Configuration Manual

MSc Research Project
Data Analytics

YUG KATHIRIYA
Student ID: x2218789

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	YUG KATHIRIYA
Student ID:	x22187839
Programme:	Data Analytics
Year:	2023
Module:	Msc Research Project
Supervisor:	Jorge Basilio
Submission Due Date:	12/08/2024
Project Title:	A comparative analysis of Deep Learning Models for Spatio-Temporal Crime Prediction in Chicago
Word Count:	XXX
Page Count:	10

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:	11th August 2024

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

YUG KATHIRIYA
x22187839

1 Introduction

In this configuration manual will explain all the necessary steps to replicate the process of the research.

2 Integrating Environment

In this section will discuss what are the necessary requirements to perform the experiments.

The entire project is done on Google Collab ensuring that while training the session and RAM doesnt crash out.

Table 1: System Configuration

Category	**Details**
IDE	Google Colab
Programming Language	Python
RAM	High RAM Environment
Disk	Google Drive Storage

Table 2: Time Series Models

Category	**Details**
Time Series Models	`tensorflow.keras.models.Sequential`
	`tensorflow.keras.layers.LSTM`
	`tensorflow.keras.layers.Bidirectional`
	`tensorflow.keras.layers.Conv1D`
	`tensorflow.keras.layers.MaxPooling1D`
	`tensorflow.keras.layers.Dense`
	`tensorflow.keras.layers.Input`
	Custom Attention Layer using `tensorflow.keras.layers.Layer`

Table 3: Scikit-Learn

Category	Details
Scikit-Learn	sklearn.metrics.mean_squared_error
	sklearn.metrics.mean_absolute_error
	sklearn.metrics.r2_score
	sklearn.preprocessing.StandardScaler
	sklearn.metrics.silhouette_score
	sklearn.metrics.davies_bouldin_score
	sklearn.metrics.calinski_harabasz_score

Table 4: Hotspot cluster

Category	Details
Map Generation	folium.Map folium.Marker folium.plugins.MarkerCluster folium.Icon

3 Data Collection and Pre-processing

- First mount the google drive to collab and upload the dataset

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/

[ ] df = pd.read_csv("/content/drive/MyDrive/Research/Crimes_-_2001_to_Present_20240718.csv")
```

Figure 1: Mounting google drive to collab

- After the mount is completed load the dataset which is in csv format downloaded from Chicago data portal.(Crimes 2001 to Present - Chicago Data Portal).
- Load the dataset into a dataframe.
- After the dataset is loaded , convert the Date into Datetime, then extract date components using .dt accessor on Date columns the Year,Month,Day,Hour,Minute,Weekday extracting them and storing it into separate columns.

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Year	Updated On	Latitude	Longitude	Location	Month	Day
371	13204489	JG416325	2023-09-06 11:00:00	0000X E 8TH ST	0810	THEFT	OVER $500	PARKING LOT / GARAGE (NON RESIDENTIAL)	False	False	...	2023	11/04/2023 03:40:18 PM	41.871835	-87.626151	(41.871834768, -87.62615082)	9	6
646	12592454	JF113025	2022-01-14 15:55:00	067XX S MORGAN ST	2826	OTHER OFFENSE	HARASSMENT BY ELECTRONIC MEANS	RESIDENCE	False	True	...	2022	09/14/2023 03:41:59 PM	41.771782	-87.649437	(41.771782439, -87.649436929)	1	14
647	12785595	JF346553	2022-08-05 21:00:00	072XX S UNIVERSITY AVE	1544	SEX OFFENSE	SEXUAL EXPLOITATION OF A CHILD	APARTMENT	True	False	...	2022	09/14/2023 03:41:59 PM	41.763338	-87.597001	(41.763337967, -87.597001131)	8	5
660	12808281	JF373517	2022-08-14 14:00:00	055XX W ARDMORE AVE	1562	SEX OFFENSE	AGGRAVATED CRIMINAL SEXUAL ABUSE	RESIDENCE	False	False	...	2022	09/14/2023 03:41:59 PM	41.985875	-87.766404	(41.985875279, -87.766403857)	8	14
661	12888104	JF469015	2022-11-10 03:47:00	072XX S MAY ST	1477	WEAPONS VIOLATION	RECKLESS FIREARM DISCHARGE	STREET	False	False	...	2022	09/14/2023 03:41:59 PM	41.762615	-87.652840	(41.76261474, -87.652840463)	11	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2445888	26601	JF132803	2022-02-03 16:27:00	000XX E 100TH PL	0110	HOMICIDE	FIRST DEGREE MURDER	AUTO	False	False	...	2022	09/19/2022 03:41:05 PM	41.711753	-87.621374	(41.711753121, -87.621374343)	2	3

Figure 2: Dataset

4 Data Exploration

4.1 Filtering data into 3 periods

The dataset was filtered from year 2018 to 2024. According to 3 periods crime counts are counted of the periods determing which day was the highest crime and lowest crime day.

```
[ ] # Function to find highest and lowest crime days
    def crime_day_stats(df):
        day_counts = df['Weekday'].value_counts().sort_index()
        highest_day = day_counts.idxmax()
        lowest_day = day_counts.idxmin()
        return highest_day, lowest_day, day_counts

    # Get stats for each period
    pre_covid_stats = crime_day_stats(pre_covid_df)
    during_covid_stats = crime_day_stats(during_covid_df)
    post_covid_stats = crime_day_stats(post_covid_df)

    pre_covid_stats, during_covid_stats, post_covid_stats
```

Figure 3: Crime Count

After the counts are defined , the plot function is made to plot the graphs

The Pre-Covid data is from 2018 to 2020, During-Covid from 2020-2022 and Post-Covid from 2022-2024.

```python
[ ] import matplotlib.pyplot as plt

    # Function to plot crime counts by weekday
    def plot_weekday_crime_counts(day_counts, title, highest_day, lowest_day):
        days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        plt.figure(figsize=(10, 5))
        plt.bar(days, day_counts)
        plt.title(title)
        plt.xlabel('Day of the Week')
        plt.ylabel('Number of Crimes')
        plt.xticks(rotation=45)
        # Add annotations for highest and lowest days
        plt.text(highest_day, day_counts[highest_day] + 10, f'Highest: {day_counts[highest_day]}', ha='center', color='red')
        plt.text(lowest_day, day_counts[lowest_day] + 10, f'Lowest: {day_counts[lowest_day]}', ha='center', color='blue')
        plt.show()

[ ] plot_weekday_crime_counts(pre_covid_stats[2], 'Crime Counts by Weekday (Pre-Covid 2018-2020)', pre_covid_stats[0], pre_covid_stats[1])
```

Figure 4: Plotting Function

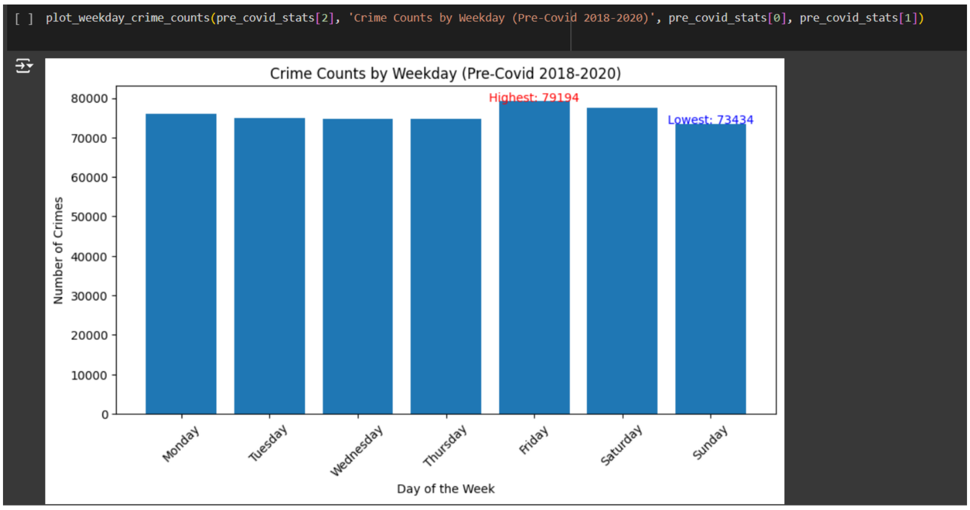


Figure 5: Pre-Covid Graph

5 Model Training

5.1 Pre-Processing

- Handling missing data by df.dropna().
- Date as index in df.
- Resample the data to monthly frequency, counting no of crime per month.
- To handle the outliers , Winsorization is used in monthly crime counts.
- Normalize the data usisng MinMax Scaler.6
- Creating the sequence of 12 months (X) and corresponding next month value as Target(y).
- Splitting the data into 80% training and testing as 20%.

```python
df.dropna(inplace=True)

# Use the 'Date' column as index
df.set_index('Date', inplace=True)

# Resample to get monthly crime counts
monthly_crime = df.resample('M').size()

# Apply Winsorization to limit extreme values
# Limits can be adjusted based on the data; here, the limits are set to trim the bottom and top 5%
monthly_crime_winsorized = winsorize(monthly_crime, limits=[0.05, 0.05])

# Normalize the data
scaler = MinMaxScaler()
monthly_crime_scaled = scaler.fit_transform(monthly_crime_winsorized.reshape(-1, 1))

# Function to create sequences for time series prediction
def create_sequences(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

seq_length = 12  # Use 12 months to predict the next month
X, y = create_sequences(monthly_crime_scaled, seq_length)

# Split the data into training and testing sets
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

Figure 6: Model Training Pre-Process

5.2 Categorize Seasons

```python
# Function to categorize data by seasons
def categorize_seasons(data):
    seasons = {'Winter': [], 'Spring': [], 'Summer': [], 'Autumn': []}
    for date, count in data.items():
        if date.month in [12, 1, 2]:  # December, January, February
            seasons['Winter'].append((date, count))
        elif date.month in [3, 4, 5]:  # March, April, May
            seasons['Spring'].append((date, count))
        elif date.month in [6, 7, 8]:  # June, July, August
            seasons['Summer'].append((date, count))
        elif date.month in [9, 10, 11]:  # September, October, November
            seasons['Autumn'].append((date, count))
    return seasons

# Use this function to split your data by season
seasons_data = categorize_seasons(monthly_crime)

# Preprocessing for each season
def preprocess_season_data(season_data, seq_length):
    dates, counts = zip(*season_data)
    data_series = pd.Series(counts, index=dates)
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data_series.values.reshape(-1, 1))

    X, y = [], []
    for i in range(len(data_scaled) - seq_length):
        X.append(data_scaled[i:i + seq_length])
        y.append(data_scaled[i + seq_length])
    return np.array(X), np.array(y), scaler
```

Figure 7: Categorize seasons

5.3 Model Definitions

- Importing necessary libraries from table 2
- Create a custom attention layer
- Three function defining the model create-attn,create-lstm and create-cnn-lstm and include hyper parameters.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Bidirectional, Conv1D, MaxPooling1D, Dense, Input, Dropout
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Custom Attention Layer
class Attention(tf.keras.layers.Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='att_weight', shape=(input_shape[-1], input_shape[-1]),
                                 initializer='uniform', trainable=True)
        self.b = self.add_weight(name='att_bias', shape=(input_shape[-1],),
                                 initializer='uniform', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, x):
        e = tf.nn.tanh(tf.tensordot(x, self.W, axes=1) + self.b)
        a = tf.nn.softmax(e, axis=1)
        output = x * a
        return tf.reduce_sum(output, axis=1)

# Define the models
def create_attn_bilstm_model(units_1, units_2, dropout_1, dropout_2):
    inputs = Input(shape=(seq_length, 1))
    x = Bidirectional(LSTM(units=units_1, return_sequences=True))(inputs)
    x = Dropout(rate=dropout_1)(x)
    x = Bidirectional(LSTM(units=units_2, return_sequences=True))(x)
    x = Dropout(rate=dropout_2)(x)
```

Figure 8: Attention Mechanism

```python
def create_lstm_model(units_1, units_2, dropout_1):
    model = Sequential()
    model.add(LSTM(units=units_1, return_sequences=True, input_shape=(seq_length, 1)))
    model.add(Dropout(rate=dropout_1))
    model.add(LSTM(units=units_2))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

def create_cnn_lstm_model(filters, units, dropout):
    model = Sequential()
    model.add(Conv1D(filters=filters, kernel_size=2, activation='relu', input_shape=(seq_length, 1)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(LSTM(units=units))
    model.add(Dropout(rate=dropout))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Hyperparameters for models
attn_bilstm_params = {'units_1': 64, 'units_2': 64, 'dropout_1': 0.2, 'dropout_2': 0.2}
lstm_params = {'units_1': 64, 'units_2': 64, 'dropout_1': 0.2}
cnn_lstm_params = {'filters': 64, 'units': 64, 'dropout': 0.2}

# Train and store models
tuned_models = {}

for season in season_datasets:
    X_train = season_datasets[season]['X_train']
    y_train = season_datasets[season]['y_train']
    scaler = season_datasets[season]['scaler']
```

Figure 9: Defining Models

5.4 Training and Evaluation

- Train function takes the model trains them on the training data,makes prediction on the test data using trained model. Inverses scaling applied on predicted values. Calculates evaluate metrics between actual and predicted values. Return a dictonary of the results.10

- Evaluation Loop: It iterates through seasons, for each season retrives the training and testing data for the current seasons.11 For each model type initializing best score with high RMSE values to track best performance. Iterating over a set of hyperparamters for current model type. Train-evaluate model: train model on training data. Best score: compares the models rmse to current. If the models rmse is better updates the best-score and stores it to corresponding hyperparamters.

```python
[ ] # Function to train and evaluate models
    def train_evaluate_model(model, X_train, y_train, X_test, y_test, scaler):
        model.fit(X_train, y_train, epochs=50, batch_size=64, validation_split=0.2, verbose=1)

        # Predict the test set
        predictions = model.predict(X_test)

        # Inverse transform the predictions and actual values
        y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))
        predictions_inverse = scaler.inverse_transform(predictions)

        # Calculate evaluation metrics
        rmse = np.sqrt(mean_squared_error(y_test_inverse, predictions_inverse))
        mae = mean_absolute_error(y_test_inverse, predictions_inverse)
        r2 = r2_score(y_test_inverse, predictions_inverse)
        n = len(y_test_inverse)
        p = predictions_inverse.shape[1] if len(predictions_inverse.shape) > 1 else 1
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

        return {'RMSE': rmse, 'MAE': mae, 'R2': r2, 'Adjusted R2': adjusted_r2}

    # Evaluate models for different seasons
    results = {}
    models = ['ATTN-BILSTM', 'LSTM', 'CNN-LSTM']

    for season in season_datasets:
        X_train = season_datasets[season]['X_train']
        y_train = season_datasets[season]['y_train']
        X_test = season_datasets[season]['X_test']
        y_test = season_datasets[season]['y_test']
        scaler = season_datasets[season]['scaler']
```

Figure 10: Training Models

```python
    for model_type in models:
        best_score = {'RMSE': float('inf')}
        best_params = None
        if model_type == 'ATTN-BILSTM':
            for params in attn_bilstm_params:
                model = create_attn_bilstm_model(**params)
                score = train_evaluate_model(model, X_train, y_train, X_test, y_test, scaler)
                if score['RMSE'] < best_score['RMSE']:
                    best_score = score
                    best_params = params
            results[season][model_type] = best_score
        elif model_type == 'LSTM':
            for params in lstm_params:
                model = create_lstm_model(**params)
                score = train_evaluate_model(model, X_train, y_train, X_test, y_test, scaler)
                if score['RMSE'] < best_score['RMSE']:
                    best_score = score
                    best_params = params
            results[season][model_type] = best_score
        elif model_type == 'CNN-LSTM':
            for params in cnn_lstm_params:
                model = create_cnn_lstm_model(**params)
                score = train_evaluate_model(model, X_train, y_train, X_test, y_test, scaler)
                if score['RMSE'] < best_score['RMSE']:
                    best_score = score
                    best_params = params
            results[season][model_type] = best_score
```

Figure 11: Evaluation Models

5.5 Plotting and Statistical Tests

- Defining function to create the graphs, six parameters were defined.Season,model-type,x-test,y-test and scaler and months.Season for specific seasons(Summer, Winter, Autumn, Spring).12 Model-type: The type of model used(ATTN-BILSTM,LSTM,CNN-LSTM) X-test: test data for model. Y-test: Actual crime count for test data. Scaler: It is used for data pre-processing. Months: List of months names for plotting. Tuned model: Trained models for specific season and the model type. Predictions: Generates predictions for test data. Y-test inverse: Re-scales actual crime count of original data. Predictions inverse: Re-scales predicted crime counts to original scale.

- Statistical Test: After the evaluation the code iterates results which contains the performance metrics for each model and seasons in summary dictonary. The ttest-rel function from Scipy library is used.14 The t-statistic and p-value for each pair

```python
def plot_predictions(season, model_type, X_test, y_test, scaler, months):
    model = tuned_models[season][model_type]
    predictions = model.predict(X_test)

    y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))
    predictions_inverse = scaler.inverse_transform(predictions)

    # Assigning colors to each month
    colors = plt.cm.get_cmap('tab10', len(months))

    plt.figure(figsize=(14, 7))
    for i in range(min(len(months), len(y_test_inverse))):  # Ensure the loop only goes through the available data
        plt.plot([i], y_test_inverse[i], 'o', color=colors(i), label=f'Actual - {months[i]}')
        plt.plot([i], predictions_inverse[i], 'x', color=colors(i), label=f'Predicted - {months[i]}')

    plt.title(f'Crime Trend Prediction ({season} - {model_type})')
    plt.xlabel('Month')
    plt.ylabel('Number of Crimes')
    plt.xticks(ticks=np.arange(len(months)), labels=months, rotation=45)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()

# Plot predictions for each model and season
for season in season_datasets:
    X_test = season_datasets[season]['X_test']
    y_test = season_datasets[season]['y_test']
    scaler = season_datasets[season]['scaler']
    months = season_datasets[season]['months']

    for model_type in tuned_models[season]:
        plot_predictions(season, model_type, X_test, y_test, scaler, months)
```

Figure 12: Plotting

is printed.

```python
    for season in results:
        for model_type in results[season]:
            summary['Model'].append(model_type)
            summary['Season'].append(season)
            summary['RMSE'].append(results[season][model_type]['RMSE'])
            summary['MAE'].append(results[season][model_type]['MAE'])
            summary['R2'].append(results[season][model_type]['R2'])
            summary['Adjusted R2'].append(results[season][model_type]['Adjusted R2'])

    summary_df = pd.DataFrame(summary)

    # Create a DataFrame with the RMSE values for each model and season
    df_rmse = summary_df.pivot(index='Season', columns='Model', values='RMSE')

    # Perform paired t-tests between models
    ttest_bilstm_lstm = ttest_rel(df_rmse['ATTN-BILSTM'], df_rmse['LSTM'])
    ttest_bilstm_cnn_lstm = ttest_rel(df_rmse['ATTN-BILSTM'], df_rmse['CNN-LSTM'])
    ttest_lstm_cnn_lstm = ttest_rel(df_rmse['LSTM'], df_rmse['CNN-LSTM'])

    # Print the t-test results
    print('Paired t-test between ATTN-BILSTM and LSTM:')
    print(f'T-statistic: {ttest_bilstm_lstm.statistic}, P-value: {ttest_bilstm_lstm.pvalue}')

    print('\nPaired t-test between ATTN-BILSTM and CNN-LSTM:')
    print(f'T-statistic: {ttest_bilstm_cnn_lstm.statistic}, P-value: {ttest_bilstm_cnn_lstm.pvalue}')

    print('\nPaired t-test between LSTM and CNN-LSTM:')
    print(f'T-statistic: {ttest_lstm_cnn_lstm.statistic}, P-value: {ttest_lstm_cnn_lstm.pvalue}')
```

Figure 13: Statistical Test

6 HDBSCAN for Hotspot

6.1 Data Pre-processing

- First mount the drive ,load the data set, handle the missing values and drop them off.
- For the clustering puprose only 6 columns were used Date,Year,Month, Primary Type, Longitude and Latitude.
- Data was filtered from 2018 to 2024.

6.2 Model Implementation

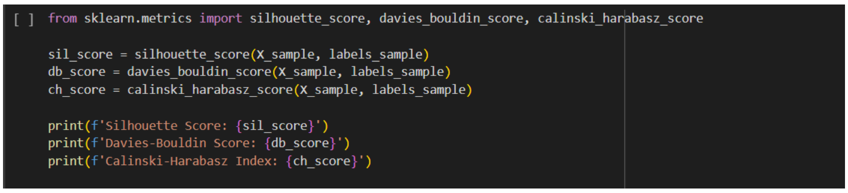
- Install hdbscan and import, cluster size is 50 , fittting hdbscan on Latitude and Longitude.

```
[ ] import hdbscan

    # Function to apply HDBSCAN clustering
    def apply_hdbscan(data, min_cluster_size=50):
        # Remove rows with missing coordinates
        data = data.dropna(subset=['Latitude', 'Longitude'])

        # Apply HDBSCAN
        clusterer = hdbscan.HDBSCAN(min_cluster_size=min_cluster_size)
        data['Cluster'] = clusterer.fit_predict(data[['Latitude', 'Longitude']])

        return data

    # Apply HDBSCAN clustering
    clustered_data = apply_hdbscan(df_filtered)

    # Display the first few rows with clusters
    clustered_data.head()
```

Figure 14: Model

6.3 Evaluating Clustering Performance

- The code calculates three common metrics to evaluate the quality of clustering result: Silhouette Score, Davies-Bouldin Score and Callinski-Harabasz Score.



```
[ ] from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score

    sil_score = silhouette_score(X_sample, labels_sample)
    db_score = davies_bouldin_score(X_sample, labels_sample)
    ch_score = calinski_harabasz_score(X_sample, labels_sample)

    print(f'Silhouette Score: {sil_score}')
    print(f'Davies-Bouldin Score: {db_score}')
    print(f'Calinski-Harabasz Index: {ch_score}')
```

Figure 15: Evaluation Performance

6.4 Visualization

- Install and import folium, crime colors for mapping crime type to colors for better visualization purposes.
- Take data , create folium maps, marker cluster to cluster for better performance. For each crime creates a marker with location based on latitude and longitude . Pop up information about crime type , year and month. Create-map create the maps, save the maps, display the map in Collab environment through IFrame.
- The final result of how the map display with crime type , year and month.17

References

```python
import folium
from folium.plugins import MarkerCluster
# Visualization
crime_colors = {
    'THEFT': 'red', 'BATTERY': 'blue', 'CRIMINAL DAMAGE': 'green', 'NARCOTICS': 'purple',
    'ASSAULT': 'orange', 'OTHER OFFENSE': 'darkred', 'MOTOR VEHICLE THEFT': 'lightblue',
    'ROBBERY': 'lightgreen', 'BURGLARY': 'pink', 'DECEPTIVE PRACTICE': 'darkblue',
    'CRIMINAL TRESPASS': 'darkgreen', 'WEAPONS VIOLATION': 'cadetblue'
}

def create_map(data):
    m = folium.Map(location=[data['Latitude'].mean(), data['Longitude'].mean()], zoom_start=10)
    marker_cluster = MarkerCluster().add_to(m)
    for _, row in data.iterrows():
        crime_type = row['Primary Type']
        color = crime_colors.get(crime_type, 'gray')
        folium.Marker(
            location=[row['Latitude'], row['Longitude']],
            popup=f"Crime: {row['Primary Type']}\nYear: {row['Year']}\nMonth: {row['Month']}",
            icon=folium.Icon(color=color)
        ).add_to(marker_cluster)
    return m

# Create and save the map
crime_map = create_map(sampled_data)
crime_map.save('/content/crime_map.html')

# Display the map in Google Colab
from IPython.display import IFrame
IFrame('/content/crime_map.html', width=800, height=600)
```

Figure 16: Map Generation

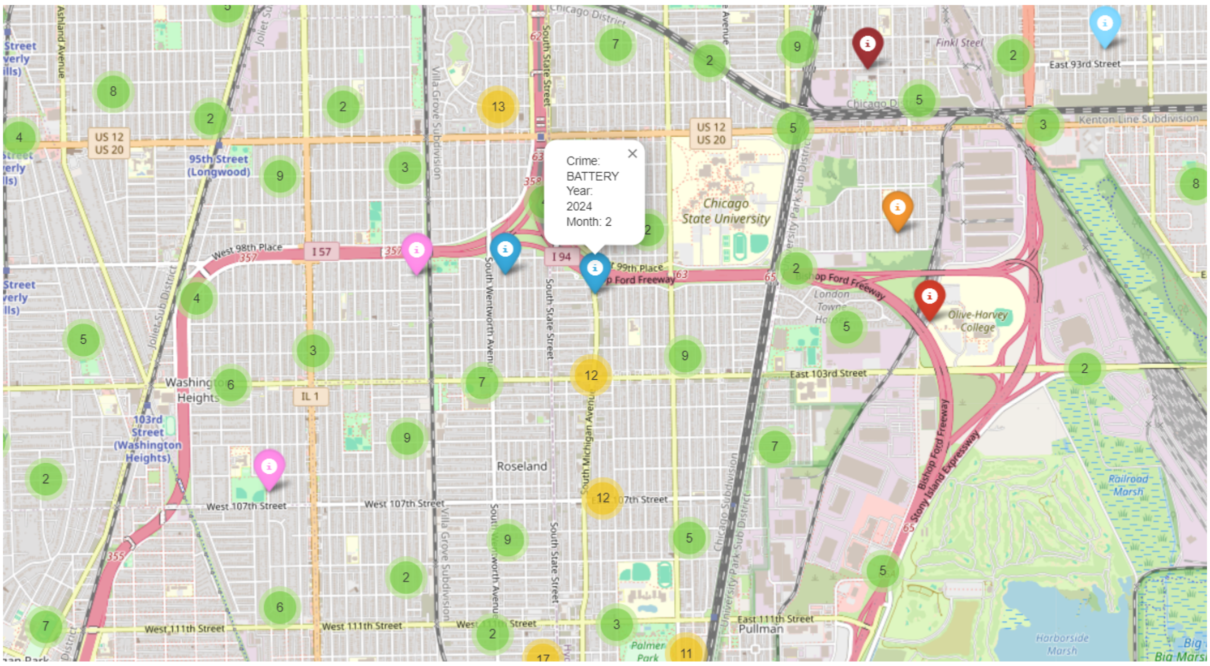


Figure 17: Neighbourhoods with clusters