(2)

	Health_Camp_ID	Camp_Start_Date	Camp_End_Date	Category1	Category2	Category3
0	6560	16-Aug-03	20-Aug-03	First	B	2
1	6530	16-Aug-03	28-Oct-03	First	C	2

Example - Data Preprocessing

- **Data Preprocessing**

```
#converting string date to datetime object
health_camp_detail['Camp_Start_Date']=health_camp_detail['Camp_Start_Date'].ap
ply(lambda x:datetime.strptime(x,'%d-%b-%y'))
health_camp_detail['Camp_End_Date']=health_camp_detail['Camp_End_Date'].appl
y(lambda x:datetime.strptime(x,'%d-%b-%y'))

#adding suffix for easy identification during one-hot encoding
health_camp_detail['Category1']=health_camp_detail['Category1']+'_cat1'
health_camp_detail['Category2']=health_camp_detail['Category2']+'_cat2'
health_camp_detail['Category3']=health_camp_detail['Category3'].apply(lambda
x:str(x)+'_cat3')
```

Example - Data Preprocessing

- **Data Preprocessing**

health_camp_detail.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Health_Camp_ID   65 non-null     int64
 1   Camp_Start_Date  65 non-null     datetime64[ns]
 2   Camp_End_Date    65 non-null     datetime64[ns]
 3   Category1        65 non-null     object
 4   Category2        65 non-null     object
 5   Category3        65 non-null     object
dtypes: datetime64[ns](2), int64(1), object(3)
memory usage: 3.2+ KB
```

Example - Data Preprocessing

- **Data Preprocessing**

health_camp_detail.head(2)

	Health_Camp_ID	Camp_Start_Date	Camp_End_Date	Category1	Category2	Category3
0	6560	2003-08-16	2003-08-20	First_cat1	B_cat2	2_cat3
1	6530	2003-08-16	2003-10-28	First_cat1	C_cat2	2_cat3

Example - Data Preprocessing

- **Data Preprocessing**

```
health_camp_detail['Health_Camp_ID'].value_counts()
```

```
6527    1
6559    1
6557    1
6556    1
6555    1
       ..
6565    1
6564    1
6563    1
6562    1
6528    1
Name: Health_Camp_ID, Length: 65, dtype: int64
```

Example - Data Preprocessing

- **Data Preprocessing**

```
sum(health_camp_detail['Health_Camp_ID'].value_counts()>1)

# No duplicate health camps, should not cause any issue during join
```

```
health_camp_detail[health_camp_detail['Camp_End_Date']<health_camp_detail['Camp_Start_Date']]
# No instances of campaign end date < camp start date
```

	Health_Camp_ID	Camp_Start_Date	Camp_End_Date	Category1	Category2	Category3

Example - Feature Engineering

- **Data Preprocessing**

```
#Camp_Start_Date
health_camp_detail['Camp_Start_Month']=health_camp_detail['Camp_Start_Date'].apply(lambda x:x.month)
health_camp_detail['Camp_Start_Day']=health_camp_detail['Camp_Start_Date'].apply(lambda x:x.day)
health_camp_detail['Camp_Start_Quarter']=health_camp_detail['Camp_Start_Date'].apply(lambda x:x.quarter)

#Camp_End_Date
health_camp_detail['Camp_End_Month']=health_camp_detail['Camp_End_Date'].apply(lambda x:x.month)
health_camp_detail['Camp_End_Day']=health_camp_detail['Camp_End_Date'].apply(lambda x:x.day)
health_camp_detail['Camp_End_Quarter']=health_camp_detail['Camp_End_Date'].apply(lambda x:x.quarter)

#Camp_Duration
health_camp_detail['Camp_Duration']=(health_camp_detail['Camp_End_Date']-health_ca
```

Example - Feature Engineering

- **Data Preprocessing**

health_camp_detail.head(2)

	Health_Camp_ID	Camp_Start_Date	Camp_End_Date	Category1	Category2	Category3	Camp_Start_Month
0	6560	2003-08-16	2003-08-20	First_cat1	B_cat2	2_cat3	8
1	6530	2003-08-16	2003-10-28	First_cat1	C_cat2	2_cat3	8

Example - Feature Engineering

- **Data Preprocessing**

```
#Creating Dummies
#Category1
category1_dummies = pd.get_dummies(health_camp_detail['Category1'],drop_first=True)
health_camp_detail =
pd.concat([health_camp_detail.drop('Category1',axis=1),category1_dummies],axis=1)

#Category2
category2_dummies = pd.get_dummies(health_camp_detail['Category2'],drop_first=True)
health_camp_detail =
pd.concat([health_camp_detail.drop('Category2',axis=1),category2_dummies],axis=1)

#Category3
category3_dummies = pd.get_dummies(health_camp_detail['Category3'],drop_first=True)
health_camp_detail =
pd.concat([health_camp_detail.drop('Category3',axis=1),category3_dummies],axis=1)

#Weekends
health_camp_detail['weekends_during_campaign']=[pd.date_range(x,y).weekday.isin([5,6]).sum() for x , y
in zip(health_camp_detail['Camp_Start_Date'],health_camp_detail['Camp_End_Date'])]
```

Example - Feature Engineering

- **Data Preprocessing**

health_camp_detail.head(2)

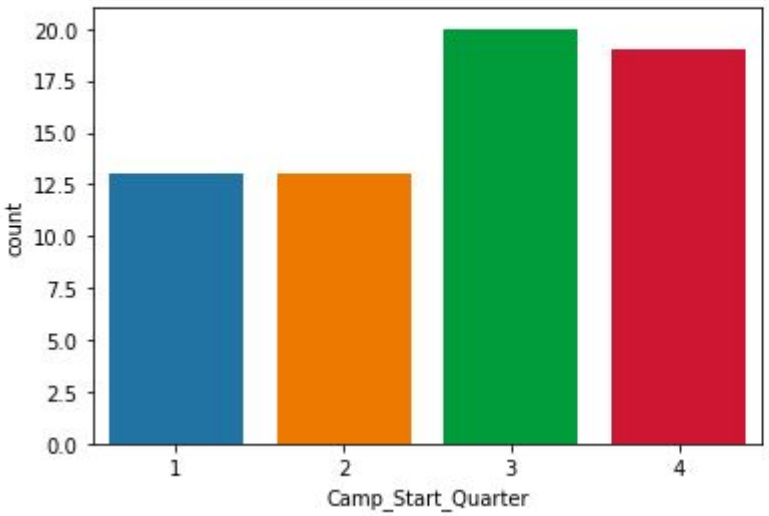
	Health_Camp_ID	Camp_Start_Date	Camp_End_Date	Camp_Start_Month	Camp_Start_Day	Camp_Start_Quarte
0	6560	2003-08-16	2003-08-20	8	16	3
1	6530	2003-08-16	2003-10-28	8	16	3

Example - Exploratory Data Analysis

- **Exploratory Data Analysis**

```
#camps by quarter
sns.countplot('Camp_Start_Quarter',data=health_camp_detail)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f52ceaf72d0>
```

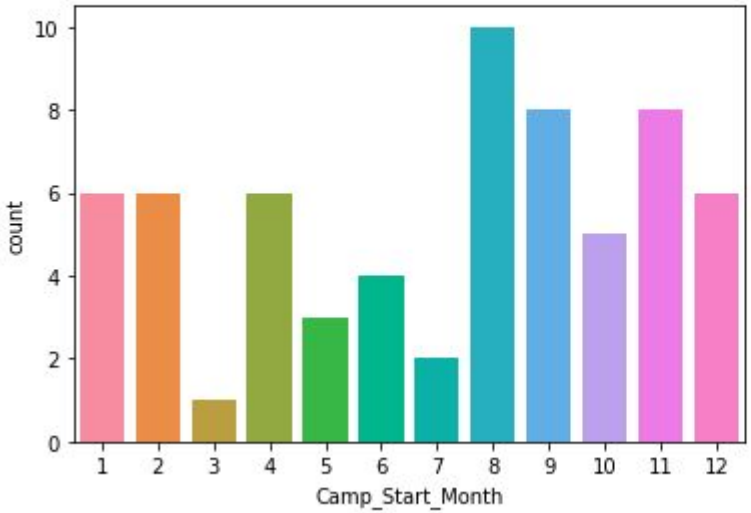


Example - Exploratory Data Analysis

- **Exploratory Data Analysis**

```
#camps by month
sns.countplot('Camp_Start_Month',data=health_camp_detail)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f52cc1c38d0>
```

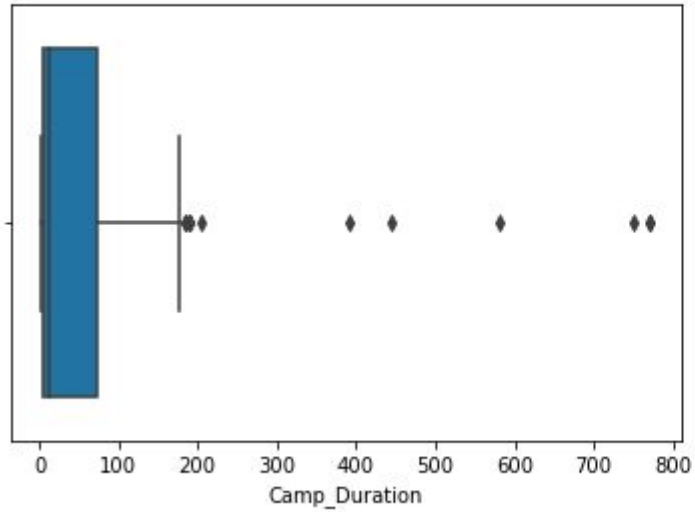


Example - Exploratory Data Analysis

- **Exploratory Data Analysis**

```
#camp duration distribution
sns.boxplot(health_camp_detail['Camp_Duration'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f52cc15ded0>
```

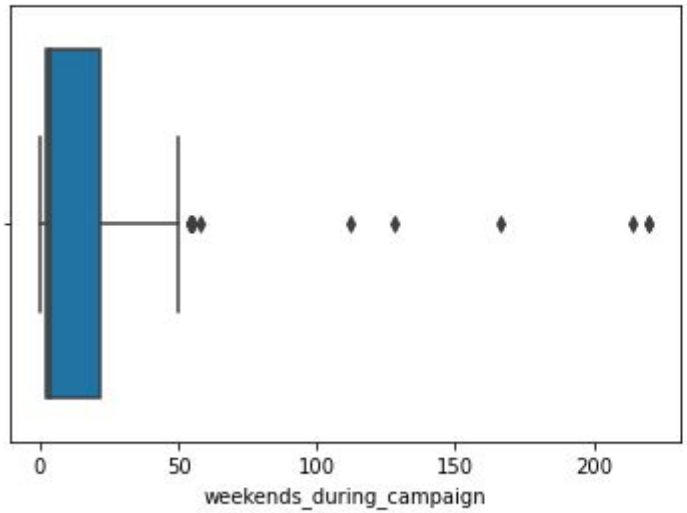


Example - Exploratory Data Analysis

- **Exploratory Data Analysis**

```
#camp weekends distribution
sns.boxplot(health_camp_detail['weekends_during_campaign'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f52cc0e1d90>
```



Example - Exploratory Data Analysis

- **Patient Profile**

patient_profile.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37633 entries, 0 to 37632
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Patient_ID          37633 non-null  int64
 1   Online_Follower     37633 non-null  int64
 2   LinkedIn_Shared     37633 non-null  int64
 3   Twitter_Shared      37633 non-null  int64
 4   Facebook_Shared     37633 non-null  int64
 5   Income              37633 non-null  object
 6   Education_Score     37633 non-null  object
 7   Age                 37633 non-null  object
 8   First_Interaction   37633 non-null  object
 9   City_Type           14249 non-null  object
 10  Employer_Category   2840 non-null   object
dtypes: int64(5), object(6)
memory usage: 3.2+ MB
```

patient_profile.head(2)

	Patient_ID	Online_Follower	LinkedIn_Shared	Twitter_Shared	Facebook_Shared	Income	Education_Score	A
0	516956	0	0	0	0	1	90	3
1	507733	0	0	0	0	1	None	4

Thanks