Titanic Data Science Project

Nursima Donuk

Problem Type: Classification

07/18/20

In this paper I will be walking you through my first data science project and how I approached the problem at hand. I acquired the problem and the data from Kaggle which is an online data science platform. On Kaggle data scientists or simply people interested in data science/machine learning can join competitions to expand their data science skills and knowledge. Here is a link to the titanic problem on Kaggle: https://www.kaggle.com/c/titanic/overview. We are given two sets of data, one of which has information about the people on the titanic (independent variables) and whether they survived the sinking (dependent variable). The other set of data only has information about people and the challenge is to predict whether they survived or not.

What do we have in our dataset?

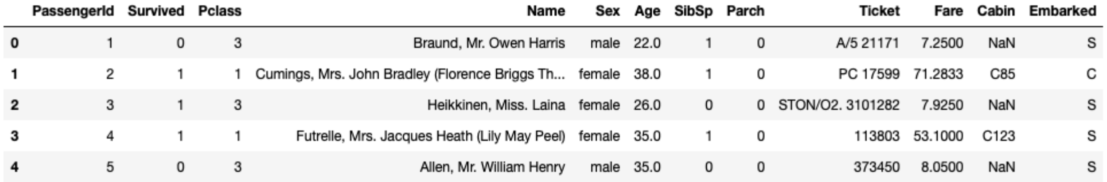
- Our training set has 891 entries, this is what we will use to predict the 418 entries in the test set. Let us take a look at the columns;
- **PassengerId**: Passenger identification number
- **Survived**: 1 if passenger survived, 0 if not (we do not have this column in the test set)
- **Pclass**: Ticket class, 1: 1st class, 2: 2nd class, 3: 3rd class
- **Name**: Name of passenger
- **Sex**: Sex/gender of passenger
- **Age**: Age of passenger
- **SibSp**: Number of siblings or spouses on board
- **Parch**: Number of parents or children on board
- **Ticket**: Ticket number
- **Fare**: Passenger fare
- **Cabin**: Cabin number
- **Embarked**: C: Cherbourg, Q: Queenstown, S: Southampton

Where do we start?

I approached the problem using python and did some visualizations with Tableau. First we start by importing the data set and start analyzing. A more detailed version of the code can be found in the .ipynb file.

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('train.csv')
dataset.head()
```

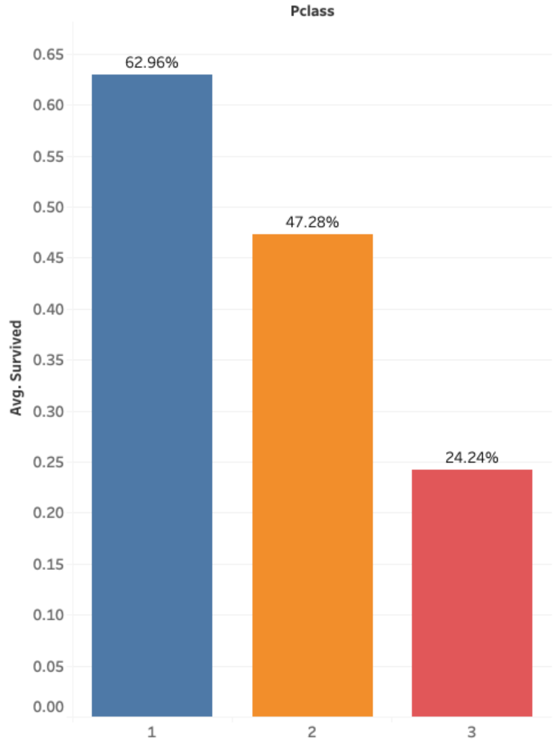
In the above code, the last line lets us take a look at the data first few entries of the data.

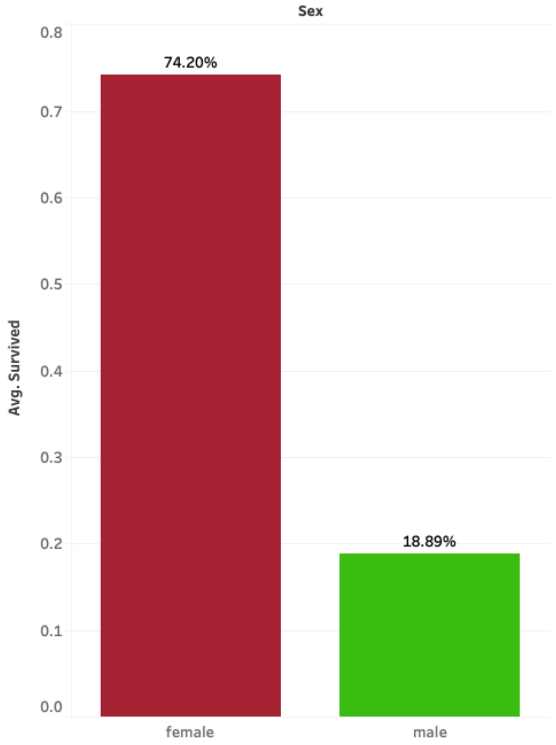


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

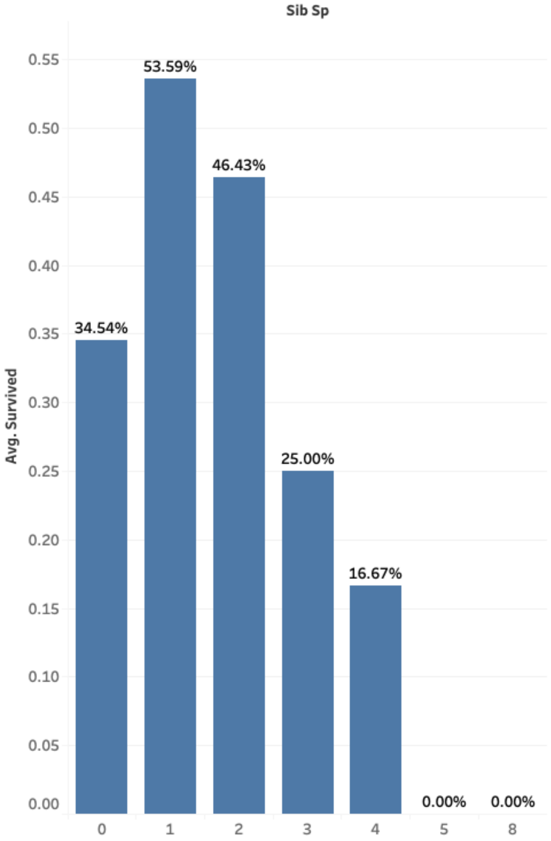
We can already see some variables that we may want to omit, such as passenger id and ticket number. The reason for this is that everyone has a unique passenger id and ticket number that would not correlate with whether they would survive the sinking of the Titanic. When we take a look at the Cabin number we see that we have a lot of missing data so it cannot be a strong factor to predict the dependent variable. This is why I decided to look into the remaining variables which are, pclass, name, sex, age, sibsp, parch, fare, embarked. Now, you may be thinking what their name has to do with the survival of a passenger. If we look closely at the entries of name, we can see that each persons name has a title and we will be working with that to find a correlation.

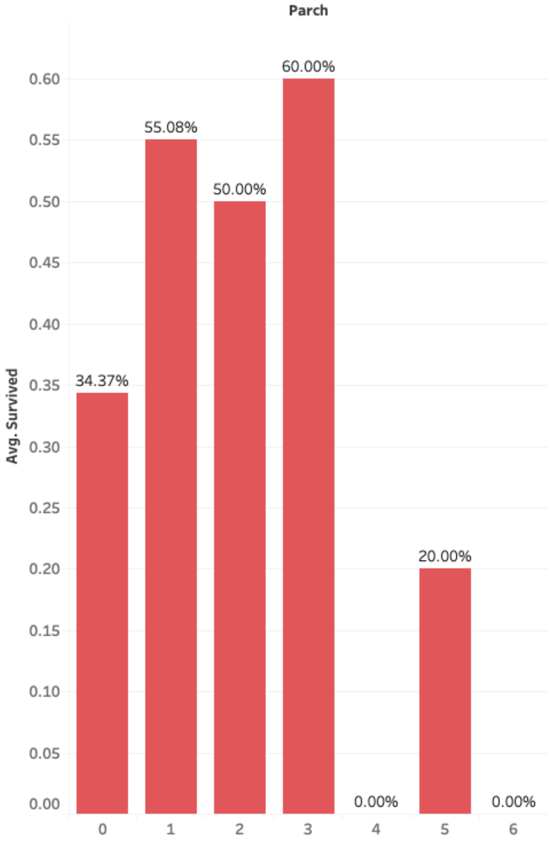
Let us start analyzing



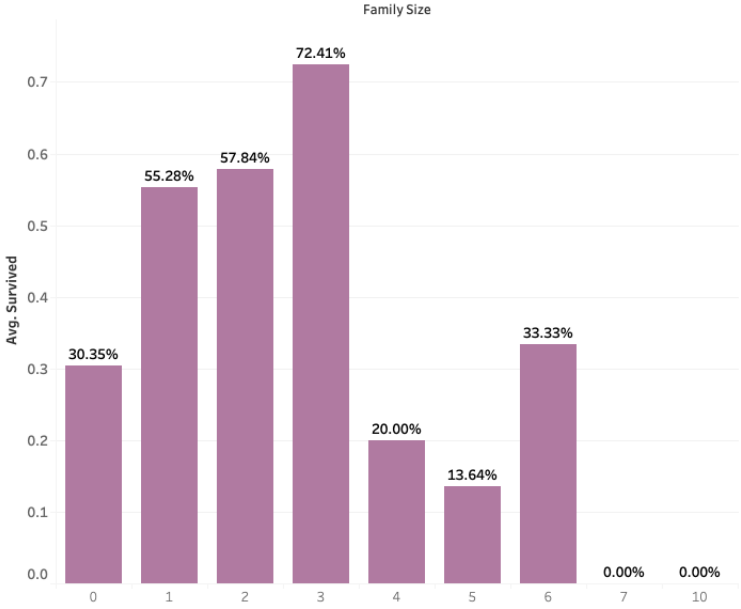


By looking at these graphs we can see that passengers from the first class were the most likely to survive where third class passengers were the least. Likewise, females were a lot more likely to survive than males. This is how we know that these two variables will help our model greatly. The next pair of data that I found interesting were the number of siblings/spouses and parents/children on board.





We can see that not only do these variables represent similar attributes, they also have similar distributions. This is why I decided to combine them into one variable which would represent family size.

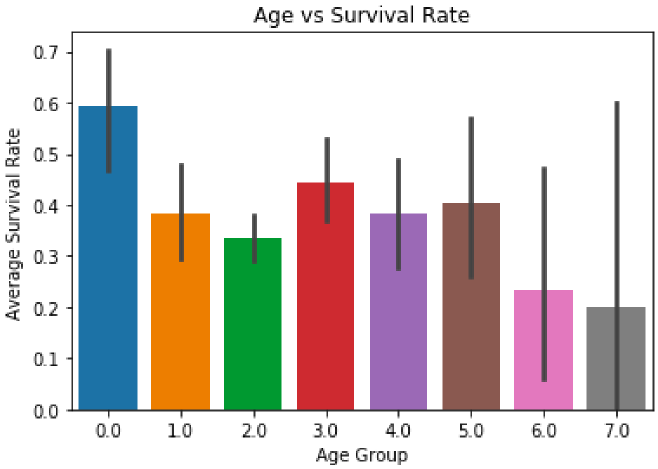


We see that our distribution has not changed as it is still skewed to the right. Even though there are some missing entries for age, it is not as many as there was for cabin so I decided to use the scikit-learn library of python to take care of missing data before visualizing.

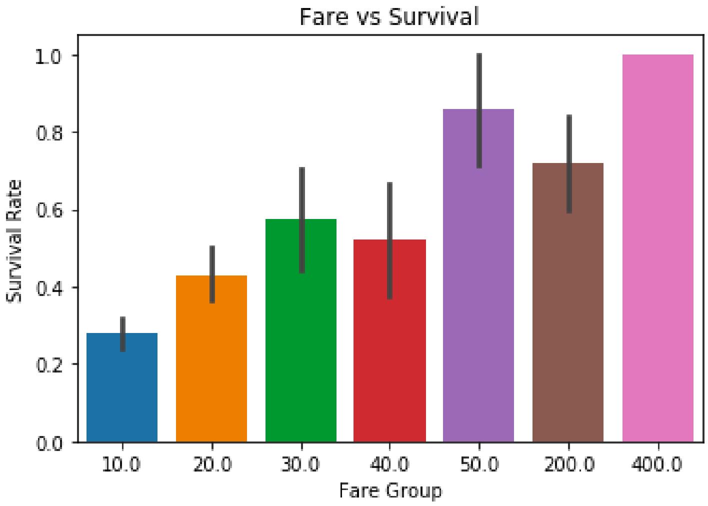
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy=’median’)
imputer.fit(X_train[:, 3:4])
Ages = imputer.transform(X_train[:, 3:4])
```

As it was in the fare variable, age is also scattered since there we many passengers of different ages. That is why I grouped the ages to make the visualization better. More detailed code can be seen in the .ipynb file. I used matplotlib and seaborn to create a histogram.

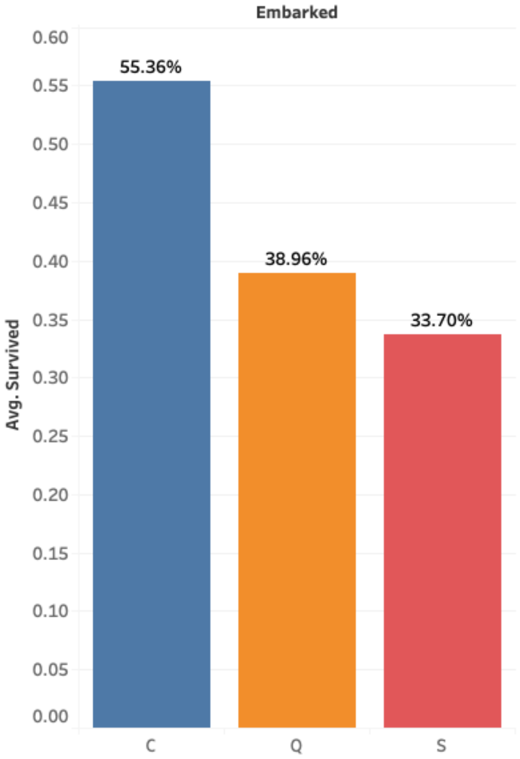
```
import matplotlib.pyplot as plt
import seaborn as sb
sb.barplot(x = Ages, y = y_train)
plt.title(’Age vs Survival Rate’)
plt.xlabel(’Age Group’)
plt.ylabel(’Average Survival Rate’)
plt.show()
```



We can see that older passengers are less likely to survive. I did the same exact process for the fare variable and below is the image that I got. We can see that passengers with higher fares seem to be more likely to survive.



One last variable that we have before we start pre-processing our data is embarked. Embarked also has missing values. Since majority of the data is not missing, I decided to again use scikit-learn to take care of the missing data. Below we can see the correlation between port of embarkation.



Since embarked is a categorical variable we cannot use the same technique as we did to take care of missing data in age. Now we use the strategy ”most frequent”.

```
imputer= SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer.fit(X[:, -1:])
X[:, -1:] = imputere.transform(X[:, -1:])
```

Let us clean the data

```python
X_train = dataset.iloc[:, 2:].values
y_train = dataset.iloc[:, 1].values # our dependent variable survived
X_test = pd.read_csv('test.csv') #importing our test set
X_test = X_test.iloc[:, 1:].values
X = np.concatenate((X_train, X_test), 0) #horizontal concatenation
```

In the above code, we split the training set into the independent and dependent variable. We import the test set and we horizontally concatenate them to make one big data-set to clean. Next, we want to take care of name and turn it into title, we do that by using the built in string split function and a for loop.

```python
i = 0
for name in X[:, 1]: # go through the name column
    X[i, 1] = name.split(',')[1].split('.')[0]
    i += 1
```

The next piece of code is to eliminate unnecessary data. We are not specifically erasing passenger id because we sliced it out in the beginning as we were creating our X train and test data-sets.

```python
X = np.delete(X, 6, axis=1) # delete ticket number
X = np.delete(X, 7, axis=1) # delete cabin number
```

Now we want to get rid of all the missing data in X. We will do this again by using scikit-learn.

```python
# taking care of missing data
from sklearn.impute import SimpleImputer
#Age, SibSp ParCh, Fare
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
imputer.fit(X[:, 3:7])
X[:, 3:7] = imputer.transform(X[:, 3:7])

#Embarked
imputere = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputere.fit(X[:, -1:])
X[:, -1:] = imputere.transform(X[:, -1:])

#Title
imputeret = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NoTitle')
imputeret.fit(X[:, 1:2])
X[:, 1:2] = imputeret.transform(X[:, 1:2])
```

Now that we have handled all missing data, we can add the number of siblings/spouses to the number of parents/children a passenger has abroad. This way we will be creating a new variable family size and we will have one less variable.

```python
i = 0
for sibsp in X[:, 4]: # go through sibsp column
    X[i, 5] = X[i, 5] + sibsp # add it to the parch colum which will create family size
    i += 1

X = np.delete(X, 4, axis=1) # delete sibsp
```

If we look through the titles that we created, it is clear that some titles can be combined. To see this I created a dictionary to keep track of how many times each title appears in the data.

```python
titles = X[:, 1]
appearances = {} #we need to organize the titles
for title in titles:
    if title not in appearances.keys():
        appearances[title] = 1
    else:
        appearances[title] += 1
```

When I printed appearances the result was as follows:

```
{' Mr': 757, ' Mrs': 197, ' Miss': 260, ' Master': 61, ' Don': 1, ' Rev': 8,
    ' Dr': 8, ' Mme': 1, ' Ms': 2, ' Major': 2, ' Lady': 1, ' Sir': 1, ' Mlle
    ': 2, ' Col': 4, ' Capt': 1, ' the Countess': 1, ' Jonkheer': 1, ' Dona':
    1}
```

So I decided some of the less common titles could be joined into a more common title. This way we will only have 6 different types of titles instead of 18.

```python
# Replacing less familiar names with more familiar names
off = ['Capt', 'Col', 'Major', 'Dr', 'Rev']
mas = ['Jonkheer', 'Master']
roy = ['Don', 'Sir', 'the Countess', 'Lady', 'Dona']
mrs = ['Mme', 'Ms', 'Mrs']
mis = ['Mlle', 'Miss']
i = 0
for title in X[:, 1]:
    if title in off:
        X[i, 1] = 'Officer'
    elif title in mas:
        X[i, 1] = 'Master'
    elif title in roy:
        X[i, 1] = 'Royalty'
    elif title in mrs:
        X[i, 1] = 'Mrs'
    elif title in mis:
        X[i, 1] = 'Miss'
    i += 1
```

Next, we need to encode our categorical data. For this we will use label encoder and one hot encoder from scikit-learn.

```python
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
for col in [1, 2]:
    X[:, col] = label.fit_transform(X[:, col])
```

The above code encodes title and gender by replacing each distinct entry with a number. when we run the command X.shape we get (1309, 7). This means we have 1309 rows and 7 columns. After we do one hot encoding we will see that these dimensions will change.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[(’encoder’, OneHotEncoder(), [-1])], remainder = ’passthrough’)
X = np.array(ct.fit_transform(X))
```

In the above code we are encoding the embarked variable, which has the values C, Q and S. We turned these values into three columns, which would be (1, 0, 0), (0, 1, 0), (0, 0, 1) not respectively. This is why if we call X.shape now we will get the dimensions (1309, 9). Now we want to split our data back into train and test.

```
# Splitting dataset into train
X_train = X[:len(X_train)]
# Splitting dataset into test
X_test = X[len(X_train):]
```

The last step is to feature scale some variables before we start modeling. This will scale down our family size and passenger fare variables.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 7:9] = sc.fit_transform(X_train[:, 7:9])
X_test[:, 7:9] = sc.transform(X_test[:, 7:9])
```

Lets us try some models

Some of the models that I trained the data with were:

- K Nearest Neighbours
- Logistic Regression
- Naive Bayes
- Support Vector Machine
- Decision Tree
- Random Forests

At the end of trying out all the models, I used k-cross validation to check the accuracy of all models.

```
from sklearn.model_selection import cross_val_score
```

K-Nearest Neighbours: Accuracy: 78.23 % Standard Deviation: 4.52 %

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = ’minkowski’, p = 2)
classifier.fit(X_train, y_train)
y_pred_KNN = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Logistic Regression: Accuracy: 79.69 % Standard Deviation: 2.45 %

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred_LR = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
```

```python
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Naive Bayes: Accuracy: 79.01 % Standard Deviation: 2.12 %

```python
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred_NB = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Support Vector Machine: Accuracy: 78.67 % Standard Deviation: 2.86 %

```python
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
y_pred_SVC = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Decision Tree Classifier: Accuracy: 79.58 % Standard Deviation: 5.19 %

```python
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
y_pred_DT = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Random Forest Classifier: Accuracy: 80.93 % Standard Deviation: 4.65 %

```python
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
y_pred_random_forest = classifier.predict(X_test)
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

We can clearly see that the winning model is the random forest classifier with a 80.9% accuracy. This model could definitely be improved which I will be working on.

Thank you :)
Nursima Donuk
-Student @ Hunter College
https://www.linkedin.com/in/nursimadonuk/
nursimadonuk@gmail.com