Name/ Mostafa Mohamed Ismail Ahmed

Email/ mostamohamed22@gmail.com

Project Name/ San Francisco Crime Classification

Project Overview

The use of machine learning and artificial intelligence for detection and prevention of crimes has increased dramatically over the past few decades. Law enforcement agencies have access to large volumes of crime data stretching back decades, and they are looking for ways that this data can be leveraged for prediction of crime patterns and types.

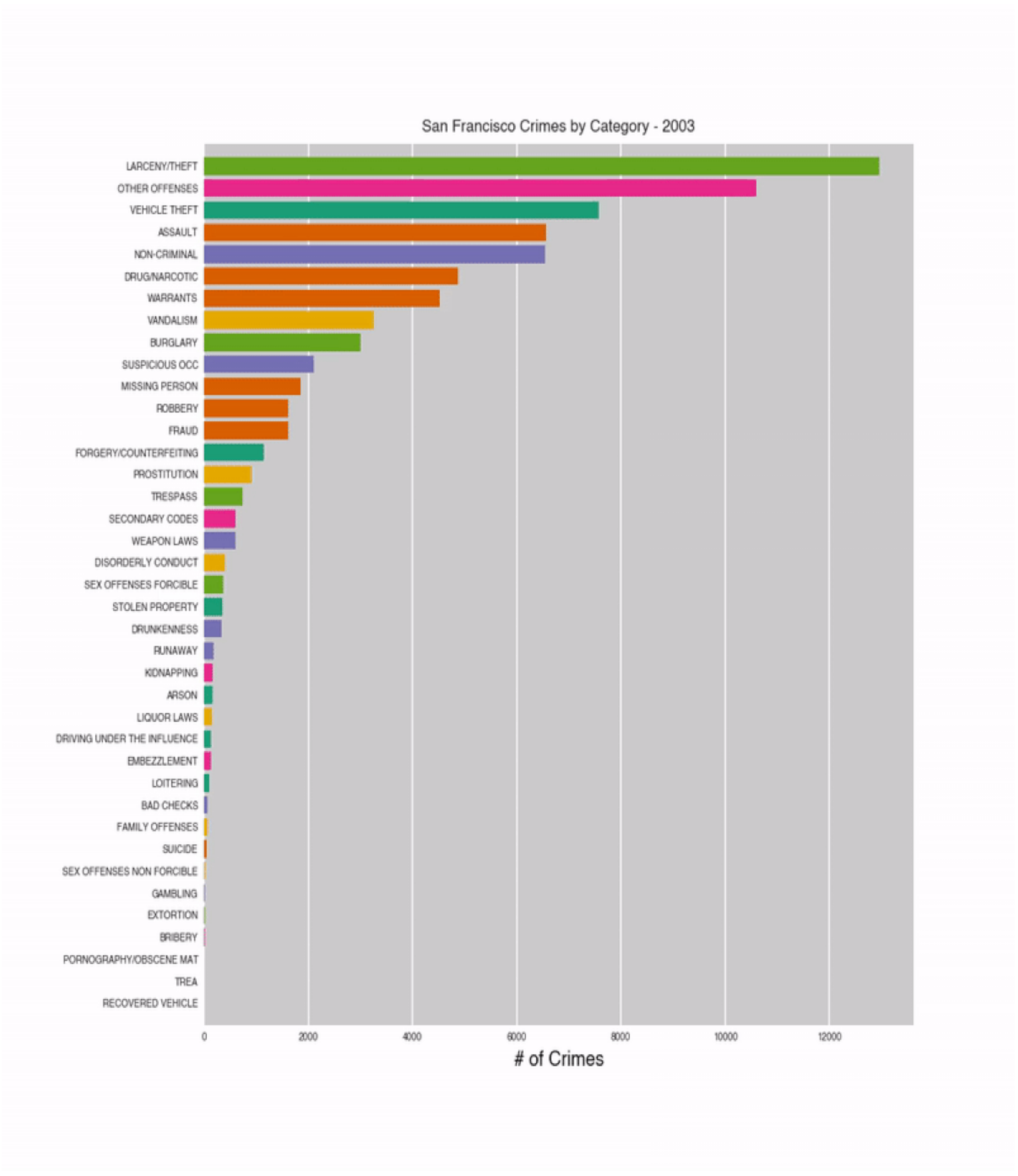
This project focuses on the use of historical crime data in San Francisco to predict the category of a crime event given only information about the event's time and location. The dataset, provided by SF OpenData, includes nearly 12 years of crime reports from the San Francisco metropolitan area collected between 2003 and 2015 .

Prediction Approach

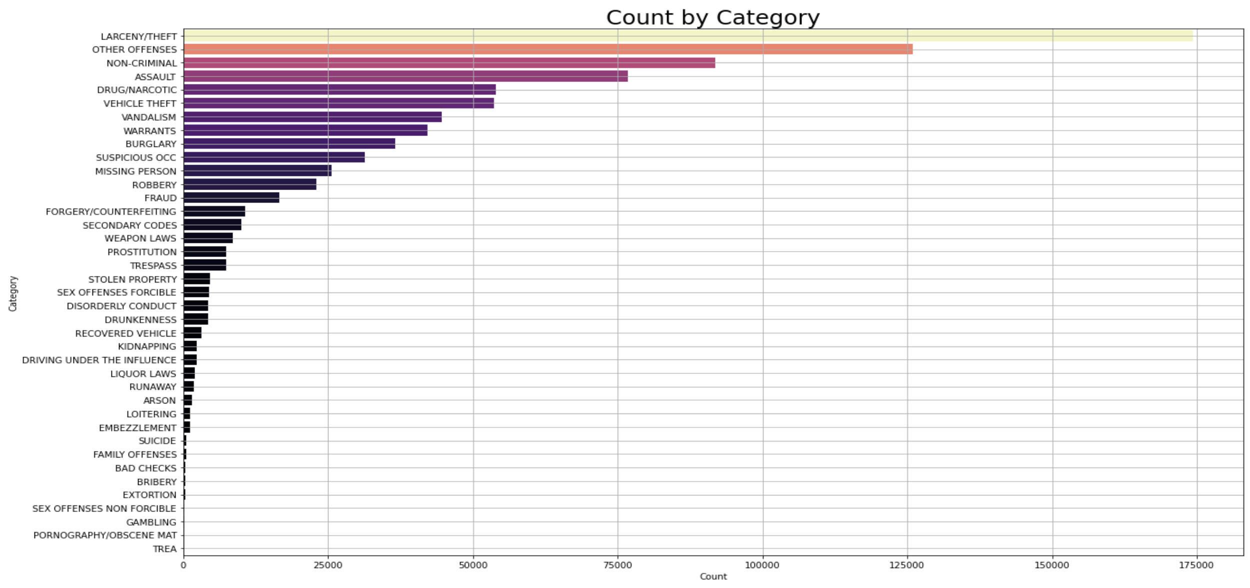
Our goal is to develop a model that correctly identifies the Category of each crime in the test dataset. This type of problem is considered a multiclass classification problem. In this instance, a total of 39 classes of crime category are represented in the training data.

Our model should assign a predicted probability to each of the 39 possible classes of crime category for each test observation, rather than simply predicting a single class. Selecting a single class is the equivalent of assigning a probability of 1 to that class and a probability of 0 to all other classes. If this approach is taken and the selected class is incorrect, the evaluation metric will assign the maximum penalty per observation. Using class probabilities as opposed to selecting a single class decreases the penalty we receive from the evaluation metric in the event that our prediction is incorrect.

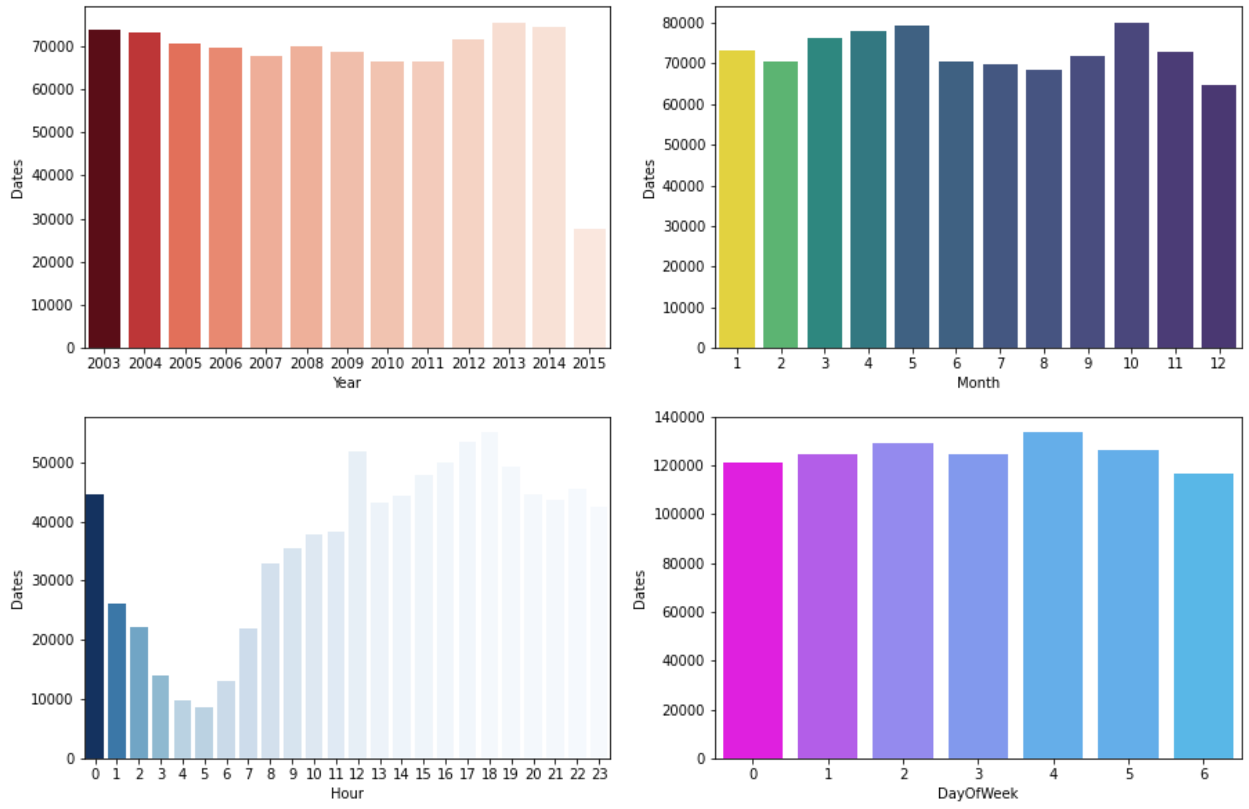
Main charts/ plots explained



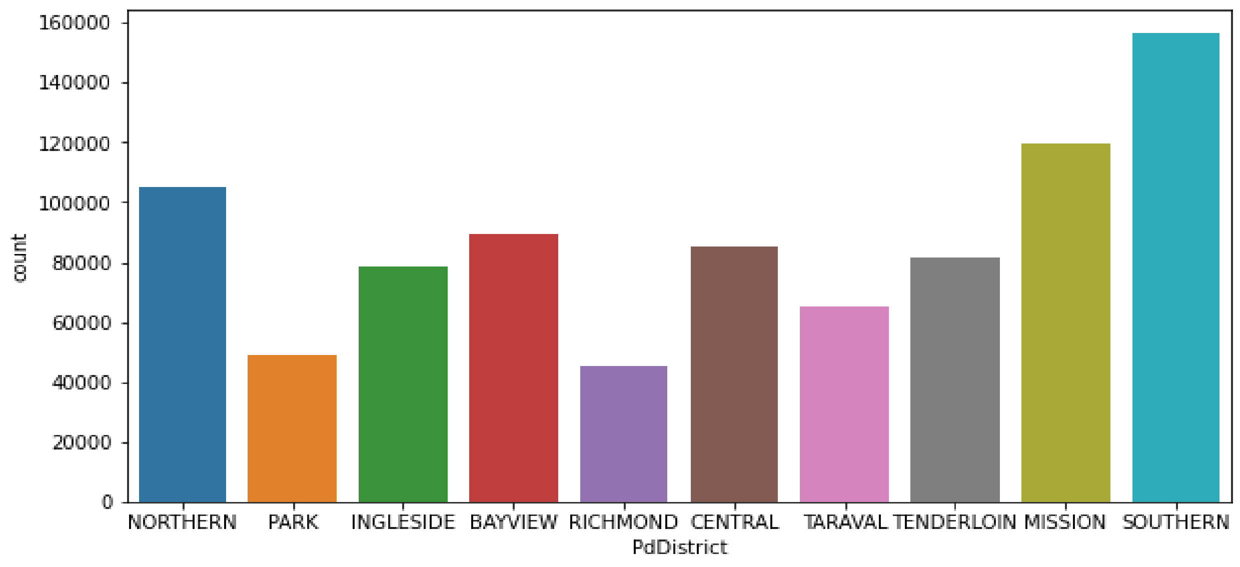
1) There are 39 discrete categories that the police department file the incidents with the most common being Larceny/Theft, Then Non/Criminal ,Then Assault.



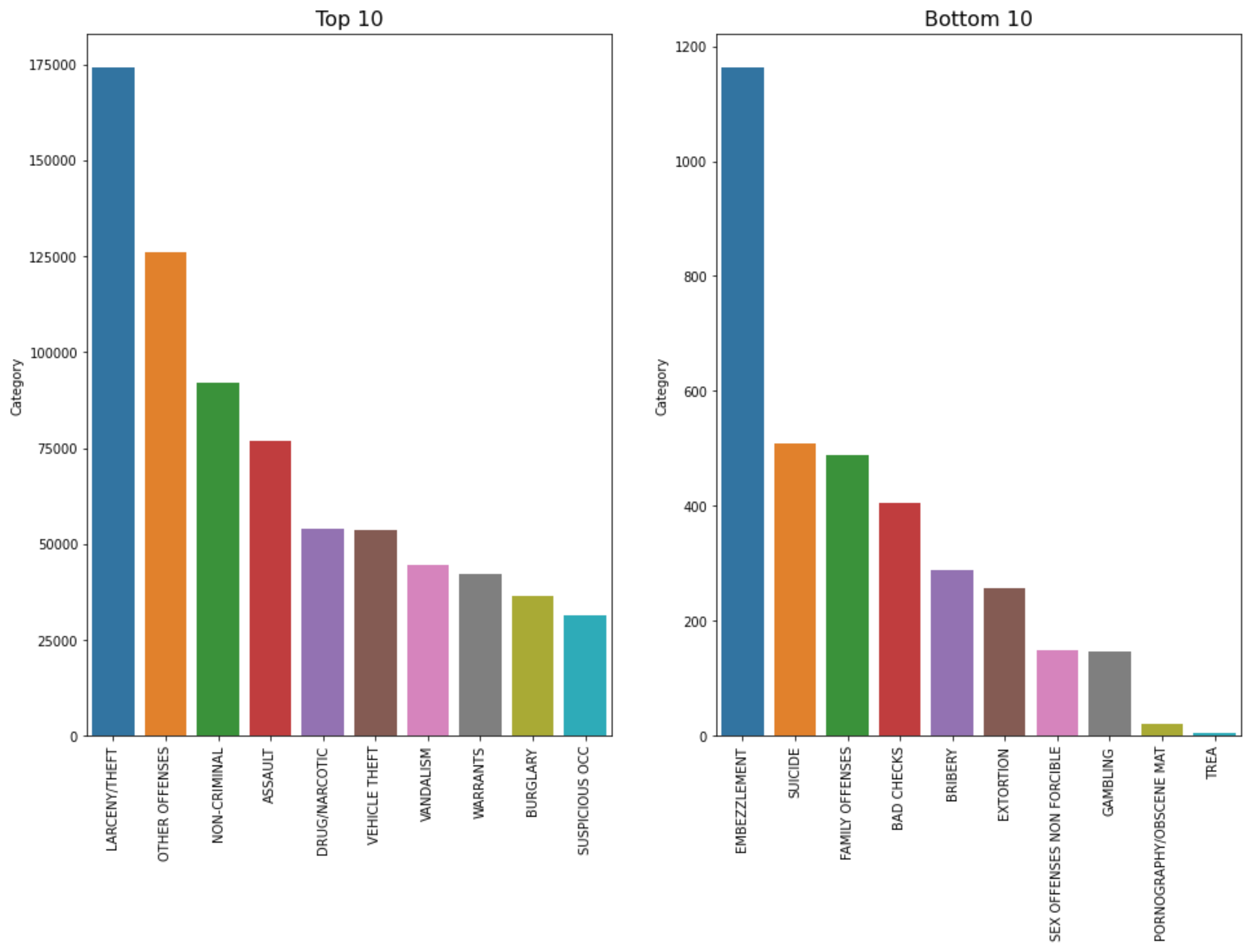
2) the amount of crime per years, day of the week, months, hours are plotted using a bar chart



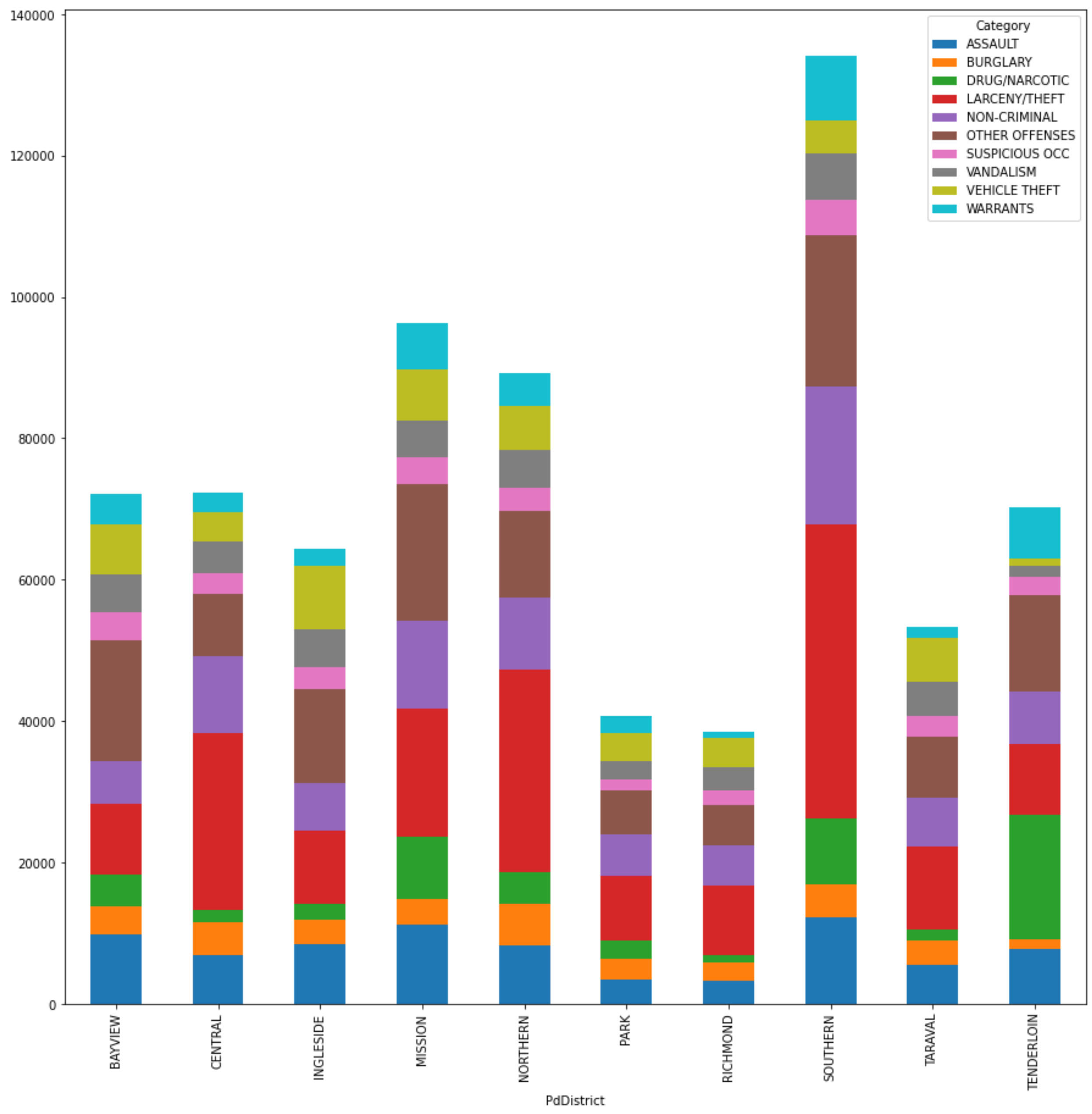
3) District Wise Crime Count ,the most common being Southern, Then Mission ,Then Northern



4) Top 10 and the least 10 occurring crimes



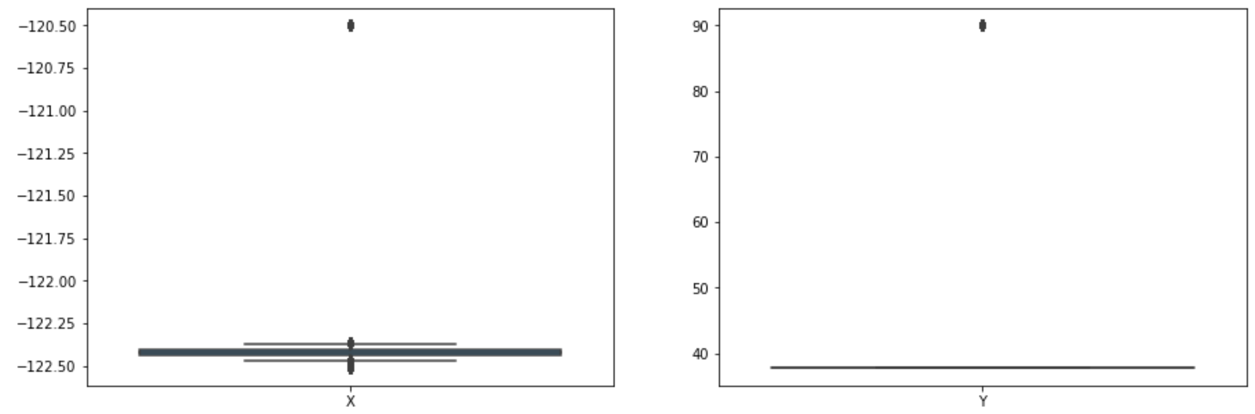
5) crimes count in the districts



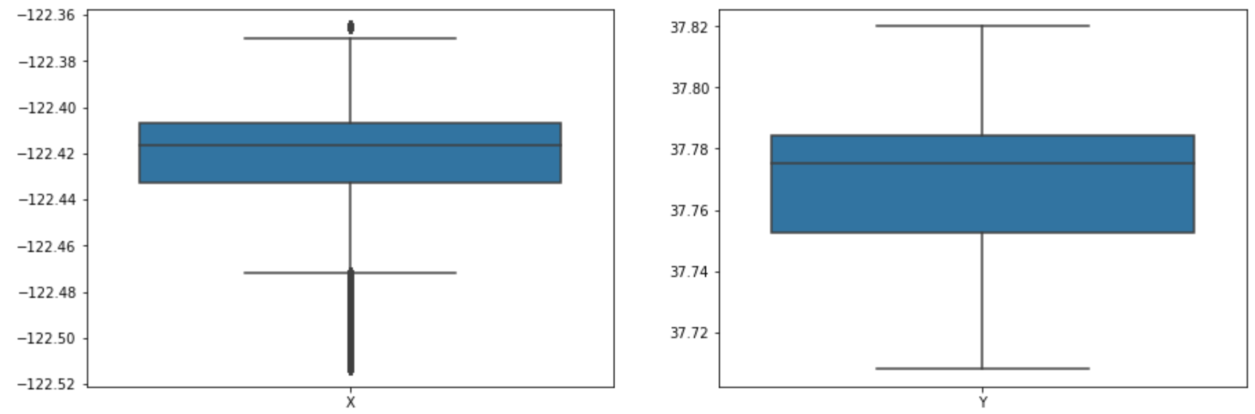
Data cleansing and features engineering steps

1) We will delete any duplicate values

2) Checking For Outliers
   We can see there are outliers in X at -120.5 and Y at 90.0 so we will remove them. We will replace the outlying coordinates with the average coordinates of the district they belong.



   Box Plot after removing outliers:



3) Data Wrangling:
   Following the methodology described in the Problem Statement, we identified 2323 duplicate values and 67 wrong latitudes. The duplicates removed and the outliers imputed.

4) Feature Engineering:
Using the Dates feature, the following temporal features were generated:

Year Month Day Hour Minute

Then, we created additional features. More specifically:

- From the ‘Dates’ field, we extracted the Day, the Month, the Year, the Hour, the Minute, the Weekday, and the number of days since the first day in the data.
- From the ‘Address’ field we extracted if the incident has taken place in a crossroad or on a building block.
- From the ‘ X and Y’ field we calculated "X-Y" And "X+Y"

Feature Selection:

- After the feature engineering described above, we ended up with 11 features. To identify if any of them increased the complexity of the model without adding significant gain to the model, we used the method of Permutation Importance.
- The idea is that the importance of a feature can be measured by looking at how much the loss decreases when a feature is not available. To do that we can remove each feature from the dataset, re-train the estimator and check the impact. Doing this would require re-training an estimator for each feature, which can be computationally intensive. Instead, we can replace it with noise by shuffle values for a feature.
- The implementation of the above technique showed that there is no need for any feature removal since all of them have a positive impact in the dataset.

5) Encoding:
Following feature generation, our dataset contained a mixture of numerical and categorical features. Numerical features are features that are represented by continuous numerical values, while categorical features are represented by a discrete number of categories, or classes. For example, a feature such as DayOfWeek has a discrete number of possible values (Sunday — Saturday).

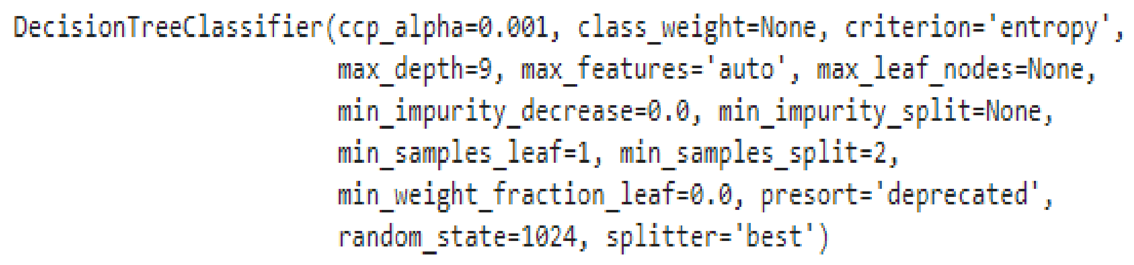
Encode target labels with value between 0 and n_classes-1
I have encoded:

- PdDistrict in train and test data
- Category in train data
- DayOfWeek in train and test data

Results of the models

I Have Used GridSearchCV :
GridSearchCV is a library function that is a member of sklearn's model selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters

1) **DecisionTreeClassifier**
After Using GridSearchCV I found the best estimator is

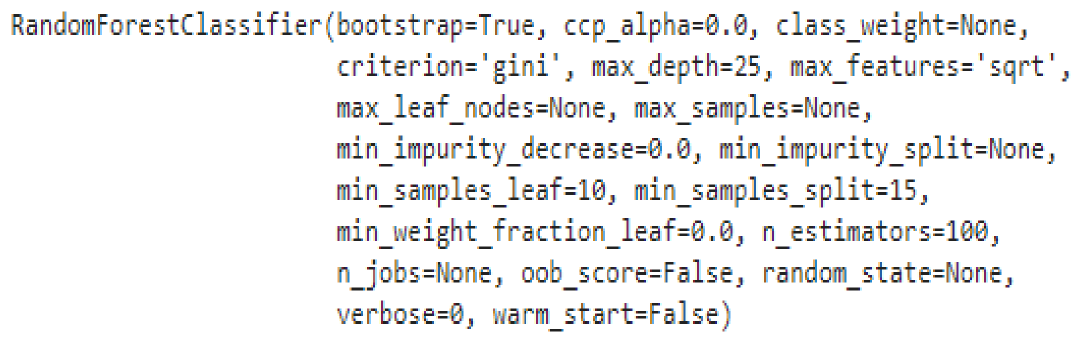


```
DecisionTreeClassifier(ccp_alpha=0.001, class_weight=None, criterion='entropy',
                       max_depth=9, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1024, splitter='best')
```

After fit (x_train , y_train)

- accuracy score = 0.25364553001495893
- log loss = 2.4835476801927263
- Time of Training= 2.57232928276062 **RAM:** 11154.40234375
- Time of predict = 0.0453183650970459 **RAM:** 11154.40234375

2) **RandomForestClassifier**
After Using GridSearchCV I found the best estimator is



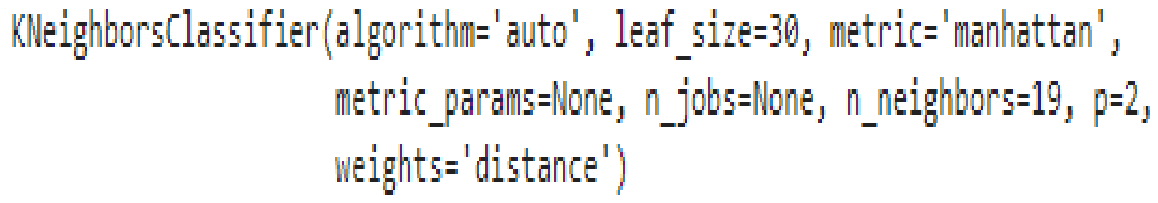
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=25, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=15,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

After fit (x_train , y_train)

- accuracy score = 0.3364792801434232
- log loss = 2.4909673078223573
- Time of Training= 231.55238270759583 **RAM:** 11157.578125
- Time of predict = 12.805527925491333 **RAM:** 11154.21484375

3) KNeighborsClassifier

After Using GridSearchCV I found the best estimator is



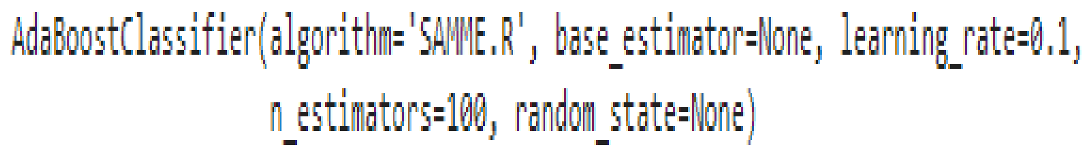
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                     weights='distance')
```

After fit (x_train , y_train)

- accuracy score = 0.2528747444988752
- log loss = 13.223281730442988
- Time of Training= 3.6131503582000732 **RAM:** 10725.4921875
- Time of predict = 12.940115690231323 **RAM:** 10725.49609375

4) AdaBoostClassifier

After Using GridSearchCV I found the best estimator is



```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=0.1,
                   n_estimators=100, random_state=None)
```

After fit (x_train , y_train)

- accuracy score = 0.23458143491715483
- log loss = 3.3754092291392994
- Time of Training= 196.20266962051392 **RAM:** 10725.4609375
- Time of predict= 24.12448000907898 **RAM:** 10725.45703125

The best model is **RandomForestClassifier** Has the most accuracy and good time in training and predict time and good use of Ram

Future work that may be made to enhance the models

1) **Looking for more data:**
    - you may still have a high variance of predictions to deal with. In this case, your only option is to increase your training set size. Try increasing your sample by providing new data, which could translate into new cases or new features.

2) **Stacking models:**
    - For the same reasons that averaging works, stacking can also provide you with better performance. In stacking, you build your machine learning models in two stages. Initially this technique predicts multiple results using different algorithms, with all of them learning from the features present in your data. During the second phase, instead of providing features that a new model will learn, you provide that model with the predictions of the other, previously trained models.

    - Using a two-stage approach is justified when guessing complex target functions. You can approximate them only by using multiple models together and then by combining the result of the multiplication in a smart way. You can use a simple logistic regression or a complex tree ensemble as a second-stage model.

3) **Averaging models:**
    - Machine learning involves building many models and creating many different predictions, all with different expected error performances. It may surprise you to know that you can get even better results by averaging the models together. The principle is quite simple: Estimate variance is random, so by averaging many different models, you can enhance the signal and rule out the noise that will often cancel itself.

4) **Testing multiple models:**
    - As a good practice, test multiple models, starting with the basic ones the models that have more bias than variance. You should always favor simple solutions over complex ones. You may discover that a simple solution performs better.

    - Representing the performance of different models using the same chart is helpful before choosing the best one to solve your problem.

    - Testing multiple models and introspecting them can also provide suggestions as to which features to transform for feature creation, or which feature to leave out when you make feature selections.

5) **Choosing the right error or score metric**

- Trying to optimize an error metric based on the median error by using a learning algorithm based on the mean error won't provide you with the best results unless you manage the optimization process in a fashion that works in favor of your chosen metric. When solving a problem using data and machine learning, you need to analyze the problem and determine the ideal metric to optimize.

6) **Ensemble methods:**

- This is the most common approach found majorly in winning solutions of Data science competitions. This technique simply combines the result of multiple weak models and produce better results. This can be achieved through many ways:

  - ♦ Bagging (Bootstrap Aggregating)
  - ♦ Boosting