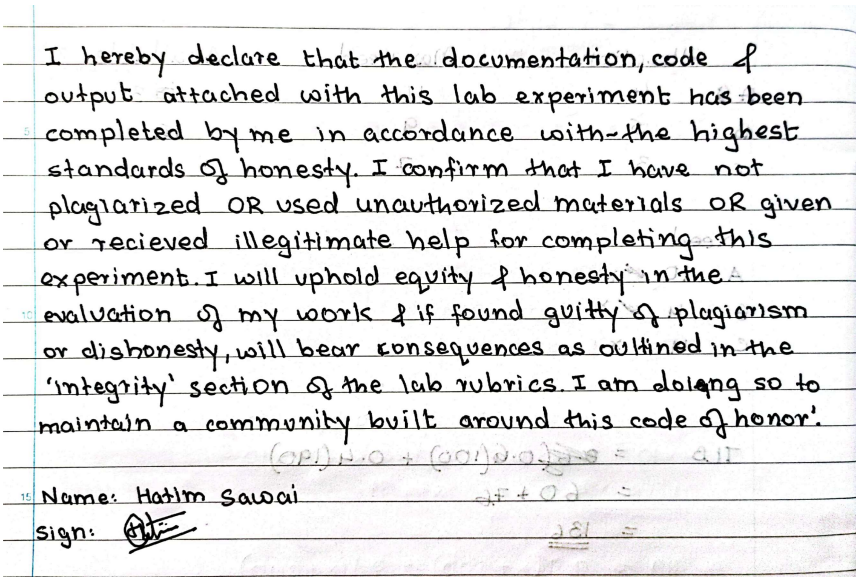
Name	Hatim Sawai
UID No.	2021300108

Experiment 10

HONOR PLEDGE	
PROBLEM STATEMENT	**Forecasting using ARIMA(p, d, q):** 1. Check stationarity of dataset using Augmented Dickey-Fuller test. If data is non-stationary, identify the value of 'd' which converts data to stationary data 2. Identify coefficients 'p' and 'q' using Auto-correlation Function (ACF) & Partial auto-correlation function (PACF) plots 3. Fit an ARIMA model on 80% of the historic data (train) using the p,q and d parameters and use the recent 20% data as 'test' 4. Evaluate the fitted model on various statistical metrics for error on 'train' and 'test' 5. Assess the model on metrics that calculate goodness of fit on 'train' and 'test' 6. Compare the performance of this model with your previously trained OLS model in Experiment 8 7. Compute Theil's coefficient of the 2 forecasts (OLS, ARIMA) for any one stock forecast
THEORY	**1. Check stationarity of dataset using Augmented Dickey-Fuller test:** The Augmented Dickey-Fuller (ADF) test is a statistical test used to determine whether a time series is stationary or not. Stationarity is an important property of time series data, where the statistical properties such as mean, variance,

and autocorrelation do not change over time. The null hypothesis of the ADF test is that the time series is non-stationary. If the p-value obtained from the test is less than a critical value, we reject the null hypothesis and conclude that the time series is stationary.

2. Identify coefficients 'p' and 'q' using Auto-correlation Function (ACF) & Partial auto-correlation function (PACF) plots:
The Auto-correlation Function (ACF) and Partial auto-correlation function (PACF) plots are used to identify the order of the ARIMA model.
- ACF plot: A plot of the autocorrelation of a time series against the number of lagged observations in the series. - PACF plot: A plot of the partial autocorrelation of a time series against the number of lagged observations in the series.
In an ACF plot, the correlation between the series and its lags are measured. In a PACF plot, the correlation between the series and its lags are measured after removing the variations already explained by the intervening comparisons.

3. Fit an ARIMA model on 80% of the historic data (train) using the p,q and d parameters and use the recent 20% data as 'test':
ARIMA (AutoRegressive Integrated Moving Average) is a popular time series forecasting model. It consists of three components: - AutoRegressive (AR) term (p): This is the number of lag observations included in the model. - Integrated (I) term (d): This is the number of times the raw observations are differenced. - Moving Average (MA) term (q): This is the size of the moving average window.

4. Evaluate the fitted model on various statistical metrics for error on 'train' and 'test':
Various statistical metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) are used to evaluate the performance of the ARIMA model on both the training and testing datasets.

5. Assess the model on metrics that calculate goodness of fit on 'train' and 'test':
Goodness of fit metrics such as R-squared, Adjusted R-squared, and Akaike Information Criterion (AIC) are used to assess the quality of the ARIMA model on both the training and testing datasets.

6. Compare the performance of this model with your previously trained OLS model in Exp 8:
The performance of the ARIMA model is compared with the previously trained Ordinary Least Squares (OLS) model

	from Experiment 8 using various evaluation metrics such as MSE, MAE, RMSE, R-squared, Adjusted R-squared, and AIC. **7. Compute Theil's coefficient of the 2 forecasts (OLS, ARIMA) for any one stock forecast:** Theil's coefficient is a measure of forecast accuracy. It compares the accuracy of forecasts from two different models (in this case, OLS and ARIMA). It is calculated as the ratio of the root mean squared error of a model to the root mean squared error of a naive model (e.g., a random walk or a simple average). The formula for Theil's coefficient (U) is: U = sqrt((MSE_model / n) / (MSE_naive / n)) where MSE_model is the mean squared error of the model, MSE_naive is the mean squared error of the naive model, and n is the number of observations. A value of Theil's U close to 1 indicates good forecast accuracy, whereas a value much greater than 1 indicates poor forecast accuracy**.**

1. Importing Libraries

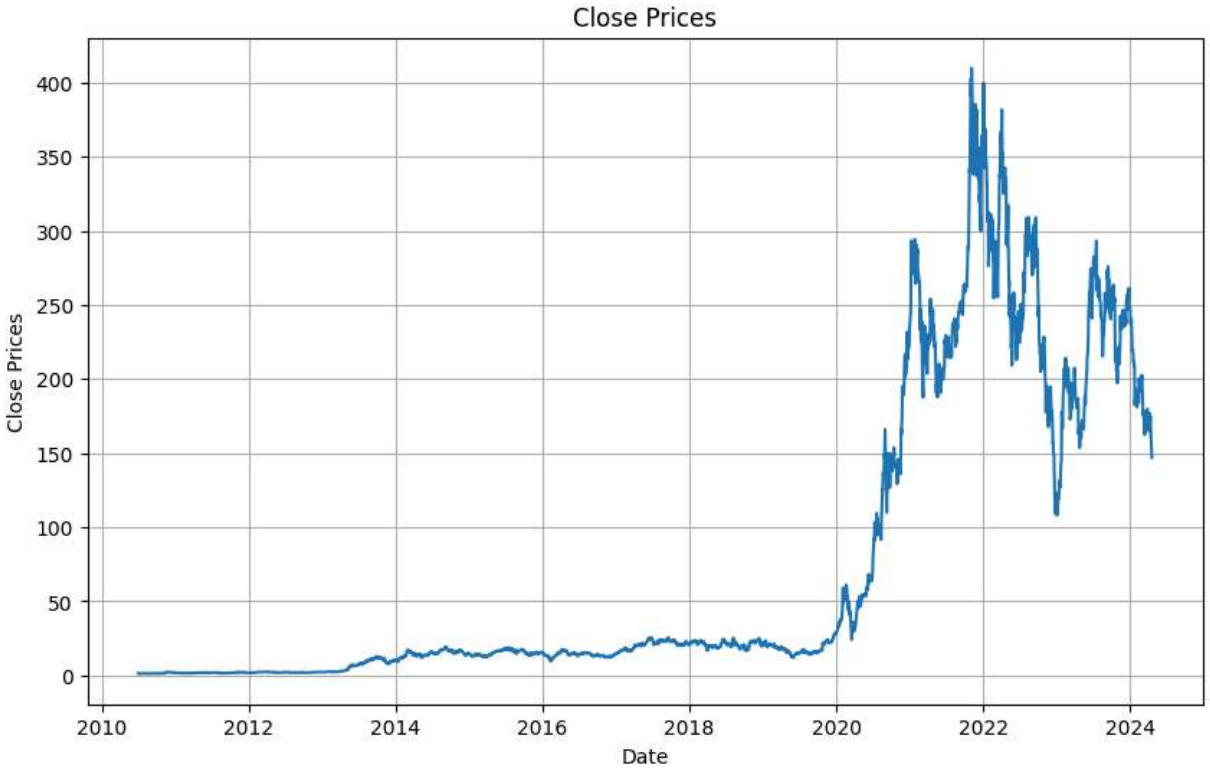
In [ ]:
```
import math
import numpy as np
import pandas as pd
import seaborn as sns
import yfinance as yf
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pmdarima.arima import auto_arima
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
```

2. Stock Data

In [ ]:
```
# Function to fetch stock data
def fetch_stock_data(ticker, start_date, end_date):
    stock_data = yf.download(ticker, start=start_date, end=end_date)
    return stock_data
```

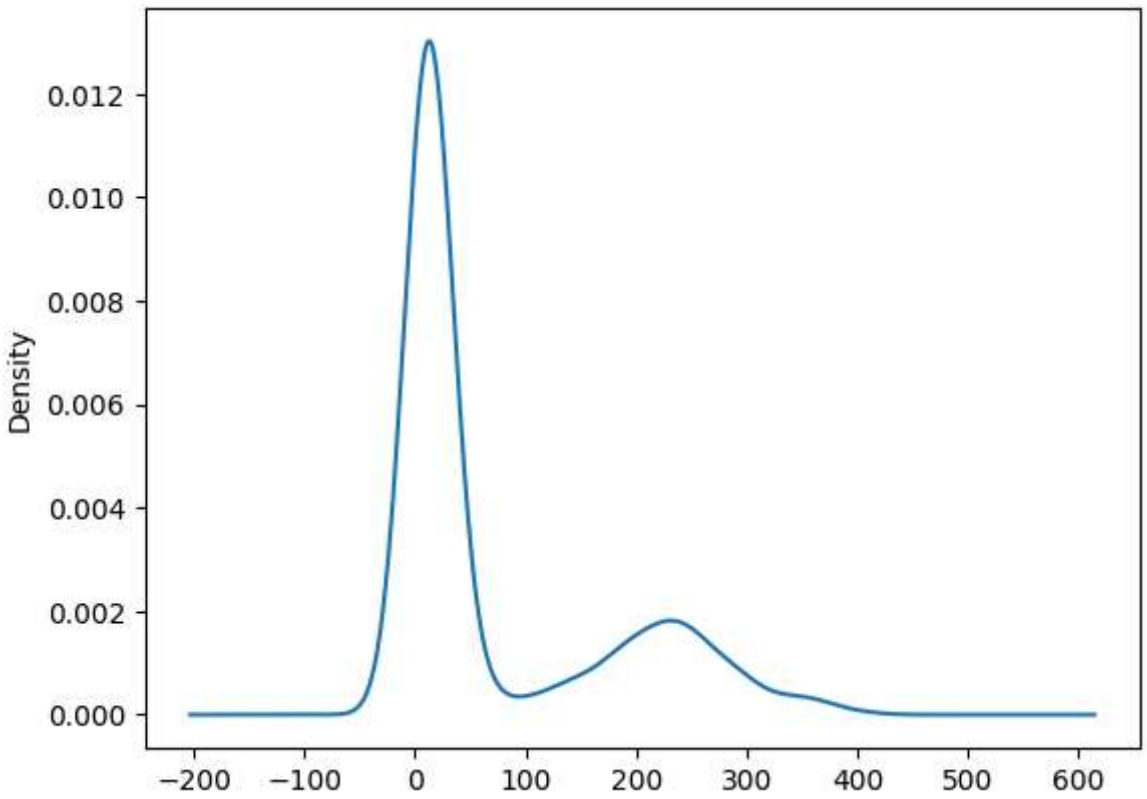
In [ ]:
```
# Define start and end dates, and tickers
start_date = "2010-01-01"
end_date = "2024-04-20"
tickers = ["TSLA"]

# Fetch stock data
stock_data = {}
for ticker in tickers:
    stock_data[ticker] = fetch_stock_data(ticker, start_date, end_date)

# plot close price
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.xlabel("Date")
plt.ylabel("Close Prices")
plt.plot(stock_data["TSLA"]["Close"], label="TSLA")
plt.title("Close Prices")
plt.show()
```

```
[*********************100%%**********************]  1 of 1 completed
```



In [ ]:
```
# Distribution of the dataset
df_close = stock_data["TSLA"]["Close"]
df_close.plot(kind="kde")
```

Out[ ]: <Axes: ylabel='Density'>



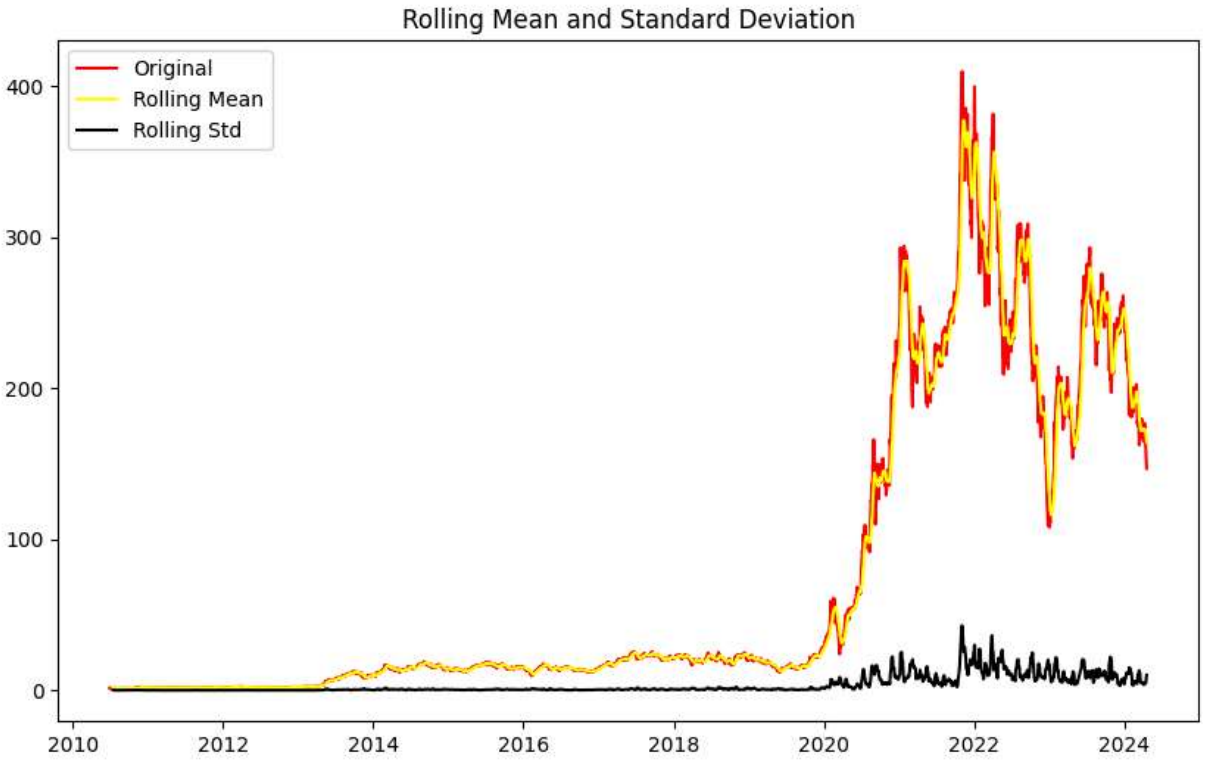
3. Augmented Dickey-Fuller Test

In [ ]:
```
# Check stationarity using ADF test
def check_stationarity(data):
    result = adfuller(data)
    print("ADF Statistic:", result[0])
    print("p-value:", result[1])
    print("Lags Used:", result[2])
    print("Number of Observations Used:", result[3])
    print("Critical Values:")
    for key, value in result[4].items():
        print(f"   {key}: {value}")
    if result[1] > 0.05:
        print("Data is non-stationary.")
    else:
        print("Data is stationary.")
```

In [ ]:
```
# Example: Check stationarity for TSLA stock
tsla_close = stock_data["TSLA"]["Close"].dropna()
check_stationarity(tsla_close)
```

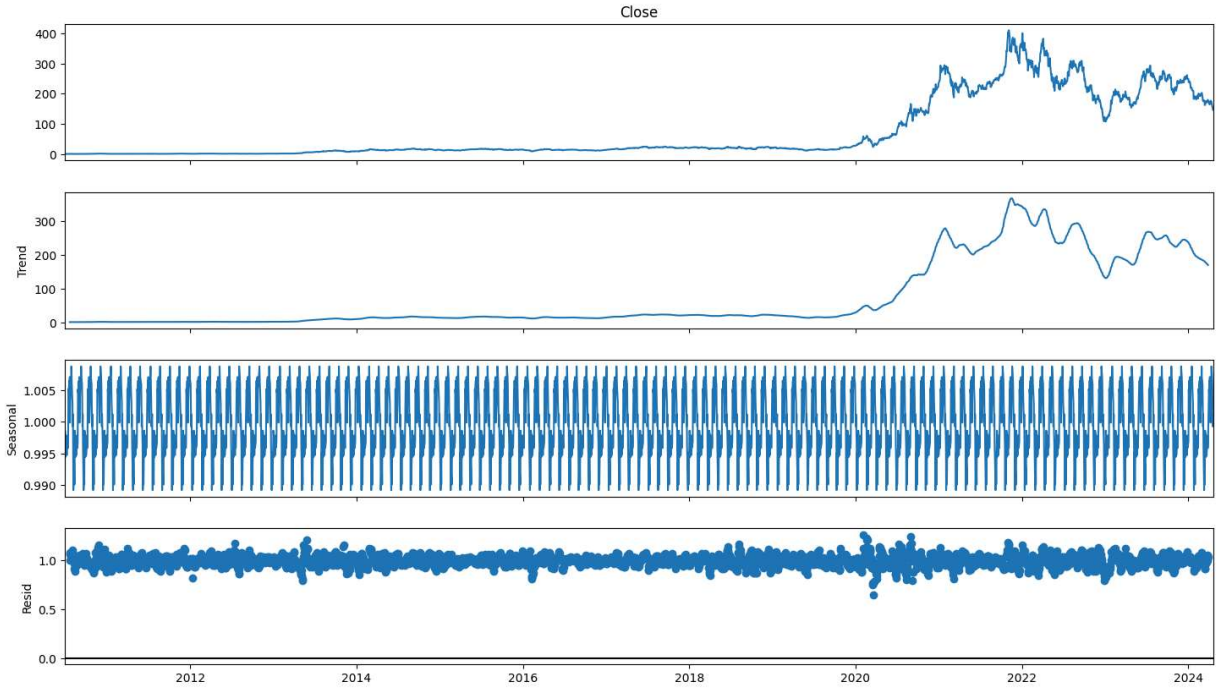
```
ADF Statistic: -1.3975426734465495
p-value: 0.5834536231157117
Lags Used: 29
Number of Observations Used: 3446
Critical Values:
   1%: -3.432249064957662
   5%: -2.8623790980971378
   10%: -2.567216667193267
Data is non-stationary.
```

In [ ]:
```
# Determing rolling statistics
rolmean = tsla_close.rolling(12).mean()
rolstd = tsla_close.rolling(12).std()
# Plot rolling statistics:
plt.figure(figsize=(10, 6))
plt.plot(tsla_close, color="red", label="Original")
plt.plot(rolmean, color="yellow", label="Rolling Mean")
plt.plot(rolstd, color="black", label="Rolling Std")
plt.legend(loc="best")
plt.title("Rolling Mean and Standard Deviation")
plt.show(block=False)
```



In [ ]:
```
# To separate the trend and the seasonality from a time series,
# we can decompose the series using the following code.
result = seasonal_decompose(tsla_close, model="multiplicative", period=30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(16, 9)
```

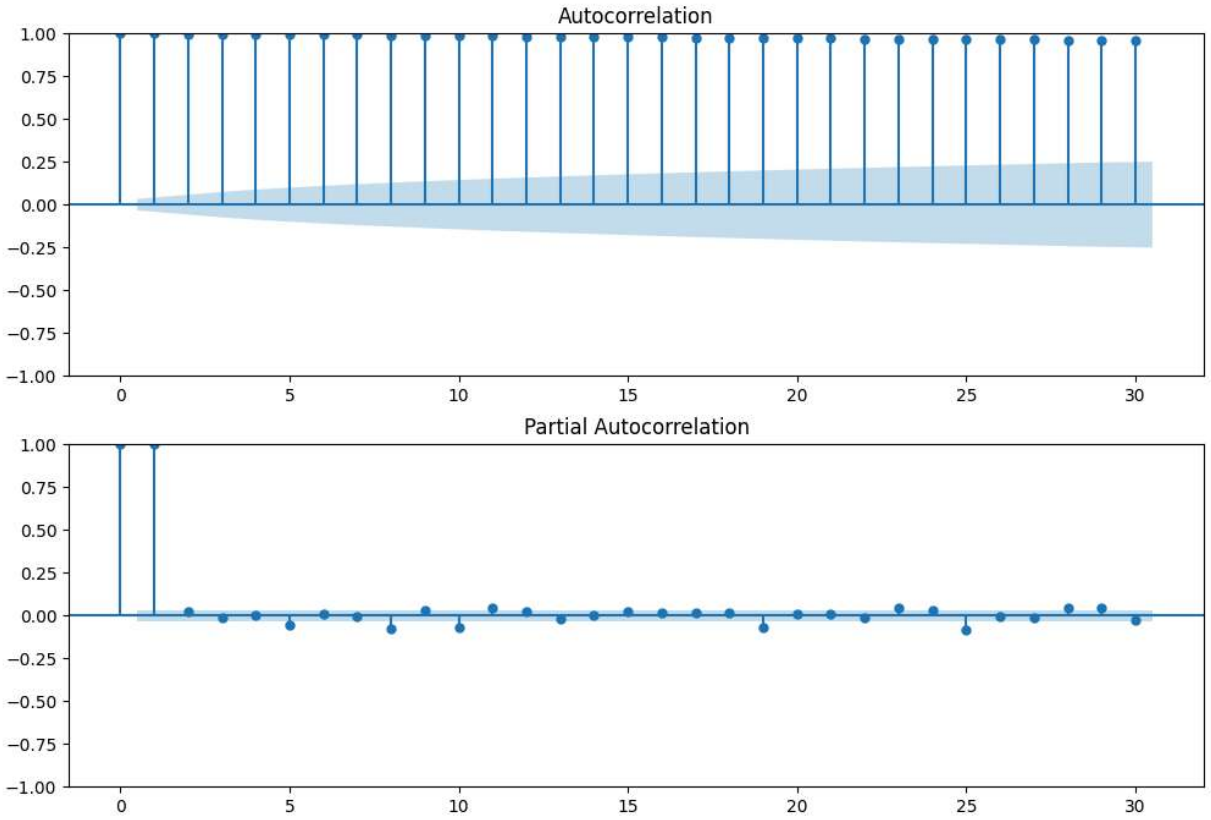
```
<Figure size 640x480 with 0 Axes>
```



4. ACF and PACF plots

In [ ]:
```
# Step 2: Identify 'p' and 'q' using ACF and PACF plots
def plot_acf_pacf(data):
    fig, ax = plt.subplots(2, 1, figsize=(12, 8))
    plot_acf(data, lags=30, ax=ax[0])
    plot_pacf(data, lags=30, ax=ax[1])
    plt.show()
```

In [ ]:
```
# Example: Plot ACF and PACF for TSLA stock
plot_acf_pacf(tsla_close)
```

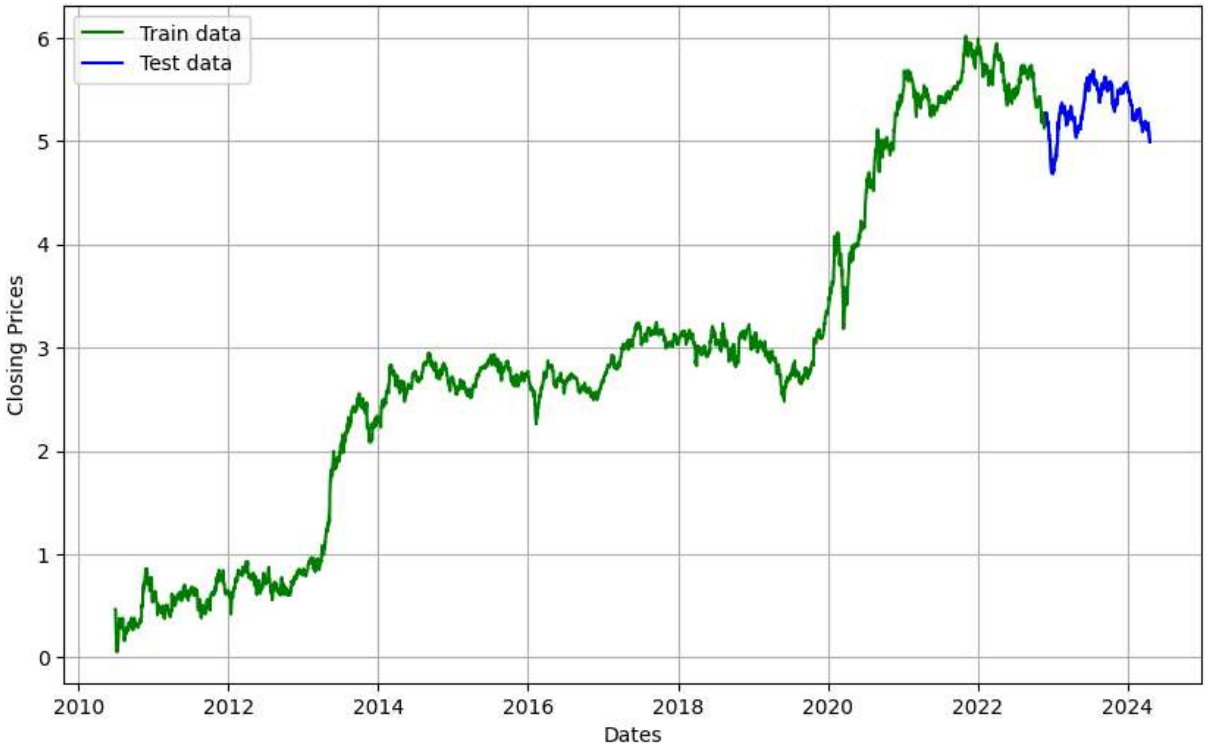


5. ARIMA Model Fitting

In [ ]:
```
# split data into train and training set
df_log = np.log(df_close)
train_data, test_data = (
    df_log[3 : int(len(df_log) * 0.9)],
    df_log[int(len(df_log) * 0.9) :],
)
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.xlabel("Dates")
plt.ylabel("Closing Prices")
plt.plot(df_log, "green", label="Train data")
```

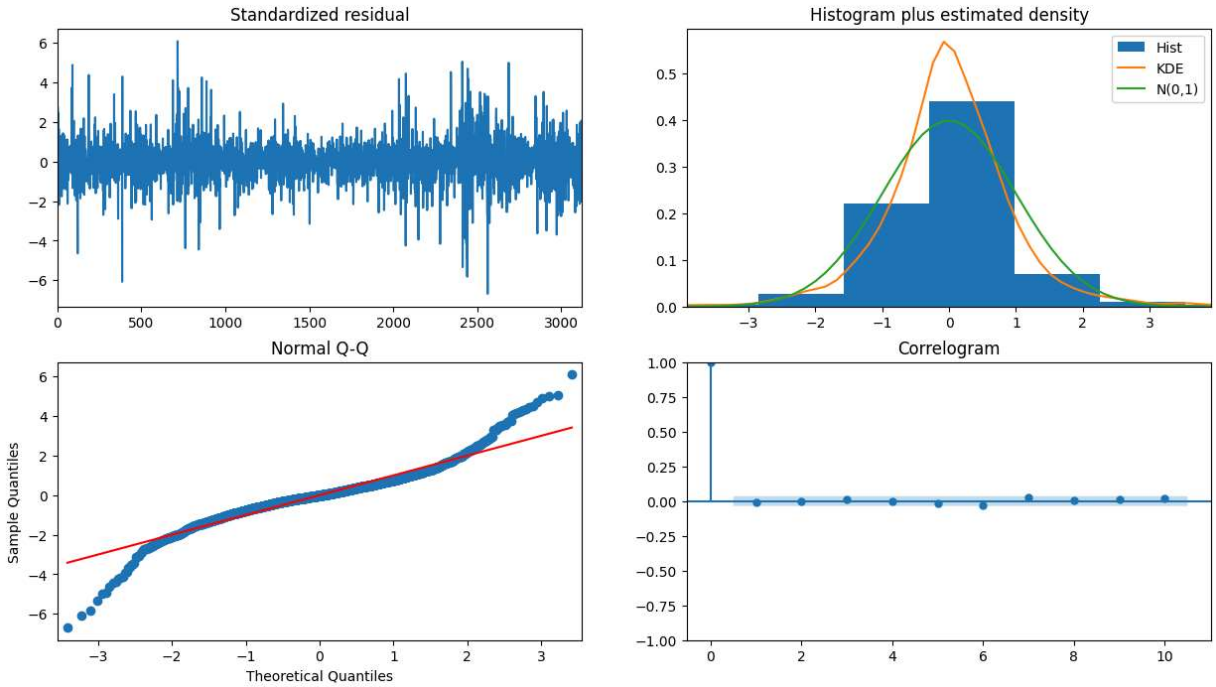
```
plt.plot(test_data, "blue", label="Test data")
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x223aee72790>



In [ ]:
```
model_autoARIMA = auto_arima(
    train_data,
    start_p=0,
    start_q=0,
    test="adf",  # use adftest to find optimal 'd'
    max_p=3,
    max_q=3,  # maximum p and q
    m=1,  # frequency of series
    d=None,  # let model determine 'd'
    seasonal=False,  # No Seasonality
    start_P=0,
    D=0,
    trace=True,
    error_action="ignore",
    suppress_warnings=True,
    stepwise=True,
)
print(model_autoARIMA.summary())
model_autoARIMA.plot_diagnostics(figsize=(15, 8))
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-11966.745, Time=0.42 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-11965.204, Time=0.59 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-11965.203, Time=0.74 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-11962.567, Time=0.31 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-11963.168, Time=1.62 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 3.740 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 3125
Model:               SARIMAX(0, 1, 0)   Log Likelihood                5985.373
Date:                Mon, 22 Apr 2024   AIC                         -11966.745
Time:                        21:28:38   BIC                         -11954.651
Sample:                             0   HQIC                        -11962.404
                                 - 3125
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0016      0.001      2.487      0.013       0.000       0.003
sigma2         0.0013   1.69e-05     75.104      0.000       0.001       0.001
===================================================================================
Ljung-Box (L1) (Q):                   0.46   Jarque-Bera (JB):              3550.22
Prob(Q):                              0.50   Prob(JB):                         0.00
Heteroskedasticity (H):               1.31   Skew:                            -0.01
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.22
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

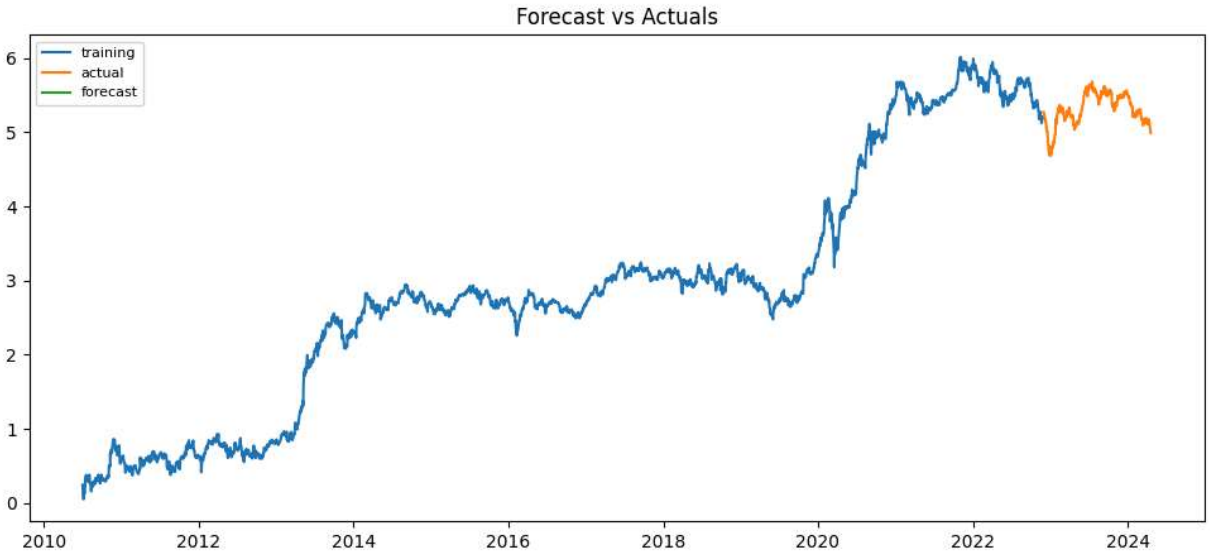


In [ ]:
```
# Build Model
model = ARIMA(train_data, order=(0, 1, 0))
fitted = model.fit()
print(fitted.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 3125
Model:                 ARIMA(0, 1, 0)   Log Likelihood                5982.284
Date:                Mon, 22 Apr 2024   AIC                         -11962.567
Time:                        21:29:44   BIC                         -11956.520
Sample:                             0   HQIC                        -11960.397
                                - 3125
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2         0.0013   1.69e-05     75.004      0.000       0.001       0.001
===================================================================================
Ljung-Box (L1) (Q):                   0.46   Jarque-Bera (JB):              3550.22
Prob(Q):                              0.50   Prob(JB):                         0.00
Heteroskedasticity (H):               1.31   Skew:                            -0.01
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.22
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

6. Model Evaluation

In [ ]:
```
# Forecast
fc = fitted.forecast(steps=len(test_data), alpha=0.05)
fc_series = pd.Series(fc, index=test_data.index)
lower_series = pd.Series(fc - 1.96, index=test_data.index)
upper_series = pd.Series(fc + 1.96, index=test_data.index)

# Plot
plt.figure(figsize=(12, 5), dpi=100)
plt.plot(train_data, label="training")
plt.plot(test_data, label="actual")
plt.plot(fc_series, label="forecast")
plt.fill_between(lower_series.index, lower_series, upper_series, color="k", alpha=0
plt.title("Forecast vs Actuals")
plt.legend(loc="upper left", fontsize=8)
plt.show()
```



In [ ]:
```
# Reset the index of both test_data and fc to ensure alignment
test_data_reset_index = test_data.reset_index(drop=True)
fc_reset_index = fc.reset_index(drop=True)

# Create a boolean mask that is True where fc_reset_index is not zero
non_zero_mask = fc_reset_index != 0

# Use the mask to filter out zero values from both test_data_reset_index and fc_res
test_data_non_zero = test_data_reset_index[non_zero_mask]
fc_non_zero = fc_reset_index[non_zero_mask]

# Calculate statistical metrics
mse = mean_squared_error(test_data_non_zero, fc_non_zero)
print("MSE: " + str(mse))
mae = mean_absolute_error(test_data_non_zero, fc_non_zero)
print("MAE: " + str(mae))
rmse = math.sqrt(mean_squared_error(test_data_non_zero, fc_non_zero))
print("RMSE: " + str(rmse))

# Calculate MAPE without zero values
mape_non_zero = (
    np.mean(np.abs(fc_non_zero - test_data_non_zero) / np.abs(test_data_non_zero))
)
print("MAPE: ", 100 - mape_non_zero)
```

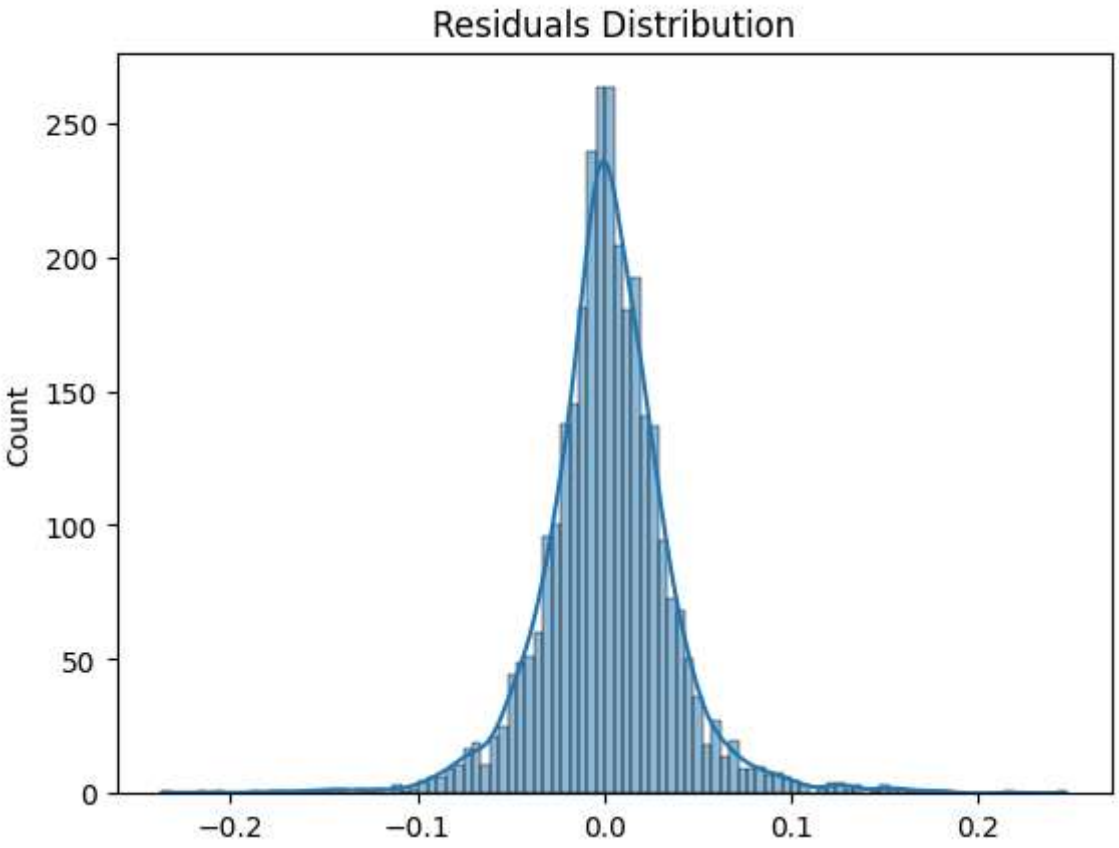
```
MSE: 0.058623916271308536
MAE: 0.19717705970069363
RMSE: 0.24212376230206845
MAPE:  96.32243381497732
```

7. Goodness of Fit Metrics

In [ ]:
```
# Assess the model on goodness of fit metrics
def goodness_of_fit(actual, forecast):
    residuals = actual - forecast
    sns.histplot(residuals, kde=True)
```

```
    plt.title("Residuals Distribution")
    plt.show()
```

In [ ]:
```
# train data
train_forecast = fitted.predict(
    start=train_data.index[0], end=train_data.index[-1], typ="levels"
)
goodness_of_fit(train_data, train_forecast)
```



8. Comparison with OLS Model

In [ ]:
```
# Compare with previously trained OLS model
def train_and_predict_regression(data):
    X = data[["Open", "High", "Low", "Volume"]]
    y = data["Close"]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, shuffle=False
    )

    X_train = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train).fit()

    X_test = sm.add_constant(X_test)
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    return model, y_pred_train, y_pred_test, y_train, y_test
```

In [ ]:
```
# Fit OLS model and make predictions
model, y_pred_train, y_pred_test, y_train, y_test = train_and_predict_regression(
    stock_data["TSLA"]
)

# Calculate RMSE for ARIMA and OLS forecasts
rmse_arima = math.sqrt(mean_squared_error(test_data_non_zero, fc_non_zero))
rmse_ols = math.sqrt(mean_squared_error(y_test, y_pred_test))
print("ARIMA RMSE: ", rmse_arima)
print("OLS RMSE: ", rmse_ols)
```

```
ARIMA RMSE:  0.24212376230206845
OLS RMSE:  3.4708442312601706
```

9. Theil's Coefficient

In [ ]:
```
# Calculate theil's U statistic
def theil_u_statistic(y_true, y_pred):
    n = len(y_true)
    num = np.sum((y_true - y_pred) ** 2)
    den = np.sum(y_true ** 2)
    return np.sqrt(num / den) / n

# Calculate Theil's U statistic for ARIMA and OLS forecasts
theil_u_arima = theil_u_statistic(test_data_non_zero, fc_non_zero)
theil_u_ols = theil_u_statistic(y_test, y_pred_test)
print("ARIMA Theil's U: ", theil_u_arima)
print("OLS Theil's U: ", theil_u_ols)
```

```
ARIMA Theil's U:  0.0001308482060916976
OLS Theil's U:  1.9908782366495322e-05
```

CONCLUSION	In this experiment we learned how to forecast using ARIMA model. We checked the stationarity of the dataset using Augmented Dickey-Fuller test and identified the value of 'd' which converts data to stationary data. We identified coefficients 'p' and 'q' using Auto-correlation Function (ACF) & Partial auto-correlation function (PACF) plots. We fitted an ARIMA model on 80% of the historic data (train) using the p,q and d parameters and used the recent 20% data as 'test'. We evaluated the fitted model on various statistical metrics for error on 'train' and 'test'. We assessed the model on metrics that calculate goodness of fit on 'train' and 'test'. We compared the performance of this model with our previously trained OLS model in Experiment 8. We computed Theil's coefficient of the 2 forecasts (OLS, ARIMA) for any one stock forecast.