Configuration Manual

MSc Research Project
Data Analytics

Ratna Pillai
Student ID: x18134297

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ratna Pillai
Student ID:	x18134297
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	1836
Page Count:	26

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:	12th December 2019

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):	

Configuration Manual

Ratna Pillai
x18134297
MSc Research Project in Data Analytics

12th December 2019

1 Introduction

This configuration manual specifies the hardware, software requirements and the programming phases of the implementation of the below research project in detail:

"Optimized Predictive Modelling to Unfold the Links of Crime with Education, Safety and Climate in Chicago"

2 System Configurations

2.1 Hardware

- **Processor:** Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
- **RAM:** 8 GB
- **System Type:** Windows OS, 64-bit
- **GPU:** Intel(R) UHD Graphics Family, 4GB
- **Storage:** 1 TB HDD

2.2 Software

- **Microsoft Excel 2016:** This spreadsheet tool offered by Microsoft has been used for storing the downloaded datasets in flat files as csv (comma separated values).
- **Anaconda Distribution-Jupyter Notebook:** This is an open source software which has been downloaded from the anaconda distribution website.[1] This distribution support platforms like jupyter notebooks, spyder, R studio to run machine learning models using R or python. Exploratory data analysis, manipulation of data, pre-processing, transformation and visualizations in this study were done using Python (version 3.6.5) on Jupyter notebooks using this distribution.

[1]https://www.anaconda.com/distribution/

- **Google Colaboratory:** Also called Colab, this is a free cloud service that aids the users with free GPU services to run machine learning models on an environment similar to Jupyter notebooks. For this study, colab is used for modelling and hyper parameter optimization. For enabling GPU settings, from the Runtime menu on the notebook screen, select change runtime type to GPU as shown in Figure 1. There is also an additional option to change it to TPU, which is efficient to use in cases of high data volume.

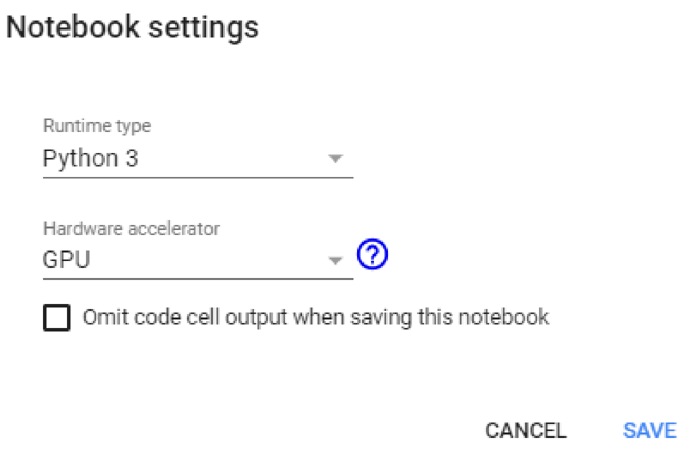

Figure 1: GPU Settings: Google Colaboratary

- **Power BI Desktop:** Part of Visualizations were done using Microsoft Power BI desktop app. This software is downloaded from Microsoft store website.[2]

3 Project Development

Implementation of this research work is entirely done using Python programming. Initial stages of the research project development includes data pre-processing, merging all the datasets in consideration, normalization and one hot encoding. Followed by the data preparation activities, predictive modelling is done using machine learning techniques in python using the sk-learn (scikit-learn) and keras libraries.

3.1 Data Preparation

Importing of the datasets and data manipulation is done using the pandas (dataframe) and numpy (arrays) libraries. Following sections detail the pre-processing steps carried out in each dataset.

3.1.1 Crime Dataset

Pre-processing of crime data primarily involved fixing the missing latitudes and longitudes as shown in Figure 2. Block addresses (which was already clean when the data was downloaded from the source) of the missing location co-ordinates were passed to the

[2]https://www.microsoft.com/en-in/p/power-bi-desktop/9ntxr16hnw1t?activetab=pivot:overviewtab

wrapper censusbatchgeocoder. This function contacts the geocodes API (Application Programming Interface) in the US (United States) and returns a dictionary of address detail for a particular address. This dictionary is converted in to a new dataframe and the missing values are filled using lookup and fill_na (Not Applicable) functions.

```python
import censusbatchgeocoder
#Extract only null rows for location & zip codes
locnull = narcotics[narcotics['Latitude'].isnull()]
locnull.shape
city = 'Chicago'
state = 'IL'
address = locnull['Block']
address = address.drop_duplicates()
addressdata = pd.DataFrame(address)
addressdata['city'] = city
addressdata['state'] = state
addressdata.columns = ['address','city','state']
addressdata = addressdata.reset_index()
addressdata['id'] = np.arange(1,len(addressdata)+1)
addressdata = addressdata.drop(columns=['index'])
#print (addressdata)
#Rearrange columns - addressdata
cols = list(addressdata.columns)
a, b = cols.index('address'), cols.index('id')
cols[b], cols[a] = cols[a], cols[b]
addressdata = addressdata[cols]
addressdata.head()
fetchaddress = addressdata.to_dict("records")
results = censusbatchgeocoder.geocode(fetchaddress.to_dict("records"),zipcode=None)
#Lookup for zipcode
narcotics.Zip_Codes.replace('NaN', np.NaN, inplace=True)
narcotics.loc[narcotics['Zip_Codes'].isnull(),'Zip_Codes'] = narcotics['Block'].map(locdata.ZipCode)
#Lookup for latitude
narcotics.loc[narcotics['Latitude'].isna(),'Latitude'] = narcotics['Block'].map(locdata.LAT)
#Lookup for longitude
narcotics.loc[narcotics['Longitude'].isna(),'Longitude'] = narcotics['Block'].map(locdata.LON)
#Fill missing Zipcodes
s = locdata.set_index('Block')['ZipCode']
narcotics['Zip_Codes'] = narcotics['Zip_Codes'].fillna(narcotics['Block'].map(s))
```

Figure 2: Handling of missing latitudes and longitudes in Crime dataset

3.1.2 High Schools Dataset

Four datasets for high school report (academic years 2015-2018) were downloaded from Chicago data portal and saved as csv (comma separated values) files. Each year school data was complex enough with more than 150 attributes and selection of relevant attributes was a challenge. Also, there were missing values which had to be handled as shown in Figure 3. Since the data was maintained school wise, the median value was used to replace the missing values in school data.

```
#2018 edu data
edurefined = edueight[['Long_Name','Community_Area','School_Latitude','School_Longitude','Primary_Category',
'Teacher Attendance Year 1 Pct','Teacher Attendance Year 2 Pct','Student Attendance Year 1 Pct',
'Suspensions_Per_100_Students_Year_1_Pct','Suspensions_Per_100_Students_Year_2_Pct',
'Average_Length_Suspension_Year_1_Pct','Average_Length_Suspension_Year_2_Pct','Mobility_Rate_Pct',
'Chronic Truancy Pct','One Year Dropout Rate Year 1 Pct',
#Compute average misconduct to suspensions rate
edurefined['Avg_Misconduct_Rate'] = (edurefined['Misconducts_To_Suspensions_Year_1_Pct']
                                     + edurefined['Misconducts_To_Suspensions_Year_2_Pct'])/2
#Compute average suspension rate
edurefined['Avg Suspension Rate'] = (edurefined['Suspensions Per 100 Students Year 1 Pct']
#Compute average attendance rate
edurefined['Avg Student Attendance Rate'] = (edurefined['Student Attendance Year 1 Pct']
#Compute average teacher attendance rate
edurefined['Avg_Teacher_Attendance_Rate'] = (edurefined['Teacher_Attendance_Year_1_Pct']
                                             + edurefined['Teacher_Attendance_Year_2_Pct'])/2
edurefined['Average_Length_Suspension_Year_1_Pct'] = edurefined['Average_Length_Suspension_Year_1_Pct'].astype(str)
edurefined['Average_Length_Suspension_Year_2_Pct'] = edurefined['Average_Length_Suspension_Year_2_Pct'].astype(str)
edurefined['Average_Length_Suspension_Year_1_Pct'] = edurefined['Average_Length_Suspension_Year_1_Pct'].str.replace(' days','')
edurefined['Average_Length_Suspension_Year_2_Pct'] = edurefined['Average_Length_Suspension_Year_2_Pct'].str.replace(' days','')
edurefined['Average_Length_Suspension_Year_1_Pct'] = edurefined['Average_Length_Suspension_Year_1_Pct'].astype(float)
edurefined['Average_Length_Suspension_Year_2_Pct'] = edurefined['Average_Length_Suspension_Year_2_Pct'].astype(float)
edurefined['Avg_Suspension_Days'] = (edurefined['Average_Length_Suspension_Year_2_Pct']
                                     + edurefined['Average_Length_Suspension_Year_1_Pct'])/2
edurefined.head()
#Replace missing values with median
edurefined['Avg Misconduct Rate'].fillna((edurefined['Avg Misconduct Rate'].median()), inplace=True)
#Convert location c-ordinates to geohash
import pygeohash as pgh
edurefined['geohash'] =  edurefined.apply(lambda x: pgh.encode(x.School_Latitude, x.School_Longitude, precision=5), axis=1)
#Group by geohash
edu_df = edurefined.groupby('geohash').mean().reset_index()
```

Figure 3: High school education data pre-processing

- Firstly, only numerical attributes were considered from each education dataset. Suspension days column consisted of "days" keyword appended to number of days which was cleaned to retain only the number and "days" keyword was removed. Then, a check for missing value was done and filled using median value for that column.
- Since the misconduct rate, suspension rate, enrollment rate, freshman track rate were all maintained academic year wise (For eg. academic year 2015-16 data was stored in the file as suspension days year 1 and suspension days year 2), these columns were combined together using mean of both the columns and stored as Average Suspension Days.
- A similar approach was followed for the remaining three years data and the latitude and longitude of each school was converted to geohash using pygeohash library in python.
- Lastly, all the pre-processed data are merged and saved in a flat file in a csv format.

3.1.3 Locations Dataset

- Locations dataset identified were three namely, police station, speed camera locations and red light camera locations in Chicago. As these datasets comprised of latitude and longitude co-ordinates which were primarily required for this research, presence of missing values were checked as highlighted in Figure 4.

```python
#Extract only required columns from police station, red-light camera and speed camera datasets
police = police[['LATITUDE','LONGITUDE']]
red = red[['LATITUDE','LONGITUDE']]
speed = speed[['LATITUDE','LONGITUDE']]
#Check for NA values in each dataset
red.isna().sum()
police.isna().sum()
speed.isna().sum()
#Verify the shape of each dataframe
print(red.shape)
print(speed.shape)
print(police.shape)
```

Figure 4: Locations dataset - Missing values check

- Nearest distance of police stations, red light cameras and speed cameras was calculated using the user defined distance function as denoted in Figure 5 for each of the crime instance.[3]

```python
#Calculate distance between two latitudes and longitudes
import math
def distance(origin, destination):
    lat1, lon1 = origin
    lat2, lon2 = destination
    radius = 6371 # km
    dlat = math.radians(lat2-lat1)
    dlon = math.radians(lon2-lon1)
    a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
        * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = round(radius * c,2)
return d
```

Figure 5: Function to compute distance between two latitude and longitude co-ordinates

- Compute nearest speed camera distance from crime location For each speed camera location as highlighted in Figure 6, the latitudes and longitudes were used in the distance computation with the crime location (latitude and longitude). Then minimum distance for that crime instance was computed using min() function in python. Similarly, the code was run for the red-light camera locations and police stations as shown in Figure 7 and Figure 8.

[3] https://gist.github.com/rochacbruno/2883505

```
import datetime
#Calculate distance between two points and return minimum for speed camera locations
distanceall = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distancemin = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distanceall['dist'] = None
distancemin['dist'] = None
print(datetime.datetime.now())
for i in range(0,len(crime),1):
    print(i)
    for j in range(0,len(speed),1):
        dist = distance((crime.iloc[i]['Latitude'].astype(float),crime.iloc[i]['Longitude'].astype(float)),
                        (speed.iloc[j]['LATITUDE'].astype(float),speed.iloc[j]['LONGITUDE'].astype(float)))
        lat = crime.iloc[i]['Latitude']
        lon = crime.iloc[i]['Longitude']
        distanceall.loc[j,'dist'] = dist
        distanceall.loc[j,'latitude'] = lat
        distanceall.loc[j,'longitude'] = lon
        distanceall.loc[j,'id'] = i
        d = len(distanceall)
        if j == (len(speed)-1):
            mindist = distanceall['dist'].min()
    distancemin.loc[i,'dist'] = mindist
    distancemin.loc[i,'latitude'] = lat
    distancemin.loc[i,'longitude'] = lon
    distancemin.loc[i,'id'] = i
print(datetime.datetime.now())
```

Figure 6: Computation of nearest speed camera distance

- Compute nearest Red-light camera distance from crime location

```
#Calculate distance between two points and return minimum for red-light camera locations
distanceall = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distancemin = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distanceall['dist'] = None
distancemin['dist'] = None
print(datetime.datetime.now())
for i in range(0,len(crime),1):
    print(i)
    for j in range(0,len(speed),1):
        dist = distance((crime.iloc[i]['Latitude'].astype(float),crime.iloc[i]['Longitude'].astype(float)),
                        (red.iloc[j]['LATITUDE'].astype(float),red.iloc[j]['LONGITUDE'].astype(float)))
        lat = crime.iloc[i]['Latitude']
        lon = crime.iloc[i]['Longitude']
        distanceall.loc[j,'dist'] = dist
        distanceall.loc[j,'latitude'] = lat
        distanceall.loc[j,'longitude'] = lon
        distanceall.loc[j,'id'] = i
        d = len(distanceall)
        if j == (len(speed)-1):
            mindist = distanceall['dist'].min()
    distancemin.loc[i,'dist'] = mindist
    distancemin.loc[i,'latitude'] = lat
    distancemin.loc[i,'longitude'] = lon
    distancemin.loc[i,'id'] = i
print(datetime.datetime.now())
```

Figure 7: Computation of nearest red light camera distance

- Compute nearest police station distance from crime location

```python
#Calculate distance between two points and return minimum for police stations
distanceall = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distancemin = pd.DataFrame(columns=['id','dist','latitude','longitude'])
distanceall['dist'] = None
distancemin['dist'] = None
print(datetime.datetime.now())
for i in range(0,len(crime),1):
    print(i)
    for j in range(0,len(speed),1):
        dist = distance((crime.iloc[i]['Latitude'].astype(float),crime.iloc[i]['Longitude'].astype(float)),
                        (police.iloc[j]['LATITUDE'].astype(float),police.iloc[j]['LONGITUDE'].astype(float)))
        lat = crime.iloc[i]['Latitude']
        lon = crime.iloc[i]['Longitude']
        distanceall.loc[j,'dist'] = dist
        distanceall.loc[j,'latitude'] = lat
        distanceall.loc[j,'longitude'] = lon
        distanceall.loc[j,'id'] = i
        d = len(distanceall)
        if j == (len(speed)-1):
            mindist = distanceall['dist'].min()
    distancemin.loc[i,'dist'] = mindist
    distancemin.loc[i,'latitude'] = lat
    distancemin.loc[i,'longitude'] = lon
    distancemin.loc[i,'id'] = i
print(datetime.datetime.now())
```

Figure 8: Computation of nearest speed camera distance

3.1.4 Weather Dataset

Daily weather data for the period 2015-2018 was scraped using (API) Application Programming Interface access from NCEI (formerly known as NOAA) portal. The data was scraped in 3 steps as followed by many data scientists as outlined in the medium portal[4]:

- Gained access to the API using token for authorization from NCEI web token site and accessed in python using requests library as demonstrated in Figure 11.[5]

Figure 9: Request the API access token

[4]https://towardsdatascience.com/getting-weather-data-in-3-easy-steps-8dc10cc5c859
[5]https://www.ncdc.noaa.gov/cdo-web/token

```python
#needed to make web requests
import requests
#store the data we get as a dataframe
import pandas as pd
#convert the response as a strcuctured json
import json
#mathematical operations on lists
import numpy as np
#parse the datetimes we get from NOAA
from datetime import datetime
#add the access token you got from NOAA
Token = 'XnvxAnqwtzCGLaepxKsuVihUwMhZkCti'
#Long Beach Airport station
station_id = 'GHCND:USW00094846'
```

Figure 10: API settings to access weather data

- Identified the relevant weather station for required data collection from the site.[6]

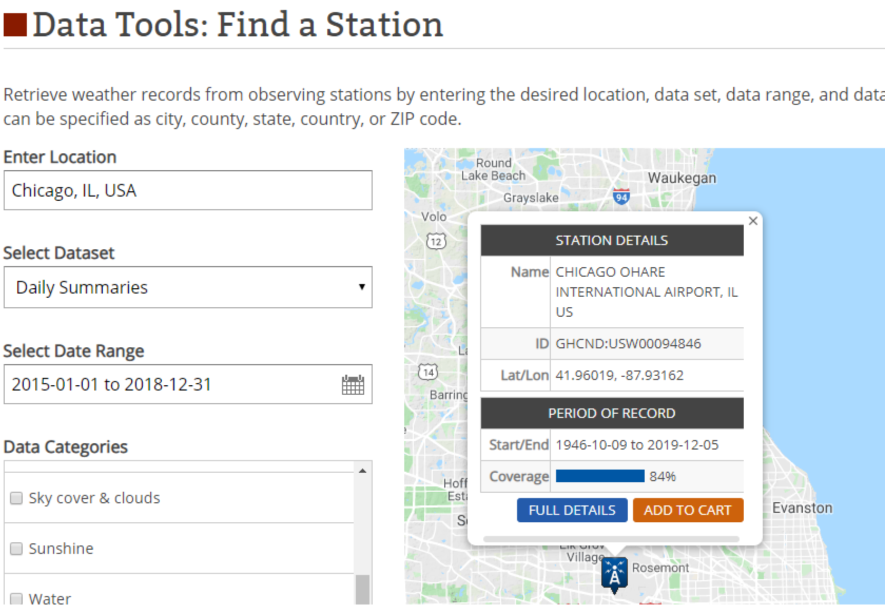

Figure 11: Find the relevant station

- With the help of python requests object, windspeed (wind), average temperature(avg_temp) and precipitation (prcp) values for three years on a daily basis was scraped. For each of the weather attribute, the datatype id was required to be modified in the URL of the request command.
- Average temperature data was fetched and stored in a dataframe. Refer Figure 12 for the python code.

[6]https://www.ncdc.noaa.gov/cdo-web/datatools/findstation

```python
#initialize lists to store data - Average tenperature (TAVG)
dates_temp = []
dates_prcp = []
temps = []
prcp = []
#for each year from 2015-2018 ...
for year in range(2015, 2019):
    year = str(year)
    print('working on year '+year)
    #make the api call
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND
    &datatypeid=TAVG&limit=1000
    &stationid=GHCND:USW00023129&startdate='+year+'-01-01
    &enddate='+year+'-12-31', headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are average temperature readings
    avg_temps = [item for item in d['results'] if item['datatype']=='TAVG']
    #get the date field from all average temperature readings
    dates_temp += [item['date'] for item in avg_temps]
    #get the actual average temperature from all average temperature readings
    temps += [item['value'] for item in avg_temps]
#initialize dataframe
df_temp = pd.DataFrame()
#populate date and average temperature fields (cast string date to datetime and convert temperature from tenths of Celsius to Fahrenh
df_temp['date'] = [datetime.strptime(d, "%Y-%m-%dT%H:%M:%S") for d in dates_temp]
df_temp['avgTemp'] = [float(v)/10.0*1.8 + 32 for v in temps]
```

Figure 12: Average temperature data scraping

- Average windspeed was fetched and stored in a dataframe.Refer Figure 13 for the python code.

```python
#initialize lists to store data - Average Wind Speed (AWND)
dates_temp = []
dates_prcp = []
temps = []
prcp = []
#for each year from 2015-2018 ...
for year in range(2015, 2019):
    year = str(year)
    print('working on year '+year)
    #make the api call
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND
    &datatypeid=AWND&limit=1000&stationid=GHCND:USW00023129
    &startdate='+year+'-01-01&enddate='+year+'-12-31', headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are average temperature readings
    avg_temps = [item for item in d['results'] if item['datatype']=='AWND']
    #get the date field from all average temperature readings
    dates_temp += [item['date'] for item in avg_temps]
    #get the actual average temperature from all average temperature readings
    temps += [item['value'] for item in avg_temps]
#initialize dataframe
df_wind = pd.DataFrame()
#populate date and average temperature fields (cast string date to datetime and convert temperature from tenths of Celsius to Fahrenh
df_wind['date'] = [datetime.strptime(d, "%Y-%m-%dT%H:%M:%S") for d in dates_temp]
df_wind['wind'] = [float(v)/10.0*1.8 + 32 for v in temps]
```

Figure 13: Average windspeed data scraping

- Precipitation was fetched and stored in a dataframe.Refer Figure 14 for the python code.

```
#initialize lists to store data - Precipitation (PRCP)
dates_temp = []
dates_prcp = []
temps = []
prcp = []
#for each year from 2015-2018 ...
for year in range(2015, 2019):
    year = str(year)
    print('working on year '+year)
    #make the api call
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND
    &datatypeid=PRCP&limit=1000&stationid=GHCND:USW00023129
    &startdate='+year+'-01-01&enddate='+year+'-12-31', headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are average temperature readings
    avg_temps = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average temperature readings
    dates_temp += [item['date'] for item in avg_temps]
    #get the actual average temperature from all average temperature readings
    temps += [item['value'] for item in avg_temps]
#initialize dataframe
df_prcp = pd.DataFrame()
#populate date and average temperature fields (cast string date to datetime and convert temperature from tenths of Celsius to Fahrenh
df_prcp['date'] = [datetime.strptime(d, "%Y-%m-%dT%H:%M:%S") for d in dates_temp]
df_prcp['prcp'] = [float(v)/10.0*1.8 + 32 for v in temps]
```

Figure 14: Average precipitation data scraping

- After scraping the required weather data in pandas dataframe, these dataframes were merged to form one dataframe and finally saved to csv. Also, the date parameter was split as Year, Month and Day for merging with crime data.

```
#Merge all weather scrapes
print(df_temp.shape)
print(df_prcp.shape)
print(df_wind.shape)
df_prcp.head()
weather = df_temp.merge(df_prcp,on=['date'])
weather = weather.merge(df_wind,on=['date'],how='left')
weather.head()
#Split date column to year, month and day
weather['date'] = pd.to_datetime(weather['date'])
weather['Month'] = weather['date'].dt.month
weather['Day'] = weather['date'].dt.day
weather['Year'] = weather['date'].dt.year
weather = weather.drop(columns=['date'])
weather.to_csv('C:\Data Analytics\Sem 3\Dataset\LocationData\Distances\weatherapi.csv',index=False)
```

Figure 15: Merge average temperature, precipitation and average wind speed data

3.1.5 Data Merging

All the location co-ordinates present in each dataset in the form of latitude and longitude were converted to geohash. Also, the date parameter was split in to Year, Month and Day in each dataset. Based on the relevant attribute, each dataset was grouped by geohash and merged with crime.

- **Merge Distances**
  The nearest distance computed for each crime instance was merged with crime dataset as shown in Figure 16.

```
#Convert latitude & longitude to geohash
import pygeohash as pgh
final['geohash'] = final.apply(lambda x: pgh.encode(x.Latitude, x.Longitude, precision=5), axis=1)
dist = final.copy()
final.shape
#Extract distances
distance = dist[['geohash','NearestPoliceDist','NearestSpeedCamDist','NearestRedCamDist']]
distance = distance.groupby(['geohash']).mean().reset_index()
distance.head()
distance = distance.round(2)
#Select required columns from crime dataset
hs = final[['geohash','Primary_Type','Year','Month','Day','WEEKDAY','Holiday','Time']]
hs.shape
#compute the crime rate for a geohash a latitude and longitude belongs to on a monthly basis
school = hs.groupby(['geohash','Primary_Type','Year','Month','WEEKDAY','Holiday','Time']).size().reset_inde
school.head()
schoolrate_df = school.rename(columns={0:'crimescount'})
schoolrate_df.head()
schoolrate_df = schoolrate_df.round(2) #Rounding to nearest place
#Merge distance data with crime
schoolrate_df = schoolrate_df.merge(distance,on=['geohash'],how='left')
schoolrate_df.shape #34440,11
schoolrate_df.columns
schoolrate_df.isna().sum()
schoolrate_df
```

Figure 16: Merge nearest distances with crime data

- **Merge Red-light and speed camera count**
  In addition to nearest distance computation, the count of red light and speed cameras in a geohash was done by grouping the location co-ordinates by geohash and merged with crime using a left join as shown in Figure 17. Left join was used because there is a possibility of a crime geohash presence with no cameras.

```
#Merge speed cams and red cams in that geohash area
scamslocs = pd.read_csv('C:/Data Analytics/Sem 3/Dataset/LocationData/Speed_Camera_Locations_withZip.cs
rcamlocs = pd.read_csv('C:/Data Analytics/Sem 3/Dataset/LocationData/Red_Camera_Locations_withZip.csv')
scamslocs.head()
rcamlocs.head()
#Convert location co-ordinates to geohash
scamslocs['geohash'] = scamslocs.apply(lambda x: pgh.encode(x.LATITUDE, x.LONGITUDE, precision=5), axis
rcamlocs['geohash'] = rcamlocs.apply(lambda x: pgh.encode(x.LATITUDE, x.LONGITUDE, precision=5), axis=1
#Find the count red light cameras and speed cameras in a geohash
slocs = scamslocs.groupby(['geohash']).size().reset_index()
rlocs = rcamlocs.groupby(['geohash']).size().reset_index()
#Rename columns
slocs = slocs.rename(columns={0:'SpeedCamCount'})
rlocs = rlocs.rename(columns={0:'RedCamCount'})
#Merge with crime data
hs = schoolrate_df.merge(slocs,on=['geohash'],how='left')
hs = hs.merge(rlocs,on=['geohash'],how='left')
hs.fillna(0,inplace = True) #Fill Speedcams and RedCams with 0 in case of no cam locations in that area
hs.shape
hs.Year.value_counts()
hs.head()
```

Figure 17: Merge safety camera counts with crime data

- **Merge high school**
  As shown in the Figure 18, pre-processed high school data was merged with crime

data based on the year and geohash attributes.

```
#Merge education data
edu = pd.read_csv('C:/Data Analytics/Sem 3/Dataset/LocationData/Distances/EducationMerged.csv')
edu.head()
list(edu.columns)
#Calculate average columns from Year1 and Year 2 data for each academic year
edu['Avg_Dropout_Rate'] = (edu['One_Year_Dropout_Rate_Year_1_Pct'] + edu['One_Year_Dropout_Rate_Year_2_Pct'])/2
edu['Avg_FreshmanTrack_Rate'] = (edu['Freshmen_On_Track_School_Pct_Year_2']+edu['Freshmen_On_Track_School_Pct_Year_1'])/2
edu['Avg_CollegeEnrollment_Rate'] = (edu['College_Enrollment_School_Pct_Year_2'] + edu['College_Enrollment_School_Pct_Year_1'])/2
edu['Avg_College_Persistence_Rate'] = (edu['College_Persistence_School_Pct_Year_2'] + edu['College_Persistence_School_Pct_Year_1'
#Select only required columns
education=edu[['geohash', 'Year','Avg_Misconduct_Rate',
 'Avg_Suspension_Rate',
 'Avg_Student_Attendance_Rate',
 'Avg_Teacher_Attendance_Rate',
 'Avg_Suspension_Days',
 'SchoolCount',
 'Avg_Dropout_Rate',
 'Avg_FreshmanTrack_Rate',
 'Avg_CollegeEnrollment_Rate',
 'Avg_College_Persistence_Rate','Mobility_Rate_Pct']]
#Merge with crime data
data = hs.merge(education,on=['geohash','Year'],how='left')
data.shape
```

Figure 18: Merge high school factors with crime data

- **Merge Weather**
  Weather data was merged with crime based on the Year, month and Day attribute as denoted in Figure 19.

```
#Merge Weather Data
weather = pd.read_csv('C:\Data Analytics\Sem 3\Dataset\LocationData\Distances\weatherapi.csv')
weather.head()
#Group by year and month
weatherm = weather.groupby(['Year','Month']).mean().reset_index()
#Merge with the crime data
alldata = data.merge(weatherm,on=['Year','Month'],how='left')
```

Figure 19: Merge weather attributes with crime data

- After merging, there were around 800 missing values which were dropped. The final dataset after merging activity consists of 33565 rows and 28 attributes as highlighted in Figure 20.

```
#Check for NA values
alldata.isna().sum()
alldata.Year.value_counts()
alldata=alldata.dropna()
alldata.Year.value_counts()
alldata.shape #33565,28 After merging all data
```

Figure 20: Merged Data - Check for NA values

- The dataset after merging, can be described as represented in the below Table 1:

Table 1: Crime Prediction Dataset Description

Attribute Code	Description	Domain
crimescount	Count of crime incidents reported	1 - 130
Primary$_T$$ype$	Type of crime	Assault, Narcotics, Homicide and Violations
Year	Year	2015 - 2018 years
Month	Month	1 - 12 months
geohash	Representation of nearby locations grouped as one area	Alphanumeric value with precision 5
WEEKDAY	Flag indicating whether the day is a weekday or not	0 or 1
Holiday	Flag indicating whether the day is a holiday or not	0 or 1
Time	Time of the day	Morning, Afternoon, Evening or Night
NearestPoliceDist	Distance in kilometers	0.78 - 6.33
NearestRedCamDist	Distance in kilometers	0.51-8.04
RedCamCount	Distance in kilometers	0 - 12
Avg_Student_Attendance_Rate	Attendance rate of student	70% - 96%
Avg_Teacher_Attendance_Rate	Attendance rate of teacher	89% - 95%
Mobility_Rate_Pct	Mobility rate	2% - 37%
SchoolCount	count of schools	1 - 48
wind	average speed of wind in km/hr	34 - 37
prcp	precipitation in mm	32 - 45
avgTemp	temperature in celsius degrees	55 - 77

3.2 Feature Engineering

Effective feature engineering before implementing the models on the data help improve the performance of the models and reduce any possible (Bocca et al.; 2016). These engineering techniques were done in three parts namely, one hot encoding to treat the categorical variables, normalization to treat the numerical features and lastly feature selection to select the best features.

3.2.1 One Hot Encoding

Using the pre-processing library for one hot encoding as well as pandas get_dummies() in python as shown in Figure 21, the categorical variables were transformed to binary encoded attributes.

```python
best_rf_dummies = pd.get_dummies(best_rf)
best_rf_dummies.head() #86 columns
```

Figure 21: One hot encoding using get$_dummies()$

Another way to one-hot encode which gives the output in same format as One Hot encoder (OHE) library as shown in Figure 22:

```python
#One hot encoding for only categorical columns
ohe     = OneHotEncoder(sparse=False)
cat =  best.select_dtypes('object')
columns_to_encode = cat.columns
encoded_columns =    ohe.fit_transform(cat[columns_to_encode])
```

Figure 22: One hot encoding using OHE library

Encoded features expressed as binary form (0 and 1) attributes are shown in the below Figure 23:

```python
narcotics_dummies = pd.get_dummies(narcotics)
narcotics_dummies.head() #95 columns
```

d	geohash_dp3sy	geohash_dp3sz	geohash_dp3t5	geohash_dp3t7	geohash_dp3td	geohash_dp3te	geohash_dp3tf	geohash_dp3tg	geohash_dp3th	geohash_dp3tj	geohash_dp3tk	geohash_dp3tm
7	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0

Figure 23: Features after One hot encoding

3.2.2 Normalization

Normalization has been done using the MinMaxScalar library in python as highlighted in the Figure 24.

```python
#Scaling only on numerical columns
numericols = ['float64','int64']
numericbest = best.select_dtypes(include=numericols)
from sklearn.preprocessing import StandardScaler,OneHotEncoder
sc = MinMaxScaler()
NUM = sc.fit_transform(numericbest)
```

Figure 24: Feature scaling using MinMaXScaler library

3.3 Feature Selection

Feature selection has been done using Recursive feature elimination (RFE) combined with random forest which has been followed in machine learning implementations in the past (Granitto et al.; 2006). This technique ranks the features by its importance and elimination is done by RFE as shown in the code using ranks function in Figure 25 and rfe-rf models as shown in the Figure 26. A function was defined to compute the ranks using RFE.[7]

```python
from sklearn.preprocessing import MinMaxScaler
ranks = {}
# Create our function which stores the feature rankings to the ranks dictionary
def ranking(ranks, names, order=1):
    minmax = MinMaxScaler()
    ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
    ranks = map(lambda x: round(x,2), ranks)
    return dict(zip(names, ranks))
```

Figure 25: Ranks Function for RFE-RF features

3.3.1 Top 20 features with One Hot Encoding

Including the categorical features encoded as binary values, a round of feature selection was performed and the ranks are listed as shown in the Figure 27

```python
# Construct our Random Forest Regression model
from sklearn.feature_selection import RFE
rr = RandomForestRegressor()
rr.fit(X,Y)
#stop the search when only the last feature is left
rfe = RFE(rr, n_features_to_select=20, verbose =3 )
rfe.fit(X,Y)
ranks["RFE_pub"] = ranking(list(map(float, rfe.ranking_)), colnames, order=-1)
print(ranks)
```

Figure 26: Code for fetching top 20 features using RFE-RF

```
ranks

{'RFE_pub': {'Avg_CollegeEnrollment_Rate': 0.93,
  'Avg_College_Persistence_Rate': 1.0,
  'Avg_Dropout_Rate': 0.91,
  'Avg_FreshmanTrack_Rate': 0.92,
  'Avg_Misconduct_Rate': 0.89,
  'Avg_Student_Attendance_Rate': 1.0,
  'Avg_Suspension_Days': 1.0,
  'Avg_Suspension_Rate': 1.0,
  'Avg_Teacher_Attendance_Rate': 0.99,
  'Holiday_False': 0.09,
  'Holiday_True': 1.0,
  'Mobility_Rate_Pct': 1.0,
  'Month_1': 0.76,
  'Month_10': 0.68,
  'Month_11': 0.73,
  'Month_12': 1.0,
  'Month_2': 0.88,
  'Month_3': 0.8,
```

Figure 27: Ranks of top 20 features

3.3.2 Top 10 numerical features

With RFE-RF (Recursive feature elimination-random forest) method, top 10 numerical features were extracted and their ranks are listed in Figure 29 and for the code, refer

[7] https://www.kaggle.com/arthurtok/feature-ranking-rfe-random-forest-linear-models

Figure 28.

```python
# Construct our Random Forest Regression model
from sklearn.feature_selection import RFE
rr = RandomForestRegressor()
rr.fit(X,Y)
#stop the search when only the last feature is left
rfe = RFE(rr, n_features_to_select=10, verbose =3 )
rfe.fit(X,Y)
ranks["RFE_pub"] = ranking(list(map(float, rfe.ranking_)), colnames, order=-1)
ranks
```

Figure 28: Code for fetching top 10 features using RFE-RF

```
ranks
```

```
{'RFE_pub': {'Avg_CollegeEnrollment_Rate': 0.67,
  'Avg_College_Persistence_Rate': 0.44,
  'Avg_Dropout_Rate': 0.11,
  'Avg_FreshmanTrack_Rate': 0.56,
  'Avg_Misconduct_Rate': 0.89,
  'Avg_Student_Attendance_Rate': 1.0,
  'Avg_Suspension_Days': 0.78,
  'Avg_Suspension_Rate': 0.33,
  'Avg_Teacher_Attendance_Rate': 1.0,
  'Mobility_Rate_Pct': 1.0,
  'NearestPoliceDist': 1.0,
  'NearestRedCamDist': 1.0,
  'NearestSpeedCamDist': 0.22,
  'RedCamCount': 1.0,
  'SchoolCount': 1.0,
  'SpeedCamCount': 0.0,
  'avgTemp': 1.0,
  'prcp': 1.0,
  'wind': 1.0}}
```

Figure 29: Ranks of top 10 features

3.4 Modelling

Modelling was done using the python scikit libraries for machine learning. XGBoost regressor, random forest regressor, keras, tensorflow and linear regression libraries were used for modelling.

3.4.1 Data Split

The models have been tested for both the versions of the split i.e. with train-test (80:20) as well as cross validation (k folds:3-30) as highlighted in the Figure 30. Cross validation techniques have enabled efficient sampling of data for the models and eventually generating better results (Ingilevich and Ivanov; 2018) and Kadar (2019).

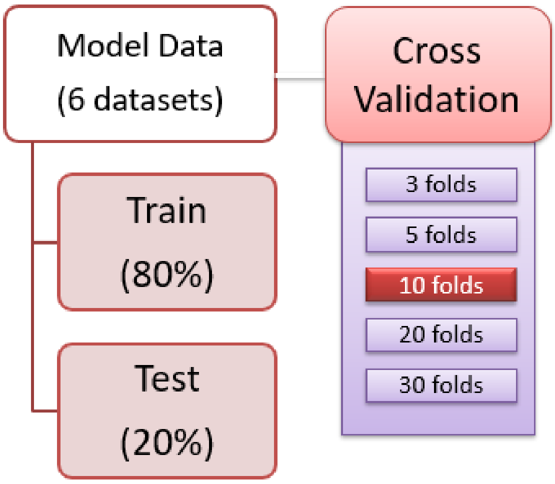

Figure 30: Split of Data

Train test split and k fold libraries were from scikit learn model selection for this task, refer Figure 31.

```
best_rf_dummies = pd.get_dummies(best_rf)
best_rf_dummies.head() #86 columns
#list(best_rf_dummies.columns) (Big output cell)
model_data = best_rf_dummies.values
#separate target and independent variables
X = model_data[:,1:]
Y = model_data[:,0]
#Set seed
np.random.seed(80)
#Split train and test data in 80:20 ratio
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=20)
```

Figure 31: Code to split data

3.4.2 Random Forest

Random forest was applied with default parameters, optimized parameters and finally optimized parameters with 10 fold cross validation.

- Refer the code shown in Figure 32 for random forest with default parameters

```python
clf = RandomForestRegressor(n_estimators=10, random_state=20, n_jobs=-1)
# Train the classifier
clf.fit(X_train, Y_train)
#Training model accuracy
trainac = clf.predict(X_train)
print("Train accuracy details of Random Forest")
print("RMSE is",np.sqrt(mean_squared_error(Y_train,trainac)))
print("R2 ",r2_score(Y_train, trainac))
print("MAE ", mean_absolute_error(Y_train, trainac))
print("MSE ",mean_squared_error(Y_train,trainac))
#Testing model accuracy
y_pred = clf.predict(X_test)
# View The Accuracy Of best features (20 Features) Model
print(clf.score(X_test,Y_test))
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, y_pred)))
```

Figure 32: Random Forest with default parameters

- Refer the code shown in Figure 33 for random forest with optimized parameters

```python
tuned_model = RandomForestRegressor(bootstrap= True,
 max_depth= 50,
 max_features= 'auto',
 min_samples_leaf= 4,
 min_samples_split= 5,
 n_estimators= 377, random_state = 20,n_jobs=-1)
 #Fit the tuned model
tuned_model.fit(X_train, Y_train)
trainac = tuned_model.predict(X_train)
#Train accuracy
print("Train accuracy details on the tuned of Random Forest")
print("RMSE is",np.sqrt(mean_squared_error(Y_train,trainac)))
print("R2 ",r2_score(Y_train, trainac))
print("MAE ", mean_absolute_error(Y_train, trainac))
print("MSE ",mean_squared_error(Y_train,trainac))
#Predict crime count
y_pred = tuned_model.predict(X_test)
# View The Accuracy Of best features (86 Features) Model with tuned parameters
tuned_model.score(X_test,Y_test)
print("RMSE is",np.sqrt(mean_squared_error(Y_test,y_pred)))
print("R2 ",r2_score(Y_test, y_pred))
print("MAE ", mean_absolute_error(Y_test, y_pred))
print("MSE ",mean_squared_error(Y_test,y_pred))
```

Figure 33: Random Forest - optimized model fit

- Random Forest with 10 fold cross validation

```
#Random Forest
kfold = model_selection.KFold(n_splits=10, random_state=200,shuffle=True)
model_kfoldrand = RandomForestRegressor(n_estimators=10, random_state=20, n_jobs=-1)
results_kfoldrand = model_selection.cross_val_score(model_kfoldrand, X, Y, cv=kfold)
print("Accuracy: %.2f%%" % (results_kfoldrand.mean()*100.0))
for train_index, test_index in kfold.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_trainkf, X_testkf = X[train_index], X[test_index]
    y_trainkf, y_testkf = Y[train_index], Y[test_index]
clfrften = model_kfoldrand.fit(X_trainkf, y_trainkf)
print("Residual sum of squares: %.2f"
#      % np.mean((model_kfoldrand.predict(X_testkf) - y_testkf) ** 2))
#Explained variance score: 1 is perfect prediction
print('Explained Variance score: %.2f' % model_kfoldrand.score(X_testkf, y_testkf))
kfoldrf = model_kfoldrand.predict(X_testkf)
print("RMSE is",np.sqrt(mean_squared_error(y_testkf,kfoldrf)))
print("R2 ",r2_score(y_testkf, kfoldrf))
print("MAE ", mean_absolute_error(y_testkf, kfoldrf))
print("MSE ",mean_squared_error(y_testkf,kfoldrf))
```

Figure 34: Random Forest - 10 fold cross validation

3.4.3 XGBoost

XGBoost was also applied with default parameters, optimized parameters and finally optimized parameters with 10 fold cross validation as shown in the codes in Figure 35, Figure 36 and Figure 37.

- XGBoost with default parameters

```
#XGBoost
xgb = xgbo.XGBRegressor(n_estimators=100, learning_rate=0.1)
xgb.fit(X_train,Y_train)
predictions = xgb.predict(X_test)
print("Variance Score is", explained_variance_score(predictions,Y_test))
print("R2 ",r2_score(Y_test, predictions))
print("MAE ", mean_absolute_error(Y_test, predictions))
print("MSE ",mean_squared_error(Y_test,predictions))
print("RMSE is",np.sqrt(mean_squared_error(Y_test,predictions)))
trainac = xgb.predict(X_train)
print("Train accuracy details on the tuned - XGBoost")
print("RMSE is",np.sqrt(mean_squared_error(Y_train,trainac)))
print("R2 ",r2_score(Y_train, trainac))
print("MAE ", mean_absolute_error(Y_train, trainac))
print("MSE ",mean_squared_error(Y_train,trainac))
```

Figure 35: XGBoost - default parameters

- XGBoost with Optimized parameters

```
#Tuned model for XGB
tuned_modelxgb = xgbo.XGBRegressor(bootstrap= True,
 colsample_bytree= 1.0,
 gamma= 1.5,
 learning_rate= 0.05,
 max_depth= 8,
 min_child_weight= 10,
 n_estimators = 200,
 subsample= 0.75, random_state = 20,n_jobs=-1)
tuned_modelxgb.fit(X_train, Y_train)
trainac = tuned_modelxgb.predict(X_train)
print("Train accuracy details on the tuned - XGBoost")
print("RMSE is",np.sqrt(mean_squared_error(Y_train,trainac)))
print("R2 ",r2_score(Y_train, trainac))
print("MAE ", mean_absolute_error(Y_train, trainac))
print("MSE ",mean_squared_error(Y_train,trainac))
y_pred = tuned_modelxgb.predict(X_test)
# View The Accuracy Of best features (86 Features) Model with tuned parameters
tuned_modelxgb.score(X_test,Y_test)
print("RMSE is",np.sqrt(mean_squared_error(Y_test,y_pred)))
print("R2 ",r2_score(Y_test, y_pred))
print("MAE ", mean_absolute_error(Y_test, y_pred))
print("MSE ",mean_squared_error(Y_test,y_pred))
```

Figure 36: XGBoost with optimized parameters

- XGBoost with tuning and 10 fold cross validation

```
#kfold on xgboost tuned
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=10, random_state=200,shuffle=True)
model_kfoldxgb = xgbo.XGBRegressor(bootstrap= True,
 colsample_bytree= 1.0,
 gamma= 1.5,
 learning_rate= 0.05,
 max_depth= 8,
 min_child_weight= 10,
 n_estimators = 200,
 subsample= 0.75, random_state = 20,n_jobs=-1)
results_kfoldxgb = cross_val_score(model_kfoldxgb, X, Y, cv=kfold)
print("Accuracy: %.2f%%" % (results_kfoldxgb.mean()*100.0))
for train_index, test_index in kfold.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_trainkf, X_testkf = X[train_index], X[test_index]
    y_trainkf, y_testkf = Y[train_index], Y[test_index]
clfxgb = model_kfoldxgb.fit(X_trainkf, y_trainkf)
print("Residual sum of squares: %.2f"
#      % np.mean((model_kfoldxgb.predict(X_testkf) - y_testkf) ** 2))
#Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % model_kfoldxgb.score(X_testkf, y_testkf))
kfoldxgb = model_kfoldxgb.predict(X_testkf)
print("RMSE is",np.sqrt(mean_squared_error(y_testkf,kfoldxgb)))
print("R2 ",r2_score(y_testkf, kfoldxgb))
print("MAE ", mean_absolute_error(y_testkf, kfoldxgb))
print("MSE ",mean_squared_error(y_testkf,kfoldxgb))
#Train accuracy
trainac = model_kfoldxgb.predict(X_trainkf)
print("RMSE is",np.sqrt(mean_squared_error(y_trainkf,trainac)))
print("R2 ",r2_score(y_trainkf, trainac))
print("MAE ", mean_absolute_error(y_trainkf, trainac))
print("MSE ",mean_squared_error(y_trainkf,trainac))
```

Figure 37: XGBoost with optimized parameters and 10 fold cross validation

3.4.4 Artificial Neural Network (ANN)

- Refer the code in Figure 38 for implementing ANN without hidden layer

```
#Define ann layers
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='mse', optimizer='rmsprop', metrics=['mse'])
model.summary()
#fit the ann model
history = model.fit(X_train, y_train, batch_size=64, epochs=50, verbose=2, validation_split=.2)
#plot loss vs. epoch curve
plt.figure(figsize=(10,5))
plt.plot(history.history['loss'],marker='o',color='orange')
plt.plot(history.history['val_loss'],marker='^',color='blue')
plt.title('Value Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
#plot the train vs test mse
plt.figure(figsize=(10,5))
plt.plot(history.history['mean_squared_error'],marker='o',color='red')
plt.plot(history.history['val_mean_squared_error'],marker='^',color = 'green')
plt.title('Value Mean Squared Error')
plt.ylabel('MSE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
#predict test data with ann
pred_crimes = model.predict(X_test)
mse_pred_score = metrics.mean_squared_error(pred_crimes, y_test)
print('mse_pred_score {}'.format(mse_pred_score))
rmse_pred_score = np.sqrt(mse_pred_score)
print('rmse_pred_score {}'.format(rmse_pred_score))
r2_pred_score = r2_score(y_test, pred_crimes, multioutput='uniform_average')
print('r2_pred_score - Coefficient of Determination {}'.format(r2_pred_score))
print("MAE ", mean_absolute_error(y_test, pred_crimes))
```

Figure 38: ANN model with one layer

- Refer the code in Figure 39 for implementing ANN with multiple layers

```
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], kernel_initializer='normal', activation='relu'))
model.add(Dense(64, kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(32, kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='mse', optimizer='rmsprop', metrics=['mse'])
model.summary()
#Set seed and fit the model
np.random.seed(80)
history = model.fit(X_train, y_train, batch_size=128, epochs=50, validation_split=.2, verbose=2)
plt.figure(figsize=(10,5))#Plot loss vs. epoch
plt.plot(history.history['loss'],marker='o',color='orange')
plt.plot(history.history['val_loss'],marker='^',color='blue')
plt.title('Value Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
#plot the train vs test mse
plt.figure(figsize=(10,5))
plt.plot(history.history['mean_squared_error'],marker='o',color='red')
plt.plot(history.history['val_mean_squared_error'],marker='^',color = 'green')
plt.title('Value Mean Squared Error')
plt.ylabel('MSE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
pred_crimes = model.predict(X_test) #Predict crime count
mse_pred_score = metrics.mean_squared_error(pred_crimes, y_test)
print('mse_pred_score {}'.format(mse_pred_score))
rmse_pred_score = np.sqrt(mse_pred_score)
print('rmse_pred_score {}'.format(rmse_pred_score))
r2_pred_score = r2_score(y_test, pred_crimes, multioutput='uniform_average')
print('r2_pred_score - Coefficient of Determination {}'.format(r2_pred_score))
print("MAE ", mean_absolute_error(y_test, pred_crimes))
```

Figure 39: ANN model with multiple layers

3.4.5 Multiple Linear Regression

- For multiple linear regression, tuning is not applicable and hence 10 fold cross validation is applied and checked as shown in the code block Figure 40.

```python
X = final.drop(columns='crimescount')
y = final.crimescount
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 20)
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
from sklearn.metrics import r2_score
#score=r2_score(y_test,y_pred)
print("R2 ",r2_score(y_test,y_pred))
print("MAE ", mean_absolute_error(y_test,y_pred))
print("MSE ",mean_squared_error(y_test,y_pred))
print("RMSE is",np.sqrt(mean_squared_error(y_test,y_pred)))
y_pred
residual = y_test-y_pred
residual
fig, ax = plt.subplots(figsize=(10,5))
_ = ax.scatter(residual, y_pred)
# Set common labels
ax.tick_params(axis="x", labelsize=15)
ax.tick_params(axis="y", labelsize=15)
ax.set_xlabel('Residuals', fontsize=15)
ax.set_ylabel('Predictions', fontsize=15)
#The residual vs predictions is clumped and the behaviour is not random, thus homoscedasticity assumption not satis
import scipy as sp
fig, ax = plt.subplots(figsize=(10,5))
_, (__, ___, r) = sp.stats.probplot(residual, plot=ax, fit=True)
ax.tick_params(axis="x", labelsize=15)
ax.tick_params(axis="y", labelsize=15)
ax.set_title('Normal Probability Plot for Errors', fontsize=16)
```

Figure 40: Multiple Linear regression code

3.4.6 Hyper Parameter Optimization

Tuning has been done using Randomized search cv for each model as shown in the below Figure 41 and Figure 42

- Random Forest Tuning

```python
#Hyper parameters for Random Forest
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 500, num = 5)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 50, num = 5)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
# Use the random grid to search for best hyperparameters
# First create the base model to tune
import datetime
print(datetime.datetime.now())
rf = RandomForestRegressor()
# Random search of parameters, using 10 fold cross validation,
# search across 1000 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=20, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, Y_train)
```

Figure 41: Code for tuning Random Forest using Randomized search cv

- XGBoost Tuning

```python
# A parameter grid for XGBoost
params = {
        'learning_rate':[0.02,0.05,0.08,0.10],
        'n_estimators':[20,50,100,150,200],
        'min_child_weight': [1, 5, 10],
        'gamma': [0,0.5, 1, 1.5, 2],
        'subsample': [0.6, 0.75 ,0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5,7,8]
        }
xgb = xgbo.XGBRegressor()
xgbkf = model_selection.KFold(n_splits=10, random_state=200,shuffle=True)
random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=100, n_jobs=-1,
cv=xgbkf.split(X,Y), verbose=3, random_state=20)
print(datetime.datetime.now())
random_search.fit(X, Y)
print(datetime.datetime.now())
```

Figure 42: Code for tuning XGBoost using Randomized search cv

References

Bocca, F. F., Henrique, L. and Rodrigues, A. (2016). The effect of tuning , feature engineering , and feature selection in data mining applied to rainfed sugarcane yield modelling, *Computers and Electronics in Agriculture* **128**: 67–76.
URL: *http://dx.doi.org/10.1016/j.compag.2016.08.015*

Granitto, P. M., Furlanello, C., Biasioli, F. and Gasperi, F. (2006). Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products, **83**: 83–90.

Ingilevich, V. and Ivanov, S. (2018). Crime rate prediction in the urban environment using social factors, *Procedia Computer Science* **136**: 472–478.
URL: *https://doi.org/10.1016/j.procs.2018.08.261*

Kadar, C. (2019). Public decision support for low population density areas : An imbalance- aware hyper-ensemble for spatio-temporal crime prediction, *Decision Support Systems* **119**(September 2018): 107–117.
URL: *https://doi.org/10.1016/j.dss.2019.03.001*