Stony Brook University

Milestone 3: Time Series Prediction using ARIMA and LSTM

ESE 577: Deep Learning Algorithms and Software.

Spring 2024.

04/13/2024

Suvab Baral, 115646344

Sakshi Nagapure, 116000503

Table of Contents

1. Introduction

Time series forecasting holds significant importance across various domains, offering insights into trends, patterns, and future outcomes. This report compares two short term prediction approaches for time series data:

ARIMA Model: A statistical method that leverages past observations to predict future values. It captures trends and seasonality by incorporating autoregressive (AR), integrated (I), and moving average (MA) components.

LSTM Networks: A type of recurrent neural network (RNN) architecture adept at learning long-term dependencies within time series data. LSTMs address the vanishing gradient problem that hinders traditional RNNs in processing long sequences.

The report will:

1. Briefly discuss the ARIMA model, including parameter selection for trend and seasonality.
    a. **p: Autoregressive order** - the number of past observations used for prediction.
    b. **d: Differencing order** - the number of times the data needs to be different to achieve stationarity (constant mean and variance).
    c. **q: Moving average order** - the number of past forecast errors included in the model.
2. Explain the LSTM network architecture used for time series forecasting.
3. Compare the performance of ARIMA and LSTM models on a selected time series for short-term prediction.

Selecting appropriate values for p, d, and q is crucial for capturing trends and seasonality. Techniques like autocorrelation function (ACF) and partial autocorrelation function (PACF) plots help identify these patterns and guide parameter selection.

We base our report on the New York City Complaint Data: NYC Crime Data - dataset by data-society | data.world . The data is not straight forward, for which we do some cleaning and preprocessing before feeding them into the model. The details are discussed in the sections below.

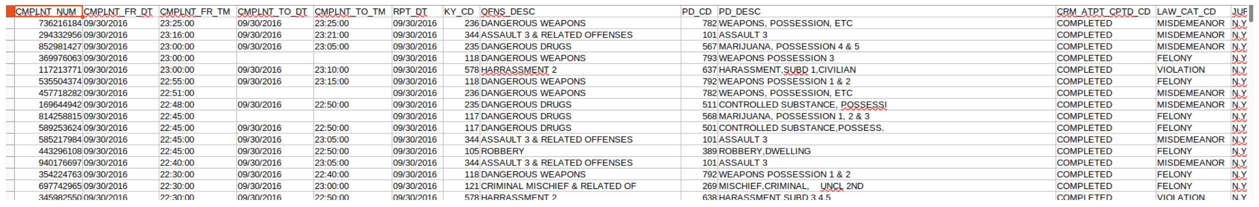
2. Description

Both ARIMA (Autoregressive Integrated Moving Average) and LSTM (Long Short Term Memory) are generally used for time series forecasting however, they may have different properties and scenarios where one may perform better than the other. Since both of these models show great results for forecasting short term data, they fit perfect for time series prediction like weather, temperature, or any other activities that may have some significant connection with the past few data in the dataset. In this report, we look at their prediction power on the NYC Crime dataset and compare their results.

I. Dataset structure:

```
import pandas as pd
df = pd.read_csv('NYPD_Complaint_Data_Current_YTD.csv')
```

The data of the NYC Crime dataset is perfect to test out the forecasting power of such models because they have a good amount of pattern into them. The dataset is provided in CSV format and contains large amount of information:



CMPLNT_NUM	CMPLNT_FR_DT	CMPLNT_FR_TM	CMPLNT_TO_DT	CMPLNT_TO_TM	RPT_DT	KY_CD	OFNS_DESC	PD_CD	PD_DESC	CRM_ATPT_CPTD_CD	LAW_CAT_CD	JUF
736216184	09/30/2016	23:25:00	09/30/2016	23:25:00	09/30/2016	236	DANGEROUS WEAPONS	782	WEAPONS, POSSESSION, ETC	COMPLETED	MISDEMEANOR	N.Y
294332956	09/30/2016	23:16:00	09/30/2016	23:21:00	09/30/2016	344	ASSAULT 3 & RELATED OFFENSES	101	ASSAULT 3	COMPLETED	MISDEMEANOR	N.Y
852981427	09/30/2016	23:00:00	09/30/2016	23:05:00	09/30/2016	235	DANGEROUS DRUGS	567	MARIJUANA, POSSESSION 4 & 5	COMPLETED	MISDEMEANOR	N.Y
369976063	09/30/2016	23:00:00			09/30/2016	118	DANGEROUS WEAPONS	793	WEAPONS POSSESSION 3	COMPLETED	FELONY	N.Y
117213771	09/30/2016	23:00:00	09/30/2016	23:10:00	09/30/2016	578	HARRASSMENT 2	637	HARASSMENT,SUBD 1,CIVILIAN	COMPLETED	VIOLATION	N.Y
535504374	09/30/2016	22:55:00	09/30/2016	23:15:00	09/30/2016	118	DANGEROUS WEAPONS	792	WEAPONS POSSESSION 1 & 2	COMPLETED	FELONY	N.Y
457718282	09/30/2016	22:51:00			09/30/2016	236	DANGEROUS WEAPONS	782	WEAPONS, POSSESSION, ETC	COMPLETED	MISDEMEANOR	N.Y
169644942	09/30/2016	22:48:00	09/30/2016	22:50:00	09/30/2016	235	DANGEROUS DRUGS	511	CONTROLLED SUBSTANCE, POSSESSI	COMPLETED	MISDEMEANOR	N.Y
814258815	09/30/2016	22:45:00			09/30/2016	117	DANGEROUS DRUGS	568	MARIJUANA, POSSESSION 1, 2 & 3	COMPLETED	FELONY	N.Y
589253624	09/30/2016	22:45:00	09/30/2016	22:50:00	09/30/2016	117	DANGEROUS DRUGS	501	CONTROLLED SUBSTANCE,POSSESS.	COMPLETED	FELONY	N.Y
585217984	09/30/2016	22:45:00	09/30/2016	23:05:00	09/30/2016	344	ASSAULT 3 & RELATED OFFENSES	101	ASSAULT 3	COMPLETED	MISDEMEANOR	N.Y
443296108	09/30/2016	22:45:00	09/30/2016	22:50:00	09/30/2016	105	ROBBERY	389	ROBBERY,DWELLING	COMPLETED	FELONY	N.Y
940176697	09/30/2016	22:40:00	09/30/2016	23:05:00	09/30/2016	344	ASSAULT 3 & RELATED OFFENSES	101	ASSAULT 3	COMPLETED	MISDEMEANOR	N.Y
354224763	09/30/2016	22:30:00	09/30/2016	22:40:00	09/30/2016	118	DANGEROUS WEAPONS	792	WEAPONS POSSESSION 1 & 2	COMPLETED	FELONY	N.Y
697742965	09/30/2016	22:30:00	09/30/2016	23:00:00	09/30/2016	121	CRIMINAL MISCHIEF & RELATED OF	269	MISCHIEF,CRIMINAL, UNCL 2ND	COMPLETED	FELONY	N.Y
345982550	09/30/2016	22:30:00	09/30/2016	22:50:00	09/30/2016	578	HARRASSMENT 2	638	HARASSMENT,SUBD 3,4,5	COMPLETED	VIOLATION	N.Y

However, for our purpose of making this dataset fit into the specific criteria of time series forecasting, we downsample the data and use the total number of complaints received in a day as a general outline of the dataset that will be fed into the model.

II. Data cleaning

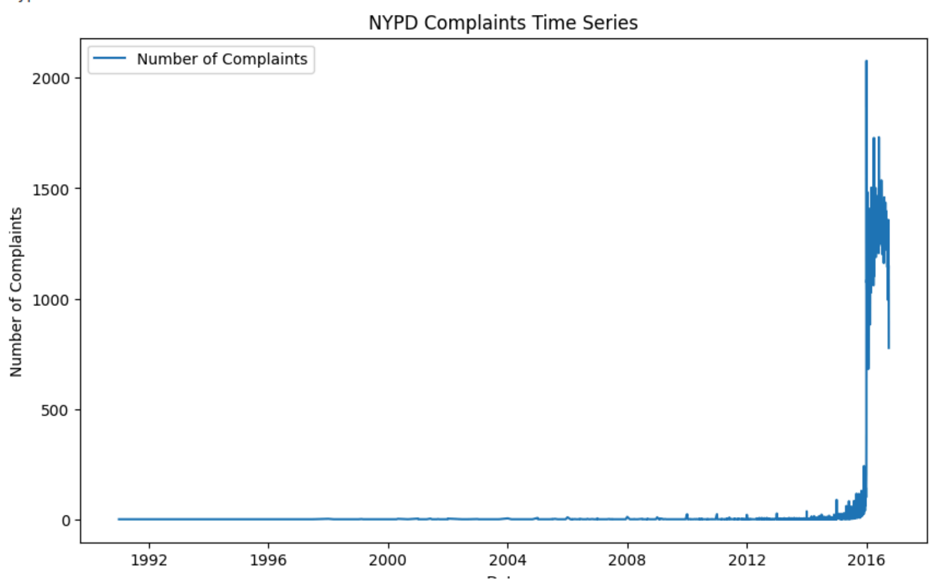
We observed that there are several rows of data that do not have correct information and dates. So, we filter them out in the following way by removing all those columns that do not have a date associated with the complaint:

```
invalid_dates = df['CMPLNT_FR_DT'][pd.to_datetime(df['CMPLNT_FR_DT'],
errors='coerce').isna()]
df = df[~pd.to_datetime(df['CMPLNT_FR_DT'], errors='coerce').isna()]
```

We convert all the dates that are there in different formats, to a standard format of datetime and group by the data column to visualize the pattern of our data:

```
df = df[~pd.to_datetime(df['CMPLNT_FR_DT'], errors='coerce').isna()]
time_series = df.groupby('CMPLNT_FR_DT').size()
```

III Initial Data Visualization:

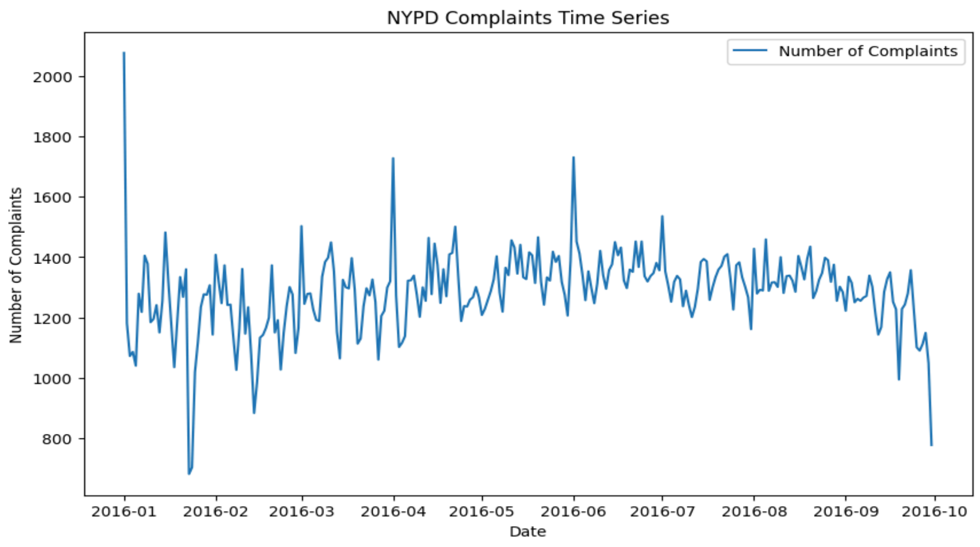


Analyzing our dataset, we can see that most of our data is populated after 2015 and everything before that is not useful or informative data. So we decided to only consider the data from 2015 as our primary dataset under consideration.

IV Data Filtration

After filtering the data starting from 2015, we again plot a section of the data, for example: from January 2016 to October 2016., we get the following plot:

```
time_series = df[(df['CMPLNT_FR_DT'].dt.year >= 2016) &
(df['CMPLNT_FR_DT'].dt.year <= 2024)]
```



We can now observe the time series pattern in our dataset that can be utilized for training a forecasting model. We will use this data for training both ARIMA and LSTM models.

V Train and Test split:

Out of the total dataset, we divide a few sets of data to be used for testing. We will show the plotted graph between the predicted value for the test data along with the actual test data. This gives us a good estimate on how our model is performing utilizing the short term memory.

```
test_data = time_series.iloc[-55:]
time_series = time_series.iloc[:-55]
```

This example shows that we are utilizing the last 55 datasets of the data to be used to test our model. The `test_data` will not be used for training the model, however, we will use it to decide the efficiency of our model. We do this because it gives us a more accurate and easy to visualize description of the accuracy rather than predicting the new results.

In the next sections we will now discuss in detail of our implementation of ARIMA and LSTM model architecture for the given dataset.

3. Algorithm and Implementation:
A. Prediction using ARIMA

ARIMA models are a powerful tool for time series forecasting, particularly when dealing with data that exhibits constant mean and variance over time. ARIMA excels at capturing trends and seasonal patterns, but may struggle with highly complex or non-linear relationships. We utilize the ARIMA model provided by `statsmodels.`

```
from statsmodels.tsa.arima.model import ARIMA
```

Since our data falls under these properties, we use the ARIMA architecture in the following way:

I. ACF and PACF plots:

In ARIMA models, selecting the right parameters (p, d, and q) is crucial for capturing trends and seasonality effectively. This is where Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) come into play.

Autocorrelation Function (ACF):

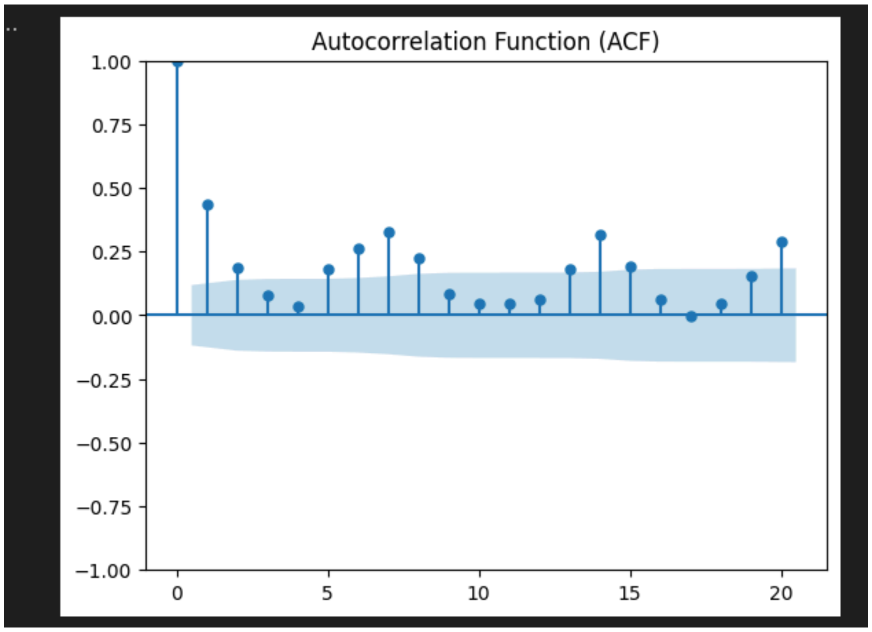
- It measures the correlation between a time series and itself at different lags (time differences).
- A high ACF value at a specific lag indicates that the value at that lag has a significant correlation with the current value.
- By analyzing the ACF plot, you can identify potential values for the AR (autoregressive) order (p) in the ARIMA model.

Partial Autocorrelation Function (PACF):

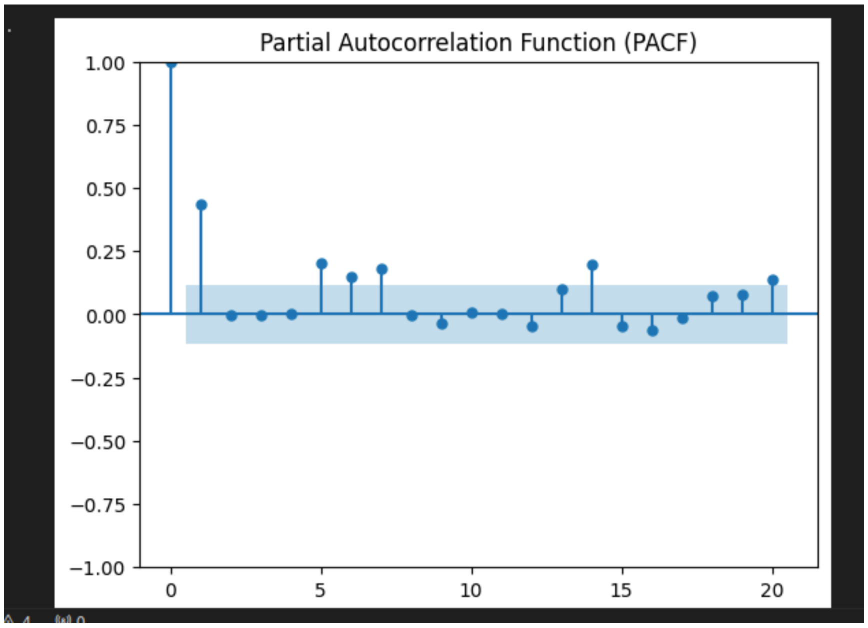
- It measures the correlation between a time series and itself at different lags, after removing the influence of past lags.
- A high PACF value at a specific lag indicates a direct correlation between the current value and the value at that lag, independent of previous lags.
- By analyzing the PACF plot, you can identify potential values for the MA (moving average) order (q) in the ARIMA model.

We use the `statsmodels` library provided by python in order to plot these graphs and analyze them to optimally choose the values for p, d, and q.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(time_series, lags=20)
plt.title('Autocorrelation Function (ACF)')
plt.show()
plot_pacf(time_series, lags=20)
plt.title('Partial Autocorrelation Function (PACF)')
plt.show()
```



The ACF plot suggests a possible correlation at lag 1, with a spike that falls outside the confidence interval. This indicates that the value at the current time step might be influenced by the value at the previous time step (lag 1). Therefore, a starting point for p could be 1.
Like by using both the PACF and ACF, we can see the starting value of q can be around 0.75.



Starting with these values of p, d, and q we start our test to find the best fit ARIMA model by training.

II. Search for best p,d and q:

Ultimately, using the starting point for p, d and q using the graph, we try to see if there is any combination of p, d and q that best fits our data.

```
for p in range(1, 6):
   for q in range(0, 3):
       for d in range(0.75, 3):
           try:
               model = ARIMA(time_series, order=(p, d, q))
               result = model.fit()
               if result.aic < best_aic:
                   best_aic = result.aic
                   best_order = (p, d, q)
           except:
               continue

print("Best ARIMA Order:", best_order)
print("Best AIC:", best_aic)
```

With this small test, we get an idea that we can try to use
p=4, d=2 and q=2.

III Training:

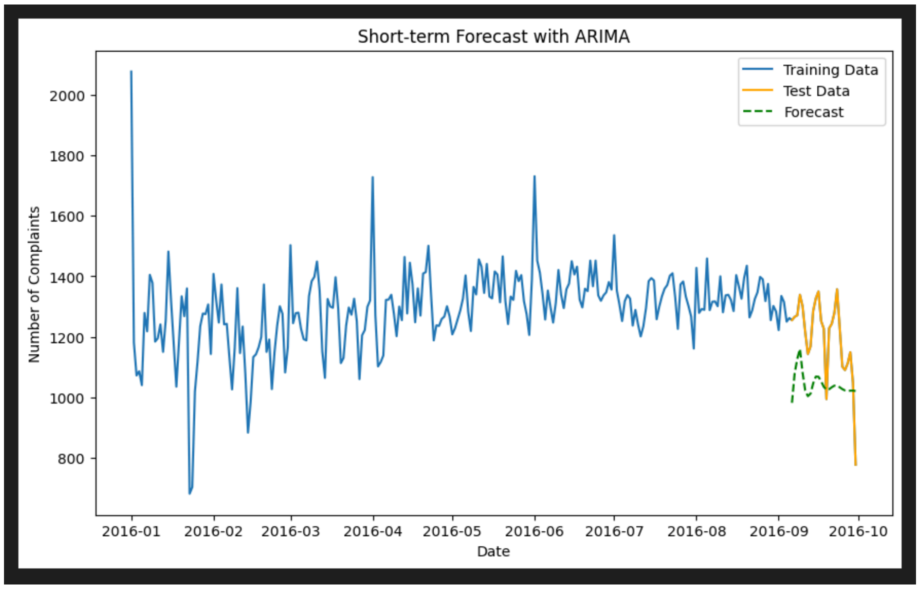
As described above, we use the pre-existing ARIMA model architecture defined by the statsmodel library of python.

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(time_series, order=best_order)
result = model.fit()
```

IV Testing and Visualization (forecast step 25):

We fix on the forecast step of 25, indicating that we want the ARIMA model to make predictions for the next 25 time steps of the time-series predictions and plot the results:

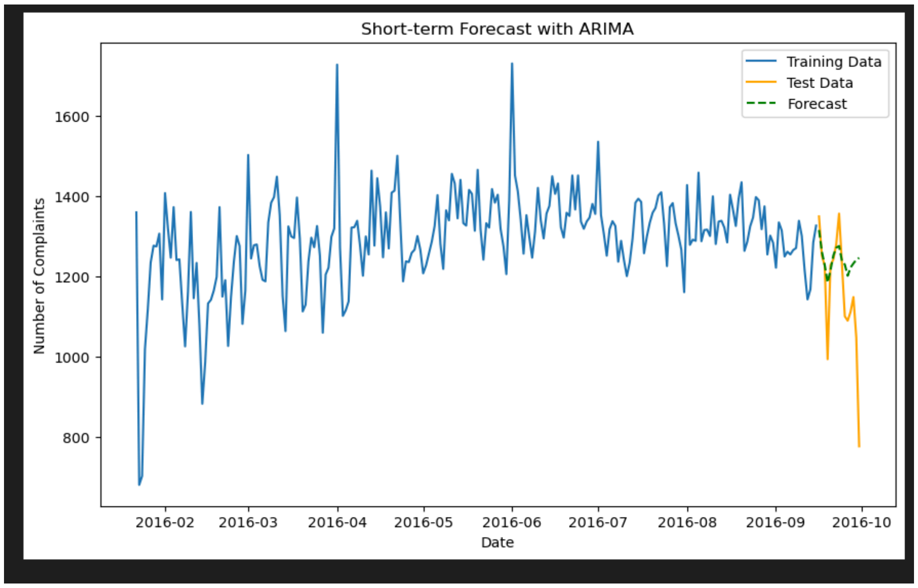
```
forecast_steps = 25
forecast = result.forecast(steps=forecast_steps)
plt.figure(figsize=(10, 6))
plt.plot(time_series.index, time_series.values, label='Training Data')
plt.plot(test_data.index, test_data.values, label='Test Data', color='orange')
plt.plot(test_data.index, forecast, label='Forecast', linestyle='--',
color='green')
```



The results here show that the ARIMA model tried to adjust to the time series data but is not able to actually learn the forecasting pattern.

V Testing and Visualization (further clean data and forecast step: 15)

We are aware that the ARIMA model is not robust enough to adjust to very abrupt spikes in the data points. Because of which we **remove** the **initial spike** that we see in the graph above at **2016-01** and also reduce the prediction window/ forecast step to 15 instead of 25 and analyze the results:



We see a huge amount of improvement in the model accuracy when comparing it with the test data. This shows that if the data is following a fixed pattern and does not have many outliers, the ARIMA model will learn the pattern and forecast pretty accurate results. However, we need to keep in mind that, although this is a good predictor, it still does not achieve the actual prediction accuracy that is very reliable.

3.B. Prediction with LSTM

LSTM Models: A Powerful Tool for Complex Time Series Forecasting

Long Short-Term Memory (LSTM) models are a type of recurrent neural network (RNN) that excel at forecasting time series data with complex or non-linear relationships. Unlike ARIMA, LSTMs are not restricted by assumptions of constant mean and variance. They can effectively capture long-term dependencies and hidden patterns within the data, making them suitable for a wider range of time series problems.

We leverage the LSTM architecture implemented by libraries like TensorFlow or Keras to build our forecasting model. Here's a general outline of the LSTM approach:

1. **Data Preprocessing:** Similar to ARIMA, the data is prepared for the LSTM model. This may involve scaling, normalization, and transforming the time series into sequences for input.
2. **Model Building:** The LSTM architecture consists of stacked LSTM layers that process the sequential data. Each layer learns to identify and retain relevant information from past time steps.
3. **Training:** The model is trained on a portion of the historical data, iteratively adjusting its internal parameters to minimize the forecasting error.
4. **Prediction/Results:** Once trained, the LSTM model can predict future values by feeding in a sequence of past observations.

In this application, LSTMs were specifically chosen due to their effectiveness in capturing long-term dependencies within the complaint data.

The unique gating mechanism of LSTMs allows them to selectively retain relevant information from past time steps, potentially making them a better fit for this problem. LSTM models offer a powerful alternative to ARIMA for tackling a broader spectrum of time series forecasting challenges.

I Model Architecture

- A stacked LSTM architecture is employed. The model uses three LSTM layers with different numbers of hidden units (50, 80, and 70).
    - The `return_sequences` parameter is set to `True` for the first two layers, allowing them to output sequences at each step. This helps capture long-term dependencies within the data.
    - The final LSTM layer has `return_sequences=False` and outputs a single value for prediction.
- A dense layer with one unit is added on top to map the LSTM output to the predicted daily complaint count.

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(sequence_length, 1)))
model.add(LSTM(80, return_sequences=True))
model.add(LSTM(70, return_sequences=False))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_absolute_error')
```

II Training

The training process plays a crucial role in evaluating the model's ability to learn the underlying patterns within the NYPD complaint data. By analyzing the training epochs, we can gain valuable insights into the model's progress.
In the context of this project, evaluating the training process of our LSTM model is crucial for understanding its capacity to forecast future NYPD complaint counts based on historical data.

- The Adam optimizer and mean absolute error (MAE) loss function are used to train the model.
- The data is split into training and testing sets (80/20 split).

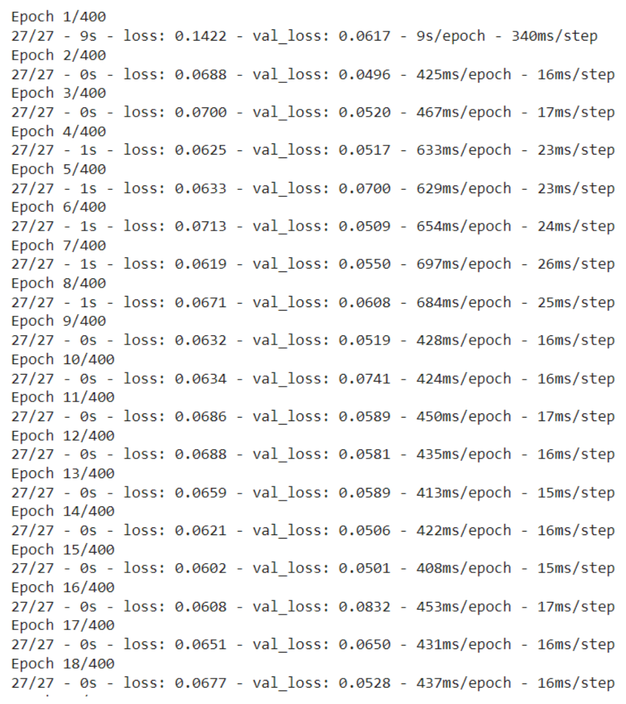
```
# Split the data into training and testing sets (80% train, 20% test)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

- The model is trained on the training set for a specified number of epochs (400 in this case), with validation performed on the testing set to monitor performance and prevent overfitting.

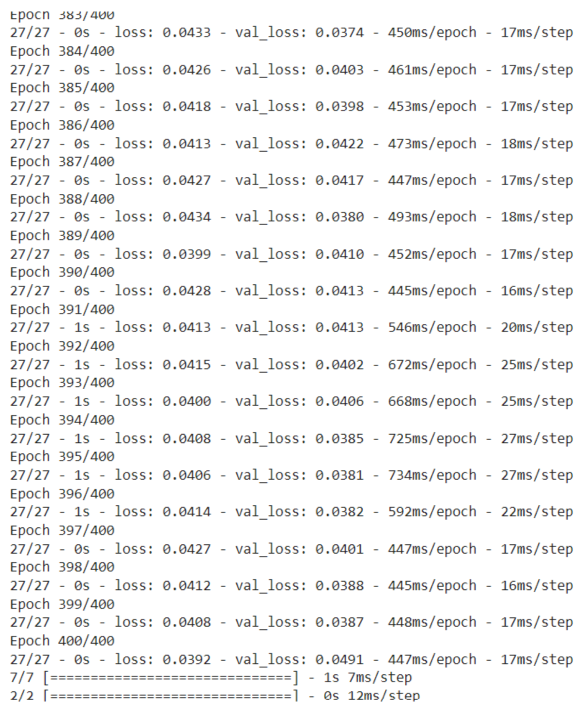
```
model.fit(X_train, y_train, batch_size=8, epochs=400, validation_data=(X_test, y_test), verbose=2)

train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

III Learning in each epoch

The training process indicates that the LSTM model is effectively learning the patterns within the complaint data.
Now, as we increase the number of epochs, The training loss shows a steady decrease, and the validation loss follows a similar trend, suggesting the model is generalizing well.



```
Epoch 1/400
27/27 - 9s - loss: 0.1422 - val_loss: 0.0617 - 9s/epoch - 340ms/step
Epoch 2/400
27/27 - 0s - loss: 0.0688 - val_loss: 0.0496 - 425ms/epoch - 16ms/step
Epoch 3/400
27/27 - 0s - loss: 0.0700 - val_loss: 0.0520 - 467ms/epoch - 17ms/step
Epoch 4/400
27/27 - 1s - loss: 0.0625 - val_loss: 0.0517 - 633ms/epoch - 23ms/step
Epoch 5/400
27/27 - 1s - loss: 0.0633 - val_loss: 0.0700 - 629ms/epoch - 23ms/step
Epoch 6/400
27/27 - 1s - loss: 0.0713 - val_loss: 0.0509 - 654ms/epoch - 24ms/step
Epoch 7/400
27/27 - 1s - loss: 0.0619 - val_loss: 0.0550 - 697ms/epoch - 26ms/step
Epoch 8/400
27/27 - 1s - loss: 0.0671 - val_loss: 0.0608 - 684ms/epoch - 25ms/step
Epoch 9/400
27/27 - 0s - loss: 0.0632 - val_loss: 0.0519 - 428ms/epoch - 16ms/step
Epoch 10/400
27/27 - 0s - loss: 0.0634 - val_loss: 0.0741 - 424ms/epoch - 16ms/step
Epoch 11/400
27/27 - 0s - loss: 0.0686 - val_loss: 0.0589 - 450ms/epoch - 17ms/step
Epoch 12/400
27/27 - 0s - loss: 0.0688 - val_loss: 0.0581 - 435ms/epoch - 16ms/step
Epoch 13/400
27/27 - 0s - loss: 0.0659 - val_loss: 0.0589 - 413ms/epoch - 15ms/step
Epoch 14/400
27/27 - 0s - loss: 0.0621 - val_loss: 0.0506 - 422ms/epoch - 16ms/step
Epoch 15/400
27/27 - 0s - loss: 0.0602 - val_loss: 0.0501 - 408ms/epoch - 15ms/step
Epoch 16/400
27/27 - 0s - loss: 0.0608 - val_loss: 0.0832 - 453ms/epoch - 17ms/step
Epoch 17/400
27/27 - 0s - loss: 0.0651 - val_loss: 0.0650 - 431ms/epoch - 16ms/step
Epoch 18/400
27/27 - 0s - loss: 0.0677 - val_loss: 0.0528 - 437ms/epoch - 16ms/step
```



```
Epoch 383/400
27/27 - 0s - loss: 0.0433 - val_loss: 0.0374 - 450ms/epoch - 17ms/step
Epoch 384/400
27/27 - 0s - loss: 0.0426 - val_loss: 0.0403 - 461ms/epoch - 17ms/step
Epoch 385/400
27/27 - 0s - loss: 0.0418 - val_loss: 0.0398 - 453ms/epoch - 17ms/step
Epoch 386/400
27/27 - 0s - loss: 0.0413 - val_loss: 0.0422 - 473ms/epoch - 18ms/step
Epoch 387/400
27/27 - 0s - loss: 0.0427 - val_loss: 0.0417 - 447ms/epoch - 17ms/step
Epoch 388/400
27/27 - 0s - loss: 0.0434 - val_loss: 0.0380 - 493ms/epoch - 18ms/step
Epoch 389/400
27/27 - 0s - loss: 0.0399 - val_loss: 0.0410 - 452ms/epoch - 17ms/step
Epoch 390/400
27/27 - 0s - loss: 0.0428 - val_loss: 0.0413 - 445ms/epoch - 16ms/step
Epoch 391/400
27/27 - 1s - loss: 0.0413 - val_loss: 0.0413 - 546ms/epoch - 20ms/step
Epoch 392/400
27/27 - 1s - loss: 0.0415 - val_loss: 0.0402 - 672ms/epoch - 25ms/step
Epoch 393/400
27/27 - 1s - loss: 0.0400 - val_loss: 0.0406 - 668ms/epoch - 25ms/step
Epoch 394/400
27/27 - 1s - loss: 0.0408 - val_loss: 0.0385 - 725ms/epoch - 27ms/step
Epoch 395/400
27/27 - 1s - loss: 0.0406 - val_loss: 0.0381 - 734ms/epoch - 27ms/step
Epoch 396/400
27/27 - 1s - loss: 0.0414 - val_loss: 0.0382 - 592ms/epoch - 22ms/step
Epoch 397/400
27/27 - 0s - loss: 0.0427 - val_loss: 0.0401 - 447ms/epoch - 17ms/step
Epoch 398/400
27/27 - 0s - loss: 0.0412 - val_loss: 0.0388 - 445ms/epoch - 16ms/step
Epoch 399/400
27/27 - 0s - loss: 0.0408 - val_loss: 0.0387 - 448ms/epoch - 17ms/step
Epoch 400/400
27/27 - 0s - loss: 0.0392 - val_loss: 0.0491 - 447ms/epoch - 17ms/step
7/7 [==============================] - 1s 7ms/step
2/2 [==============================] - 0s 12ms/step
```

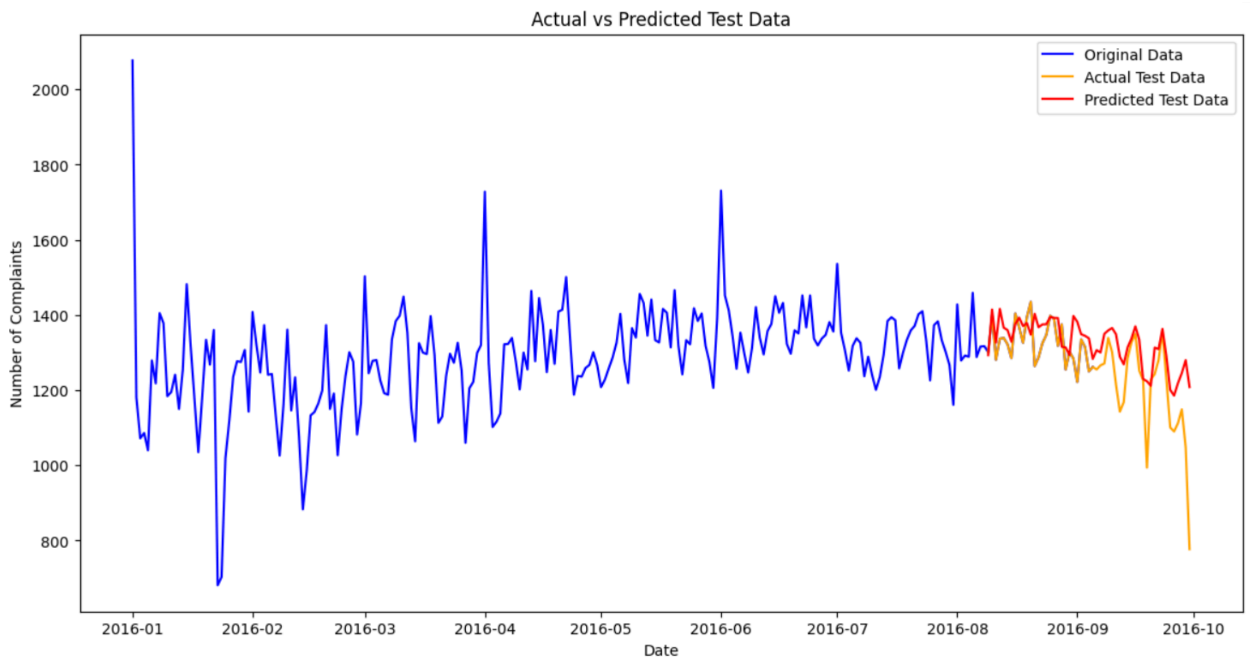
After almost 320 epochs, the losses appear to be stabilizing, indicating that the model may have reached a good level of performance.
This trend indicates that the model is successfully adjusting its internal parameters to better represent the relationship between historical complaint counts and future values.

IV Testing and Visualization

- The trained model is used to predict complaint counts for the unseen test data.
- Scaling is reversed to obtain the predicted complaint counts in the original data scale.(It scales the data using MinMaxScaler to normalize it between 0 and 1)
- The actual and predicted complaint counts for the test set are visualized on a time series plot to assess the model's forecasting accuracy.

```
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))
plt.figure(figsize=(14, 7))

plt.plot(time_series.index[:-25], time_series['count'][:-25], label='Original Data', color='blue')
plt.plot(time_series.index[split + sequence_length:], y_test_actual, label='Actual Test Data', color='orange')
plt.plot(time_series.index[split + sequence_length:], test_predict, label='Predicted Test Data', color='red')

plt.legend()
plt.xlabel('Date')
plt.ylabel('Number of Complaints')
plt.title('Actual vs Predicted Test Data')
plt.show()
```

- Subsequently, it plots the original data (up to the 75th percentile), actual test data, and predicted test data against the corresponding dates.



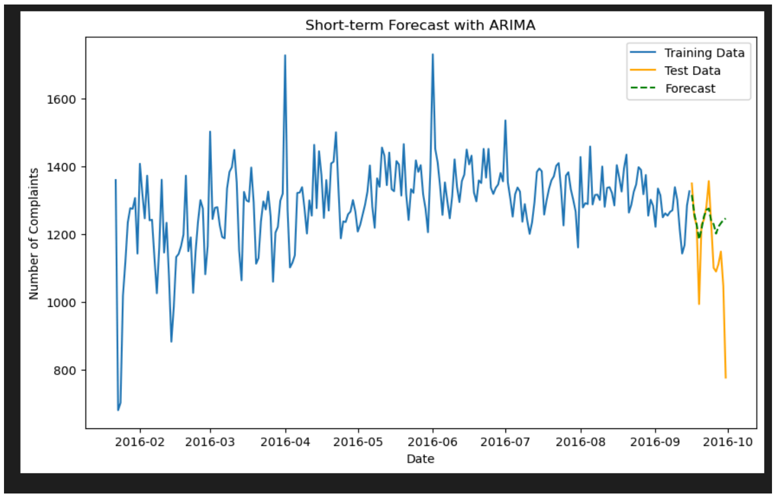
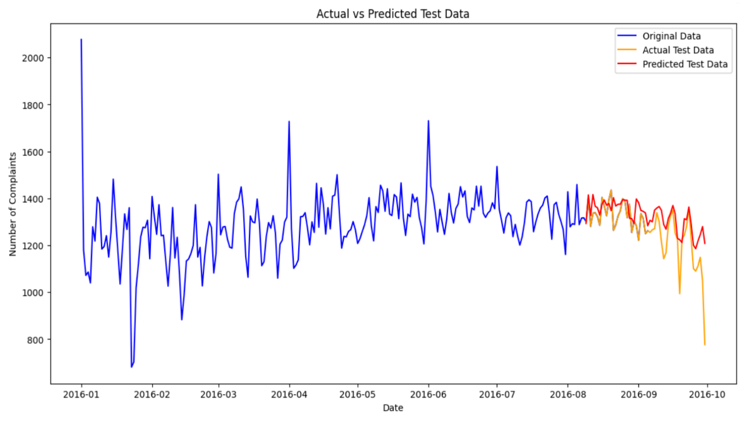
The original data is represented in blue, the actual test data in orange, and the predicted test data in red.

V Results:

The implemented LSTM model shows promise in predicting short-term trends in crime incidents based on historical NYPD complaint data. Overall, the training process demonstrates the model's ability to learn from the data. The observed loss reduction and validation performance trends suggest the model is generalizing well. However, a more comprehensive evaluation using additional metrics on the test set is recommended for a definitive assessment of the model's forecasting accuracy.

5. Comparison: ARIMA vs LSTM

Comparing the final prediction results we get the following:

ARIMA	LSTM
	

From the journey of implementing both these algorithms on the same dataset, and from the figure above, we can finalize on the following comparison points.

1. **Time taken:** The time taken to train an ARIMA model is relatively shorter compared to training a LSTM model. The LSTM model took approximately 15 minutes to train while the ARIMA model was less than a couple of minutes.
2. **Impact of forecast step:** Increasing the forecasting step for LSTM did not have much impact on the performance of the algorithm, however, for ARIMA, the model produced significantly better results when the forecasting step was smaller.
   As seen in section **2.IV)** when the forecasting step was high as 25, the ARIMA model did not produce good results.
3. **Accuracy and loss:** As seen in the discussions above, we can see that **LSTM models** fit very well with the given data, with a **validation error of less than 10%** but the **ARIMA** model shows a validation error of around **40% for the same** volume and type of data.
4. **Ability to handle change:** ARIMA model performs better with a rather stationary data that does not have sudden change in mean and variance, however, LSTM has the ability to learn rather complex and variable datasets.

6. Conclusion

To conclude, this report compared ARIMA and LSTM models for short-term time series forecasting. ARIMA models are much faster to train but struggle with larger forecasting steps and complex data. LSTMs, while requiring longer training times, deliver superior accuracy on diverse datasets and maintain performance even with increased forecast horizons. For short-term forecasts on relatively stable data, ARIMA's speed and interpretability might be advantageous. However, for high-accuracy predictions on potentially intricate data structures, LSTMs are the stronger choice. Future work could explore the long-term forecasting capabilities of both models and investigate methods to enhance ARIMA's accuracy on non-stationary data.

6. References

1. D. Kobiela, D. Krefta, W. Król, and P. Weichbroth, "ARIMA vs LSTM on NASDAQ stock exchange data," in *Proceedings of the 26th International Conference KES2022*, vol. 207, pp. 3836-3845, 2022.
2. S. Siami-Namini and A. Siami Namin, "Forecasting Economics and Financial Time Series: ARIMA vs. LSTM," 2018. [Online]. Available: arXiv:1803.06386.
3. Ian Goodfellow et al., Deep Learning, The MIT Press, 2016.
4. Abbasimehr H, Shabani M, Yousefi M (2020) An optimized model using LSTM network for demand forecasting. Comput Ind Eng 143:106435. https://doi.org/10.1016/j.cie.2020.106435
5. Abbasimehr, H., Paki, R. Improving time series forecasting using LSTM and attention models. J Ambient Intell Human Comput 13, 673–691 (2022). https://doi.org/10.1007/s12652-020-02761-x