Crime Rate NYPD Predictions

Milestone: - Project Report

Group 14

Mayur Mahanta

Aniket Sakharkar

857-437-9190 (Mayur Mahanta)

857-230-5126 (Aniket Sakharkar)

mahanta.m@northeastern.edu

sakharkar.a@northeastern.edu

Percentage of Effort Contributed by Mayur: 50%

Percentage of Effort Contributed by Aniket: 50%

Signature of Mayur: *Mayur*

Signature of Aniket: *Aniket*

Submission Date: 24th June 2022

Problem Setting: This is a breakdown of every arrest made in New York City by the NYPD this year. Every quarter, this data is manually extracted and reviewed by the Office of Management Analysis and Planning. Each record represents an arrest made by the NYPD in NYC and contains information about the type of crime, the location, and the time of enforcement. Additionally, information about suspect demographics is included.

Problem Definition: Crime is a worldwide issue that can harm a country in both social and economic terms. Data mining provides us with several practical and convenient methods for evaluating large and diverse sets of information. It assists organizations and users in uncovering hidden information from a large database of criminal records to investigate, control, and prevent crime.

As the above setting suggests, we are in possession of data in regards of the crime rates reported by the NYPD during the year 2021.

We can use the same to display insights on:

- The crime type (Rape, Assault, etc.) that is frequently occurring.
- The age groups most likely to commit the crimes.
- The gender and race categories are mostly involved in the crimes throughout the city.

Data Sources: Website Link: NYPD Arrest Data (Year to Date) | NYC Open Data (cityofnewyork.us)

Agency: Police Department NYPD **Dataset Owner:** NYC Open Data

Data Description:

Rows: - 156K **Columns: - 19**

Data Mining techniques or algorithms will be applied to understand which are the areas that are prone to crime and profiling of the individuals involved in committing the crime will be done. Our target variable is a column, "*AGE_GROUP*". A few of the predictor variables/columns of interest are, "PERP_RACE" which represents the Perpetrator's race, and "*PERP_SEX*" which gives the Perpetrator's sex description, etc.

Data Collection:

Here, we are gathering data in a measured and systematic manner to ensure accuracy and facilitate data analysis. The "NYPD_Arrest_Data_Year_to_Date_".csv file was uploaded to Google Collaboratory and read with pd.read_csv() before being stored as a data frame.

Data Processing:

The first set of operations we performed with the uploaded data frame are as follows:

Step 1: Using shape () function on the data frame, we found out the number of rows to be 155507 and the number of columns to be 19.

Step 2: Using info () function, we found out the datatypes of each variable and their corresponding null values

Step 3: In this step, we have assigned a new Model Data Variable to store the same dataset using the function copy () whilst keeping the original data frame intact for visualisation purposes.

Step 4: Here, we are transforming all categorical variables into numeric variables using labelencoder() from the library sklearn. The variables on which we are performing this operation here are, "*PERP_SEX*," "*PERP_RACE*," "*AGE_GROUP*," "*ARREST_BORO*" and "*OFNS_DESC*", "*PD_DESC*", "*LAW_CODE*", "*LAW_CAT_CD*", "*New Geo-referenced column*", and "*ARREST_DATE*".

Step 5: After changing the categorical variables to numerical variables, we found out a few variables of interest that are "*PERP_SEX*," "*PERP_RACE*," "*ARREST_PRECINCT*," "*ARREST_BORO*" and "*OFNS_DESC*" etc.

Step 6: In order to double check null values, we have omitted all non-important object datatypes and selected only the float64 and int64 datatypes. We have executed this using isnull().sum() through which we have eliminated all null values in our Model Data frame.

Step 7: Now, using describe () function, we are retrieving the summary of the statistics pertaining to the Model dataframe.

Step 8: Moving on, we are trying to check for outliers without using visualization at this point. Hence, we are using a predefined statistical function called find_outliers_IQR() which uses the concept of finding outliers using the Interquartile formula. We have called the function on one variable called "*JURISDICTION_CODE*" to find length of the outliers, maximum outlier value and the minimum outlier value.

Data Exploration:

Data exploration is the first step in the data analysis of our dataset "*NYPD_Arrest_Data__Year_to_Date_*".csv. It is a crucial step used to investigate and envision data in order to comprehend insights from the beginning or to identify areas or patterns to investigate further. Data exploration allows us to quickly explore large amounts of data in order to better understand the next steps in terms of additional analysis. This gives us a more manageable starting point and allows us to target areas of interest in the project's later stages. In most cases, data exploration entails examining the data at a high-level using data visualizations. As a result, we stand to benefit greatly from this.

Data Visualization and Insights:

Step – 1) For the first visualization, we are going forward with plotting a Heat-Map in order to understand how the variables are correlated. In order to do that, we have utilized our original data frame (df) variable for which we have performed the corr() function. Next, we are using the Seaborn Python Data Visualization Library built on top of Matplotlib to plot a customized Heat-Map on the correlation values for our dataset variables.

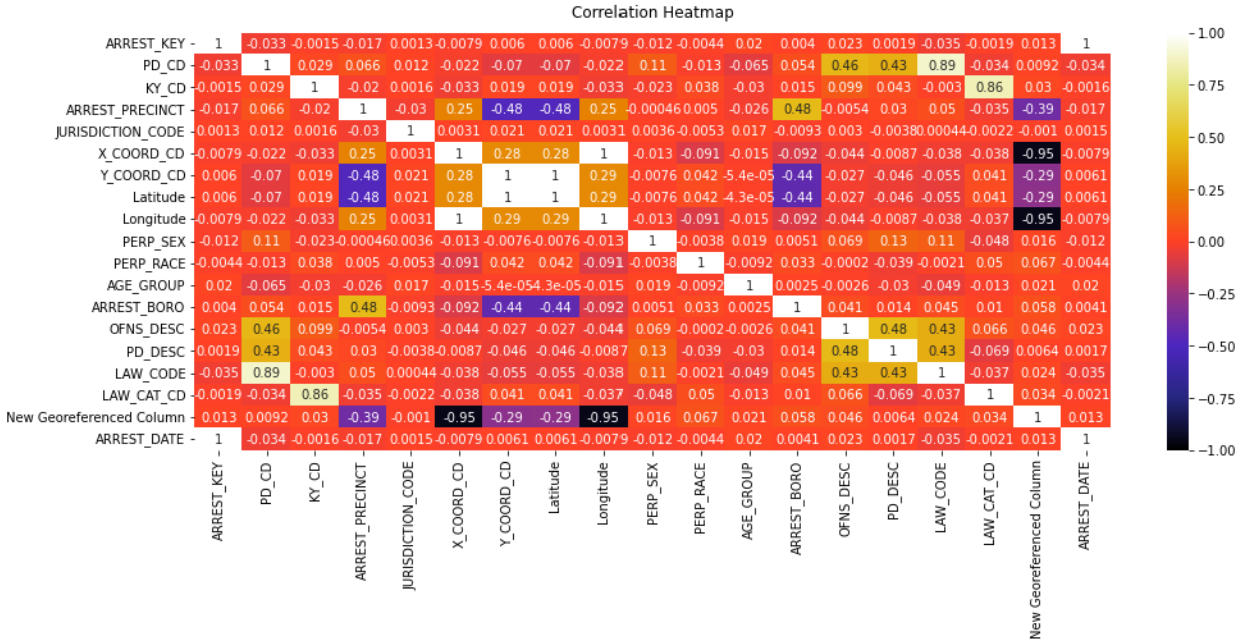

Fig-1 Heatmap

As you can see from the above Heat-Map, we are using only variables that are numeric in nature to understand the correlation between them. We have set a range of [1, -1] and then checked for the high threshold values of 0.75 and 0.50 respectively. The observation is that there are no highly correlated variables in our dataframe.

We went ahead and selected 7 variables which have slightly high negative correlation with our target variable “AGE_GROUP”. The dropped 7 variables are: “X_COORD_CD”, “Y_COORD_CD”, “LATITUDE”, “LONGITUDE”, “PD_DESC”, “LAW_CODE”, and “LAW_CAT_CD”.

```
x.drop(['X_COORD_CD','Y_COORD_CD','Latitude','Longitude','PD_DESC','LAW_CODE','LAW_CAT_CD'],axis=1)
```

	ARREST_KEY	PD_CD	KY_CD	ARREST_PRECINCT	JURISDICTION_CODE	PERP_SEX	PERP_RACE	ARREST_BORO	OFNS_DESC	New Georeferenced Column	ARREST_DATE
0	238013474	157.0	104.0	105	97	1	2	3	56	243	351
1	236943583	263.0	114.0	69	71	1	2	1	5	11340	328
2	234938876	594.0	116.0	61	0	1	5	1	58	24550	286

Fig-3: Data after dropping variables

Step – 2) A crucial step at this point is standardization. We must scale the data so we can compare the graphs easily and the visualization will be improved. The scaled data is displayed below:

```
standardized_final_data=pd.DataFrame(data=data_normalized, columns = x.columns)
standardized_final_data
```

ARREST_PRECINCT	JURISDICTION_CODE	X_COORD_CD	Y_COORD_CD	Latitude	Longitude	PERP_SEX	PERP_RACE	ARREST_BORO	OFNS_DESC	PD_DESC	LAW_CODE	LAW_CAT_CD	New Georeferenced Column	ARREST_DATE
1.185448	12.171946	2.445133	-0.140764	-0.143842	2.445638	0.454227	-0.742541	1.190968	1.479490	0.959686	-0.628739	-1.072619	-1.876072	1.587132
0.174417	8.878379	0.434042	-1.232046	-1.231973	0.431830	0.454227	-0.742541	-0.511117	-1.158436	-1.174062	-0.137556	-1.072619	-0.691980	1.366848
-0.052479	-0.115590	-0.452963	-1.753075	-1.752436	-0.454685	0.454227	0.915658	-0.511117	1.582938	0.606243	1.544255	-1.072619	0.717578	0.970242
-0.591356	8.878379	0.188695	1.051774	1.051884	0.189951	0.454227	-0.742541	-1.362159	-1.158436	-1.174062	-0.141486	-1.072619	-0.317662	0.941771
-0.534632	-0.115590	0.008051	1.287559	1.287783	0.009249	0.454227	-0.742541	-1.362159	1.841558	1.823657	1.088437	0.864425	-0.009929	0.818393
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
-0.477908	-0.115590	0.298967	1.318926	1.318939	0.300849	0.454227	-0.742541	-1.362159	1.531214	1.129862	0.007834	-1.072619	-0.506848	-1.658655
0.117693	-0.115590	-0.002586	-0.910320	-0.909937	-0.004016	0.454227	-0.742541	-0.511117	-0.744644	1.745115	1.626774	-1.072619	0.010452	-1.430882
1.649239	-0.115590	-2.887814	-1.397674	-1.398150	-2.886577	0.454227	-0.742541	2.042010	-1.106712	-1.147881	-1.229948	0.864425	1.661054	-1.335976
0.287865	-0.115590	0.117441	-0.655707	-0.655417	0.116286	0.454227	-0.742541	-0.511117	-0.744644	1.745115	1.626774	-1.072619	-0.192819	-1.715600
1.110362	-0.115590	1.227126	-0.581069	-0.581942	1.225936	0.454227	-1.295275	1.190968	-0.899816	-0.663534	-0.212216	-1.072619	-1.386728	-1.402410

Fig-4 Standardized Data

Step – 3) Next, we are plotting a scatter plot matrix to see the relationships between combination of variables. The scatter plot matrix is plotted on the original data frame (**Fig -4**) after label encoding. The figure is plotted on the next page.

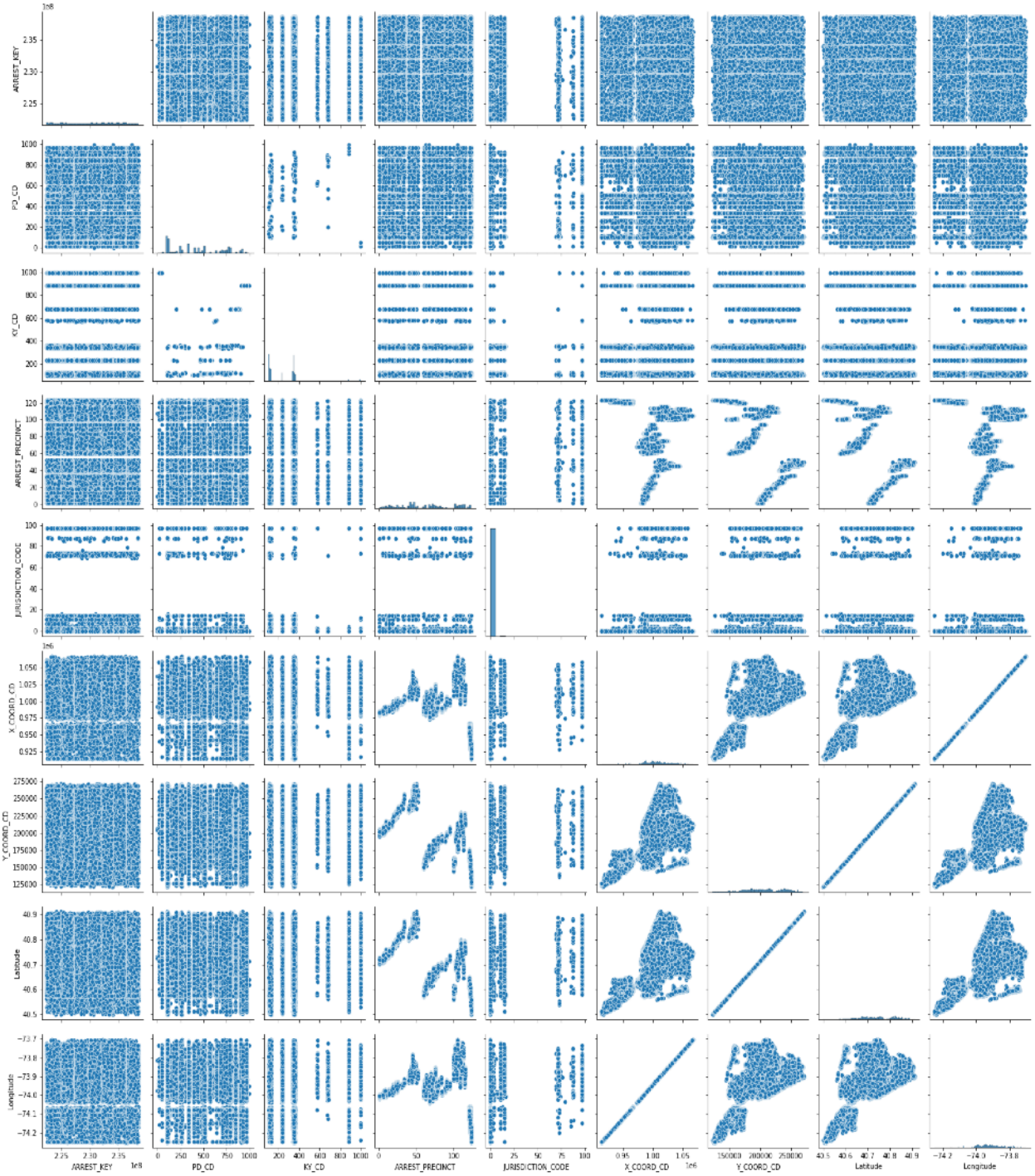

Fig-4: Scatter Plot Matrix (Original Dataframe)

Step –4) Then, we have plotted the scatter plot matrix on the data after applying PCA (**Fig-5**). The scatter plot matrix is plotted on the next page.

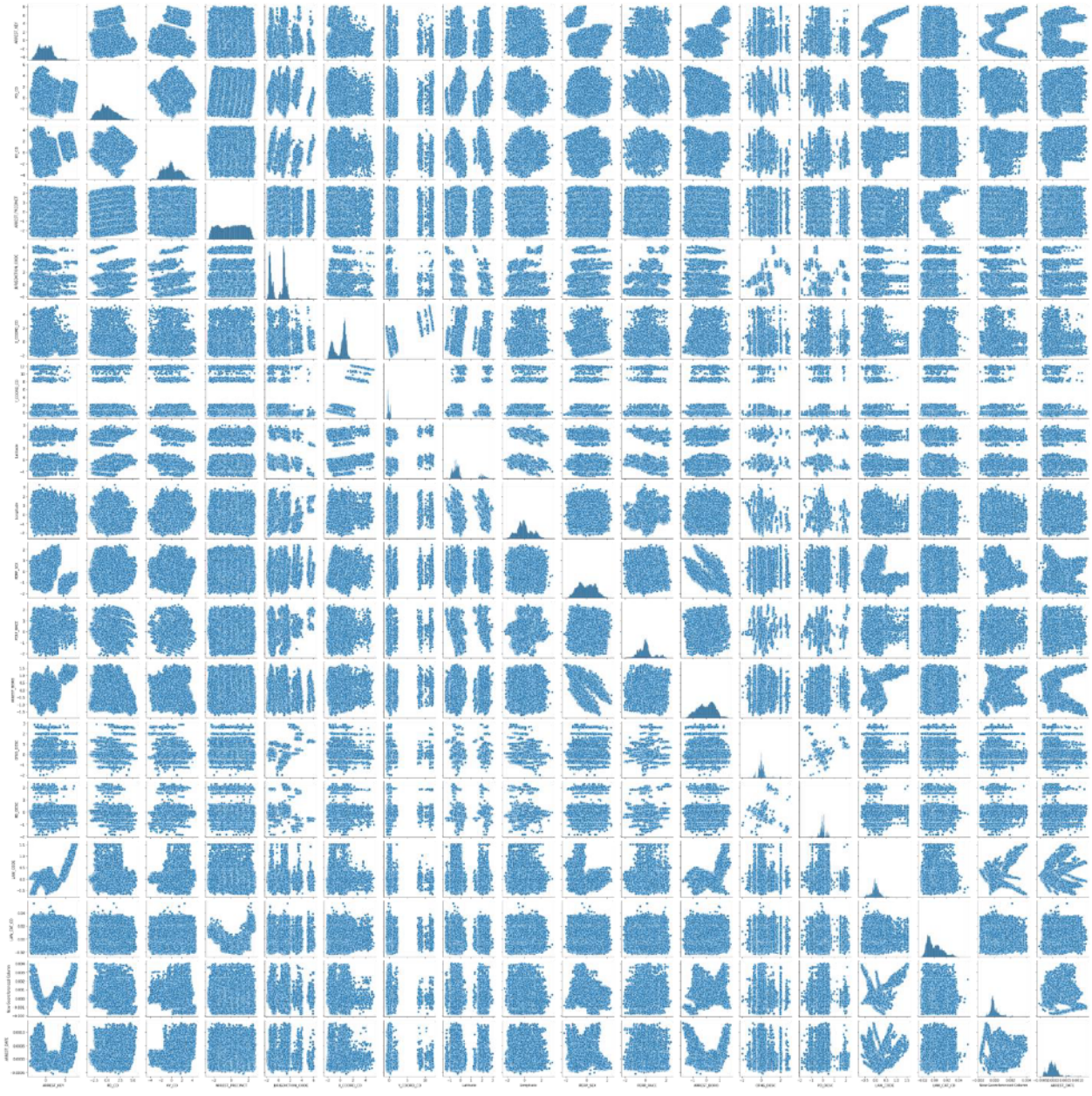

Fig-5: Scatter Plot Matrix (PCA applied Dataframe)

One of the observations from the above Scatter Plot Matrix is that most relationships are weak in nature. And negative correlation can be seen for example between “Sex” and “Offence”.

Step – 5) Moving on, we have added a boxplot for the following variables: “PERP_SEX”, “PERP_RACE”,” OFNS_DESC”, “ARREST_PRECINCT”, “ARREST_KEY”, “ARREST_BORO”, to see the outliers and the distribution of the predictors. Below is the box plot displayed along with the outliers.

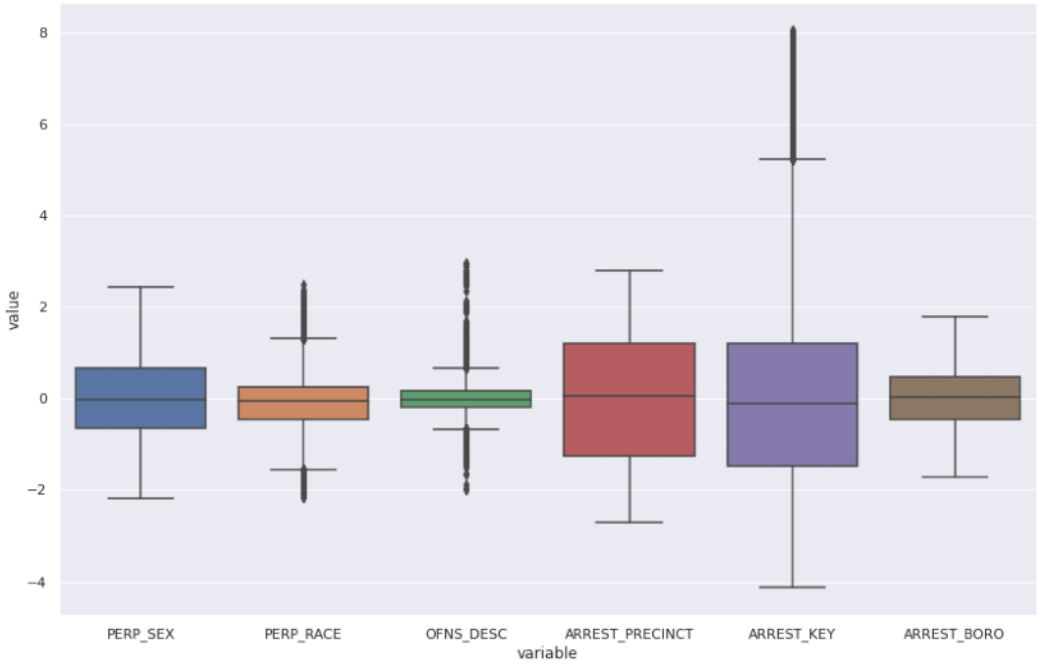

Fig-6: Box Plot with outliers

To observe the distribution of the predictor variables more precisely, we have removed the outliers and have plotted it below:

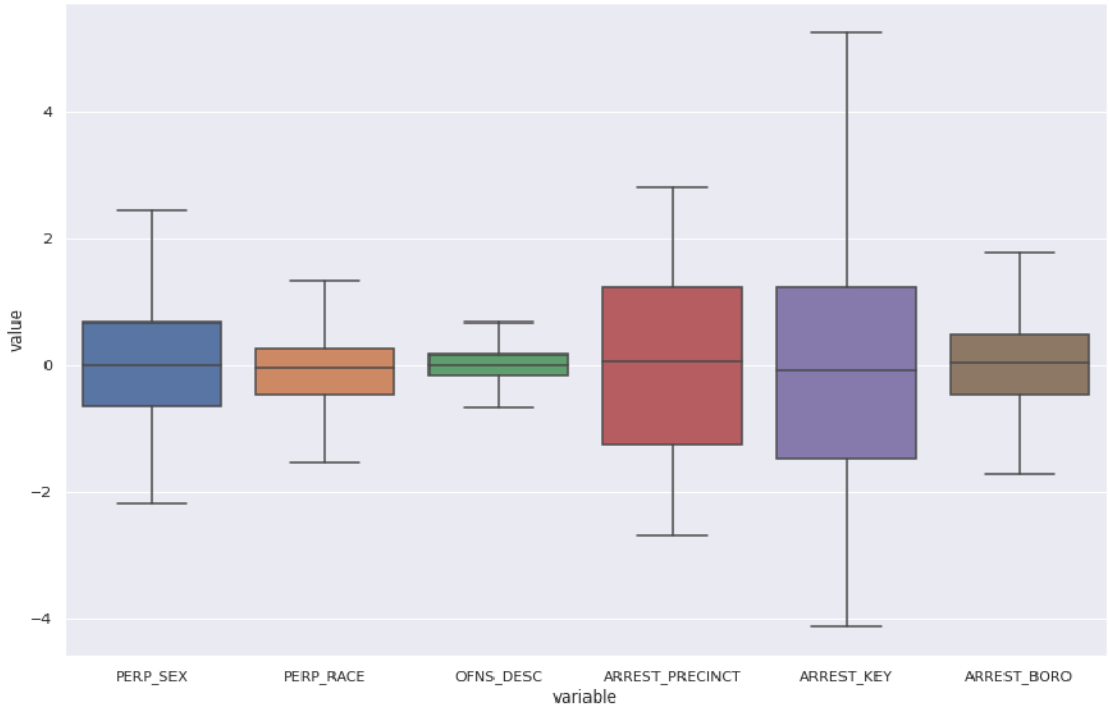

Fig -7: Box Plot without outliers

As we can see from the above figure that the distribution of the predictor variables is varying significantly, we can conclude that they are good predictors.

Step - 6) Next, we are plotting the pie chart using the pie() function from Plotly Express or PX Module. The variables concerned for this pie chart are “ARREST_PRECINCT” and “PERP_RACE”.

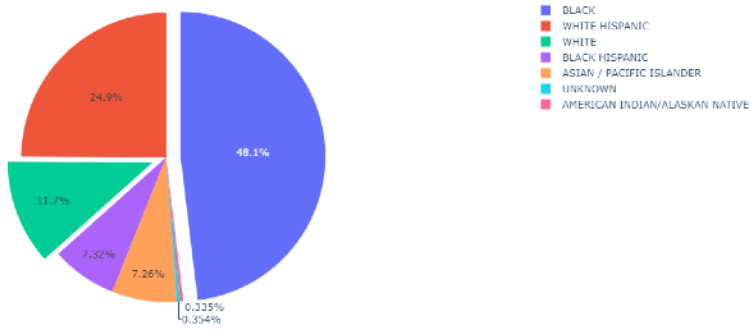

Fig-8: Pie chart

According to this Pie chart, people of the black race had the highest percentage of Arrest Precinct records, followed by White Hispanics with 24 percent and White with more than 11 percent. Furthermore, when compared to all other races, American Indians/Alaska Natives have the lowest Arrest Precinct. Black Hispanic and Asian/Pacific Islanders had identical percentages of more than 7%.

Step -7) Lastly, as our dataset corresponds to the state of New York and its Boroughs, we decided to display an interactive animated map box giving us a quick glance at the Race types and Age Groups involved in committing crime and mapped over a daily period for the year 2021.

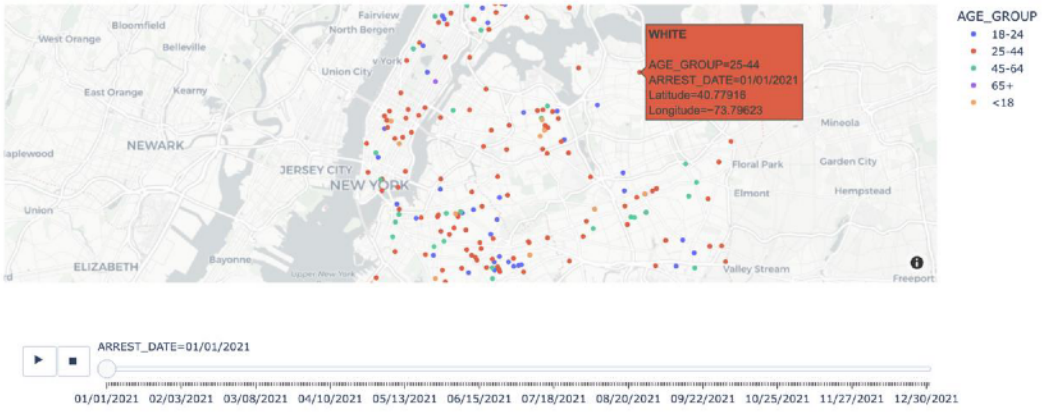

Fig-9: Scatter Map Box

For data visualization, we used Plotly Express, which allows us to quickly and efficiently visualize small amounts of data. In this step, we show how to create efficient animations with a scatter map box. The map depicts the arrest dates in New York City, as well as the age groups committing crime and their race.

Model Exploration:

When we talk about Model Exploration, the first thing we should know about is Model Taxonomy. Let us start with a categorization that broadly describes two types of models:

- Supervised
- Unsupervised
- Combination of both

In supervised settings, we have a target or group of targets that we want to predict using a set of predictor variables. This is the most prevalent scenario, and we will focus on supervised learning for our dataset "Crime Rate NYPD." It is typical in machine learning to distinguish between regression and classification among supervised models or between numerical and categorical target variables. The primary goal of this model exploration from our project perspective is to predict the target variable. It will also make way for explanation in the later stages of our project which is also one of the purposes of Model Exploration. Moving on, firstly we are checking for any missing null values in our standardized dataframe using the isnull().sum() function. Below is the output for the same.

```
[ ] print(standardized_final_data.isnull().sum())

    ARREST_KEY           0
    ARREST_PRECINCT      0
    JURISDICTION_CODE    0
    X_COORD_CD           0
    Y_COORD_CD           0
    Latitude             0
    Longitude            0
    PERP_SEX             0
    PERP_RACE            0
    AGE_GROUP            0
    ARREST_BORO          0
    OFNS_DESC            0
    dtype: int64
```

Fig 10: Null Values check in the standardized dataframe

Next, we are treating "AGE_GROUP" as the target variable for this model evaluation. Consequently, we removed the same variable from our standardized data set to obtain input and output sets.

For the model selection, we have performed a splitting operation using the train_test_split function with a ratio of 80:20 or 80% split for the training data and 20% split for the test data corresponding to both input predictor variables and output response variable. Below is the shape of the sliced data to understand if the split worked accordingly.

```
#from sklearn.model_selection import train_test_split

y= standardized_final_data.AGE_GROUP

#input
x=standardized_final_data.drop('AGE_GROUP',axis=1)

#splitting
#x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
#Splitting the dataset into training and testing variables
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=20)

#printing shapes of testing and training sets :
print("shape of original dataset :", standardized_final_data.shape)
print("shape of input - training set", x_train.shape)
print("shape of output - training set", y_train.shape)
print("shape of input - testing set", x_test.shape)
print("shape of output - testing set", y_test.shape)
```

```
shape of original dataset : (155507, 12)
shape of input - training set (124405, 11)
shape of output - training set (124405,)
shape of input - testing set (31102, 11)
shape of output - testing set (31102,)
```

Fig 11: Predictor variables and Response Variable Split

Model Selection:

In the context of machine learning, model selection can have a variety of interpretations, each corresponding to a distinct level of abstraction. For example, we might be interested in determining which hyperparameters are appropriate for a particular machine learning algorithm. Hyperparameters are learning method parameters that must be specified before model fitting.

For this milestone, we are concentrating on evaluating the performance of our training and testing data using 5 models. The dataset and cross-validation parameters will be passed to the crossvalscore() method, which will return a list of scores calculated for each fold. Throughout the

models, we are using cross_val_score to return score of the test fold (in all the cases, K-folds = 2). The score for each model is obtained (displayed below) based on which the model is selected

Step – 1) First model we are using is the KNeighborsClassifier. K-Nearest Neighbours is one of Machine Learning's most basic but important classification algorithms. Pattern recognition, data mining, and intrusion detection are just a few of the applications it finds in the supervised learning domain. KNN classifies the new data points based on the similarity measure of the earlier stored data points.

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

Model1=KNeighborsClassifier()

cv1 = cross_val_score(Model1, x_train.astype(int), y_train.astype(int), cv= 2)

cv1
```

```
array([0.41353311, 0.47715508])
```

Fig 12: KNeighborsClassifier

Step –2) Second model is based on RandomForestClassifier. From a randomly selected portion of the training set, the Random Forest classifier generates a set of decision trees. It consists of a set of decision trees (DT) derived from a randomly selected subset of the training set, which then accumulates votes from various decision trees to get the final prediction.

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

Model2= RandomForestClassifier()

cv2 = cross_val_score(Model2,x_train.astype(int), y_train.astype(int), cv= 2)

cv2
```

```
array([0.56762536, 0.56660558])
```

Fig 13: RandomForestClassifier

Step – 3) Next model we are working on is the MLPClassifier. MLP stands for multi-layer perception. It consists of dense layers that are totally connected and turn any input dimension into the required dimension. A neural network with numerous layers is known as a multi-layer

perception. A multi-layer perceptron contains one input layer with one neuron (or node) for each input, one output layer with a single node for each output, and it can have any number of hidden layers with any number of nodes for each hidden layer.

```
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
Model3= MLPClassifier()

cv3 = cross_val_score(Model3, x_train.astype(int), y_train.astype(int), cv= 2)

cv3
```

```
array([0.5712747 , 0.57155718])
```

Fig 14: MLPClassifier

Step – 4) For Model 4, we are utilizing LogisticRegression. Logistic regression is a supervised classification algorithm. For a given collection of features (or inputs), X, the target variable (or output), y, can only take discrete values in a classification problem. When a decision threshold is introduced, logistic regression transforms into a classification technique. Setting the threshold value is a crucial part of Logistic regression, and it is influenced by the classification problem. It is not the best model for our data.

```
[ ] from sklearn.model_selection import cross_val_score
    from sklearn.linear_model import LogisticRegression
    Model4= LogisticRegression()
    cv4 = cross_val_score(Model4, x_train.astype(int), y_train.astype(int), cv= 2)
    cv4
```

```
array([0.571548  , 0.57155718])
```

Fig 15: LogisticRegression

Step – 5) Lastly, we are using the Gaussian based on the Naïve-Bayes Classification. The Naive Bayes Classifier Algorithm is a collection of probabilistic algorithms based on Bayes' theorem and the "naive" assumption of conditional independence between each pair of features. The Bayes theorem determines the probability P(c|x), where c is the class of probable outcomes and x is the supplied case to be identified, which represents some specific characteristics.

$$P(c|x) = P(x|c) * P(c) / P(x)$$

Natural language processing (NLP) problems are where Naïve-Bayes are most utilized.

```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
Model5= GaussianNB()
cv5 = cross_val_score(Model5, x_train.astype(int), y_train.astype(int), cv= 2)
cv5
```

```
array([0.56691799, 0.56662165])
```

Fig 16: GaussianNB()

The models for which we checked are KNeighborsClassifier, RandomForestClassifier, MLPClassifier, LogisticRegression, GaussianNB.

Next, we have incorporated a DecisionTreeClassifier. Below, we are computing the score obtained from the cross_val_score and then we have found out the accuracy of the same.

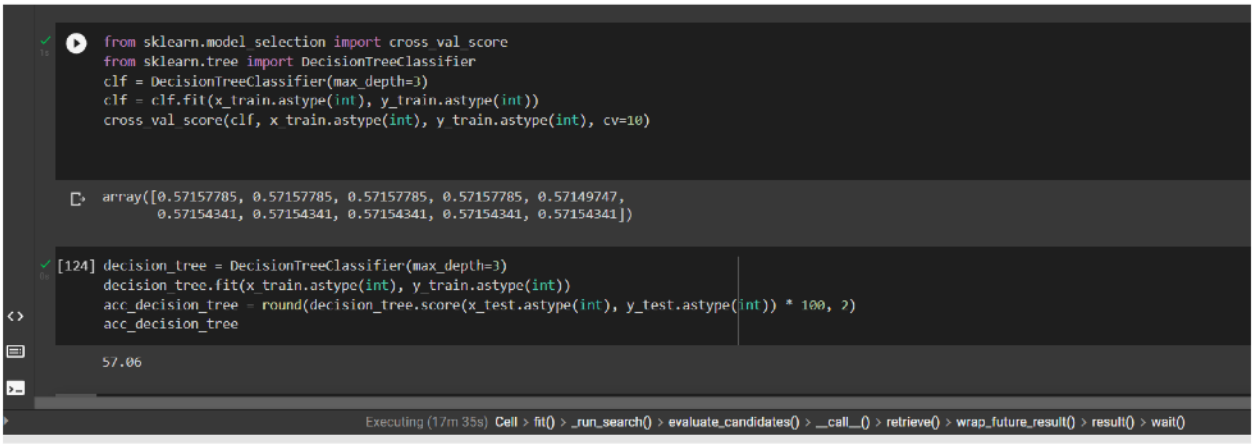

Fig 17: Cross_val_score and accuracy score of the decision tree

After that, we have plotted the decision tree with a depth of 3 where we can see particular gini_index for each level and split. The figure is plotted below for reference.

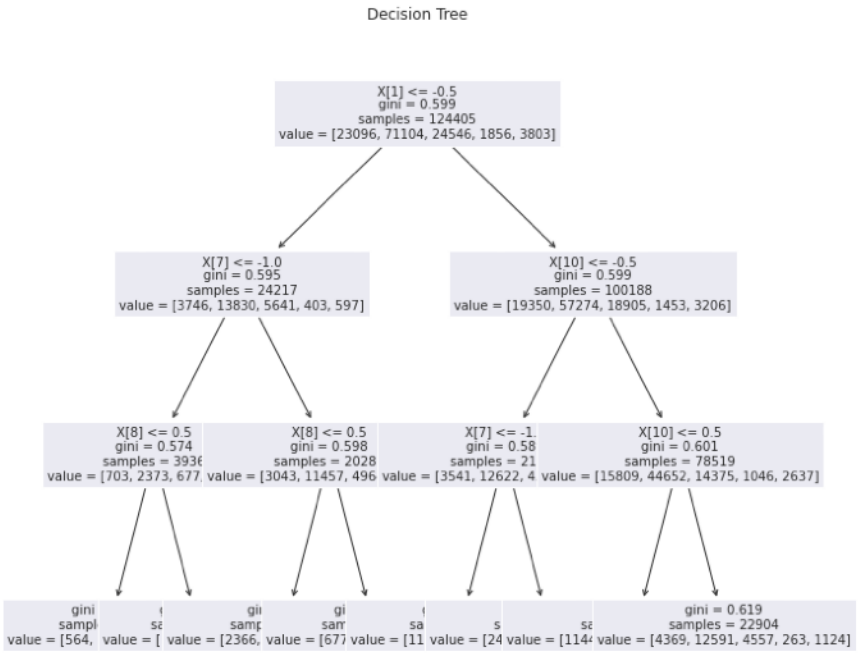

Fig 18: Decision Tree

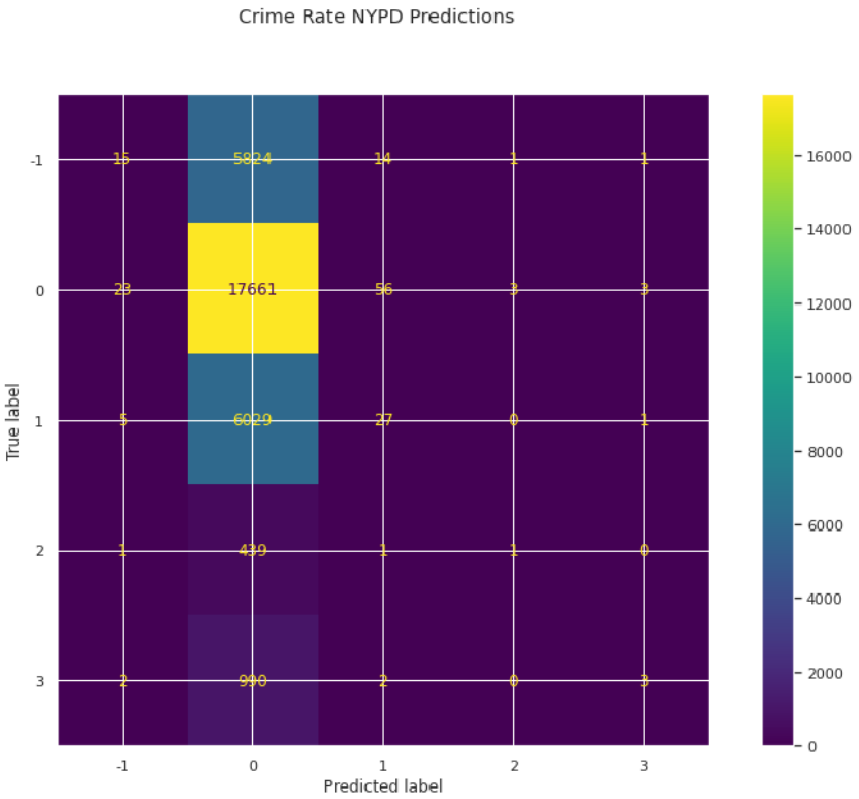

Fig 19: Confusion Matrix for MLP Classifier

Now, we observed that out of all the models evaluated above, we saw the highest cross_val_score in the MLP Model. So, we are interested in finding the confusion matrix and the accuracy score, precision, recall and f-1 score of only the MLP Model and have disregarded the other classifiers. As a result, at the last stage of this project milestone, we are plotting a confusion matrix using the plot_confusion_matrix function where we have passed the test data along with the MLP Model. Using the above confusion matrix (predicted class and actual class), we can compute the precision, recall, f1-score etc. And finally, we are finding the accuracy score for the MLP Model.

```
[133] print(classification_report(y_test.astype(int), y_pred))

                  precision    recall  f1-score   support

              -1       0.33      0.00      0.01      5855
               0       0.57      1.00      0.73     17746
               1       0.27      0.00      0.01      6062
               2       0.20      0.00      0.00       442
               3       0.38      0.00      0.01       997

        accuracy                           0.57     31102
       macro avg       0.35      0.20      0.15     31102
    weighted avg       0.45      0.57      0.42     31102
```

Fig 20: Classification Report for MLP Classifier

From the above score, we can conclude which model to use for our final implementation. We will implement Random Forest Classifier Model for our project as it has the highest accuracy amongst our models.

Implementation of the Selected Models:

After overviewing the accuracy scores from the Model Exploration milestone, for the models, KNeighborsClassifier, RandomForestClassifier, MLPClassifier, LogisticRegression and GaussianNB. We attempted to tweak the accuracy scores by re-evaluating some of the predictor variables and their correlation with our target variable 'AGE_GROUP.' After careful observation, we find that the removal of a few variables slightly impacts the accuracy score. We removed the variables based on their correlation values with our target variable. The variables that were removed are: "X_COORD_CD," "Y_COORD_CD," "Latitude," "Longitude", "PD_DESC," "LAW_CODE," "LAW_CAT_CD."

```
[26] x.drop(['X_COORD_CD','Y_COORD_CD','Latitude','Longitude','PD_DESC','LAW_CODE','LAW_CAT_CD'],axis=1)
```

	ARREST_KEY	PD_CD	KY_CD	ARREST_PRECINCT	JURISDICTION_CODE	PERP_SEX	PERP_RACE	ARREST_BORO	OFNS_DESC	New Georeferenced Column	ARREST_DATE
0	238013474	157.0	104.0	105	97	1	2	3	56	243	351
1	236943583	263.0	114.0	69	71	1	2	1	5	11340	328
2	234938876	594.0	116.0	61	0	1	5	1	58	24550	286
3	234788259	263.0	114.0	42	71	1	2	0	5	14848	283
4	234188790	578.0	NaN	44	0	1	2	0	63	17732	270
...	...	...	...	...	...	...	...	...	...	...	...
155502	222884924	397.0	105.0	46	0	1	2	0	57	13075	9
155503	223918625	792.0	118.0	67	0	1	2	1	13	17923	33
155504	224323770	101.0	344.0	121	0	1	2	4	6	33392	43
155505	222599533	792.0	118.0	73	0	1	2	1	13	16018	3
155506	224056881	268.0	121.0	102	0	1	1	3	10	4829	36

155507 rows × 11 columns

Fig 21: Data frame before standardizing and after dropping the irrelevant variables

We can see the increase in accuracy for few of the models below:

LogisticRegression		precision	recall	f1-score	support
	0	0.00	0.00	0.00	7129
	1	0.57	1.00	0.73	22338
	2	0.00	0.00	0.00	7623
	3	0.00	0.00	0.00	586
	4	0.00	0.00	0.00	1201
accuracy				0.57	38877
macro avg		0.11	0.20	0.15	38877
weighted avg		0.33	0.57	0.42	38877

Fig 22: Precision, Recall, F1-Score, Support and Accuracy for Logistic Regression

GaussianNB		precision	recall	f1-score	support
	0	0.32	0.04	0.07	7129
	1	0.58	0.97	0.72	22338
	2	0.00	0.00	0.00	7623
	3	0.03	0.02	0.02	586
	4	0.44	0.00	0.01	1201
accuracy				0.57	38877
macro avg		0.27	0.21	0.16	38877
weighted avg		0.40	0.57	0.43	38877

Fig 23: Precision, Recall, F1-Score, Support and Accuracy for GaussianNB

RandomForestClassifier		precision	recall	f1-score	support
	0	0.43	0.26	0.32	7129
	1	0.63	0.83	0.72	22338
	2	0.47	0.27	0.34	7623
	3	0.31	0.07	0.11	586
	4	0.55	0.29	0.38	1201
accuracy				0.59	38877
macro avg		0.48	0.34	0.37	38877
weighted avg		0.56	0.59	0.55	38877

Fig 24: Precision, Recall, F1-Score, Support and Accuracy for Random Forest Classifier

MLP Classifier		precision	recall	f1-score	support
	0	0.41	0.03	0.05	7129
	1	0.58	0.98	0.73	22338
	2	0.44	0.04	0.07	7623
	3	0.00	0.00	0.00	586
	4	0.33	0.00	0.01	1201
accuracy				0.57	38877
macro avg		0.35	0.21	0.17	38877
weighted avg		0.50	0.57	0.44	38877

Fig 25: Precision, Recall, F1-Score, Support and Accuracy for MLPClassifier

KNeighborsClassifier		precision	recall	f1-score	support
	0	0.46	0.00	0.00	7129
	1	0.58	1.00	0.73	22338
	2	0.45	0.02	0.03	7623
	3	0.00	0.00	0.00	586
	4	0.00	0.00	0.00	1201
accuracy				0.58	38877
macro avg		0.30	0.20	0.15	38877
weighted avg		0.51	0.58	0.43	38877

Fig 26: Precision, Recall, F1-Score, Support and Accuracy for KNeighborsClassifier

The MSE, MAE and RMSE are commonly used in regression analysis to assess prediction error rates and model performance. The difference between the actual and predicted values obtained by averaging the absolute difference over the data set is represented by MAE (Mean absolute error). MSE (Mean Squared Error) represents the difference between the original and predicted values extracted by squared the average difference over the data set. RMSE (Root Mean Squared Error) is the error rate by the square root of MSE. After the above scores, we all have computed the confusion matrix for all the above-mentioned models.

```
Confusion Matrix Report for the models are:
LogisticRegression
 [[    0  7129     0     0     0]
 [    0 22338     0     0     0]
 [    0  7623     0     0     0]
 [    0   586     0     0     0]
 [    0  1201     0     0     0]]
GaussianNB
 [[  270  6815     0    43     1]
 [  421 21742     1   171     3]
 [   85  7471     0    66     1]
 [    5   572     0     9     0]
 [   58  1132     0     7     4]]
MLP Classifier
 [[  204  6856    65     0     4]
 [  215 21828   289     0     6]
 [   22  7293   306     0     2]
 [    3   557    26     0     0]
 [   57  1127    11     0     6]]
RandomForestClassifier
 [[ 1827  4728   421    10   143]
 [ 1773 18646  1760    43   116]
 [  400  5113  2048    34    28]
 [   31   401   113    40     1]
 [  222   589    44     1   345]]
KNeighborsClassifier
 [[   13  7083    33     0     0]
 [   11 22230    97     0     0]
 [    0  7508   115     0     0]
 [    0   579     7     0     0]
 [    4  1196     1     0     0]]
```

Fig 27: Confusion Matrix for all the models

Now, we know that a N x N confusion matrix is used to evaluate the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values to the machine learning model's predictions. This provides us with a comprehensive picture of how well our classification model is working and the types of errors it makes.

Next, we are checking the accuracy of the split data using DecisionTreeClassifier to draw comparison with the other models on visualization grounds. The classifier is instantiated, and the model is fitted onto the data now that the data is completely prepared. The Gini Index criterion was chosen for this classifier by default, but the entropy might also be employed. We use the classifier model to predict values once our model fits the data. We then calculate the accuracy score using the score on the decision tree which gives us accuracy percentage to work out whether underfitting and overfitting is occurring (which in a default case, we need to ensure). However, we can notice some overfitting. Overfitting is a frequent problem with Decision Tree Classifiers. Changes in data may result in unnecessary changes in the result.

```
[42] from sklearn.model_selection import cross_val_score
     from sklearn.tree import DecisionTreeClassifier
     clf_dc = DecisionTreeClassifier(max_depth=3)
     clf_dc = clf_dc.fit(x_train, y_train)
     cross_val_score(clf_dc, x_train, y_train, cv=10)

     array([0.57035068, 0.57035068, 0.57026494, 0.57026494, 0.57026494,
            0.57026494, 0.57026494, 0.57026494, 0.57026494, 0.57026494])

[43] decision_tree = DecisionTreeClassifier(max_depth=3,random_state=42)
     decision_tree.fit(x_train, y_train)
     acc_decision_tree = round(decision_tree.score(x_test, y_test) * 100, 2)
     acc_decision_tree

     57.46
```

Fig 28: Plotting the decision tree

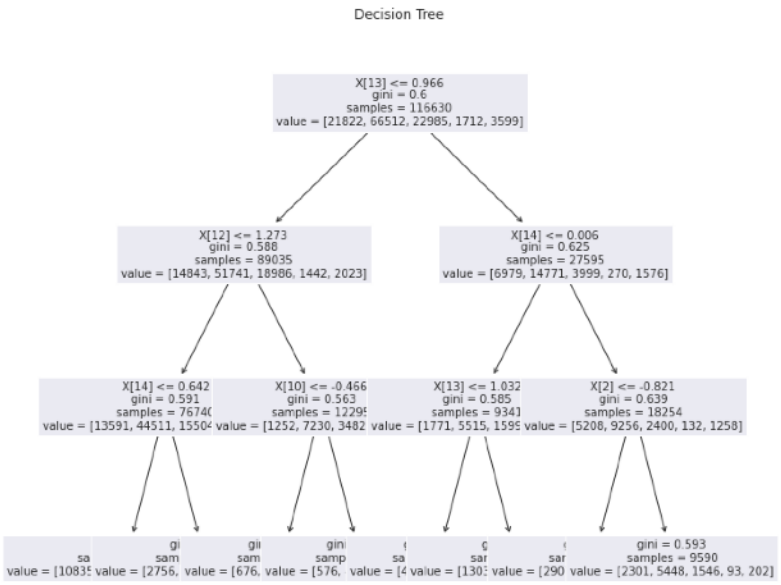

Fig 29: Decision Tree Map (DecisionTreeClassifier)

As it was suggested, we have attempted hyper-parameter tuning. We have used the cost_complexity_pruning_path. Pruning is one of the approaches used to overcome our Overfitting problem. Pruning, in its most literal sense, is the removal of specific sections of a tree (or plant), such as branches, buds, or roots, to improve the tree's structure and encourage healthy growth. Pruning our Decision Trees accomplishes the same thing. It makes it adaptable, allowing it to adapt to whatever new type of data we feed it, avoiding the problem of overfitting. It shrinks the size of a Decision Tree, which may increase your training error marginally but reduces your testing error, making it more adaptive.

```
[ ] y_predict = clf.predict(x_test)

[ ] path = clf.cost_complexity_pruning_path(x_train,y_train)

    ccp_alphas, impurities = path.ccp_alphas, path.impurities

[ ] ccp_alphas

    array([0.00000000e+00, 2.44974951e-06, 2.85804110e-06, ...,
           1.27601809e-03, 1.27846576e-03, 2.78603767e-03])

[ ] impurities

    array([1.28611849e-04, 1.35961098e-04, 1.38819139e-04, ...,
           5.94420528e-01, 5.96977460e-01, 5.99763498e-01])
```

Fig 30: Hyper-parameter tuning (Effective Alpha and Impurities)

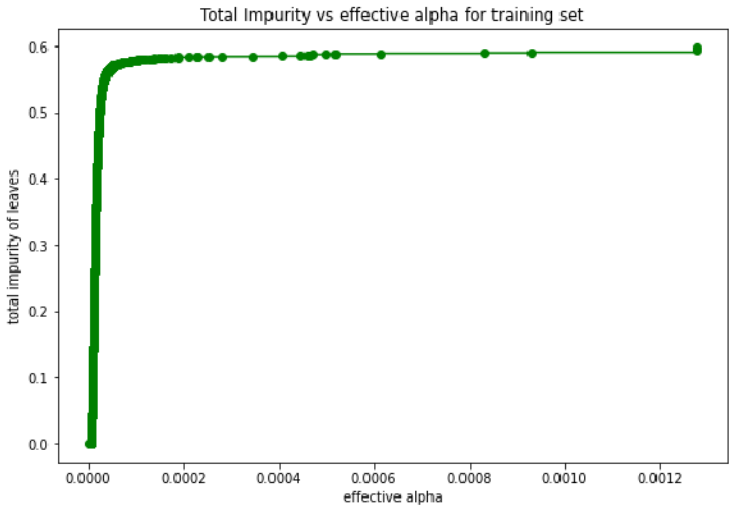

Fig 31: Effective alpha vs Total Impurity

By observing the above graph, the effective or optimal alpha value to be selected for minimizing the cost_complexity and for pruning is around 0.00001 which can be used to optimize the overfitting observed in our dataset.

All the models were evaluated with different hyperparameters to fit the data and find out the classification report (Precision, Recall, F-1 Score and Support), their respective accuracy along with the RMSE, MSE and MAE scores. We have incorporated a loop to evaluate all the models at a single go for maximum optimization. It is displayed below for reference:

```
lr={}
gnb={}
mlp={}
rfc={}
classifier={}
regr_1={}
regr_2={}
import datetime
datetime.datetime.now()

for i in range(0, 1):

    # Create classifiers

    lr[i] = LogisticRegression(C=20,random_state= 82).fit(x_train, y_train).predict(x_test)
    gnb[i] = GaussianNB().fit(x_train,y_train).predict(x_test)
    mlp[i]= MLPClassifier(random_state=42).fit(x_train, y_train).predict(x_test)
    rfc[i] = RandomForestClassifier(n_estimators= 700).fit(x_train, y_train).predict(x_test)
    classifier[i] = KNeighborsClassifier(n_neighbors=500).fit(x_train, y_train).predict(x_test)

    print(datetime.datetime.now())
    print("Accuracy Score for the models are: ")

    print('LogisticRegression',accuracy_score(y_test,lr[i]))
    print('GaussianNB',accuracy_score(y_test,gnb[i]))
    print('MLP Classifier',accuracy_score(y_test,mlp[i]))
    print('RandomForestClassifier',accuracy_score(y_test,rfc[i]))
    print('KNeighborsClassifier',accuracy_score(y_test,classifier[i]))
```

Fig 32: Running all the models and evaluating scores in the loop

```
print("Classification Report for the models are: ")

print('LogisticRegression',classification_report(y_test,lr[i]))
print('GaussianNB',classification_report(y_test,gnb[i]))
print('MLP Classifier',classification_report(y_test,mlp[i]))
print('RandomForestClassifier',classification_report(y_test,rfc[i]))
print('KNeighborsClassifier',classification_report(y_test,classifier[i]))

print("Confusion Matrix Report for the models are: ")

print('LogisticRegression\n',confusion_matrix(y_test,lr[i]))
print('GaussianNB\n',confusion_matrix(y_test,gnb[i]))
print('MLP Classifier\n',confusion_matrix(y_test,mlp[i]))
print('RandomForestClassifier\n',confusion_matrix(y_test,rfc[i]))
print('KNeighborsClassifier\n',confusion_matrix(y_test,classifier[i]))
```

Fig 33: Classification report code for all models (Contd.)

```
print("\n\t RMSE \n")
lrrmse = math.sqrt(mse(y_test,lr[i]))
gnbrmse = math.sqrt(mse(y_test,gnb[i]))
mlprmse = math.sqrt(mse(y_test,mlp[i]))
rfcrmse = math.sqrt(mse(y_test,rfc[i]))
knnrmse = math.sqrt(mse(y_test,classifier[i]))
print('LogisticRegression',lrrmse)
print('GaussianNB',gnbrmse)
print('MLP Classifier',mlprmse)
print('RandomForestClassifier',rfcrmse)
print('KNeighborsClassifier',knnrmse)

print("\n\t MSE \n")
lrmse = mse(y_test,lr[i])
gnbmse = mse(y_test,gnb[i])
mlpmse = mse(y_test,mlp[i])
rfcmse = mse(y_test,rfc[i])
knnmse = mse(y_test,classifier[i])
print('LogisticRegression',lrmse)
print('GaussianNB',gnbmse)
print('MLP Classifier',mlpmse)
print('RandomForestClassifier',rfcmse)
print('KNeighborsClassifier',knnmse)
```

Fig 34: RMSE and MSE code for all models (Contd.)

```
print("\n\t MAE \n")
lrmae = mae(y_test,lr[i])
gnbmae = mae(y_test,gnb[i])
mlpmae = mae(y_test,mlp[i])
rfcmae = mae(y_test,rfc[i])
knnmae = mae(y_test,classifier[i])
print('LogisticRegression',lrmae)
print('GaussianNB',gnbmae)
print('MLP Classifier',mlpmae)
print('RandomForestClassifier',rfcmae)
print('KNeighborsClassifier',knnmae)

N = 5
ind = np.arange(N)
width = 0.25
f, ax = plt.subplots(figsize=(25,15))

allrmsemodelval = np.array([0.001,0.002,0.003,0.004,0.005],dtype=float)
allrmsemodelval[0] = lrrmse
allrmsemodelval[1] = gnbrmse
allrmsemodelval[2] = mlprmse
allrmsemodelval[3] = rfcrmse
allrmsemodelval[4] = knnrmse
#allrmsemodelval = [lrrmse, gnbrmse, mlprmse, rfcrmse, knnrmse]
bar1 = plt.bar(ind, allrmsemodelval, width, color = 'r')
```

Fig 35: MAE along with bar chart performance evaluation code for all models (Contd.)

```
allmsemodelval = np.array([0.001,0.002,0.003,0.004,0.005],dtype=float)
allmsemodelval[0] = lrmse
allmsemodelval[1] = gnbmse
allmsemodelval[2] = mlpmse
allmsemodelval[3] = rfcmse
allmsemodelval[4] = knnmse
#allmsemodelval = [lrmse, gnbmse, mlpmse, rfcmse, knnmse]
bar2 = plt.bar(ind+width, allmsemodelval, width, color='g')

allmaemodelval = np.array([0.001,0.002,0.003,0.004,0.005],dtype=float)
allmaemodelval[0] = lrmae
allmaemodelval[1] = gnbmae
allmaemodelval[2] = mlpmae
allmaemodelval[3] = rfcmae
allmaemodelval[4] = knnmae
#allmaemodelval = [lrmae, gnbmae, mlpmae, rfcmae, knnmae]
bar3 = plt.bar(ind+width*2, allmaemodelval, width, color = 'b')

plt.xlabel("Models")
plt.ylabel('Scores')
plt.title("RMSE vs MSE vs MAE for all Models", fontsize=18)

ax.set_xticklabels(x, fontsize=18)
ax.set_yticklabels(y, fontsize=18)
plt.yticks(fontsize=18)
plt.legend(fontsize = 18)

plt.xticks(ind+width,['LogisticRegression', 'GaussianNB', 'MLPClassifier', 'RandomForestClassifier', 'KNeighborsClassifier'])
plt.legend( (bar1, bar2, bar3), ('RMSE', 'MSE', 'MAE') )
plt.show()
```

Fig 36: Bar chart performance evaluation code for all models (Contd.)

After computing all the above-mentioned scores, we have plotted a bar chart to compare the RMSE vs MSE vs MAE for all the 5 models and as we can see RandomForestClassifier clearly has better scores when compared with the other models.

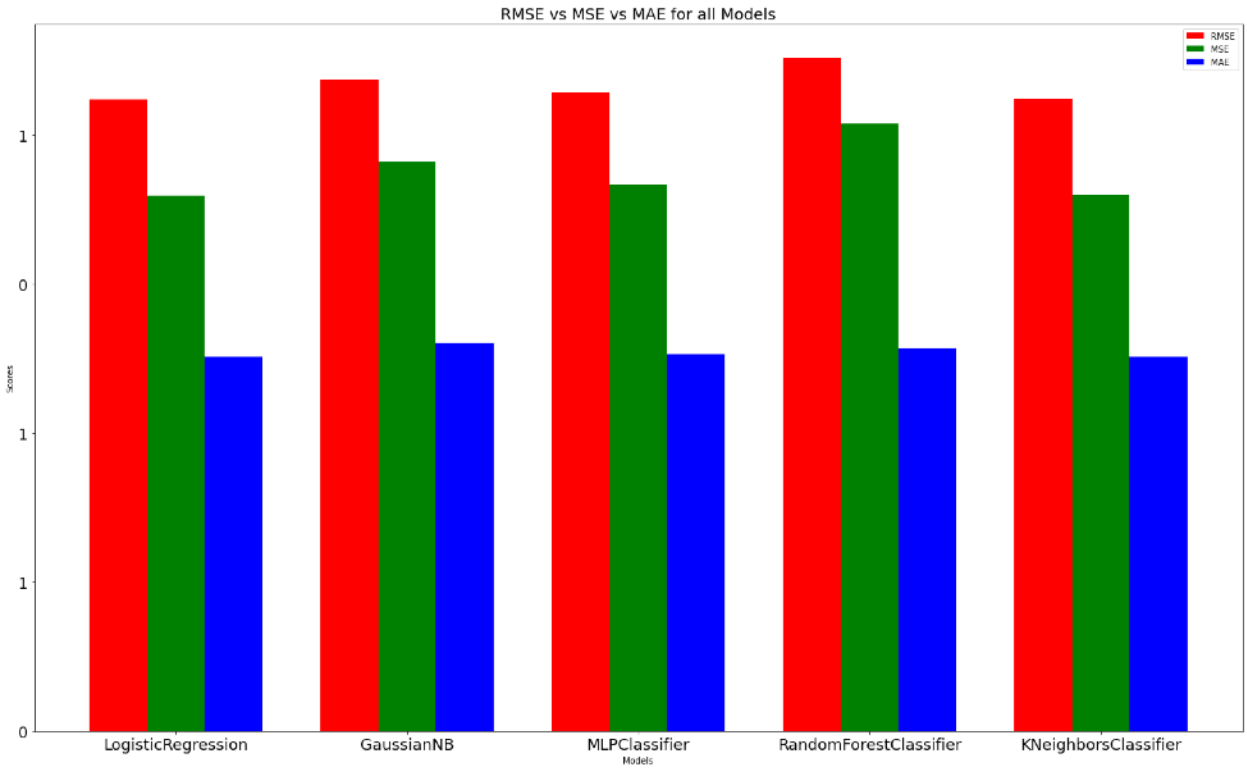

Fig 37: Comparison of RMSE vs MSE vs MAE for all the 5 models

Now, after reviewing the accuracy scores for one final time before selecting and implementing it, we can conclude that RandomForestClassifier would be used for our project as it has the highest accuracy of all other models. We had gone forth with MLPClassifier initially, however after dropping a few more variables and again standardizing and using the same data frame to split as well as fit into the models, we see RandomForestClassifier would be the best fit model for our project.

Next, we have done a performance visualization of the confusion matrix for the best suited models. The first one being RandomForestClassifier and the latter being MLPClassifer. The visualizations can be found below:

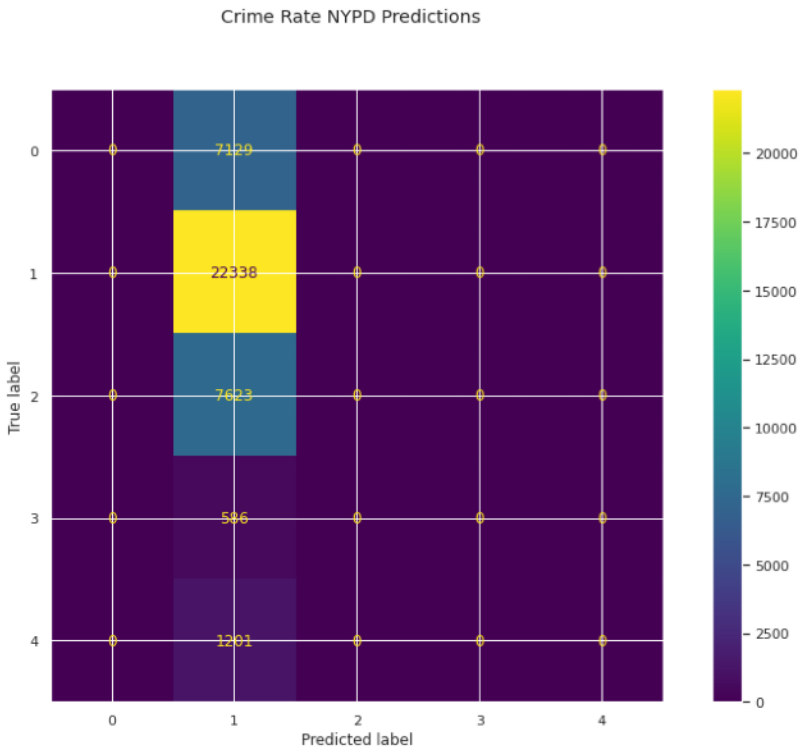

Fig 38: Confusion Matrix for RandomForestClassifier

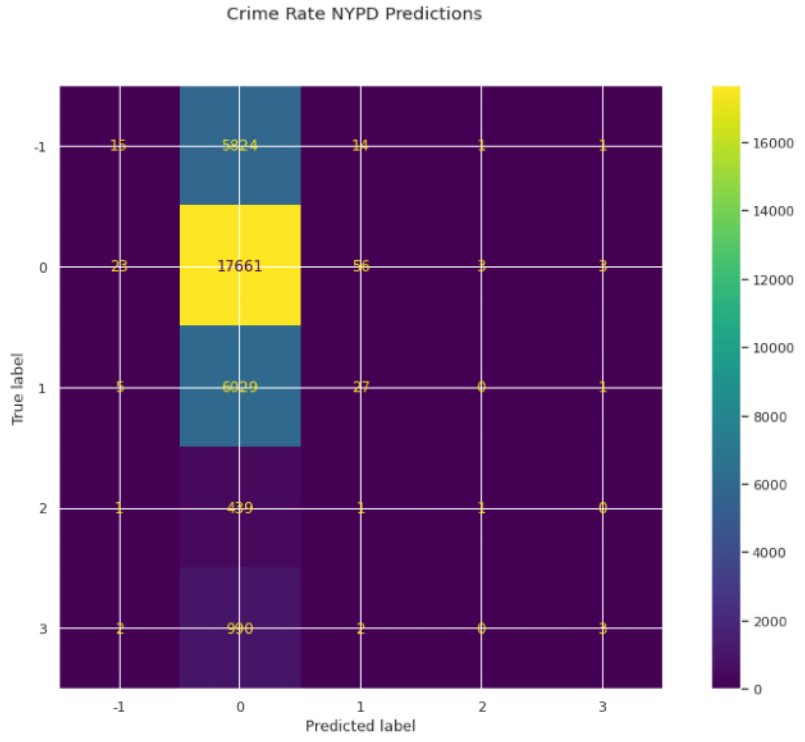

Fig 39: Confusion Matrix for MLP Classifier

Results

Actual vs Predicted Values

Now, after implementing our Model (RandomForestClassifier), we have plotted a few comparison graphs for the actual values and the predicted values for our target variable.

First, we are plotting a scatter plot which can be seen below. We can deduce issues with our model from this scatter plot. Heteroskedastic residuals are present. This indicates that the error variance is not constant across various levels of your dependent variable. As a result, our standard errors are unreliable and may be understated and consecutively the statistical significance of our independent variables might be overstated. To put it another way, they could be statistically insignificant and because of the heteroskedastic issue, we cannot tell.

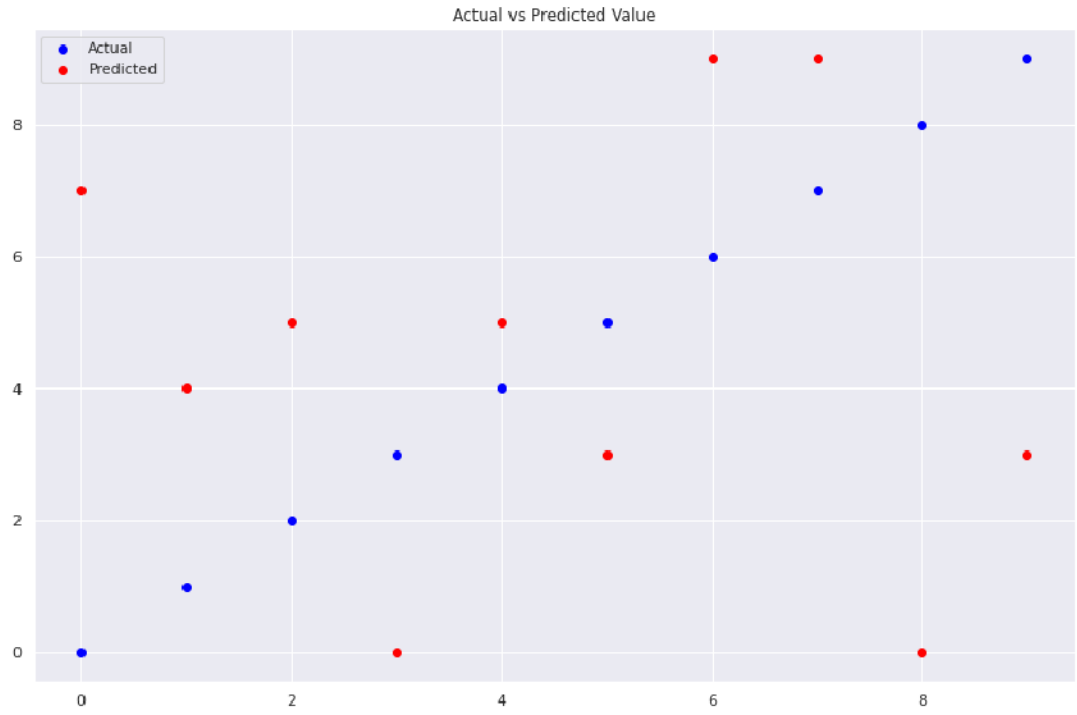

Fig 40: Scatter Plot for Actual vs Predicted Value

Second, we are going forward with a line chart to see the fitting for our target variable or for the Actual vs Predicted values. We know a fit refers to how closely you approximate a target function in statistics. The goodness of fit is a term used in statistics to describe metrics used to estimate how well a function's approximation matches the target function. Observing our line chart, we see that the predicted values do not match the actual values and severe overfitting is demonstrated by the model.

When our model learns the information and noise in the training data to the point, it degrades the model's performance on fresh data. This means that the model notices noise or random fluctuations in the training data and learns them as ideas. This can be overcome by hyperparameter tuning or further optimization of the fitted data.

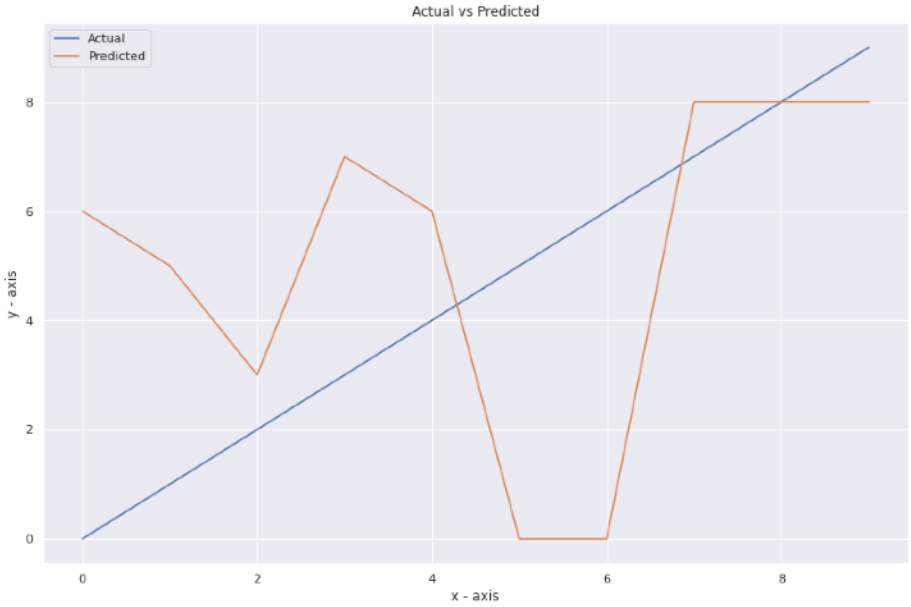

Fig 41: Line Chart of the Actual vs Predicted Values

And, lastly, we have plotted a multi-Class confusion matrix of the Actual and Predicted values (as our target variable is multi-Class variate) and have displayed it below.

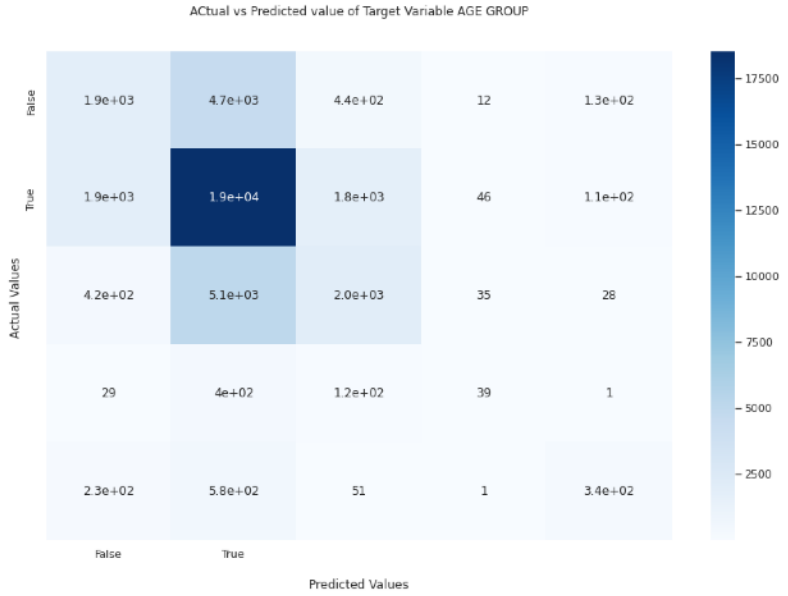

Fig 42: Multi-class confusion matrix of Actual vs Predicted Values

Impact of Project Outcome:

As the data being available to us as a data miner, we can finally get into the depths of the crimes happening around the society. We have performed a sort of a profiling prediction of the age group of the people involved in the crime & likely to commit the crime by understanding the pattern or occurrence of one or two age groups.

For the investigation process, law enforcement officials and investigators collect raw data from various sources such as telephone records, social networks, police records, and transaction records.

This is a time-consuming task, but new technologies and tools can assist law enforcement in efficiently identifying criminals.

This project examined various data mining tools and techniques for identifying specific crime patterns in a specific location at a specific time.

According to the reports, crime data mining has the potential to improve national security as well as intelligence productivity and efficiency.

There are numerous future pieces of research on it that are still in the exploratory stage.

We have just caressed the surface of the prediction capability of the datamining algorithms which we can incur & utilize in other real-world application & domains.

References:

1. Jabeen, Nahid, and Parul Agarwal. "Data Mining in Crime Analysis." *SpringerLink*, Springer Singapore, 1 Jan. 1970, https://link.springer.com/chapter/10.1007/978-981-15-6707-0_10.