TSA assignment

Name : Radhika Madavi

Roll no: 20

1. Performing Time series analysis on Google Stock Data

In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")
```

In [10]:

```
# Load the dataset
df = pd.read_csv("GOOGL.csv")
df.describe()
```

Out[10]:

	Open	High	Low	Close	Adj Close	Volume
count	2515.000000	2515.000000	2515.000000	2515.000000	2515.000000	2.515000e+03
mean	32.595077	32.864632	32.304183	32.591929	32.591929	6.773364e+07
std	16.418988	16.559502	16.279947	16.426766	16.426766	5.365061e+07
min	10.968719	11.068068	10.851602	10.912663	10.912663	1.041200e+07
25%	15.962837	16.068944	15.785286	15.892267	15.892267	3.102300e+07
50%	28.601000	28.798048	28.289499	28.536501	28.536501	4.928600e+07
75%	47.353001	47.629250	47.001250	47.354749	47.354749	8.902489e+07
max	68.199997	68.352501	67.650002	68.123497	68.123497	5.923990e+08

In [11]:

```
df.head()
```

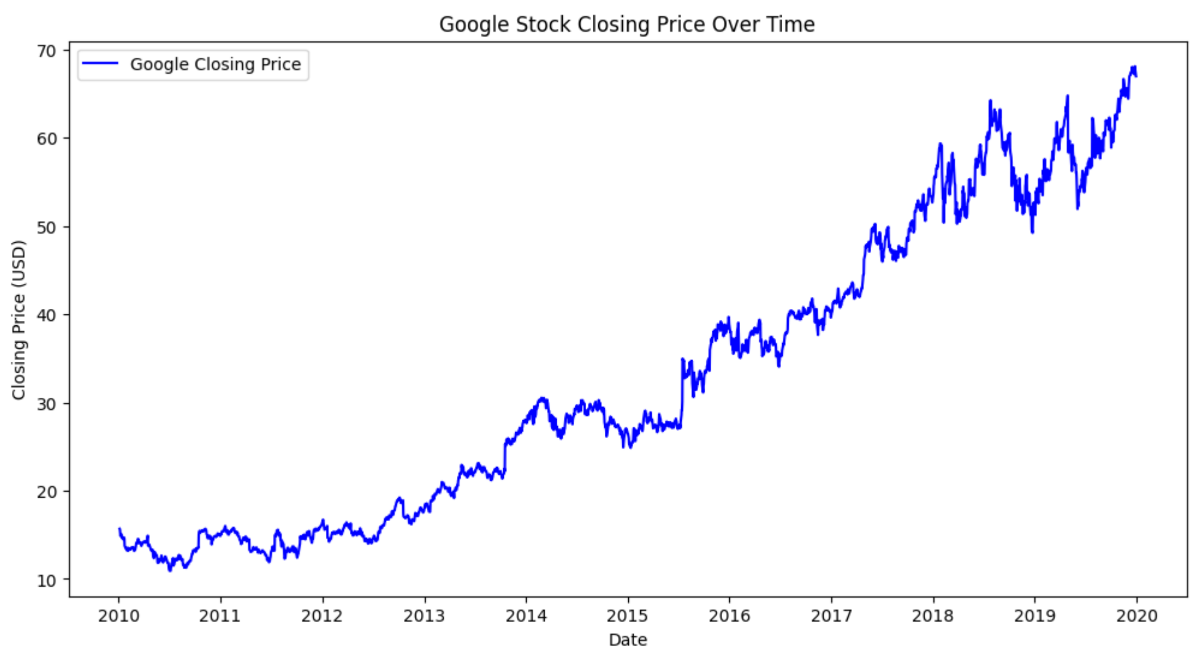
Out[11]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-01-04	15.689439	15.753504	15.621622	15.684434	15.684434	78169752
1	2010-01-05	15.695195	15.711712	15.554054	15.615365	15.615365	120067812
2	2010-01-06	15.662162	15.662162	15.174174	15.221722	15.221722	158988852
3	2010-01-07	15.250250	15.265265	14.831081	14.867367	14.867367	256315428
4	2010-01-08	14.814815	15.096346	14.742492	15.065566	15.065566	188783028

2. Plot the data

In [12]:

```
# Convert 'Date' column to datetime format and set as index
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
# Plot the time series data (Closing price)
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label="Google Closing Price", color='blue')
plt.xlabel("Date")
plt.ylabel("Closing Price (USD)")
plt.title("Google Stock Closing Price Over Time")
plt.legend()
plt.show()
```



In [13]:

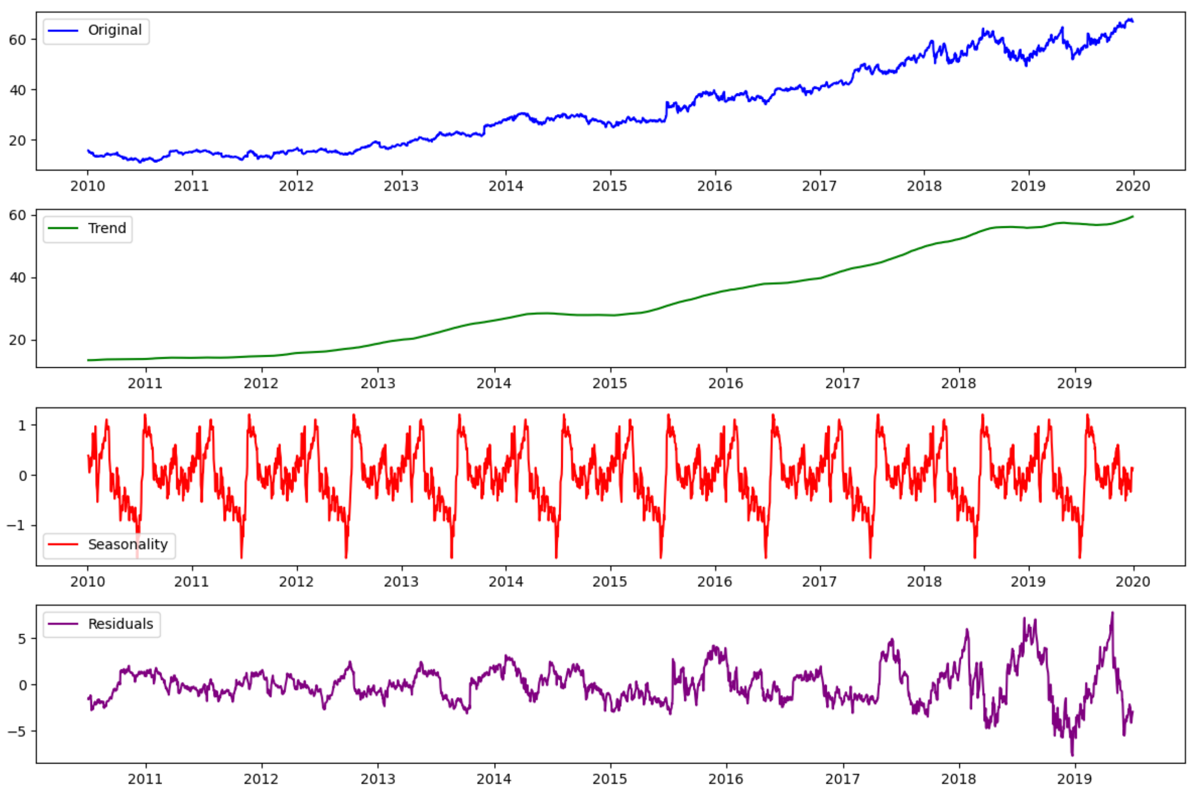
```
# Drop missing values and sort
print("Shape before dropping missing values:", df.shape)
df = df.dropna().sort_values(by="Date")
print("Shape after dropping missing values:", df.shape)
```

```
Shape before dropping missing values: (2515, 6)
Shape after dropping missing values: (2515, 6)
```

3. Extract components of time series analysis using Seasonal_decompose

In [13]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df['Close'], model='additive', period=252)  # As

# Plot the decomposed components
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(df['Close'], label='Original', color='blue')
plt.legend()

plt.subplot(412)
plt.plot(decomposition.trend, label='Trend', color='green')
plt.legend()

plt.subplot(413)
plt.plot(decomposition.seasonal, label='Seasonality', color='red')
plt.legend()

plt.subplot(414)
plt.plot(decomposition.resid, label='Residuals', color='purple')
plt.legend()

plt.tight_layout()
plt.show()
```



4.Checking the stationarity

In [23]:

```
# Perform the Augmented Dickey-Fuller test
adf_test = adfuller(df['Close'])
# Print whether the data is stationary or not
if adf_test[1] < 0.05:
    print("The data is stationary.")
else:
    print("The data is non-stationary.")
```

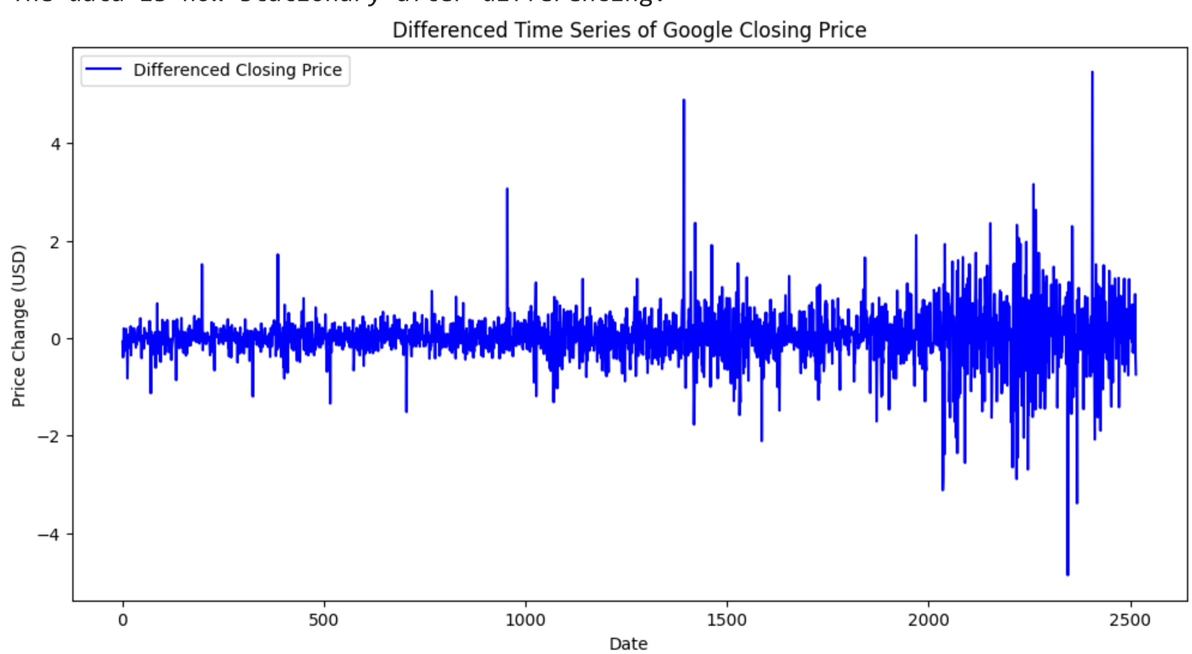
```
The data is non-stationary.
```

5.Converting Data to Stationary

In [4]:

```
# Apply first-order differencing to make the series stationary
df['Close_diff'] = df['Close'].diff()
# Perform ADF test again after differencing
adf_test_diff = adfuller(df['Close_diff'].dropna())
# Print whether the transformed data is stationary or not
if adf_test_diff[1] < 0.05:
    print("The data is now stationary after differencing.")
else:
    print("The data is still non-stationary.")
# Plot the differenced data
plt.figure(figsize=(12, 6))
plt.plot(df['Close_diff'], label="Differenced Closing Price", color='blue')
plt.xlabel("Date")
plt.ylabel("Price Change (USD)")
plt.title("Differenced Time Series of Google Closing Price")
plt.legend()
plt.show()
```

```
The data is now stationary after differencing.
```



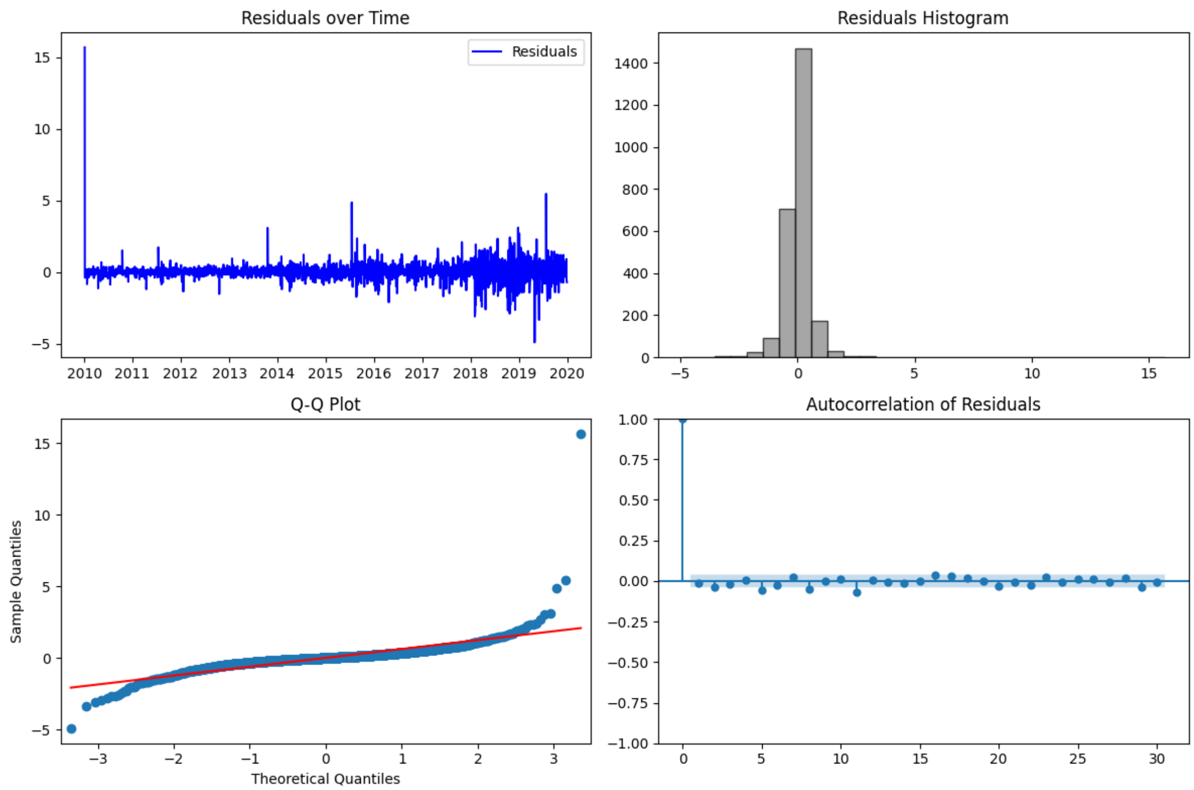
5. Implementing an ARIMA Model

In [24]:
```
model = ARIMA(df['Close'], order=(1, 1, 1)) # (p=1, d=1, q=1)
arima_result = model.fit()
print(arima_result.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 2515
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1997.460
Date:                Sat, 08 Mar 2025   AIC                           4000.920
Time:                        20:25:30   BIC                           4018.409
Sample:                             0   HQIC                          4007.267
                               - 2515
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7599      0.107     -7.113      0.000      -0.969      -0.551
ma.L1          0.7932      0.101      7.889      0.000       0.596       0.990
sigma2         0.2868      0.003    101.957      0.000       0.281       0.292
===================================================================================
Ljung-Box (L1) (Q):                   0.44   Jarque-Bera (JB):             22525.90
Prob(Q):                              0.50   Prob(JB):                         0.00
Heteroskedasticity (H):               9.94   Skew:                             0.11
Prob(H) (two-sided):                  0.00   Kurtosis:                        17.66
===================================================================================

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

6. Model validation

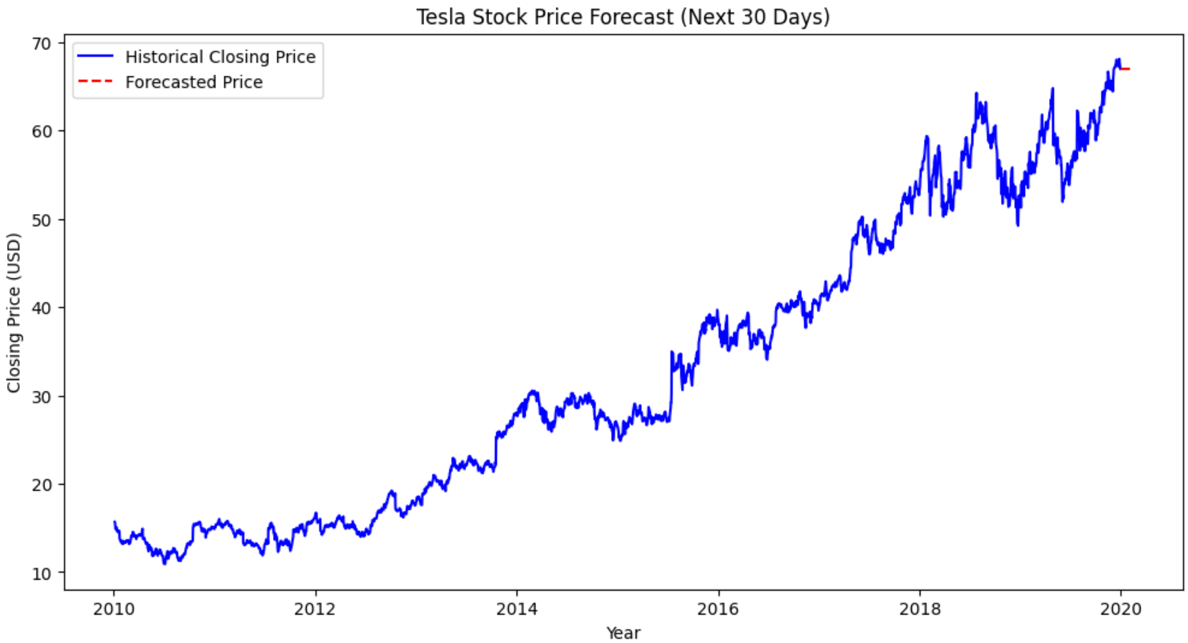
In [26]:
```
# Plot residual diagnostics
fig, ax = plt.subplots(2, 2, figsize=(12, 8))
# Residual plot
ax[0, 0].plot(arima_result.resid, label="Residuals", color='blue')
ax[0, 0].set_title("Residuals over Time")
ax[0, 0].legend()
# Histogram of residuals
ax[0, 1].hist(arima_result.resid, bins=30, color='gray', edgecolor='black',alpha=0.
ax[0, 1].set_title("Residuals Histogram")
# Q-Q plot
sm.qqplot(arima_result.resid, line="s", ax=ax[1, 0])
ax[1, 0].set_title("Q-Q Plot")
# ACF plot of residuals
sm.graphics.tsa.plot_acf(arima_result.resid, lags=30, ax=ax[1, 1])
ax[1, 1].set_title("Autocorrelation of Residuals")
plt.tight_layout()
plt.show()
```



7.Forecast future values for the next 30 days

In [27]:

```
forecast_steps = 30
forecast = arima_result.forecast(steps=forecast_steps)
# Plot the forecasted values
plt.figure(figsize=(12, 6))
plt.plot(df['Close'], label="Historical Closing Price", color='blue')
plt.plot(pd.date_range(df.index[-1], periods=forecast_steps+1, freq='B')[1:],
forecast, label="Forecasted Price", color='red', linestyle='dashed')
plt.xlabel("Year")
plt.ylabel("Closing Price (USD)")
plt.title("Google Stock Price Forecast (Next 30 Days)")
plt.legend()
plt.show()
# Display forecasted values
print(forecast)
```




```
2515    66.994541
2516    66.987668
2517    66.992891
2518    66.988922
2519    66.991938
2520    66.989646
2521    66.991388
2522    66.990064
2523    66.991070
2524    66.990306
2525    66.990886
2526    66.990445
2527    66.990780
2528    66.990526
2529    66.990719
2530    66.990572
2531    66.990684
2532    66.990599
2533    66.990664
2534    66.990615
2535    66.990652
2536    66.990624
2537    66.990645
2538    66.990629
2539    66.990641
2540    66.990632
2541    66.990639
2542    66.990633
2543    66.990638
2544    66.990634
Name: predicted_mean, dtype: float64
```

In [ ]: