**Calgary Crime Data Analysis and Neural Network Prediction**

The aim of this project is to use the Crime and Disorder Data provided by the City of Calgary's data website to analyze the data and predict the number of crimes that will occur in the future. The data is from 2018 to 2024 and contains the number of crimes that occurred in Calgary for each month. After throughly analyzing the data, I will be building a neural network model and optimizing it to predict the number of crimes that will occur in the future.

# Data Dictionary

**Column Name Description**

Community Name The name of the community in Calgary

|  |  |
| --- | --- |
| Category | The type of crime that occurred |
| Crime Count | The number of crimes that occurred in that month |
| Year | The year the crime occurred |
| Month | The month the crime occurred |

# Strategy

1. Loading the data and understanding the data
2. Data Preprocessing - cleaing the data and preparing it for analysis
3. Exploratory Data Analysis - Analyzing the data to understand the trends and patterns
4. Building a Neural Network Model
5. Optimizing the model
6. Training the model
7. Predicting the number of crimes that will occur in the future

In [ ]: *#Importing the Libraries* **import** numpy **as** np **import** matplotlib.pyplot **as** plt **import** pandas **as** pd **import** seaborn **as** sns

In [ ]: *#loading the dataset* df **=** pd**.**read\_csv('/content/Community\_Crime\_Statistics\_20240522.csv') df**.**head()

**Community**

**Category**

**Crime Count**

**Year**

**Month**

**0**

01

B

Assault (Non-domestic)

1

2022

11

**1**

01

B

Break & Enter - Commercial

1

2019

6

**2**

01

B

Break & Enter - Commercial

1

2019

8

**3**

01

B

Break & Enter - Commercial

2

2020

3

**4**

01

B

Break & Enter - Commercial

2

2020

7

Out[ ]:

Here the is the representation of first 5 records of the data, which gives a brief informaton about the data. Since the dataset is alphabetically sorted by the community name, the data is not in a chronological order.

## Data Preprocessing

In [ ]: *#shape of the dataset* df**.**shape

Out[ ]: (70661, 5)

Here we have bearly 70661 records and 5 columns. Therefore, we have enough data for preparing an analysis and developing a model for prediction.

In [ ]: *#checking for missing values* df**.**isnull()**.**sum()

Out[ ]: Community 0 Category 0

Crime Count 0

Year 0 Month 0 dtype: int64

The dataset is pretty clean and does not have any missing values.

In [ ]: *#checking for the datatypes* df**.**dtypes

Out[ ]: Community object Category object

Crime Count int64

Year int64 Month int64 dtype: object

Making sure that the columns have correct datatype, before I proceed with the analysis.

In [ ]: *#Descriptive statistics* df**.**describe()

Out[ ]: **Crime Count Year Month**

**count** 70661.000000 70661.000000 70661.000000

|  |  |  |  |
| --- | --- | --- | --- |
| **mean** | 2.855748 | 2020.618616 | 6.369242 |
| **std** | 3.664965 | 1.825330 | 3.451445 |
| **min** | 1.000000 | 2018.000000 | 1.000000 |
| **25%** | 1.000000 | 2019.000000 | 3.000000 |
| **50%** | 2.000000 | 2021.000000 | 6.000000 |
| **75%** | 3.000000 | 2022.000000 | 9.000000 |
| **max** | 111.000000 | 2024.000000 | 12.000000 |

# Exploratory Data Analysis

In the exploraotry data analysis, I will be analyzing the data to understand the trends and patterns in the data. Through this analysis, I will be able to understand the data better and build a better model for prediction.

## Community Distribution

In [ ]: fig, ax **=** plt**.**subplots(1, 2, figsize**=**(15, 5))

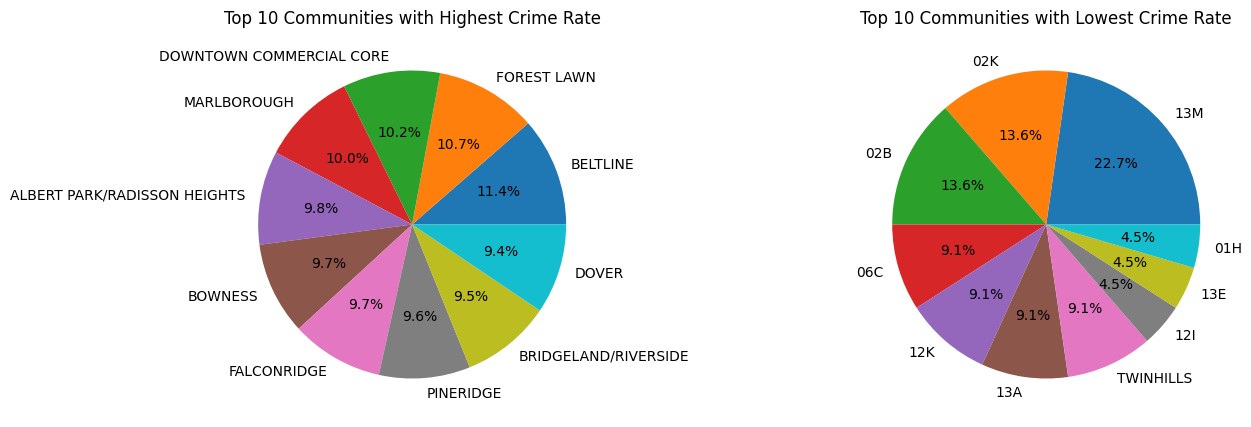
*#Top 10 Communities with Highest Crime Rate*

df['Community']**.**value\_counts()**.**head(10)**.**plot**.**pie(autopct**=**'%1.1f%%', ax **=** ax[0]) ax[0]**.**set\_title('Top 10 Communities with Highest Crime Rate') ax[0]**.**set\_ylabel('')

*#Top 10 Communities with Lowest Crime Rate*

df['Community']**.**value\_counts()**.**tail(10)**.**plot**.**pie(autopct**=**'%1.1f%%', ax **=** ax[1]) ax[1]**.**set\_title('Top 10 Communities with Lowest Crime Rate') ax[1]**.**set\_ylabel('')

|  |  |
| --- | --- |
| Out[ ]: | Text(0, 0.5, '') |



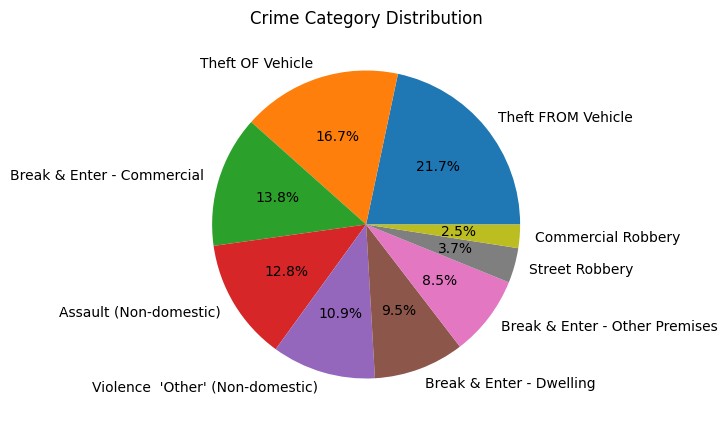
These pie charts show the distribution of crimes in each community. The first pie chart shows the top 10 most dangerous communities in Calgary. The second pie chart shows the distribution of top 10 safest communities in Calgary. In the first pie chart, Beltline is the most dangerous community in Calgary with 11.4% of the top crimes in number, followed by Forest Lawn with 10.7% and Downtown Commercial Core with 10.2%. In the second pie chart, the safest community is 13M with 22.7% of the least crimes in number, followed by 02K with 13.6% and 02B with 13.6%.

*This is note that all these observations are without any bias and completely based on the data from the city of Calgary website.*

## Crime Category Distribution

In [ ]: plt**.**figure(figsize**=**(5, 5)) df['Category']**.**value\_counts()**.**plot**.**pie(autopct**=**'%1.1f%%') plt**.**title('Crime Category Distribution') plt**.**ylabel('')

|  |  |
| --- | --- |
| Out[ ]: | Text(0, 0.5, '') |



This graph shows the distribution of crimes in each category by the number of crimes.

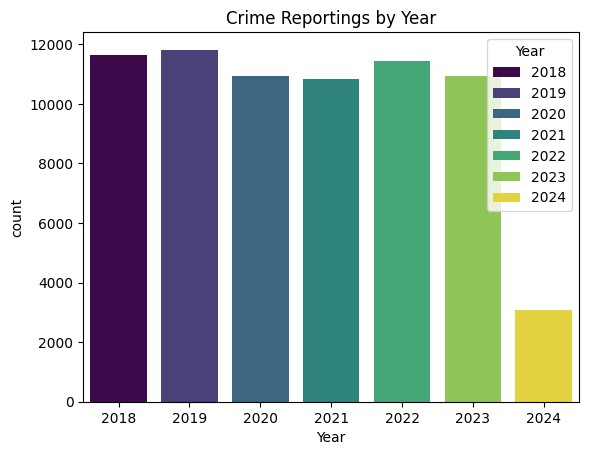
The top crime category is Theft from Vehicle with 21.7% of the total crimes, followed by Theft of Vehicle with 16.7% and Break and Enter - Commercial with 13.8%. The least crime category inc;udes commerical or street robbery.

## Crime Reportings Over the Years

|  |
| --- |
| sns**.**countplot(x **=** 'Year', data **=** df, hue **=** 'Year', palette**=**'viridis')**.**set\_title |

In [ ]:(

Out[ ]: Text(0.5, 1.0, 'Crime Reportings by Year')



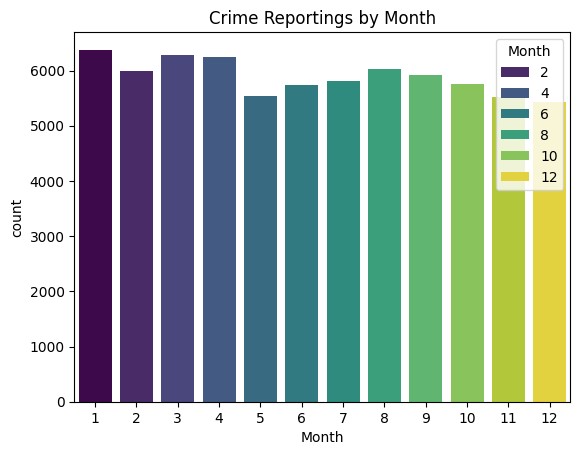
This bar graph shows the distribution of number of crimes reported in the year. The year 2019 had the highest reportings of crimes followed by 2022 and 2018. The crime reportings in 2024 are less due to limited data till April 2024.

## Crime Reportings by Month

|  |
| --- |
| sns**.**countplot(x **=** 'Month', data **=** df, hue **=** 'Month', palette**=**'viridis')**.**set\_tit |

In [ ]:l

Out[ ]: Text(0.5, 1.0, 'Crime Reportings by Month')



## Community and Category Analysis

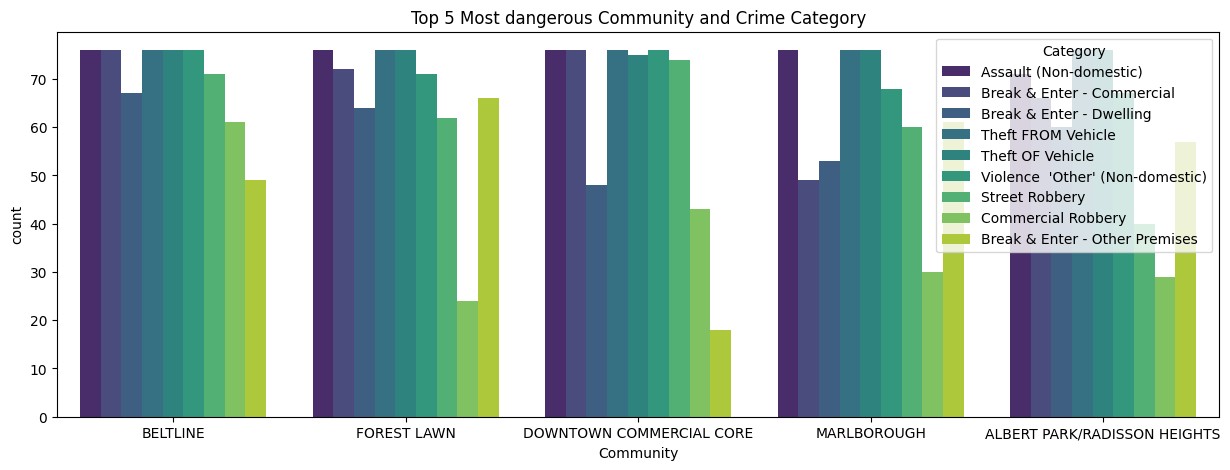
In [ ]:

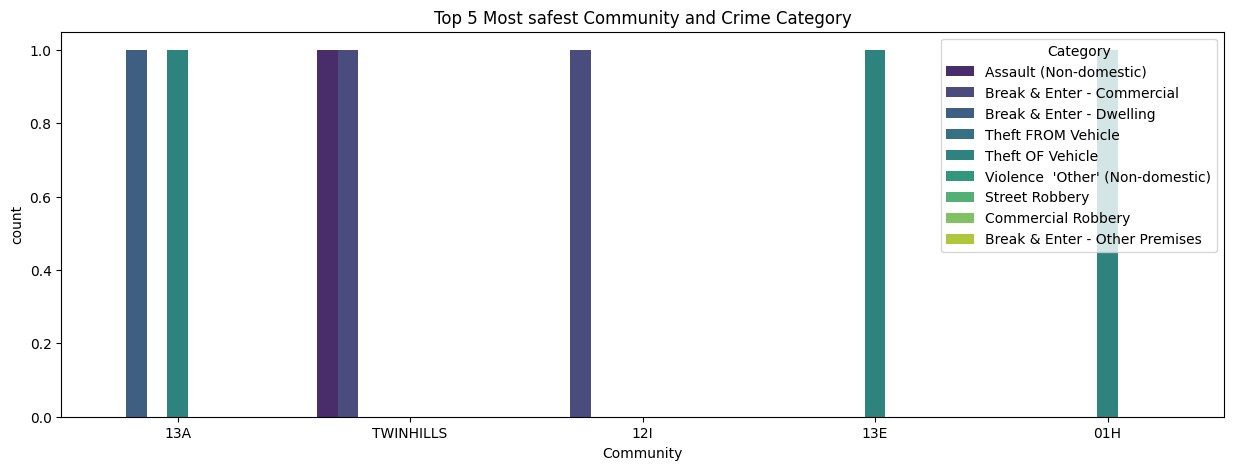
|  |
| --- |
| plt**.**figure(figsize**=**(15, 5)) sns**.**countplot(x **=** 'Community', data **=** df, hue **=** 'Category', palette**=**'viridis', sns**.**move\_legend(plt**.**gca(), "upper right") plt**.**figure(figsize**=**(15, 5)) sns**.**countplot(x **=** 'Community', data **=** df, hue **=** 'Category', palette**=**'viridis', |

o

o

Out[ ]: Text(0.5, 1.0, 'Top 5 Most safest Community and Crime Category')



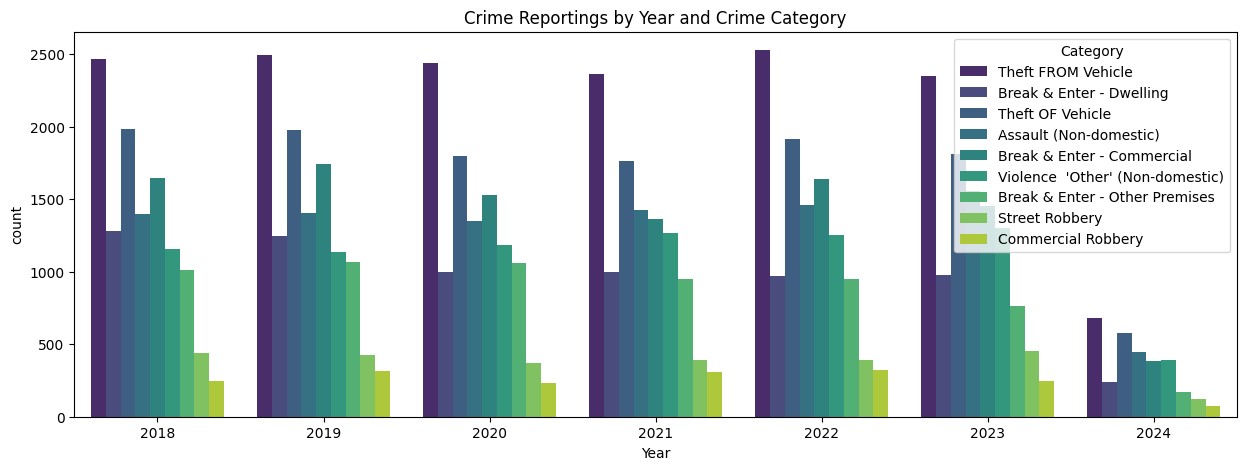


These two graphs shows the analysis of communities with the crime category. This help us to visualize the pattern of crime in each community. We can see that certain cateogries are more common in certain communities than others. In the top 5 dangerous communities, Forest Lawn has the highest of Break & Enter - other premises, Malbrough has the lowest Commerical Robbery. These are the few examples of the analysis.

## Year and Category Analysis

In [ ]: plt**.**figure(figsize**=**(15, 5)) sns**.**countplot(x **=** 'Year', data **=** df, hue **=** 'Category', palette**=**'viridis')**.**set\_ti

Out[ ]: Text(0.5, 1.0, 'Crime Reportings by Year and Crime Category')

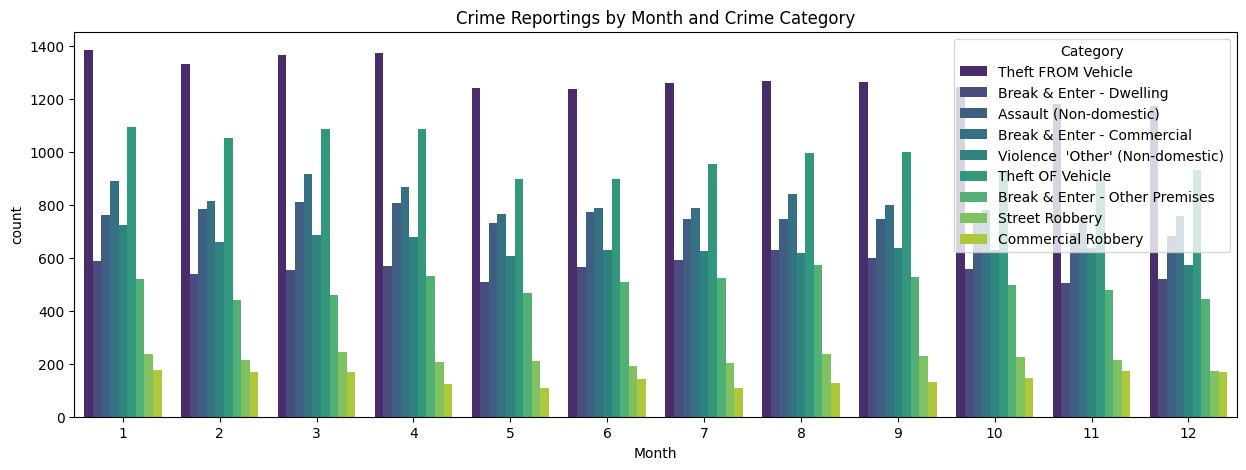


## Month and Category Analysis

|  |
| --- |
| plt**.**figure(figsize**=**(15, 5))  sns**.**countplot(x **=** 'Month', data **=** df, hue **=** 'Category', palette**=**'viridis')**.**set\_ |

In [ ]: t

Out[ ]: Text(0.5, 1.0, 'Crime Reportings by Month and Crime Category')



From the above, graphs, charts, and visualization I have studied the patterns, trends and relationships in the data. This will help me to build a better model for prediction.

# Data Preprocessing Part 2

|  |
| --- |
| **from** sklearn.preprocessing **import** LabelEncoder  *#Label Encoding Object* le **=** LabelEncoder()  *#Object type columns*  object\_type\_columns **=** df**.**select\_dtypes(include**=**'object')**.**columns  *#Label Encoding* **for** col **in** object\_type\_columns:  df[col] **=** le**.**fit\_transform(df[col]) df**.**head() |

In [ ]:

Out[ ]: **Community Category Crime Count Year Month**

**0** 0 0 1 2022 11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | 0 | 1 | 1 | 2019 | 6 |
| **2** | 0 | 1 | 1 | 2019 | 8 |
| **3** | 0 | 1 | 2 | 2020 | 3 |
| **4** | 0 | 1 | 2 | 2020 | 7 |

# Building a Neural Network Model

|  |
| --- |
| *# Prepare sequences for LSTM* **def** create\_sequences(data, seq\_length):  xs **=** [] ys **=** [] **for** i **in** range(len(data) **-** seq\_length):  x **=** data**.**iloc[i:(i **+** seq\_length)]**.**to\_numpy() y **=** data**.**iloc[i **+** seq\_length]['Crime Count'] xs**.**append(x) |

In [ ]:

|  |  |
| --- | --- |
| ys**.**append(y) **return** np**.**array(xs), | np**.**array(ys) |
|  |  |
| seq\_length **=** 3 X, y **=** create\_sequences( | df, seq\_length) |

In [ ]:

## Train Test Split

In [ ]: **from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_temp, y\_train, y\_temp **=** train\_test\_split(X, y, test\_size**=**0.3, random\_ X\_val, X\_test, y\_val, y\_test **=** train\_test\_split(X\_temp, y\_temp, test\_size**=**0.5, r

## Building and Training the LSTM Model

In [ ]: **from** tensorflow.keras.models **import** Sequential **from** tensorflow.keras.layers **import** LSTM, Dense, Dropout **from** tensorflow.keras.optimizers **import** Adam

In [ ]:

|  |
| --- |
| *# Build the LSTM model* model **=** Sequential() model**.**add(LSTM(50, activation**=**'relu', input\_shape**=**(seq\_length, X\_train**.**shape[2] model**.**add(Dropout(0.2)) model**.**add(Dense(1))  *# Compile the model*  optimizer **=** Adam(learning\_rate**=**0.001) model**.**compile(optimizer**=**optimizer, loss**=**'mse')  *# Train the model*  history **=** model**.**fit(X\_train, y\_train, epochs**=**100, validation\_data**=**(X\_val, y\_val |

) ) Epoch 1/100

3092/3092 [==============================] - 9s 3ms/step - loss: 368.3250 - val\_l oss: 12.2236 Epoch 2/100

3092/3092 [==============================] - 9s 3ms/step - loss: 13.1753 - val\_lo ss: 9.2046 Epoch 3/100

3092/3092 [==============================] - 7s 2ms/step - loss: 10.1402 - val\_lo ss: 5.4251 Epoch 4/100

3092/3092 [==============================] - 7s 2ms/step - loss: 6.6290 - val\_los s: 4.8992 Epoch 5/100

3092/3092 [==============================] - 7s 2ms/step - loss: 6.6735 - val\_los s: 4.9798 Epoch 6/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.6858 - val\_los s: 5.1378 Epoch 7/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.8741 - val\_los s: 5.0281 Epoch 8/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.6256 - val\_los s: 4.8595 Epoch 9/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.8799 - val\_los s: 5.1668

Epoch 10/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.6717 - val\_los s: 4.9109

Epoch 11/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.7738 - val\_los s: 5.1230

Epoch 12/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.8282 - val\_los s: 5.1127

Epoch 13/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.7929 - val\_los s: 4.7676

Epoch 14/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6273 - val\_los s: 4.8039

Epoch 15/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.8839 - val\_los s: 4.8101

Epoch 16/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.6465 - val\_los s: 4.7539

Epoch 17/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.5960 - val\_los s: 5.1675

Epoch 18/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4866 - val\_los s: 6.1844

Epoch 19/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5935 - val\_los s: 4.6616

Epoch 20/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.9473 - val\_los s: 5.1832

Epoch 21/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5673 - val\_los s: 4.7347

Epoch 22/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5726 - val\_los s: 5.0483

Epoch 23/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.4310 - val\_los s: 4.7352

Epoch 24/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5576 - val\_los s: 4.7369

Epoch 25/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.6070 - val\_los s: 4.7857

Epoch 26/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.4853 - val\_los s: 4.9823

Epoch 27/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4301 - val\_los s: 4.6676

Epoch 28/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.7564 - val\_los s: 6.6096

Epoch 29/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.6978 - val\_los s: 5.3235

Epoch 30/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.6758 - val\_los s: 5.1226

Epoch 31/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.5496 - val\_los s: 4.8413

Epoch 32/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.3276 - val\_los s: 4.9500

Epoch 33/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5483 - val\_los s: 4.9332

Epoch 34/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.5993 - val\_los s: 4.9068

Epoch 35/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5452 - val\_los s: 4.7737

Epoch 36/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.7006 - val\_los s: 4.7271

Epoch 37/100

3092/3092 [==============================] - 8s 2ms/step - loss: 5.4117 - val\_los s: 4.9678

Epoch 38/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4684 - val\_los s: 5.2002

Epoch 39/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.6481 - val\_los s: 5.6867

Epoch 40/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.5738 - val\_los s: 4.7146

Epoch 41/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4375 - val\_los s: 4.7131

Epoch 42/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4098 - val\_los s: 4.8912

Epoch 43/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4567 - val\_los s: 4.8491

Epoch 44/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5003 - val\_los s: 4.9664

Epoch 45/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5602 - val\_los s: 4.6880

Epoch 46/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.4180 - val\_los s: 5.6115

Epoch 47/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4679 - val\_los s: 4.8983

Epoch 48/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.8144 - val\_los s: 4.8802

Epoch 49/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.5283 - val\_los s: 4.6533

Epoch 50/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4935 - val\_los s: 4.7072

Epoch 51/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5349 - val\_los s: 5.1914

Epoch 52/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4986 - val\_los s: 4.6977

Epoch 53/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5809 - val\_los s: 5.3368

Epoch 54/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5513 - val\_los s: 4.6322

Epoch 55/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.6919 - val\_los s: 4.6831

Epoch 56/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.3303 - val\_los s: 4.8628

Epoch 57/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5249 - val\_los s: 4.8582

Epoch 58/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.6541 - val\_los s: 4.9105

Epoch 59/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6767 - val\_los s: 4.8064

Epoch 60/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5925 - val\_los s: 4.8082

Epoch 61/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.5196 - val\_los s: 4.6729

Epoch 62/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5900 - val\_los s: 4.7126

Epoch 63/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.3842 - val\_los s: 5.2713

Epoch 64/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.8055 - val\_los s: 5.0269

Epoch 65/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5551 - val\_los s: 5.0761

Epoch 66/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6513 - val\_los s: 4.9608

Epoch 67/100

3092/3092 [==============================] - 7s 2ms/step - loss: 5.4326 - val\_los s: 4.9216

Epoch 68/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4771 - val\_los s: 5.0531

Epoch 69/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4703 - val\_los s: 4.7350

Epoch 70/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5293 - val\_los s: 4.9257

Epoch 71/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.3698 - val\_los s: 4.8255

Epoch 72/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4702 - val\_los s: 5.0725

Epoch 73/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6325 - val\_los s: 4.7330

Epoch 74/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4535 - val\_los s: 5.1479

Epoch 75/100

3092/3092 [==============================] - 10s 3ms/step - loss: 5.4698 - val\_lo ss: 4.6433 Epoch 76/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6245 - val\_los s: 4.9227

Epoch 77/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5196 - val\_los s: 4.7539

Epoch 78/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5683 - val\_los s: 4.6994

Epoch 79/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5508 - val\_los s: 4.7919

Epoch 80/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5035 - val\_los s: 5.4767

Epoch 81/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5963 - val\_los s: 4.9065

Epoch 82/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5145 - val\_los s: 5.0828

Epoch 83/100

3092/3092 [==============================] - 10s 3ms/step - loss: 5.5162 - val\_lo ss: 4.8715 Epoch 84/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6411 - val\_los s: 5.6281

Epoch 85/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.7950 - val\_los s: 4.8154

Epoch 86/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5023 - val\_los s: 4.8846

Epoch 87/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.5841 - val\_los s: 4.6612

Epoch 88/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4837 - val\_los s: 4.7223

Epoch 89/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4101 - val\_los s: 4.9879

Epoch 90/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4269 - val\_los s: 4.9915

Epoch 91/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.7826 - val\_los s: 5.6587

Epoch 92/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.6136 - val\_los s: 4.9782

Epoch 93/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.4317 - val\_los s: 4.9384

Epoch 94/100

3092/3092 [==============================] - 10s 3ms/step - loss: 5.6433 - val\_lo ss: 5.0109 Epoch 95/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.8415 - val\_los s: 4.9563

Epoch 96/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.7377 - val\_los s: 4.7079

Epoch 97/100

3092/3092 [==============================] - 9s 3ms/step - loss: 5.4252 - val\_los s: 4.7204

Epoch 98/100

3092/3092 [==============================] - 10s 3ms/step - loss: 5.5803 - val\_lo ss: 4.9075 Epoch 99/100

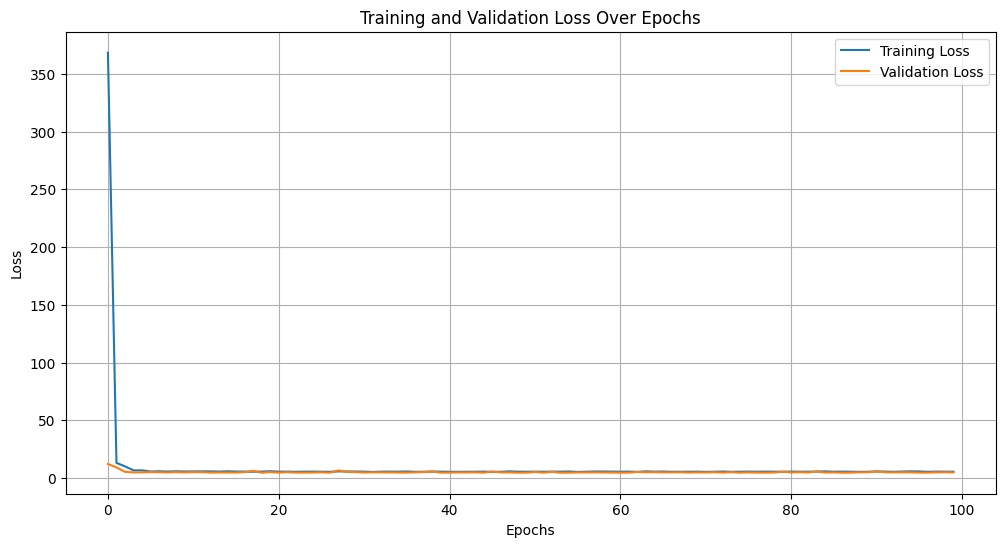
3092/3092 [==============================] - 9s 3ms/step - loss: 5.5518 - val\_los s: 5.0284

Epoch 100/100

3092/3092 [==============================] - 8s 3ms/step - loss: 5.5170 - val\_los s: 4.8733

|  |
| --- |
| plt**.**figure(figsize**=**(12, 6)) plt**.**plot(history**.**history['loss'], label**=**'Training Loss') plt**.**plot(history**.**history['val\_loss'], label**=**'Validation Loss') plt**.**title('Training and Validation Loss Over Epochs') plt**.**xlabel('Epochs') plt**.**ylabel('Loss') plt**.**legend() plt**.**grid(**True**) plt**.**show() |

In [ ]:



|  |
| --- |
| *# Evaluate the model*  test\_loss **=** model**.**evaluate(X\_test, y\_test) print(f'Test Loss: {test\_loss}')  *# Predictions*  y\_pred **=** model**.**predict(X\_test)  print(f'Predictions: {y\_pred**.**flatten()}') print(f'True Values: {y\_test**.**flatten()}') |

In [ ]:

332/332 [==============================] - 0s 1ms/step - loss: 4.8978

Test Loss: 4.897756576538086

332/332 [==============================] - 1s 1ms/step Predictions: [3.5760155 2.0850606 2.3349957 ... 3.710539 1.8771566 3.2992563]

True Values: [2 1 1 ... 1 2 2]

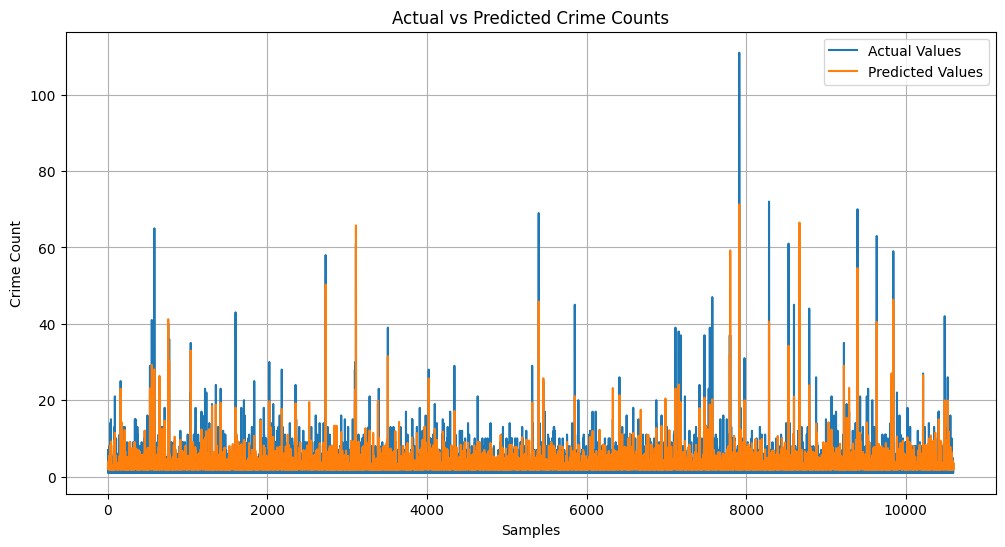
# Model Evaluation

## Actual vs Predicted Values

|  |
| --- |
| *# Plotting Actual vs Predicted Values* plt**.**figure(figsize**=**(12, 6)) plt**.**plot(y\_test, label**=**'Actual Values') plt**.**plot(y\_pred, label**=**'Predicted Values') plt**.**title('Actual vs Predicted Crime Counts') plt**.**xlabel('Samples') plt**.**ylabel('Crime Count') |

In [ ]:

plt**.**legend() plt**.**grid(**True**) plt**.**show()



**Residual Plot**

|  |
| --- |
| *# Calculating residuals*  residuals **=** y\_test**.**flatten() **-** y\_pred**.**flatten()  *# Plotting residuals* plt**.**figure(figsize**=**(12, 6)) plt**.**plot(residuals, label**=**'Residuals') plt**.**title('Residuals (Actual - Predicted) Over Samples') plt**.**xlabel('Samples') plt**.**ylabel('Residuals') plt**.**legend() plt**.**grid(**True**) plt**.**show() |

In [ ]:

