National College of Ireland

Analysing Crime Patterns using Machine Learning: A case study in Chicago

MSc in Data Analytics (MSCDAD - B)
Configuration Manual

Himanshi Himanshi
Student ID: x20218001

School of Computing
National College of Ireland

Supervisor: Dr.Paul Stynes, Musfira Jilani, Dr.Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Himanshi Himanshi
Student ID:	x20218001
Programme:	Configuration Manual
Year:	2022
Module:	MSc in Data Analytics (MSCDAD - B)
Supervisor:	Dr.Paul Stynes, Musfira Jilani, Dr.Pramod Pathak
Submission Due Date:	15/08/2022
Project Title:	Analysing Crime Patterns using Machine Learning: A case study in Chicago
Word Count:	1590
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Analysing Crime Patterns using Machine Learning: A case study in Chicago

Himanshi Himanshi
x20218001

1 Introduction

Detailed instructions on the hardware, software and programming requirements for implementing this research project are included in this configuration manual:
"Analysing Crime Patterns using Machine Learning: A case study in Chicago"

2 System Configuration

2.1 Hardware

- **Processor:** Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz 3.10 GHz
- **RAM:** 16 GB
- **System Type:** Windows OS, 64-bit
- **GPU:** Intel(R) UHD Graphics 630
- **Storage:** 500 GB HDD

2.2 Software

- **Microsoft Excel 2016:** This study stored downloaded datasets as csv files (comma separated values) in this spreadsheet tool provided by Microsoft.
- **Anaconda Distribution-Jupyter Notebook:** Anaconda distribution's website [1] provided access to the open source software. R studio or Python can be used to run machine learning models using platforms like jupyter notebooks, spyder, or R studio. In this study, exploratory data analysis (EDA), data manipulation, preprocessing of data, transformation of data, and data visualization were done using Python (version 3.10.5) on Jupyter notebooks.
- **Tableau CRM:** Initially, this study made some visualization to know more about crime patterns using Tableau CRM[2] provided by salesforce

[1]https://www.anaconda.com/products/distribution
[2]https://www.salesforce.com/eu/products/einstein-analytics/overview/

3 Project Development

Python is used exclusively to implement this research work. Pre-processing of data, merging of all datasets, normalization, and visualization are the initial steps of a research project.Using Python machine learning libraries such as sk-learn (scikit-learn), pandas, and keras, predictive modelling is then performed following the data preparation activities.

3.1 Data Preparation

The pandas (dataframe) and numpy (arrays) libraries are used for data import and data processing. The pre-processing procedures used for crime dataset are described in the sections that follow.

This study used Chicago city crime dataset from 2001 to 2021 December.Instead of downloading one csv file, 8 csv files from different time periods downloaded from chicago government data portal [3] then concatenated them using concat function from pandas library and formed one dataset having 91,59,279 rows and 22 columns as shown in Figure 1. The merging of one dataset from different datasets has done to reduce the load of reading one big csv file of size 8GB and make processing faster.Then using count_nonzero function present in numpy library is used to get all the null values present in dataset and there are 2033173 values are present in dataset. This study dropped all the values present in dataset using dropna function because even after removing them, dataset is enough values for proper modelling. As per [4], Chicago has boundaries with latitude and logitude as 41.6439,-87.9401; 41.9437,-87.5878 respectively. This study considered only those rows which comes under this boundary value and removed otherwise. After removing, dataset has 83,39,817 crime incidents. Dataset contain date column that indicates the date of crime and also the year column that tells the year of crime incident. To make sure dataset contains accurate values, this study validate the year column value and year from date column. To accomplish this, there is a need to create dateTime index using DatetimeIndex function but there is no mismatch present.

```
# Individual CSV Files downloaded from data portal
chicago_df_1 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2001_to_2004.csv/Chicago_Crimes_2001_to_2004.csv', error_bad_lines=False)
chicago_df_2 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2005_to_2007.csv/Chicago_Crimes_2005_to_2007.csv', error_bad_lines=False)
chicago_df_3 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2008_to_2011.csv/Chicago_Crimes_2008_to_2011.csv', error_bad_lines=False)
chicago_df_4 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2012_to_2016.csv/Chicago_Crimes_2012_to_2017.csv', error_bad_lines=False)
chicago_df_5 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2018.csv\Crimes_-_2018.csv', error_bad_lines=False)
chicago_df_6 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crime_2017.csv\Crimes_-_2017.csv', error_bad_lines=False)
chicago_df_7 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS/Chicago_Crimes_2019.csv\Crimes_-_2019.csv', error_bad_lines=False)
chicago_df_8 = pd.read_csv('C:/Users/OwenB/Desktop/THESIS\Chicago_Crime_2020_to_2021.csv\Chicago_Crime_2020-2021.csv', error_bad_lines=False)
# Removed the unwanted column present in 4 dataframes
df1=chicago_df_1.drop(['Unnamed: 0'], axis=1)
df2=chicago_df_2.drop(['Unnamed: 0'], axis=1)
df3=chicago_df_3.drop(['Unnamed: 0'], axis=1)
df4=chicago_df_4.drop(['Unnamed: 0'], axis=1)
#Concatenate different datasets into one dataframe
chicago_df = pd.concat([df1, df2, df3, df4, chicago_df_5,chicago_df_6,chicago_df_7,chicago_df_8], ignore_index=False, axis=0)
#To know the shape and size of final datasets
chicago_df.shape
#To know the columns and data types of columns present in datasets
chicago_df.info()
#To know the null values present in dataset
np.count_nonzero(chicago_df.isnull())
#To drop the null values present in dataset
chicago_df.dropna(how='any', axis=0, inplace=True)
#exploring location column and drop this because dataset already have latitude and logitude columns available
chicago_df['Location']
chicago_df.drop(['Location'], axis=1, inplace=True)
#Exploring Latitude and Longitude
#we'll remove all rows outside of this range
print('Current rows:', chicago_df.shape[0])
chicago_df = chicago_df[((chicago_df.Latitude >= 41.64) & (chicago_df.Longitude <= -87.50)) |
        ((chicago_df.Latitude <= 41.94) & (chicago_df.Longitude >= -87.94))]
print('Rows after removing out of box points:', chicago_df.shape[0])
#Now for each row we'll check if year in Date column matches year in Year column
#creating datetime index
chicago_df.index = pd.DatetimeIndex(chicago_df.Date)
#removing rows with mismatch in year
print('Current rows:', chicago_df.shape[0])
chicago_df = chicago_df[chicago_df.index.year == chicago_df.Year]
print('Rows after removing mismatch:', chicago_df.shape[0])
```

Figure 1: Handling of Erroneous data

To study crime pattern based on monthly, weekly and daily data there is a need to

[3] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2
[4] https://boundingbox.klokantech.com/

Table 1: Chicago crime dataset

Attribute Code	Description	Domain
Date	Date when the incident occurred	2001-2021
Primary type	Type of crime	24 different types
Description	Subcategory of the primary description	Narcotics, Theft, battery etc.
Location Description	Description of the location where the incident occurred	Text
Arrest	Indicates whether an arrest was made	Boolean (0,1)
Domestic	Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act	Boolean (0,1)
District	Indicates the police district where the incident occurred.	Number
Community area	Indicates the community area where the incident occurred	Number
X Coordinate	The x coordinate of the location where the incident occurred in State Plane Illinois	Location co-ordinates
Y Coordinate	The y coordinate of the location where the incident occurred in State Plane Illinois	Location co-ordinates
Year	Year the incident occurred	2001-2021
Latitude	The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.	Location co-ordinates
Longitude	The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.	Location co-ordinates
Month	Month when crime occurred created from date of crime	01-12
dayOfWeek	Which day of week crime occurred	Monday-Sunday
dayOfMonth	Which day of month crime occurred	01-31
dayOfYear	Which day of year crime occurred	01-365
weekOfMonth	Which week of month crime occurred	01-07
weekOfYear	Which week of year crime occurred	01-53

create new columns like month, dayOfWeek, dayOfMonth, and weekOfMonth(wom) from date column present in dataset. Crime dataset is pretty big and there may be possibility of duplication in dataset therefore, in this study, drop_duplicates method has used to drop duplicate value which identified by ID and case_number column. As a result of this step, 16,66,851 rows were remove from dataset. There are many irrelevant columns present in dataset like updatedOn, Location because dataset has latitude and longitude columns available, ID and case number so using drop method these columns have been removed from dataset as shown in Figure 2.After removing and creating multiple columns, Chicago crime dataset is shown in Table1.

```
#we'll create some new columns like month, dayOfWeek, dayOfMonth, weekOfMonth(wom)
chicago_df['Month'] = chicago_df.index.month
chicago_df['dayOfWeek'] = chicago_df.index.dayofweek
chicago_df['dayOfMonth'] = chicago_df.index.day
chicago_df['dayOfYear'] = chicago_df.index.dayofyear
chicago_df['weekOfMonth'] = chicago_df.dayOfMonth.apply(lambda d: (d - 1) // 7 + 1)
dayOfYear = list(chicago_df.index.dayofyear)
weekOfYear = [math.ceil(i/7) for i in dayOfYear]
chicago_df['weekOfYear'] = weekOfYear
#de-duplication of data
print('Current rows:', chicago_df.shape[0])
chicago_df.drop_duplicates(subset=['ID', 'Case Number'], inplace=True)
print('Rows after deduplication:', chicago_df.shape[0])
#Drop irrelavant columns
chicago_df.drop(['Location'], axis=1, inplace=True)
chicago_df.drop(['ID', 'Case Number'], axis=1, inplace=True)
chicago_df.drop(['Updated On'], axis=1, inplace=True)
```

Figure 2: Data Transformation

3.2 Feature Engineering

Before applying the models to the data, effective feature engineering Palanivinayagam et al. (2021) helps the models perform better and reduce any potential errors. These engineering procedures were carried out in two steps: normalizing to handle the numerical features and feature selection to choose the best features.

3.2.1 Feature Selection

There is a need to make data appropriate to predict with great accuracy. Therefore, feature selection is a technique to take highly relevant features in dataset and drop others. There is a column present in Chicago crime dataset i.e. Primary type which signifies the type of crime like theft, robbery , assault etc. Initially it was 35 types of crime but there

are some primary crimes which occurs frequently like theft, battery, criminal damage, narcotics etc. and in contrast crime like ritualism, non-criminal are less in number as shown in Figure 3. so, this study has dropped these crime and treat them as outliers. A few hundred rows are dropped, and since this is a categorical feature, the problem's complexity is also decreased.

```
#Checking Primary type of crime
plt.figure(figsize=(8,10))
chicago_df.groupby([chicago_df['Primary Type']]).size().sort_values(ascending=True).plot(kind='barh')
plt.title('Number of crimes by type')
plt.ylabel('Crime Type')
plt.xlabel('Number of crimes')
plt.show()
chicago_df['Primary Type'].value_counts(normalize=True)
#Dropping crimes which are less in numbers
print('Current rows:', chicago_df.shape[0])
chicago_df['Primary Type'] = chicago_df['Primary Type'].astype(str)
chicago_df = chicago_df[(chicago_df['Primary Type'] != 'OBSCENITY') &
            (chicago_df['Primary Type'] != 'PUBLIC INDECENCY') &
            (chicago_df['Primary Type'] != 'CONCEALED CARRY LICENSE VIOLATION') &
            (chicago_df['Primary Type'] != 'NON-CRIMINAL') &
            (chicago_df['Primary Type'] != 'OTHER NARCOTIC VIOLATION') &
            (chicago_df['Primary Type'] != 'NON - CRIMINAL') &
            (chicago_df['Primary Type'] != 'HUMAN TRAFFICKING') &
            (chicago_df['Primary Type'] != 'NON-CRIMINAL (SUBJECT SPECIFIED)')]
#THEFT                              0.211649
#BATTERY                            0.184187
#CRIMINAL DAMAGE                    0.114832
#NARCOTICS                          0.097184
#ASSAULT                            0.064863
#OTHER OFFENSE                      0.062123
#BURGLARY                           0.056336
#MOTOR VEHICLE THEFT                0.045721
#DECEPTIVE PRACTICE                 0.041444
#ROBBERY                            0.037967
#CRIMINAL TRESPASS                  0.028085
#WEAPONS VIOLATION                  0.013242
#PROSTITUTION                       0.008917
#PUBLIC PEACE VIOLATION             0.007087
#OFFENSE INVOLVING CHILDREN         0.006861
#SEX OFFENSE                        0.003625
#CRIM SEXUAL ASSAULT                0.003587
#INTERFERENCE WITH PUBLIC OFFICER   0.002571
#GAMBLING                           0.001985
#LIQUOR LAW VIOLATION               0.001844
#HOMICIDE                           0.001674
#ARSON                              0.001674
#KIDNAPPING                         0.000864
#INTIMIDATION                       0.000589
#STALKING                           0.000568
#CRIMINAL SEXUAL ASSAULT            0.000520
#RITUALISM                          0.000002
```

Figure 3: Primary Types of Crime

There is a column Location Description which signifies the specific area where crime happened and there are more than 200 different types of location but this study considered only top 50 locations where maximum crime happened and considered othere locations as outliers so after removing locations 65,17,971 rows left in dataset. Similarly, only top 100 location descriptions were considered by this study. This study check the number of crimes happened on district level and found out that there are 27 districts present in Chicago city but out of those four districts have nearly zero crime rate so, we dropped all the rows related to those districts.77 different community sections are available. A different hue is used to represent each neighborhood. The crime rate in Community Area Number 25 is extremely high. 0.0 has an incorrect community area (community areas should be from 1-77). This study removed these rows as mentioned in Figure 4. After handling outliers, final dataset has 6259111 rows and 24 columns.

3.2.2 Normalization

The MinMaxScalar module in Python was used for normalization, as shown in Figure 5.

3.3 Data Modelling

The machine learning scikit libraries for Python were used for modelling. For modelling, the libraries keras, tensorflow, random forest regressor, and linear regression were utilized.

```
plt.figure(figsize=(8,10))
chicago_df.groupby([chicago_df['Location Description']]).size().sort_values(ascending=True)[-70:].plot(kind='barh')
plt.title('Number of crimes by locations')
plt.ylabel('Locations')
plt.xlabel('Number of crimes')
plt.show()
#Getting top 50 locations where maximum crime happened and removing rest
top50Locations = list(chicago_df.groupby([chicago_df['Location Description']]).size().sort_values(ascending=True)[-50:].index)
print('Current rows:', chicago_df.shape[0])
chicago_df = chicago_df[chicago_df['Location Description'].isin(top50Locations)]
print('Rows after removing location outliers:', chicago_df.shape[0])
chicago_df['Location Description'].value_counts()
#Similarly, only top 100 location descriptions were considered by this study
top100Desc = list(chicago_df.groupby([chicago_df['Description']]).size().sort_values(ascending=True)[-100:].index)
#Checking district in Chicago city
plt.figure(figsize=(8,10))
chicago_df.groupby([chicago_df['District']]).size().sort_values(ascending=True).plot(kind='barh')
plt.title('Number of crimes by District')
plt.ylabel('Districts')
plt.xlabel('Number of crimes')
plt.show()
#removing districts which have nearly zero crime rate
chicago_df = chicago_df[chicago_df['District'] != 21.0]
chicago_df = chicago_df[chicago_df['District'] != 23.0]
chicago_df = chicago_df[chicago_df['District'] != 13.0]
#Checking different communities
print('Current rows:', chicago_df.shape[0])
chicago_df = chicago_df[chicago_df['Community Area'] != 0.0]
print('Rows after removing description outliers:', chicago_df.shape[0])
```

Figure 4: Handling Outliers

```
from sklearn.preprocessing import MinMaxScaler
# Data Normalization using MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_data = sc.fit_transform(training_set)

#Sliding Function
seq_length = 6
x, y = sliding_windows_DataLoader(training_data, seq_length)
```

Figure 5: Feature Scaling using MinMaxScaler

3.3.1 Data Split

The dataset has split between training dataset and validating dataset. For Arima, LSTM and Random Forest regression data split has done train-test as 80 and 20 respectively.For moving average and CNN-LSTM models data from January 2001 to 2020 December has kept under training dataset and data from 2021 January to 2021 December kept under validation dataset.

3.3.2 ARIMA

Arima model was performed by aggregating the number of crimes per month for all years as shown in Figure 6.After that, stationarity of model has checked to make sure that mean and variance of time series does not vary over time. It was developed a new stationarity function that would run the Dickey-Fuller test. statistic value is greater than critical value, that means time series is stationary. We made interactive plot to check the pattern over different period of time. Then splitting of dataset into train and test datasets has done as shown in Figure 7. As illustrated in Figure 6, the ARIMA model for this study was created with order (0,1,3) and fitted using pmdarima library.Next,.predict() predicts the number of additional time steps. ARIMA model was evaluated on the basis of RMSE,MAE and R2 score.

3.3.3 LSTM Model

Second Model was LSTM and Dickey Fuller testwas used to check the stationarity of data and then we plot the responses for different events and regions as shown in Figure 9.Number of epochs kept 2500, learning rate as 0.01, number of hidden layers as 4 and total

```
# resample into Months

plt.plot(df.resample('M').size())
plt.title('Crimes Count Per Month')
plt.xlabel('Months')
plt.ylabel('Number of Crimes')
# aggregating the number of cases per month for all years
ts_df = pd.DataFrame(df.resample('M').size().reset_index())
ts_df.columns = ['Date', 'Crime Count'] # renaming the columns
ts_df.head()
ts_df = ts_df.set_index('Date')
```

```
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
adf_test(ts_df['Crime Count'])
```

```
# plot interactive slider chart
fig = px.line(ts_df, x='Date',y='Crime Count', title= 'Crime count')

fig.update_xaxes(
rangeslider_visible =True,
rangeselector=dict(
        buttons=list([
                dict(count=1,label="1y",step="year",stepmode="backward"),
                dict(count=2,label="3y",step="year",stepmode="backward"),
                dict(count=3,label="5y",step="year",stepmode="backward"),
                dict(step="all")
                    ])
                )
                )
fig.show()
```

Figure 6: Data Resampling and Stationarity Check

```
# splitting into train and test set
train = ts_df[:181]
test = ts_df[181:]
print(train.shape)
print(test.shape)
#Data plotting
plt.plot(train)
plt.plot(test)
autocorrelation_plot(train)
plot_pacf(train,lags=20)
```

```
#Modelling
model = pm.auto_arima(train,m=12,start_p=0,start_q=3, max_order=5,
                      error_action='ignore',test='adf',seasonal=True,
                      trace=True,stepwise=True)
model.summary()
```

```
#Predictions
prediction = pd.DataFrame(model.predict(n_periods=71),index = test.index,columns =['Predicted Crime Count'])
prediction
test
```

```
# plot the predictions
plt.plot(train, label ='Train')
plt.plot(test['Crime Count'], label='Test')
plt.plot(prediction, label='Prediction')
plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Plotting Train vs Test vs Predicted Crime rate')
plt.legend()
plt.show()
```

```
# calculate error
test['arima_error'] = test['Crime Count'] - prediction['Predicted Crime Count']

rmse = np.sqrt(np.mean(test.arima_error**2)).round(2)
mape = np.round(np.mean(np.abs(100*(test.arima_error/test['Crime Count']))), 0))
mae=np.mean(np.abs(test.arima_error))
r2 = r2_score(test['Crime Count'], prediction['Predicted Crime Count'])
print('RMSE = $', rmse)
print('mae: %f' % mae)
print('Test R2: %.3f' % r2)
```

Figure 7: ARIMA Modelling

number of layers are 1 as shown in Figure 10. This study has noticed that loss value remains same after 13th iteration. Actuals vs Predicts plot is mentioned in Figure 8.

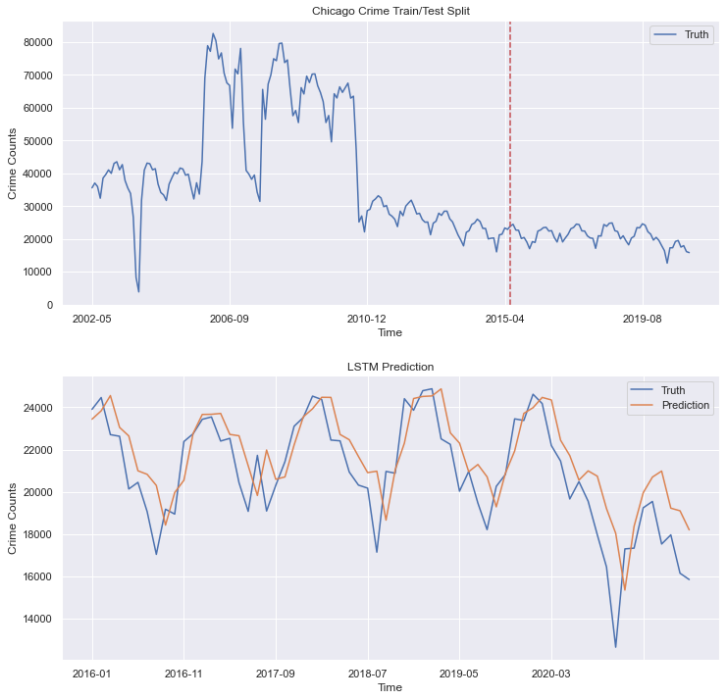


Figure 8: Actual vs Prediction plot

3.3.4 Random Forest Regression

Another model for rpredicting time series of crime in Chicago is random forest regression. Grouping of crime data is done by year and month value. First we need to convert our time series dataset into a supervised machine learning dataset [5] then splitting of data into train and test dataset has done. This study applied random forest regression using RandomForestRegressor method present in sklearn.ensemble library in python as shown in Figure 11 Plot for Actual vs prediction value has mentioned in Figure 12.

3.3.5 Moving Average Models

Moving Average model was performed on Chicago crime prediction on the basis of monthly, weekly and daily crime data of different districts present in Chicago city as shown in Figure 17. A moving average (MA) is a indicator that is frequently used in technical analysis. It creates a continuously updated average to assist smooth out data. There are three types of moving average performed in this study. First is Simple Moving average as shown in Figure 17. then weighted moving average has performed as shown in Figure 14. and at last exponential moving average has performed as mentioned in Figure 15 over crime date distributed as district wise.

[5]https://machinelearningmastery.com/random-forest-for-time-series-forecasting/

```
sns.set_theme(style="darkgrid")
fig, ax = plt.subplots(figsize=(12, 6))
# Plot the responses for different events and regions
sns.lineplot(data=training_set, linewidth=2, ax=ax, legend=None)
ax.set_xlabel('Months')
ax.set_xticklabels(['2001-05','2002-05','2006-07','2010-09','2014-12','2019-02','2021-05'])
ax.set_ylabel('Crime Counts')
ax.set_title('Chicago Crime Counts Time Series')
```

```
def sliding_windows_DataLoader(data, seq_length):
    x = []
    y = []

    for i in range(len(data)-seq_length-1):
        _x = data[i:(i+seq_length)]
        _y = data[i+seq_length]
        x.append(_x)
        y.append(_y)
        # print("X",x,"\n\n\ny",y)
    return np.array(x),np.array(y)
from sklearn.preprocessing import MinMaxScaler
# Data Normalization using MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_data = sc.fit_transform(training_set)

#Sliding Function
seq_length = 6
x, y = sliding_windows_DataLoader(training_data, seq_length)

# train / test split
train_size = int(len(y) * 0.70)
test_size = len(y) - train_size
print('train_size:',train_size,' test_size:',test_size)

# ndarray => tensor
dataX = torch.Tensor(np.array(x))
dataY = torch.Tensor(np.array(y))


trainX, trainY = torch.Tensor(np.array(x[0:train_size])), torch.Tensor(np.array(y[0:train_size]))
testX, testY = torch.Tensor(np.array(x[train_size:len(x)])), torch.Tensor(np.array(y[train_size:len(y)]))
```

Figure 9: LSTM Modelling

```
        return out
```

In [ ]:
```
class Optimization:
    def __init__(self, model, loss_fn, optimizer):
        self.model = model
        self.loss_fn = loss_fn
        self.optimizer = optimizer
        self.train_losses = []
        self.val_losses = []
    def train_step(self, x, y):
        # Sets model to train mode
        self.model.train()
        # Makes predictions
        yhat = self.model(x)
        # Computes loss
        loss = self.loss_fn(y, yhat)
        # Computes gradients
        loss.backward()
        # Updates parameters and zeroes gradients
        self.optimizer.step()
        self.optimizer.zero_grad()
        # Returns the loss
        return loss.item()
```

In [ ]:
```
num_epochs = 2500
learning_rate = 0.01

input_size = 1
hidden_size = 4
num_layers = 1

num_classes = 1

lstm = LSTM(num_classes, input_size, hidden_size, num_layers)

criterion = torch.nn.MSELoss()    # mean-squared error for regression
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)
#optimizer = torch.optim.SGD(lstm.parameters(), lr=learning_rate)


# Train the model
for epoch in range(num_epochs):
    outputs = lstm(trainX)
    optimizer.zero_grad()



    # obtain the loss function
    loss = criterion(outputs, trainY)

    loss.backward()

    optimizer.step()
    if epoch % 100 == 0:
      print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))
```

In [ ]:
```
# Predict on testing dataset

with torch.no_grad():
  lstm.eval()
  train_predict = lstm(dataX)
```

Figure 10: LSTM Prediction

```
In [16]: # transform a time series dataset into a supervised learning dataset
         def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
             n_vars = 1 if type(data) is list else data.shape[1]
             df = DataFrame(data)
             cols = list()
             # input sequence (t-n, ... t-1)
             for i in range(n_in, 0, -1):
                 cols.append(df.shift(i))
             # forecast sequence (t, t+1, ... t+n)
             for i in range(0, n_out):
                 cols.append(df.shift(-i))
             # put it all together
             agg = concat(cols, axis=1)
             # drop rows with NaN values
             if dropnan:
                 agg.dropna(inplace=True)
             return agg.values

In [17]: # split a univariate dataset into train/test sets
         def train_test_split(data, n_test):
             return data[:-n_test, :], data[-n_test:, :]

In [18]: # fit an random forest model and make a one step prediction
         def random_forest_forecast(train, testX):
             # transform list into array
             train = asarray(train)
             # split into input and output columns
             trainX, trainy = train[:, :-1], train[:, -1]
             # fit model
             model = RandomForestRegressor(n_estimators=1000)
             model.fit(trainX, trainy)
             # make a one-step prediction
             yhat = model.predict([testX])
             return yhat[0]

In [39]: # walk-forward validation for univariate data
         def walk_forward_validation(data, n_test):
             predictions = list()
             # split dataset
             train, test = train_test_split(data, n_test)
             # seed history with training dataset
             history = [x for x in train]
             # step over each time-step in the test set
             for i in range(len(test)):
                 # split test row into input and output columns
                 testX, testy = test[i, :-1], test[i, -1]
                 # fit model on history and make a prediction
                 yhat = random_forest_forecast(history, testX)
                 # store forecast in list of predictions
                 predictions.append(yhat)
                 # add actual observation to history for the next loop
                 history.append(test[i])
                 # summarize progress
                 print('>expected=%.1f, predicted=%.1f' % (testy, yhat))
             # estimate prediction error
             error = mean_absolute_error(test[:, -1], predictions)
             rmse = sqrt(mean_squared_error(test[:, -1], predictions))
             return error, test[:, -1], predictions,rmse
```

Figure 11: Random Forest Regression Modelling

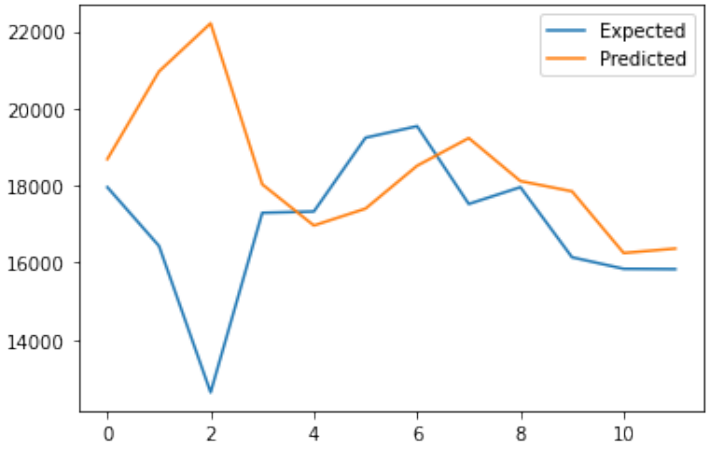
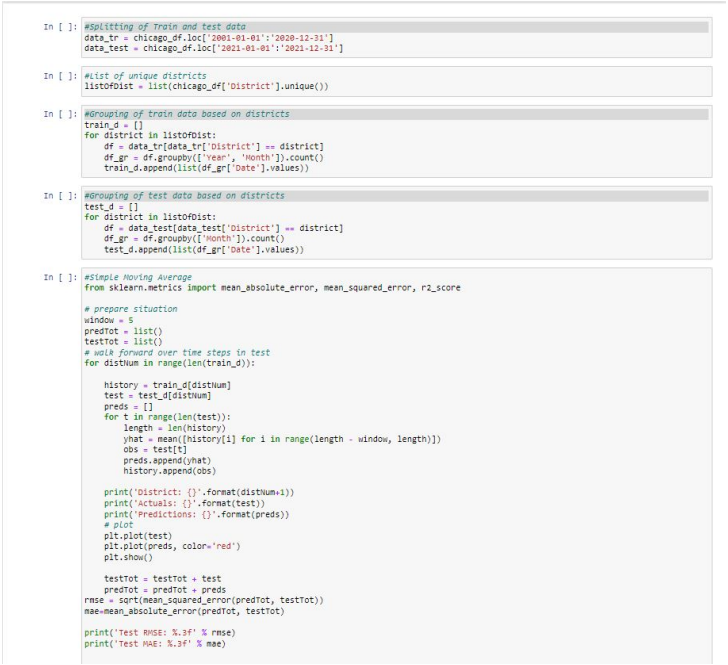


Figure 12: Random Forest Actual vs Prediction plot



```
In [ ]: #Splitting of Train and test data
        data_tr = chicago_df.loc['2001-01-01':'2020-12-31']
        data_test = chicago_df.loc['2021-01-01':'2021-12-31']

In [ ]: #List of unique districts
        listOfDist = list(chicago_df['District'].unique())

In [ ]: #Grouping of train data based on districts
        train_d = []
        for district in listOfDist:
            df = data_tr[data_tr['District'] == district]
            df_gr = df.groupby(['Year', 'Month']).count()
            train_d.append(list(df_gr['Date'].values))

In [ ]: #Grouping of test data based on districts
        test_d = []
        for district in listOfDist:
            df = data_test[data_test['District'] == district]
            df_gr = df.groupby(['Month']).count()
            test_d.append(list(df_gr['Date'].values))

In [ ]: #Simple Moving Average
        from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

        # prepare situation
        window = 5
        predTot = list()
        testTot = list()
        # walk forward over time steps in test
        for distNum in range(len(train_d)):

            history = train_d[distNum]
            test = test_d[distNum]
            preds = []
            for t in range(len(test)):
                length = len(history)
                yhat = mean([history[i] for i in range(length - window, length)])
                obs = test[t]
                preds.append(yhat)
                history.append(obs)

            print('District: {}'.format(distNum+1))
            print('Actuals: {}'.format(test))
            print('Predictions: {}'.format(preds))
            # plot
            plt.plot(test)
            plt.plot(preds, color='red')
            plt.show()

            testTot = testTot + test
            predTot = predTot + preds
        rmse = sqrt(mean_squared_error(predTot, testTot))
        mae=mean_absolute_error(predTot, testTot)

        print('Test RMSE: %.3f' % rmse)
        print('Test MAE: %.3f' % mae)
```

Figure 13: Simple Moving Average model for Monthly Prediction

```
#Weighted Moving Average
# prepare situation
window = 5
predTot = list()
testTot = list()
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in range(len(test)):
        length = len(history)
        yhat = np.average([history[i] for i in range(length - window, length)], weights=[1,2,3,4,5])
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds))
    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 14: Weighted Moving Average model for month wise prediction

```
In [ ]: #Exponential Moving Average
# prepare situation
predTot = list()
testTot = list()
alpha = 0.6
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    lastPred = 0
    for t in range(len(test)):
        yhat = ((1-alpha)*lastPred + (alpha*history[-1]))
        lastPred = yhat
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds))

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 15: Exponential Moving Average model for month wise prediction

District wise predictions for a week

```
In [104]: data_tr = chicago_df.loc['2001-01-01':'2020-12-31']
          data_test = chicago_df.loc['2021-01-01':'2021-12-31']

In [105]: listOfDist = list(chicago_df['District'].unique())

In [106]: train_d = []
          for district in listOfDist:
              df = data_tr[data_tr['District'] == district]
              df_gr = df.groupby(['Year', 'weekOfYear']).count()
              train_d.append(list(df_gr['Date'].values))

In [107]: test_d = []
          for district in listOfDist:
              df = data_test[data_test['District'] == district]
              df_gr = df.groupby(['weekOfYear']).count()
              test_d.append(list(df_gr['Date'].values))
```

Simple Moving Average

```
In [108]: # prepare situation
window = 5
predTot = list()
testTot = list()
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in range(len(test)):
        length = len(history)
        yhat = mean([history[i] for i in range(length - window, length)])
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 16: Simple Moving Average model for week wise Prediction

Weighted Moving Average

```
# prepare situation
window = 5
predTot = list()
testTot = list()
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in range(len(test)):
        length = len(history)
        yhat = np.average([history[i] for i in range(length - window, length)], weights=[1,2,3,4,5])
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

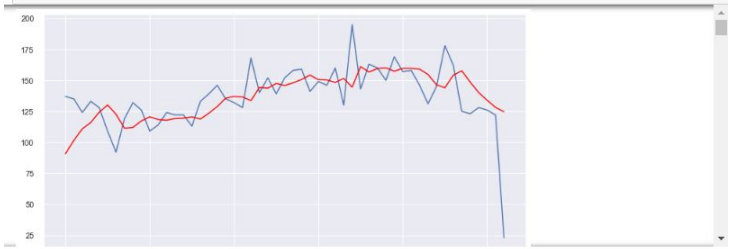


Figure 17: Weighted Moving Average model for Week wise Prediction

Exponential moving average

In [110]:

```
# prepare situation
predTot = list()
testTot = list()
alpha = 0.6
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    lastPred = 0
    for t in range(len(test)):
        yhat = ((1-alpha)*lastPred + (alpha*history[-1]))
        lastPred = yhat
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

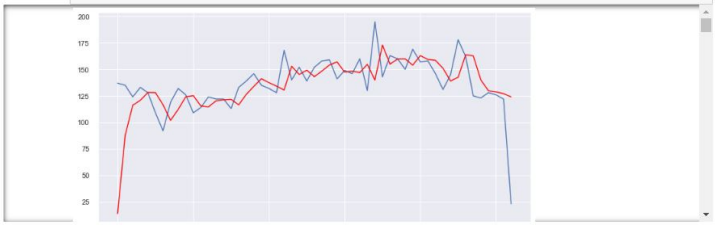
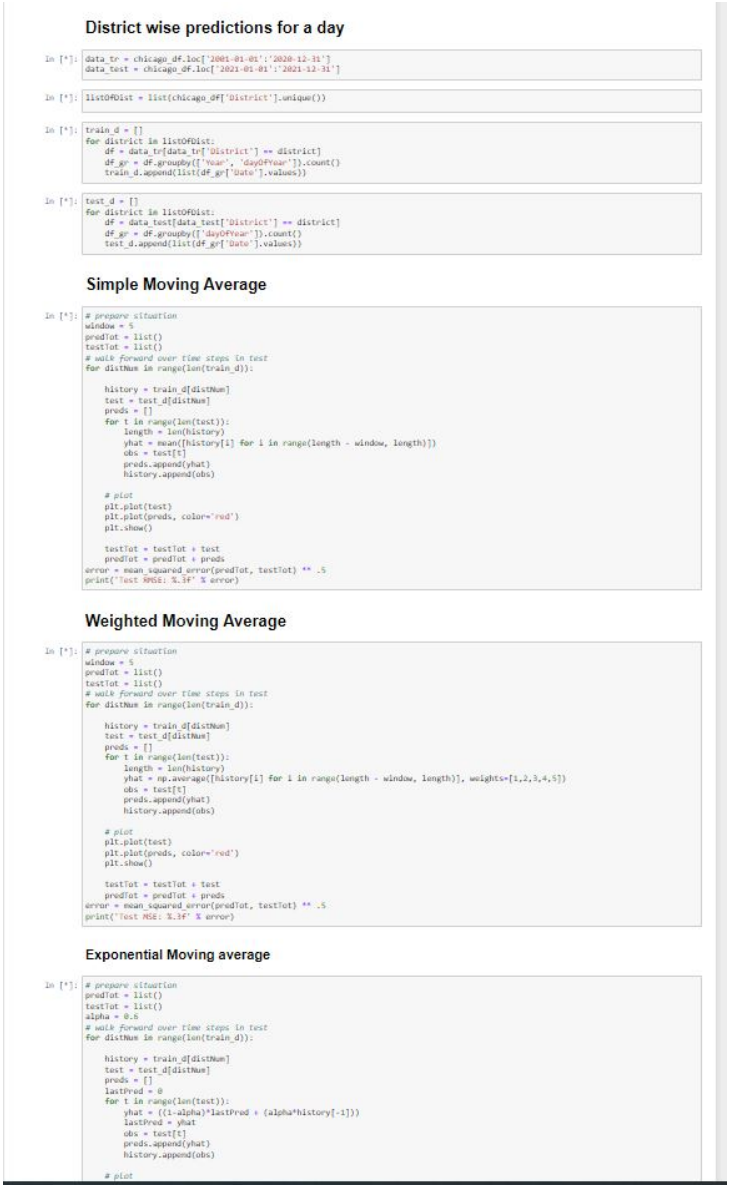


Figure 18: Exponential Moving Average model for Week wise Prediction



District wise predictions for a day

```
data_tr = chicago_df.loc['2001-01-01':'2020-12-31']
data_test = chicago_df.loc['2021-01-01':'2021-12-31']
```

```
listOfDist = list(chicago_df['District'].unique())
```

```
train_d = []
for district in listOfDist:
    df = data_tr[data_tr['District'] == district]
    df_gr = df.groupby(['Year', 'dayOfYear']).count()
    train_d.append(list(df_gr['Date'].values))
```

```
test_d = []
for district in listOfDist:
    df = data_test[data_test['District'] == district]
    df_gr = df.groupby(['dayOfYear']).count()
    test_d.append(list(df_gr['Date'].values))
```

Simple Moving Average

```
# prepare situation
window = 5
predTot = list()
testTot = list()
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in range(len(test)):
        length = len(history)
        yhat = mean([history[i] for i in range(length - window, length)])
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
error = mean_squared_error(predTot, testTot) ** .5
print('Test RMSE: %.3f' % error)
```

Weighted Moving Average

```
# prepare situation
window = 5
predTot = list()
testTot = list()
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in range(len(test)):
        length = len(history)
        yhat = np.average([history[i] for i in range(length - window, length)], weights=[1,2,3,4,5])
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
error = mean_squared_error(predTot, testTot) ** .5
print('Test MSE: %.3f' % error)
```

Exponential Moving average

```
# prepare situation
predTot = list()
testTot = list()
alpha = 0.5
# walk forward over time steps in test
for distNum in range(len(train_d)):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    lastPred = 0
    for t in range(len(test)):
        yhat = ((1-alpha)*lastPred + (alpha*history[-1]))
        lastPred = yhat
        obs = test[t]
        preds.append(yhat)
        history.append(obs)

    # plot
```

Figure 19: Simple/Weighted/Exponential Moving Average model for day wise Prediction

3.3.6 LSTM Models for district wise prediction

LSTM models are performed to get more accurate prediction of Chicago crime rate on the basis of different districts present in Chicago city. Three types of LSTM models has performed in this study which are unidirectional LSTM as shown in Figure 23, bi-directional lSTM model as shown in Figure 24 and CNN LSTM model as shown in Figure 25

```
#unidirectional LSTM for month wise prediction
# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        # define model
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds))

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 20: Uni-directional LSTM model for month wise Prediction

```
#bi-directional LSTM for month wise prediction
# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        model = Sequential()
        model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    preds_new = [i[0] for i in preds]
    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds_new))

    # plot
    plt.plot(test)
    plt.plot(preds_new, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds_new
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 21: bi-directional LSTM model for month wise prediction

```
In [ ]: #CNN-LSTM for month wise prediction
        # prepare situation
        window = 4
        predTot = list()
        testTot = list()

        # walk forward over time steps in test
        for distNum in tqdm_notebook(range(len(train_d))):

            history = train_d[distNum]
            test = test_d[distNum]
            preds = []
            for t in tqdm_notebook(range(len(test)), leave=False):

                length = len(history)

                # split into samples
                X, y = split_sequence(history, window)

                # reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
                n_features = 1
                n_seq = 2
                n_steps = 2
                X = X.reshape((X.shape[0], n_seq, n_steps, n_features))

                model = Sequential()
                model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(None, n_steps, n_features)))
                model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
                model.add(TimeDistributed(Flatten()))
                model.add(LSTM(50, activation='relu'))
                model.add(Dense(1))
                model.compile(optimizer='adam', loss='mse')

                # fit model
                model.fit(X, y, epochs=500, verbose=0)

                X_test = array([history[i] for i in range(length-window, length)])
                X_test = X_test.reshape((1, n_seq, n_steps, n_features))
                yhat = model.predict(X_test, verbose=0)

                obs = test[t]
                preds.append(yhat.reshape((1,)))
                history.append(obs)

            preds_new = [i[0] for i in preds]
            print('District: {}'.format(distNum+1))
            print('Actuals: {}'.format(test))
            print('Predictions: {}'.format(preds_new))

            # plot
            plt.plot(test)
            plt.plot(preds_new, color='red')
            plt.show()

            testTot = testTot + test
            predTot = predTot + preds_new
        rmse = sqrt(mean_squared_error(predTot, testTot))
        mae=mean_absolute_error(predTot, testTot)

        print('Test RMSE: %.3f' % rmse)
        print('Test MAE: %.3f' % mae)
```

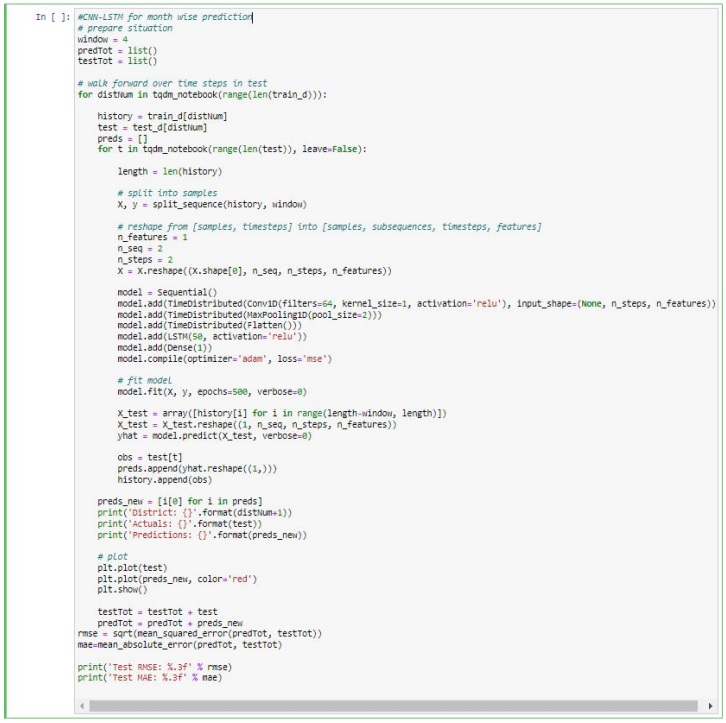
Figure 22: CNN-LSTM model for month wise prediction

```
#unidirectional LSTM for month wise prediction
# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        # define model
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds))

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 23: Uni-directional LSTM model for week wise Prediction

```
#bi-directional LSTM for month wise prediction
# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        model = Sequential()
        model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    preds_new = [i[0] for i in preds]
    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds_new))

    # plot
    plt.plot(test)
    plt.plot(preds_new, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds_new
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 24: bi-directional LSTM model for week wise prediction



```
In [ ]: #CNN-LSTM for month wise prediction
        # prepare situation
        window = 4
        predTot = list()
        testTot = list()

        # walk forward over time steps in test
        for distNum in tqdm_notebook(range(len(train_d))):

            history = train_d[distNum]
            test = test_d[distNum]
            preds = []
            for t in tqdm_notebook(range(len(test)), leave=False):

                length = len(history)

                # split into samples
                X, y = split_sequence(history, window)

                # reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
                n_features = 1
                n_seq = 2
                n_steps = 2
                X = X.reshape((X.shape[0], n_seq, n_steps, n_features))

                model = Sequential()
                model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(None, n_steps, n_features))
                model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
                model.add(TimeDistributed(Flatten()))
                model.add(LSTM(50, activation='relu'))
                model.add(Dense(1))
                model.compile(optimizer='adam', loss='mse')

                # fit model
                model.fit(X, y, epochs=500, verbose=0)

                X_test = array([history[i] for i in range(length-window, length)])
                X_test = X_test.reshape((1, n_seq, n_steps, n_features))
                yhat = model.predict(X_test, verbose=0)

                obs = test[t]
                preds.append(yhat.reshape((1,)))
                history.append(obs)

            preds_new = [i[0] for i in preds]
            print('District: {}'.format(distNum+1))
            print('Actuals: {}'.format(test))
            print('Predictions: {}'.format(preds_new))

            # plot
            plt.plot(test)
            plt.plot(preds_new, color='red')
            plt.show()

            testTot = testTot + test
            predTot = predTot + preds_new
        rmse = sqrt(mean_squared_error(predTot, testTot))
        mae=mean_absolute_error(predTot, testTot)

        print('Test RMSE: %.3f' % rmse)
        print('Test MAE: %.3f' % mae)
```

Figure 25: CNN-LSTM model for week wise prediction

```
#Uni-directional LSTM for weekly prediction
# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        # define model
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds))

    # plot
    plt.plot(test)
    plt.plot(preds, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 26: Uni-directional LSTM model for day wise Prediction

```
#Bi-directional LSTM for weekly prediction

# prepare situation
window = 3
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X = X.reshape((X.shape[0], X.shape[1], n_features))

        model = Sequential()
        model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(window, n_features)))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=200, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, window, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    preds_new = [i[0] for i in preds]
    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds_new))

    # plot
    plt.plot(test)
    plt.plot(preds_new, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds_new
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 27: bi-directional LSTM model for day wise prediction

```
In [ ]: #CNN LSTM for weekly prediction

# prepare situation
window = 4
predTot = list()
testTot = list()

# walk forward over time steps in test
for distNum in tqdm_notebook(range(len(train_d))):

    history = train_d[distNum]
    test = test_d[distNum]
    preds = []
    for t in tqdm_notebook(range(len(test)), leave=False):

        length = len(history)

        # split into samples
        X, y = split_sequence(history, window)

        # reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
        n_features = 1
        n_seq = 2
        n_steps = 2
        X = X.reshape((X.shape[0], n_seq, n_steps, n_features))

        model = Sequential()
        model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(None, n_steps, n_features)))
        model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
        model.add(TimeDistributed(Flatten()))
        model.add(LSTM(50, activation='relu'))
        model.add(Dense(1))
        model.compile(optimizer='adam', loss='mse')

        # fit model
        model.fit(X, y, epochs=500, verbose=0)

        X_test = array([history[i] for i in range(length-window, length)])
        X_test = X_test.reshape((1, n_seq, n_steps, n_features))
        yhat = model.predict(X_test, verbose=0)

        obs = test[t]
        preds.append(yhat.reshape((1,)))
        history.append(obs)

    preds_new = [i[0] for i in preds]
    print('District: {}'.format(distNum+1))
    print('Actuals: {}'.format(test))
    print('Predictions: {}'.format(preds_new))

    # plot
    plt.plot(test)
    plt.plot(preds_new, color='red')
    plt.show()

    testTot = testTot + test
    predTot = predTot + preds_new
rmse = sqrt(mean_squared_error(predTot, testTot))
mae=mean_absolute_error(predTot, testTot)

print('Test RMSE: %.3f' % rmse)
print('Test MAE: %.3f' % mae)
```

Figure 28: CNN-LSTM model for day wise prediction

References

Palanivinayagam, A., Gopal, S. S., Bhattacharya, S., Anumbe, N., Ibeke, E. and Biamba, C. (2021). An optimized machine learning and big data approach to crime detection, *Wireless Communications and Mobile Computing* **2021**.