Table of Contents

1. Checking Missing Values and Splitting the Data

```
#Check the range of dates to confirm if any are missing
date_range = pd.date_range(start=time_series_data['Month_Year'].min(),
                           end=time_series_data['Month_Year'].max(),
                           freq='MS')  # 'MS' stands for Month Start frequency
#Identify if there are any missing months in the data
missing_dates = date_range.difference(time_series_data['Month_Year'])
missing_dates
```

```
DatetimeIndex([], dtype='datetime64[ns]', freq='MS')
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
#Splitting the data: 75% for training and 25% for testing
train_data, test_data = train_test_split(time_series_data, test_size=0.25, shuffle=False)
#Normalizing the data
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train_data['Count'].values.reshape(-1, 1))
test_scaled = scaler.transform(test_data['Count'].values.reshape(-1, 1))
# Reshaping the data to fit the RNN input requirements
# RNNs require input shape of the form [samples, time steps, features]
# Here, each sample is one month, and we have one feature - the count
#We have only one feature, so we reshape the data to [samples, time steps=1, features=1]
train_scaled = train_scaled.reshape((train_scaled.shape[0], 1, train_scaled.shape[1]))
test_scaled = test_scaled.reshape((test_scaled.shape[0], 1, test_scaled.shape[1]))
#Check the shapes of the processed data
(train_scaled.shape, test_scaled.shape)
```

```
((3, 1, 1), (1, 1, 1))
```

Figure 1: Missing Values and Data Split

Above it can be seen that there are no missing values in the Month_Year column of the dataset. This is the column that we are using for time-series analysis.

Similarly, we can also see that we have divided the data into training and testing phases. 75% of the data has been used for training and remaining 25% will be used for testing.

2. Training the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
#Define the LSTM model
model = Sequential()
model.add(LSTM(units=8, input_shape=(train_scaled.shape[1], train_scaled.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
#Early stopping callback to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
#Train the model with validation split
history = model.fit(
    train_scaled,
    train_scaled[:, :, 0],  # The target is the same as the input in this case
    epochs=200,
    batch_size=1,
    verbose=1,
    validation_split=0.2,  # Use part of the training data for validation
    callbacks=[early_stopping]
)
#Evaluate the model on training data
train_mae = model.evaluate(train_scaled, train_scaled[:, :, 0], verbose=0)[1]
#Evaluate the model on test data
test_mae = model.evaluate(test_scaled, test_scaled[:, :, 0], verbose=0)[1]
print(f'Train MAE: {train_mae}')
print(f'Test MAE: {test_mae}')
```

```
2/2 [==============================] - 0s 30ms/step - loss: 0.0154 - mae: 0.1188 - val_loss: 0.0281 - val_mae: 0.1676
Epoch 193/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0152 - mae: 0.1180 - val_loss: 0.0275 - val_mae: 0.1660
Epoch 194/200
2/2 [==============================] - 0s 54ms/step - loss: 0.0151 - mae: 0.1173 - val_loss: 0.0272 - val_mae: 0.1649
Epoch 195/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0149 - mae: 0.1166 - val_loss: 0.0268 - val_mae: 0.1639
Epoch 196/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0147 - mae: 0.1158 - val_loss: 0.0265 - val_mae: 0.1627
Epoch 197/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0145 - mae: 0.1150 - val_loss: 0.0259 - val_mae: 0.1611
Epoch 198/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0143 - mae: 0.1142 - val_loss: 0.0254 - val_mae: 0.1595
Epoch 199/200
2/2 [==============================] - 0s 54ms/step - loss: 0.0141 - mae: 0.1134 - val_loss: 0.0250 - val_mae: 0.1580
Epoch 200/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0140 - mae: 0.1127 - val_loss: 0.0247 - val_mae: 0.1570
Train MAE: 0.12705300748348236
Test MAE: 0.12359803915023804
```

Figure 2: Training the Model

Above, the dataset has been trained on 200 number of epochs and their performance has been evaluated with the help of MAE. In the next section, we have presented these values in form of a table.

3. Summary

The following are the results that we have achieved in terms of MAE.

Table 1: Results

Data Split	Training Results	MAE
0	Training	0.127053
1	Testing	0.123598

The table above shows the LSTM model's Mean Absolute Error (MAE) on both the training and test data sets. The model achieved an MAE of about 0.127 on the training data and about 0.124 on the unseen test data, indicating that it performs similarly on both datasets within a small margin.

4. Conclusion

LSTM model has been trained to predict the monthly number of crimes with a respectable degree of accuracy, as shown by the Mean Absolute Error (MAE). Since the test error is somewhat lower than the training error, the MAE of 0.127 for the training set and 0.124 for the test set indicates that the model is not overfitting. Considering the difficulties faced by the little data, this is a promising result.