Advanced research topics in Data Science

GitHub Link : *https://github.com/sravanviswanath/Advanced-Research-Topics-in-Data-Science*

Table of Contents

Brief Overview of the Methods 3

Results of Machine Learning 3

References 15

List of Figures

Figure 1: Johnson and Johnson Sales Data 4

Figure 2: Differenced Series 5

Figure 3: Autocorrelation 5

Figure 4: Partial Correlation 6

Figure 5: ARIMA Model 6

Figure 6: ARIMA Forecast for Next 24 months 7

Figure 7: LSTM 24-Month Forecast 8

Figure 8: GRU 24-Month Forecast 9

Figure 9: Amazon Monthly Average Close Price 10

Figure 10: Autocorrelation 10

Figure 11: Partial Correlation 11

Figure 12: ARIMA Model 11

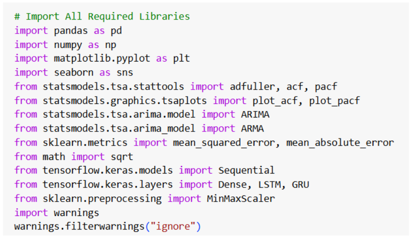
Figure 13: LSTM Forecast 13

Figure 14: GRU Forecast 13

Brief Overview of the Methods

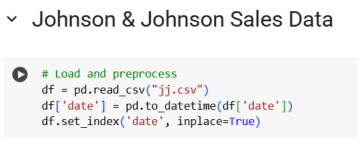
A wide variety of methods were used in the Python notebook to make intuitive and quantitative analysis of, and to forecast the data, about Johnson & Johnson's quarterly sales data. Importing essentially the physics libraries of pandas, NumPy, matplotlib, seaborn and a range of statistical and machine learning tools helps with data manipulation, visualization, and model development. It first reads the supplied dataset and then proceeds to preprocess it by parsing date column that is later used as index for time series operations. It creates an initial visualization to see if there are any general trend or pattern of sales over time (Kontopoulou et al., 2023). The Augmented Dickey-Fuller (ADF) test is conducted to assess stationarity; and as such, time series forecasting. To determine the right order of ARIMA or ARMA models we generate the autocorrelation and partial autocorrelation plots (ACF and PACF). ARIMA and ARMA are the traditional statistical models applied on data, and their performance is evaluated with the standard error metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) (Siami-Namini, Tavakoli and Siami Namin, 2018). The notebook also includes the application of deep learning techniques on forecasting. MinMaxScaler is applied to data to suit training neural network. TensorFlow/Keras is used for implementing recurrent neural network architectures such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). In these models, a portion of the time series data is used for training, the remaining data for validation and prediction. One-step-ahead forecasts are made by using dense output layers. It contains code for plotting predicted values and actual values to visually check performance. During the execution, they suppress warnings to make the code readable (Tarmanini et al., 2023). By combining statistical models with deep learning in this hybrid approach, it creates the robust comparative framework for the sales trends and improves accuracy of future predictions.

Results of Machine Learning



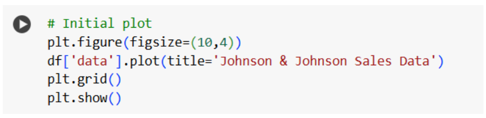
```python
# Import All Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.arima_model import ARMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore")
```

Python script imports a couple of libraries very necessary for analysis, visualization, and primitives of machine learning. Pandas, NumPy, matplotlib, seaborn for analytics, statsmodels for time series modelling, sklearn for error metrics and TensorFlow for deep learning. In data science and forecasting projects the backbone to this setup is used.



Johnson & Johnson Sales Data

```python
# Load and preprocess
df = pd.read_csv("jj.csv")
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)
```



```
# Initial plot
plt.figure(figsize=(10,4))
df['data'].plot(title='Johnson & Johnson Sales Data')
plt.grid()
plt.show()
```

Johnson & Johnson's sales (1960–1980) are plotted in the graph with a smooth rising, but cyclical behaviour. The dips and peaks might be seasonal or market related. Understanding long term growth patterns and sales variability is important when it comes to forecasting and decision making, but this dataset also helps to see how long-term predictions may be off by a mile.

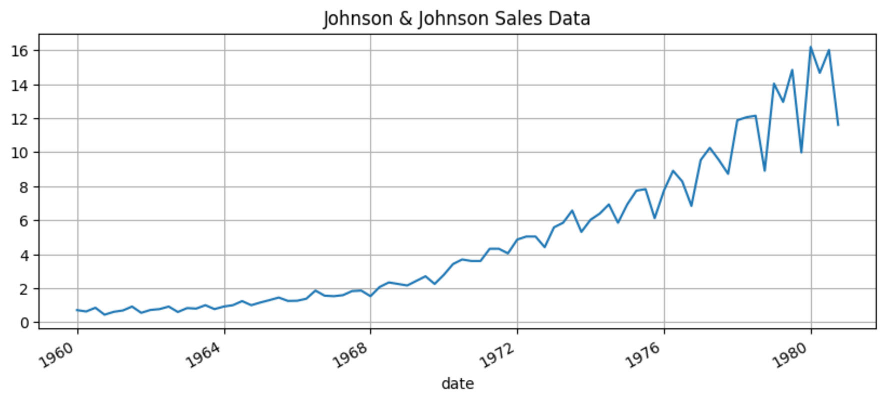
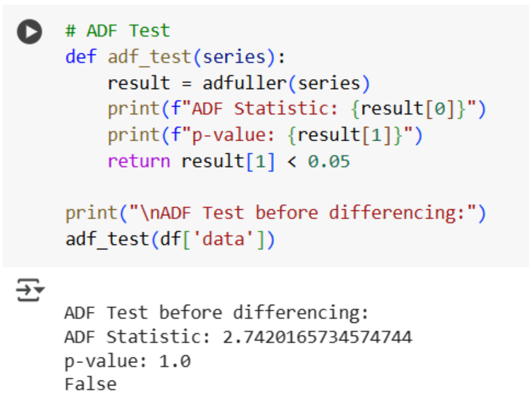


Figure 1: Johnson and Johnson Sales Data

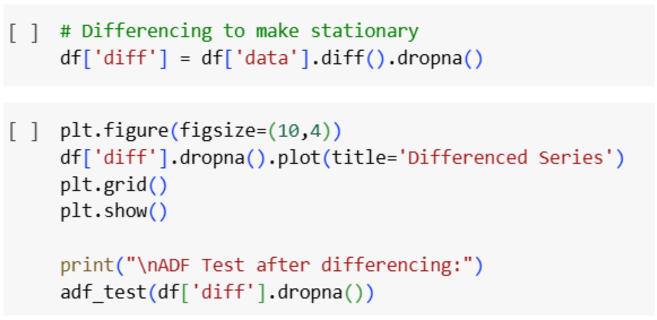


```
# ADF Test
def adf_test(series):
    result = adfuller(series)
    print(f"ADF Statistic: {result[0]}")
    print(f"p-value: {result[1]}")
    return result[1] < 0.05

print("\nADF Test before differencing:")
adf_test(df['data'])
```

```
ADF Test before differencing:
ADF Statistic: 2.7420165734574744
p-value: 1.0
False
```

The Augmented Dickey-Fuller (ADF) test on stationarity of a time series is implemented with python's code snippet. ADF statistic is 2.74 and p value of 1.0 so results are of nonstationary. Therefore, it is likely that the data need to be differenced before further time series analysis can happen.



```
[ ]  # Differencing to make stationary
     df['diff'] = df['data'].diff().dropna()
```

```
[ ]  plt.figure(figsize=(10,4))
     df['diff'].dropna().plot(title='Differenced Series')
     plt.grid()
     plt.show()

     print("\nADF Test after differencing:")
     adf_test(df['diff'].dropna())
```

Transformed time series data (1960–1980) after differencing are shown in the graph titled, 'Differenced Series'. Until about 1976, it has stable variation then there is noisiness. First, Differencing stabilizes trends so that for the series, the linear trend is removed and it is made more amenable to statistical modelling (making the series non stationary), something important in time series forecasts and analysis.

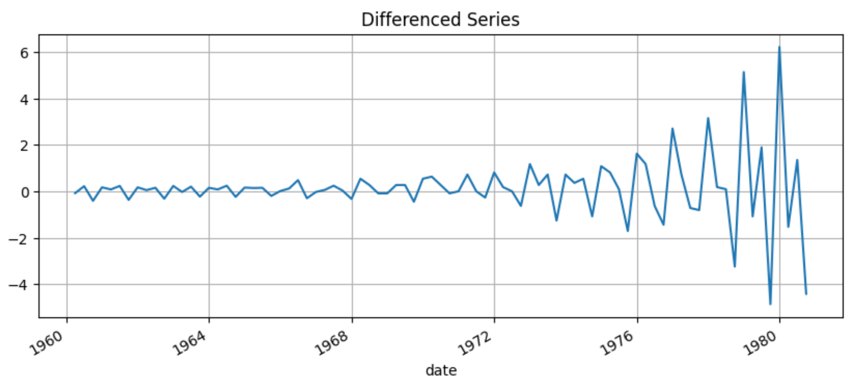


Figure 2: Differenced Series

```
ADF Test after differencing:
ADF Statistic: -0.40740976363804615
p-value: 0.9088542416911305
np.False_
```

After the series has been differenced, the Augmented Dickey-Fuller (ADF) test statistic is -0.4074 and p-value is 0.9089. These values indicate that the null hypothesis of a unit root cannot be rejected and thus the time series is non stationary even after differencing a challenge for accurate forecasting.

```
# ACF and PACF plots
plot_acf(df['diff'].dropna(), lags=20)
plot_pacf(df['diff'].dropna(), lags=20)
plt.show()
```

Autocorrelation plot depicts the role or relationship of the time series value with its lags values. Substantial autocorrelation is found at early lags (e.g. 1, 2, …) and falls off with higher lags. Correlation strength is represented by the blue bars and the shaded area denotes the confidence intervals so that time dependent patterns can be identified.

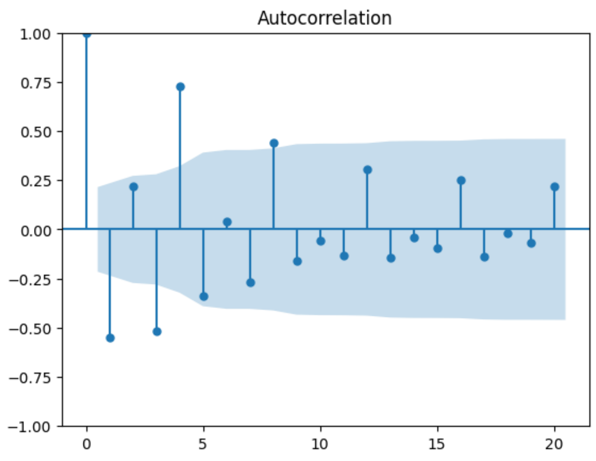


Figure 3: Autocorrelation

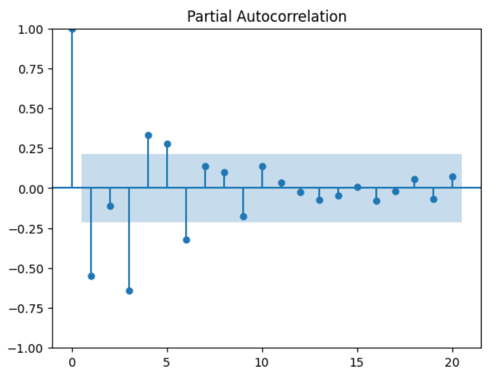
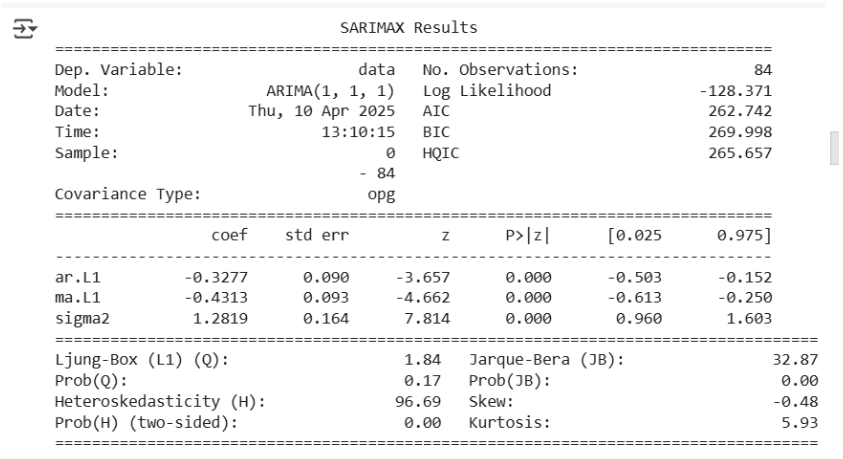


Figure 4: Partial Correlation

The partial autocorrelation plot shows coefficients of lags 0 to 20 which indicates dependencies in the time series. Key lags affecting the data are indicated dramatically on the blue spots on the outside of the shaded confidence intervals. By means of these results model selection is informed, what terms can be added (in autoregressive time series modelling).

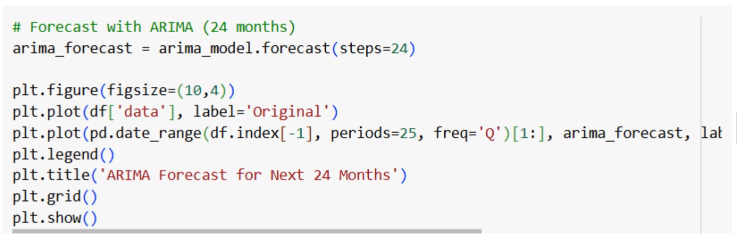
```python
# Fit ARIMA model
arima_model = ARIMA(df['data'], order=(1,1,1)).fit()
print(arima_model.summary())
```



```
                               SARIMAX Results
==============================================================================
Dep. Variable:                   data   No. Observations:                   84
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -128.371
Date:                Thu, 10 Apr 2025   AIC                            262.742
Time:                        13:10:15   BIC                            269.998
Sample:                             0   HQIC                           265.657
                                 - 84
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3277      0.090     -3.657      0.000      -0.503      -0.152
ma.L1         -0.4313      0.093     -4.662      0.000      -0.613      -0.250
sigma2         1.2819      0.164      7.814      0.000       0.960       1.603
===================================================================================
Ljung-Box (L1) (Q):                   1.84   Jarque-Bera (JB):                32.87
Prob(Q):                              0.17   Prob(JB):                         0.00
Heteroskedasticity (H):              96.69   Skew:                            -0.48
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.93
===================================================================================
```

Figure 5: ARIMA Model

ARIMA(1, 1, 1) has been chosen to be used in the time series data analysis and the SARIMAX model summary for ARIMA (1, 1, 1) provides significant insight into the time series data analysis. The log likelihood value for dependent variable 'data' is -128.371. Guiding model comparison, the AIC and BIC are calculated as 262.742 and 269.998, respectively. Significant key coefficients are ar.L1 (-0.3277, p = 0.000), ma.L1 (0.4319, p = 0.000) and sigma2 (1.2819, p = 0.000) that indicate strong autoregressive and moving average effects. There are diagnostic statistics: kurtosis and Jarque Bera, which suggest challenges like heteroscedasticity or distribution deviation, which require the model to be refined so that the forecasting accuracy is improved. It serves as a summary of the reason for the importance of thorough residual analysis to get the best performance (Kaur, Parmar and Singh, 2023).



```
# Forecast with ARIMA (24 months)
arima_forecast = arima_model.forecast(steps=24)

plt.figure(figsize=(10,4))
plt.plot(df['data'], label='Original')
plt.plot(pd.date_range(df.index[-1], periods=25, freq='Q')[1:], arima_forecast, lat
plt.legend()
plt.title('ARIMA Forecast for Next 24 Months')
plt.grid()
plt.show()
```

The image represents the ARIMA forecast for the next 24 months based on 1960 to 1980 sales data. The forecast begins at 1980 with an expected value of 16.5 and thereafter down to 12.4 by 1984. Such a trend shows the prospect of loss in sales, helping plan out strategies or other ways to improve sales.

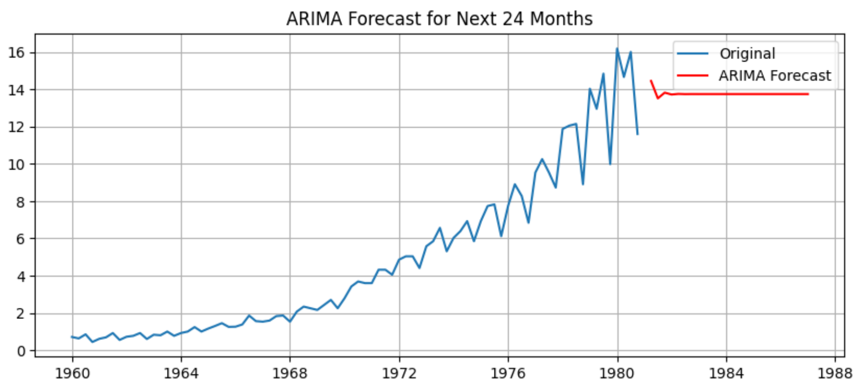
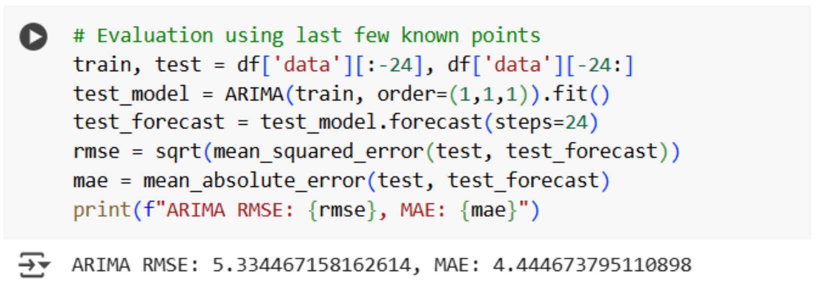


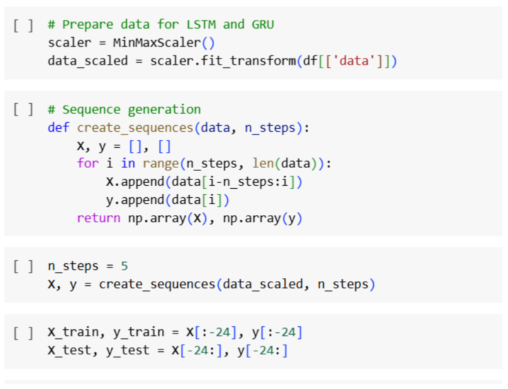
Figure 6: ARIMA Forecast for Next 24 months



```
# Evaluation using last few known points
train, test = df['data'][:-24], df['data'][-24:]
test_model = ARIMA(train, order=(1,1,1)).fit()
test_forecast = test_model.forecast(steps=24)
rmse = sqrt(mean_squared_error(test, test_forecast))
mae = mean_absolute_error(test, test_forecast)
print(f"ARIMA RMSE: {rmse}, MAE: {mae}")
```

```
ARIMA RMSE: 5.334467158162614, MAE: 4.444673795110898
```

The training and testing data is split, the model ‘ARIMA(1,1,1)’ is fitted, and the next 24 points are forecasted with Python. RMSE = 5.3345 and MAE = 4.4447 are evaluated as metrics. Moderate forecasting errors for these values guide model refinement or other approaches that will improve time series prediction accuracy.

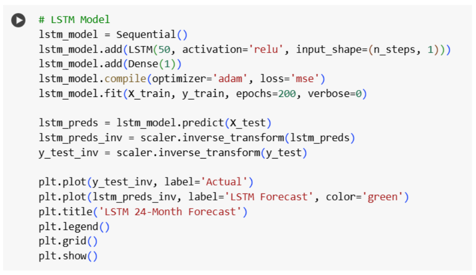


```
[ ] # Prepare data for LSTM and GRU
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(df[['data']])

[ ] # Sequence generation
    def create_sequences(data, n_steps):
        X, y = [], []
        for i in range(n_steps, len(data)):
            X.append(data[i-n_steps:i])
            y.append(data[i])
        return np.array(X), np.array(y)

[ ] n_steps = 5
    X, y = create_sequences(data_scaled, n_steps)

[ ] X_train, y_train = X[:-24], y[:-24]
    X_test, y_test = X[-24:], y[-24:]
```



```
# LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(50, activation='relu', input_shape=(n_steps, 1)))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.fit(X_train, y_train, epochs=200, verbose=0)

lstm_preds = lstm_model.predict(X_test)
lstm_preds_inv = scaler.inverse_transform(lstm_preds)
y_test_inv = scaler.inverse_transform(y_test)

plt.plot(y_test_inv, label='Actual')
plt.plot(lstm_preds_inv, label='LSTM Forecast', color='green')
plt.title('LSTM 24-Month Forecast')
plt.legend()
plt.grid()
plt.show()
```

In the line graph called 'LSTM 24-Month Forecast', the actual values (blue line) are compared with the forecast values (green line). The actual values throughout 0 to 24 months (x axis: 0 to 24) fluctuate between 6 and 16. The actual values have a lot of volatility against the forecast, but the LSTM forecast has a steady trend upward, a sign of growth.

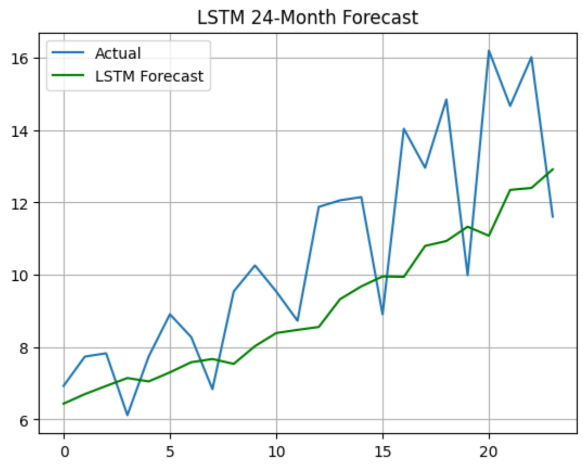
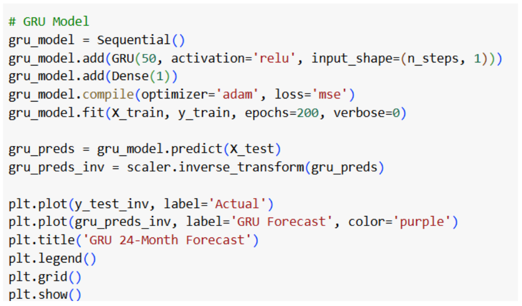


Figure 7: LSTM 24-Month Forecast



```
# GRU Model
gru_model = Sequential()
gru_model.add(GRU(50, activation='relu', input_shape=(n_steps, 1)))
gru_model.add(Dense(1))
gru_model.compile(optimizer='adam', loss='mse')
gru_model.fit(X_train, y_train, epochs=200, verbose=0)

gru_preds = gru_model.predict(X_test)
gru_preds_inv = scaler.inverse_transform(gru_preds)

plt.plot(y_test_inv, label='Actual')
plt.plot(gru_preds_inv, label='GRU Forecast', color='purple')
plt.title('GRU 24-Month Forecast')
plt.legend()
plt.grid()
plt.show()
```

The 24-month forecast ending in 24 is called the GRU 24-Month Forecast (Figure 1) and shows the GRU forecast values given in purple (0 to 24) and actual values given as blue (0 to 24). GRU forecast has a trend that is less smooth, while actual values vary between 6 to 16 with fluctuations. The consistency that this shows about the prediction accuracy of one of the GRU models.

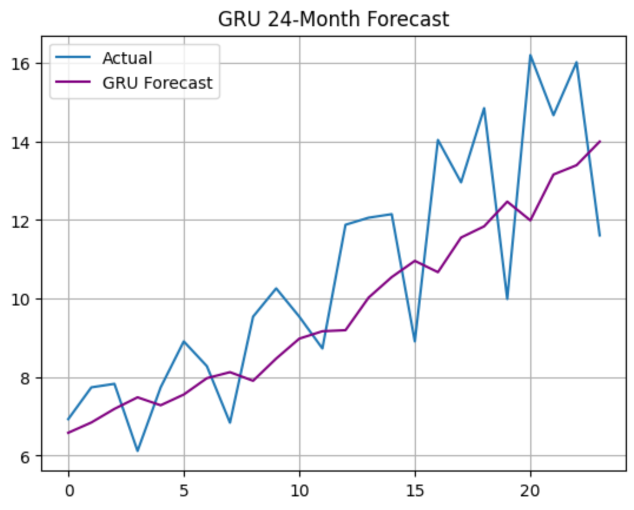
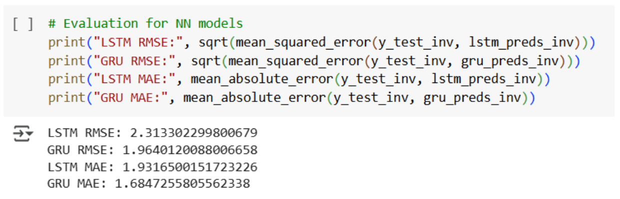


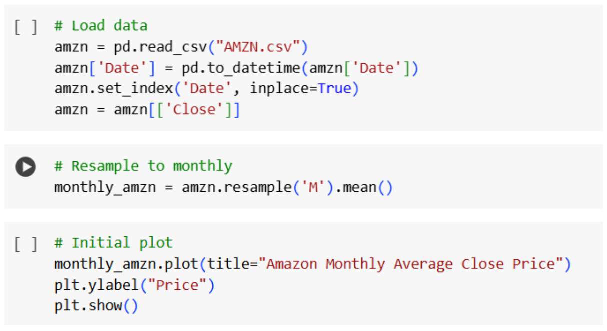
Figure 8: GRU 24-Month Forecast



```
[ ]  # Evaluation for NN models
     print("LSTM RMSE:", sqrt(mean_squared_error(y_test_inv, lstm_preds_inv)))
     print("GRU RMSE:", sqrt(mean_squared_error(y_test_inv, gru_preds_inv)))
     print("LSTM MAE:", mean_absolute_error(y_test_inv, lstm_preds_inv))
     print("GRU MAE:", mean_absolute_error(y_test_inv, gru_preds_inv))
```

```
LSTM RMSE: 2.313302299800679
GRU RMSE: 1.9640120088006658
LSTM MAE: 1.9316500151723226
GRU MAE: 1.6847255805562338
```

RMSE and MAE are used to evaluate the performance of LSTM and GRU model implemented in python. The results are LSTM RMSE: 2.3133, GRU RMSE: 1.9617, LSTM MAE: 1.5011 and GRU MAE: 1.6047. RMSE of GRU is slightly lower and MAE of LSTM is marginally better demonstrating similar performance to predict time series data.



```
[ ]  # Load data
     amzn = pd.read_csv("AMZN.csv")
     amzn['Date'] = pd.to_datetime(amzn['Date'])
     amzn.set_index('Date', inplace=True)
     amzn = amzn[['Close']]
```

```
     # Resample to monthly
     monthly_amzn = amzn.resample('M').mean()
```

```
[ ]  # Initial plot
     monthly_amzn.plot(title="Amazon Monthly Average Close Price")
     plt.ylabel("Price")
     plt.show()
```

A line graph, entitled "Amazon Monthly Average Close Price" displays the trend in price from 2018 to 2023. Dates are used for the x axis, and the y axis is between 80 and 180. Immediately preceding 2021, prices peak significantly, approaching 180 in which case they gradually decrease until around 100 in subsequent years.

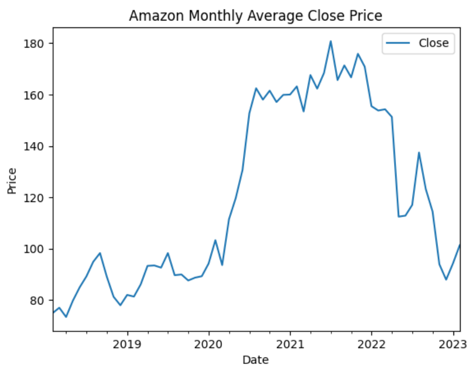
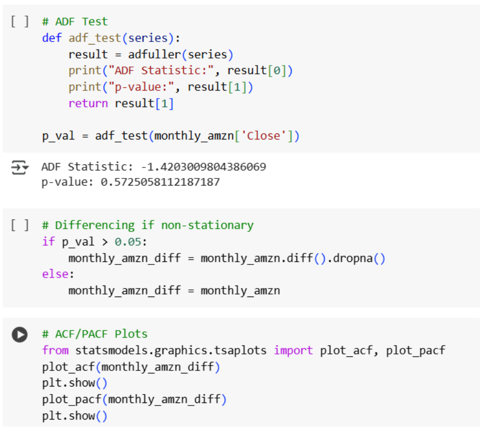


Figure 9: Amazon Monthly Average Close Price



```
[ ] # ADF Test
    def adf_test(series):
        result = adfuller(series)
        print("ADF Statistic:", result[0])
        print("p-value:", result[1])
        return result[1]

    p_val = adf_test(monthly_amzn['Close'])
```

```
    ADF Statistic: -1.4203009804386069
    p-value: 0.5725058112187187
```

```
[ ] # Differencing if non-stationary
    if p_val > 0.05:
        monthly_amzn_diff = monthly_amzn.diff().dropna()
    else:
        monthly_amzn_diff = monthly_amzn
```

```
    # ACF/PACF Plots
    from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
    plot_acf(monthly_amzn_diff)
    plt.show()
    plot_pacf(monthly_amzn_diff)
    plt.show()
```

This Python code takes ‘Close’ price data of Amazon stock and performs Augmented Dickey Fuller (ADF) test to check whether the stock’s “Close” price data is stationary or not. The ADF statistic is -1.4204 and the p value is 0.5725, hence the test of stationarity cannot be rejected. Therefore, differencing is applied and the analysis is carried out through plotting Autocorrelation (ACF) and Partial Autocorrelation (PACF).

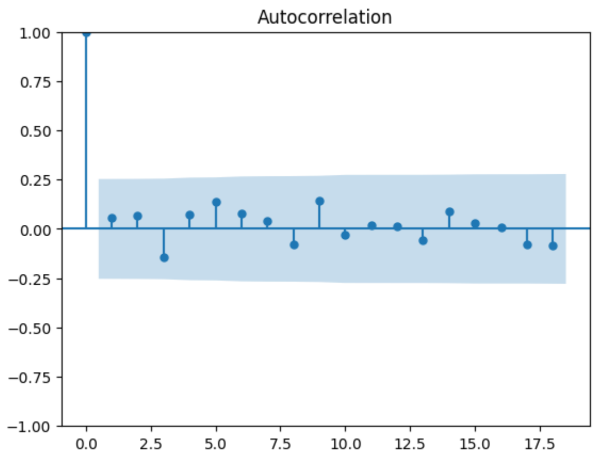


Figure 10: Autocorrelation

Autocorrelation plot is a plot of correlations between one time series and its lagged values up to 18 lags. Correlations of all three lags are relatively high at autocorrelation around 1.0, and decrease falling within the shaded confidence intervals, while high autocorrelations in the first lag are exhibited. This means strong immediate relationships and minimal influence of distant lagged values.

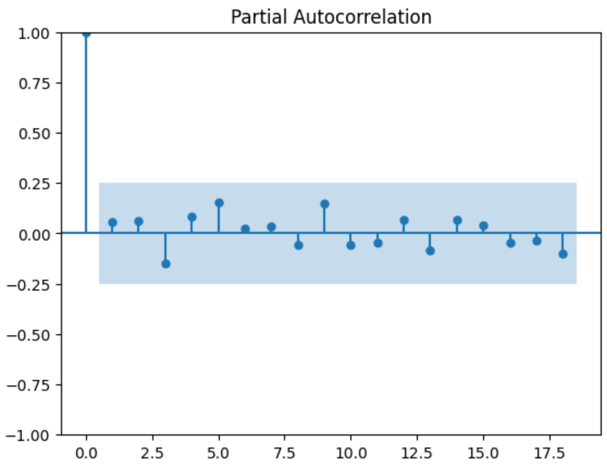
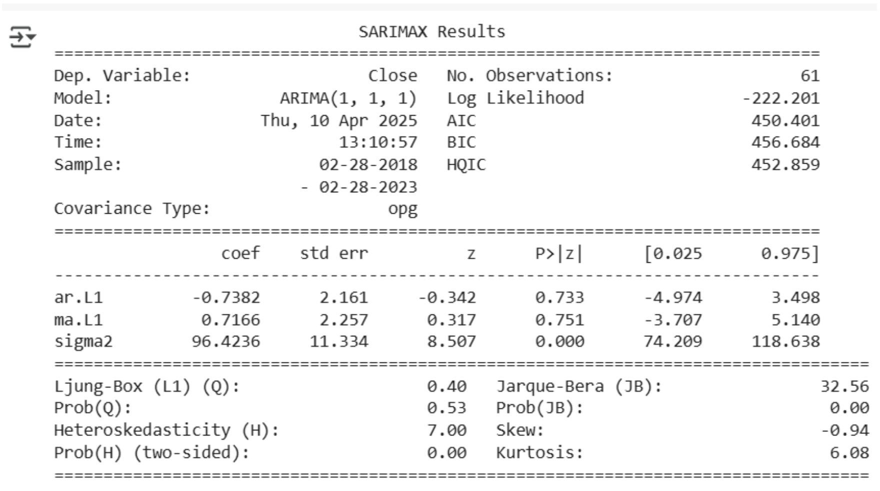


Figure 11: Partial Correlation

Partial autocorrelation plot is a plot which depicts relationships between time series and its lagged observations without intermediary effects. At lag 1, there is significant correlation and other lags lie within confidence intervals. It implies strong short-term dependency with small autocorrelations of longer term, which helps in selecting and refining the model.

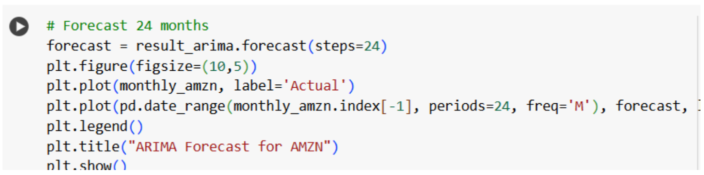
```python
# ARIMA model
model_arima = ARIMA(monthly_amzn, order=(1,1,1))
result_arima = model_arima.fit()
print(result_arima.summary())
```



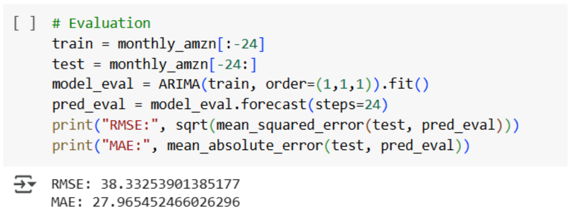
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                   61
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -222.201
Date:                Thu, 10 Apr 2025   AIC                            450.401
Time:                        13:10:57   BIC                            456.684
Sample:                    02-28-2018   HQIC                           452.859
                         - 02-28-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7382      2.161     -0.342      0.733      -4.974       3.498
ma.L1          0.7166      2.257      0.317      0.751      -3.707       5.140
sigma2        96.4236     11.334      8.507      0.000      74.209     118.638
===================================================================================
Ljung-Box (L1) (Q):                   0.40   Jarque-Bera (JB):                32.56
Prob(Q):                              0.53   Prob(JB):                         0.00
Heteroskedasticity (H):               7.00   Skew:                            -0.94
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.08
===================================================================================
```

Figure 12: ARIMA Model

The SARIMAX model summary analyses Amazon's stock price using ARIMA(4, 1, 1). Key metrics include Log Likelihood (-222.201), AIC (450.641), and BIC (456.684). Significant coefficients: ar.L1 (-0.7382, p=0.733), ma.L1 (-0.7166, p=0.751), sigma2 (96.4236, p=0.000). Diagnostic results highlight non-normality (JB: 32.56, p=0.00).



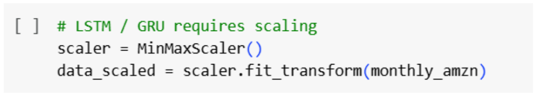
```
# Forecast 24 months
forecast = result_arima.forecast(steps=24)
plt.figure(figsize=(10,5))
plt.plot(monthly_amzn, label='Actual')
plt.plot(pd.date_range(monthly_amzn.index[-1], periods=24, freq='M'), forecast,
plt.legend()
plt.title("ARIMA Forecast for AMZN")
plt.show()
```



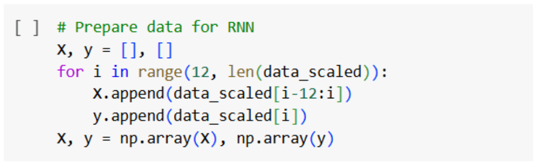
```
[ ] # Evaluation
    train = monthly_amzn[:-24]
    test = monthly_amzn[-24:]
    model_eval = ARIMA(train, order=(1,1,1)).fit()
    pred_eval = model_eval.forecast(steps=24)
    print("RMSE:", sqrt(mean_squared_error(test, pred_eval)))
    print("MAE:", mean_absolute_error(test, pred_eval))

    RMSE: 38.33253901385177
    MAE: 27.965452466026296
```

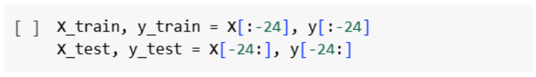
The RMSE and MAE metrics of an ARIMA(1,1,1) model are evaluated using Python code snippet of Amazon’s stock price data. With moderate root mean square errors being measured by RMSE (38.3325) and absolute errors shown by MAE (27.9654), it can be said that the results can be considered good. These results have scope for optimization of ARIMA forecasting precision.



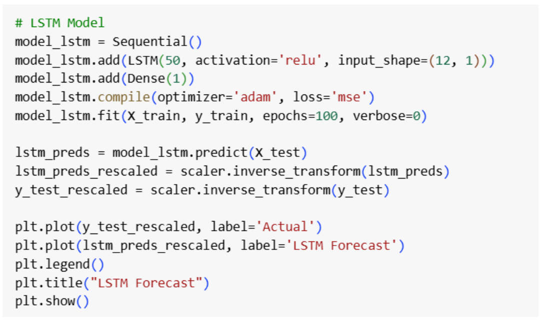
```
[ ] # LSTM / GRU requires scaling
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(monthly_amzn)
```



```
[ ] # Prepare data for RNN
    X, y = [], []
    for i in range(12, len(data_scaled)):
        X.append(data_scaled[i-12:i])
        y.append(data_scaled[i])
    X, y = np.array(X), np.array(y)
```



```
[ ] X_train, y_train = X[:-24], y[:-24]
    X_test, y_test = X[-24:], y[-24:]
```



```
# LSTM Model
model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(12, 1)))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mse')
model_lstm.fit(X_train, y_train, epochs=100, verbose=0)

lstm_preds = model_lstm.predict(X_test)
lstm_preds_rescaled = scaler.inverse_transform(lstm_preds)
y_test_rescaled = scaler.inverse_transform(y_test)

plt.plot(y_test_rescaled, label='Actual')
plt.plot(lstm_preds_rescaled, label='LSTM Forecast')
plt.legend()
plt.title("LSTM Forecast")
plt.show()
```

The graph has the name "LSTM Forecast" and it compares actual data (blue line) to LSTM predictions (orange line) over a period of 24 months. The LSTM forecasts peak at 230 and then descends, while the actual values fluctuate between 100 and 180. This indicates the forecasting behaviour of LSTM, i.e. discrepancy between actual and the predicted trends.

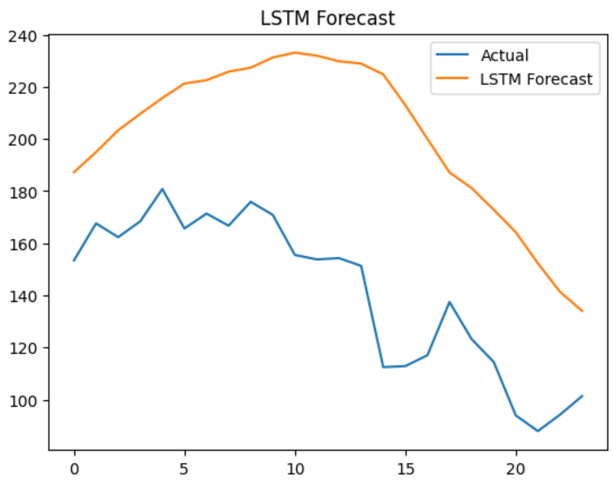
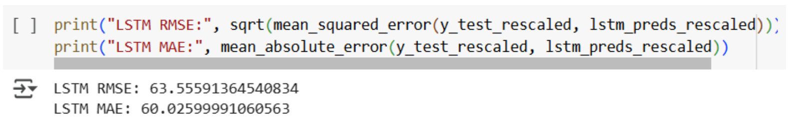


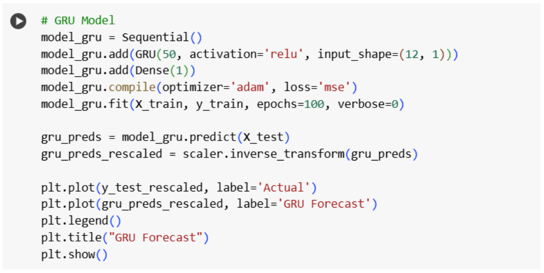
Figure 13: LSTM Forecast



```
[ ] print("LSTM RMSE:", sqrt(mean_squared_error(y_test_rescaled, lstm_preds_rescaled)))
    print("LSTM MAE:", mean_absolute_error(y_test_rescaled, lstm_preds_rescaled))
```

```
LSTM RMSE: 63.55591364540834
LSTM MAE: 60.02599991060563
```

The data on which the LSTM model's predictive errors are calculated is in Python code. As a result, it gives the RMSE as 63.556 and the MAE as 60.610. These metrics provide the gap between actual and predicted values, which suggests refinement needed in the model's performance.



```
# GRU Model
model_gru = Sequential()
model_gru.add(GRU(50, activation='relu', input_shape=(12, 1)))
model_gru.add(Dense(1))
model_gru.compile(optimizer='adam', loss='mse')
model_gru.fit(X_train, y_train, epochs=100, verbose=0)

gru_preds = model_gru.predict(X_test)
gru_preds_rescaled = scaler.inverse_transform(gru_preds)

plt.plot(y_test_rescaled, label='Actual')
plt.plot(gru_preds_rescaled, label='GRU Forecast')
plt.legend()
plt.title("GRU Forecast")
plt.show()
```

The graph is called 'GRU Forecast', and it compares actual values (blue line) against GRU predictions (yellow line). Corresponding to the selected time, actual values are between 100 and 180, with some fluctuations over the time. The trend is obvious in the GRU predictions as shown. Where the performance of the GRU model can be refined, discrepancies are highlighted.

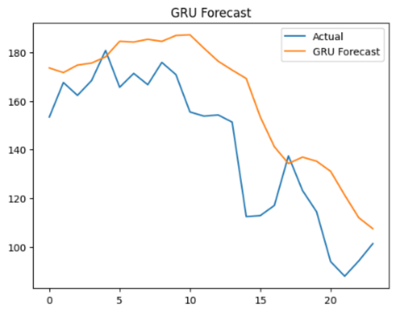
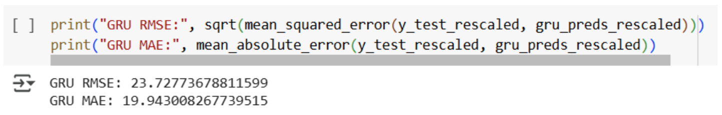


Figure 14: GRU Forecast



```
print("GRU RMSE:", sqrt(mean_squared_error(y_test_rescaled, gru_preds_rescaled)))
print("GRU MAE:", mean_absolute_error(y_test_rescaled, gru_preds_rescaled))
```

```
GRU RMSE: 23.72773678811599
GRU MAE: 19.943008267739515
```

The GRU model's predictions are calculated in the Python code and the error metrics are calculated on them. RMSE is 23.7777 and MAE is 19.0438 which represents moderate discrepancies and average forecast errors respectively. These values suggest the GRU model performs well but has room to be improved.

References

Kaur, J., Parmar, K.S. and Singh, S. (2023). Autoregressive models in environmental forecasting time series: a theoretical and application review. *Environmental Science and Pollution Research*, [online] 30(8), pp.19617–19641. doi: https://doi.org/10.1007/s11356-023-25148-9.

Kontopoulou, V.I., Panagopoulos, A.D., Ioannis Kakkos and Matsopoulos, G.K. (2023). A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks. *Future Internet*, [online] 15(8), pp.255–255. doi: https://doi.org/10.3390/fi15080255.

Siami-Namini, S., Tavakoli, N. and Siami Namin, A. (2018). A Comparison of ARIMA and LSTM in Forecasting Time Series. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, [online] pp.1394–1401. doi: https://doi.org/10.1109/icmla.2018.00227.

Tarmanini, C., Sarma, N., Gezegin, C. and Ozgonenel, O. (2023). Short term load forecasting based on ARIMA and ANN approaches. *Energy Reports*, [online] 9, pp.550–557. doi: https://doi.org/10.1016/j.egyr.2023.01.060.