CRIME DATA ANALYSIS BY EFFECTIVE PREDICTION AND CLASSIFICATION

Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements
for the award of the degree of

BCSCCS801: PROJECT WORK

Submitted by

SMRUTHI MOHANKUMAR
(Reg. No.: 121003266,B.Tech Computer Science)
MANASA K
(Reg. No.: 121003165, B.Tech Computer Science)
PARVATHI N
(Reg. No.:121003196, B.Tech Computer Science)

June 2021

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

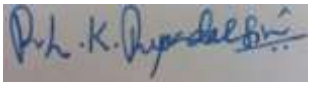
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the project report titled **"Crime Data Analysis by Effective Prediction and Classification"** submitted in partial fulfillment of the requirements for the award of the degree of B. Tech. Computer Science and Engineering to the SASTRA Deemed to be University, is a bona-fide record of the work done by **Ms.Parvathi N(Reg. No. 121003196), Ms.Manasa K(Reg. No. 121003165) and Ms.Smruthi Mohankumar(Reg. No. 121003266)** during the final semester of the academic year 2020-21, in the **School of Computing**, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor: 

Name with Affiliation : Dr. PLK Priyadarsini, Department of IT, SASTRA Deemed University

Date : 31/05/2021

Project *Vivavoce* held on _____

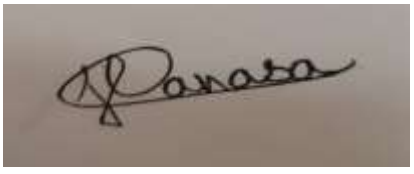
Examiner 1 **Examiner 2**

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Declaration

We declare that the project report titled **"Crime Data Analysis by Effective Prediction and Classification"** submitted by us is an original work done by us under the guidance of **Dr. P.L.K.Priyadarsini, Senior Assistant Professor, School of Computing, SASTRA Deemed to be University** during the final semester of the academic year 2020-21, in the **School of Computing**. The work is original and wherever we have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s) **:** 

Name of the candidate(s) **:** Manasa K

Date **:** 31/05/2021

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology and **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering for their motivation and support offered in materializing this project.

Our guide **Dr. P.L.K. Priyadarsini**, Senior Assistant Professor, School of Computing was the driving force behind this whole idea from the beginning. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from our families and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through this project.

Table of Contents

List of Figures

List of Tables

Abstract

The influential field of Data Science has the potential to not only solve recurring, present real-world problems but also identify and handle the foreseeable problems in the future with ease. With different pattern recognition, visualization, prediction and classification techniques in the domain of Machine Learning, in this project, we apply data analytics on criminal database of cities in the United States of America only to find interesting facts and patterns. We employ state-of-the-art prediction algorithms like LSTM, Neural Networks and Prophet Model to derive at the number of crimes happening on a day-to-day basis followed by classification algorithms like GaussianNB, K-Nearest Neighbors, Random Forest Regressor and Artificial Neural Networks to differentiate between the crimes that happen. The results are visualized in a human-friendly way so that one can identify and track illicit activities. These outcomes provide assurance for preventing crimes in the future, ultimately, benefiting the police department and detectives.

Specific Contribution

- Visualisation mechanisms including interactive google maps
- Pre-processing mechanisms for classification
- Handling multi-class classification algorithms
- Finding an efficient way of classification with an optimal run time, space and accuracy

Specific Learning

- Databricks Community Edition for executing Machine Learning Models
- Handling a large quantity of records
- Unique characteristics of the classification algorithms used and their hyper parameters
- Different visualization techniques

CHAPTER 1

INTRODUCTION

1.1 Data Analytics

Data Analytics (DA), in recent years, has emerged as a popular method for analysing data and extracting knowledge and relationships in a variety of fields. To analyse this multi- sourced and heterogeneous data, new methods and technologies must be developed. We can easily keep track of incidents, identify correlations between events, make use of resources, and take fast decisions based on this information. The exponential development of cloud computation and data collection and storage technology has resulted in a massive increase in the variety and complexity of data that has been obtained and made publicly accessible. Data mining is a creative, growing and an interdisciplinary research area that can create techniques and paradigms across numerous fields for extracting useful knowledge and hidden trends from data. It is one of the central approaches of Data Science. Data mining can be used to uncover new knowledge or phenomena as well as improve our understanding of existing ones.

1.2 Crime Investigation

One of the main concerns of residents of any large city is their safety. It is extremely difficult for the police department to keep track of incidents and reduce the crime rate. It is impossible to get a bird's eye view of the whole city at all times. It is now possible to predict such features after studying historical data using different machine learning techniques, which is one of the data analytics approaches. The implementation of machine learning methods and statistical mechanisms on crime data analysis or other big data applications such as road accidents or time series data would allow for the study, extraction, and interpretation of related trends and patterns, utilized in crime control and prevention.

1.3 Motivation

The media seems to be continually inundating us with information about crime rates around the world. It's sickening to see so many places that are beautiful to live in but compromise on protection. This was particularly noticeable in cities such as San Francisco, Chicago, and Philadelphia. With this understanding of machine learning, the City Police Department would benefit greatly if they were aware of crime trends in the near future. With a secure database in hand, this analysis can be done for a number of other cities.

CHAPTER 2

OBJECTIVES

2.1 General Objective

To analyse the criminal incidents of three cities namely San Francisco, Chicago and Philadelphia, identify potential crime patterns and predict future crime happenings with respect to the attributes given in the datasets for those three cities.

2.2 Specific Objectives

- To visualise different perceptions of the dataset in the form of pie charts, bar charts, donut charts and time series plots and gather informative insights.
- To predict the number of crime incidents that may happen on a daily basis for the upcoming years which will potentially help the police department of the respective cities.
- To multi-classify the crimes based on the day of occurrence and the district number so that the detectives would have an idea of what type of crime might happen given a date.
- To compare the various prediction and classification algorithms used here.

CHAPTER 3

EXPERIMENTAL WORK/METHEDOLOGY

3.1 Dataset

The dataset is collected from the City's Government website for all the three cities. It has a record of crime incidents from 2003 till date. The dataset has attributes viz Case number of the incident (IncidentNum), Timestamp and Date of the incident, the Category of crimes, Description of the crime scene, Day of the week, Police Department District ID (PdDistrict), how the crime was resolved (Resolution), Address, Latitude and Longitude.The links for the dataset are:

- San Francisco-https://data.sfgov.org/ Public-Safety/Police-Department-Incident-Reports-Historical-2003/tmnf-yvry
- Chicago - https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present
- Philadelphia-https://www. opendataphilly.org/dataset/crime-incidents

3.2 Preprocessing

Before implementation, the dataset needs to be cleaned. The following pre-processing steps were carried out after importing the dataset:

- Time discretization to represent time series forecasting of overall trend (i.e) Division of the dates into Year, Month, Day and time into Hour, Minute and Second.
- NaN values were removed.
- Unwanted features like IndicentNum were omitted.

3.3 Narrative Visualization

3.3.1 Interactive Google Map-based Clustering

An interactive map based on Google Maps was visualised to get a spatial view of the cities, with crime events clustered based on latitude and longitude points. The marker cluster algorithm assisted in clustering the coordinates at various zooming speeds. Person markers are visible on the chart at high zoom levels, allowing you to pinpoint the exact position of the incident. By using this interactive google map, users can take a look at all the crimes on unique dates, time, streets and towns.

3.3.2 Time Series Plot

Crime cases were categorised by year of occurrence, resulting in a year-by-year count of the crimes. For all three towns, this value was plotted using the Plotly library.

3.3.3 Top-10 Crime Cases (Pie chart)

Despite the fact that there are more than 30 crime categories, the distribution of these categories seems to be distorted. As a result, for each of the three towns, the top ten crime scenes were extracted and a pie chart was plotted using the Plotly library.

3.3.4 Hourly Trend of Crimes

For all three towns, the crime count was clustered hourly and plotted using the Plotly library, hovering the count. As a result, one can deduce how crime has changed hour by hour and which crimes have occurred during peak hours.

3.3.5 Wordcloud of Crime Description

Another powerful visualisation technique is wordcloud, which highlights terms or phrases based on their frequency. As a result, crime descriptors were drawn as a Wordcloud for all three cities in order to gain insight into the most commonly occurring crime category. The most frequently occurring descriptors have a big font, while the least frequently occurring descriptors have a small font.

3.3.6 Box Plot for Monthly Trend

The box plot is one of the most powerful visualisation techniques for determining the distribution of values for a given set of values, as well as the mean, median, and interquartile range. For each of the three towns, a monthly crime count is shown in a box map.

3.3.7 Top-10 Dates with the Most and Least Crimes

Donut Chart (also recognized as Doughnut Chart) is a Pie chart with a round hole in the middle, giving it the appearance of a donut, hence the name. Additional data can be shown in the empty space in the middle. Summarisation of top 10 dates with the most and least number of crimes has been represented by a dount chart for all the three cities.

3.4 Prediction Models

3.4.1 Prophet Model

The Prophet model is an additive model that fits non-linear patterns with annual, weekly, and/or regular seasonality, as well as holiday impacts, to forecast time series analysis. It is best for time series with heavy seasonal effects and several periods of historical data. Missing data and pattern changes are not a problem for the Prophet model, and outliers are usually handled well. This model is designed to properly handle time series features while still having intuitive parameters that can be modified without understanding the underlying model's information. The seasonal function uses the Fourier series defined below to achieve this, where P denotes the regular period:

$$s(t) = \sum_{1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right)$$

3.4.2 Neural Network Model

A neural network is made up of a number of neurons, or nodes in the network, that are arranged in multiple layers and linked to each other using different types of connections. layering Here, a multi-layer feed-forward network is used. The input above will be modified by the hidden layer using a nonlinear function.

$$s(z) = \frac{1}{1 + e^{-z}}$$

3.4.3 LSTM Model

The LSTM model is a kind of Recurrent Neural Network (RNN) that can learn long-term dependencies. Due to their ability to preserve the state when identifying trends over time series, LSTMs are particularly useful in prediction for time series involving auto-correlation, i.e., the existence of correlation between the lagged versions of itself and the time series. When each epoch progresses, the recurrent architecture allows the states to be continued or communicated between modified weights. Furthermore, the LSTM cell architecture can improve the RNN by allowing long-term as well as short-term persistence.

3.5 Classification Models

Multi-class classification is the concept of using more than one class to classify instances. In the case of splitting instances into two classes, it is called as binary classification. However, this is very different from multi-label classification where prediction of instances is in the form of lables.

3.5.1 k- Nearest Neighbours Model

This is a non-parametric classification algorithm. The distance from an example to every other example in the training set is measured. ‘k’ examples with the smallest distances are taken and these are called the “k- Nearest Neighbours”. The most represented classes are taken as the output.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.5.2 Gaussian Naïve Bayes

When Gaussian normal distribution is followed and it supports continuous data in addition to Naïve Bayes algorithm, it is called Gaussian Naïve Bayes. As the name suggests, it is based on the Bayes theorem and has high functionality.

P

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

3.5.3 Random Forest

Random decision forest is used both classification and regression to predict the data. They generate reasonable predictions across a wide range of data while requiring little configuration. Random forest uses decision trees to predict results.

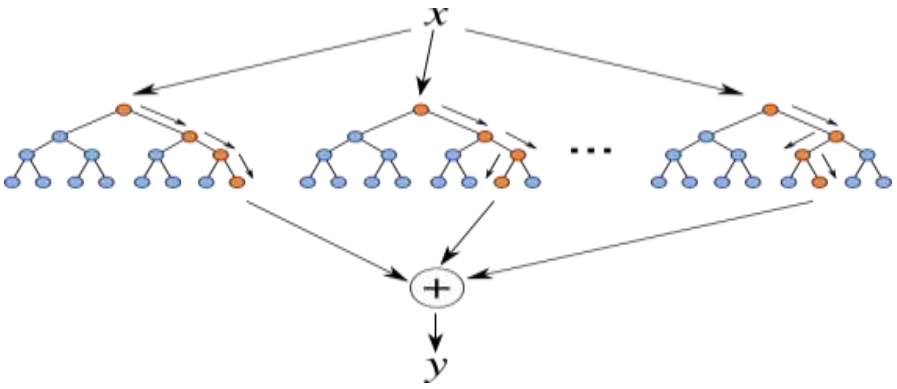


Fig 3.5.3 Random Forest

3.5.4 Artificial Neural Network

Artificial neurons are at the heart of ANNs. Each neuron takes input from a number of other neurons, multiplies it by weights assigned to it, adds it up, and sends the result to one or more neurons. Before transferring the output to the next variable, certain artificial neurons may apply an activation function to it. Furthermore, there is no central CPU managing the logic, regardless of the underlying hardware. Rather, the logic is spread out among thousands of smaller artificial neurons.

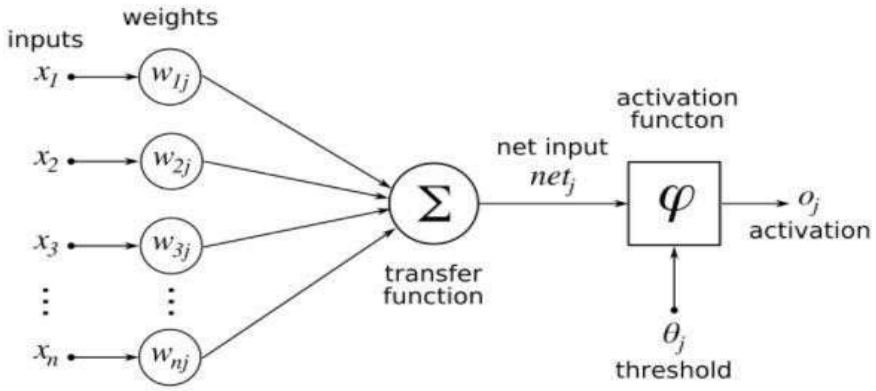


Fig 3.5.4 Artificial Neural Networks

3.6 Comparison metrics

To forecast crime trends, we looked at time series forecast and deep learning models. In terms of varying arguments and different amounts of training samples, the Root Mean Square Error (RMSE), Spearman correlation, and Mean Absolute Error (MAE) are employed for performance evaluation.

3.6.1 Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) or the Root Mean Square Deviation (RMSD) is a commonly used metric for comparing predicted and observed values (sample or population values) by a model or estimator. The square root of the second sample moment of the discrepancies between anticipated and observed values, or the quadratic mean of these differences, is represented by the RMSD.

$$\text{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

RMSD = Root Mean Square Deviation

i = variable i

N = Number of non-missing data points

x_i = Actual observations time series

$\hat{x}_i$= Estimated time series

3.6.2 Spearman Correlation

Spearman's rank correlation coefficient, sometimes known as Spearman's, is a nonparametric rank correlation measure (statistical dependence between the rankings of two variables). It determines how well a monotonic function can describe the relationship between two variables. When findings have a similar (or identical for a correlation of 1) rank between the two variables, the Spearman correlation will be high, and when observations have a dissimilar (or fully opposed for a correlation of 1) rank between the two variables, the Spearman correlation will be low.

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

cov is the covariance; σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y.

3.6.3 Mean Absolute Error

A measure of mistakes between paired observations describing the same phenomenon is called mean absolute error (MAE). Comparisons of predicted against observed, subsequent time versus starting time, and one measuring technique versus another measurement technique are examples of Y against X.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

As a result, it is an arithmetic average of the absolute errors $|e_i|=|y_i-x_i|$, where y_i is the prediction and x_i is the true value.

We use MAE for evaluating our classification model's accuracy since it allows us to differentiate between forecasts of varied time series in varied scaling. It is less susceptible to outliers than MSE because it does not penalise large errors. It produces a linear value that equalises the weighted individual disparities. The model's performance improves as the value decreases.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Experimental Results of Philadelphia

In Figure 4.1.1, using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. Fig 4.1.2 is a time series plot wherein the number of crimes is plotted against the year. In Figure 4.1.3, a pie chart for the top 10 crimes is displayed. Theft is one of the most occurring crimes. In Figure 4.1.4, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. In Figure 4.1.5, a box plot between the month and number of crimes is plotted. In Figure 4.1.6, a doughnut plot of the top 10 crime occurring dates is depicted. 27-10-2020 and 28-10-2020 show significant results. Figure 4.1.7 is an hour-wise plot.

Fig 4.1.1 Marker Cluster - Philadelphia

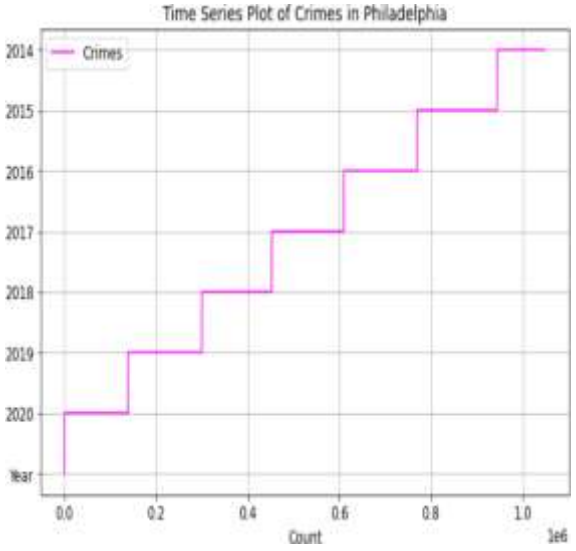


Fig 4.1.2 Time Series - Philadelphia

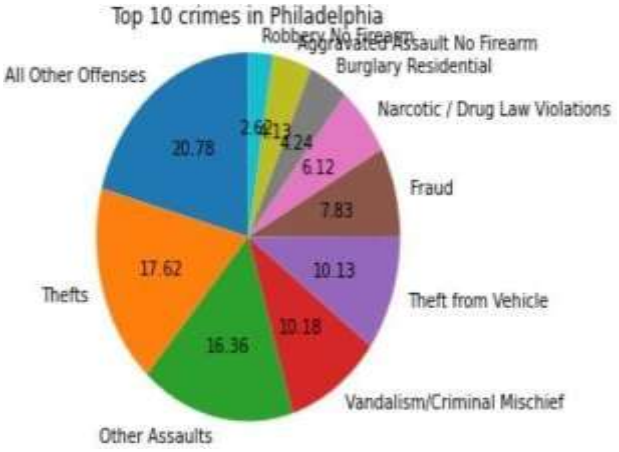


Fig 4.1.3 Pie Chart - Philadelphia

Fig 4.1.4 Word Cloud - Philadelphia

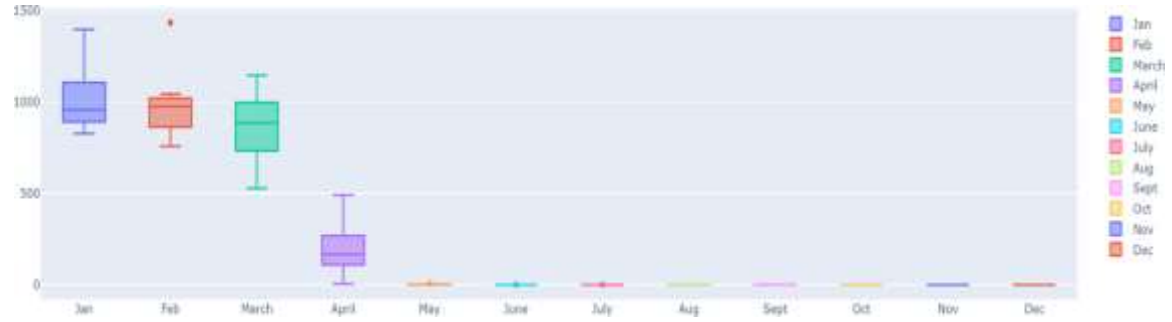


Fig 4.1.5 Box Plot - Philadelphia

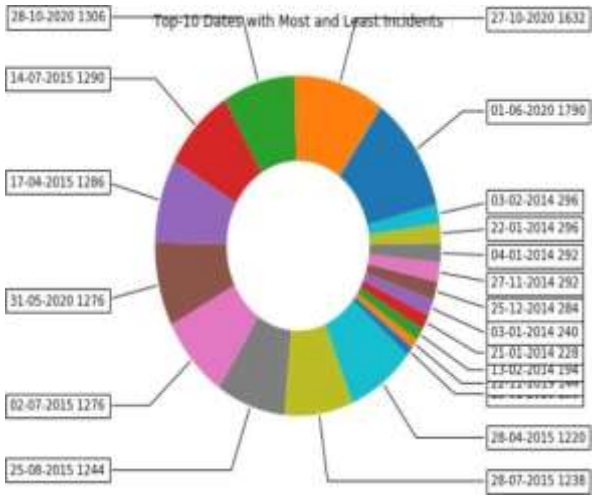


Fig 4.1.6 Doughnut Chart - Philadelphia

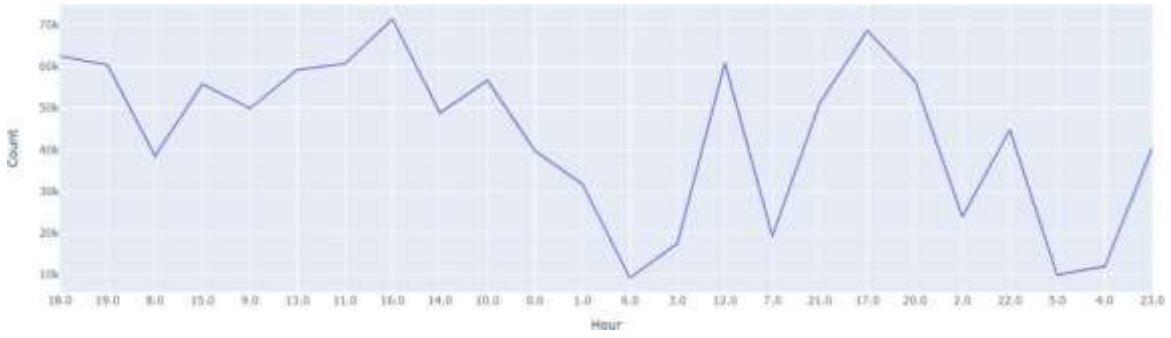


Fig 4.1.7 Hourly Plot - Philadelphia

4.2 Experimental Results of Chicago

In Figure 4.2.1, a pie chart for the top 10 crimes is displayed. In Figure 4.2.2, a box plot between the month and number of crimes is plotted. In Figure 4.2.3, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. In Figure 4.2.4, using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. Figure 4.2.5 is an hour-wise plot.

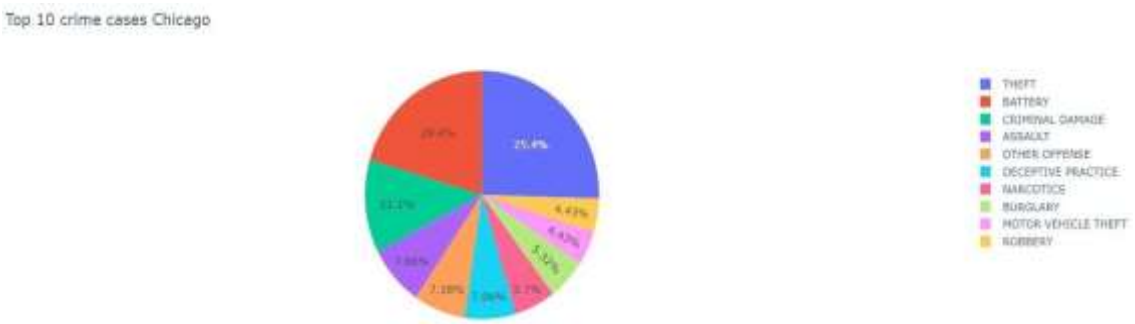


Fig 4.2.1 Pie Chart - Chicago

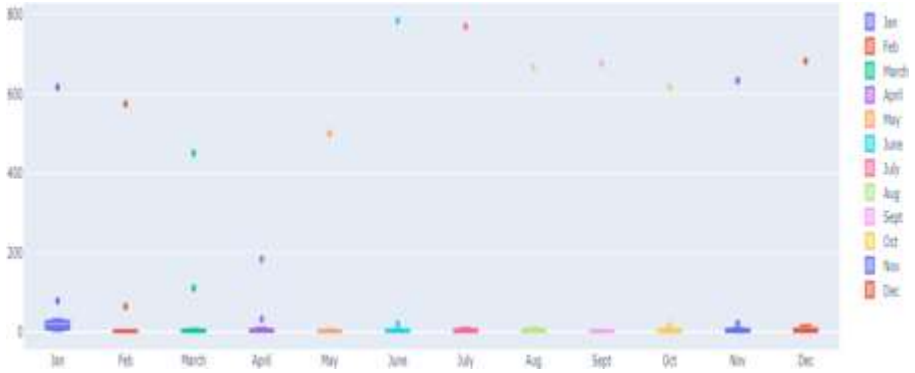


Fig 4.2.2 Box Plot - Chicago

Fig 4.2.3 Word Cloud - Chicago

Fig 4.2.4 Marker Cluster - Chicago

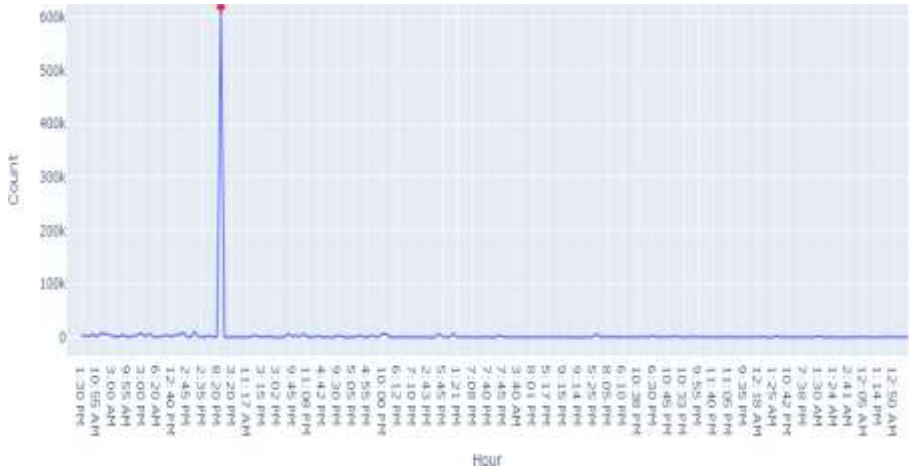


Fig 4.2.5 Hourly Plot - Chicago

4.3 Experimental Results of San Francisco

In Figure 4.3.1, using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. In Figure 4.3.2, a doughnut plot of the top 10 crime occurring dates is depicted. 1-1-2011 and 1-1-2013 show significant results. In Figure 4.3.3, a pie chart for the top 10 crimes is displayed. Larceny/Theft is one of the most occurring crimes. In Figure 4.3.4, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. Figure 4.3.5 is an hour-wise plot. In Figure 4.3.6, a box plot between the month and number of crimes is plotted.

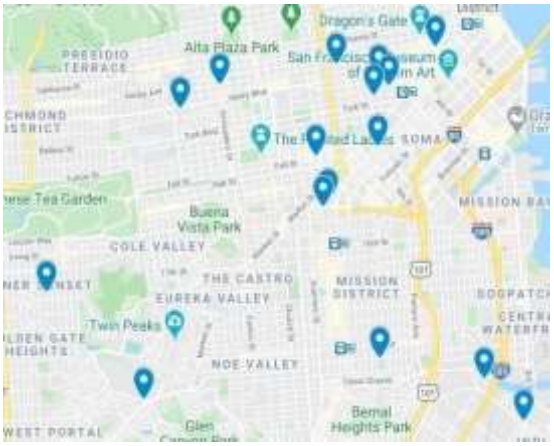


Fig 4.3.1 Marker Cluster – San Francisco

Fig 4.3.2 Doughnut Chart – San Francisco

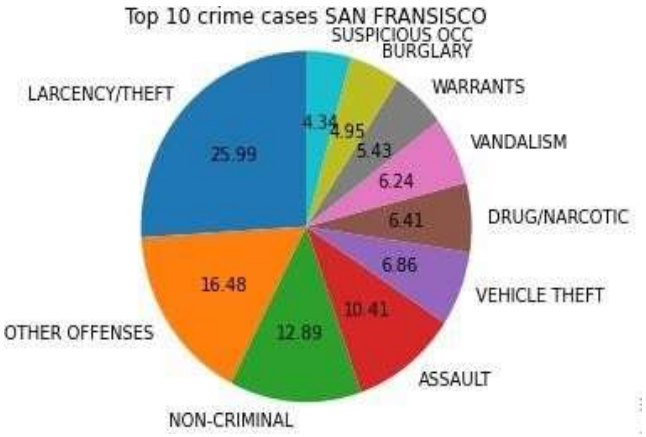


Fig 4.3.3 Pie Chart – San Francisco

Fig 4.3.4 Word Cloud – San Francisco

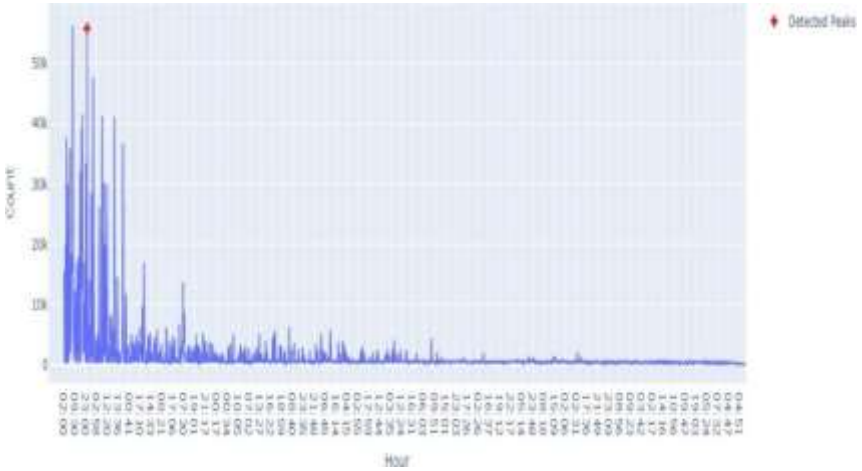


Fig 4.3.5 Hourly Plot – San Francisco

Fig 4.3.6 Box Plot – San Francisco

4.4 Prophet Model Results (San Francisco)

Figure 4.4.1 shows the overall trend of crimes. 2007-2008 is the predicted period. In Figure 4.4.2, we can see the components decomposition plot. Figure 4.4.3 is the trend decomposition plot. There is a decline in crime trend from 2007 to 2008.

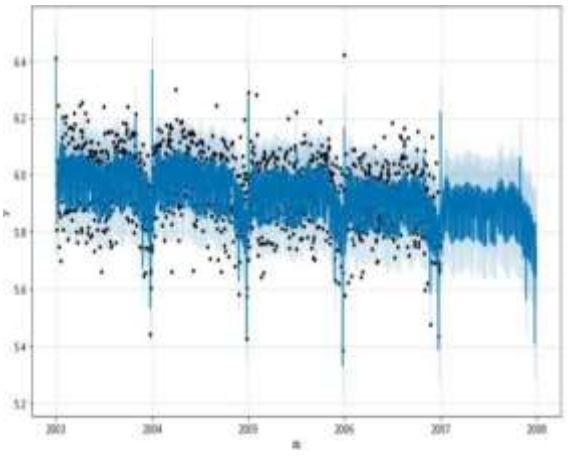


Fig 4.4.1 Overall Trend – San Francisco

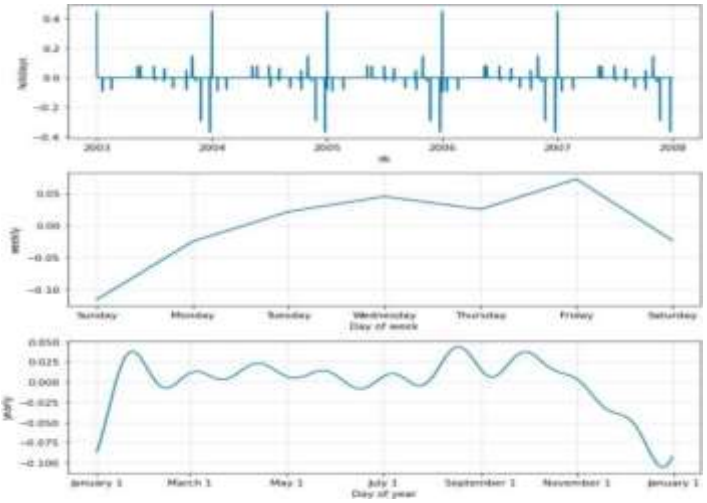


Fig 4.4.2 Component Decomposition Plot – San Francisco

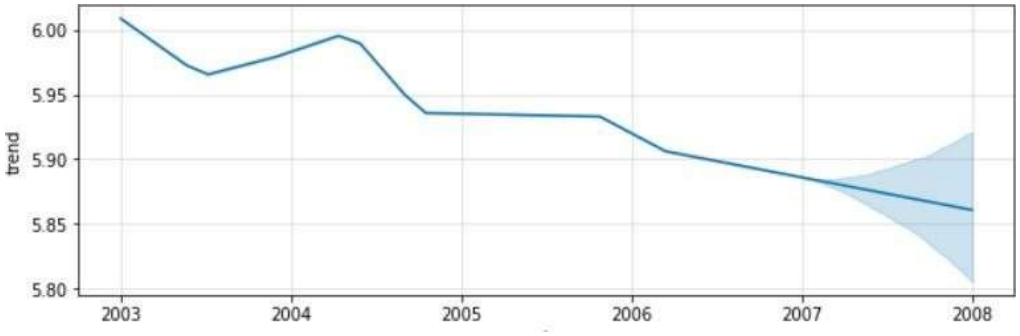


Fig 4.4.3 Trend Decomposition – San Francisco

4.5 Prophet Model Results (Philadelphia)

Figure 4.5.1 is the overall trend plot with 2012-2013 being the predicted period. Figure 4.5.2 is the component decomposition plot. Figure 4.5.3 shows the yearly trend.

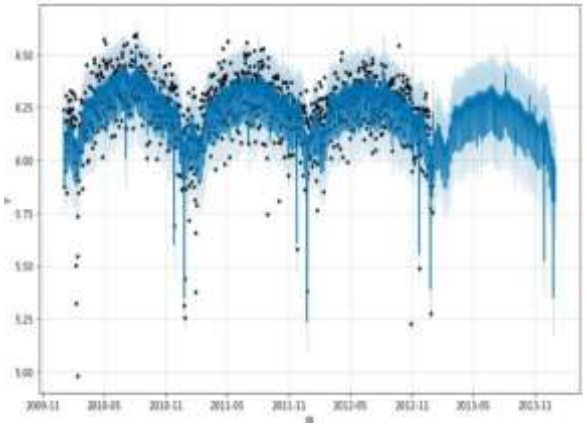


Fig 4.5.1 Overall Trend - Philadelphia

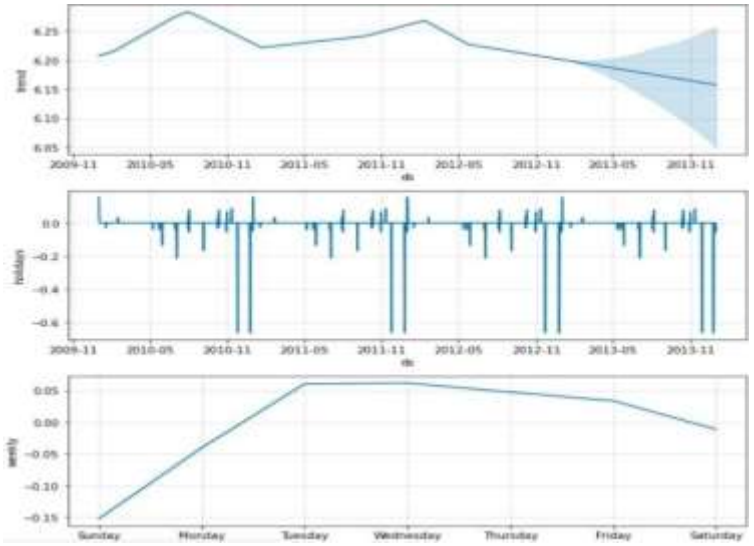


Fig 4.5.2 Component Decomposition Plot - Philadelphia

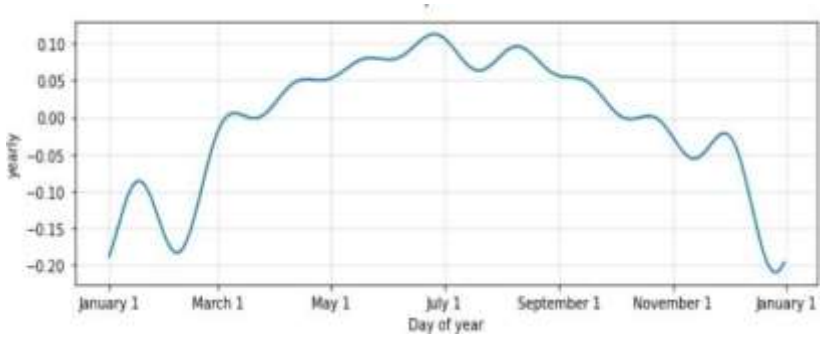


Fig 4.5.3 Yearly Decomposition

4.6 Prophet Model Results (Chicago)

Figure 4.6.1 is the overall trend plot with 2011-2012 being the predicted period. Figure 4.6.2 is the trend and holiday component decomposition plot and Figure 4.6.3 is the yearly and weekly component decomposition plot. There is a peak in the crime rate on Tuesday.

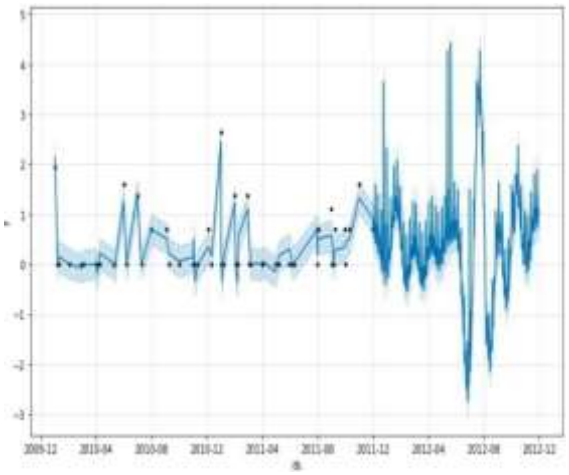


Fig 4.6.1 Overall Trend - Chicago

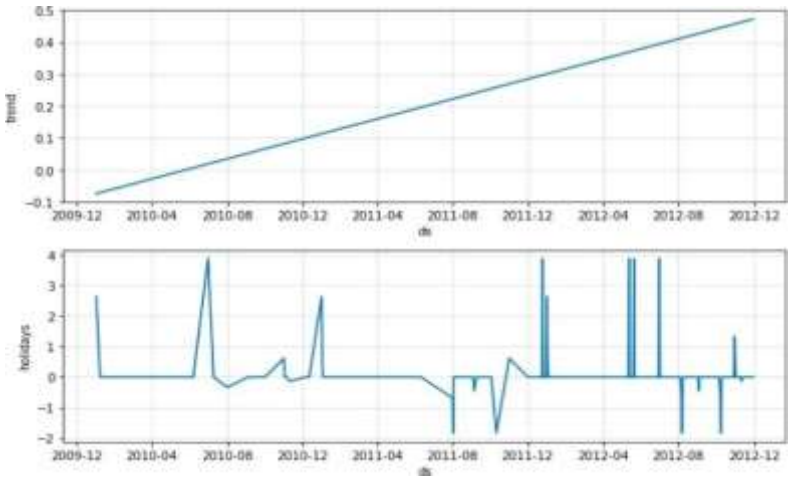


Fig 4.6.2 Trend and Holiday Pattern

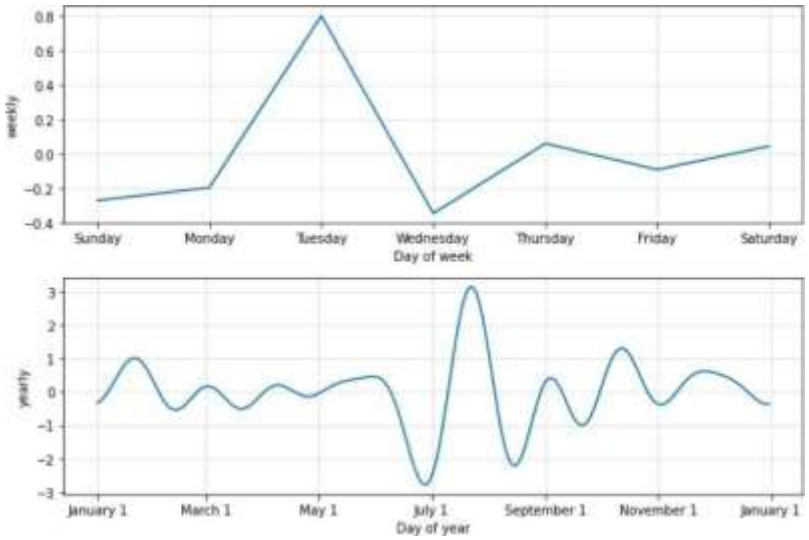


Fig 4.6.3 Weekly and Yearly Pattern

4.7 Results Analysis of Prophet Model

By comparing RMSE values and Spearman correlation values after training the model with a different number of years for all the three cities, we can infer that three years of training is optimal which is on par with the experimental results.

	City	Years of training	RMSE value	Spearman Coeff
0	SanFrancisco	10	49.266170	0.560835
1	SanFrancisco	5	48.174355	0.276535
2	SanFrancisco	4	40.868186	0.249719
3	SanFrancisco	3	44.155286	0.466847
4	SanFrancisco	2	36.926642	0.566736
5	SanFrancisco	1	52.911833	0.498256

	City	Years of training	RMSE value	Spearman Coeff
0	Chicago	10	140.409756	-0.194194
1	Chicago	5	479.216332	-0.225123
2	Chicago	4	4.528871	-0.286346
3	Chicago	3	2.185276	0.171907
4	Chicago	2	2.273025	0.138696
5	Chicago	1	29390.856642	0.063238

	City	Years of training	RMSE value	Spearman Coeff
0	Philadelphia	10	101.415746	0.116549
1	Philadelphia	5	242.582045	0.477680
2	Philadelphia	4	190.191071	0.383935
3	Philadelphia	3	44.337297	0.748318
4	Philadelphia	2	78.308025	0.604010
5	Philadelphia	1	69.265398	0.730905

Fig 4.7.1 Results Analysis of Prophet Model

4.8 Result Analysis of Neural Network Model

The neural network model seems to underperform compared to other models in terms of running time and results. The RMSE value is low and the spearman correlation coefficient is high for three years of training.

	City	Years of training	RMSE value	Spearman Coeff
0	Chicago	10	397.374128	0.314775
1	Chicago	5	353.813001	-0.336692
2	Chicago	4	5.027509	0.234054
3	Chicago	3	5.031015	0.282029
4	Chicago	2	3.588048	-0.748661

	City	Years of training	RMSE value	Spearman Coeff
0	SanFrancisco	10	67.134051	-0.077332
1	SanFrancisco	5	49.728525	-0.017970
2	SanFrancisco	4	47.349213	-0.089991
3	SanFrancisco	3	43.351544	0.163083
4	SanFrancisco	2	46.370308	0.097745

	City	Years of training	RMSE value	Spearman Coeff
0	Philadelphia	10	96.361413	-0.122616
1	Philadelphia	5	77.594485	-0.054211
2	Philadelphia	4	218.326507	0.001103
3	Philadelphia	3	68.483715	0.038390
4	Philadelphia	2	69.852773	-0.068216

Fig 4.8.1 Result Analysis of Neural Network Model

4.9 Result Analysis of LSTM Model

A graph between true values and predicted values is plotted and both these values almost coincide.

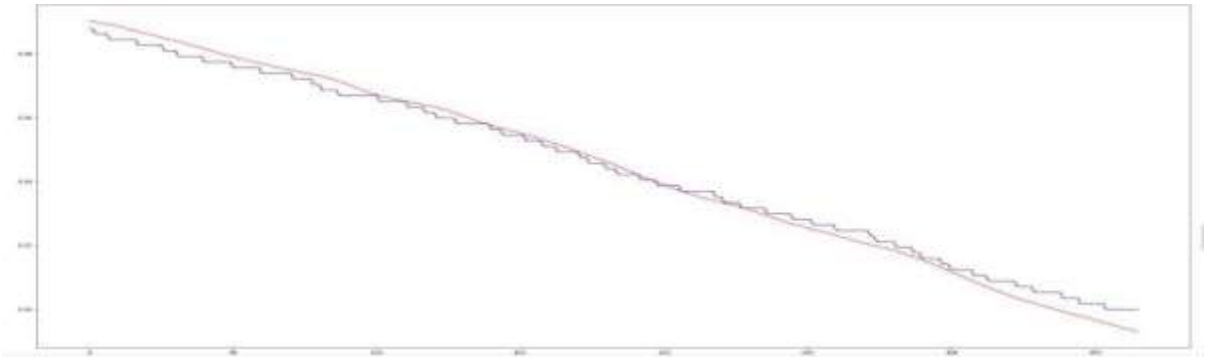


Fig 4.9.1 LSTM Graph

4.10 Sampling

A sampling of attributes is performed to remove the bias.

```
1  counter=Counter(y_over['Category_Id'])
2  plt.bar(counter.keys(), counter.values())
3  plt.show()
```

```
1  counter=Counter(Y_train['Category_Id'])
2  plt.bar(counter.keys(), counter.values())
3  plt.show()
```

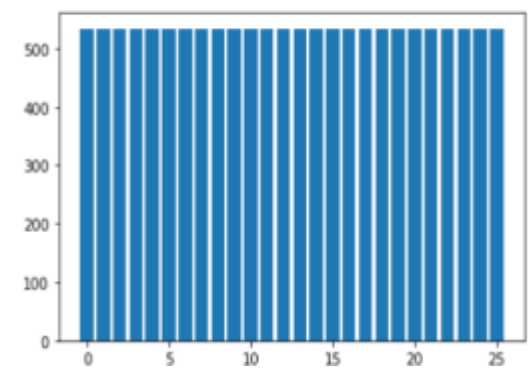


Fig 4.10.1 Attributes with sampling

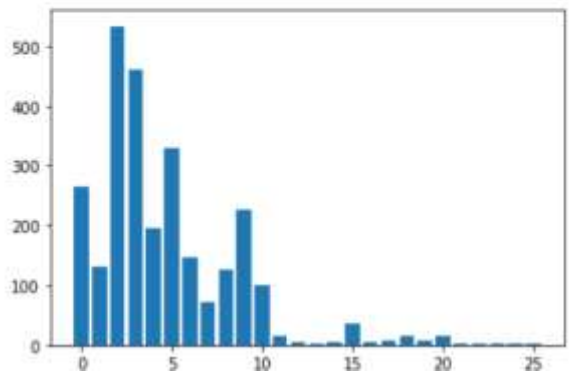


Fig 4.10.2 Attributes without sampling

4.11 Interactive Google map clustering

Using the Maker Cluster Algorithm, the location with latitude, longitude, hour, day and the month of the occurrence of the crimes is shown for all three cities.

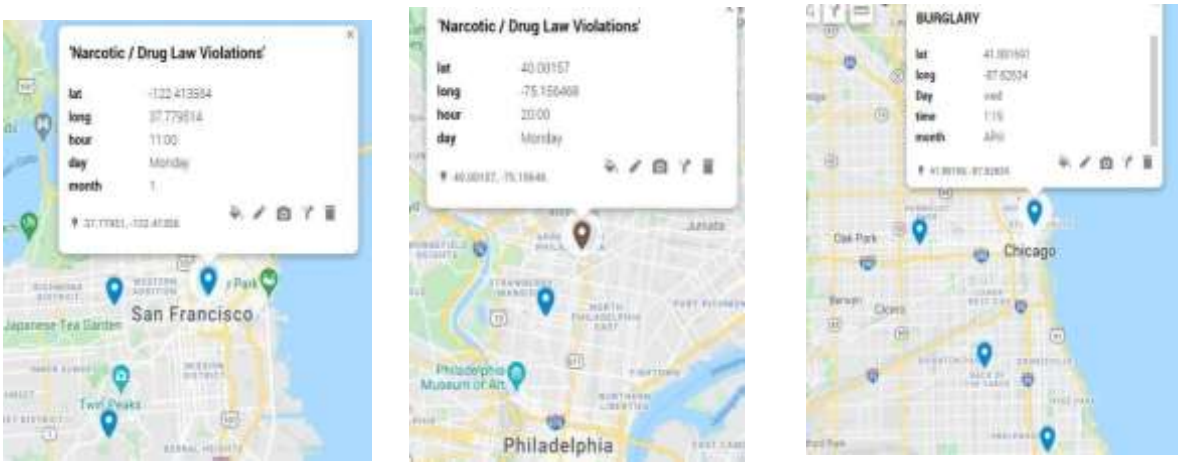


Fig 4.11.1 Interactive Google Maps

Change point	RMSE SF	Spearman SF	RMSE Philly	Spearman Philly	RMSE Chicago	Spearman Chicago
0.9	42.0214	0.48583	48.62192	0.11634	70.6589	0.74863
0.8	35.1743	0.27653	48.4157	0.69875	68.9752	0.70246
0.7	49.7415	0.26984	49.989902	0.12902	68.8136	0.63214
0.6	50.6684	0.26656	49.3163	0.12968	69.74156	0.62635
0.5	51.17328	0.26538	50.3798	0.11549	68.9821	0.79238
0.4	51.34859	0.26517	51.399	0.12969	70.3256	0.6412
0.3	52.4782677	0.25999	51.6845	0.66666	68.1726	0.60235
0.2	52.3014757	0.26000	51.8756	0.67895	77.96548	0.60001
0.1	52.004985	0.26097	51.2548	0.11654	78.98125	0.59217

Table 4.1 Prophet Model Change Point Analysis

City	Random Forest	Gaussian NB(with and without sampling resp for Chicago)		KNN(with and without resp for Chicago)		Neural Network
San Francisco	5.28	0.006944444		2.75		5.9
Philadelphia	4.79	5.108706012		5.126830759		5.58
Chicago	3.28	3.39733	3.59004	3.271920	3.81465	3.93

Table 4.2 Classification Models' Performance

Layers	RMSE SFO	Correlation SFO	RMSE Chicago	Correlation Chicago	RMSE Philly	Correlation Philly
80	43.64	0.366	67.83	0.604	53.68	0.735
70	42.97	0.405	66.34	0.653	52.78	0.727
60	42.44	0.402	66.19	0.638	51.35	0.711
50	42.12	0.422	66.02	0.665	51.21	0.750
40	43.78	0.366	69.71	0.635	52.64	0.684
30	44.74	0.345	70.94	0.629	55.73	0.671
20	43.79	0.211	71.87	0.601	58.20	0.699

Table 4.3 LSTM Performance

Table 4.1 shows the different change point analysis of the Prophet Model along with its evaluation using RMSE and Spearman Correlation. Table 4.2 shows the Mean Absolute Error Rates of different classification algorithms used in this project. It is clear that GaussianNB works better than the other models such as kNN, Random Forest and Artificial Neural Networks as it has considerable MAE values. Table 4.3 shows the performance of LSTM when the number of layers was changed and its corresponding RMSE and Spearman Correlation values. All these metrics work for all cities (i.e) Philadelphia, Chicago and San Francisco.

CHAPTER 5

CONCLUSION AND FUTURE PLANS

5.1 Conclusion

By analysing various prediction and classification models and evaluating their accuracy for this particular data we have at hand, LSTM and Prophet Models exceed the efficiency when compared to Neural Networks (for prediction) and similarly GaussianNB proves to be adopting the right classification method in terms of Mean Absolute Error in comparison to the other classification models. With the advancement of technology in almost every field of profession, it is of paramount importance that the crime rate in the entire world be curbed by utilizing the intelligence of machines. This project witnessed robust and easy-to-use, human-friendly visualisations and future predictions and classifications that would invariably help the police department and detectives to catch the perpetrators and know the occurrence of crimes prior to its actual time. The later results also forecast the type of crime that may happen, given an area and time. With these interesting trends and patterns, the crime rate can be curbed significantly.

5.2 Future Plans

- We plan to finish our ongoing framework for generic data analytics in the future, which will be able to manipulate a variety of data types for a variety of applications.
- To discover more possible interesting trends within these datasets, we expect to use graph mining techniques and fine-grained spatial analysis.
- Furthermore, we plan to perform more practical case studies in order to assess the efficacy and scalability of the various models in our system.

REFERENCES

[1] A. Gandomi and M. Haider, *''Beyond the hype: Big data concepts, methods, and analytics,''* Int. . nf. anage., vol. 35, no. 2, pp. 137–144, Apr. 2015.

[2] J. Zakir and T. Seymour, *''Big data analytics,''* Issues nf. Syst., vol. 16, no. 2, pp. 81–90, 2015.

[3] Y. Wang, L. Kung, W. Y. C. Wang, and C. G. Cegielski, *''An integrated big data analytics-enabled transformation model: Application to health care,''* Inf. anage., vol. 55, no. 1, pp. 64–79, Jan. 2018.

[4] U. Thongsatapornwatana, *''A survey of data mining techniques for analyzing crime patterns,''* in Proc. 2nd sian Conf. efence Technol., Chiang Mai, Thailand, 2016, pp. 123–128.

[5] W. Raghupathi and V. Raghupathi, *''Big data analytics in healthcare: Promise and potential,''* Health nf. Sci. Syst., vol. 2, no. 1, pp. 1–10, Feb. 2014.

[6] J. Archenaa and E. A. M. Anita, *''A survey of big data analytics in healthcare and government,''* Procedia Comput. Sci., vol. 50, pp. 408–413, Apr. 2015.

CHAPTER 7

APPENDIX

7.1 SIMILARITY REPORT

Report

by Par Pa

Submission date: 31-May-2021 11:36AM (UTC+0530)
Submission ID: 1597614875
File name: REPORT_for_PLAG_CHECK.pdf (1.97M)
Word count: 3917
Character count: 19342

Abstract

The influential field of Data Science has the }xitential to not only solve recurring, present real-world problems but also identify and handle the foreseeable problems in the future with ease. With different pattern recognition, v'isualization, prediction and classification techniques in the domain of Machine Learning, in this project, we apply data analytics on criminal database of cities in the United States of America only to find interesting facts and patterns. We employ state-of-the-art prediction algorithms like LSTM, Neural Networks and Prophet Model to derive at the number of crimes happening on a day-to-day basis followed by classification algorithms like GaussianN B, K-Nearest Neighbours, Random Forest Regressor and Anificial Neural Networks to differentiate between the crimes that happen. The results are v'isualized in a human-friendly way so that one can identify and track illicit activities. These outcomes provide assurance for preventing crimes in the future, ultimately, benefiting the police department and detectives.

Specific Contribution

- Visualisation mechanisms including interactis'e google maps
- Pre-processing mechanisms for classification
- Handling multi-class classification algorithms
- Finding an efficient way of classification with an optimal run time, space and accuracy

Specific Learning

- Databricks Community Edition for executing Machine Learning Models
- Handling a large quantity of records
- Unique characteristics of the classification algorithms used and their hyper parameters
- Different visualisation techniques

CHAPTER 1

INTRODUCTION

1.1 Data Analytics

Data Analytics (DA), in recent years, has emerged as a popular method for analysing data and extracting knowledge and relationships in a variety of fields. To analyse this multi-sourced and heterogeneous data, new methods and technologies must be developed. We can easily keep track of incidents, identify correlations between events, make use of resources, and take fast decisions based on this information. The exponential development of cloud computation and data collection and storage technology has resulted in a massive increase in the variety and complexity of data that has been obtained and made publicly accessible. Data mining is a creative, growing and an interdisciplinary research area that can create techniques and paradigms across numerous fields for extracting useful knowledge and hidden trends from data. It is one of the central approaches of Data Science. Data mining can be used to uncover new knowledge or phenomena as well as improve our understanding of existing ones.

1.2 Crime Investigation

One of the main concerns of residents of any large city is their safety. It is extremely difficult for the police department to keep track of incidents and reduce the crime rate. It is impossible to get a bird's eye view of the whole city at all times. It is now possible to predict such features after studying historical data using different machine learning techniques, which is one of the data analytics approaches. The implementation of machine learning methods and statistical mechanisms on crime data analysis or other big data applications such as road accidents or time series data would allow for the study, extraction, and interpretation of related trends and patterns, utilized in crime control and prevention.

1.3 Motivation

The media seems to be continually inundating us with information about crime rates around the world. It's sickening to see so many places that are beautiful to live in but compromise on protection. This was particularly noticeable in cities such as San Francisco, Chicago, and Philadelphia. With this understanding of machine learning, the City Police Department would benefit greatly if they were aware of crime trends in the near future. With a secure database in hand, this analysis can be done for a number of other cities.

CHAPTER 2

OBJECTIVES

2.1 General Objective

To analyse the criminal incidents of three cities namely San Francisco, Chicago and Philadelphia, identify potential crime patterns and predict future crime happenings with respect to the attributes given in the datasets for those three cities.

2.2 Specific Objectives

- To visualise different perceptions of the dataset in the form of pie charts, bar charts, donut charts and time series plots and gather informative insights.
- To predict the number of crime incidents that may happen on a daily basis for the upcoming years which will potentially help the police department of the respective cities.
- To multi-classify the crimes based on the day of occurrence and the district number so that the detectives would have an idea of what type of crime might happen given a date.
- To compare the various prediction and classification algorithms used here.

CHAPTER 3

EXPERIMENTAL WORK/METHEDOLOGY

3.1 Dataset

The dataset is collected from the City's Government website for all the three cities. It has a record of crime incidents from 2003 till date. The dataset has attributes viz Case number of the incident (IncidentNum), Timestamp and Date of the incident, the Category of crimes, Description of the crime scene, Day of the week, Police Department District ID (PdDistrict), how the crime was resolved (Resolution), Address, Latitude and Longitude.The links for the dataset are:

- San Francisco-https://data.sfgov.org/ Public-Safety/Police-Department-Incident-Reports-Historical-2003/tmnf-yvry
- Chicago - https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present
- Philadelphia-https://www. opendataphilly.org/dataset/crime-incidents

3.2 Preprocessing

Before implementation, the dataset needs to be cleaned. The following pre-processing steps were carried out after importing the dataset:

- Time discretization to represent time series forecasting of overall trend (i.e) Division of the dates into Year, Month, Day and time into Hour, Minute and Second.
- NaN values were removed.
- Unwanted features like IndicentNum were omitted.

3.3 Narrative Visualization

3.3.1 Interactive Google Map-based Clustering

An interactive map based on Google Maps was visualised to get a spatial view of the cities, with crime events clustered based on latitude and longitude points. The marker cluster algorithm assisted in clustering the coordinates at various zooming speeds. Person markers are visible on the chart at high zoom levels, allowing you to pinpoint the exact position of the incident. By using this interactive google map, users can take a look at all the crimes on unique dates, time, streets and towns.

3.3.2 Time Series Plot

Crime cases were categorised by year of occurrence, resulting in a year-by-year count of the crimes. For all three towns, this value was plotted using the Plotly library.

3.3.3 Top-10 Crime Cases (Pie chart)

Despite the fact that there are more than 30 crime categories, the distribution of these categories seems to be distorted. As a result, for each of the three towns, the top ten crime scenes were extracted and a pie chart was plotted using the Plotly library.

3d *A* Hourly Trend of Crimes

For all three towns, the crime count was clustered hourly and plotted using the Plotly library, hovering the count. As a result, one can deduce how crime has changed hour by hour and which crimes hav'e occurred during peak hours.

3 d **JWordcloud of Crime Description**

Another }xiwerful visualisation technique is wordcloud, which highlights term s or phrases based on their frequency. As a result, crime descriptors were drawn as a Wordcloud for all three cities in order to gain insight into the most commonly occurring crime category. The most frequently occurring descriptors have a big font, while the least frequently occurring descriptors hav'e a small font.

3 d.fi Box **Plot for Monthly Trend**

The box plot is one of the most powerful visualisation techniques for determining the distribution of values for a given set of values, as well as the mean, median, and interquartile range. For each of the three towns, a monthly crime count is shown in a box map.

3 d.7 Top-lfl **Dates with the Most and Least Crimes**

Donut Chart (also recognized as Doughnut Chart) is a Pie chart with a round hole in the middle, giv'ing it the appearance of a donut, hence the name. Additional data can be shown in the empty space in the middle. Surnrnarisation of top 10 dates with the most and least number of crimes has been represented by a dount chart for all the three cities.

3 A **Prediction Models**

3.4.1 **Prophet** Model

The Prophet model is an additive model that fits non-linear patterns with annual, weekly, and/or regular seasonalit y, as well as holiday impacts, to forecast time series analysis. lt is best for time series with heav'y seasonal effects and several periods of historical data. Missing data and pattern changes are not a problem for the Prophet model, and outliers are usually handled well. This model is designed to properly handle time series features while still having intuitive parameters that can be modified without understanding the underlying mtxlel's information. The seasonal function uses the Fourier series defined below to achieve this, where P denotes the regular pericd:

$$s(f) = \sum^{N} \left(n, \cos \left(\frac{2 \quad t)}{} + \text{basin} \left(\frac{2 \quad t)}{} \right) \right.$$

3.4.2 Neural Network Model

A neural network is made up of a number of neurons, or nodes in the network, that are arranged in multiple layers and linked to each other using different types of connections. layering Here, a multi-layer feed-forward network is used. The input above will be modified by the hidden layer using a nonlinear function.

$$s(z) = \frac{1}{1 + e^{-z}}$$

3.4.3 LSTM Model

The LSTM model is a kind of Recurrent Neural Network (RNN) that can learn long-term dependencies. Due to their ability to preserve the state when identifying trends over time series, LSTMs are particularly useful in prediction for time series involving auto-correlation, i.e., the existence of correlation between the lagged versions of itself and the time series. When each epoch progresses, the recurrent architecture allows the states to be continued or communicated between modified weights. Furthermore, the LSTM cell architecture can improve the RNN by allowing long-term as well as short-term persistence.

3.5 Classification Models

Multi-class classification is the concept of using more than one class to classify instances. In the case of splitting instances into two classes, it is called as binary classification. However, this is very different from multi-label classification where prediction of instances is in the form of lables.

3.5.1 k- Nearest Neighbours Model

This is a non-parametric classification algorithm. The distance from an example to every other example in the training set is measured. 'k' examples with the smallest distances are taken and these are called the "k- Nearest Neighbours". The most represented classes are taken as the output.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.5.2 Gaussian Naïve Bayes

When Gaussian normal distribution is followed and it supports continuous data in addition to Naïve Bayes algorithm, it is called Gaussian Naïve Bayes. As the name suggests, it is based on the Bayes theorem and has high functionality.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

3.5.3 Random Forest

Random decision forest is used both classification and regression to predict the data. They generate reasonable predictions across a wide range of data while requiring little configuration. Random forest uses decision trees to predict results.

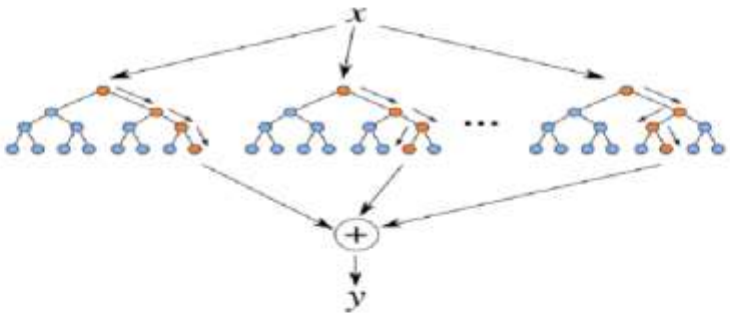


Fig 3.5.3 Random Forest

3.5.4 Artificial Neural Network

Artificial neurons are at the heart of ANNs. Each neuron takes input from a number of other neurons, multiplies it by weights assigned to it, adds it up, and sends the result to one or more neurons. Before transferring the output to the next variable, certain artificial neurons may apply an activation function to it. Furthermore, there is no central CPU managing the logic, regardless of the underlying hardware. Rather, the logic is spread out among thousands of smaller artificial neurons.

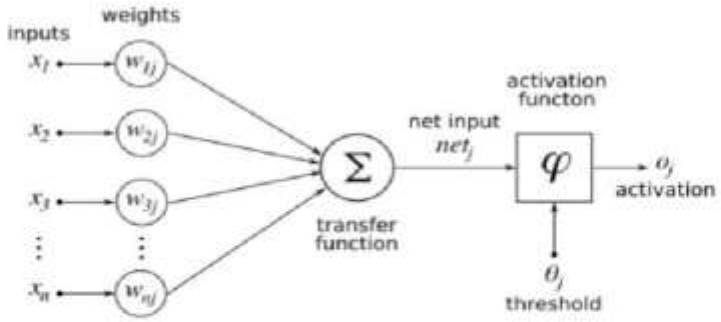


Fig 3.5.4 Artificial Neural Networks

3.6 Comparison metrics

To forecast crime trends, we looked at time series forecast and deep learning models. In terms of varying arguments and different amounts of training samples, the Root Mean Square Error (RMSE), Spearman correlation, and Mean Absolute Error (MAE) are employed for performance evaluation.

3.6.1 Root Mean Square Error (RMSE)

The Root Mean Square Error (RMSE) or the Root Mean Square Deviation (RMSD) is a commonly used metric for comparing predicted and observed values (sample or population values) by a model or estimator. The square root of the second sample moment of the discrepancies between anticipated and observed values, or the quadratic mean of these differences, is represented by the RMSD.

$$\text{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

RMSD = Root Mean Square Deviation

i = variable i

N = Number of non-missing data points

x_i = Actual observations time series

$\hat{x}_i$ = Estimated time series

3.6.2 Spearman Correlation

Spearman's rank correlation coefficient, sometimes known as Spearman's, is a nonparametric rank correlation measure (statistical dependence between the rankings of two variables). It determines how well a monotonic function can describe the relationship between two variables. When findings have a similar (or identical for a correlation of 1) rank between the two variables, the Spearman correlation will be high, and when observations have a dissimilar (or fully opposed for a correlation of 1) rank between the two variables, the Spearman correlation will be low.

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

cov is the covariance; σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y.

3.6.3 Mean Absolute Error

A measure of mistakes between paired observations describing the same phenomenon is called mean absolute error (MAE). Comparisons of predicted against observed, subsequent time versus starting time, and one measuring technique versus another measurement technique are examples of Y against X.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

As a result, it is an arithmetic average of the absolute errors $|e_i|=|y_i-x_i|$, where y_i is the prediction and x_i is the true value.

We use MAE for evaluating our classification model's accuracy since it allows us to differentiate between forecasts of varied time series in varied scaling. It is less susceptible to outliers than MSE because it does not penalise large errors. It produces a linear value that equalises the weighted individual disparities. The model's performance improves as the value decreases.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Experimental Results of Philadelphia

In Figure 4.1.1, using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. Fig 4.1.2 is a time series plot wherein the number of crimes is plotted against the year. In Figure 4.1.3, a pie chart for the top 10 crimes is displayed. Theft is one of the most occurring crimes. In Figure 4.1.4, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. In Figure 4.1.5, a box plot between the month and number of crimes is plotted. In Figure 4.1.6, a doughnut plot of the top 10 crime occurring dates is depicted. 27-10-2020 and 28-10-2020 show significant results. Figure 4.1.7 is an hour-wise plot.

Fig 4.1.1 Marker Cluster - Philadelphia

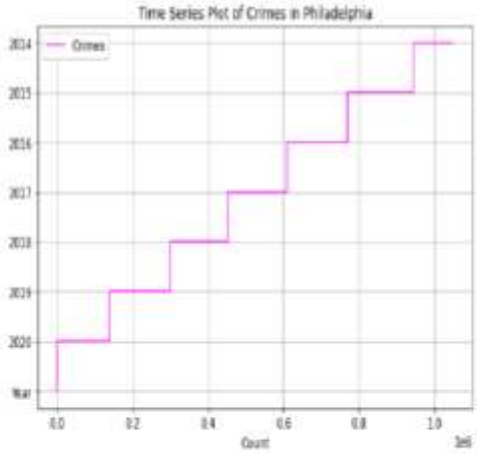


Fig 4.1.2 Time Series - Philadelphia

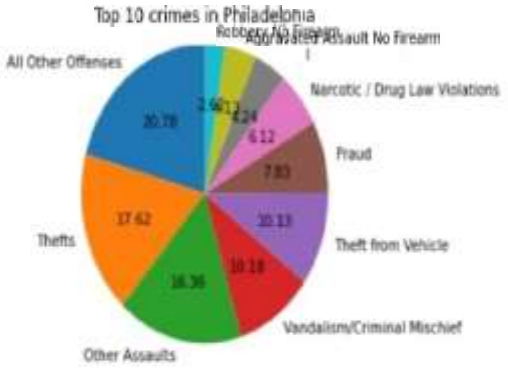


Fig 4. 1 .3 Pie Chart - Philadelphia

Fig 4.1.4 Word Cloud - Philadelphia

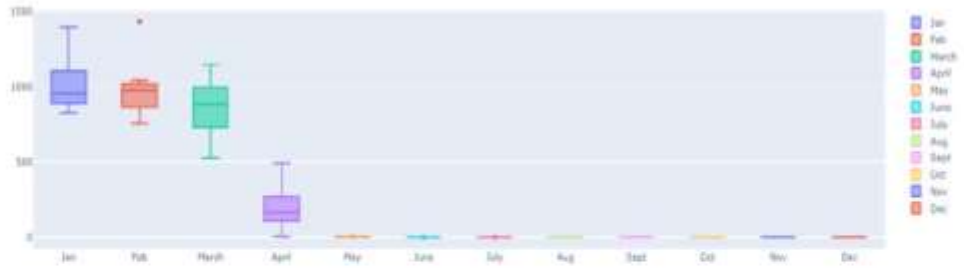


Fig 4.1.5 Box Plot - Philadelphia

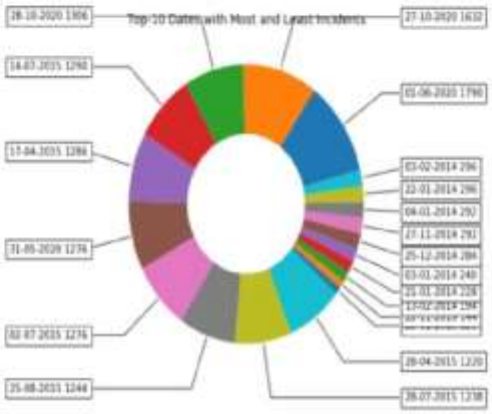


Fig 4.1.6 Doughnut Chart - Philadelphia

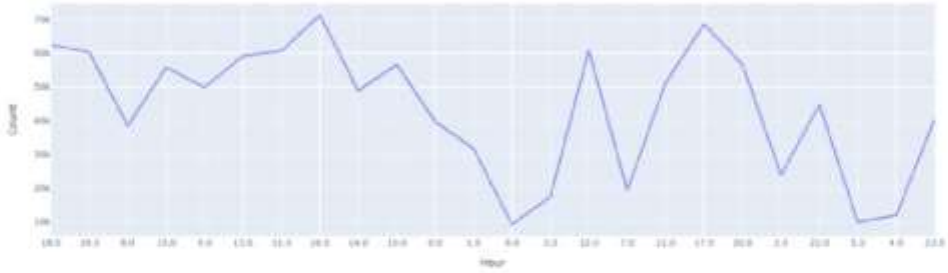


Fig 4.1.7 Hourly Plot - Philadelphia

4.2 Experimental Results of Chicago

In Figure 4.2.1, a pie chart for the top 10 crimes is displayed. In Figure 4.2.2, a box plot between the month and number of crimes is plotted. In Figure 4.2.3, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. In Figure 4.2.4, using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. Figure 4.2.5 is an hour-wise plot.

18

Fig 4.2.1 Pie Chart - Chicago

Fig 4.2.2 Box Plot - Chicago

Fig 4.2.3 Word Cloud - Chicago

Fig 4.2.4 Marker Cluster - Chicago

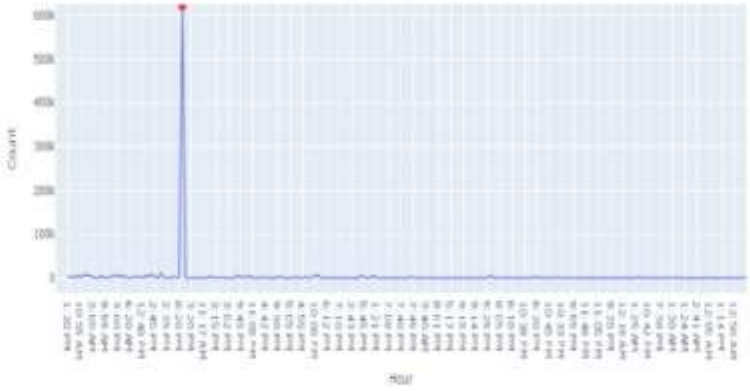


Fig 4.2.5 Hourly Plot - Chicago

4d **Experimental Results of San Francisco**

In Figure 4.3.1 , using the marker cluster algorithm, the markers are placed in their respective locations where crimes happen. In Figure 4.3.2, a doughnut pl ot of the top 10 crime occurring dates is depicted. I - I -20 I I and I - I -20 1 3 show significant results. In Figure 4.3.3, a pie chart for the t op 10 crimes is displayed. Larceny/Theft is one of the most occurring crimes. In Figure 4.3.4, the word cloud of crimes is displayed. The crime words with higher depth indicate that they occur more. Figure 4.3.5 is an hour-wise plot. In Figure 4.3.6, a box plot between the month and number of crimes is plotted.

Fig 4.3. I Marker Cluster — San Francisco

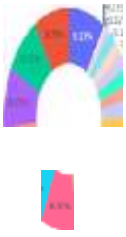


Fig 4.3.2 Doughnut Chart — San Francisco

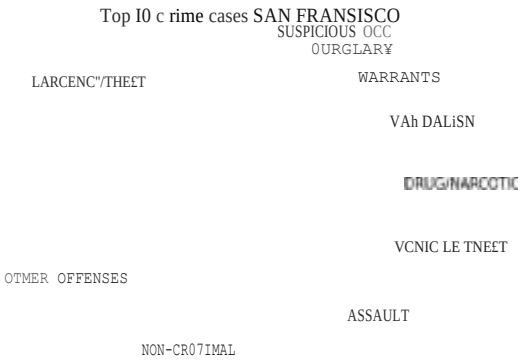


Fig 4.3.3 Pie Chart — San Francisco

Fig 4.3.4 Word Cloud– San Francisco

Fig 4.3.5 Hourly Plot– San Francisco

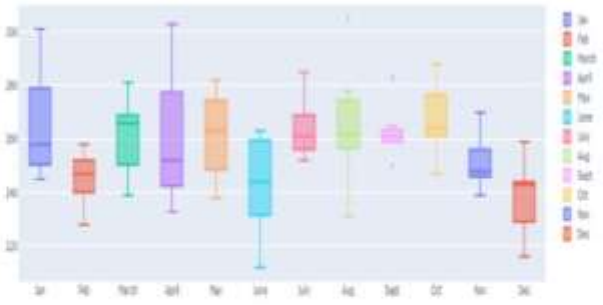


Fig 4.3.6 Box Plot — San Francisco

4 *A* Prophet Model Results (San Francisco)

Figure 4.4.1 shows the overall trend of crimes. 2007-2008 is the predicted period. In Figure 4.4.2, we can see the components decomposition plot. Figure 4.4.3 is the trend decomposition plot. There is a decline in crime trend from 2007 to 2005.

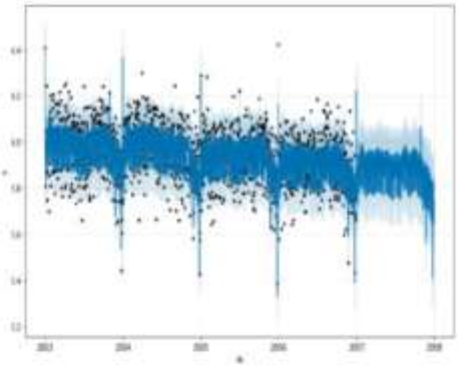


Fig 4.4.1 Overall Trend— San Francisco

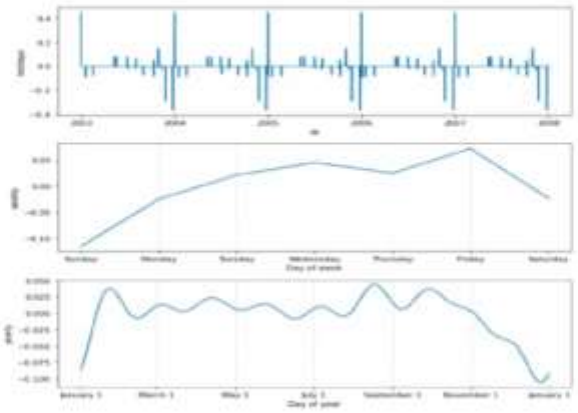


Fig 4.4.2 CoinJx›nent Decomposition Plot — San Francisco

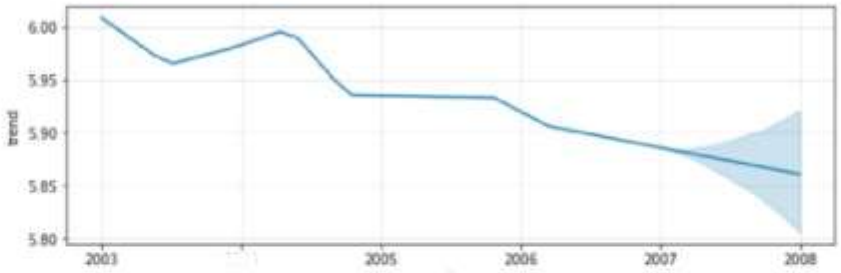


Fig 4.4.3 Trend Decomposition — San Francisco

4J Prophet Model Results (Philadelphia)

Figure 4.5.1 is the overall trend plot with 201 2-2013 being the predicted peri‹xl. Figure 4.5.2 is the component decomposition plot. Figure 4.5.3 shows the yearly trend.

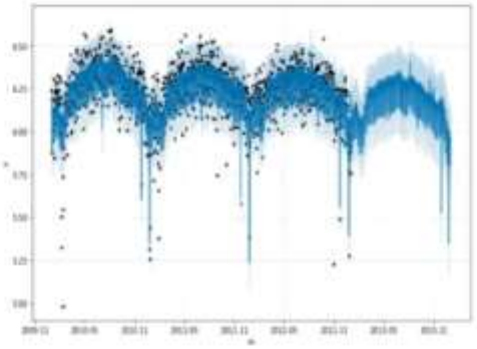


Fig 4.5.1 Os'erall Trend - Philadelphia

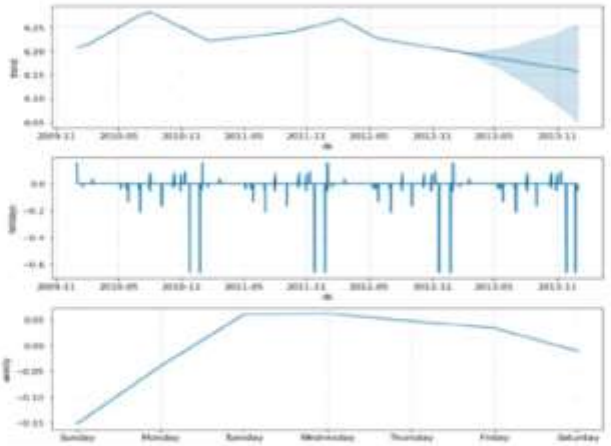


Fig 4.5.2 Component Oecomposition Plot - Phil adel phia

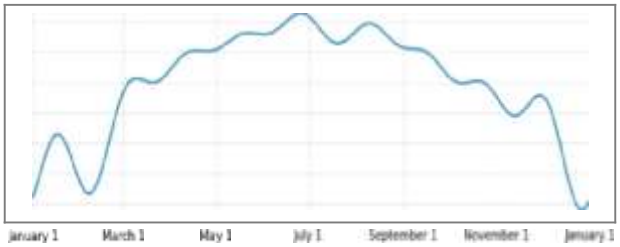


Fig 4.5.3 Yearly Decomposition

4ñ Prophet Model **Results** (Chicago)

Figure 4.6.1 is the overall trend plot with 2011-2012 being the predicted period. Figure 4.6.2 is the trend and holiday component decomposition plot and Figure 4.6.3 is the yearly and weekly component decomposition plot. There is a peak in the crime rate on Tuesday.

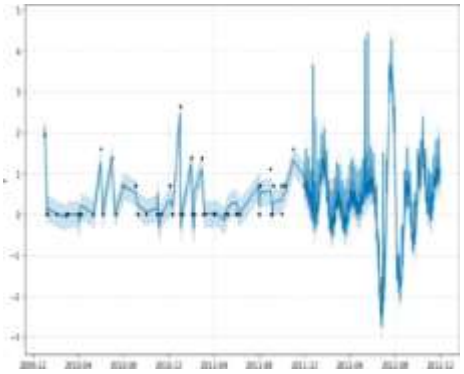


Fig 4.6.1 Overall Trend - Chicago

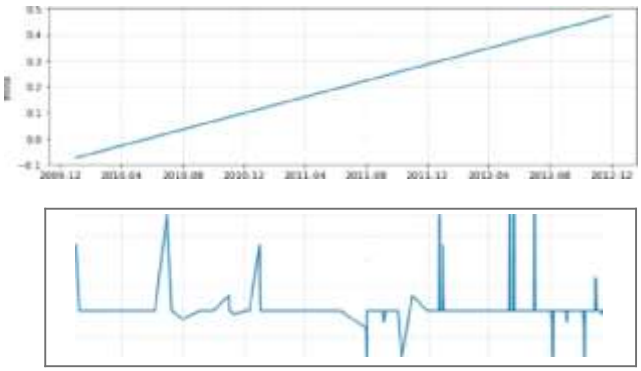


Fig 4.6.2 Trend and Holiday Pattern

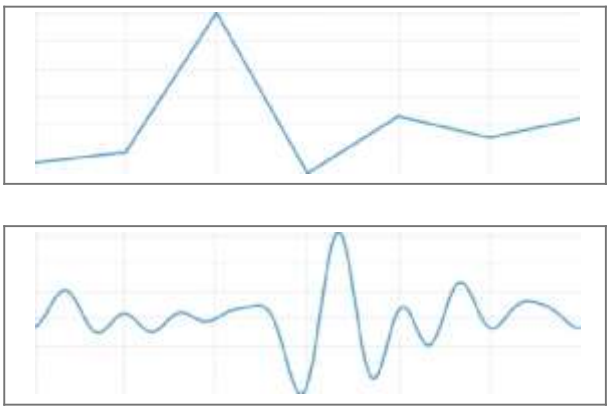
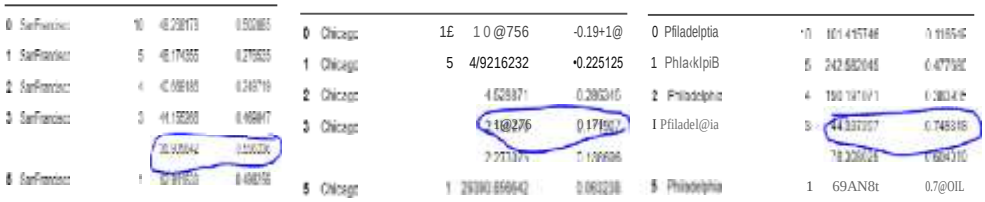


Fig 4.6.3 Weekly and Yearly Pattern

4.7 Remlts Analysis of Prophet Model

By comparing RMSE values and Spearman correlation values after training the model with a different number of years for all the three cities, we can infer that three years of training is optimal which is on par with the experimental results.



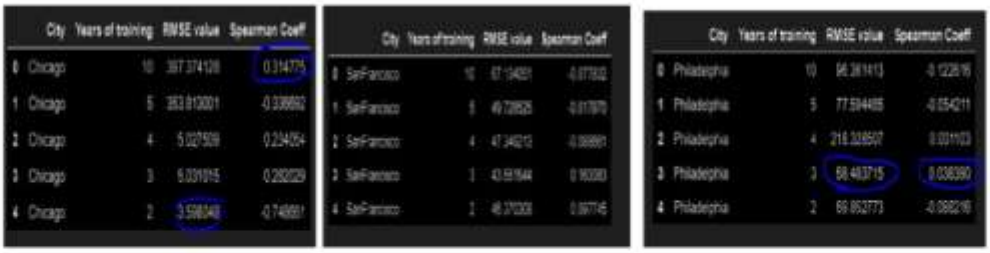
0 Chicago	1£	1 0 @756	-0.19+1@
1 Chicago	5	4/9216232	•0.225125
2 Chicago			
3 Chicago		2 16@276	
5 Chicago	1	28390.656642	

0 Pfiladelptia			
1 Phla<klpiB			
2 Philadelphia			
I Pfiladel@ia			
5 Philadelphia	1	69AN8t	0.7@OIL

Fig 4.7.1 Results Analysis of Prophet Model

4& Remlt **Analysis of Neurol Network** Model

The neural network model seems to underperform compared to other models in **terms** of running time and results. The RMSE value is low and the spearman correlation coefficient is high for three years of training.



	City	Years of training	RMSE value	Spearman Coeff
0	Chicago	10	397.374128	0.314775
1	Chicago	5	383.810001	-0.339892
2	Chicago	4	5.027509	0.234054
3	Chicago	3	5.031015	0.260529
4	Chicago	2	5.598048	-0.748691

	City	Years of training	RMSE value	Spearman Coeff
0	SanFrancisco	10	67.134081	-0.077502
1	SanFrancisco	5	49.728525	-0.017970
2	SanFrancisco	4	47.349213	-0.088891
3	SanFrancisco	3	43.591544	0.163083
4	SanFrancisco	2	46.272203	0.097745

	City	Years of training	RMSE value	Spearman Coeff
0	Philadelphia	10	66.261413	-0.122816
1	Philadelphia	5	77.594485	-0.054211
2	Philadelphia	4	218.328507	0.031103
3	Philadelphia	3	68.483715	0.036390
4	Philadelphia	2	69.852773	-0.088216

Fig 4.8. I Result Analysis of Neural Network Model

4fi Result Analysis of LSTM Model

A graph between true values and predicted values is plotted and both these values almost coincide.

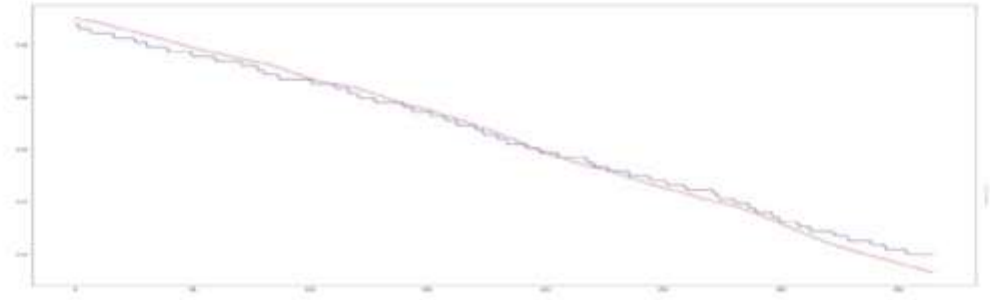


Fig 4.9.1 LSTM Graph

4.10 Sampling

A sampling of attributes is performed to remove the bias.

```
counter=Counter(y_over['Category_Id'])
plt.bar(counter.keys(), counter.values())
plt.show()
```

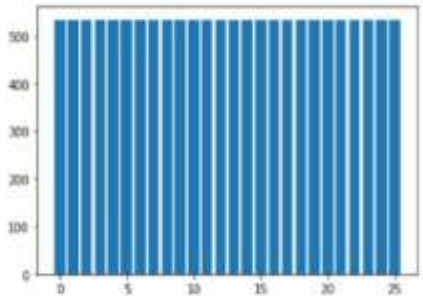


Fig 4.10.1 Attributes with sampling

```
counter=Counter(Y_train['Category_Id'])
plt.bar(counter.keys(), counter.values())
plt.show()
```

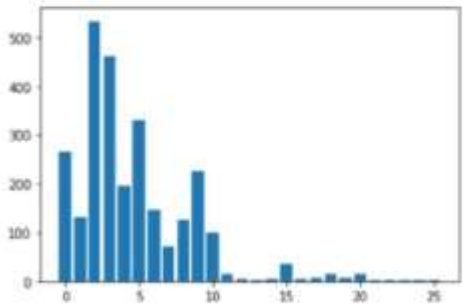


Fig 4.10.2 Attributes without sampling

4.11 **Interactive** C’.ongle **map clustering**

Using the Maker Cluster Algorithm, the location with latitude, longitude, hour, day and the month of the occurrence of the crimes is shown for all three cities.

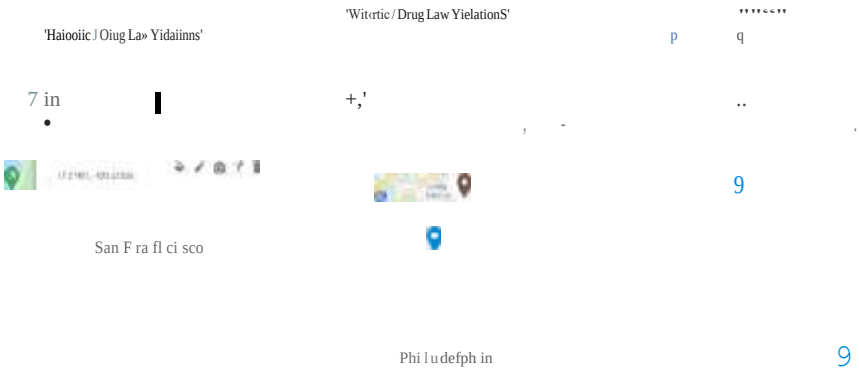


Fig 4.1 I .1 Interactive Google Maps

Change point	RMSE SF	Spearman SF	RMSE Phill y	Spearman Philly	RMSE Chicago	Spearman Chicago
0.9	42.0214	0.48583	48.62192	0.11634	70.6589	0.74563
0.8	35.1743	0.27653	48.4157	0.69875	68.975 2	0.70246
0.7	49.7415	0.26984	49.989902	0. 1 2002	68.8136	0.63214
0.6	50.6684	0.26656	49.3163	0. 1 2968	69.74 156	0.62635
05	5 I . 17328	0.26538	50.3795	0.1 1549	68.982 I	0.79238
0.4	5 I .34859	0.26517	5 I .399	0.1 2969	70.3256	0.641 2
0.3	52.4782677	0.25999	5 I .6845	0.66666	68.1726	0.60235
0.2	52.3014757	0.26000	5 I .8756	0.67895	77.96548	0.6000 I
0.1	52.004985	0.26097	5 I .2545	0. 1 1 654	78.981 25	0.59217

Table 4.1 Prophet Model Change Point Analysis

City	Random Forest	Gaussian NB(with and without sampling rasp for Chicago)		KNN(with and without resp for Chicago)		Neural Network
San Francisco	5.28	0.006944444		2.75		5.9
Philadelphia	4.79	5.108706012		5.126830759		5.58
Chicago	3.28	3.39733	3.59004	3.271920	3.81465	3.93

Table 4.2 Classification Models' Performance

Layers	RMSE SFO	Correlation SFO	RMSE Chicago	Correlation Chicago	RMSE Philly	Correlation Philly
80	43.64	0.366	67.83	0.604	53.68	0.735
70	42.97	0.405	66.34	0.653	52.78	0.727
60	42.44	0.402	66.19	0.638	5 l .35	0.711
50	42.12	0.422	66.02	0.665	5 l .21	0.750
40	43.78	0.366	69.71	0.635	52.64	0.684
30	44.74	0.345	70.94	0.629	55.73	0.671
20	43.79	0.21 l	7 l .87	0.60 l	58.20	0.699

Table 4.3 LSTM Performance

Table 4.1 shows the different change point analysis of the Prophet Model along with its evaluation using RMSE and Spearman Correlation. Table 4.2 shows the Mean Absolute Error Rates of different classification algorithms used in this project. lt is clear that GaussianN B works better than the other rnrxlels such as kNN, Random Forest and Artificial Neural Networks as it has considerable MAE values. Table 4.3 shows the performance of LSTM when the number of layers was changed and its corresponding RMSE and Spearman Correlation values. All these metrics work for all cities (i.e) Philadelphia, Chicago and San Francisco.

CHAPTER 5

CONCLUSION AND FUTURE PLANS

5.1 Conclusion

By analysing various prediction and classification models and evaluating their accuracy for this particular data we have at hand, LSTM and Prophet Models exceed the efficiency when compared to Neural Networks (for prediction) and similarly GaussianNB proves to be adopting the right classification method in terms of Mean Absolute Error in comparison to the other classification models. With the advancement of technology in almost every field of profession, it is of paramount importance that the crime rate in the entire world be curbed by utilizing the intelligence of machines. This project witnessed robust and easy-to-use, human-friendly visualisations and future predictions and classifications that would invariably help the police department and detectives to catch the perpetrators and know the occurrence of crimes prior to its actual time. The later results also forecast the type of crime that may happen, given an area and time. With these interesting trends and patterns, the crime rate can be curbed significantly.

5.2 Future Plans

- We plan to finish our ongoing framework for generic data analytics in the future, which will be able to manipulate a variety of data types for a variety of applications.
- To discover more possible interesting trends within these datasets, we expect to use graph mining techniques and fine-grained spatial analysis.
- Furthermore, we plan to perform more practical case studies in order to assess the efficacy and scalability of the various models in our system.

Report

ORIGINALITY REPORT

10%	4%	6%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Mingchen Feng, Jiangbin Zheng, Jinchang Ren, Amir Hussain, Xiuxiu Li, Yue Xi, Qiaoyuan Liu. "Big Data Analytics and Mining for Effective Visualization and Trends Forecasting of Crime Data", IEEE Access, 2019 Publication	2%
2	Submitted to Huddersfield New College Student Paper	1%
3	Submitted to Al Qasimia University Student Paper	1%
4	Submitted to University of Strathclyde Student Paper	1%
5	Submitted to Aligarh Muslim University, Aligarh Student Paper	1%
6	Jian Yuan, Chunhui Yu, Ling Wang, Wanjing Ma. "Driver Back-Tracing Based on Automated Vehicle Identification Data", Transportation Research Record: Journal of the Transportation Research Board, 2019 Publication	1%

7	Submitted to Pandit Deendayal Petroleum University Student Paper	1%
8	Submitted to Northcentral Student Paper	1%
9	Submitted to CSU, San Jose State University Student Paper	<1%
10	prism.ucalgary.ca Internet Source	<1%
11	Submitted to Bournemouth University Student Paper	<1%
12	Submitted to University of Florida Student Paper	<1%
13	Submitted to Melbourne Institute of Technology Student Paper	<1%
14	Submitted to International Institute of Information Technology, Hyderabad Student Paper	<1%
15	synapse.koreamed.org Internet Source	<1%
16	Submitted to Temple University Student Paper	<1%
17	Submitted to Queen's University of Belfast Student Paper	<1%

18	www.repository.cam.ac.uk Internet Source	<1%
19	constellation.uqac.ca Internet Source	<1%
20	Submitted to Queen Mary and Westfield College Student Paper	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

7.2 Source Code

#Visualisation (Philadelphia)

```
file_location = "/FileStore/tables/Philadelphia_dataset.csv"
file_type = "csv"
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)
display(df)

file = df.toPandas()
file.head()
pip install folium
pip install geopy
pip install wordcloud
!pip install plotly

import folium
from branca.element import Figure
from geopy.geocoders import Nominatim
fig_size = Figure(width = 500, height = 500)
geolocator = Nominatim(user_agent="geoapiExercises")
latitude = [39.88388114, 39.98199552, 40.03120371, 39.92328314, 40.0021239,
39.98254529, 40.0287841, 39.97876149, 40.02518448, 39.98453229, 39.87883267,
39.9540352999999, 40.08244258, 39.99379565]
longitude = [-75.23070645, -75.12379868, -75.15409118, -75.16265119, -75.14409686, -
75.12473953, -75.05731383, -75.16960489, -75.10319215, -75.15700603, -75.24313797, -
75.16244935, -74.96471021, -75.11168277]
draw_map = folium.Map(width=500,height=500, location = [39.96233372, -75.16144594],
zoom_start = 11, min_zoom = 7, max_zoom = 18)

for i in latitude:
```

```
  for j in longitude:
    location = geolocator.reverse(str(i)+","+str(j))
    address = location.raw['address']
    city = address.get('city',")
    folium.Marker(location = [i,j], popup = city, tooltip = 'Click here to know the
city').add_to(draw_map)
draw_map

import matplotlib.pyplot as plt
%matplotlib inline
plt.rc('font', size = 12)
figure, axis = plt.subplots(figsize = (10,6))
axis.plot(file._c8, color = 'magenta', label = 'Crimes')
axis.set_xlabel('Count')
axis.set_title("Time Series Plot of Crimes in Philadelphia")
axis.grid(True)
axis.legend(loc = 'upper left')
crime_list = file['_c16'].tolist()
crime_list = [x for x in crime_list[1:] if str(x)!='nan']
crime_list

crime_count = len(crime_list)
temp = set(crime_list)
unique_crime_list = list(temp)
unique_crime_count = len(unique_crime_list)
print("Before : "+str(crime_count))
print("After : "+str(unique_crime_count))
print("CRIMES IN PHILADELPHIA :")
unique_crime_list

pie_chart_dict = {}
for i in unique_crime_list:
pie_chart_dict[i] = 0
pie_chart_dict

for x in crime_list:
pie_chart_dict[x]+=1
pie_chart_dict

from operator import itemgetter
for x in crime_list:
pie_chart_dict[x]+=1
N = 10
```

```
sorted_pie_chart_dict = dict(sorted(pie_chart_dict.items(), key =
itemgetter(1), reverse = True)[:N])
print("The top ten crimes are : ")
sorted_pie_chart_dict

import matplotlib.pyplot as plt
pie_labels = 'All Other Offenses','Thefts', 'Other Assaults', 'Vandalism/Criminal Mischief',
'Theft from Vehicle', 'Fraud', 'Narcotic / Drug Law Violations', 'Burglary Residential',
'Aggravated Assault No Firearm', 'Robbery No Firearm'
values = [756380, 641260, 595584, 370680, 368636, 284884, 222740, 154416, 150456,
95288]
figure, axes = plt.subplots()
axes.pie(values, labels = pie_labels, autopct = '%.2f', startangle = 90)
axes.axis('equal')
plt.title('Top 10 crimes in Philadelphia')
plt.show()

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
words = ''
stop = set(STOPWORDS)
for i in unique_crime_list:
  i = str(i)
  tokens = i.split()
  for j in range(len(tokens)):
    tokens[j] = tokens[j].lower()
  words+=" ".join(tokens)+" "
wordcloud = WordCloud(width = 500, height = 500, background_color = 'grey',
                stopwords = stop, min_font_size = 10).generate(words)

plt.figure(figsize = (8,8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()

new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file

import pandas as pd

new_header = file.iloc[0]
```

```
file = file[1:]
file.columns = new_header
file['Date'] = pd.to_datetime(file['Date'])
file['DayOfWeek'] = file['Date'].dt.day_name()
file

values = []
labels = file.DayOfWeek.unique()

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='1':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values.append(count)

values1=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='2':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values1.append(count)

values2=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='3':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values2.append(count)

values3=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='4':#(1-12 values)
```

```
      if list1['DayOfWeek']==each:
        count=count+1
  values3.append(count)

values4=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='5':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values4.append(count)

values5=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='6':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values5.append(count)

values6=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='7':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values6.append(count)

values7=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='8':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values7.append(count)
```

```
values8=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='9':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values8.append(count)

values9=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='10':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values9.append(count)

values10=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='11':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values10.append(count)

values11=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='12':#(1-12 values)
      if list1['DayOfWeek']==each:
      count=count+1
  values11.append(count)

import plotly.graph_objects as go
fig=go.Figure()
fig.add_trace(go.Box(y=values,name="Jan"))
fig.add_trace(go.Box(y=values1,name="Feb"))
```

```
fig.add_trace(go.Box(y=values2,name="March"))
fig.add_trace(go.Box(y=values3,name="April"))
fig.add_trace(go.Box(y=values4,name="May"))
fig.add_trace(go.Box(y=values5,name="June"))
fig.add_trace(go.Box(y=values6,name="July"))
fig.add_trace(go.Box(y=values7,name="Aug"))
fig.add_trace(go.Box(y=values8,name="Sept"))
fig.add_trace(go.Box(y=values9,name="Oct"))
fig.add_trace(go.Box(y=values10,name="Nov"))
fig.add_trace(go.Box(y=values11,name="Dec"))
fig.update_traces(quartilemethod="inclusive")
fig.show()

date_count = len(date_list)
temp = set(date_list)
unique_date_list = list(temp)
unique_date_count = len(unique_date_list)
print("Before : "+str(date_count))
print("After : "+str(unique_date_count))

pie_chart_dict2 = {}
for i in unique_date_list:
  pie_chart_dict2[i] = 0
pie_chart_dict2

from operator import itemgetter
for x in date_list:
  pie_chart_dict2[x]+=1
N = 10
sorted_pie_chart_dict2 = dict(sorted(pie_chart_dict2.items(), key =
itemgetter(1), reverse = False)[:N])
print("Dates with the least number of crimes: ")
sorted_pie_chart_dict2

from operator import itemgetter
N = 10
reverse_sorted_pie_chart_dict2 = dict(sorted(pie_chart_dict2.items(), key
= itemgetter(1), reverse = True)                                    [:N])
print("Dates with the most number of crimes: ")
reverse_sorted_pie_chart_dict2
def Merge(dict1, dict2):
  result = {**dict1, **dict2}
  return result
final_dict = Merge(sorted_pie_chart_dict2, reverse_sorted_pie_chart_dict2)
```

```
print(final_dict)

def Merge(dict1, dict2):
  result = {**dict1, **dict2}
  return result

final_dict = Merge(sorted_pie_chart_dict2, reverse_sorted_pie_chart_dict2)
print(final_dict)for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(pie_labels[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)
ax.set_title("Top-10 Dates with Most and Least Incidents")
plt.show()

import pandas as pd
cate2 = pd.DataFrame({"Hour":file.Hour.unique()})
labels2 = file.Hour.unique()
values2=[]
for each in labels2:
    values2.append(len(file[file.Hour==each]))
val2=pd.DataFrame({"Count":values2})
result2=pd.concat([cate2,val2],axis=1)

from scipy.signal import find_peaks
import plotly.graph_objects as go
import plotly.express as px
indices = find_peaks(result2['Count'], threshold=50000)[0]
fig = px.line(result2,x='Hour',y='Count')
fig.add_trace(go.Scatter(
    x = result2['Hour'][indices],
    y = [result2['Count'][j] for j in indices],
    mode = 'markers',
    marker = dict(
        size = 8,
        color = 'red',
        symbol = 'cross'
    ),
    name = 'Detected Peaks'
))
```

```
fig.show()

#Visualisation (Chicago)
file_location = "/FileStore/tables/Chicago_dataset.csv"
file_type = "csv"
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .load(file_location)
display(df)

import folium
from branca.element import Figure
from geopy.geocoders import Nominatim
fig_size = Figure(width = 500, height = 500)
geolocator = Nominatim(user_agent="geoapiExercises")
latitude = [41.81511728, 41.89508047, 41.93740577, 41.88190344, 41.74437888,
41.914635600000004, 41.85198889, 41.88281374, 41.76364755, 41.97596842, 41.70715492
, 41.80967831, 41.90712726, 41.74809734]
longitude = [-87.66999956, -87.76540045, -87.71664969, -87.75512115, -87.65843064, -
87.68163091, -87.68921912, -87.70432572, -87.72234469, -87.76801426, -87.62424499, -
87.59363893, -87.67823202, -87.66216618]
draw_map = folium.Map(width=500,height=500, location = [41.89386992, -
87.75434109999999], zoom_start = 11, min_zoom = 7, max_zoom = 18)
for i in latitude:
  for j in longitude:
    location = geolocator.reverse(str(i)+","+str(j))
    address = location.raw['address']
    city = address.get('city',")
    folium.Marker(location = [i,j], popup = city, tooltip = 'Click here to know the
city').add_to(draw_map)
draw_map

file.dropna(inplace=True)
file.isnull().sum()

import pandas as pd
cate2=pd.DataFrame({"Year":file.Year.unique()})
cate2

labels2 = file.Year.unique()
```

```
values2=[]
for each in labels2:
    values2.append(len(file[file.Year==each]))
val2=pd.DataFrame({"Count":values2})
result2=pd.concat([cate2,val2],axis=1)
result2

result2.Year=result2.Year.replace({"#VALUE!":"2003"})
result2.sort_values(by=['Year'],inplace=True)

from scipy.signal import find_peaks
import plotly.graph_objects as go
import plotly.express as px
indices = find_peaks(result2['Count'], threshold=610000)[0]
indices

fig=px.line(result2,x='Year',y='Count')
fig.show()

import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
plt.plot(result2.Year,result2.Count)
plt.show()

cate3=pd.DataFrame({"Hour":file.Time.unique()})
cate3
labels2 = file.Time.unique()
values2=[]
for each in labels2:
    values2.append(len(file[file.Time==each]))
val2=pd.DataFrame({"Count":values2})
result3=pd.concat([cate3,val2],axis=1)
result3

indices = find_peaks(result3['Count'], threshold=11000)[0]
indices

fig=px.line(result3,x='Hour',y='Count')
fig.add_trace(go.Scatter(
    x=result3['Hour'][indices],
    y=[result3['Count'][j] for j in indices],
    mode='markers',
    marker=dict(
        size=8,
```

```
        color='red',
        symbol='cross'
    ),
    name='Detected Peaks'
))
fig.show()

import numpy as np

values=[]
labels=file.Day.unique()
labels=np.delete(labels,np.where(labels=="#VALUE!"))

l=file.iloc[2]
l

l['Month']
values=[]
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='1':
      if list1['Day']==each:
        count=count+1
values.append(count)
values

values=[]
labels=file.Day.unique()
labels=np.delete(labels,np.where(labels=="#VALUE!"))
for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='1':
      if list1['Day']==each:
      count=count+1
  values.append(count)
values1=[]

for each in labels:
  count=0
  for i in range(0,21609):
```

```
    list1=file.iloc[i]
    if list1['Month']=='2':
      if list1['Day']==each:
      count=count+1
  values1.append(count)
values2=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='3':
      if list1['Day']==each:
      count=count+1
  values2.append(count)
values3=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='4':
      if list1['Day']==each:
      count=count+1
  values3.append(count)
values4=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='5':
      if list1['Day']==each:
      count=count+1
  values4.append(count)
values5=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='6':
      if list1['Day']==each:
      count=count+1
```

```
  values5.append(count)
values6=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='7':
      if list1['Day']==each:
      count=count+1
  values6.append(count)
values7=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='8':
      if list1['Day']==each:
      count=count+1
  values7.append(count)
values8=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='9':
      if list1['Day']==each:
      count=count+1
  values8.append(count)
values9=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='10':
      if list1['Day']==each:
      count=count+1
  values9.append(count)
values10=[]

for each in labels:
```

```
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='11':
      if list1['Day']==each:
        count=count+1
  values10.append(count)
values11=[]

for each in labels:
  count=0
  for i in range(0,21609):
    list1=file.iloc[i]
    if list1['Month']=='12':
      if list1['Day']==each:
        count=count+1
  values11.append(count)

from plotly.graph_objs import *
layout=Layout(boxgap=0.5)
fig=go.Figure(layout=layout)
fig.add_trace(go.Box(y=values,name="Jan"))
fig.add_trace(go.Box(y=values1,name="Feb"))
fig.add_trace(go.Box(y=values2,name="March"))
fig.add_trace(go.Box(y=values3,name="April"))
fig.add_trace(go.Box(y=values4,name="May"))
fig.add_trace(go.Box(y=values5,name="June"))
fig.add_trace(go.Box(y=values6,name="July"))
fig.add_trace(go.Box(y=values7,name="Aug"))
fig.add_trace(go.Box(y=values8,name="Sept"))
fig.add_trace(go.Box(y=values9,name="Oct"))
fig.add_trace(go.Box(y=values10,name="Nov"))
fig.add_trace(go.Box(y=values11,name="Dec"))
fig.update_traces(quartilemethod="inclusive")
fig.show()

labels = file.Day.unique()
values=[]
for each in labels:
   values.append(len(file[file.Day==each]))

from wordcloud import WordCloud
temp=file['Primary Type']
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,15))
wordcloud = WordCloud(
                      background_color='black',
                      width=1920,
                      height=1080
                     ).generate(" ".join(temp))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()

from operator import itemgetter
crime_list=file['Primary Type'].tolist()
crime_list=[x for x in crime_list[1:] if str(x)!='nan']
crime_list

crime_count = len(crime_list)
temp = set(crime_list)
unique_crime_list = list(temp)
unique_crime_count = len(unique_crime_list)
print("Before : "+str(crime_count))
print("After : "+str(unique_crime_count))
print("CRIMES IN CHICAGO :")
unique_crime_list

pie_chart_dict = {}
for i in unique_crime_list:
  pie_chart_dict[i] = 0
for x in crime_list:
  pie_chart_dict[x]+=1
N = 10
sorted_pie_chart_dict = dict(sorted(pie_chart_dict.items(), key =itemgetter(1), reverse =
True)[:N])
print("The top ten crimes are : ")
sorted_pie_chart_dict

N = 10
sorted_pie_chart_dict2 = dict(sorted(pie_chart_dict.items(), key =
itemgetter(1), reverse = False)[:N])
print("Dates with the least number of crimes: ")
sorted_pie_chart_dict2

pie_labels1=sorted_pie_chart_dict.keys()
pie_values1=sorted_pie_chart_dict.values()
```

```
fig=px.pie(values=pie_values1,names=pie_labels1,title="Top 10 crime cases Chicago")
fig.show()

pie_values2=sorted_pie_chart_dict2.values()
pie_labels2=sorted_pie_chart_dict2.keys()
def Merge(dict1, dict2):
    res = {**dict1, **dict2}
    return res

dict3=Merge(sorted_pie_chart_dict,sorted_pie_chart_dict2)
labell=np.array(dict3.keys())
valuee=dict3.values()
unique_date_list=file.Date.unique()
pie_chart_dict2 = {}
for i in unique_date_list:
  pie_chart_dict2[i] = 0
pie_chart_dict2

date_list = file['Date'].tolist()
date_list = [x for x in date_list[1:] if str(x)!='nan']
date_list

from operator import itemgetter
for x in date_list:
  pie_chart_dict2[x]+=1
N = 10
sorted_pie_chart_dict2 = dict(sorted(pie_chart_dict2.items(), key =
itemgetter(1), reverse = False)[:N])
print("Dates with the least number of crimes: ")
sorted_pie_chart_dict2

from operator import itemgetter
N = 10
reverse_sorted_pie_chart_dict2 = dict(sorted(pie_chart_dict2.items(), key
= itemgetter(1), reverse = True)                                    [:N])
print("Dates with the most number of crimes: ")
reverse_sorted_pie_chart_dict2

def Merge(dict1, dict2):
  result = {**dict1, **dict2}
  return result
final_dict = Merge(sorted_pie_chart_dict2, reverse_sorted_pie_chart_dict2)
print(final_dict)
dff=pd.DataFrame({"Year":final_dict.keys(),"Value":final_dict.values()})
```

```
dff

dff.drop(10,axis=0,inplace=True)
lab=dff['Year']
val=dff['Value']
layout=go.Layout(title='Top 10 and least 10 Crime date Chicago')
fig = go.Figure(data=[go.Pie(labels=lab, values=val, hole=.5)],layout=layout)
fig.show()
```

#Prophet Model (Chicago)

```
%pip install pystan==2.19.1.1
pip install fbprophet
file=df.toPandas()
file.drop(['_c0', 'ID', 'Case Number', 'Date_Time','Year', 'Month', 'Day',
      'Time', 'Hour', 'Minute', 'Block', 'IUCR','Description', 'Location Description', 'Arrest',
'Domestic', 'Beat',
      'District', 'Ward', 'Community Area', 'FBI Code', 'X Coordinate',
      'Y Coordinate', 'Year.1', 'Updated On', 'Latitude', 'Longitude',
      'Location'],axis=1,inplace=True)

file=file[file['Date']!="#VALUE!"]
file.head()

import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.Date).dt.year
file['month'] = pd.to_datetime(file.Date).dt.month
file['day'] = pd.to_datetime(file.Date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file.drop('Date',inplace=True,axis=1)
file = file.groupby('DATE').count()['Primary Type'].to_frame()
file.reset_index(inplace=True)
file.columns = ['ds','y']
file.head()

plt.plot(file['ds'],file['y'])
plt.xlabel('Year')
plt.ylabel('No.of crimes')
df_m1 = file.copy()
df_m2=file.copy()
df_m2.sort_values(by='ds',inplace=True)
df_m2['year'] = pd.to_datetime(df_m2.ds).dt.year
df_m2['month'] = pd.to_datetime(df_m2.ds).dt.month
```

```
df_m2['day'] = pd.to_datetime(df_m2.ds).dt.day
df_m2.year.unique()
dataframe1=df_m2[df_m2['year'] == 2010]
dataframe2=df_m2[df_m2['year'] == 2011]
dataframe3=df_m2[df_m2['year'] == 2012]
dataframe4=df_m2[df_m2['year'] == 2013]
dataframe5=df_m2[df_m2['year'] == 2014]
dataframe6=df_m2[df_m2['year'] == 2015]
dataframe7=df_m2[df_m2['year'] == 2016]
dataframe8=df_m2[df_m2['year'] == 2017]
dataframe9=df_m2[df_m2['year'] == 2018]
dataframe10=df_m2[df_m2['year'] == 2019]
dataframe11=df_m2[df_m2['year'] == 2020]
train1=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe6,dataframe7,da
taframe8,dataframe9,dataframe10]
train1=pd.concat(train1)
test1=[dataframe11]
test1=pd.concat(test1)
train1.drop(['year','month','day'],axis=1,inplace=True)
test1.drop(['year','month','day'],axis=1,inplace=True)


train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
train3.drop(['year','month','day'],axis=1,inplace=True)
test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
train4.drop(['year','month','day'],axis=1,inplace=True)
test4.drop(['year','month','day'],axis=1,inplace=True)

train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
train5.drop(['year','month','day'],axis=1,inplace=True)
test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
```

```
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
train6.drop(['year','month','day'],axis=1,inplace=True)
test6.drop(['year','month','day'],axis=1,inplace=True)

train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
train7.drop(['year','month','day'],axis=1,inplace=True)
test7.drop(['year','month','day'],axis=1,inplace=True)

import numpy as np
from fbprophet import Prophet
 holidays_0 = pd.DataFrame({
       'holiday': '0 window',
       'ds' :pd.to_datetime(
          ['2010-05-09','2011-05-08','2012-05-13','2013-05-12','2014-05-11','2015-05-10','2016-
05-08','2017-05-14','2018-05-13','2019-05-12','2020-05-10','2010-05-24','2011-05-23','2012-
05-21','2013-05-20','2014-05-19','2015-05-18','2016-05-23','2017-05-22','2018-05-21','2019-
05-20','2020-05-18','2010-07-01','2011-07-01','2012-07-01','2013-07-01','2014-07-01','2015-
07-01','2016-07-01','2017-07-01','2018-07-01','2019-07-01','2020-07-01','2010-09-06','2011-
09-05','2012-09-03','2013-09-02','2014-09-01','2015-09-07','2016-09-05','2017-09-04','2018-
09-03','2019-09-02','2020-09-07','2010-11-11','2011-11-11','2012-11-11','2013-11-11','2014-
11-11','2015-11-11','2016-11-11','2017-11-11','2018-11-11','2019-11-11','2020-11-11','2010-
12-25','2011-12-25','2012-12-25','2013-12-25','2014-12-25','2015-12-25','2016-12-25','2017-
12-25','2018-12-25','2019-12-25','2020-12-25']),
       'lower_window' : 0,
       'upper_window' : 0,
   })
holidays_1 = pd.DataFrame({
       'holiday': '1  window',
       'ds' :pd.to_datetime(
          ['2010-10-31','2011-10-31','2012-10-31','2013-10-31','2014-10-31','2015-10-31','2016-
10-31','2017-10-31','2018-10-31','2019-10-31','2020-10-31','2010-01-01','2011-01-01','2012-
01-01','2013-01-01','2014-01-01','2015-01-01','2016-01-01','2017-01-01','2018-01-01','2019-
01-01','2020-01-01']),
       'lower_window' : -1,
       'upper_window' : 1,
   })

holidays_2 = pd.DataFrame({
       'holiday': '2 window',
```

```
        'ds' :pd.to_datetime(
            ['2010-08-02','2011-08-01','2012-08-06','2013-08-05','2014-08-04','2015-08-03','2016-
08-01','2017-08-07','2018-08-06','2019-08-05','2020-08-03','2010-10-11','2011-10-10','2012-
10-08','2013-10-14','2014-10-13','2015-10-12','2016-10-10','2017-10-09','2018-10-08','2019-
10-14','2020-10-12']),
        'lower_window' : -2,
        'upper_window' : 1,
    })
holidays_list = pd.concat((holidays_0, holidays_1, holidays_2))

train1['y'] = np.log(train1['y'])
m1_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m1_plain.add_country_holidays(country_name='US')
m1_plain.fit(train1)
future = m1_plain.make_future_dataframe(periods=365)
forecast_m1 = m1_plain.predict(future)
m1_plain.plot(forecast_m1)
m1_plain.plot_components(forecast_m1)

y1 = test1['y'].to_frame()
y1.index = test1['ds']
forecast_m1_exp = np.exp(forecast_m1[['yhat','yhat_lower','yhat_upper']])
forecast_m1_exp.index = forecast_m1['ds']
se = np.square(forecast_m1_exp.loc[:, 'yhat'] - y1['y'])
mse = np.mean(se)
rmse1 = np.sqrt(mse)
print(rmse1)

train3['y']=np.log(train3['y'])
m2_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m2_plain.add_country_holidays(country_name='US')
m2_plain.fit(train3)
future = m2_plain.make_future_dataframe(periods=365)
forecast_m2 = m2_plain.predict(future)
m2_plain.plot(forecast_m2)
m2_plain.plot_components(forecast_m2)
y2 = test3['y'].to_frame()
y2.index = test3['ds']
forecast_m2_exp = np.exp(forecast_m2[['yhat','yhat_lower','yhat_upper']])
forecast_m2_exp.index = forecast_m2['ds']
se = np.square(forecast_m2_exp.loc[:, 'yhat'] - y2['y'])
mse = np.mean(se)
```

```
rmse2 = np.sqrt(mse)
print(rmse2)

train4['y']=np.log(train4['y'])
m3_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m3_plain.add_country_holidays(country_name='US')
m3_plain.fit(train4)
future = m3_plain.make_future_dataframe(periods=365)
forecast_m3 = m3_plain.predict(future)
m3_plain.plot(forecast_m3)
y3 = test4['y'].to_frame()
y3.index = test4['ds']
forecast_m3_exp = np.exp(forecast_m3[['yhat','yhat_lower','yhat_upper']])
forecast_m3_exp.index = forecast_m3['ds']
se = np.square(forecast_m3_exp.loc[:, 'yhat'] - y3['y'])
mse = np.mean(se)
rmse3 = np.sqrt(mse)
print(rmse3)

train5['y']=np.log(train5['y'])
m4_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m4_plain.add_country_holidays(country_name='US')
m4_plain.fit(train5)
future = m4_plain.make_future_dataframe(periods=365)
forecast_m4 = m4_plain.predict(future)
m4_plain.plot(forecast_m4)
m4_plain.plot_components(forecast_m4)
y4 = test5['y'].to_frame()
y4.index = test5['ds']
forecast_m4_exp = np.exp(forecast_m4[['yhat','yhat_lower','yhat_upper']])
forecast_m4_exp.index = forecast_m4['ds']
se = np.square(forecast_m4_exp.loc[:, 'yhat'] - y4['y'])
mse = np.mean(se)
rmse4 = np.sqrt(mse)
print(rmse4)

train6['y']=np.log(train6['y'])
m5_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m5_plain.add_country_holidays(country_name='US')
m5_plain.fit(train6)
future = m5_plain.make_future_dataframe(periods=365)
```

```
forecast_m5 = m5_plain.predict(future)
m5_plain.plot(forecast_m5)
m5_plain.plot_components(forecast_m5)
y5 = test6['y'].to_frame()
y5.index = test6['ds']
forecast_m5_exp = np.exp(forecast_m5[['yhat','yhat_lower','yhat_upper']])
forecast_m5_exp.index = forecast_m5['ds']
se = np.square(forecast_m5_exp.loc[:, 'yhat'] - y5['y'])
mse = np.mean(se)
rmse5 = np.sqrt(mse)
print(rmse5)

train7['y']=np.log(train7['y'])
m6_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m6_plain.add_country_holidays(country_name='US')
m6_plain.fit(train7)
future = m6_plain.make_future_dataframe(periods=365)
forecast_m6 = m6_plain.predict(future)
m6_plain.plot(forecast_m6)
m6_plain.plot_components(forecast_m6)
y6 = test7['y'].to_frame()
y6.index = test7['ds']
forecast_m6_exp = np.exp(forecast_m6[['yhat','yhat_lower','yhat_upper']])
forecast_m6_exp.index = forecast_m6['ds']
se = np.square(forecast_m6_exp.loc[:, 'yhat'] - y6['y'])
mse = np.mean(se)
rmse6 = np.sqrt(mse)
print(rmse6)

from scipy.stats import spearmanr
file_copy6=forecast_m6_exp.copy()
file_copy6.reset_index(inplace=True)
file_copy6['year']=pd.to_datetime(file_copy6.ds).dt.year
data1=file_copy6[file_copy6['year']==2011]
data2=test7['y']
data1=data1['yhat'][:31]
coef6, p = spearmanr(data1, data2)
file_copy5=forecast_m5_exp.copy()
file_copy5.reset_index(inplace=True)
file_copy5['year']=pd.to_datetime(file_copy5.ds).dt.year
data1=file_copy5[file_copy5['year']==2012]
data1=data1['yhat'][:37]
data2=test6['y']
```

```
coef5, p = spearmanr(data1, data2)
print(coef5)

file_copy4=forecast_m4_exp.copy()
file_copy4.reset_index(inplace=True)
file_copy4['year']=pd.to_datetime(file_copy4.ds).dt.year
data1=file_copy4[file_copy4['year']==2013]
data1=data1['yhat'][:45]
data2=test5['y']
coef4, p = spearmanr(data1, data2)
print(coef4)

file_copy3=forecast_m3_exp.copy()
file_copy3.reset_index(inplace=True)
file_copy3['year']=pd.to_datetime(file_copy3.ds).dt.year
data1=file_copy3[file_copy3['year']==2014]
data1=data1['yhat'][:87]
data2=test4['y']
coef3, p = spearmanr(data1, data2)
print(coef3)

file_copy2=forecast_m2_exp.copy()
file_copy2.reset_index(inplace=True)
file_copy2['year']=pd.to_datetime(file_copy2.ds).dt.year
data1=file_copy2[file_copy2['year']==2015]
data1=data1['yhat'][:142]
data2=test3['y']
coef2, p = spearmanr(data1, data2)
print(coef2)

file_copy1=forecast_m1_exp.copy()
file_copy1.reset_index(inplace=True)
file_copy1['year']=pd.to_datetime(file_copy1.ds).dt.year
data1=file_copy1[file_copy1['year']==2020]
data1=data1['yhat'][:76]
data2=test1['y']
coef1, p = spearmanr(data1, data2)
print(coef1)

data=[["Chicago",'10',rmse1,coef1],["Chicago",'5',rmse2,coef2],["Chicago",'4',rmse3,coef3],[
"Chicago",'3',rmse4,coef4],["Chicago",'2',rmse5,coef5],["Chicago",'1',rmse6,coef6]]
result=pd.DataFrame(data,columns=['City','Years of training','RMSE value','Spearman
Coeff'])
result
```

#Neural Networks (Chicago)

```
pip install keras
pip install tensorflow

from sklearn.metrics import accuracy_score,mean_squared_error
file=df.toPandas()

import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.Date).dt.year
file['month'] = pd.to_datetime(file.Date).dt.month
file['day'] = pd.to_datetime(file.Date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file.drop('Date',inplace=True,axis=1)
file = file.groupby('DATE').count()['Primary Type'].to_frame()
file.reset_index(inplace=True)
file['year'] = pd.to_datetime(file.DATE).dt.year
file['month'] = pd.to_datetime(file.DATE).dt.month
file['day'] = pd.to_datetime(file.DATE).dt.day
file.drop(['DATE'],inplace=True,axis=1)
file.columns=['Category','year','month','day']
df_m2=file.copy()
dataframe1=df_m2[df_m2['year'] == 2010]
dataframe2=df_m2[df_m2['year'] == 2011]
dataframe3=df_m2[df_m2['year'] == 2012]
dataframe4=df_m2[df_m2['year'] == 2013]
dataframe5=df_m2[df_m2['year'] == 2014]
dataframe6=df_m2[df_m2['year'] == 2015]
dataframe7=df_m2[df_m2['year'] == 2016]
dataframe8=df_m2[df_m2['year'] == 2017]
dataframe9=df_m2[df_m2['year'] == 2018]
dataframe10=df_m2[df_m2['year'] == 2019]
dataframe11=df_m2[df_m2['year'] == 2020]
train1=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe6,dataframe7,da
taframe8,dataframe9,dataframe10]
train1=pd.concat(train1)
test1=[dataframe11]
test1=pd.concat(test1)
train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
```

```
train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)

import keras
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import math
from scipy.stats import spearmanr
trainX=train1[['year','month','day']]
trainY=train1[['Category']]
testX=test1[['year','month','day']]
testY=test1[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY, epochs=300)

y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy1 = mean_squared_error(y_pred,testY)
accuracy1=math.sqrt(accuracy1)
print(accuracy1)

coef1, p = spearmanr(y_pred,testY)
print(coef1)
```

```
trainX=train3[['year','month','day']]
trainY=train3[['Category']]
testX=test3[['year','month','day']]
testY=test3[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY, epochs=300)

y_pred=model.predict(testX)
u=0
for i in y_pred:
  y_pred[u]=i*150
  u=u+1
accuracy2 = mean_squared_error(y_pred,testY)
accuracy2=math.sqrt(accuracy2)
print(accuracy2)

coef2, p = spearmanr(y_pred,testY)
print(coef2)

trainX=train4[['year','month','day']]
trainY=train4[['Category']]
testX=test4[['year','month','day']]
testY=test4[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)

y_pred=model.predict(testX)
accuracy3 = mean_squared_error(y_pred,testY)
accuracy3=math.sqrt(accuracy3)
print(accuracy3)

coef3, p = spearmanr(y_pred,testY)
```

```
print(coef3)

trainX=train5[['year','month','day']]
trainY=train5[['Category']]
testX=test5[['year','month','day']]
testY=test5[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)

y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy4 = mean_squared_error(y_pred,testY)
accuracy4=math.sqrt(accuracy4)
print(accuracy4)

coef4, p = spearmanr(y_pred,testY)
print(coef4)

trainX=train6[['year','month','day']]
trainY=train6[['Category']]
testX=test6[['year','month','day']]
testY=test6[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)

y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy5 = mean_squared_error(y_pred,testY)
accuracy5=math.sqrt(accuracy5)
print(accuracy5)

coef5, p = spearmanr(y_pred,testY)
print(coef5)
```

```
trainX=train7[['year','month','day']]
trainY=train7[['Category']]
testX=test7[['year','month','day']]
testY=test7[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)

y_pred=model.predict(testX)
#y_pred = np.asarray(y_pred).astype(np.int64)
#testY=testY.to_numpy()
accuracy6 = mean_squared_error(y_pred,testY)
accuracy6=math.sqrt(accuracy6)
print(accuracy6)

coef6, p = spearmanr(y_pred,testY)
print(coef6)

data=[["Chicago",'10',accuracy1,coef1],["Chicago",'5',accuracy2,coef2],["Chicago",'4',accura
cy3,coef3],["Chicago",'3',accuracy4,coef4],["Chicago",'2',accuracy5,coef5]]
result=pd.DataFrame(data,columns=['City','Years of training','RMSE value','Spearman
Coeff'])
result
```

#Artificial Neural Networks (Philadelphia)

```
pip install keras
pip install tensorflow

file = df.toPandas()
file.head()

new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [3, 10, 19, 20, 11]]
file1.head()
```

```
file1['Day'] = file1['Day'].factorize()[0]
file1.head()

file1.dropna(inplace = True)
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, confusion_matrix,
classification_report, accuracy_score, f1_score
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['text_general_code']]).size().sort_values(ascending=True).plot(kind='bar
h')
plt.show()

all_classes = file1.groupby(['text_general_code'])['dc_dist'].size().reset_index()
all_classes['Amt'] = all_classes['dc_dist']
all_classes = all_classes.drop(['dc_dist'], axis=1)
all_classes = all_classes.sort_values(['Amt'], ascending=[False])
unwanted_classes = all_classes.tail(13)
unwanted_classes

file1.loc[file1['text_general_code'].isin(unwanted_classes['text_general_code']),
'text_general_code'] = 'OTHERS'
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['text_general_code']]).size().sort_values(ascending=True).plot(kind='bar
h')
plt.show()

Classes = file1['text_general_code'].unique()
Classes
```

```
file1['text_general_code'] = pd.factorize(file1["text_general_code"])[0]
file1['text_general_code'].unique()
x, y = train_test_split(file1,
                        test_size = 0.2,
                        train_size = 0.8,
                        random_state= 3)
Features = ['dc_dist', 'Day', 'lat', 'lng']
Target = ['text_general_code']
x1 = x[Features]
x2 = x[Target]
y1 = y[Features]
y2 = y[Target]
print('Feature Set Used    : ', Features)
print('Target Class        : ', Target)
print('Training Set Size : ', x.shape)
print('Test Set Size       : ', y.shape)

nn_model = MLPClassifier(solver='adam',
                         alpha=1e-5,
                         hidden_layer_sizes=(40,),
                         random_state=1,
                         max_iter=1000
                        )
nn_model.fit(X=x1,
             y=x2)
result = nn_model.predict(y[Features])

from sklearn.metrics import mean_absolute_error
mean_absolute_error(y2, result)
```

#Artificial Neural Networks (Chicago)

```
file = df.toPandas()
file.head()

new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [7, 19, 27, 28, 13]]
file1.head()

file1['Day'] = file1['Day'].factorize()[0]
file1.dropna(inplace = True)
```

```
file1.head()

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, confusion_matrix, accuracy_score,
f1_score
from sklearn.neural_network import MLPClassifier
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics

plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')
plt.show()

all_classes = file1.groupby(['Primary Type'])['District'].size().reset_index()
all_classes['Amt'] = all_classes['District']
all_classes = all_classes.drop(['District'], axis=1)
all_classes = all_classes.sort_values(['Amt'], ascending=[False])
unwanted_classes = all_classes.tail(13)
unwanted_classes

file1.loc[file1['Primary Type'].isin(unwanted_classes['Primary Type']), 'Primary Type'] =
'OTHERS'
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')
plt.show()

Classes = file1['Primary Type'].unique()
Classes

file1['Primary Type'] = pd.factorize(file1["Primary Type"])[0]
file1['Primary Type'].unique()
file1.head()

x, y = train_test_split(file1,
```

```
                    test_size = 0.2,
                    train_size = 0.8,
                    random_state= 3)
Features = ['District', 'Day', 'Latitude', 'Longitude']
Target = ['Primary Type']
x1 = x[Features]
x2 = x[Target]
y1 = y[Features]
y2 = y[Target]
print('Feature Set Used    : ', Features)
print('Target Class        : ', Target)
print('Training Set Size : ', x.shape)
print('Test Set Size       : ', y.shape)

nn_model = MLPClassifier(solver='adam',
                     alpha=1e-5,
                     hidden_layer_sizes=(40,),
                     random_state=1,
                     max_iter=1000
                    )
nn_model.fit(X=x1,
           y=x2)
result = nn_model.predict(y[Features])

from sklearn.metrics import mean_absolute_error
mean_absolute_error(y2, result)
```

#Artifical Neural Networks (San Francisco)

```
file = df.toPandas()
file.head()
new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [5, 11, 12, 15, 3]]
file1.head()

file1['Day'] = file1['Day'].factorize()[0]
file1.dropna(inplace = True)
file1.head()

import matplotlib
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, confusion_matrix, accuracy_score,
f1_score
from sklearn.neural_network import MLPClassifier
from sklearn import metrics

plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')
plt.show()

all_classes = file1.groupby(['Primary Type'])['Current Police Districts 2
2'].size().reset_index()
all_classes['Amt'] = all_classes['Current Police Districts 2 2']
all_classes = all_classes.drop(['Current Police Districts 2 2'], axis=1)
all_classes = all_classes.sort_values(['Amt'], ascending=[False])
unwanted_classes = all_classes.tail(13)
unwanted_classes

file1.loc[file1['Primary Type'].isin(unwanted_classes['Primary Type']), 'Primary Type'] =
'OTHERS'
plt.figure(figsize=(14,10))
plt.title('Amount of Crimes by Primary Type')
plt.ylabel('Crime Type')
plt.xlabel('Amount of Crimes')
file1.groupby([file1['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')
plt.show()

Classes = file1['Primary Type'].unique()
Classes

file1['Primary Type'] = pd.factorize(file1["Primary Type"])[0]
file1['Primary Type'].unique()
file1.head()

x, y = train_test_split(file1,
                        test_size = 0.2,
                        train_size = 0.8,
                        random_state= 3)
Features = ['Day', 'Latitude', 'Longitude', 'Current Police Districts 2 2']
```

```
Target = ['Primary Type']

x1 = x[Features]
x2 = x[Target]
y1 = y[Features]
y2 = y[Target]
print('Feature Set Used    : ', Features)
print('Target Class        : ', Target)
print('Training Set Size : ', x.shape)
print('Test Set Size       : ', y.shape)

nn_model = MLPClassifier(solver='adam',
                         alpha=1e-5,
                         hidden_layer_sizes=(40,),
                         random_state=1,
                         max_iter=1000
                        )
nn_model.fit(X=x1,
             y=x2)
result = nn_model.predict(y[Features])

from sklearn.metrics import mean_absolute_error
mean_absolute_error(y2, result)
```

#RandomForest (Philadelphia)

```
file = df.toPandas()
file.head()
new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [3, 10, 19, 20, 11]]
file1

file1.dropna(inplace = True)

import numpy as np
file1['dc_dist'] = file1['dc_dist'].values.astype(np.float)
file1['lat'] = file1['lat'].values.astype(np.float)
file1['lng'] = file1['lng'].values.astype(np.float)
file1['Category_Id'] = file1['text_general_code'].factorize()[0]
```

```
Category_Id_df = file1[['text_general_code',
'Category_Id']].drop_duplicates().sort_values('Category_Id')
Category_To_Id = dict(Category_Id_df.values)
Id_to_Category = dict(Category_Id_df[['Category_Id', 'text_general_code']].values)
file1.head()

file1 = file1.drop('text_general_code', axis = 1)

import pandas as pd
file1['Category_Id'] = pd.to_numeric(file1['Category_Id'])
file1 = pd.get_dummies(file1)
file1.head()
labels = file1['Category_Id']
file1 = file1.drop('Category_Id', axis = 1)
feature_list = list(file1.columns)

import numpy as np
file1 = np.array(file1)
labels = np.array(labels)

from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(file1, labels, test_size
= 0.25, random_state = 42)
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators= 1000, random_state=42)
original_feature_indices = [feature_list.index(feature) for feature in feature_list]
original_test_features = test_features[:, original_feature_indices]
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
errors = abs(predictions - test_labels)
print('Metrics for Random Forest Trained on Original Data')
print('Average absolute error:', round(np.mean(errors), 2), 'degrees.')
mape = 100 * (errors / test_labels)

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000, random_state=42)
original_feature_indices = [feature_list.index(feature) for feature in feature_list]
original_test_features = test_features[:, original_feature_indices]
```

```
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
errors = abs(predictions - test_labels)
print('Metrics for Random Forest Trained on Original Data')
print('Average absolute error:', round(np.mean(errors), 2))
```

#RandomForest (Chicago)

```
file = df.toPandas()
file.head()

new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [7, 13, 19, 27, 28]]
file1

file1.dropna(inplace = True)

import numpy as np
file1['District'] = file1['District'].values.astype(np.float)
file1['Latitude'] = file1['Latitude'].values.astype(np.float)
file1['Longitude'] = file1['Longitude'].values.astype(np.float)
file1['Category_Id'] = file1['Primary Type'].factorize()[0]
Category_Id_df = file1[['Primary Type',
'Category_Id']].drop_duplicates().sort_values('Category_Id')
Category_To_Id = dict(Category_Id_df.values)
Id_to_Category = dict(Category_Id_df[['Category_Id', 'Primary Type']].values)
file1.head()

file1 = file1.drop('Primary Type', axis = 1)

import pandas as pd
file1['Category_Id'] = pd.to_numeric(file1['Category_Id'])
file1 = pd.get_dummies(file1)
file1.head()

labels = file1['Category_Id']
file1 = file1.drop('Category_Id', axis = 1)
feature_list = list(file1.columns)

import numpy as np
file1 = np.array(file1)
```

```
labels = np.array(labels)

from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(file1, labels, test_size
= 0.25, random_state = 42)

df = pd.DataFrame(test_features, columns = ['dc_dist','lat','lng', 'Category_ID', 'Day_Friday',
        'Day_Monday',        'Day_Saturday',        'Day_Sunday', 'Day_Thursday',
'Day_Monday', 'Day_Tuesday'])
df.head(10)

print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000, random_state=42)
original_feature_indices = [feature_list.index(feature) for feature in feature_list]
original_test_features = test_features[:, original_feature_indices]
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
errors = abs(predictions - test_labels)
print('Metrics for Random Forest Trained on Original Data')
print('Average absolute error:', round(np.mean(errors), 2))
```

#RandomForest (SanFrancisco)

```
file = df.toPandas()
file.head()

new_header = file.iloc[0]
file = file[1:]
file.columns = new_header
file.head()

file1 = file.iloc[1:100000, [3, 5, 11, 12, 15]]
file1

file1.dropna(inplace = True)

import numpy as np
file1['Current Police Districts 2 2'] = file1['Current Police Districts 2
2'].values.astype(np.float)
file1['Latitude'] = file1['Latitude'].values.astype(np.float)
```

```
file1['Longitude'] = file1['Longitude'].values.astype(np.float)
file1['Category_Id'] = file1['Primary Type'].factorize()[0]
Category_Id_df = file1[['Primary Type',
'Category_Id']].drop_duplicates().sort_values('Category_Id')
Category_To_Id = dict(Category_Id_df.values)
Id_to_Category = dict(Category_Id_df[['Category_Id', 'Primary Type']].values)
file1.head()

file1 = file1.drop('Primary Type', axis = 1)

import pandas as pd
file1['Category_Id'] = pd.to_numeric(file1['Category_Id'])
file1 = pd.get_dummies(file1)
file1.head()

labels = file1['Category_Id']
file1 = file1.drop('Category_Id', axis = 1)
feature_list = list(file1.columns)

import numpy as np
file1 = np.array(file1)
labels = np.array(labels)

from sklearn.model_selection import train_test_split
train_features, test_features, train_labels, test_labels = train_test_split(file1, labels, test_size
= 0.25, random_state = 42)

print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shap

df = pd.DataFrame(test_features, columns = ['dc_dist','lat','lng', 'Category_ID', 'Day_Friday',
        'Day_Monday',        'Day_Saturday',        'Day_Sunday', 'Day_Thursday',
        'Day_Tuesday'])
df.head(10)

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000, random_state=42)
original_feature_indices = [feature_list.index(feature) for feature in feature_list]
original_test_features = test_features[:, original_feature_indices]
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
errors = abs(predictions - test_labels)
```

```
print('Metrics for Random Forest Trained on Original Data')
print('Average absolute error:', round(np.mean(errors), 2))

#KNN Classification
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/2010
-1.csv",inferSchema=True,header=True)
file1=df1.toPandas()
file = file1.iloc[:, [1,6,10,13,14,15]]
file.head()
file.columns = ['dist','hour','crime', 'lat','long','day']
file=file[file['crime']!='All Other Offenses']
file=file[file['crime']!='Other Assaults']
file=file[file['crime']!='Homicide - Criminal']
file=file[file['crime']!='Homicide - Criminal']
file=file[file['crime']!='Homicide - Justifiable ']
file=file[file['crime']!='Gambling Violations']
file=file[file['crime']!='Arson']
file=file[file['crime']!='Receiving Stolen Property']
file=file[file['crime']!='Offenses Against Family and Children']
col = ['dist','hour','crime', 'lat','long','day']
file = file[col]
file.dropna(inplace=True)
#file = file[pd.notnull(file['long'])]
#file = file[pd.notnull(file['lat'])]
#file.columns = ['dist','crime', 'long','lat']
file['Category_Id'] = file['crime'].factorize()[0]
file['Day_Id']=file['day'].factorize()[0]
#file.drop(['day'],inplace=True,axis=1)
Category_Id_df = file[['crime', 'Category_Id']].drop_duplicates().sort_values('Category_Id')
Day_Id_df=file[['day','Day_Id']].drop_duplicates().sort_values('Day_Id')
Category_To_Id = dict(Category_Id_df.values)
Day_To_Id=dict(Day_Id_df.values)
Id_to_Category = dict(Category_Id_df[['Category_Id', 'crime']].values)
Id_to_Day=dict(Day_Id_df[['Day_Id','day']].values)
file.head()
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8,6))
file.groupby('crime').dist.count().plot.bar(ylim=0)
plt.show()
```

```
X[['dist','hour','lat', 'long', 'Day_Id']]=file[['dist','hour', 'lat', 'long', 'Day_Id']]
Y[['label']]=file[['Category_Id']]
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state = 25)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=200)
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
scaler = MinMaxScaler()
features = [['dist','hour','lat', 'long', 'Day_Id']]
for feature in features:
    X[feature] = scaler.fit_transform(X[feature])
knn.fit(X_train, Y_train)
y_pred3 = knn.predict(X_test)
list1=[]
for i in y_pred3:
  list1.append(Id_to_Category.get(i))
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_pred3,Y_test))
```

#Gaussian NB(CONTINUATION)

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train,Y_train)
y_pred2=clf.predict(X_test)
print(mean_absolute_error(y_pred2,Y_test))
list1=[]
for i in y_pred2:
  list1.append(Id_to_Category.get(i))
```

#Prophet model(Philly)

```
%pip install pystan==2.19.1.1
pip install fbprophet
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/phila
_final-2.csv",inferSchema=True,header=True)
file=df1.toPandas()
file.drop(['_c0', 'objectid', 'dc_dist', 'psa','dispatch_date_time','dispatch_time', 'hour_',
'dc_key', 'location_block',
      'ucr_general','point_x', 'point_y', 'lat', 'lng','Unnamed: 0'],axis=1,inplace=True)
import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.dispatch_date).dt.year
file['month'] = pd.to_datetime(file.dispatch_date).dt.month
file['day'] = pd.to_datetime(file.dispatch_date).dt.day
```

```
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file.drop('dispatch_date',inplace=True,axis=1)
file = file.groupby('DATE').count()['text_general_code'].to_frame()
file.reset_index(inplace=True)
file.columns = ['ds','y']
file.head()
df_m1 = file.copy()
df_m2=file.copy()
df_m2.sort_values(by='ds',inplace=True)
df_m2['year'] = pd.to_datetime(df_m2.ds).dt.year
df_m2['month'] = pd.to_datetime(df_m2.ds).dt.month
df_m2['day'] = pd.to_datetime(df_m2.ds).dt.day
dataframe1=df_m2[df_m2['year'] == 2010]
dataframe2=df_m2[df_m2['year'] == 2011]
dataframe3=df_m2[df_m2['year'] == 2012]
dataframe4=df_m2[df_m2['year'] == 2013]
dataframe5=df_m2[df_m2['year'] == 2014]
dataframe6=df_m2[df_m2['year'] == 2015]
dataframe7=df_m2[df_m2['year'] == 2016]
dataframe8=df_m2[df_m2['year'] == 2017]
dataframe9=df_m2[df_m2['year'] == 2018]
dataframe10=df_m2[df_m2['year'] == 2019]
dataframe11=df_m2[df_m2['year'] == 2020]
train1=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe6,dataframe7,da
taframe8,dataframe9,dataframe10]
train1=pd.concat(train1)
test1=[dataframe11]
test1=pd.concat(test1)
train1.drop(['year','month','day'],axis=1,inplace=True)
test1.drop(['year','month','day'],axis=1,inplace=True)
train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
train3.drop(['year','month','day'],axis=1,inplace=True)
test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
train4.drop(['year','month','day'],axis=1,inplace=True)
test4.drop(['year','month','day'],axis=1,inplace=True)
```

```
train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
train5.drop(['year','month','day'],axis=1,inplace=True)
test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
train6.drop(['year','month','day'],axis=1,inplace=True)
test6.drop(['year','month','day'],axis=1,inplace=True)

train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
train7.drop(['year','month','day'],axis=1,inplace=True)
test7.drop(['year','month','day'],axis=1,inplace=True)
import numpy as np
from fbprophet import Prophet
 holidays_0 = pd.DataFrame({
       'holiday': '0 window',
       'ds' :pd.to_datetime(
          ['2010-05-09','2011-05-08','2012-05-13','2013-05-12','2014-05-11','2015-05-10','2016-
05-08','2017-05-14','2018-05-13','2019-05-12','2020-05-10','2010-05-24','2011-05-23','2012-
05-21','2013-05-20','2014-05-19','2015-05-18','2016-05-23','2017-05-22','2018-05-21','2019-
05-20','2020-05-18','2010-07-01','2011-07-01','2012-07-01','2013-07-01','2014-07-01','2015-
07-01','2016-07-01','2017-07-01','2018-07-01','2019-07-01','2020-07-01','2010-09-06','2011-
09-05','2012-09-03','2013-09-02','2014-09-01','2015-09-07','2016-09-05','2017-09-04','2018-
09-03','2019-09-02','2020-09-07','2010-11-11','2011-11-11','2012-11-11','2013-11-11','2014-
11-11','2015-11-11','2016-11-11','2017-11-11','2018-11-11','2019-11-11','2020-11-11','2010-
12-25','2011-12-25','2012-12-25','2013-12-25','2014-12-25','2015-12-25','2016-12-25','2017-
12-25','2018-12-25','2019-12-25','2020-12-25']),
       'lower_window' : 0,
       'upper_window' : 0,
   })
holidays_1 = pd.DataFrame({
       'holiday': '1  window',
       'ds' :pd.to_datetime(
          ['2010-10-31','2011-10-31','2012-10-31','2013-10-31','2014-10-31','2015-10-31','2016-
10-31','2017-10-31','2018-10-31','2019-10-31','2020-10-31','2010-01-01','2011-01-01','2012-
```

```
01-01','2013-01-01','2014-01-01','2015-01-01','2016-01-01','2017-01-01','2018-01-01','2019-
01-01','2020-01-01']),
      'lower_window' : -1,
      'upper_window' : 1,
   })

holidays_2 = pd.DataFrame({
      'holiday': '2  window',
      'ds' :pd.to_datetime(
         ['2010-08-02','2011-08-01','2012-08-06','2013-08-05','2014-08-04','2015-08-03','2016-
08-01','2017-08-07','2018-08-06','2019-08-05','2020-08-03','2010-10-11','2011-10-10','2012-
10-08','2013-10-14','2014-10-13','2015-10-12','2016-10-10','2017-10-09','2018-10-08','2019-
10-14','2020-10-12']),
      'lower_window' : -2,
      'upper_window' : 1,
   })
holidays_list = pd.concat((holidays_0, holidays_1, holidays_2))
from fbprophet.plot import add_changepoints_to_plot
train1['y'] = np.log(train1['y'])
m1_plain = Prophet(yearly_seasonality=True,changepoint_range=0.4,
weekly_seasonality=True, holidays=holidays_list)
m1_plain.add_country_holidays(country_name='US')
m1_plain.fit(train1)

# Let's try a forecast for 1 year
future = m1_plain.make_future_dataframe(periods=366)
forecast_m1 = m1_plain.predict(future)
m1_plain.plot(forecast_m1)
m1_plain.plot_components(forecast_m1)
y1 = test1['y'].to_frame()
y1.index = test1['ds']
forecast_m1_exp = np.exp(forecast_m1[['yhat','yhat_lower','yhat_upper']])
forecast_m1_exp.index = forecast_m1['ds']
se = np.square(forecast_m1_exp.loc[:, 'yhat'] - y1['y'])
mse = np.mean(se)
rmse1 = np.sqrt(mse)
print(rmse1)
dummy=train3.copy()

train3['y']=np.log(train3['y'])
m2_plain = Prophet(yearly_seasonality=True,changepoint_range=0.5,
weekly_seasonality=True, holidays=holidays_list1)
m2_plain.add_country_holidays(country_name='US')
m2_plain.fit(train3)
```

```
# Let's try a forecast for 1 year
future = m2_plain.make_future_dataframe(periods=365)
forecast_m2 = m2_plain.predict(future)
fig=m2_plain.plot(forecast_m2)
a=add_changepoints_to_plot(fig.gca(),m2_plain,forecast_m2)
m2_plain.plot_components(forecast_m2)
y2 = test3['y'].to_frame()
y2.index = test3['ds']
forecast_m2_exp = np.exp(forecast_m2[['yhat','yhat_lower','yhat_upper']])
forecast_m2_exp.index = forecast_m2['ds']
se = np.square(forecast_m2_exp.loc[:, 'yhat'] - y2['y'])
mse = np.mean(se)
rmse2 = np.sqrt(mse)
print(rmse2)
train4['y']=np.log(train4['y'])
m3_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m3_plain.add_country_holidays(country_name='US')
m3_plain.fit(train4)

# Let's try a forecast for 1 year
future = m3_plain.make_future_dataframe(periods=365)
forecast_m3 = m3_plain.predict(future)
m3_plain.plot(forecast_m3)
y3 = test4['y'].to_frame()
y3.index = test4['ds']
forecast_m3_exp = np.exp(forecast_m3[['yhat','yhat_lower','yhat_upper']])
forecast_m3_exp.index = forecast_m3['ds']
se = np.square(forecast_m3_exp.loc[:, 'yhat'] - y3['y'])
mse = np.mean(se)
rmse3 = np.sqrt(mse)
print(rmse3)
train5['y']=np.log(train5['y'])
m4_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m4_plain.add_country_holidays(country_name='US')
m4_plain.fit(train5)

# Let's try a forecast for 1 year
future = m4_plain.make_future_dataframe(periods=365)
forecast_m4 = m4_plain.predict(future)
y4 = test5['y'].to_frame()
y4.index = test5['ds']
```

```
forecast_m4_exp = np.exp(forecast_m4[['yhat','yhat_lower','yhat_upper']])
forecast_m4_exp.index = forecast_m4['ds']
se = np.square(forecast_m4_exp.loc[:, 'yhat'] - y4['y'])
mse = np.mean(se)
rmse4 = np.sqrt(mse)
print(rmse4)
train6['y']=np.log(train6['y'])
m5_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m5_plain.add_country_holidays(country_name='US')
m5_plain.fit(train6)

# Let's try a forecast for 1 year
future = m5_plain.make_future_dataframe(periods=365)
forecast_m5 = m5_plain.predict(future)
y5 = test6['y'].to_frame()
y5.index = test6['ds']
forecast_m5_exp = np.exp(forecast_m5[['yhat','yhat_lower','yhat_upper']])
forecast_m5_exp.index = forecast_m5['ds']
se = np.square(forecast_m5_exp.loc[:, 'yhat'] - y5['y'])
mse = np.mean(se)
rmse5 = np.sqrt(mse)
print(rmse5)
train7['y']=np.log(train7['y'])
m6_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m6_plain.add_country_holidays(country_name='US')
m6_plain.fit(train7)

# Let's try a forecast for 1 year
future = m6_plain.make_future_dataframe(periods=365)
forecast_m6 = m6_plain.predict(future)
y6 = test7['y'].to_frame()
y6.index = test7['ds']
forecast_m6_exp = np.exp(forecast_m6[['yhat','yhat_lower','yhat_upper']])
forecast_m6_exp.index = forecast_m6['ds']
se = np.square(forecast_m6_exp.loc[:, 'yhat'] - y6['y'])
mse = np.mean(se)
rmse6 = np.sqrt(mse)
print(rmse6)
from scipy.stats import spearmanr
file_copy6=forecast_m6_exp.copy()
file_copy6.reset_index(inplace=True)
file_copy6['year']=pd.to_datetime(file_copy6.ds).dt.year
```

```
data1=file_copy6[file_copy6['year']==2011]
data2=test7['y']
data1=data1['yhat']
coef6, p = spearmanr(data1, data2)
file_copy5=forecast_m5_exp.copy()
file_copy5.reset_index(inplace=True)
file_copy5['year']=pd.to_datetime(file_copy5.ds).dt.year
data1=file_copy5[file_copy5['year']==2012]
data1=data1['yhat']
data2=test6['y'][:-1]
coef5, p = spearmanr(data1, data2)
print(coef5)
file_copy4=forecast_m4_exp.copy()
file_copy4.reset_index(inplace=True)
file_copy4['year']=pd.to_datetime(file_copy4.ds).dt.year
data1=file_copy4[file_copy4['year']==2013]
data1=data1['yhat']
data2=test5['y']
coef4, p = spearmanr(data1, data2)
print(coef4)
file_copy3=forecast_m3_exp.copy()
file_copy3.reset_index(inplace=True)
file_copy3['year']=pd.to_datetime(file_copy3.ds).dt.year
data1=file_copy3[file_copy3['year']==2014]
data1=data1['yhat']
data2=test4['y']
coef3, p = spearmanr(data1, data2)
print(coef3)
file_copy2=forecast_m2_exp.copy()
file_copy2.reset_index(inplace=True)
file_copy2['year']=pd.to_datetime(file_copy2.ds).dt.year
data1=file_copy2[file_copy2['year']==2015]
data1=data1['yhat']
data2=test3['y']
coef2, p = spearmanr(data1, data2)
print(coef2)
file_copy1=forecast_m1_exp.copy()
file_copy1.reset_index(inplace=True)
file_copy1['year']=pd.to_datetime(file_copy1.ds).dt.year
data1=file_copy1[file_copy1['year']==2020]
data1=data1['yhat']
data2=test1['y']
coef1, p = spearmanr(data1, data2)
print(coef1)
```

```
data=[["Philadelphia",'10',rmse1,coef1],["Philadelphia",'5',rmse2,coef2],["Philadelphia",'4',r
mse3,coef3],["Philadelphia",'3',rmse4,coef4],["Philadelphia",'2',rmse5,coef5],["Philadelphia",
'1',rmse6,coef6]]
result=pd.DataFrame(data,columns=['City','Years of training','RMSE value','Spearman
Coeff'])
result
```

#Prophet(San Francisco)

```
%pip install pystan==2.19.1.1
pip install fbprophet
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/SF_2
003_to_May_2018-4.csv",inferSchema=True,header=True)
file=df1.toPandas()
file.drop(['_c0', 'objectid', 'dc_dist', 'psa','dispatch_date_time','dispatch_time', 'hour_',
'dc_key', 'location_block',
      'ucr_general','point_x', 'point_y', 'lat', 'lng','Unnamed: 0'],axis=1,inplace=True)
import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.dispatch_date).dt.year
file['month'] = pd.to_datetime(file.dispatch_date).dt.month
file['day'] = pd.to_datetime(file.dispatch_date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file.drop('dispatch_date',inplace=True,axis=1)
file = file.groupby('DATE').count()['text_general_code'].to_frame()
file.reset_index(inplace=True)
file.columns = ['ds','y']
file.head()
df_m1 = file.copy()
df_m2=file.copy()
df_m2.sort_values(by='ds',inplace=True)
df_m2['year'] = pd.to_datetime(df_m2.ds).dt.year
df_m2['month'] = pd.to_datetime(df_m2.ds).dt.month
df_m2['day'] = pd.to_datetime(df_m2.ds).dt.day
dataframe1=df_m2[df_m2['year'] == 2010]
dataframe2=df_m2[df_m2['year'] == 2011]
dataframe3=df_m2[df_m2['year'] == 2012]
dataframe4=df_m2[df_m2['year'] == 2013]
dataframe5=df_m2[df_m2['year'] == 2014]
dataframe6=df_m2[df_m2['year'] == 2015]
dataframe7=df_m2[df_m2['year'] == 2016]
dataframe8=df_m2[df_m2['year'] == 2017]
dataframe9=df_m2[df_m2['year'] == 2018]
dataframe10=df_m2[df_m2['year'] == 2019]
```

```
dataframe11=df_m2[df_m2['year'] == 2020]
train1=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe6,dataframe7,da
taframe8,dataframe9,dataframe10]
train1=pd.concat(train1)
test1=[dataframe11]
test1=pd.concat(test1)
train1.drop(['year','month','day'],axis=1,inplace=True)
test1.drop(['year','month','day'],axis=1,inplace=True)
train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
train3.drop(['year','month','day'],axis=1,inplace=True)
test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
train4.drop(['year','month','day'],axis=1,inplace=True)
test4.drop(['year','month','day'],axis=1,inplace=True)

train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
train5.drop(['year','month','day'],axis=1,inplace=True)
test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
train6.drop(['year','month','day'],axis=1,inplace=True)
test6.drop(['year','month','day'],axis=1,inplace=True)

train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
train7.drop(['year','month','day'],axis=1,inplace=True)
test7.drop(['year','month','day'],axis=1,inplace=True)
import numpy as np
from fbprophet import Prophet
```

```
holidays_0 = pd.DataFrame({
      'holiday': '0  window',
      'ds' :pd.to_datetime(
         ['2010-05-09','2011-05-08','2012-05-13','2013-05-12','2014-05-11','2015-05-10','2016-
05-08','2017-05-14','2018-05-13','2019-05-12','2020-05-10','2010-05-24','2011-05-23','2012-
05-21','2013-05-20','2014-05-19','2015-05-18','2016-05-23','2017-05-22','2018-05-21','2019-
05-20','2020-05-18','2010-07-01','2011-07-01','2012-07-01','2013-07-01','2014-07-01','2015-
07-01','2016-07-01','2017-07-01','2018-07-01','2019-07-01','2020-07-01','2010-09-06','2011-
09-05','2012-09-03','2013-09-02','2014-09-01','2015-09-07','2016-09-05','2017-09-04','2018-
09-03','2019-09-02','2020-09-07','2010-11-11','2011-11-11','2012-11-11','2013-11-11','2014-
11-11','2015-11-11','2016-11-11','2017-11-11','2018-11-11','2019-11-11','2020-11-11','2010-
12-25','2011-12-25','2012-12-25','2013-12-25','2014-12-25','2015-12-25','2016-12-25','2017-
12-25','2018-12-25','2019-12-25','2020-12-25']),
      'lower_window' : 0,
      'upper_window' : 0,
   })
holidays_1 = pd.DataFrame({
      'holiday': '1  window',
      'ds' :pd.to_datetime(
         ['2010-10-31','2011-10-31','2012-10-31','2013-10-31','2014-10-31','2015-10-31','2016-
10-31','2017-10-31','2018-10-31','2019-10-31','2020-10-31','2010-01-01','2011-01-01','2012-
01-01','2013-01-01','2014-01-01','2015-01-01','2016-01-01','2017-01-01','2018-01-01','2019-
01-01','2020-01-01']),
      'lower_window' : -1,
      'upper_window' : 1,
   })

holidays_2 = pd.DataFrame({
      'holiday': '2  window',
      'ds' :pd.to_datetime(
         ['2010-08-02','2011-08-01','2012-08-06','2013-08-05','2014-08-04','2015-08-03','2016-
08-01','2017-08-07','2018-08-06','2019-08-05','2020-08-03','2010-10-11','2011-10-10','2012-
10-08','2013-10-14','2014-10-13','2015-10-12','2016-10-10','2017-10-09','2018-10-08','2019-
10-14','2020-10-12']),
      'lower_window' : -2,
      'upper_window' : 1,
   })
holidays_list = pd.concat((holidays_0, holidays_1, holidays_2))
from fbprophet.plot import add_changepoints_to_plot
train1['y'] = np.log(train1['y'])
m1_plain = Prophet(yearly_seasonality=True,changepoint_range=0.4,
weekly_seasonality=True, holidays=holidays_list)
m1_plain.add_country_holidays(country_name='US')
m1_plain.fit(train1)
```

```
# Let's try a forecast for 1 year
future = m1_plain.make_future_dataframe(periods=366)
forecast_m1 = m1_plain.predict(future)
m1_plain.plot(forecast_m1)
m1_plain.plot_components(forecast_m1)
y1 = test1['y'].to_frame()
y1.index = test1['ds']
forecast_m1_exp = np.exp(forecast_m1[['yhat','yhat_lower','yhat_upper']])
forecast_m1_exp.index = forecast_m1['ds']
se = np.square(forecast_m1_exp.loc[:, 'yhat'] - y1['y'])
mse = np.mean(se)
rmse1 = np.sqrt(mse)
print(rmse1)
dummy=train3.copy()

train3['y']=np.log(train3['y'])
m2_plain = Prophet(yearly_seasonality=True,changepoint_range=0.5,
weekly_seasonality=True, holidays=holidays_list1)
m2_plain.add_country_holidays(country_name='US')
m2_plain.fit(train3)

# Let's try a forecast for 1 year
future = m2_plain.make_future_dataframe(periods=365)
forecast_m2 = m2_plain.predict(future)
fig=m2_plain.plot(forecast_m2)
a=add_changepoints_to_plot(fig.gca(),m2_plain,forecast_m2)
m2_plain.plot_components(forecast_m2)
y2 = test3['y'].to_frame()
y2.index = test3['ds']
forecast_m2_exp = np.exp(forecast_m2[['yhat','yhat_lower','yhat_upper']])
forecast_m2_exp.index = forecast_m2['ds']
se = np.square(forecast_m2_exp.loc[:, 'yhat'] - y2['y'])
mse = np.mean(se)
rmse2 = np.sqrt(mse)
print(rmse2)
train4['y']=np.log(train4['y'])
m3_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m3_plain.add_country_holidays(country_name='US')
m3_plain.fit(train4)

# Let's try a forecast for 1 year
future = m3_plain.make_future_dataframe(periods=365)
```

```
forecast_m3 = m3_plain.predict(future)
m3_plain.plot(forecast_m3)
y3 = test4['y'].to_frame()
y3.index = test4['ds']
forecast_m3_exp = np.exp(forecast_m3[['yhat','yhat_lower','yhat_upper']])
forecast_m3_exp.index = forecast_m3['ds']
se = np.square(forecast_m3_exp.loc[:, 'yhat'] - y3['y'])
mse = np.mean(se)
rmse3 = np.sqrt(mse)
print(rmse3)
train5['y']=np.log(train5['y'])
m4_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m4_plain.add_country_holidays(country_name='US')
m4_plain.fit(train5)

# Let's try a forecast for 1 year
future = m4_plain.make_future_dataframe(periods=365)
forecast_m4 = m4_plain.predict(future)
y4 = test5['y'].to_frame()
y4.index = test5['ds']
forecast_m4_exp = np.exp(forecast_m4[['yhat','yhat_lower','yhat_upper']])
forecast_m4_exp.index = forecast_m4['ds']
se = np.square(forecast_m4_exp.loc[:, 'yhat'] - y4['y'])
mse = np.mean(se)
rmse4 = np.sqrt(mse)
print(rmse4)
train6['y']=np.log(train6['y'])
m5_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m5_plain.add_country_holidays(country_name='US')
m5_plain.fit(train6)

# Let's try a forecast for 1 year
future = m5_plain.make_future_dataframe(periods=365)
forecast_m5 = m5_plain.predict(future)
y5 = test6['y'].to_frame()
y5.index = test6['ds']
forecast_m5_exp = np.exp(forecast_m5[['yhat','yhat_lower','yhat_upper']])
forecast_m5_exp.index = forecast_m5['ds']
se = np.square(forecast_m5_exp.loc[:, 'yhat'] - y5['y'])
mse = np.mean(se)
rmse5 = np.sqrt(mse)
print(rmse5)
```

```
train7['y']=np.log(train7['y'])
m6_plain = Prophet(yearly_seasonality=True, weekly_seasonality=True,
holidays=holidays_list)
m6_plain.add_country_holidays(country_name='US')
m6_plain.fit(train7)

# Let's try a forecast for 1 year
future = m6_plain.make_future_dataframe(periods=365)
forecast_m6 = m6_plain.predict(future)
y6 = test7['y'].to_frame()
y6.index = test7['ds']
forecast_m6_exp = np.exp(forecast_m6[['yhat','yhat_lower','yhat_upper']])
forecast_m6_exp.index = forecast_m6['ds']
se = np.square(forecast_m6_exp.loc[:, 'yhat'] - y6['y'])
mse = np.mean(se)
rmse6 = np.sqrt(mse)
print(rmse6)
from scipy.stats import spearmanr
file_copy6=forecast_m6_exp.copy()
file_copy6.reset_index(inplace=True)
file_copy6['year']=pd.to_datetime(file_copy6.ds).dt.year
data1=file_copy6[file_copy6['year']==2011]
data2=test7['y']
data1=data1['yhat']
coef6, p = spearmanr(data1, data2)
file_copy5=forecast_m5_exp.copy()
file_copy5.reset_index(inplace=True)
file_copy5['year']=pd.to_datetime(file_copy5.ds).dt.year
data1=file_copy5[file_copy5['year']==2012]
data1=data1['yhat']
data2=test6['y'][:-1]
coef5, p = spearmanr(data1, data2)
print(coef5)
file_copy4=forecast_m4_exp.copy()
file_copy4.reset_index(inplace=True)
file_copy4['year']=pd.to_datetime(file_copy4.ds).dt.year
data1=file_copy4[file_copy4['year']==2013]
data1=data1['yhat']
data2=test5['y']
coef4, p = spearmanr(data1, data2)
print(coef4)
file_copy3=forecast_m3_exp.copy()
file_copy3.reset_index(inplace=True)
file_copy3['year']=pd.to_datetime(file_copy3.ds).dt.year
```

```
data1=file_copy3[file_copy3['year']==2014]
data1=data1['yhat']
data2=test4['y']
coef3, p = spearmanr(data1, data2)
print(coef3)
file_copy2=forecast_m2_exp.copy()
file_copy2.reset_index(inplace=True)
file_copy2['year']=pd.to_datetime(file_copy2.ds).dt.year
data1=file_copy2[file_copy2['year']==2015]
data1=data1['yhat']
data2=test3['y']
coef2, p = spearmanr(data1, data2)
print(coef2)
file_copy1=forecast_m1_exp.copy()
file_copy1.reset_index(inplace=True)
file_copy1['year']=pd.to_datetime(file_copy1.ds).dt.year
data1=file_copy1[file_copy1['year']==2020]
data1=data1['yhat']
data2=test1['y']
coef1, p = spearmanr(data1, data2)
print(coef1)
```

#Neural Network(San Francisco)

```
pip install keras
pip install tensorflow
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/SF_2
003_to_May_2018-5.csv",inferSchema=True,header=True)
file=df1.toPandas()
file.drop(['PdId','IncidntNum','Incident Code','PdDistrict','Resolution','Address', 'X',
      'Y', 'location', 'SF Find Neighborhoods 2 2',
      'Current Police Districts 2 2', 'Current Supervisor Districts 2 2',
      'Analysis Neighborhoods 2 2', 'DELETE - Fire Prevention Districts 2 2',
      'DELETE - Police Districts 2 2', 'DELETE - Supervisor Districts 2 2',
      'DELETE - Zip Codes 2 2', 'DELETE - Neighborhoods 2 2',
      'DELETE - 2017 Fix It Zones 2 2',
      'Civic Center Harm Reduction Project Boundary 2 2',
      'Fix It Zones as of 2017-11-06 2 2', 'DELETE - HSOC Zones 2 2',
      'Fix It Zones as of 2018-02-07 2 2',
      'CBD, BID and GBD Boundaries as of 2017 2 2',
      'Areas of Vulnerability, 2016 2 2',
      'Central Market/Tenderloin Boundary 2 2',
      'Central Market/Tenderloin Boundary Polygon - Updated 2 2',
      'HSOC Zones as of 2018-06-05 2 2', 'OWED Public Spaces 2 2',
```

```
    'Neighborhoods 2'],axis=1,inplace=True)
import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.Date).dt.year
file['month'] = pd.to_datetime(file.Date).dt.month
file['day'] = pd.to_datetime(file.Date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file = file.groupby('DATE').count()['Category'].to_frame()
file['year'] = pd.to_datetime(file.DATE).dt.year
file['month'] = pd.to_datetime(file.DATE).dt.month
file['day'] = pd.to_datetime(file.DATE).dt.day
file.reset_index(inplace=True)
file.drop(['DATE'],inplace=True,axis=1)
file.dropna(inplace=True)
df_m3=df_m2.copy()
df_m3 = df_m3.groupby('year').sum()['y'].to_frame()
dataframe1=df_m2[df_m2['year'] == 2003]
dataframe2=df_m2[df_m2['year'] == 2004]
dataframe3=df_m2[df_m2['year'] == 2005]
dataframe4=df_m2[df_m2['year'] == 2006]
dataframe5=df_m2[df_m2['year'] == 2007]
dataframe6=df_m2[df_m2['year'] == 2008]
dataframe7=df_m2[df_m2['year'] == 2009]
dataframe8=df_m2[df_m2['year'] == 2010]
dataframe9=df_m2[df_m2['year'] == 2011]
dataframe10=df_m2[df_m2['year'] == 2012]
dataframe11=df_m2[df_m2['year'] == 2013]
dataframe12=df_m2[df_m2['year'] == 2014]
dataframe13=df_m2[df_m2['year'] == 2015]
dataframe14=df_m2[df_m2['year'] == 2016]
dataframe15=df_m2[df_m2['year'] == 2017]
dataframe16=df_m2[df_m2['year'] == 2018]
train2=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe5,dataframe6,da
taframe7,dataframe8,dataframe9,dataframe10]
train2=pd.concat(train2)
test2=[dataframe11]
test2=pd.concat(test2)
#train2.drop(['year','month','day'],axis=1,inplace=True)
#test2.drop(['year','month','day'],axis=1,inplace=True)

train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
```

```
#train3.drop(['year','month','day'],axis=1,inplace=True)
#test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
#train4.drop(['year','month','day'],axis=1,inplace=True)
#test4.drop(['year','month','day'],axis=1,inplace=True)

train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
#train5.drop(['year','month','day'],axis=1,inplace=True)
#test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
#train6.drop(['year','month','day'],axis=1,inplace=True)
#test6.drop(['year','month','day'],axis=1,inplace=True)

train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
#train7.drop(['year','month','day'],axis=1,inplace=True)
#test7.drop(['year','month','day'],axis=1,inplace=True)
import keras
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
trainX=train2[['year','month','day']]
trainY=train2[['Category']]
testX=test2[['year','month','day']]
testY=test2[['Category']]
import numpy as np
trainX = np.asarray(trainX).astype(np.uint8)
trainY = np.asarray(trainY).astype(np.uint8)
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(512,activation='relu'))
```

```
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
import math
from scipy.stats import spearmanr
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
from sklearn.metrics import accuracy_score,mean_squared_error
accuracy1 = mean_squared_error(y_pred,testY)
accuracy1=math.sqrt(accuracy1)
print(accuracy1)
coef1, p = spearmanr(y_pred,testY)
print(coef1)
trainX=train3[['year','month','day']]
trainY=train3[['Category']]
testX=test3[['year','month','day']]
testY=test3[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY, epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy2 = mean_squared_error(y_pred,testY)
accuracy2=math.sqrt(accuracy2)
print(accuracy2)
coef2, p = spearmanr(y_pred,testY)
print(coef2)
trainX=train4[['year','month','day']]
trainY=train4[['Category']]
```

```
testX=test4[['year','month','day']]
testY=test4[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
"""model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))"""
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy3 = mean_squared_error(y_pred,testY)
accuracy3=math.sqrt(accuracy3)
print(accuracy3)
coef3, p = spearmanr(y_pred,testY)
print(coef3)
trainX=train5[['year','month','day']]
trainY=train5[['Category']]
testX=test5[['year','month','day']]
testY=test5[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
#model.add(Dense(64,activation='relu'))
#model.add(Dense(32,activation='relu'))
#model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy4 = mean_squared_error(y_pred,testY)
accuracy4=math.sqrt(accuracy4)
```

```
print(accuracy4)
coef4, p = spearmanr(y_pred,testY)
print(coef4)
trainX=train6[['year','month','day']]
trainY=train6[['Category']]
testX=test6[['year','month','day']]
testY=test6[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
#model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy5 = mean_squared_error(y_pred,testY)
accuracy5=math.sqrt(accuracy5)
print(accuracy5)
coef5, p = spearmanr(y_pred,testY)
print(coef5)
trainX=train7[['year','month','day']]
trainY=train7[['Category']]
testX=test7[['year','month','day']]
testY=test7[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy6 = mean_squared_error(y_pred,testY)
accuracy6=math.sqrt(accuracy6)
print(accuracy6)
coef6, p = spearmanr(y_pred,testY)
print(coef6)
```

#Neural Network(Philly)

```
pip install keras
pip install tensorflow
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/phila
_final-2.csv",inferSchema=True,header=True)
file=df1.toPandas()
file.drop(['PdId','IncidntNum','Incident Code','PdDistrict','Resolution','Address', 'X',
      'Y', 'location', 'SF Find Neighborhoods 2 2',
      'Current Police Districts 2 2', 'Current Supervisor Districts 2 2',
      'Analysis Neighborhoods 2 2', 'DELETE - Fire Prevention Districts 2 2',
      'DELETE - Police Districts 2 2', 'DELETE - Supervisor Districts 2 2',
      'DELETE - Zip Codes 2 2', 'DELETE - Neighborhoods 2 2',
      'DELETE - 2017 Fix It Zones 2 2',
      'Civic Center Harm Reduction Project Boundary 2 2',
      'Fix It Zones as of 2017-11-06 2 2', 'DELETE - HSOC Zones 2 2',
      'Fix It Zones as of 2018-02-07 2 2',
      'CBD, BID and GBD Boundaries as of 2017 2 2',
      'Areas of Vulnerability, 2016 2 2',
      'Central Market/Tenderloin Boundary 2 2',
      'Central Market/Tenderloin Boundary Polygon - Updated 2 2',
      'HSOC Zones as of 2018-06-05 2 2', 'OWED Public Spaces 2 2',
      'Neighborhoods 2'],axis=1,inplace=True)
import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.Date).dt.year
file['month'] = pd.to_datetime(file.Date).dt.month
file['day'] = pd.to_datetime(file.Date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file = file.groupby('DATE').count()['Category'].to_frame()
file['year'] = pd.to_datetime(file.DATE).dt.year
file['month'] = pd.to_datetime(file.DATE).dt.month
file['day'] = pd.to_datetime(file.DATE).dt.day
file.reset_index(inplace=True)
```

```
file.drop(['DATE'],inplace=True,axis=1)
file.dropna(inplace=True)
df_m3=df_m2.copy()
df_m3 = df_m3.groupby('year').sum()['y'].to_frame()
dataframe1=df_m2[df_m2['year'] == 2003]
dataframe2=df_m2[df_m2['year'] == 2004]
dataframe3=df_m2[df_m2['year'] == 2005]
dataframe4=df_m2[df_m2['year'] == 2006]
dataframe5=df_m2[df_m2['year'] == 2007]
dataframe6=df_m2[df_m2['year'] == 2008]
dataframe7=df_m2[df_m2['year'] == 2009]
dataframe8=df_m2[df_m2['year'] == 2010]
dataframe9=df_m2[df_m2['year'] == 2011]
dataframe10=df_m2[df_m2['year'] == 2012]
dataframe11=df_m2[df_m2['year'] == 2013]
dataframe12=df_m2[df_m2['year'] == 2014]
dataframe13=df_m2[df_m2['year'] == 2015]
dataframe14=df_m2[df_m2['year'] == 2016]
dataframe15=df_m2[df_m2['year'] == 2017]
dataframe16=df_m2[df_m2['year'] == 2018]
train2=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe5,dataframe6,da
taframe7,dataframe8,dataframe9,dataframe10]
train2=pd.concat(train2)
test2=[dataframe11]
test2=pd.concat(test2)
#train2.drop(['year','month','day'],axis=1,inplace=True)
#test2.drop(['year','month','day'],axis=1,inplace=True)

train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
#train3.drop(['year','month','day'],axis=1,inplace=True)
#test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
#train4.drop(['year','month','day'],axis=1,inplace=True)
#test4.drop(['year','month','day'],axis=1,inplace=True)

train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
```

```
test5=[dataframe4]
test5=pd.concat(test5)
#train5.drop(['year','month','day'],axis=1,inplace=True)
#test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
#train6.drop(['year','month','day'],axis=1,inplace=True)
#test6.drop(['year','month','day'],axis=1,inplace=True)

train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
#train7.drop(['year','month','day'],axis=1,inplace=True)
#test7.drop(['year','month','day'],axis=1,inplace=True)
import keras
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
trainX=train2[['year','month','day']]
trainY=train2[['Category']]
testX=test2[['year','month','day']]
testY=test2[['Category']]
import numpy as np
trainX = np.asarray(trainX).astype(np.uint8)
trainY = np.asarray(trainY).astype(np.uint8)
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
import math
from scipy.stats import spearmanr
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
```

```
from sklearn.metrics import accuracy_score,mean_squared_error
accuracy1 = mean_squared_error(y_pred,testY)
accuracy1=math.sqrt(accuracy1)
print(accuracy1)
coef1, p = spearmanr(y_pred,testY)
print(coef1)
trainX=train3[['year','month','day']]
trainY=train3[['Category']]
testX=test3[['year','month','day']]
testY=test3[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY, epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy2 = mean_squared_error(y_pred,testY)
accuracy2=math.sqrt(accuracy2)
print(accuracy2)
coef2, p = spearmanr(y_pred,testY)
print(coef2)
trainX=train4[['year','month','day']]
trainY=train4[['Category']]
testX=test4[['year','month','day']]
testY=test4[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
"""""model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))"""""
model.add(Dense(1, activation='relu'))
```

```
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy3 = mean_squared_error(y_pred,testY)
accuracy3=math.sqrt(accuracy3)
print(accuracy3)
coef3, p = spearmanr(y_pred,testY)
print(coef3)
trainX=train5[['year','month','day']]
trainY=train5[['Category']]
testX=test5[['year','month','day']]
testY=test5[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
#model.add(Dense(64,activation='relu'))
#model.add(Dense(32,activation='relu'))
#model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy4 = mean_squared_error(y_pred,testY)
accuracy4=math.sqrt(accuracy4)
print(accuracy4)
coef4, p = spearmanr(y_pred,testY)
print(coef4)
trainX=train6[['year','month','day']]
trainY=train6[['Category']]
testX=test6[['year','month','day']]
testY=test6[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
```

```
#model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy5 = mean_squared_error(y_pred,testY)
accuracy5=math.sqrt(accuracy5)
print(accuracy5)
coef5, p = spearmanr(y_pred,testY)
print(coef5)
trainX=train7[['year','month','day']]
trainY=train7[['Category']]
testX=test7[['year','month','day']]
testY=test7[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy6 = mean_squared_error(y_pred,testY)
accuracy6=math.sqrt(accuracy6)
print(accuracy6)
coef6, p = spearmanr(y_pred,testY)
print(coef6)
```

#Neural Network(Chicago)

```
pip install keras
```

```
pip install tensorflow
df1 =
spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/121003196@sastra.ac.in/chica
go.csv",inferSchema=True,header=True)
file=df1.toPandas()
file.drop(['PdId','IncidntNum','Incident Code','PdDistrict','Resolution','Address', 'X',
      'Y', 'location', 'SF Find Neighborhoods 2 2',
      'Current Police Districts 2 2', 'Current Supervisor Districts 2 2',
      'Analysis Neighborhoods 2 2', 'DELETE - Fire Prevention Districts 2 2',
      'DELETE - Police Districts 2 2', 'DELETE - Supervisor Districts 2 2',
      'DELETE - Zip Codes 2 2', 'DELETE - Neighborhoods 2 2',
      'DELETE - 2017 Fix It Zones 2 2',
      'Civic Center Harm Reduction Project Boundary 2 2',
      'Fix It Zones as of 2017-11-06 2 2', 'DELETE - HSOC Zones 2 2',
      'Fix It Zones as of 2018-02-07 2 2',
      'CBD, BID and GBD Boundaries as of 2017 2 2',
      'Areas of Vulnerability, 2016 2 2',
      'Central Market/Tenderloin Boundary 2 2',
      'Central Market/Tenderloin Boundary Polygon - Updated 2 2',
      'HSOC Zones as of 2018-06-05 2 2', 'OWED Public Spaces 2 2',
      'Neighborhoods 2'],axis=1,inplace=True)
import pandas as pd
import matplotlib.pyplot as plt
file['year'] = pd.to_datetime(file.Date).dt.year
file['month'] = pd.to_datetime(file.Date).dt.month
file['day'] = pd.to_datetime(file.Date).dt.day
file['DATE'] = pd.to_datetime({'year':file['year'], 'month':file['month'], 'day':file['day']})
file = file.groupby('DATE').count()['Category'].to_frame()
file['year'] = pd.to_datetime(file.DATE).dt.year
file['month'] = pd.to_datetime(file.DATE).dt.month
file['day'] = pd.to_datetime(file.DATE).dt.day
file.reset_index(inplace=True)
file.drop(['DATE'],inplace=True,axis=1)
file.dropna(inplace=True)
df_m3=df_m2.copy()
df_m3 = df_m3.groupby('year').sum()['y'].to_frame()
dataframe1=df_m2[df_m2['year'] == 2003]
dataframe2=df_m2[df_m2['year'] == 2004]
dataframe3=df_m2[df_m2['year'] == 2005]
dataframe4=df_m2[df_m2['year'] == 2006]
dataframe5=df_m2[df_m2['year'] == 2007]
dataframe6=df_m2[df_m2['year'] == 2008]
dataframe7=df_m2[df_m2['year'] == 2009]
dataframe8=df_m2[df_m2['year'] == 2010]
```

```
dataframe9=df_m2[df_m2['year'] == 2011]
dataframe10=df_m2[df_m2['year'] == 2012]
dataframe11=df_m2[df_m2['year'] == 2013]
dataframe12=df_m2[df_m2['year'] == 2014]
dataframe13=df_m2[df_m2['year'] == 2015]
dataframe14=df_m2[df_m2['year'] == 2016]
dataframe15=df_m2[df_m2['year'] == 2017]
dataframe16=df_m2[df_m2['year'] == 2018]
train2=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5,dataframe5,dataframe6,da
taframe7,dataframe8,dataframe9,dataframe10]
train2=pd.concat(train2)
test2=[dataframe11]
test2=pd.concat(test2)
#train2.drop(['year','month','day'],axis=1,inplace=True)
#test2.drop(['year','month','day'],axis=1,inplace=True)

train3=[dataframe1,dataframe2,dataframe3,dataframe4,dataframe5]
train3=pd.concat(train3)
test3=[dataframe6]
test3=pd.concat(test3)
#train3.drop(['year','month','day'],axis=1,inplace=True)
#test3.drop(['year','month','day'],axis=1,inplace=True)

train4=[dataframe1,dataframe2,dataframe3,dataframe4]
train4=pd.concat(train4)
test4=[dataframe5]
test4=pd.concat(test4)
#train4.drop(['year','month','day'],axis=1,inplace=True)
#test4.drop(['year','month','day'],axis=1,inplace=True)

train5=[dataframe1,dataframe2,dataframe3]
train5=pd.concat(train5)
test5=[dataframe4]
test5=pd.concat(test5)
#train5.drop(['year','month','day'],axis=1,inplace=True)
#test5.drop(['year','month','day'],axis=1,inplace=True)

train6=[dataframe1,dataframe2]
train6=pd.concat(train6)
test6=[dataframe3]
test6=pd.concat(test6)
#train6.drop(['year','month','day'],axis=1,inplace=True)
#test6.drop(['year','month','day'],axis=1,inplace=True)
```

```
train7=[dataframe1]
train7=pd.concat(train7)
test7=[dataframe2]
test7=pd.concat(test7)
#train7.drop(['year','month','day'],axis=1,inplace=True)
#test7.drop(['year','month','day'],axis=1,inplace=True)
import keras
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
trainX=train2[['year','month','day']]
trainY=train2[['Category']]
testX=test2[['year','month','day']]
testY=test2[['Category']]
import numpy as np
trainX = np.asarray(trainX).astype(np.uint8)
trainY = np.asarray(trainY).astype(np.uint8)
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
import math
from scipy.stats import spearmanr
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
from sklearn.metrics import accuracy_score,mean_squared_error
accuracy1 = mean_squared_error(y_pred,testY)
accuracy1=math.sqrt(accuracy1)
print(accuracy1)
coef1, p = spearmanr(y_pred,testY)
print(coef1)
trainX=train3[['year','month','day']]
trainY=train3[['Category']]
testX=test3[['year','month','day']]
testY=test3[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
```

```
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY, epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy2 = mean_squared_error(y_pred,testY)
accuracy2=math.sqrt(accuracy2)
print(accuracy2)
coef2, p = spearmanr(y_pred,testY)
print(coef2)
trainX=train4[['year','month','day']]
trainY=train4[['Category']]
testX=test4[['year','month','day']]
testY=test4[['Category']]
model = Sequential()
model.add(Dense(12, input_dim=3, activation='relu'))
"""""model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))"""""
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy3 = mean_squared_error(y_pred,testY)
accuracy3=math.sqrt(accuracy3)
print(accuracy3)
coef3, p = spearmanr(y_pred,testY)
print(coef3)
trainX=train5[['year','month','day']]
trainY=train5[['Category']]
```

```
testX=test5[['year','month','day']]
testY=test5[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
#model.add(Dense(64,activation='relu'))
#model.add(Dense(32,activation='relu'))
#model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy4 = mean_squared_error(y_pred,testY)
accuracy4=math.sqrt(accuracy4)
print(accuracy4)
coef4, p = spearmanr(y_pred,testY)
print(coef4)
trainX=train6[['year','month','day']]
trainY=train6[['Category']]
testX=test6[['year','month','day']]
testY=test6[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
#model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy5 = mean_squared_error(y_pred,testY)
```

```
accuracy5=math.sqrt(accuracy5)
print(accuracy5)
coef5, p = spearmanr(y_pred,testY)
print(coef5)
trainX=train7[['year','month','day']]
trainY=train7[['Category']]
testX=test7[['year','month','day']]
testY=test7[['Category']]
model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu'))
#model.add(Dense(512,activation='relu'))
#model.add(Dense(256,activation='relu'))
#model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))
#model.add(Dense(4,activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam')
model.fit(trainX,trainY,epochs=300)
y_pred=model.predict(testX)
y_pred = np.asarray(y_pred).astype(np.int64)
testY=testY.to_numpy()
accuracy6 = mean_squared_error(y_pred,testY)
accuracy6=math.sqrt(accuracy6)
print(accuracy6)
coef6, p = spearmanr(y_pred,testY)
print(coef6)
```

#LSTM

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc


import seaborn as sns
import pandas as pd
```

```
df=pd.read_csv("Philadelphia_dataset.csv")
df.head()

df['Date_Time']= pd.to_datetime(df['Date_Time'])
df1=df[["Date_Time","Date"]]
df.dtypes

df1.head()
df1['Date']= pd.to_datetime(df1['Date'])

df1["Date"].value_counts().sort_index()

df1.shape

df1.dropna(inplace=True)
tdf = pd.DataFrame({"Date" :df1["Date_Time"].dt.date.unique(),"Crimes" :df1["Date"].value
_counts()})
tdf=tdf.reset_index(drop=True)
tdf["Date"]=pd.to_datetime(tdf["Date"])
tdf.head()

test=pd.DataFrame()
for index, row in tdf.iterrows():
  if str(row[0].year) == str(2020):
    print(index)
    break

tdf_1=tdf[["Crimes"]]
train=tdf_1[:2091]
valid=tdf_1[2191:]

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(train)
scaled_train_data = scaler.transform(train)
scaled_test_data = scaler.transform(valid)
scaled_train_data
generator = TimeseriesGenerator(scaled_train_data, scaled_train_data, length=n_input, batch
_size=1)
print(generator)

from keras.preprocessing.sequence import TimeseriesGenerator
```

```
n_input = 1
n_features= 1
generator = TimeseriesGenerator(scaled_train_data, scaled_train_data, length=n_input, batch
_size=1)
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

lstm_model = Sequential()
lstm_model.add(LSTM(24, activation='relu', input_shape=(n_input, n_features),return_seque
nces=False))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.fit_generator(generator,epochs=15)

import numpy as np
lstm_predictions_scaled = list()

batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))

for i in range(len(valid)):
   lstm_pred = lstm_model.predict(current_batch)[0]
   lstm_predictions_scaled.append(lstm_pred)
   current_batch = np.append(current_batch[:,1:,:],[[lstm_pred]],axis=1)
lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)

from math import sqrt
from sklearn.metrics import mean_squared_error
rms = sqrt(mean_squared_error(valid.value,lstm_predictions))
print(rms)

lstm_predictions

def load_data(stock, seq_len,n_periods):
   X_train = []
   y_train = []
   for i in range(seq_len, len(stock)):
      X_train.append(stock.iloc[i-seq_len : i, 0])
      y_train.append(stock.iloc[i, 0])

   val=len(X_train)-n_periods
```

```
    X_test = X_train[val:]
    y_test = y_train[val:]
    X_train = X_train[:val]
    y_train = y_train[:val]
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)
    X_train = np.reshape(X_train, (val, seq_len, 1))
    X_test = np.reshape(X_test, (X_test.shape[0], seq_len, 1))
    return [X_train, y_train, X_test, y_test]

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import numpy as np
import pandas as pd
train = df[:int(0.85*(len(df)))]
valid = df[int(0.85*(len(df))):]

from sklearn import preprocessing
import sklearn
scaler = sklearn.preprocessing.MinMaxScaler()
train["Crimes"]=scaler.fit_transform(train["Crimes"].values.reshape(-1,1))
df_norm = train
df_norm.shape
seq_len = 30 #choose sequence length
import numpy as np
period=len(valid)
X_train, y_train, X_test, y_test = load_data(train, seq_len,period)
print('X_train.shape = ',X_train.shape)
print('y_train.shape = ', y_train.shape)
print('X_test.shape = ', X_test.shape)
print('y_test.shape = ',y_test.shape)

from keras.layers import Dense,Dropout,SimpleRNN,LSTM
from keras.models import Sequential
lstm_model = Sequential()
lstm_model.add(LSTM(50,activation="tanh",return_sequences=True, input_shape=(X_train.
shape[1],1)))
lstm_model.add(Dropout(0.15))
lstm_model.add(LSTM(50,activation="tanh",return_sequences=True))
```

```
lstm_model.add(Dropout(0.15))
lstm_model.add(LSTM(50,activation="tanh",return_sequences=True))
lstm_model.add(Dropout(0.15))
lstm_model.add(LSTM(50,activation="tanh",return_sequences=False))
lstm_model.add(Dropout(0.15))
lstm_model.add(Dense(1))
lstm_model.summary()

lstm_model.compile(optimizer="adam",loss="MSE")
lstm_model.fit(X_train, y_train, epochs=10,batch_size=20)

lstm_predictions=lstm_model.predict(X_test)
from math import sqrt
from sklearn.metrics import mean_squared_error
rms = sqrt(mean_squared_error(y_test, lstm_predictions))
print(rms)

import matplotlib.pyplot as plt
plt.figure(figsize=(30, 15))
plt.plot(y_test,color='blue')
plt.plot(lstm_predictions,color='red')

y_test_trans=scaler.inverse_transform(y_test.reshape(-1,1))
y_test_trans=scaler.inverse_transform(y_test.reshape(-1,1))
lstm_trans=scaler.inverse_transform(lstm_predictions.reshape(-1,1))
dff=df

X_train.shape[0]

forecast = pd.DataFrame(lstm_trans,columns=['Prediction'])
actual=pd.DataFrame(y_test_trans,columns=['Actual'])
pdcat=pd.concat([actual, forecast], axis=1)
pdcat

from scipy.stats import spearmanr
spearmanr(pdcat["Actual"],pdcat["Prediction"])
```