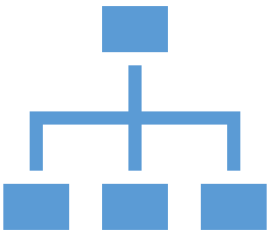
Project (1)

San Francisco Crime Classification



Nouran Hisham

hishamnouran4@gmail.com

San Francisco Crime Dataset:

It's a dataset from Kaggle about all the crimes taking place in San Fransisco for the past 12 years, how often each crime is, when it took place, where it took place and how the police forces there handled it.

Our job is to predict what crime is most likely to take place given a certain time and location, this should be done depending on the 12-year worth of data on this matter provided by Kaggle.

All work in this project is done only on the training data excel file.

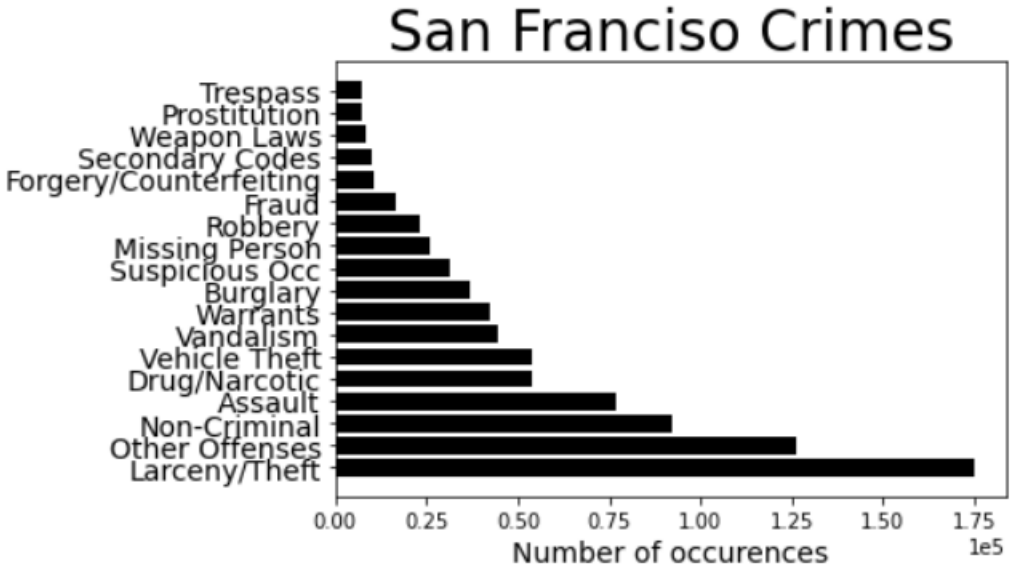
Our data consists of the following:

- Dates - timestamp of the crime incident
- Category - category of the crime incident (only in train.csv). This is the target variable you are going to predict.
- Descript - detailed description of the crime incident (only in train.csv)
- DayOfWeek - the day of the week
- PdDistrict - name of the Police Department District
- Resolution - how the crime incident was resolved (only in train.csv)
- Address - the approximate street address of the crime incident
- X - Longitude
- Y – Latitude

Main charts/plots explained:

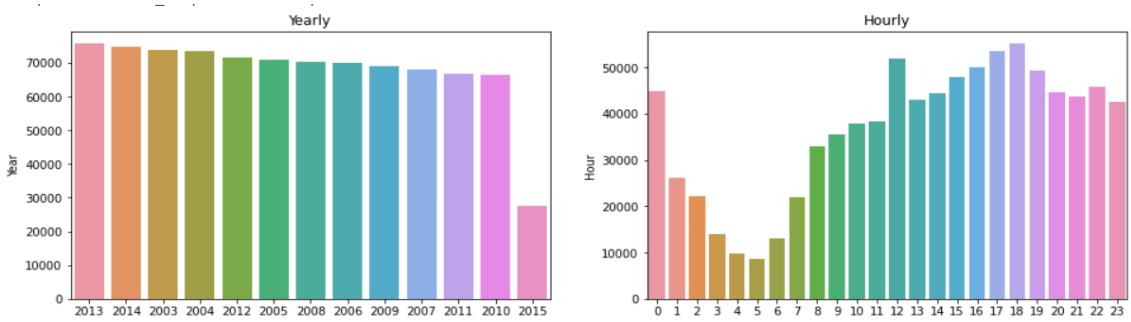
1) Bar Plot and Crime Count

This plot shows how many crimes took place over the past 12 years showing the most and least frequent crimes.



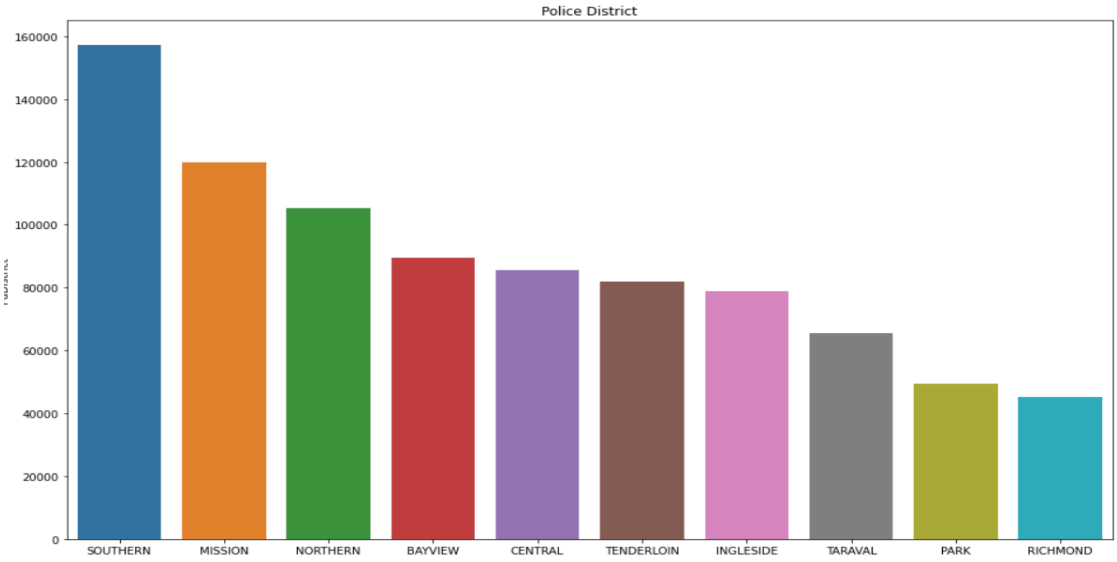
2) Bar Plot (yearly – hourly)

This plot shows how many crimes took place each year and shows the times when crimes are at their highest and lowest levels.



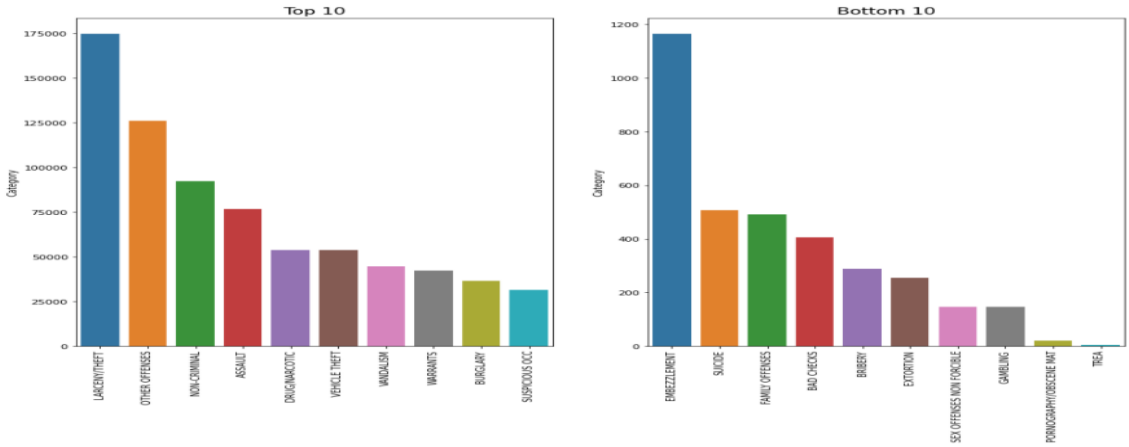
3) Bar Plot (district)

This plot shows the names of the districts where crimes most and least took place.



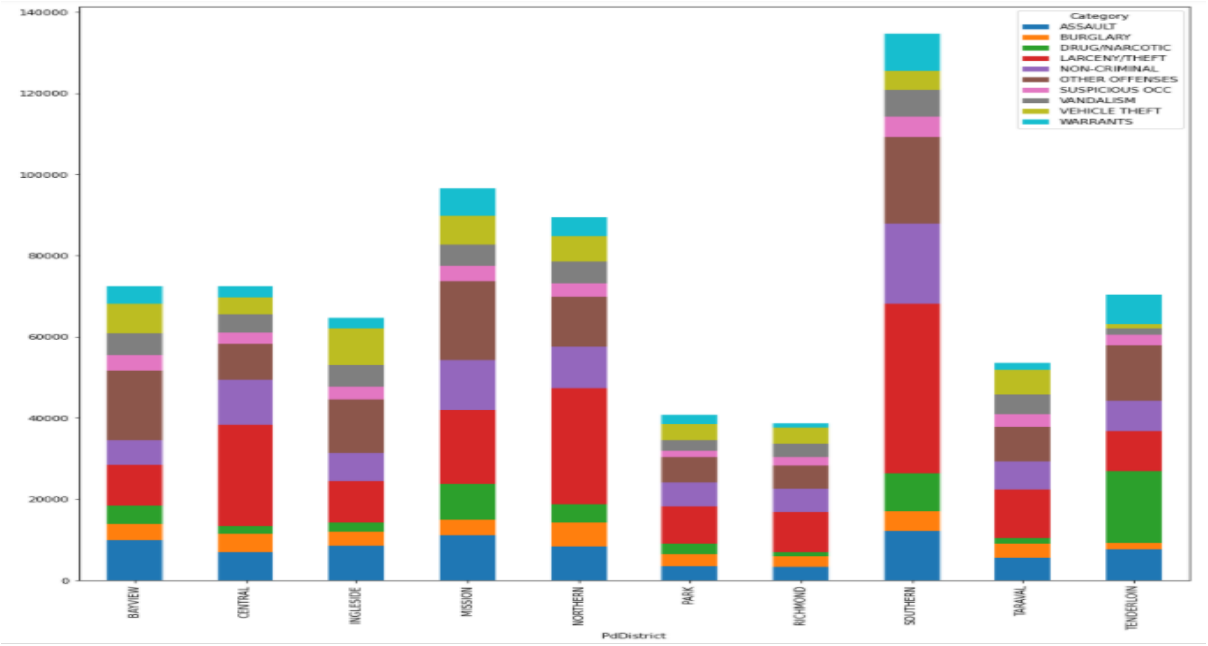
4) Bar Plot (top/bottom 10)

This plot focuses on the top 10 and bottom 10 crimes in the whole dataset to make it easier for anyone to analyse the data and reach a conclusion.



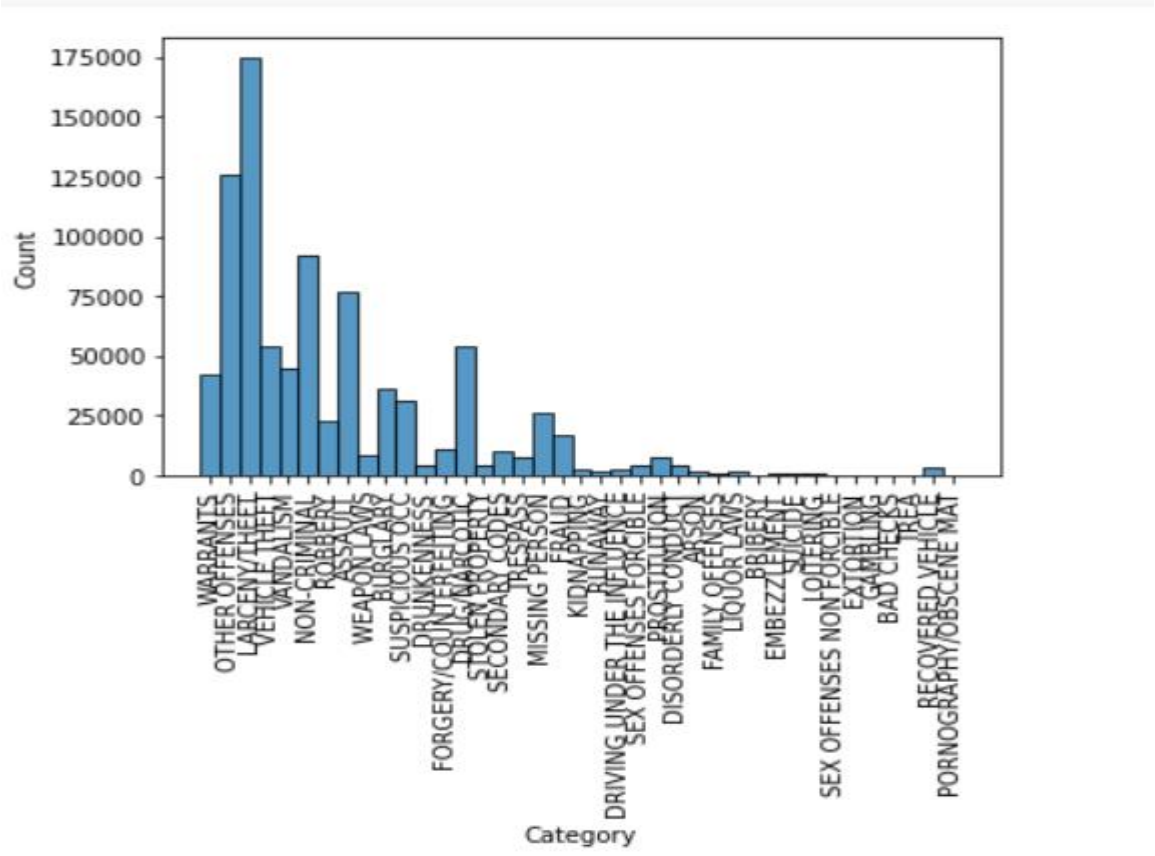
5) Stacked Bar Chart

This chart shows how often each crime is on each district.



6) Histogram Plot

This plot counts the number of each crime over the past 12 years.

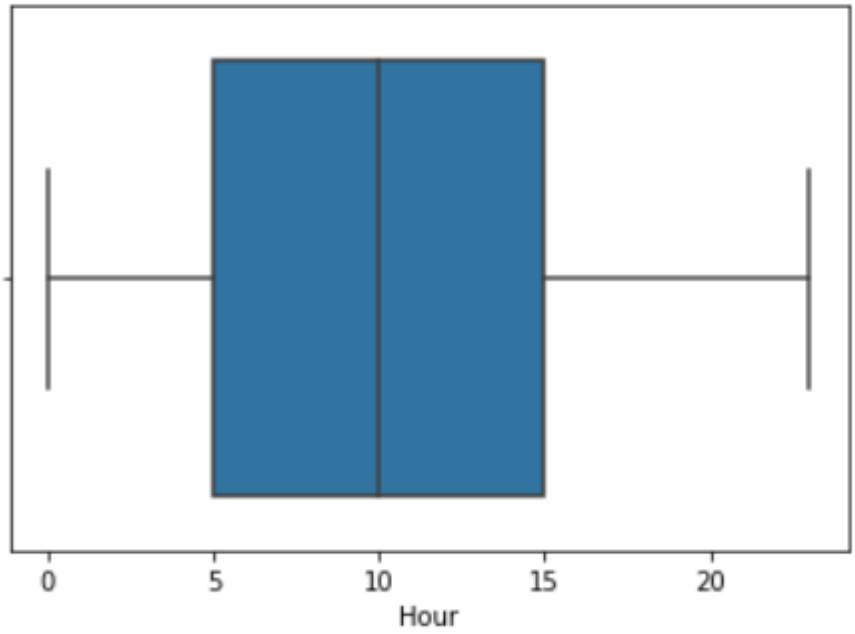


Data Cleansing:

1) Boxplots to check for outliers

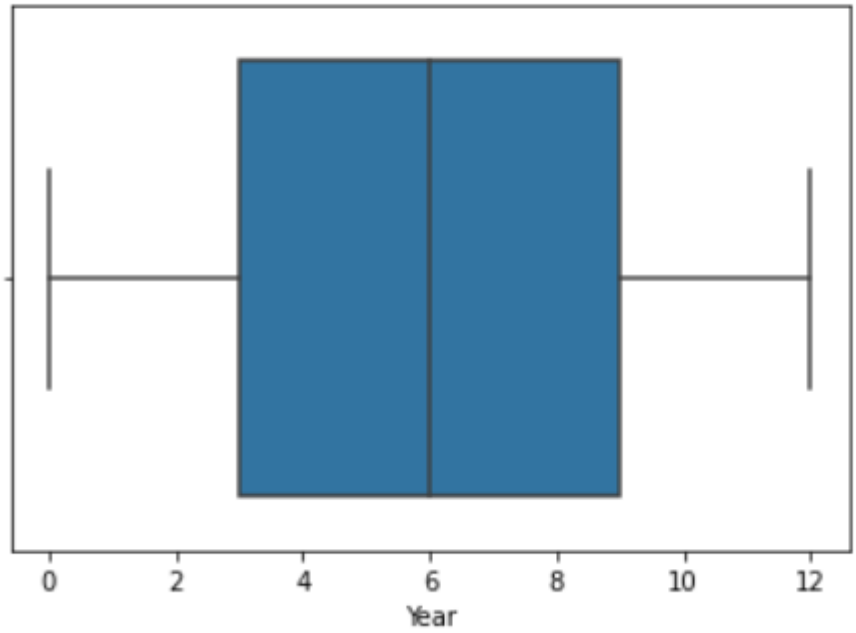
Hour (no outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3eae111fd0>
```



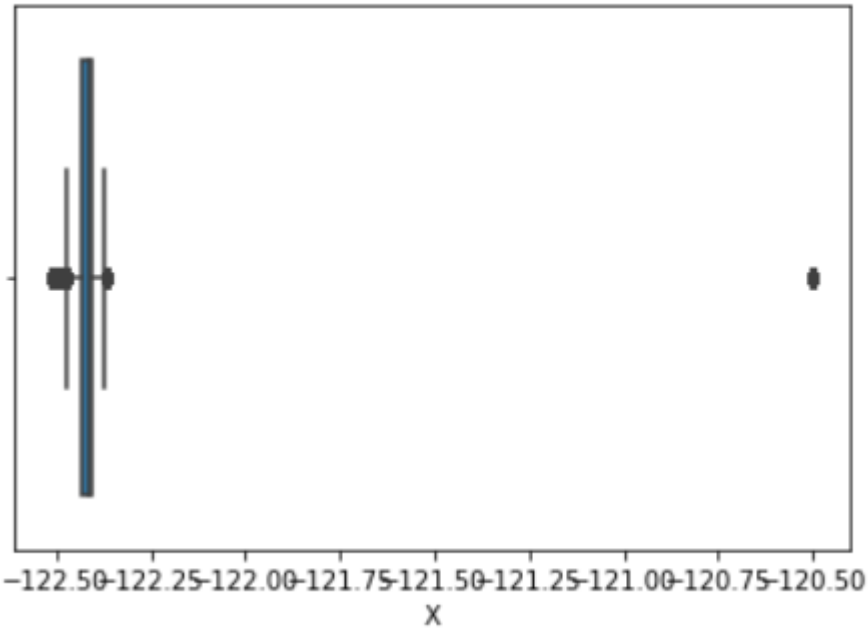
Year (no outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3eae004a50>
```



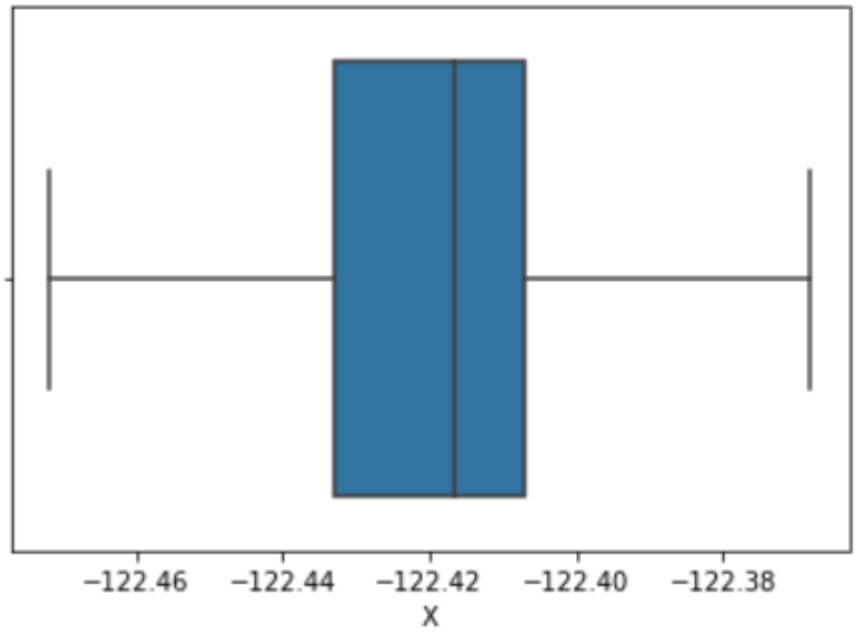
X-Longitude (before fixing outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e936677d0>
```



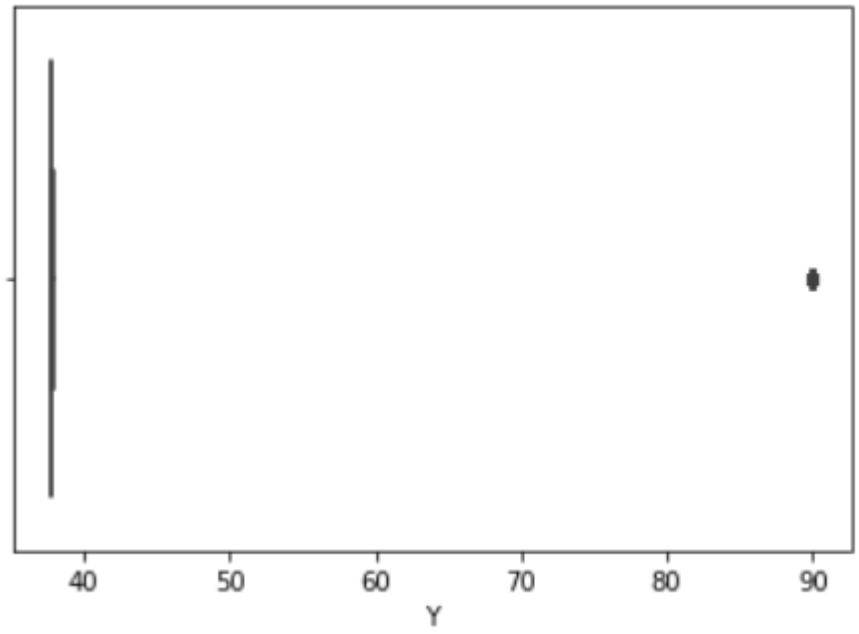
X-Longitude (after fixing outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e93531ed0>
```



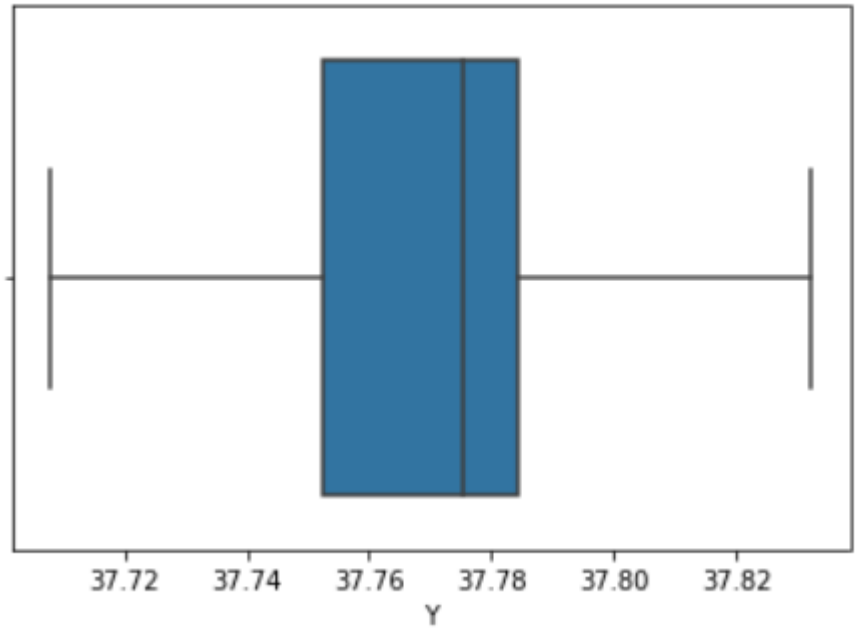
Y-Latitude (before fixing outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e935da3d0>
```



Y-Latitude (after fixing outliers)

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3e93619510>
```



Fixing outliers was done using the following function:

```
#function for outliers
def fix_outliers(df_, colName):
  q1 = df_[colName].quantile(0.25)
  q3 = df_[colName].quantile(0.75)
  range = q3-q1
  whisker_upper = q3+1.5*range
  whisker_lower = q1-1.5*range
  df_[colName] = np.where(df_[colName]>whisker_upper, whisker_upper, np.where(df_[colName]<whisker_lower, whisker_lower, df_[colName]))
  return df_

tr_data= fix_outliers(tr_data, 'X')
tr_data= fix_outliers(tr_data, 'Y')
```

2) Checking for null values

This was done using the describe(include="all") and shape functions which counts the number of fields not having null values in each column. All columns were found to have 878049 value which is the actual total number of rows in the dataset which means that no null values were found.

```
tr_data.describe(include='all')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: Fut
  """Entry point for launching an IPython kernel.
```

	Dates	Category	Descript	DayOfWeek	PdDistrict
count	878049	878049	878049	878049	878049
unique	389257	39	879	7	10
top	2011-01-01 00:01:00	LARCENY/THEFT	GRAND THEFT FROM LOCKED AUTO	Friday	SOUTHERN
freq	185	174900	60022	133734	157182
first	2003-01-06 00:01:00	NaN	NaN	NaN	NaN
last	2015-05-13 23:53:00	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN

3) Minimising features

This facilitates the training and testing processes as there are usually some features in any dataset that are of no help in our classification so in this step, I dropped these useless features.

```
[267] #dropping useless features
      tr_data.drop('Descript', inplace=True, axis=1)
      tr_data.drop('Resolution', inplace=True, axis=1)
      tr_data.drop('Dates', inplace=True, axis=1)
      tr_data.drop('Address', inplace=True, axis=1)
```

4) Changing irrelevant data types

This was done using LabelEncoder() function to change data types of type object to float or int so that the classification models can deal with easily.

```
[271] l1 = LabelEncoder()
      tr_data['PdDistrict'] = l1.fit_transform(tr_data['PdDistrict'])
      tr_data['Month'] = l1.fit_transform(tr_data['Month'])
      tr_data['DayOfWeek'] = l1.fit_transform(tr_data['DayOfWeek'])
```

Results of the models:

1) Random Forest Classifier model

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Classification report (before tuning)

4	0.00	0.00	0.00	7286
5	0.00	0.00	0.00	863
6	0.00	0.00	0.00	430
7	0.28	0.35	0.31	10751
8	0.00	0.00	0.00	829
9	0.00	0.00	0.00	229
10	0.00	0.00	0.00	55
11	0.00	0.00	0.00	101
12	0.00	0.00	0.00	2038
13	0.00	0.00	0.00	3315
14	0.00	0.00	0.00	40
15	0.00	0.00	0.00	457
16	0.27	0.82	0.40	34947
17	0.00	0.00	0.00	408
18	0.00	0.00	0.00	243
19	0.90	0.06	0.11	5109
20	0.35	0.03	0.06	18342
21	0.20	0.37	0.26	25574
22	0.00	0.00	0.00	5
23	0.76	0.11	0.20	1469
24	0.00	0.00	0.00	639
25	0.00	0.00	0.00	4665
26	0.00	0.00	0.00	381
27	0.00	0.00	0.00	2017
28	0.00	0.00	0.00	883
29	0.00	0.00	0.00	28
30	0.00	0.00	0.00	906
31	0.00	0.00	0.00	115
32	0.00	0.00	0.00	6237
33	0.00	0.00	0.00	1
34	0.00	0.00	0.00	1483
35	0.00	0.00	0.00	9005
36	0.23	0.14	0.17	10689
37	0.00	0.00	0.00	8380
38	0.00	0.00	0.00	1732
accuracy			0.25	175610
macro avg	0.08	0.05	0.04	175610
weighted avg	0.21	0.25	0.16	175610

Accuracy Score (before tuning)

```
[ ]  print("Random forest classification accuracy:",accuracy_score(y_test, rfc_pred))
```

```
Random forest classification accuracy: 0.2527817322475941
```

Random Forest Classifier has a hyperparameter that when changed can affect the accuracy of the model, this hyperparameter is n_estimators which is the number of trees in the forest.

Before tuning this parameter was set to 10 and after tuning it was set to 80.

Classification report (after tuning)

4	0.00	0.00	0.00	7286
5	0.00	0.00	0.00	863
6	0.00	0.00	0.00	430
7	0.28	0.35	0.31	10751
8	0.00	0.00	0.00	829
9	0.00	0.00	0.00	229
10	0.00	0.00	0.00	55
11	0.00	0.00	0.00	101
12	0.00	0.00	0.00	2038
13	0.00	0.00	0.00	3315
14	0.00	0.00	0.00	40
15	0.00	0.00	0.00	457
16	0.27	0.82	0.40	34947
17	0.00	0.00	0.00	408
18	0.00	0.00	0.00	243
19	0.76	0.08	0.15	5109
20	0.33	0.03	0.06	18342
21	0.20	0.37	0.26	25574
22	0.00	0.00	0.00	5
23	0.63	0.12	0.19	1469
24	0.00	0.00	0.00	639
25	0.00	0.00	0.00	4665
26	0.00	0.00	0.00	381
27	0.00	0.00	0.00	2017
28	0.00	0.00	0.00	883
29	0.00	0.00	0.00	28
30	0.00	0.00	0.00	906
31	0.00	0.00	0.00	115
32	0.00	0.00	0.00	6237
33	0.00	0.00	0.00	1
34	0.00	0.00	0.00	1483
35	0.00	0.00	0.00	9005
36	0.23	0.16	0.19	10689
37	0.00	0.00	0.00	8380
38	0.00	0.00	0.00	1732
accuracy			0.25	175610
macro avg	0.08	0.05	0.04	175610
weighted avg	0.20	0.25	0.16	175610

Accuracy Score (after tuning)

```
print("Random forest classification after tuning accuracy:",accuracy_score(y_test, rfc_pred))
```

```
Random forest classification after tuning accuracy: 0.2527817322475941
```

2) Decision Tree Classifier model

It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node.

It has no hyperparameters, so no tuning was done on it.

Classification report

4	0.11	0.12	0.12	7286
5	0.03	0.04	0.03	863
6	0.02	0.02	0.02	430
7	0.33	0.47	0.39	10751
8	0.01	0.01	0.01	829
9	0.01	0.01	0.01	229
10	0.00	0.00	0.00	55
11	0.05	0.06	0.06	101
12	0.09	0.10	0.09	2038
13	0.06	0.08	0.07	3315
14	0.10	0.07	0.08	40
15	0.02	0.03	0.03	457
16	0.37	0.34	0.36	34947
17	0.04	0.05	0.05	408
18	0.15	0.17	0.16	243
19	0.47	0.54	0.50	5109
20	0.20	0.18	0.19	18342
21	0.24	0.22	0.23	25574
22	0.00	0.00	0.00	5
23	0.52	0.52	0.52	1469
24	0.04	0.03	0.03	639
25	0.07	0.06	0.06	4665
26	0.21	0.18	0.19	381
27	0.00	0.00	0.00	2017
28	0.12	0.11	0.12	883
29	0.00	0.00	0.00	28
30	0.01	0.01	0.01	906
31	0.03	0.03	0.03	115
32	0.06	0.06	0.06	6237
33	0.00	0.00	0.00	1
34	0.04	0.03	0.03	1483
35	0.12	0.10	0.11	9005
36	0.40	0.43	0.41	10689
37	0.11	0.09	0.10	8380
38	0.11	0.08	0.09	1732
accuracy			0.24	175610
macro avg	0.11	0.12	0.11	175610
weighted avg	0.24	0.24	0.24	175610

Accuracy Score

```
[ ]  print("Decision tree accuracy:" ,accuracy_score(y_test, dtree_pred1))
```

```
Decision tree accuracy: 0.2387164740048972
```

3) K-Nearest Neighbours Classifier model

KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbours.

Classification report (before tuning)

4	0.07	0.12	0.09	7286
5	0.03	0.06	0.04	863
6	0.02	0.04	0.03	430
7	0.26	0.48	0.33	10751
8	0.01	0.02	0.01	829
9	0.01	0.02	0.01	229
10	0.01	0.02	0.01	55
11	0.03	0.04	0.03	101
12	0.06	0.10	0.08	2038
13	0.05	0.07	0.06	3315
14	0.03	0.03	0.03	40
15	0.02	0.03	0.03	457
16	0.29	0.34	0.31	34947
17	0.06	0.08	0.07	408
18	0.09	0.09	0.09	243
19	0.25	0.31	0.28	5109
20	0.14	0.13	0.13	18342
21	0.19	0.15	0.17	25574
22	0.00	0.00	0.00	5
23	0.35	0.29	0.32	1469
24	0.01	0.00	0.00	639
25	0.06	0.03	0.04	4665
26	0.07	0.04	0.05	381
27	0.00	0.00	0.00	2017
28	0.11	0.06	0.07	883
29	0.00	0.00	0.00	28
30	0.01	0.00	0.00	906
31	0.02	0.01	0.01	115
32	0.05	0.02	0.03	6237
33	0.00	0.00	0.00	1
34	0.03	0.01	0.02	1483
35	0.09	0.03	0.04	9005
36	0.27	0.08	0.13	10689
37	0.10	0.01	0.02	8380
38	0.20	0.02	0.04	1732
accuracy			0.19	175610
macro avg	0.08	0.08	0.07	175610
weighted avg	0.18	0.19	0.17	175610

Accuracy Score (before tuning)

```
[ ]  print("K-nearest neighbours accuracy:",accuracy_score(y_test, knn_pred))
```

```
K-nearest neighbours accuracy: 0.18590626957462558
```

KNN classifier has a hyperparameter called n_neighbours which is the number of neighbours to check for the distance from the starting point.

Before tuning this parameter was set to 2 and after tuning it was set to 15.

Classification report (after tuning)

4	0.08	0.04	0.06	7286
5	0.07	0.02	0.03	863
6	0.00	0.00	0.00	430
7	0.22	0.34	0.27	10751
8	0.03	0.00	0.00	829
9	0.00	0.00	0.00	229
10	0.00	0.00	0.00	55
11	0.00	0.00	0.00	101
12	0.07	0.02	0.03	2038
13	0.07	0.02	0.03	3315
14	0.00	0.00	0.00	40
15	0.05	0.00	0.00	457
16	0.28	0.54	0.37	34947
17	0.20	0.02	0.04	408
18	0.20	0.03	0.06	243
19	0.13	0.08	0.10	5109
20	0.15	0.12	0.13	18342
21	0.20	0.25	0.22	25574
22	0.00	0.00	0.00	5
23	0.28	0.34	0.31	1469
24	0.00	0.00	0.00	639
25	0.07	0.01	0.02	4665
26	0.08	0.00	0.01	381
27	0.06	0.00	0.00	2017
28	0.03	0.00	0.00	883
29	0.00	0.00	0.00	28
30	0.00	0.00	0.00	906
31	0.00	0.00	0.00	115
32	0.05	0.01	0.01	6237
33	0.00	0.00	0.00	1
34	0.12	0.01	0.02	1483
35	0.11	0.03	0.04	9005
36	0.22	0.16	0.19	10689
37	0.10	0.02	0.04	8380
38	0.05	0.00	0.01	1732
accuracy			0.21	175610
macro avg	0.08	0.06	0.05	175610
weighted avg	0.17	0.21	0.17	175610

Accuracy Score (after tuning)

```
[ ] print("K-nearest neighbours after tuning accuracy:",accuracy_score(y_test, knn_pred))

K-nearest neighbours after tuning accuracy: 0.18590626957462558
```

The best model of these three models was Random Forest Classifier model.

Future work that may be made to enhance the models is getting a larger dataset and decreasing the non-necessary features.