Supervised Learning Report

Assignments 2A & 2B

Rishabh Kaushick
NU ID: 002808996
College of Engineering
Northeastern University
Toronto, ON
kaushick.r@northeastern.edu

Assignment 2A

Abstract

In this report, I explored two different supervised learning algorithms – Decision Trees and Artificial Neural Networks, on two different datasets. I understood how overfitting occurs and made corrective measures to the model to better generalize the data. I also got an understanding of the working of neural networks.

Datasets

For this assignment, I used two datasets from Kaggle as mentioned in Table 1 below.

Dataset	Name	Dataset Characteristics	Attribute Characteristics	Associated Task	Number of Instances	Number of Attributes
1	NASA Near Earth Objects (NEOs)	Multivariate	Real	Classification	904	27
2	Heart Attack Risk Prediction	Multivariate	Real	Classification	8764	25

Table 1: Features of both datasets

Data Characteristics

Analyzing the data in the output variable to understand the class frequency. Based on Figure 1 below, we can see that there is 2 times more data with values as 'False' as compared to 'True' in the 'Is Potentially Hazardous' attribute for Dataset #1. The Figure 1 also shows us that this attribute has only 2 types of data 'True' or 'False', and that there are no null values or other bad data.

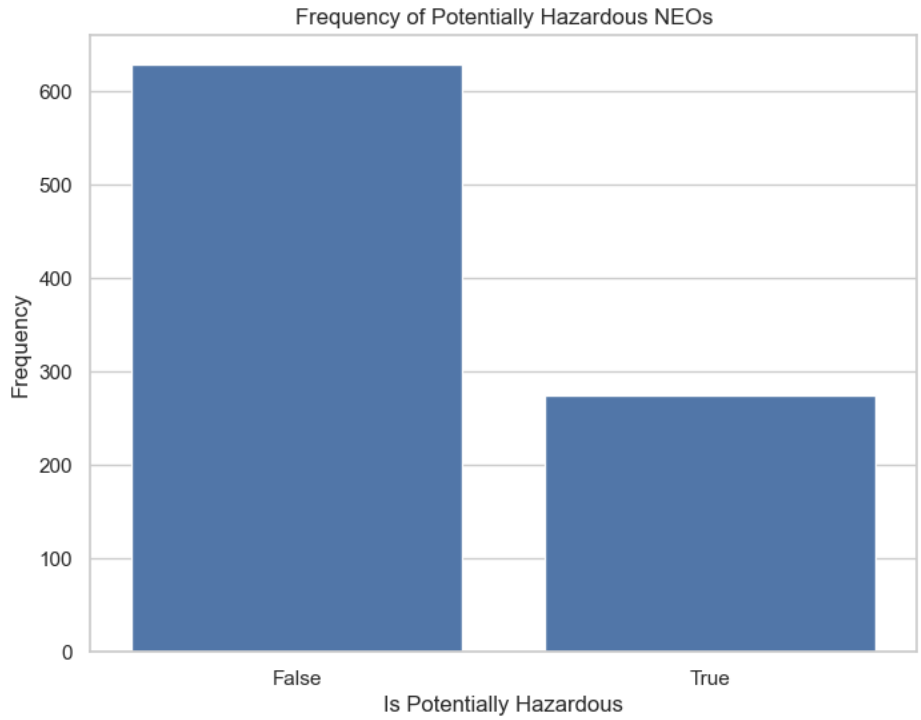


Figure 1: Frequency distribution of the values in the 'Is Potentially Hazardous' attribute.

Similarly Figure 2 depicts that the frequency of the people with no heart attack risk (denoted in the figure by '0'), is significantly more than the frequency of people with a heart attack risk (denoted in the figure by '1').

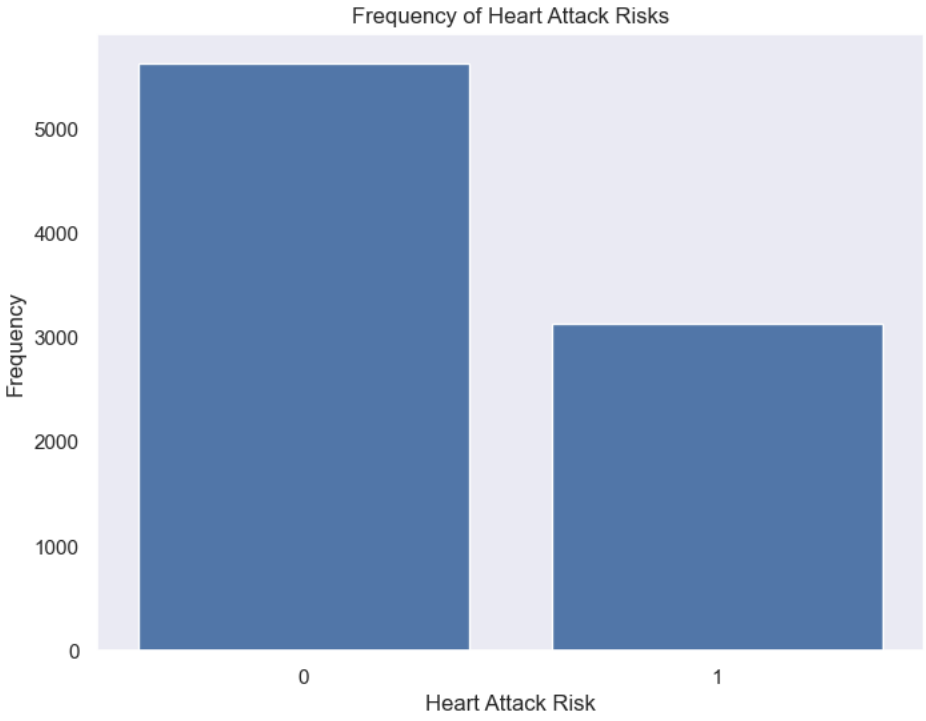


Figure 2: Frequency distribution of the values in the 'Heart Attack Risk' attribute.

Why I Find the Datasets Interesting

Both the datasets were chosen with different aspects in mind. As an Astro-Physics enthusiast, the first one containing the data about the Near-Earth Objects (NEOs) was strikingly interesting. Earth, the only known planet with life, must be protected from threats including foreign objects in our galaxy. The data of NEOs collected by NASA can help for us to be prepared and devise solutions if an asteroid or another natural satellite approaches in a potentially hazardous route towards the Earth. The second dataset was chosen because it aligns with my values to positively give back to the society. Unlike the thread of NEOs perhaps many light years away, the threat of a heart attack is much more urgent. Machine Learning can aid doctors to be largely able to detect those patients having a high risk of heart attacks to take preventive measures.

Diving Into Dataset #1

Understanding & Visualizing the Data

The data has a total of 26 attributes. From the info method we can see that there are a total of 904 rows. However, when we check for how many non-null unique values are present in each attribute, we can see that there are 20 unique NEOs, 5 unique Orbiting Bodies, 2 unique values in Potentially Hazardous NEOs and 904 unique values in most of the other columns except for 'Close Approach Date'. For this column we can see 903 values, which means one row contains a null data.

```
neo_df.info()
[4]  ✓ 0.0s
...  Data columns (total 27 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ID                              904 non-null    int64
 1   Neo Reference ID                904 non-null    int64
 2   Name                            904 non-null    object
 3   Limited Name                    904 non-null    object
 4   Designation                     904 non-null    int64
 5   NASA JPL URL                    904 non-null    object
 6   Absolute Magnitude (H)          904 non-null    float64
 7   Min Diameter (km)               904 non-null    float64
 8   Max Diameter (km)               904 non-null    float64
 9   Min Diameter (m)                904 non-null    float64
 10  Max Diameter (m)                904 non-null    float64
 11  Min Diameter (miles)            904 non-null    float64
 12  Max Diameter (miles)            904 non-null    float64
 13  Min Diameter (feet)             904 non-null    float64
 14  Max Diameter (feet)             904 non-null    float64
 15  Is Potentially Hazardous        904 non-null    bool
 16  Close Approach Date             904 non-null    object
 17  Close Approach Date (Full)      904 non-null    object
 18  Epoch Date Close Approach       904 non-null    int64
 19  Relative Velocity (km/s)        904 non-null    float64
 20  Relative Velocity (km/h)        904 non-null    float64
 21  Relative Velocity (miles/h)     904 non-null    float64
 22  Miss Distance (astronomical)    904 non-null    float64
 23  Miss Distance (lunar)           904 non-null    float64
 24  Miss Distance (km)              904 non-null    float64
 25  Miss Distance (miles)           904 non-null    float64
 26  Orbiting Body                   904 non-null    object
```

```
neo_df.nunique()
[20]  ✓ 0.0s
...  ID                              20
     Neo Reference ID                20
     Name                            20
     Limited Name                    20
     Designation                     20
     NASA JPL URL                    20
     Absolute Magnitude (H)          20
     Min Diameter (km)               20
     Max Diameter (km)               20
     Min Diameter (m)                20
     Max Diameter (m)                20
     Min Diameter (miles)            20
     Max Diameter (miles)            20
     Min Diameter (feet)             20
     Max Diameter (feet)             20
     Is Potentially Hazardous         2
     Close Approach Date            903
     Close Approach Date (Full)     904
     Epoch Date Close Approach      904
     Relative Velocity (km/s)       904
     Relative Velocity (km/h)       904
     Relative Velocity (miles/h)    904
     Miss Distance (astronomical)   904
     Miss Distance (lunar)          904
     Miss Distance (km)             904
     Miss Distance (miles)          904
     Orbiting Body                    5
     dtype: int64
```

Figure 3: Attribute information in the dataset.

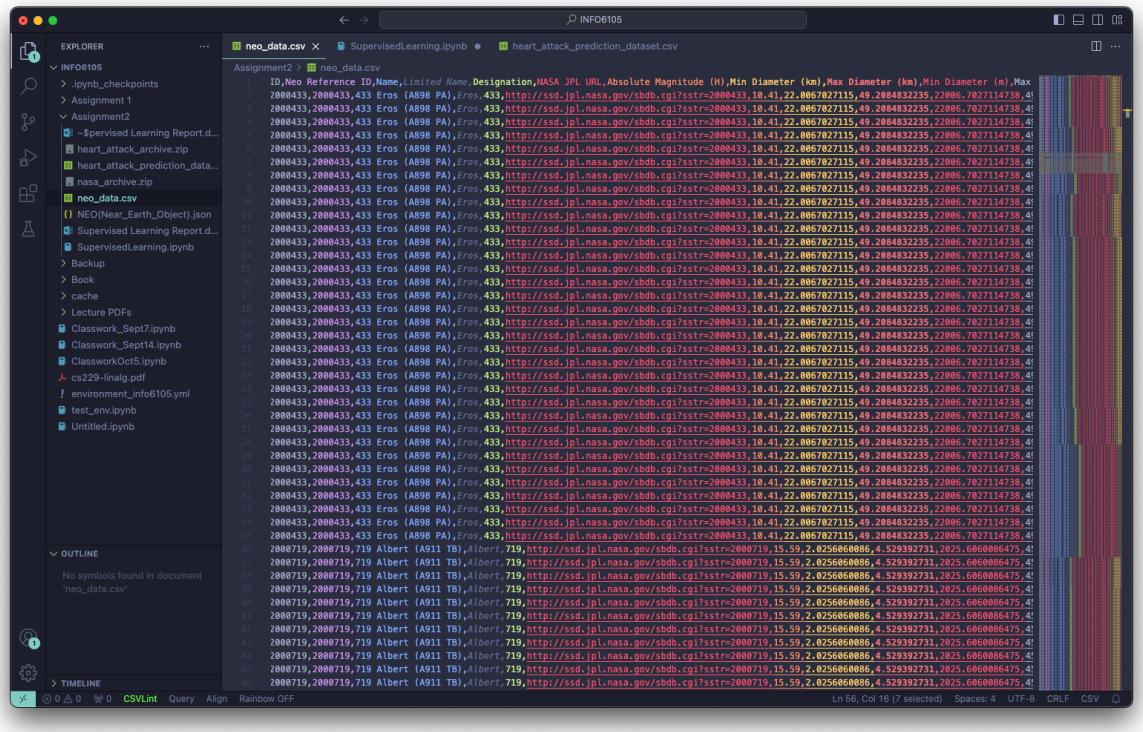


Figure 4: The type of data in the CSV file.

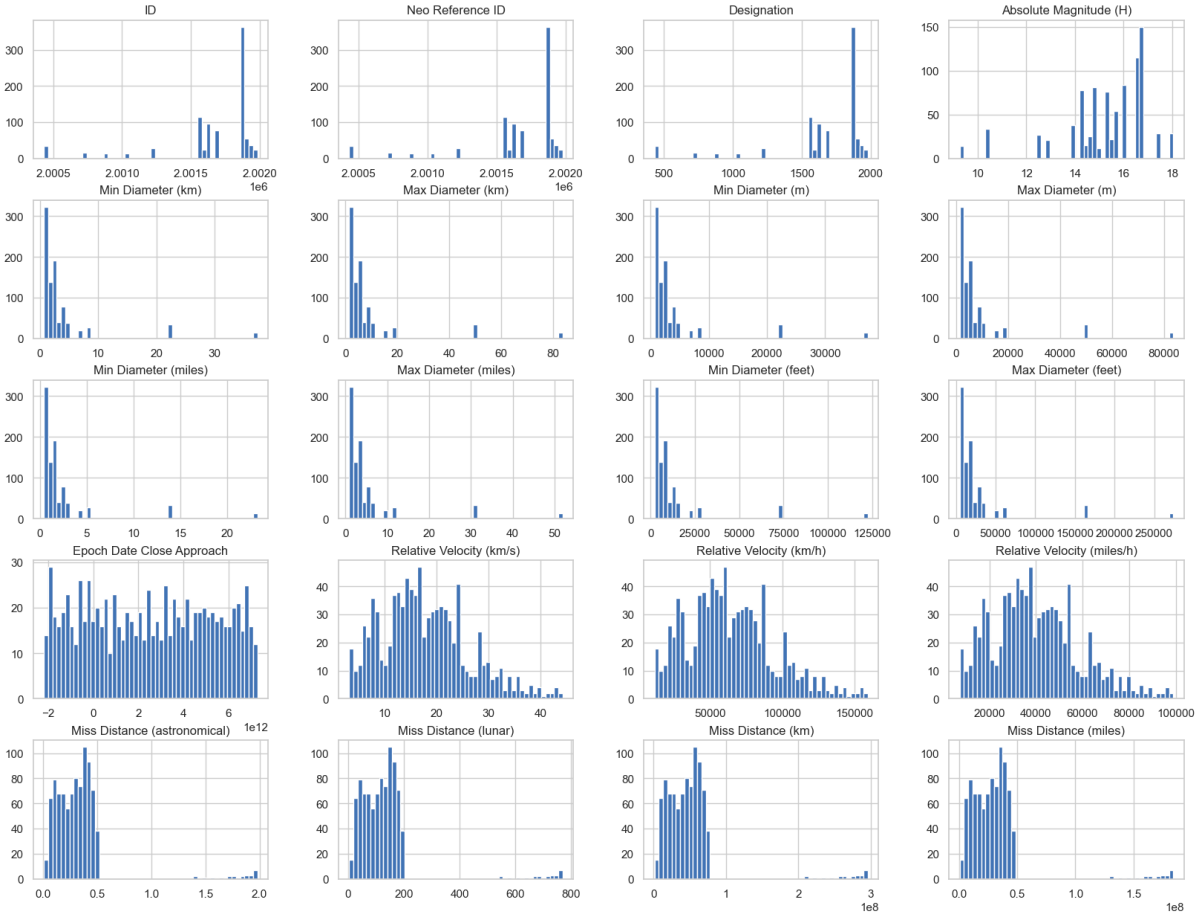


Figure 5: With a graph of the frequency of all the attributes of the NEOs.

Firstly, from Figure 4 above, we can see that the ID, Neo Reference ID, NASA JPL URL, Maximum Diameter, Min Diameter, and a few more columns are the duplicated in each row.

The data shown above as well as the codes below show that the 'ID' and the 'Neo Reference ID' columns both contain the same identical values. Therefore we can drop the 'Neo Reference ID' column.

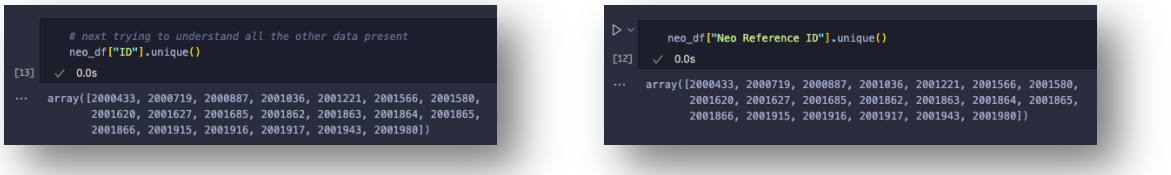


Figure 6: Code shows the unique values in both columns.

There are many more columns which are redundant. For example, we have columns 'Minimum Diameter' and 'Maximum Diameter' four times – in units of kilometers, meters, miles and feet. Therefore, we will use only unit – miles, since it is the largest value comparatively. Similarly, some other columns which hold the same data in different units:

1. Close Approach Date and Close Approach Date (full)
2. Relative Velocity (km/s), Relative Velocity (km/h) and Relative Velocity (miles/h)
3. Miss Distance (astronomical), Miss Distance (lunar), Miss Distance (km), Miss Distance (miles)

After dropping all the similar columns, we are left with the following columns mentioned in Figure 7 below.

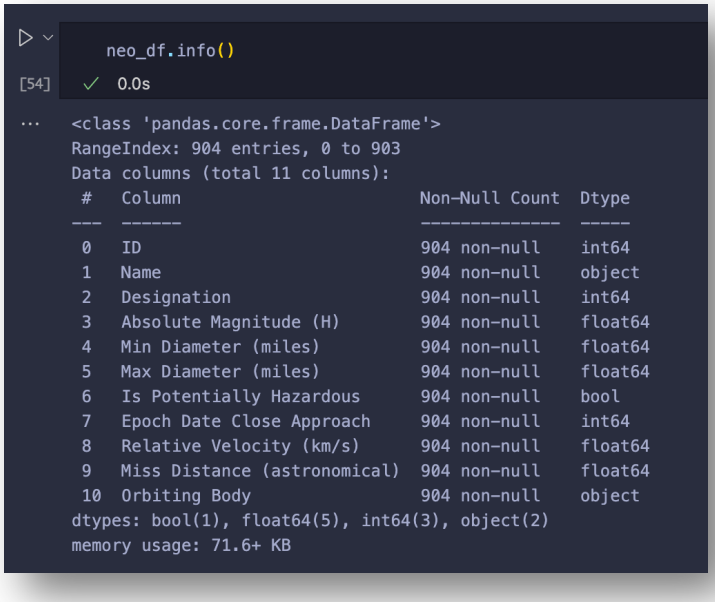


Figure 7: Set of columns remaining after the removal of redundant columns.

Splitting Training & Test Set

Before performing any other analysis on the data, it must be split into training and test sets. This is a crucial step early in the machine learning pipeline, because performing analysis on the entire dataset

has a likely chance to overfit the model. In this specific model, the training and test sets have been split into 70% and 30% respectively.

Visualizing the Data

The figure below shows the Epoch Date Close Approach on the X-axis, and Miss Distance in the Y-axis. The size of each of the data point is dependent on the absolute magnitude of the NEO. Additionally, the color coding refers to if the NEO is potentially hazardous. The red color denotes that it is potentially hazardous, and the blue dots are those which are not potentially hazardous.

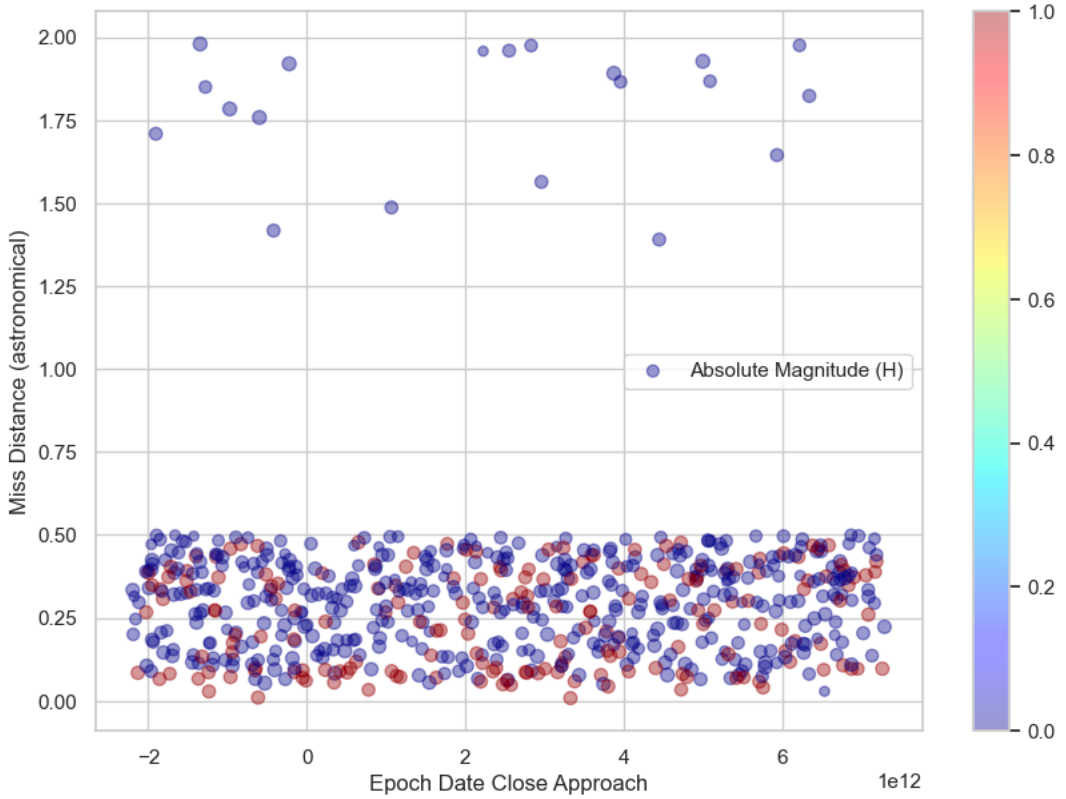


Figure 8: Plot of Epoch Date Close Approach & Miss Distance while considering Absolute Magnitude & whether it is potentially hazardous.

Figure 8 shows that most of the objects have a miss distance of less than 0.5 astronomical units. Also, we can see that those which are much closer to miss distance of 0.00 are potentially hazardous, however those which have a miss distance much farther away are usually not potentially hazardous. However, these generalizations cannot be made because there may be other factors which would decide whether a near earth object is potentially hazardous.

Preparing the Model

Our new training data frame has a total of 633 rows. Based on the Figure 9, we can see that there are a total of 633 rows and none of the columns have any null values. If columns contain null values, we may need to either remove those rows or fill the null values with some dummy values (such as mean or median).

Figure 9: Columns have

Furthermore, the machine learning models cannot understand string values. In our dataset, we have columns such as 'Name' and 'Orbiting Body' which both contain string values. Therefore, we must convert them into some numeric values. For the Orbiting body, based on my intuition, I have set the values for each of the orbiting body to have a value from 0 to 4 based on their size. Therefore:

- *# 0 = Merc*
- *# 1 = Mars*
- *# 2 = Venus*
- *# 3 = Earth*
- *# 4 = Juptr*

Additionally, the values True and False which are present in the 'Is Potentially Hazardous' column must also be converted to 1 and 0 respectively.

Figure 10: Converting True & False to 1 & 0.

Decision Tree Classifier

To create the decision tree classifier and train the data, we must first set our X matrix and y vector. Next, we can fit the Decision Tree classifier:

```
from sklearn.tree import DecisionTreeClassifier
y = neo_trainset["Is Potentially Hazardous"]
X = neo_trainset.drop(columns="Is Potentially Hazardous")

tree_clf = DecisionTreeClassifier(max_depth=10)
tree_clf.fit(X, y)
```

```
[37]  ✓ 0.6s
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10)
```

Figure 11: Training the decision tree/ fitting the model.

Once the decision tree is trained, we can see how the machine has created a decision tree based on the export_graphviz library. The below figure shows how the decision tree has been made:

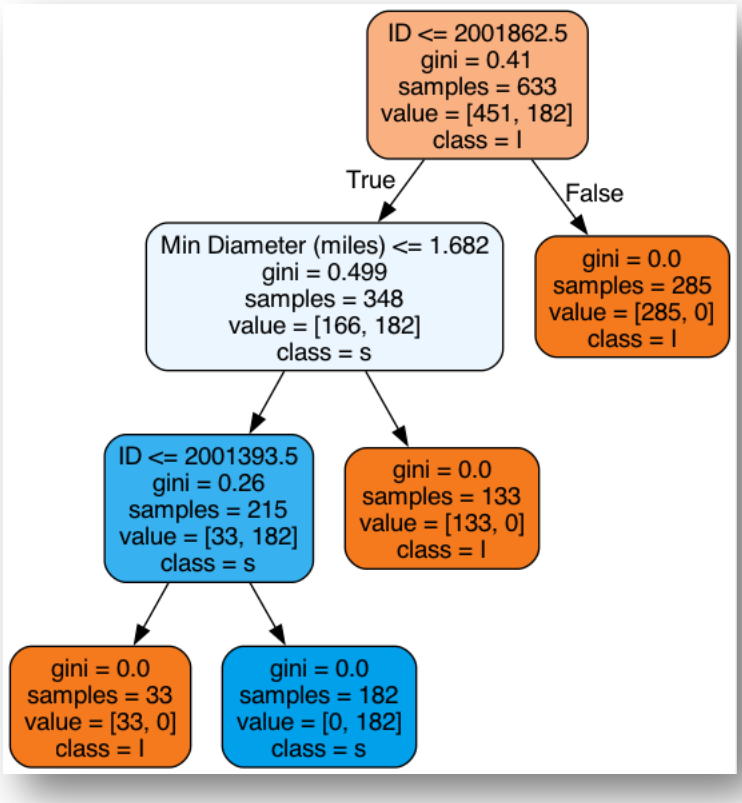


Figure 12: Showing the decision tree made by the system.

We can use this tree to make predictions and then check how well it is performing based on the training dataset. However, to do this, we need to have a way of measuring how well the decision

tree has made a prediction. Therefore, we need an error evaluation criterion. In this case, we can use the root mean squared error.

When we check the mean squared error, we can see that it gives us 0.0. This means that there was no error, and the model correctly predicted all the scenarios for all the data in the training dataset. Therefore, in this case, the model has overfit the data to suit the training dataset and is very likely to have very poor results for the test dataset.

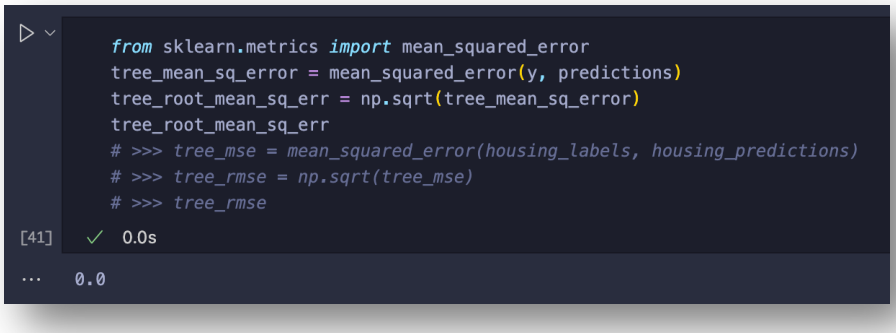


Figure 13: Showing the root means square error is 0.

This could have happened since the dataset is very small. There are only 633 rows in our training dataset, and this covers only 20 unique near-earth objects. Another reason to overfitting could be because the dataset contains a large amount of noisy data.

Restarting the Jupyter notebook kernel and making changes to the decision tree classifier to see if the model has better generalization of the data. In this second attempt the validation and test datasets have been split 80%, and 20% respectively.

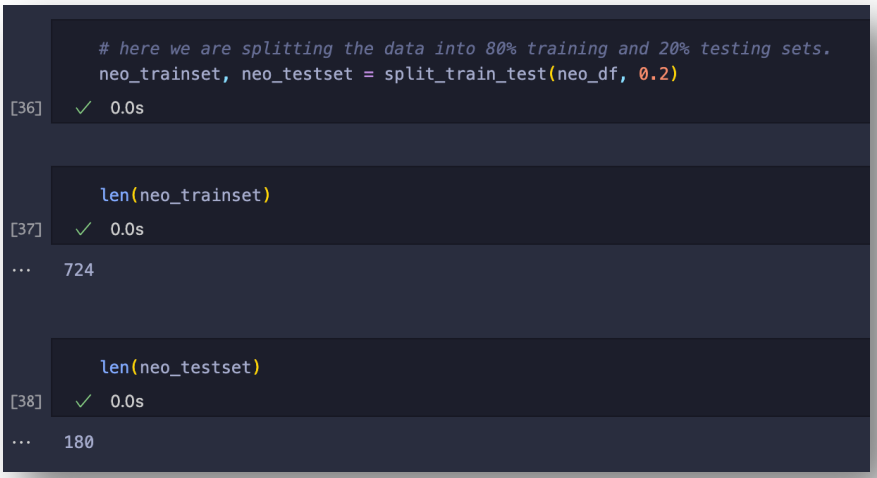


Figure 14: Splitting the data into a trainset and a test set.

Also reducing the maximum depth of the model from 10 to 5, and adding a random state as shown below:

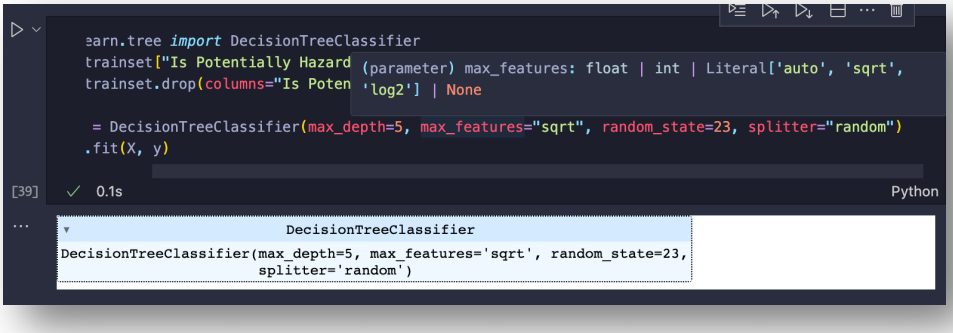


Figure 15

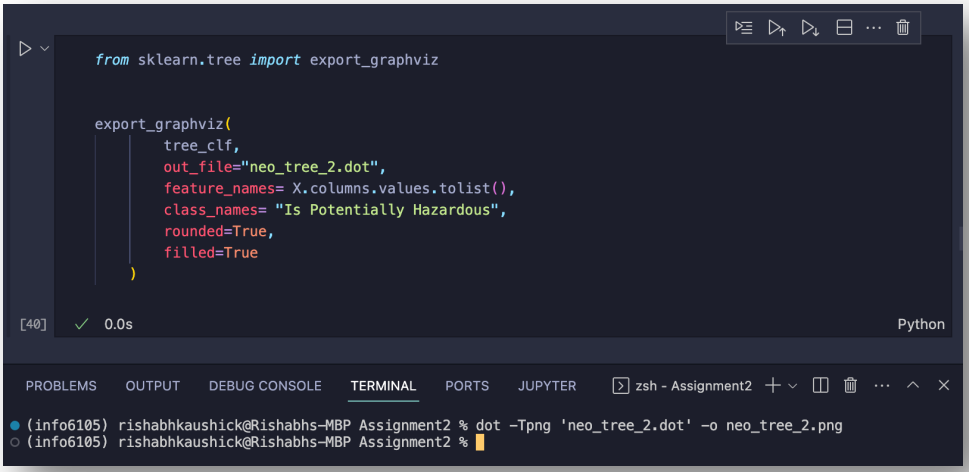


Figure 16

The new decision tree is as follows:

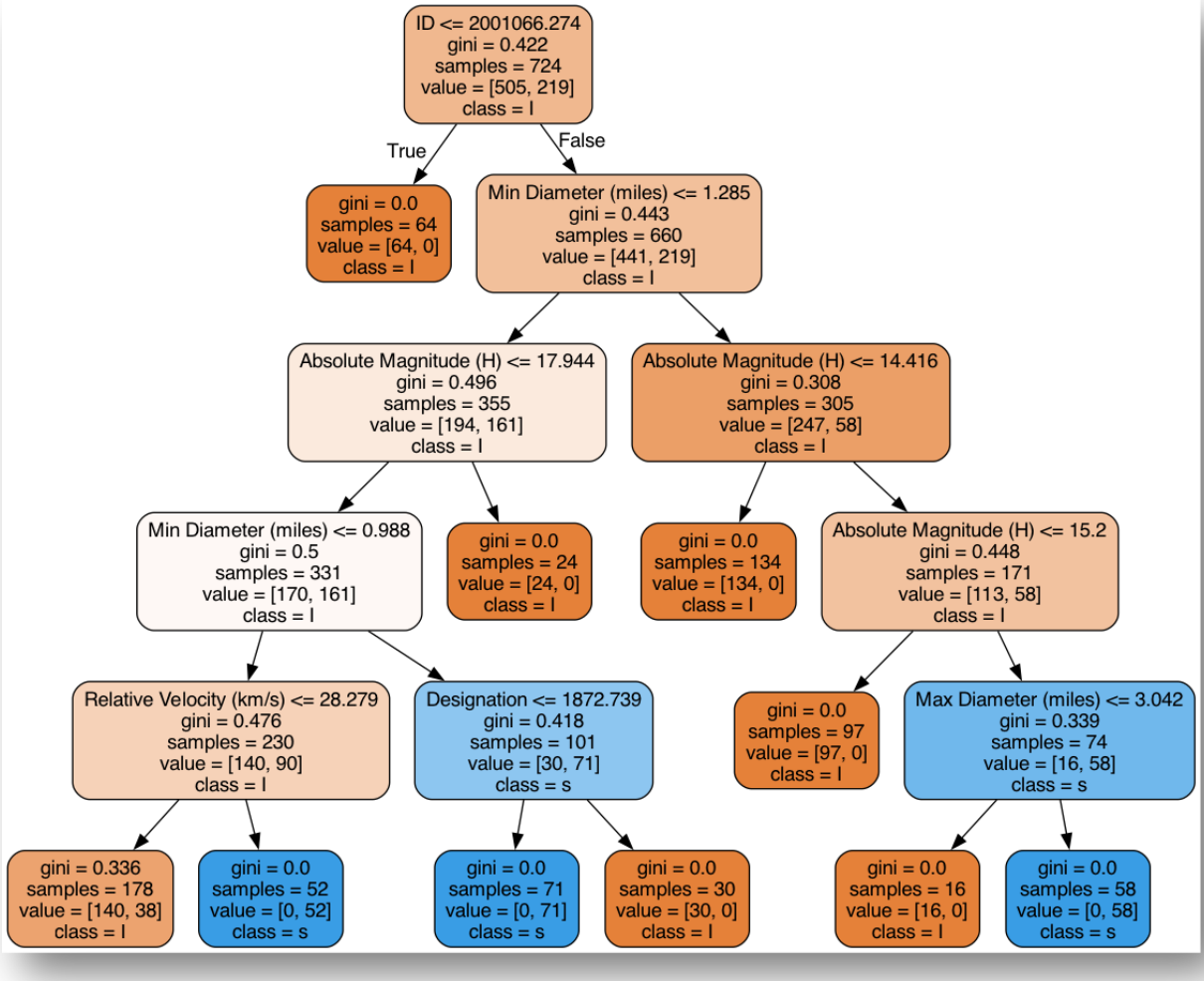


Figure 17: Decision Tree

```
from sklearn.metrics import mean_squared_error
tree_mean_sq_error = mean_squared_error(y, predictions)
tree_root_mean_sq_err = np.sqrt(tree_mean_sq_error)
tree_root_mean_sq_err
# >>> tree_mse = mean_squared_error(housing_labels, housing_predictions)
# >>> tree_rmse = np.sqrt(tree_mse)
# >>> tree_rmse
```

[42] ✓ 0.0s Python

```
0.2290986421725451
```

Figure 18

This time the error is 0.229, i.e., 22.9%, and therefore accuracy on the training dataset is 77.1%.

Artificial Neural Networks (ANNs)

Artificial Neural Networks are models which have taken inspiration from the human brain, which has billions of neurons. Here we build multiple layers of neural networks each having their own set of inputs, weights as well as the output.

The below neural network has been constructed of the following layers:

1. Dense layer of 50 neurons
2. Dense layer of 25 neurons
3. Dense layer of 10 neurons
4. Dense layer of 5 neurons
5. Output layer of 1 neuron – giving the output based on the results of each of the hidden layers.

Figure 19

All the hidden layers and the first layer have the non-linear activation function as ‘relu’. However, we chose the sigmoid nonlinear activation function in the last layer because it is better suited to predict the result of the classification.

Next, we must train the model. To avoid overfitting, I have considered to use a low number of epochs in this case 20. I have also used the ‘adam’ optimizer.

```
# similar to the above decision tree example, we are adding the loss to be binary cross entropy using the adam optimizer
ann_model.compile(loss="binary_crossentropy", optimizer="adam", metrics=['accuracy'])

# now running the training on the ANNs
# we will train the neural network 20 times with a batch size of 5.
ann_model.fit(X, y, epochs=20, batch_size=5)
```

[45] ✓ 2.2s

```
Epoch 1/20
145/145 [==============================] - 0s 685us/step - loss: 1141942400.0000 - accuracy: 0.5414
Epoch 2/20
145/145 [==============================] - 0s 674us/step - loss: 586262208.0000 - accuracy: 0.5815
Epoch 3/20
145/145 [==============================] - 0s 654us/step - loss: 928914688.0000 - accuracy: 0.6602
Epoch 4/20
145/145 [==============================] - 0s 745us/step - loss: 0.6769 - accuracy: 0.6975
Epoch 5/20
145/145 [==============================] - 0s 707us/step - loss: 0.6660 - accuracy: 0.6975
Epoch 6/20
145/145 [==============================] - 0s 723us/step - loss: 0.6564 - accuracy: 0.6975
Epoch 7/20
145/145 [==============================] - 0s 708us/step - loss: 0.6483 - accuracy: 0.6975
Epoch 8/20
145/145 [==============================] - 0s 607us/step - loss: 0.6413 - accuracy: 0.6975
Epoch 9/20
145/145 [==============================] - 0s 635us/step - loss: 0.6357 - accuracy: 0.6975
Epoch 10/20
145/145 [==============================] - 0s 633us/step - loss: 0.6309 - accuracy: 0.6975
Epoch 11/20
```

Figure 20

Let's evaluate the model by testing the loss and the accuracy:

```
loss, accuracy = ann_model.evaluate(X, y)
print(accuracy)
```

[46] ✓ 0.1s

```
23/23 [==============================] - 0s 517us/step - loss: 0.6140 - accuracy: 0.6975
0.6975138187408447
```

Figure 21

From the above figure, we can see that the training accuracy of the model is 69.75 %. Let us now find out the testing accuracy:

```
test_loss, test_accuracy = ann_model.evaluate(X_test, y_test)
test_accuracy
```

[51] ✓ 0.0s Python

```
6/6 [==============================] - 0s 1ms/step - loss: 0.6371 - accuracy: 0.6889

0.6888889074325562
```

Figure 22

The above figure shows us that the testing accuracy is around 68.88 %. Here we can see that this ANN model does not perform as good as the decision tree. However, the training and testing accuracy figures are very close to each other. This means that the model was able to generalize based on the data.

Assignment 2B

Abstract

In this Assignment 2B, we have covered the following machine learning algorithms for the following datasets:

Dataset #	Dataset	Machine Learning Algorithm
Dataset #1	Near Earth Objects	Ada-Boost
Dataset #1	Near Earth Objects	Gradient Boost
Dataset #2	Heart Attack Risk Prediction	Decision Trees
Dataset #2	Heart Attack Risk Prediction	Artificial Neural Networks (ANNs)
Dataset #2	Heart Attack Risk Prediction	Ada-Boost
Dataset #1	Near Earth Objects	Support Vector Machines (SVMs)
Dataset #1	Near Earth Objects	k-Nearest Neighbors (kNNs)
Dataset #2	Heart Attack Risk Prediction	Support Vector Machines (SVMs)
Dataset #2	Heart Attack Risk Prediction	k-Nearest Neighbors (kNNs)

Table 2: The algorithms applied on the datasets in the Assignment 2B

Ada Boost (Adaptive Boosting)

Boosting is an ensemble learning method. The concept behind Ada Boost is that it starts by using weak classifiers to classify the data. Initially the classification has many errors. Based on the data points which were correctly classified and misclassified, the model creates a weighted sample. In the weighted sample the ones which were wrongly classified by the weak classifier are given more weightage as compared to those which were correctly classified.

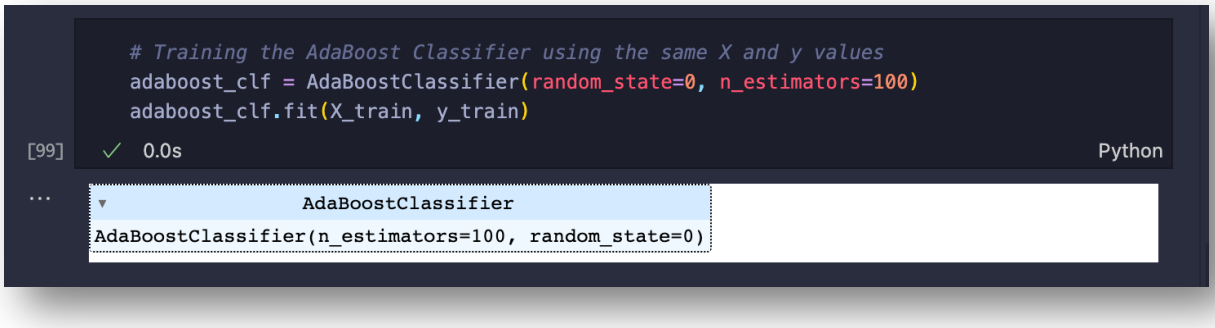


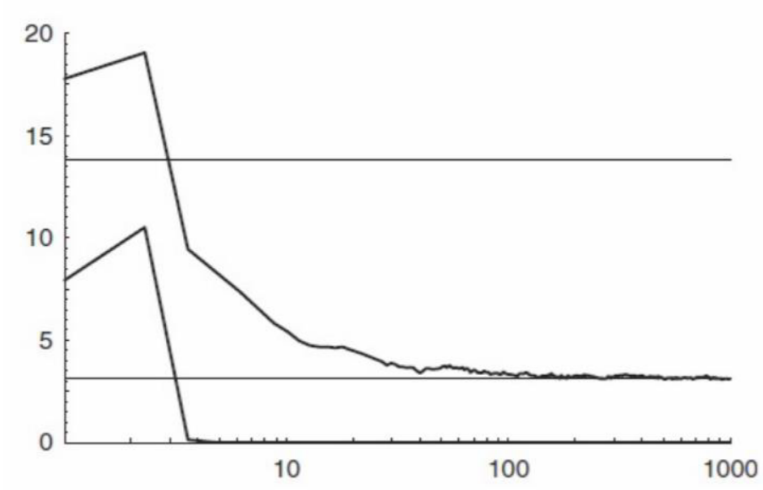
Figure 23: Training the Ada Boost Model

Surprisingly as shown in Figure below, the training and the test accuracy both were 100%. This means that for the training set as well as our testing set, our model was able to correctly predict all of the values of the target column. If the training accuracy is 100 % it usually leads to overfitting. However, in this case, since the testing accuracy is also 100%, we cannot say that it has overfit the data. Usually in practice the AdaBoost model has a very low chance of overfitting the data, and at times it continues to learn from the training dataset even after it has reached 100% accuracy. This can be seen in the Figure below.

```
print("Accuracy on training set: {:.3f}".format(adaboost_clf.score(X_train, y_train)))
[100]  ✓ 0.0s                                                                    Python
...  Accuracy on training set: 1.000

print("Accuracy on test set: {:.3f}".format(adaboost_clf.score(X_test, y_test)))
[101]  ✓ 0.0s                                                                    Python
...  Accuracy on test set: 1.000
```

Figure 24: Showing that the train and the test accuracy of the AdaBoost Model is 100%.



https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec19_20_handout.pdf

Figure 25: Showing the graph trend of the number of iterations on x-axis and the error percent on y-axis.

Additionally, since the test and train accuracy were 100% which sounds too good to be true, I have also trained a few more Ada Boost models as well as Gradient Boosting model with different parameters. Nonetheless each boosting model still had a train and test accuracy of 100% as shown in the below figure.

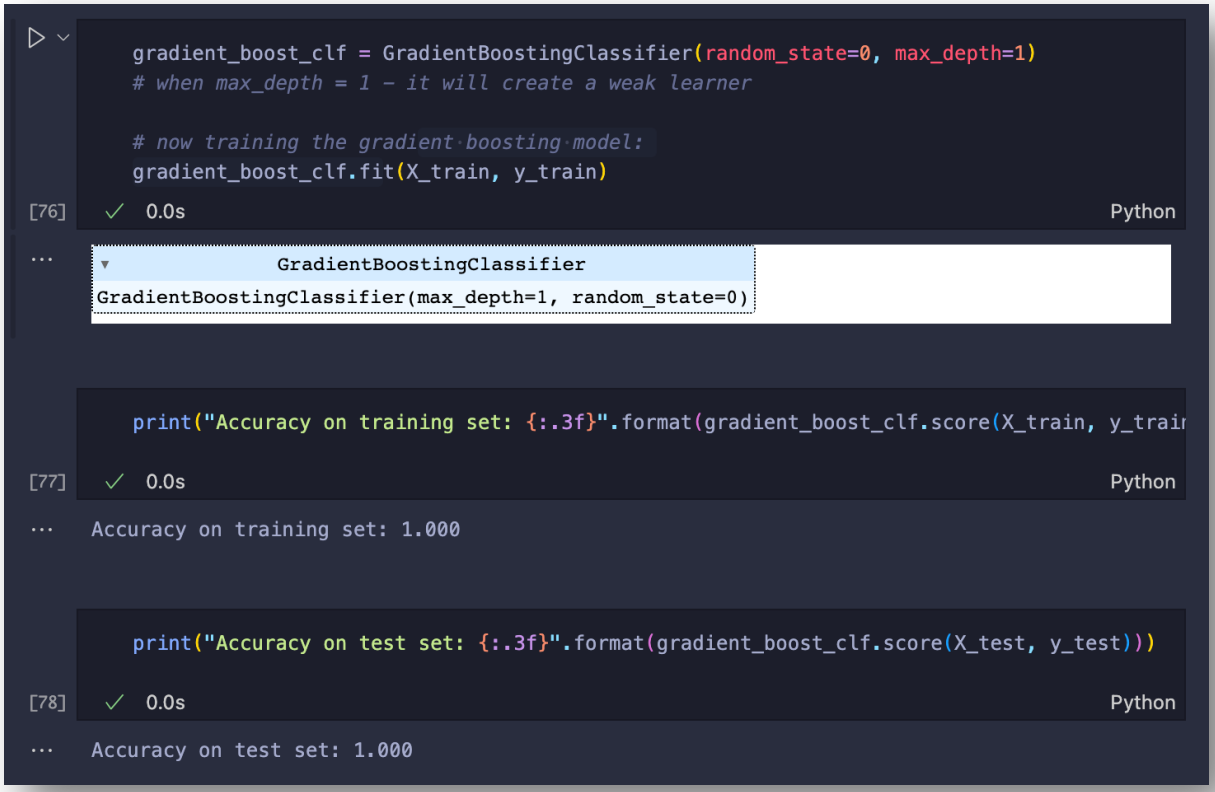


Figure 26: Gradient Boost method also yielded 100% accuracy on the test and the train sets.

One of the possible reasons why the boosting algorithms gives 100% accuracy is perhaps because the dataset is small with only around 900 rows. It is possible that with this small data it can predict all the values correctly.

K-Nearest Neighbors Classifier (KNNs)

K-Nearest Neighbors is an instance-based machine learning model. This algorithm can be implemented for classification as well as regression problems. In this case I used it for classification. I trained the KNN model on different values of k to see which one performs the best. The figure below shows the models trained on values of k as 3, 5 and 10.

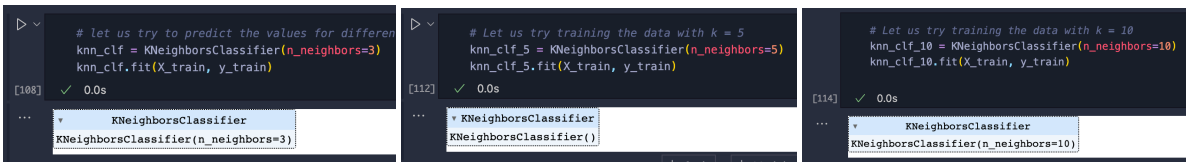


Figure 27: KNN Classifiers trained on k=3, k=5 and k=10

Based on the three models the training and testing accuracy is shown in the below figure.

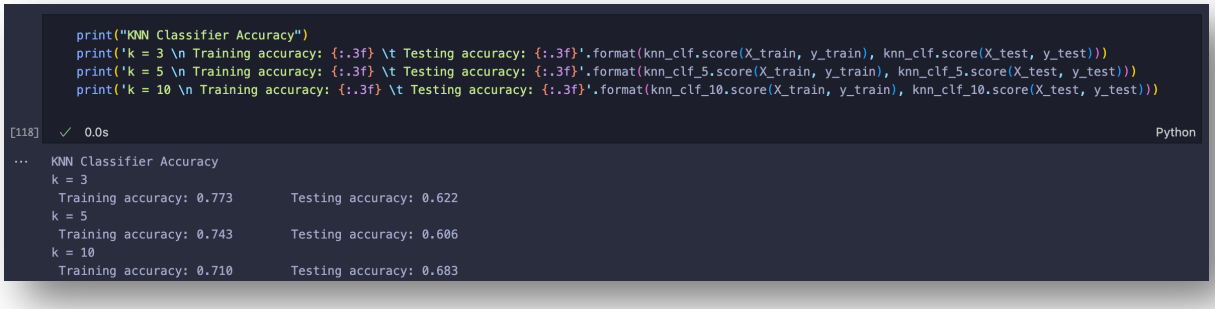


Figure 28: Training & Testing accuracy of the KNN model

The same training & testing accuracy can be seen in the below table:

k	Training Accuracy	Testing Accuracy
3	77.30 %	62.20 %
5	74.30 %	60.60 %
10	71.00 %	68.30 %

Table 3: Training and testing accuracy based on different values of k

As we increase the value of k, the training accuracy reduces, however the testing accuracy increases. Therefore we need to find the sweet spot where the training accuracy is also almost equal to the testing accuracy. From the above result, we can see that when k=3, the training accuracy is very high at 77.3 % however the testing accuracy is only 62.2 %. When we look at k=10, we can see that the training accuracy is lower at 71 % but the testing accuracy is much better at 68.30 %. Therefore the model where k=10 is better at generalizing the data.

Support Vector Machines (SVMs)

Linear Support Vector Classifier (Linear SVC)

Support vector machine using Linear SVC:

Figure 29: Linear SVC model

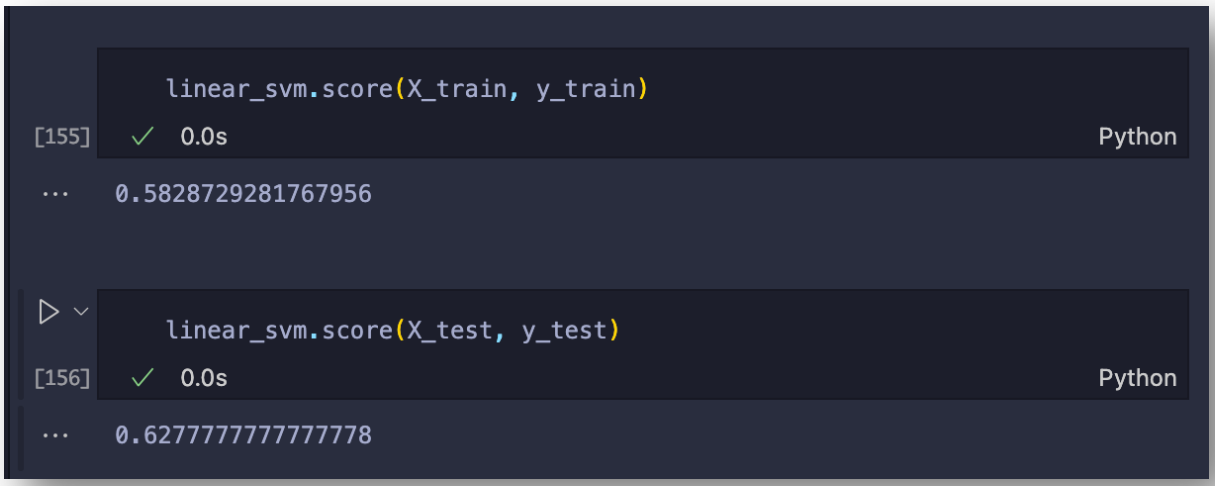


Figure 30: Linear SVC training & testing accuracy

Radial Basis Function Support Vector Classifier (RBF SVC)

To improve the linear support vector, I have also tried training the support vector machine using the RBF kernel.

Figure 31: RBF SVC model

Results

Below table shows the results of the different machine learning models which were used to train and test data from the first dataset.

Dataset	Machine Learning Model	Accuracy	
Dataset #1: Near Earth Objects	Decision Trees	Training	77.10 %
		Testing	77.64 %
	Artificial Neural Networks	Training	69.75 %
		Testing	68.88 %
	Ada Boost	Training	100 %
		Testing	100 %
	Gradient Boosting	Training	100 %
		Testing	100 %
	K-Nearest Neighbors Classifier (Best k = 10)	Training	71.00 %
		Testing	68.30 %
	Support Vector Machines (using RBF kernel)	Training	100 %
		Testing	68.89 %

Table 4

Diving Into Dataset #2

Understanding the Data

```
# Fist 5 rows
heart_df.head()
```

[6] Python

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	Sedentary Hours Per Day	Income	BMI	Triglycerides	Physical Activity Days Per Week	Sleep Hours Per Day	Cou
0	BMW7812	67	Male	208	158/88	72	0	0	1	0	...	6.615001	261404	31.251233	286	0	6	Arger
1	CZE1114	21	Male	389	165/93	98	1	1	1	1	...	4.963459	285768	27.194973	235	1	7	Car
2	BNI9906	21	Female	324	174/99	72	1	0	0	0	...	9.463426	235282	28.176571	587	4	4	Fra
3	JLN3497	84	Male	383	163/100	73	1	1	1	0	...	7.648981	125640	36.464704	378	3	4	Car
4	GFO8847	66	Male	318	91/88	93	1	1	1	1	...	1.514821	160555	21.809144	231	1	5	Thai

5 rows × 26 columns

Figure 32: The type of data in the pandas data-frame object

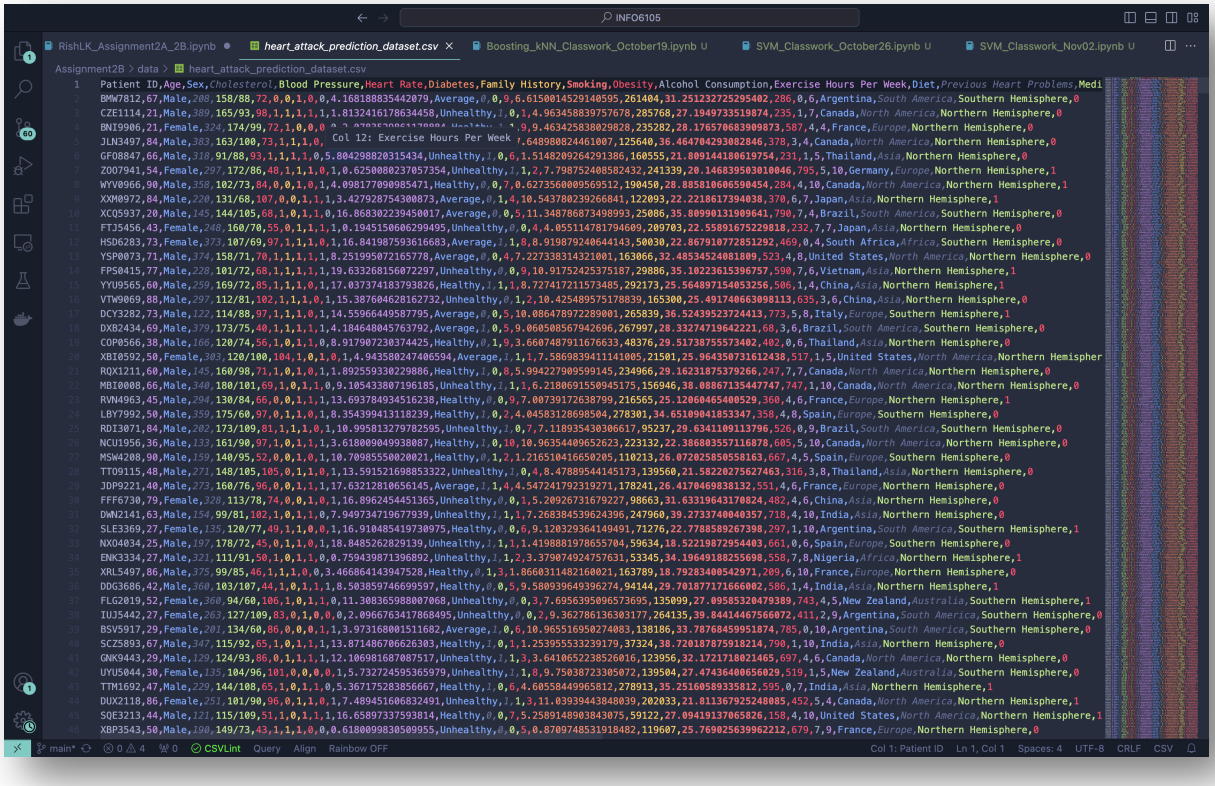
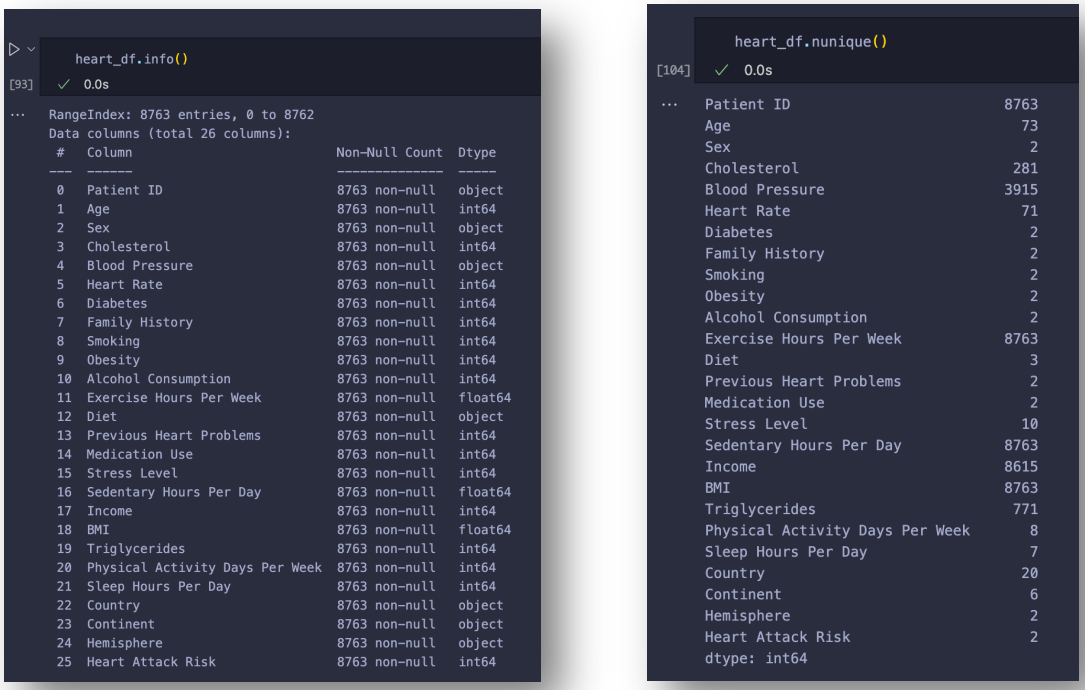


Figure 33: The data in csv format

Data Characteristics

The below figures show the different columns that are present in the second dataset, and how many rows have duplicate data.



```
heart_df.info()
[93]  ✓ 0.0s

RangeIndex: 8763 entries, 0 to 8762
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Patient ID                      8763 non-null   object
 1   Age                             8763 non-null   int64
 2   Sex                             8763 non-null   object
 3   Cholesterol                     8763 non-null   int64
 4   Blood Pressure                  8763 non-null   object
 5   Heart Rate                      8763 non-null   int64
 6   Diabetes                        8763 non-null   int64
 7   Family History                  8763 non-null   int64
 8   Smoking                         8763 non-null   int64
 9   Obesity                         8763 non-null   int64
 10  Alcohol Consumption             8763 non-null   int64
 11  Exercise Hours Per Week         8763 non-null   float64
 12  Diet                            8763 non-null   object
 13  Previous Heart Problems         8763 non-null   int64
 14  Medication Use                  8763 non-null   int64
 15  Stress Level                    8763 non-null   int64
 16  Sedentary Hours Per Day         8763 non-null   float64
 17  Income                          8763 non-null   int64
 18  BMI                             8763 non-null   float64
 19  Triglycerides                   8763 non-null   int64
 20  Physical Activity Days Per Week 8763 non-null   int64
 21  Sleep Hours Per Day             8763 non-null   int64
 22  Country                         8763 non-null   object
 23  Continent                       8763 non-null   object
 24  Hemisphere                      8763 non-null   object
 25  Heart Attack Risk               8763 non-null   int64
```

```
heart_df.nunique()
[104]  ✓ 0.0s

Patient ID                      8763
Age                               73
Sex                                2
Cholesterol                      281
Blood Pressure                  3915
Heart Rate                        71
Diabetes                           2
Family History                     2
Smoking                            2
Obesity                            2
Alcohol Consumption                2
Exercise Hours Per Week         8763
Diet                               3
Previous Heart Problems            2
Medication Use                     2
Stress Level                      10
Sedentary Hours Per Day         8763
Income                          8615
BMI                             8763
Triglycerides                    771
Physical Activity Days Per Week    8
Sleep Hours Per Day                7
Country                           20
Continent                          6
Hemisphere                         2
Heart Attack Risk                  2
dtype: int64
```

Figure 34: Shows the columns, datatypes and number of unique values are present.

The above figure 34 shows us that there are a total of 8763 rows in the pandas data frame object. We can see that for each column there are 8763 non-null rows. Therefore, this data set does not contain any columns with null or NaN data values. If there were such columns containing null records, we will need to either replace those null values with certain dummy data (for example the mean or median) otherwise we must delete the row entirely.

Figure 34, also shows that most of the columns have data which are relevant for the machine learning model. Apart from the Patient ID column, all the rest of the columns can be used to train the models. The Patient ID column can be dropped since it is a primary key column which does not hold significance to the model.

From this data let us further visualize the target column – ‘Heart Attack Risk’. The figure below shows us that there is more data wherein there is No Heart Attack Risk compared to ‘High Heart Attack Risk’.

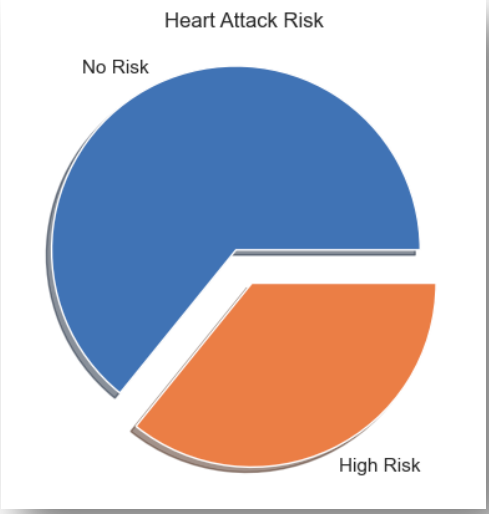


Figure 35: Heart Attack Risk in a pie chart.

Next, we can see that there are many columns which have data which are not in integer or floating-point values such as:

1. Sex
2. Blood Pressure
3. Diet
4. Country
5. Continent
6. Hemisphere

Data Cleaning & Preprocessing

Let us explore the type of data stored in these columns.

Sex Column

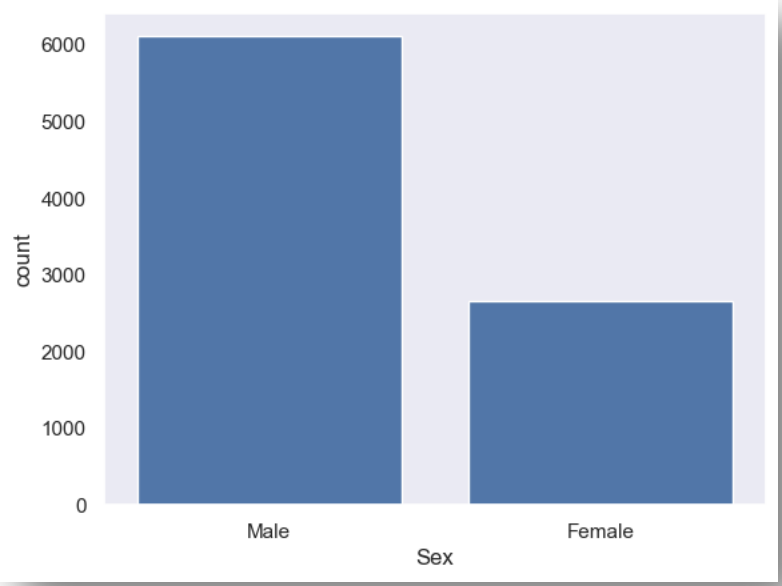


Figure 36

We can see that there are around 6000 males and less than 3000 females. Next, I have converted this categorical data into a numeric value using Label Encoders.

```
label_encoder = LabelEncoder()
heart_df['Sex'] = label_encoder.fit_transform(heart_df['Sex'])
```

[125] ✓ 0.0s

Figure 37

Blood Pressure

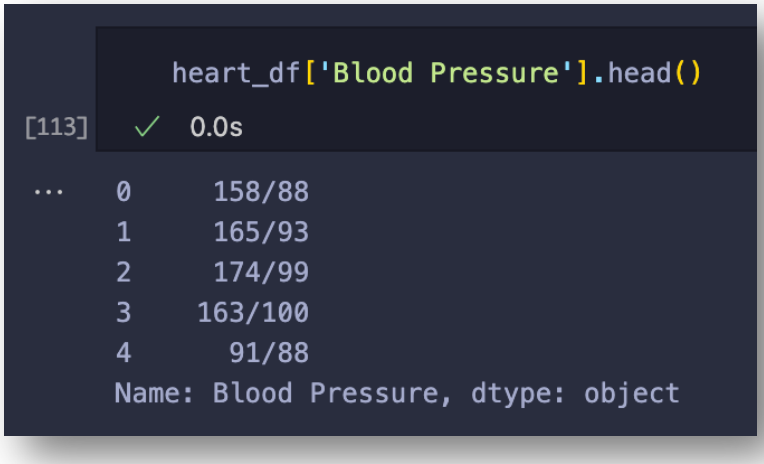
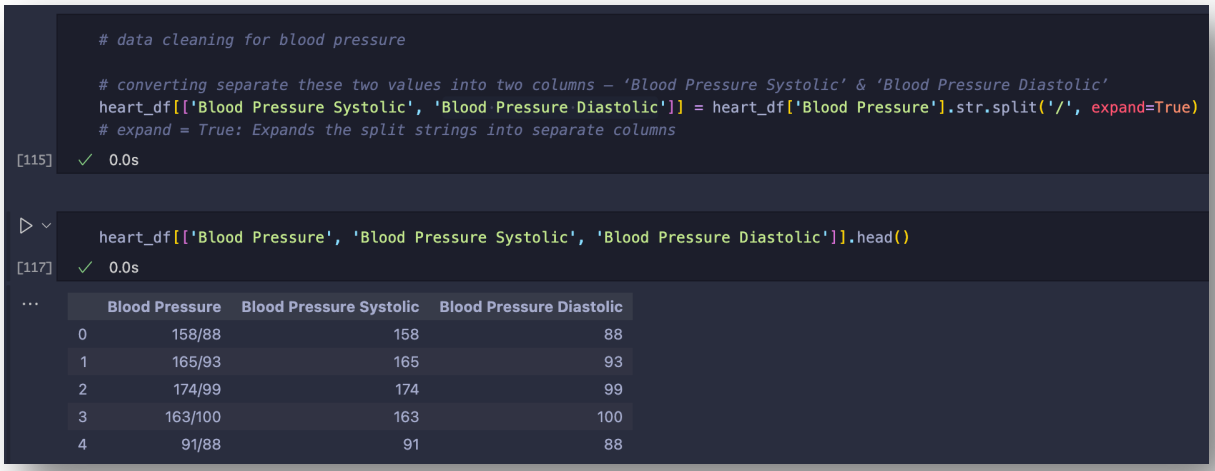


Figure 38

We can see that the blood pressure data is present in the 'Systolic/Diastolic' format. Let us separate these two values into two different columns – 'Blood Pressure Systolic' & 'Blood Pressure Diastolic'.



	Blood Pressure	Blood Pressure Systolic	Blood Pressure Diastolic
0	158/88	158	88
1	165/93	165	93
2	174/99	174	99
3	163/100	163	100
4	91/88	91	88

Figure 39: New columns for Blood Pressure

From figure 39, we can see that the data has been correctly split and transformed. Now we can drop the Blood Pressure column.

Diet

Based on Figure 40 below, we can see that the diet can be either:

1. Healthy
2. Average
3. Unhealthy

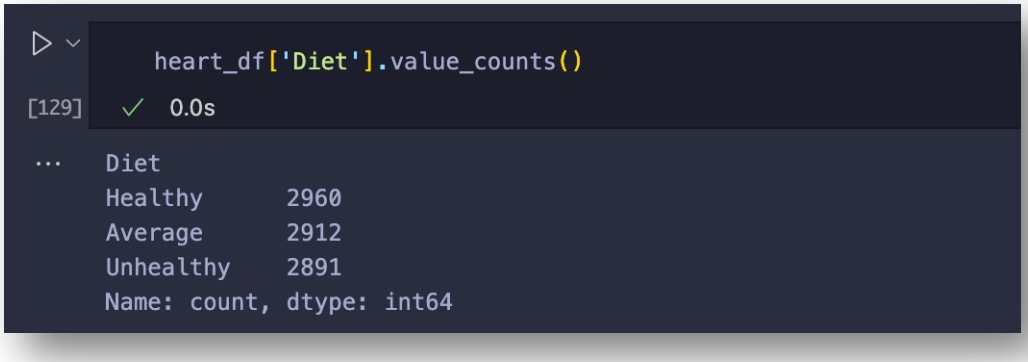


Figure 40: Values present in the 'Diet' column.

Again, we must convert this into numeric data for the classifier. A good diet is important when we must predict if a patient might have a heart attack risk. Based on my intuition of the subject matter, I have decided to change the data to the following:

1. Healthy = 1.0
2. Average = 0.5
3. Unhealthy = 0.0

Figure 41: New Values for 'Diet' column.

Country, Continent & Hemisphere

The data in these columns are as follows:

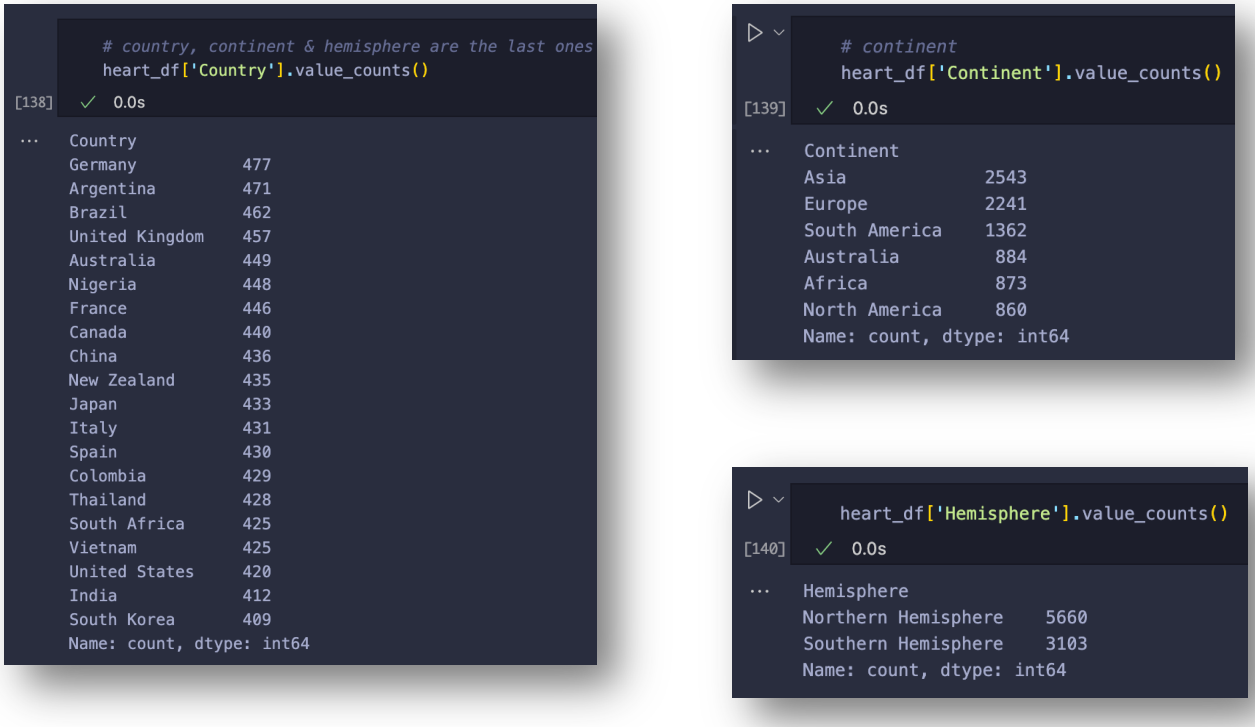


Figure 42: The data in the three columns

Just like the 'Sex' column, we can use the label encoders to convert this categorical data into numeric values.

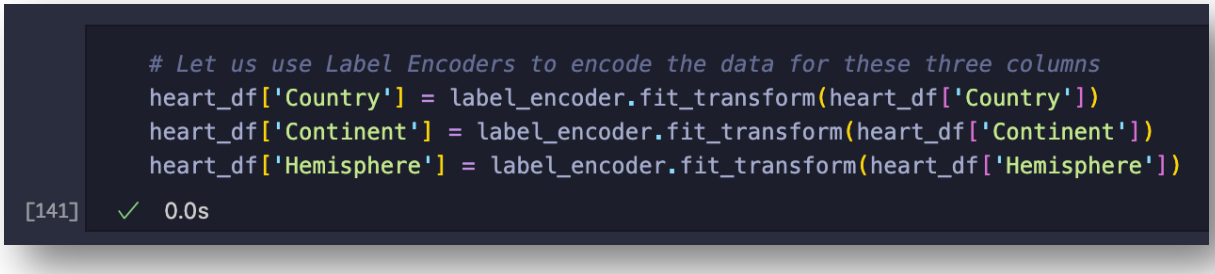


Figure 43

Now, all the columns have numeric data:

```
heart_df.info()
```

```
[142] ✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 26 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Age                              8763 non-null   int64  
 1   Sex                              8763 non-null   int64  
 2   Cholesterol                      8763 non-null   int64  
 3   Heart Rate                       8763 non-null   int64  
 4   Diabetes                         8763 non-null   int64  
 5   Family History                   8763 non-null   int64  
 6   Smoking                          8763 non-null   int64  
 7   Obesity                          8763 non-null   int64  
 8   Alcohol Consumption              8763 non-null   int64  
 9   Exercise Hours Per Week          8763 non-null   float64
 10  Diet                             8763 non-null   float64
 11  Previous Heart Problems          8763 non-null   int64  
 12  Medication Use                   8763 non-null   int64  
 13  Stress Level                     8763 non-null   int64  
 14  Sedentary Hours Per Day          8763 non-null   float64
 15  Income                           8763 non-null   int64  
 16  BMI                              8763 non-null   float64
 17  Triglycerides                    8763 non-null   int64  
 18  Physical Activity Days Per Week  8763 non-null   int64  
 19  Sleep Hours Per Day              8763 non-null   int64  
 20  Country                          8763 non-null   int64  
 21  Continent                        8763 non-null   int64  
 22  Hemisphere                       8763 non-null   int64  
 23  Heart Attack Risk                8763 non-null   int64  
 24  Blood Pressure Systolic          8763 non-null   int64  
 25  Blood Pressure Diastolic         8763 non-null   int64  
```

Figure 44: Preprocessed Data

Splitting the Data

75 % of the data has been segregated into a train set and the remaining 25 % of the data has been segregated into a test set.

```
heart_X_train, heart_X_test, heart_y_train, heart_y_test = train_test_split(heart_df.drop(columns=['Heart Attack Risk']), heart_df['Heart Attack Risk'], random_state=0)
# by default it sets the train size to be 25 %
```

```
[143] ✓ 0.0s
```

Figure 45: Train Test Split

Correlation

From the correlation matrix shown in the Jupyter notebook, we can see that no column has high correlation when compared to the target column. However, we can see some interesting similarities:

1. Sex column and smoking column have high correlation.
   This means that Male person is more likely to smoke as compared to a Female person.

Data Visualization for Dataset #2

Let us try to visualize the data in the following graph:

- X-axis: Sex (0 – female, 1 – male)
- Y-axis: Age
- Color: Red – high heart attack risk, Blue – low heart attack risk

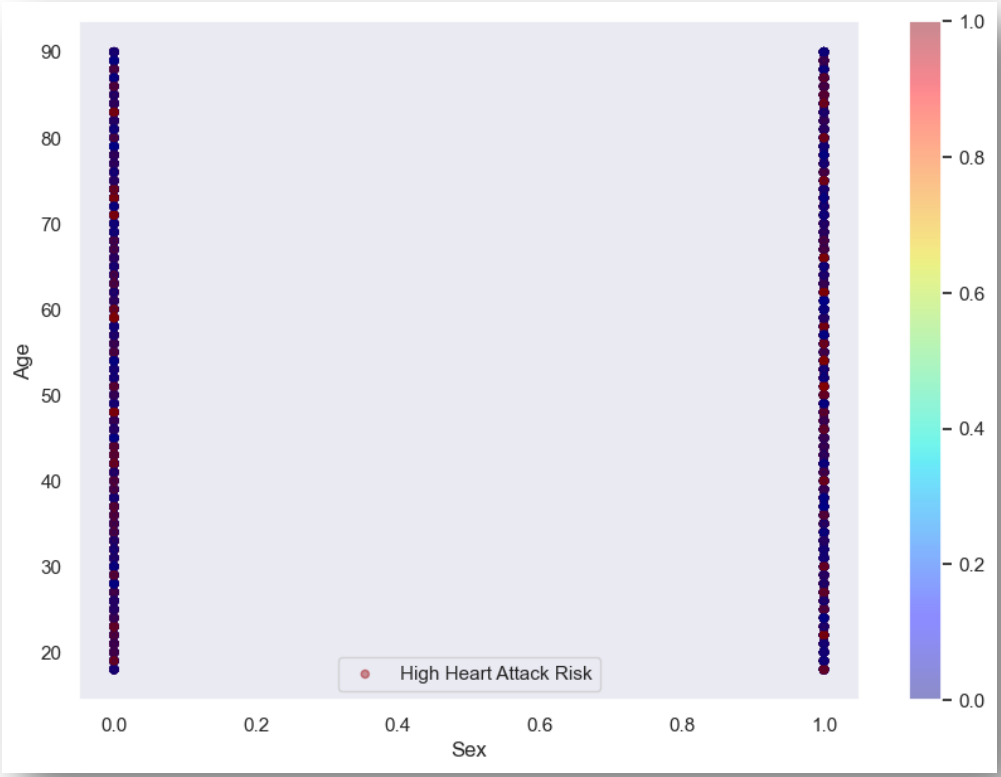


Figure 46

This figure does not give us much information about the target column. Therefore, let us try to visualize some other columns with each other:

- X-axis: Smoking (0 – not a smoker, 1 – is a smoker)
- Y-axis: Age
- Size of data point: Exercise (Hours Per Week)
- Color: Red – high heart attack risk, Blue – low heart attack risk

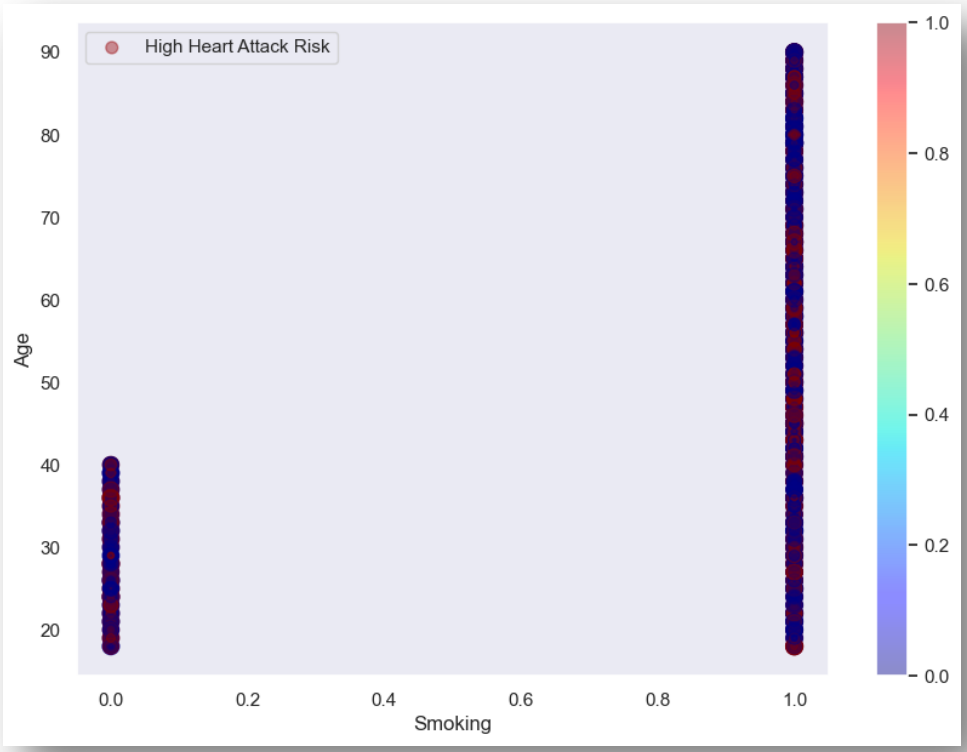

High Heart Attack Risk
90
80
70
60
50
40
30
20
Age
0.0
0.2
0.4
0.6
0.8
1.0
Smoking
1.0
0.8
0.6
0.4
0.2
0.0

Figure 47

...

Decision Tree Model

```
# Further increasing the depth of the model, using splitter = "best" instead of "random" and criterion="gini"
heart_decision_tree_clf3 = DecisionTreeClassifier(max_depth=15, random_state=23, splitter="best", criterion="gini")
heart_decision_tree_clf3.fit(heart_X_train_matrix, heart_y_train)
```

[200] ✓ 0.2s

```
DecisionTreeClassifier(max_depth=15, random_state=23)
```

Figure 48

Once we have trained the decision tree model, we are going to try to visualize it. Since the decision tree is quite large with 15 layers of depth, it is difficult to visualize it on Jupyter Notebook as shown in Figure 49 below:

```
heart_dot_data3 = StringIO()
export_graphviz(
        heart_decision_tree_clf3,
        out_file=heart_dot_data3,
        feature_names= heart_X_train_matrix.columns.values.tolist(),
        class_names= "Heart Attack Risk",
        rounded=True,
        filled=True,
        special_characters=True
    )
graph = pydotplus.graph_from_dot_data(heart_dot_data3.getvalue())
graph.write_png('heart_attack_decision_tree_model3.png')
Image(graph.create_png())
```

[204] ✓ 10.5s

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.58378 to fit

dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.58378 to fit
```

Figure 49

```
heart_decision_tree_clf3.score(heart_X_train_matrix, heart_y_train)
```

[202] ✓ 0.0s

```
0.8603164942178941
```

The final decision tree model has a training accuracy of 86.03 %. Let us see the testing accuracy:

```
heart_decision_tree_clf3.score(heart_X_test.drop(columns=['Heart Attack Risk']), heart_y_test)
```

[205] ✓ 0.0s

```
0.5723413966225468
```

Figure 50: The training accuracy = 86.03% & testing accuracy = 57.23%

We can see that the model did much better in the training – 86.03 % but it did not do very well in the test set where it got the accuracy as 57.23 %. This shows that the model is not generalizing the data well, and rather fitting it very close to the training dataset.

Artificial Neural Networks (ANN) Model

The below Figure 51 shows that ANN model with the following hidden layers:

1. Dense layer consisting of 60 neurons, with the reLU non-linear activation function.
2. Dense layer consisting of 30 neurons, with the reLU non-linear activation function.
3. Dense layer consisting of 10 neurons, with the reLU non-linear activation function.
4. Dense layer consisting of 3 neurons, with the reLU non-linear activation function.
5. Dense layer consisting of 1 neuron, with the sigmoid non-linear activation function.

Figure 51: ANN Model

The training and testing accuracy of the model is as follows:

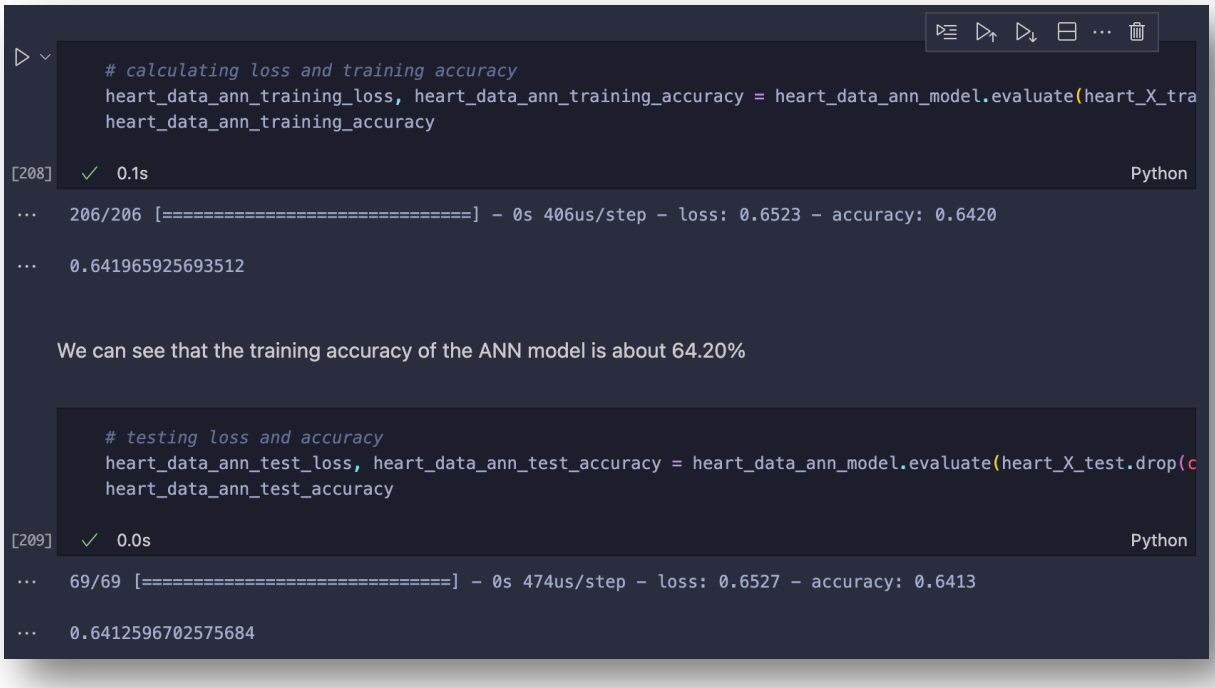


Figure 52: ANN Training & testing accuracy

Ada-Boost (Adaptive Boosting) Model

Training the Ada Boost model on n_estimators = 100 and a random state of 0.

Figure 53: Ada-Boost model

The training and the test accuracy based on the Ada-Boost model is as follows:

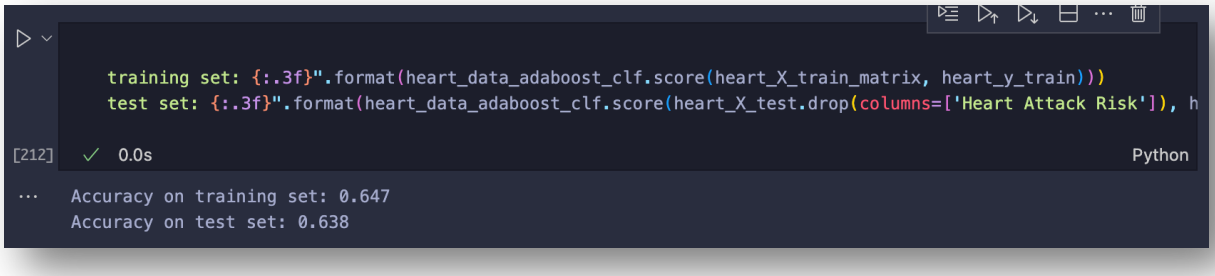


Figure 54: Ada-Boost training and testing accuracy

The Ada-Boost model has received the accuracy of 64.70 % on the training data and 63.80% on the testing data.

K-Nearest Neighbor (kNN) Classifier

While training the kNN model, we realized that there are 25 features in the dataset. Due to the curse of high dimensionality, we cannot use KNN models for data which have more than 9 features. The curse of high dimensionality states that as we keep going into higher and higher dimensions, the space between each data point also grows exponentially. It may so happen that in such high dimensions, we would not be able to find even one neighboring point nearby.

Therefore, for this dataset it is not advisable to use KNN model as we did in the previous dataset.

Support Vector Machines (SVMs)

Linear Support Vector Machine

Initially we have trained a support vector machine using linear kernel, with maximum iterations as 2000. We can see in Figure 55 below that the model was able to predict values for training dataset with 64.19 % accuracy, and for test set with 64.13 %. This model has a consistent error in both training and testing datasets.

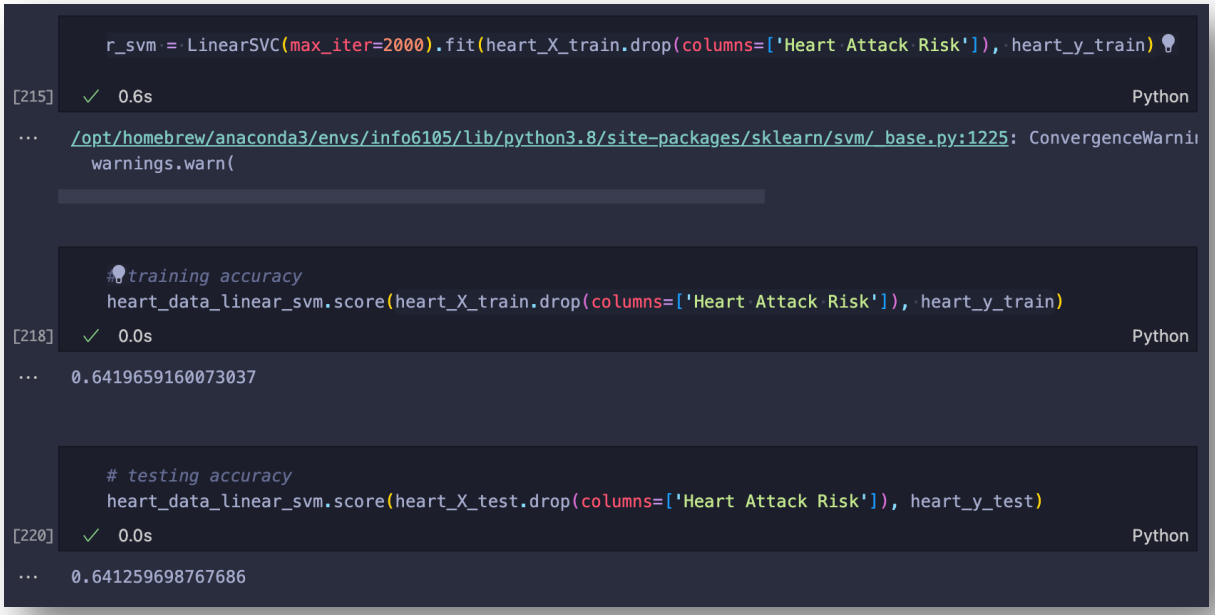


Figure 55

Support Vector Machines with RBF Kernel

In this model I used the value of C as 1, and gamma as 1. The model was able to correctly classify all the training data with a 100 % accuracy, but as shown below it was only able to get a testing accuracy of 64.13 %. Therefore this model has clearly overfit the data.

Figure 56

Results

Dataset	Machine Learning Model	Accuracy	
Dataset #2: Heart Attack Risk Dataset	Decision Trees	Training	86.03 %
		Testing	57.23 %
	Artificial Neural Networks	Training	64.20 %
		Testing	64.13 %
	Ada Boost	Training	64.70 %
		Testing	63.80 %
	K-Nearest Neighbors Classifier (Best k = 10)	Training	- %
		Testing	- %
	Linear Support Vector Machines	Training	64.19 %
		Testing	64.13 %
	Support Vector Machines using RBF Kernel	Training	100 %
		Testing	64.13 %

Conclusion

In this report, I have analyzed the performance of around 6 supervised learning algorithms – decision trees, artificial neural networks, Boosting Algorithms (such as Ada-Boost & Gradient Boosting), K-Nearest Neighbors and Support Vector machines. These models were trained on very interesting datasets – Near Earth Object Dataset and Heart Attack Risk Prediction Dataset. I learnt how models sometimes tend to overfit the data and made changes to reduce overfitting. Additionally, I understood how each of the machine learning – supervised models work.

Acknowledgements

The datasets have been obtained from Kaggle as follows, without which this project would not be possible.

1. https://www.kaggle.com/datasets/shivd24coder/nasa-neo-near-earth-object-dataset/data
2. https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset

References

1. GERON, A. (2019). *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools and techniques to build Intelligent Systems* (Vol. 2). O'Reilly.
2. Mikedelong. (2023a, September 21). *Understand danger with scatter plots*. Kaggle. https://www.kaggle.com/code/mikedelong/understand-danger-with-scatter-plots
3. https://docs.python.org/3/library/collections.html#collections.Counter
4. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.nunique.html
5. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html
6. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.hist.html
7. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html
8. https://aws.amazon.com/what-is/overfitting/#:~:text=Overfitting%20occurs%20when%20the%20model,to%20several%20reasons%2C%20such%20as%3A&text=The%20training%20data%20size%20is,all%20possible%20input%20data%20values.
9. *Sklearn.model_selection.train_test_split*. scikit. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

10. Kurama, V. (2021, April 9). *A guide to understanding AdaBoost*. Paperspace Blog. https://blog.paperspace.com/adaboost-optimizer/
11. Lecture 19-20: Ensembles, Department of Computer Science, University of Toronto. (n.d.). https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec19_20_handout.pdf
12. *Sklearn.svm.LinearSVC*. scikit. (n.d.-c). https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
13. *Sklearn.neighbors.kneighborsclassifier*. scikit. (n.d.-c). https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
14. *Sklearn.svm.SVC*. scikit. (n.d.-e). https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
15. Eskandar, S. (2023, March 26). *Introduction to RBF SVM: A powerful machine learning algorithm for non-linear data*. Medium. https://medium.com/@eskandar.sahel/introduction-to-rbf-svm-a-powerful-machine-learning-algorithm-for-non-linear-data-1d1cfb55a1a#:~:text=Radial%20Basis%20Function%20Support%20Vector,linear%20and%20high%2Ddimensional%20data.
16. *Matplotlib.pyplot.pie#*. matplotlib.pyplot.pie - Matplotlib 3.8.1 documentation. (n.d.). https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html
17. *Matplotlib.pyplot#*. matplotlib.pyplot - Matplotlib 3.8.1 documentation. (n.d.). https://matplotlib.org/stable/api/pyplot_summary.html
18. *Understanding blood pressure readings*. www.heart.org. (2023, October 17). https://www.heart.org/en/health-topics/high-blood-pressure/understanding-blood-pressure-readings
19. *Pandas.series.str.split#*. pandas.Series.str.split - pandas 2.1.2 documentation. (n.d.). https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.split.html
20. *Pandas.DataFrame.astype#*. pandas.DataFrame.astype - pandas 2.1.2 documentation. (n.d.). https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.astype.html
21. *Sklearn.preprocessing.LabelEncoder*. scikit. (n.d.-d). https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.preprocessing.LabelEncoder.fit_transform
22. Grant, P. (2019, July 24). *K-nearest neighbors and the curse of dimensionality*. Medium. https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d
23.