Name - Siddharth Paraag Nilakhe

Email - snilakhe@stevens.edu

Fundamentals of Time Series Analysis: Understanding Seasonality, Stationarity, and Forecasting Models

Non-Seasonal Data

Non-seasonal data does not show periodic patterns. Fluctuations in such data are not tied to a specific season or time of year and can arise from a variety of non-cyclical factors.

Seasonal Data

Seasonal data exhibits patterns or behaviors that repeat over a specific period, such as monthly or quarterly. This cyclical nature often corresponds to external factors like weather or holidays.

ADF Test for Stationarity

The Augmented Dickey-Fuller (ADF) test is a statistical test used to determine the stationarity of a time series. It tests the null hypothesis that a unit root is present, where its absence (p-value < 0.05) indicates stationarity.

Differencing for Stationarity

Differencing is a method to make a time series stationary by subtracting the current observation from the previous one. This process, often repeated, removes trends and cycles, making the data more suitable for ARIMA modeling.

ARIMA Models

ARIMA (AutoRegressive Integrated Moving Average) models are used for forecasting non-seasonal time series data. It combines autoregressive (AR) terms, differencing for stationarity (I), and moving average (MA) terms, represented as ARIMA(p, d, q), where p, d, and q are non-negative integers.

GARCH Models

GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) models describe the variance of the current error term or innovation as a function of the past squared error terms. Primarily used in financial time series, it captures volatility clustering, where high volatility tends to follow high volatility.

SARIMA Models

SARIMA (Seasonal AutoRegressive Integrated Moving Average) extends the ARIMA model by incorporating seasonal elements. It's defined as SARIMA(p, d, q)(P, D, Q)s, where (p, d, q) are non-seasonal orders, (P, D, Q) are seasonal orders, and s is the seasonality period.

Ljung-Box Test

The Ljung-Box test assesses whether any of a group of autocorrelations of a time series are different from zero. It tests the null hypothesis that the data are independently distributed. Low p-values (typically < 0.05) indicate significant autocorrelation.

Please go through this article to understand a few more concepts that we have used in this project - https://medium.com/@siddharthnilakhe/aic-bic-hqic-jarque-bera-test-and-heteroskedasticity-test-4ef7e1fa19af

NON SEASONAL DATASET

Initial Steps

In [50]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
import scipy
import warnings
import arch
warnings.filterwarnings('ignore')
```

In [51]:
```
df = pd.read_csv('silver.csv')
df.head()
```

Out[51]:

	Date	Open	High	Low	Close	Volume	Currency
0	2000-01-04	5.420	5.420	5.32	5.375	27560	USD
1	2000-01-05	5.375	5.380	5.16	5.210	13515	USD
2	2000-01-06	5.205	5.215	5.15	5.167	4729	USD
3	2000-01-07	5.170	5.215	5.15	5.195	5375	USD
4	2000-01-10	5.190	5.230	5.17	5.190	4278	USD

In [52]:
```
df.describe()
```

Out[52]:

	Open	High	Low	Close	Volume
count	5708.000000	5708.000000	5708.000000	5708.000000	5708.000000
mean	15.913846	16.132912	15.671124	15.905674	42003.550981
std	8.503551	8.642820	8.336264	8.492578	32912.765504
min	4.020000	4.050000	4.015000	4.028000	0.000000
25%	7.648750	7.758750	7.565000	7.647500	16191.000000
50%	16.032500	16.227500	15.815000	16.048000	35335.500000
75%	19.865000	20.140000	19.620000	19.847250	59418.750000
max	48.490000	49.820000	47.550000	48.599000	355275.000000

In [53]:
```
df.isna().sum()
```

Out[53]:
```
Date        0
Open        0
High        0
Low         0
Close       0
Volume      0
Currency    0
dtype: int64
```

In [54]:
```
# Check unique values in categorical columns
unique_values = df['Currency'].unique()
print("Unique values in 'Currency' column:", unique_values)
```

```
Unique values in 'Currency' column: ['USD']
```

Checking for any unexpected values in the currenncy column. Non found

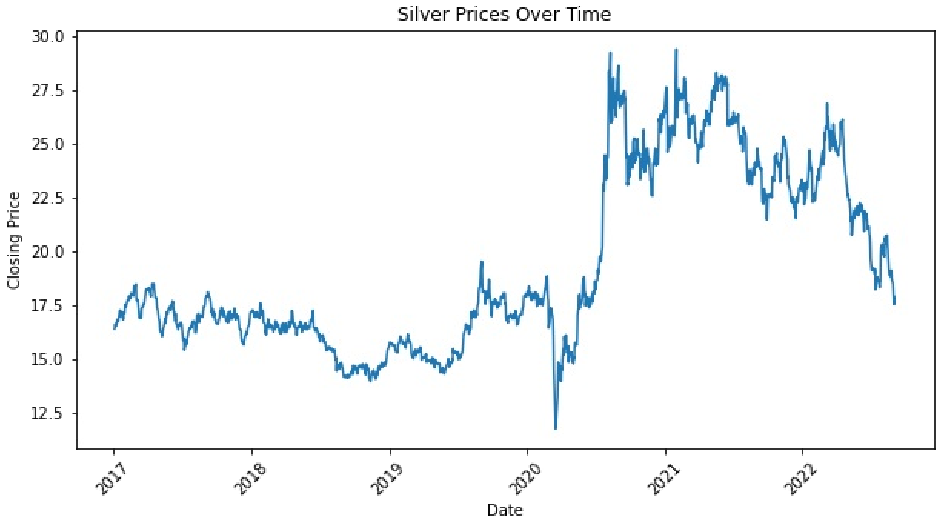
In [55]:
```
# Check the time range of the dataset
print("Min Date:", df['Date'].min())
print("Max Date:", df['Date'].max())
```

```
Min Date: 2000-01-04
Max Date: 2022-09-02
```

This dataset ranges from the year 2000 to to 2022 i.e 22 years of data. We will only be using 5 years of data for our analysis.

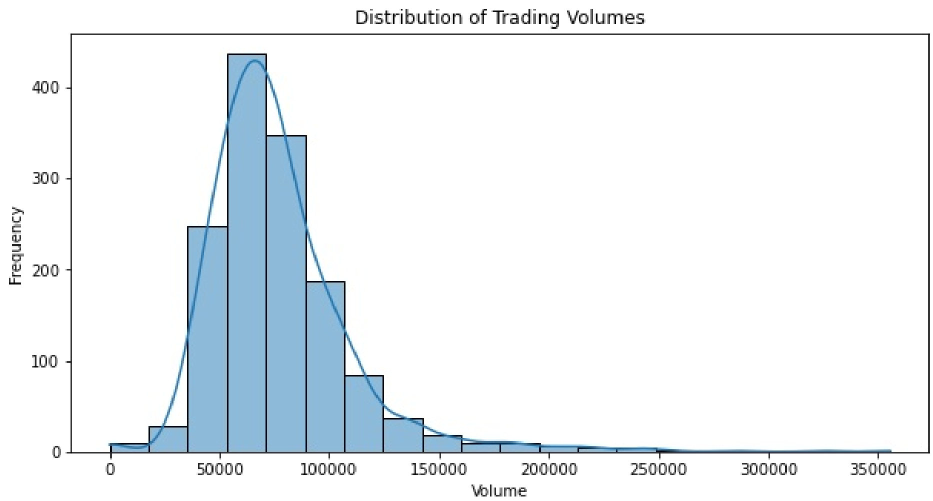
In [56]:
```python
df = df[df['Date'] >= '2017-01-01']
```

In [57]:
```python
import matplotlib.dates as mdates

# Converting 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Plotting the closing prices over time with proper date formatting
plt.figure(figsize=(10, 5))
plt.plot(df['Date'], df['Close'])
plt.title('Silver Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price')

# Using YearLocator to set ticks at the start of each year
plt.gca().xaxis.set_major_locator(mdates.YearLocator(base=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

plt.xticks(rotation=45)
plt.show()
```



Visualizing the price of Silver Prices over the period of 22 years

In [58]:
```python
# Distribution of trading volumes
plt.figure(figsize=(10, 5))
sns.histplot(df['Volume'], bins=20, kde=True)
plt.title('Distribution of Trading Volumes')
plt.xlabel('Volume')
plt.ylabel('Frequency')
plt.show()
```



The graph depicts the frequency distribution of trading volumes, where the data appears to be normally distributed with a slight right skew, indicating most trading volumes cluster around a central volume but with a tail extending towards higher volumes.

In [59]:
```python
df.dtypes
```

```
Out[59]: Date        datetime64[ns]
         Open               float64
         High               float64
         Low                float64
         Close              float64
         Volume               int64
         Currency            object
         dtype: object
```

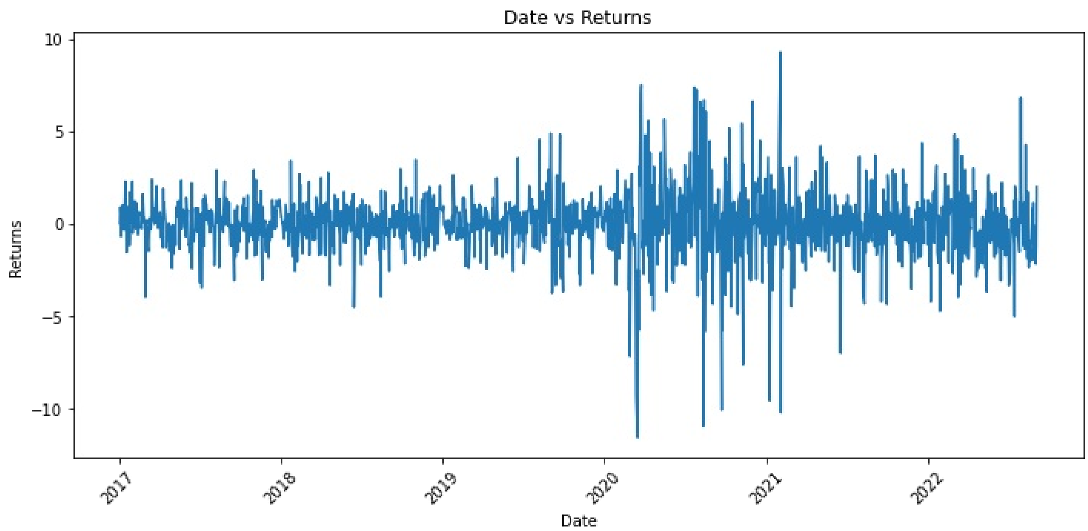
```
In [60]: df['Date'] = pd.to_datetime(df['Date'])
```

```
In [61]: df['Returns'] = 100 * df['Close'].pct_change()
         df.iloc[0, 7] = 0
         df.head()
```

Out[61]:

	Date	Open	High	Low	Close	Volume	Currency	Returns
4275	2017-01-03	15.970	16.550	15.935	16.409	81143	USD	0.000000
4276	2017-01-04	16.345	16.570	16.300	16.552	52451	USD	0.871473
4277	2017-01-05	16.495	16.760	16.455	16.637	67641	USD	0.513533
4278	2017-01-06	16.635	16.715	16.260	16.519	68136	USD	-0.709262
4279	2017-01-09	16.520	16.735	16.455	16.683	46502	USD	0.992796

```
In [62]: plt.figure(figsize=(10, 5))
         plt.plot(df['Date'].values, df['Returns'].values)
         plt.title('Date vs Returns')
         plt.xlabel('Date')
         plt.ylabel('Returns')
         plt.xticks(rotation=45)
         plt.tight_layout()
         plt.show()
```



This time series plot represents the volatility of returns from 2017 to 2022.

Stationary Test

```
In [63]: def test_stationarity(ts):
             result = adfuller(ts)

             print(f'ADF Statistic: {result[0]:.4f}')
             print(f'p-value: {result[1]:.4f}')
             print('Critical Values:')
             for key, value in result[4].items():
                 print(f'\t{key}: {value:.4f}')

             if result[1] > 0.05:
                 print("Failed to reject the null hypothesis (H0), the data has a unit root and is non-stationary.")
             else:
                 print("Reject the null hypothesis (H0), the data does not have a unit root and is stationary.")

         ts = df.set_index('Date')['Returns']
         test_stationarity(ts)
```

```
ADF Statistic: -12.5759
p-value: 0.0000
Critical Values:
	1%: -3.4350
	5%: -2.8636
	10%: -2.5679
Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
```

The strongly negative ADF statistic (−38.3077) and a negligible p-value (0.0000) confirm the rejection of the null hypothesis, indicating the time series data is stationary.

In [64]:
```
ts = pd.DataFrame(ts)
ts.head()
```

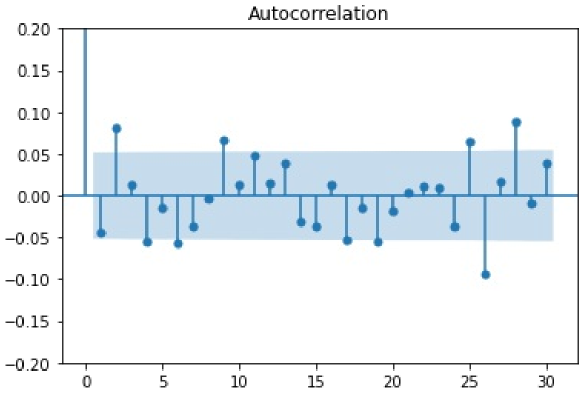
Out[64]:

	Returns
Date	
2017-01-03	0.000000
2017-01-04	0.871473
2017-01-05	0.513533
2017-01-06	-0.709262
2017-01-09	0.992796

ACF, PACF

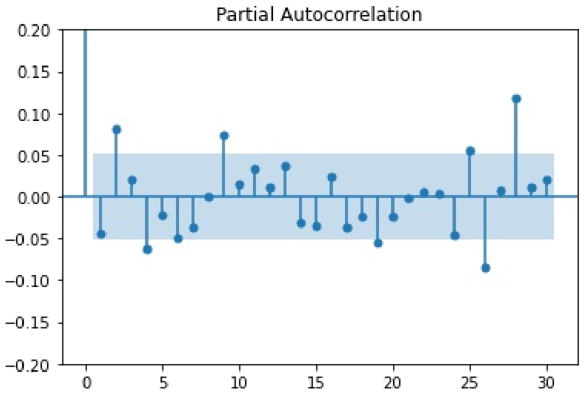
In [66]:
```
plot_acf(ts['Returns'], lags = 30);
plt.ylim(-0.2,0.2)
```

Out[66]:
```
(-0.2, 0.2)
```



In [67]:
```
plot_pacf(ts['Returns'], lags = 30);
plt.ylim(-0.2,0.2)
```

Out[67]:
```
(-0.2, 0.2)
```



ARMA Models

ARMA(1,1)

In [68]:
```
from statsmodels.tsa.arima.model import ARIMA

# ARMA(1, 1) model
arma_11_model = ARIMA(ts['Returns'], order=(1, 0, 1))
```

```python
arma_11_results = arma_11_model.fit()

# Print model summary
print(arma_11_results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1433
Model:                 ARIMA(1, 0, 1)   Log Likelihood               -2871.393
Date:                Tue, 12 Dec 2023   AIC                           5750.785
Time:                        14:50:03   BIC                           5771.855
Sample:                             0   HQIC                          5758.652
                               - 1433
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0223      0.046      0.480      0.631      -0.069       0.113
ar.L1         -0.4635      0.206     -2.252      0.024      -0.867      -0.060
ma.L1          0.4077      0.210      1.945      0.052      -0.003       0.819
sigma2         3.2209      0.059     54.688      0.000       3.105       3.336
===================================================================================
Ljung-Box (L1) (Q):                   0.36   Jarque-Bera (JB):              2837.40
Prob(Q):                              0.55   Prob(JB):                         0.00
Heteroskedasticity (H):               3.09   Skew:                            -0.42
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.84
===================================================================================

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

ARMA(1,2)

In [69]:
```python
# ARMA(1, 2) model
arma_12_model = ARIMA(ts['Returns'], order=(1, 0, 2))
arma_12_results = arma_12_model.fit()

# Print model summary
print(arma_12_results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1433
Model:                 ARIMA(1, 0, 2)   Log Likelihood               -2867.452
Date:                Tue, 12 Dec 2023   AIC                           5744.904
Time:                        14:50:06   BIC                           5771.241
Sample:                             0   HQIC                          5754.738
                               - 1433
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0224      0.051      0.439      0.660      -0.078       0.122
ar.L1          0.0669      0.195      0.343      0.731      -0.315       0.449
ma.L1         -0.1084      0.196     -0.552      0.581      -0.494       0.277
ma.L2          0.0956      0.020      4.879      0.000       0.057       0.134
sigma2         3.2031      0.060     53.598      0.000       3.086       3.320
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              2618.64
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               3.14   Skew:                            -0.42
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.57
===================================================================================

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

ARMA (2,1)

In [70]:
```python
# ARMA(2, 1) model
arma_21_model = ARIMA(ts['Returns'], order=(2, 0, 1))
arma_21_results = arma_21_model.fit()

# Print model summary
print(arma_21_results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1433
Model:                 ARIMA(2, 0, 1)   Log Likelihood               -2868.140
Date:                Tue, 12 Dec 2023   AIC                           5746.280
Time:                        14:50:07   BIC                           5772.618
Sample:                             0   HQIC                          5756.114
                               - 1433
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0224      0.051      0.442      0.659      -0.077       0.122
ar.L1          0.0553      0.212      0.261      0.794      -0.360       0.471
ar.L2          0.0849      0.020      4.247      0.000       0.046       0.124
ma.L1         -0.0960      0.215     -0.447      0.655      -0.517       0.325
sigma2         3.2062      0.060     53.827      0.000       3.089       3.323
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              2651.68
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               3.14   Skew:                            -0.43
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.61
===================================================================================

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

ARMA(2,2)

In [71]:

```python
# ARMA(2, 2) model
order = (2,0, 2)
model_arma_2_2 = ARIMA(ts, order=order)
results_arma_2_2 = model_arma_2_2.fit()

# Print results
print(results_arma_2_2.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1433
Model:                 ARIMA(2, 0, 2)   Log Likelihood               -2866.277
Date:                Tue, 12 Dec 2023   AIC                           5744.553
Time:                        14:50:09   BIC                           5776.158
Sample:                             0   HQIC                          5756.354
                               - 1433
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0224      0.048      0.463      0.643      -0.072       0.117
ar.L1         -0.4774      0.143     -3.341      0.001      -0.757      -0.197
ar.L2         -0.6043      0.126     -4.787      0.000      -0.852      -0.357
ma.L1          0.4308      0.141      3.055      0.002       0.154       0.707
ma.L2          0.6574      0.118      5.565      0.000       0.426       0.889
sigma2         3.1978      0.059     53.830      0.000       3.081       3.314
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):              2704.09
Prob(Q):                              0.80   Prob(JB):                         0.00
Heteroskedasticity (H):               3.11   Skew:                            -0.43
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.67
===================================================================================

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

Lets put various p and q values in a loop to check the best aic score that we get

In [72]:

```python
# Initialize an empty list to store results
results = []

# Loop for every p and q value between 1 and 4 (inclusive)
for p in range(1, 5):
    for q in range(1, 5):
        try:
            # Fit the ARIMA model with the current p and q values
            model = ARIMA(ts['Returns'], order=(p, 0, q))
            model_fit = model.fit()

            # Store p, q, and AIC values
            results.append((p, q, model_fit.aic))
        except:
            # In case the model does not converge or other errors occur
            results.append((p, q, float('inf')))

# Print results
for p, q, aic in results:
```

```
    print(f"ARIMA({p}, 0, {q}): AIC = {aic}")
```

```
ARIMA(1, 0, 1): AIC = 5750.785033445905
ARIMA(1, 0, 2): AIC = 5744.903731872094
ARIMA(1, 0, 3): AIC = 5745.210206892842
ARIMA(1, 0, 4): AIC = 5743.8710885916635
ARIMA(2, 0, 1): AIC = 5746.279923906656
ARIMA(2, 0, 2): AIC = 5744.553339614351
ARIMA(2, 0, 3): AIC = 5736.890856564958
ARIMA(2, 0, 4): AIC = 5744.801790187042
ARIMA(3, 0, 1): AIC = 5744.529598425695
ARIMA(3, 0, 2): AIC = 5746.124877175601
ARIMA(3, 0, 3): AIC = 5738.013630318176
ARIMA(3, 0, 4): AIC = 5739.316644629436
ARIMA(4, 0, 1): AIC = 5741.838578864301
ARIMA(4, 0, 2): AIC = 5737.976960745561
ARIMA(4, 0, 3): AIC = 5738.217630065887
ARIMA(4, 0, 4): AIC = 5739.102752524929
```

In [73]:

```
# Find the combination with the lowest AIC
lowest_aic = min(results, key=lambda x: x[2])

# Print the result
lowest_aic_p, lowest_aic_q, lowest_aic_value = lowest_aic
print(f"Lowest AIC is {lowest_aic_value} for ARMA({lowest_aic_p}, {lowest_aic_q})")
```

```
Lowest AIC is 5736.890856564958 for ARMA(2, 3)
```

In [74]:

```
# ARMA model with lowest aic
order = (lowest_aic_p,0, lowest_aic_q)
model_arma_lowest_aic = ARIMA(ts, order=order)
results_arma_lowest_aic = model_arma_lowest_aic.fit()

# Print results
print(results_arma_lowest_aic.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1433
Model:                 ARIMA(2, 0, 3)   Log Likelihood               -2861.445
Date:                Tue, 12 Dec 2023   AIC                           5736.891
Time:                        14:54:32   BIC                           5773.764
Sample:                             0   HQIC                          5750.659
                               - 1433                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0227      0.044      0.511      0.609      -0.064       0.110
ar.L1          1.4296      0.071     20.262      0.000       1.291       1.568
ar.L2         -0.7769      0.068    -11.474      0.000      -0.910      -0.644
ma.L1         -1.4752      0.073    -20.314      0.000      -1.618      -1.333
ma.L2          0.9085      0.067     13.625      0.000       0.778       1.039
ma.L3         -0.1126      0.018     -6.397      0.000      -0.147      -0.078
sigma2         3.1762      0.060     52.940      0.000       3.059       3.294
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):              2495.25
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               3.10   Skew:                            -0.40
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.41
===================================================================================

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

In [76]:

```
sm.stats.acorr_ljungbox(results_arma_lowest_aic.resid)
```

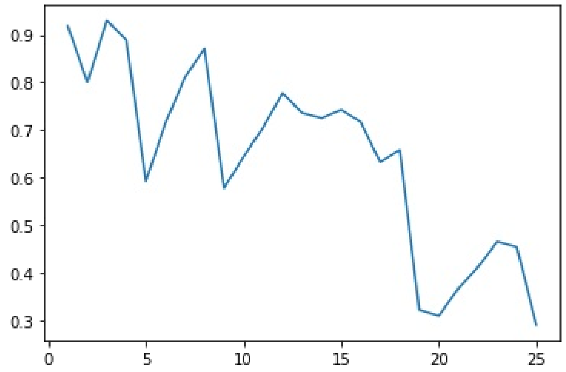
Out[76]:

	lb_stat	lb_pvalue
1	0.010633	0.917870
2	0.445620	0.800267
3	0.449037	0.929936
4	1.131379	0.889262
5	3.705450	0.592553
6	3.719047	0.714634
7	3.726876	0.810643
8	3.838968	0.871352
9	7.564335	0.578575
10	7.845517	0.643924

In [106…

```
sm.stats.acorr_ljungbox(results_arma_lowest_aic.resid, lags = 25)['lb_pvalue'].plot()
```

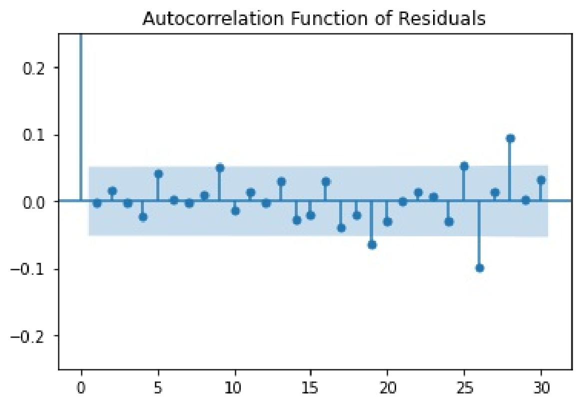
Out[106]: <AxesSubplot:>

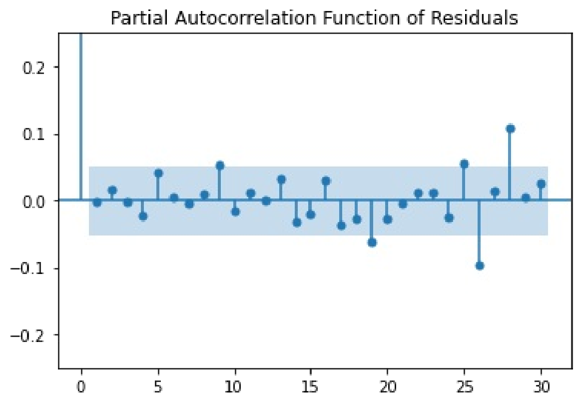


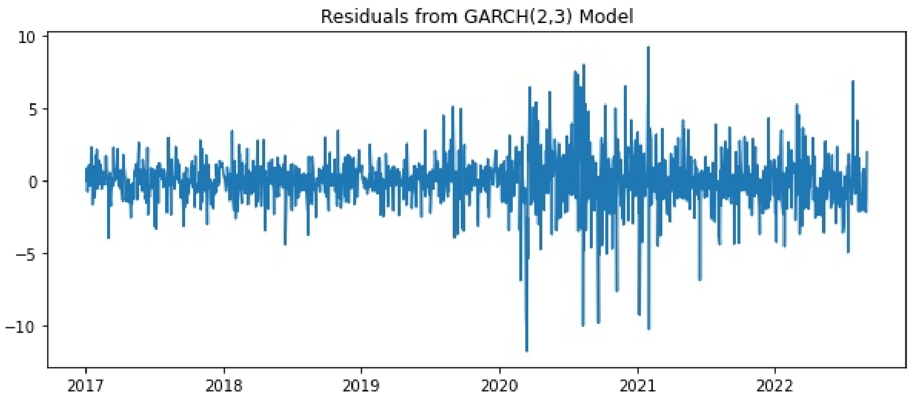
Ljung-box test results p-value for first 20 lags appear to be greater than 0.05. This shows that there is no correlation that can be seen amongst the residuals. This can also be confirmed by checking the ACF and PACF plots of the residuals.

In [107…

```
ljung_box_result, p_value = sm.stats.acorr_ljungbox(results_arma_lowest_aic.resid, lags=[25], return_df=False)

plot_acf(results_arma_lowest_aic.resid, lags=30)
plt.title('Autocorrelation Function of Residuals')
plt.ylim(-0.25,0.25)
plt.show()

plot_pacf(results_arma_lowest_aic.resid, lags=30)
plt.title('Partial Autocorrelation Function of Residuals')
plt.ylim(-0.25,0.25)
plt.show()

plt.figure(figsize=(10,4))
plt.plot(results_arma_lowest_aic.resid)
plt.title('Residuals from GARCH(2,3) Model')
plt.show()

print('Ljung-Box test statistic:', ljung_box_result)
print('Ljung-Box test p-value:', p_value)
```








```
Ljung-Box test statistic: [28.36792954]
Ljung-Box test p-value: [0.29119595]
```

After studying the residuals pacf and acf, we can conlclude that ARIMA model did a descent job in capturing the volatility.We will be implementing the GARCH model now. As ARMA is a linear model and can fail to capture the entire volatility in silver prices, we expect GARCH to perform better.

GARCH Models

In [108…

```
aic_dict = {}
```

GARCH (1,1)

In [109…

```
p=1
q=1
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 29111551823.370583
Iteration:      2,   Func. Count:     14,   Neg. LLF: 74678868882.759
Iteration:      3,   Func. Count:     22,   Neg. LLF: 3538.145771979087
Iteration:      4,   Func. Count:     30,   Neg. LLF: 2688.253729921461
Iteration:      5,   Func. Count:     36,   Neg. LLF: 2686.8730259685794
Iteration:      6,   Func. Count:     42,   Neg. LLF: 2678.470671048053
Iteration:      7,   Func. Count:     48,   Neg. LLF: 2701.2036827314855
Iteration:      8,   Func. Count:     54,   Neg. LLF: 2677.4481970070947
Iteration:      9,   Func. Count:     60,   Neg. LLF: 2677.0304975664762
Iteration:     10,   Func. Count:     65,   Neg. LLF: 2677.029268792728
Iteration:     11,   Func. Count:     70,   Neg. LLF: 2677.0292643160765
Iteration:     12,   Func. Count:     74,   Neg. LLF: 2677.0292643074845
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2677.0292643160765
            Iterations: 12
            Function evaluations: 74
            Gradient evaluations: 12
```

In [110…

```
results.summary()
```

Out[110]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2677.03
Distribution:	Normal	**AIC:**	5362.06
Method:	Maximum Likelihood	**BIC:**	5383.13
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:21	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-5.3110e-03	3.777e-02	-0.141	0.888	[-7.934e-02,6.872e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0171	1.722e-02	0.994	0.320	[-1.663e-02,5.087e-02]
alpha[1]	0.0367	1.877e-02	1.958	5.027e-02	[-4.307e-05,7.354e-02]
beta[1]	0.9583	2.335e-02	41.049	0.000	[0.913, 1.004]

Covariance estimator: robust

In [111…

```
aic_dict[(p, q)] = results.aic
```

GARCH (2,1)

In [112…

```
p=2
q=1
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 120594162243.16264
Iteration:      2,   Func. Count:     16,   Neg. LLF: 78214.38755645664
Iteration:      3,   Func. Count:     24,   Neg. LLF: 2688.7830618937087
Iteration:      4,   Func. Count:     31,   Neg. LLF: 4705.3325037237255
Iteration:      5,   Func. Count:     41,   Neg. LLF: 69706.11840722222
Iteration:      6,   Func. Count:     48,   Neg. LLF: 2678.3995236660176
Iteration:      7,   Func. Count:     54,   Neg. LLF: 2907.484367851497
Iteration:      8,   Func. Count:     62,   Neg. LLF: 2825.611528199318
Iteration:      9,   Func. Count:     70,   Neg. LLF: 2677.2483334776166
Iteration:     10,   Func. Count:     76,   Neg. LLF: 2677.0414662563844
Iteration:     11,   Func. Count:     82,   Neg. LLF: 2677.029841003957
Iteration:     12,   Func. Count:     88,   Neg. LLF: 2677.029271192978
Iteration:     13,   Func. Count:     94,   Neg. LLF: 2677.0292683988882
Iteration:     14,   Func. Count:    100,   Neg. LLF: 2677.0292670916215
Iteration:     15,   Func. Count:    106,   Neg. LLF: 2677.029263932179
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2677.029263932179
            Iterations: 15
            Function evaluations: 106
            Gradient evaluations: 15
```

In [113…

```
results.summary()
```

Out[113]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2677.03
Distribution:	Normal	**AIC:**	5364.06
Method:	Maximum Likelihood	**BIC:**	5390.40
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:22	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-5.3104e-03	3.776e-02	-0.141	0.888	[-7.932e-02,6.870e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0171	2.277e-02	0.752	0.452	[-2.750e-02,6.174e-02]
alpha[1]	0.0367	2.734e-02	1.344	0.179	[-1.683e-02,9.032e-02]
alpha[2]	0.0000	4.938e-02	0.000	1.000	[-9.679e-02,9.679e-02]
beta[1]	0.9583	3.813e-02	25.135	2.051e-139	[0.884, 1.033]

Covariance estimator: robust

In [114…

```
aic_dict[(p, q)] = results.aic
```

GARCH(1,2)

In [115…

```
p=1
q=2
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 7905.087211928217
Iteration:      2,   Func. Count:     17,   Neg. LLF: 163037200005.47656
Iteration:      3,   Func. Count:     26,   Neg. LLF: 21782504.954914853
Iteration:      4,   Func. Count:     33,   Neg. LLF: 32192164.57631154
Iteration:      5,   Func. Count:     40,   Neg. LLF: 2714.1897015785016
Iteration:      6,   Func. Count:     47,   Neg. LLF: 2700.0791287206257
Iteration:      7,   Func. Count:     54,   Neg. LLF: 2693.5435757810014
Iteration:      8,   Func. Count:     61,   Neg. LLF: 2677.6006813296567
Iteration:      9,   Func. Count:     68,   Neg. LLF: 3622.824072469873
Iteration:     10,   Func. Count:     76,   Neg. LLF: 2677.61380136129
Iteration:     11,   Func. Count:     83,   Neg. LLF: 2685.109600558062
Iteration:     12,   Func. Count:     90,   Neg. LLF: 2674.166780380154
Iteration:     13,   Func. Count:     97,   Neg. LLF: 2674.09065454805
Iteration:     14,   Func. Count:    104,   Neg. LLF: 2674.0900600446494
Iteration:     15,   Func. Count:    110,   Neg. LLF: 2674.0900577713837
Iteration:     16,   Func. Count:    115,   Neg. LLF: 2674.090057771346
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2674.0900577713837
            Iterations: 16
            Function evaluations: 115
            Gradient evaluations: 16
```

In [116…

```
results.summary()
```

Out[116]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2674.09
Distribution:	Normal	**AIC:**	5358.18
Method:	Maximum Likelihood	**BIC:**	5384.52
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:23	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-4.9122e-03	3.768e-02	-0.130	0.896	[-7.876e-02,6.894e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0258	2.302e-02	1.121	0.262	[-1.930e-02,7.094e-02]
alpha[1]	0.0550	2.302e-02	2.389	1.688e-02	[9.880e-03, 0.100]
beta[1]	0.2821	0.127	2.228	2.588e-02	[3.393e-02, 0.530]
beta[2]	0.6550	0.135	4.856	1.198e-06	[0.391, 0.919]

Covariance estimator: robust

In [117…
```
aic_dict[(p, q)] = results.aic
```

GARCH (2,2)

In [118…
```
p=2
q=2
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      8,   Neg. LLF: 7891.12580948271
Iteration:      2,   Func. Count:     19,   Neg. LLF: 150715925499.52448
Iteration:      3,   Func. Count:     29,   Neg. LLF: 7280202.293843483
Iteration:      4,   Func. Count:     37,   Neg. LLF: 2691.9474700245596
Iteration:      5,   Func. Count:     45,   Neg. LLF: 289238253.07334113
Iteration:      6,   Func. Count:     53,   Neg. LLF: 2680.744806879434
Iteration:      7,   Func. Count:     61,   Neg. LLF: 2677.674951661186
Iteration:      8,   Func. Count:     69,   Neg. LLF: 2693.1246733510006
Iteration:      9,   Func. Count:     77,   Neg. LLF: 2685.7703512126045
Iteration:     10,   Func. Count:     85,   Neg. LLF: 2674.1105098235857
Iteration:     11,   Func. Count:     92,   Neg. LLF: 2674.094373046738
Iteration:     12,   Func. Count:     99,   Neg. LLF: 2674.0902509866264
Iteration:     13,   Func. Count:    106,   Neg. LLF: 2674.090388434922
Iteration:     14,   Func. Count:    114,   Neg. LLF: 2674.090057952222
Iteration:     15,   Func. Count:    120,   Neg. LLF: 2674.0900579516547
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2674.090057952222
            Iterations: 15
            Function evaluations: 120
            Gradient evaluations: 15
```

In [119…
```
results.summary()
```

Out[119]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2674.09
Distribution:	Normal	**AIC:**	5360.18
Method:	Maximum Likelihood	**BIC:**	5391.79
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:24	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-4.9125e-03	3.766e-02	-0.130	0.896	[-7.873e-02,6.890e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0258	3.137e-02	0.823	0.411	[-3.566e-02,8.729e-02]
alpha[1]	0.0550	2.905e-02	1.893	5.833e-02	[-1.939e-03, 0.112]
alpha[2]	6.3574e-10	5.620e-02	1.131e-08	1.000	[-0.110, 0.110]
beta[1]	0.2821	0.320	0.882	0.378	[-0.345, 0.909]
beta[2]	0.6550	0.280	2.341	1.923e-02	[0.107, 1.203]

Covariance estimator: robust

In [120…

```
aic_dict[(p, q)] = results.aic
```

GARCH (3,2)

In [121…

```
p=3
q=2
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      9,   Neg. LLF: 7892.912852424675
Iteration:      2,   Func. Count:     21,   Neg. LLF: 6470.994101608909
Iteration:      3,   Func. Count:     32,   Neg. LLF: 2693.6598273451605
Iteration:      4,   Func. Count:     41,   Neg. LLF: 2755.142467760965
Iteration:      5,   Func. Count:     51,   Neg. LLF: 2706.20516428135
Iteration:      6,   Func. Count:     60,   Neg. LLF: 2677.4746455947907
Iteration:      7,   Func. Count:     68,   Neg. LLF: 2690.685373559765
Iteration:      8,   Func. Count:     77,   Neg. LLF: 2726.923844916855
Iteration:      9,   Func. Count:     87,   Neg. LLF: 2684.73536049788
Iteration:     10,   Func. Count:     96,   Neg. LLF: 2674.41421381144
Iteration:     11,   Func. Count:    104,   Neg. LLF: 2674.102236365575
Iteration:     12,   Func. Count:    112,   Neg. LLF: 2674.0910772535244
Iteration:     13,   Func. Count:    120,   Neg. LLF: 2674.09009014611
Iteration:     14,   Func. Count:    128,   Neg. LLF: 2674.0900631534605
Iteration:     15,   Func. Count:    136,   Neg. LLF: 2674.0900588466075
Iteration:     16,   Func. Count:    144,   Neg. LLF: 2674.090057913075
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2674.090057913075
            Iterations: 16
            Function evaluations: 144
            Gradient evaluations: 16
```

In [122…

```
results.summary()
```

Out[122]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2674.09
Distribution:	Normal	**AIC:**	5362.18
Method:	Maximum Likelihood	**BIC:**	5399.05
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:25	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-4.9119e-03	3.770e-02	-0.130	0.896	[-7.881e-02,6.898e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0258	3.227e-02	0.800	0.424	[-3.744e-02,8.907e-02]
alpha[1]	0.0550	3.312e-02	1.660	9.688e-02	[-9.930e-03, 0.120]
alpha[2]	2.7802e-10	4.840e-02	5.744e-09	1.000	[-9.487e-02,9.487e-02]
alpha[3]	9.3033e-10	4.797e-02	1.939e-08	1.000	[-9.403e-02,9.403e-02]
beta[1]	0.2821	0.251	1.125	0.261	[-0.209, 0.774]
beta[2]	0.6550	0.222	2.946	3.222e-03	[0.219, 1.091]

Covariance estimator: robust

In [123…

```
aic_dict[(p, q)] = results.aic
```

GARCH (2,3)

In [124…

```
p=2
q=3
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:      9,   Neg. LLF: 335171.41447253595
Iteration:      2,   Func. Count:     20,   Neg. LLF: 3824.8863706998172
Iteration:      3,   Func. Count:     29,   Neg. LLF: 2735.4478285228743
Iteration:      4,   Func. Count:     38,   Neg. LLF: 2744.0230819618714
Iteration:      5,   Func. Count:     48,   Neg. LLF: 2695.4144647925605
Iteration:      6,   Func. Count:     57,   Neg. LLF: 2678.516203759991
Iteration:      7,   Func. Count:     66,   Neg. LLF: 2679.5764120046433
Iteration:      8,   Func. Count:     75,   Neg. LLF: 2673.6000791873457
Iteration:      9,   Func. Count:     83,   Neg. LLF: 2673.3002762219326
Iteration:     10,   Func. Count:     91,   Neg. LLF: 2760.9942098739793
Iteration:     11,   Func. Count:    101,   Neg. LLF: 2685.675355341701
Iteration:     12,   Func. Count:    110,   Neg. LLF: 2677.930846777089
Iteration:     13,   Func. Count:    119,   Neg. LLF: 2673.0578315630755
Iteration:     14,   Func. Count:    127,   Neg. LLF: 2673.041213347201
Iteration:     15,   Func. Count:    135,   Neg. LLF: 2673.0410128442527
Iteration:     16,   Func. Count:    144,   Neg. LLF: 2673.0399589079066
Iteration:     17,   Func. Count:    152,   Neg. LLF: 2673.0399579462473
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2673.0399579462473
            Iterations: 17
            Function evaluations: 152
            Gradient evaluations: 17
```

In [125…

```
results.summary()
```

Out[125]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2673.04
Distribution:	Normal	**AIC:**	5360.08
Method:	Maximum Likelihood	**BIC:**	5396.95
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:29	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-4.6557e-03	3.775e-02	-0.123	0.902	[-7.864e-02,6.933e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0321	2.866e-02	1.121	0.262	[-2.403e-02,8.831e-02]
alpha[1]	0.0674	2.674e-02	2.521	1.172e-02	[1.499e-02, 0.120]
alpha[2]	7.2387e-16	3.261e-02	2.220e-14	1.000	[-6.392e-02,6.392e-02]
beta[1]	0.0415	0.482	8.609e-02	0.931	[-0.903, 0.986]
beta[2]	0.5950	0.172	3.468	5.244e-04	[0.259, 0.931]
beta[3]	0.2860	0.357	0.802	0.423	[-0.413, 0.985]

Covariance estimator: robust

In [126…

```
aic_dict[(p, q)] = results.aic
```

GARCH (3,3)

In [127…

```
p=3
q=3
model = arch.arch_model(ts['Returns'], p = p, q = q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:     10,   Neg. LLF: 6464.538928135811
Iteration:      2,   Func. Count:     22,   Neg. LLF: 2075856.0645453567
Iteration:      3,   Func. Count:     33,   Neg. LLF: 2892.828453121702
Iteration:      4,   Func. Count:     45,   Neg. LLF: 2707.5645691677123
Iteration:      5,   Func. Count:     55,   Neg. LLF: 2718.7223015081618
Iteration:      6,   Func. Count:     65,   Neg. LLF: 2706.592258413841
Iteration:      7,   Func. Count:     75,   Neg. LLF: 2722.0993829794174
Iteration:      8,   Func. Count:     85,   Neg. LLF: 2673.8231759716737
Iteration:      9,   Func. Count:     94,   Neg. LLF: 2673.2343319598394
Iteration:     10,   Func. Count:    103,   Neg. LLF: 2683.563910132843
Iteration:     11,   Func. Count:    114,   Neg. LLF: 2673.126184907592
Iteration:     12,   Func. Count:    123,   Neg. LLF: 2673.063830287952
Iteration:     13,   Func. Count:    132,   Neg. LLF: 2673.045330803035
Iteration:     14,   Func. Count:    141,   Neg. LLF: 2673.0429019457424
Iteration:     15,   Func. Count:    150,   Neg. LLF: 2673.040389879418
Iteration:     16,   Func. Count:    159,   Neg. LLF: 2673.040005331934
Iteration:     17,   Func. Count:    168,   Neg. LLF: 2673.0399613220716
Iteration:     18,   Func. Count:    177,   Neg. LLF: 2673.039958403644
Iteration:     19,   Func. Count:    185,   Neg. LLF: 2673.039958402591
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2673.039958403644
            Iterations: 19
            Function evaluations: 185
            Gradient evaluations: 19
```

In [128…

```
results.summary()
```

Out[128]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2673.04
Distribution:	Normal	**AIC:**	5362.08
Method:	Maximum Likelihood	**BIC:**	5404.22
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:16:32	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-4.6537e-03	3.776e-02	-0.123	0.902	[-7.867e-02,6.936e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0321	2.875e-02	1.118	0.264	[-2.422e-02,8.849e-02]
alpha[1]	0.0674	2.514e-02	2.681	7.341e-03	[1.813e-02, 0.117]
alpha[2]	0.0000	5.473e-02	0.000	1.000	[-0.107, 0.107]
alpha[3]	2.2835e-11	2.673e-02	8.544e-10	1.000	[-5.238e-02,5.238e-02]
beta[1]	0.0414	0.806	5.138e-02	0.959	[-1.537, 1.620]
beta[2]	0.5952	0.415	1.435	0.151	[-0.218, 1.408]
beta[3]	0.2860	0.422	0.678	0.498	[-0.541, 1.113]

Covariance estimator: robust

In [129…

```
aic_dict[(p, q)] = results.aic
```

AIC Scores of GARCH Model

In [130…

```
aic_dict
```

Out[130]:

```
{(1, 1): 5362.058528632153,
 (2, 1): 5364.058527864358,
 (1, 2): 5358.180115542767,
 (2, 2): 5360.180115904444,
 (3, 2): 5362.18011582615,
 (2, 3): 5360.079915892495,
 (3, 3): 5362.079916807288}
```

In [131…

```
best_aic = min(aic_dict, key=aic_dict.get) #getting the lowest aic score
best_aic
```

Out[131]:

```
(1, 2)
```

Lets try to loop here too

In [133…

```
# Initialize an empty list to store GARCH model results
garch_results = []

# Loop for every p and q value between 1 and 4 (inclusive)
for p in range(1, 5):
    for q in range(1, 5):
        try:
            # Fit the GARCH model with the current p and q values
            model = arch.arch_model(ts['Returns'], p=p, q=q)
            model_fit = model.fit(disp='off')  # Disable printing of the fit results

            # Store p, q, and the model's AIC
            garch_results.append((p, q, model_fit.aic))
        except Exception as e:
            # In case the model does not converge or other errors occur
            garch_results.append((p, q, str(e)))

# Print results
for p, q, aic in garch_results:
    print(f"GARCH({p}, {q}): AIC = {aic}")
```

```
GARCH(1, 1): AIC = 5362.058528632153
GARCH(1, 2): AIC = 5358.180115542767
GARCH(1, 3): AIC = 5358.079916507573
GARCH(1, 4): AIC = 5355.7256956998735
GARCH(2, 1): AIC = 5364.058527864358
GARCH(2, 2): AIC = 5360.180115904444
GARCH(2, 3): AIC = 5360.079915892495
GARCH(2, 4): AIC = 5355.396095372535
GARCH(3, 1): AIC = 5366.058537288884
GARCH(3, 2): AIC = 5362.18011582615
GARCH(3, 3): AIC = 5362.079916807288
GARCH(3, 4): AIC = 5357.088444191628
GARCH(4, 1): AIC = 5368.058527972304
GARCH(4, 2): AIC = 5364.180115696041
GARCH(4, 3): AIC = 5364.079916337084
GARCH(4, 4): AIC = 5359.088443681341
```

In [134…

```
# Find the combination with the lowest AIC
lowest_aic = min(garch_results, key=lambda x: x[2])

# Print the result
lowest_aic_p, lowest_aic_q, lowest_aic_value = lowest_aic
print(f"Lowest AIC is {lowest_aic_value} for GARCH({lowest_aic_p}, {lowest_aic_q})")
```

```
Lowest AIC is 5355.396095372535 for GARCH(2, 4)
```

Residual Analysis for the lowest AIC score model

In [135…

```
model = arch.arch_model(ts['Returns'], p = lowest_aic_p, q = lowest_aic_q)
results = model.fit()
```

```
Iteration:      1,   Func. Count:     10,   Neg. LLF: 6559.313700918368
Iteration:      2,   Func. Count:     22,   Neg. LLF: 1893294.8899657007
Iteration:      3,   Func. Count:     34,   Neg. LLF: 2707.079827095005
Iteration:      4,   Func. Count:     44,   Neg. LLF: 2680.7572144265177
Iteration:      5,   Func. Count:     54,   Neg. LLF: 2677.858746901991
Iteration:      6,   Func. Count:     64,   Neg. LLF: 2701.061117512839
Iteration:      7,   Func. Count:     74,   Neg. LLF: 2828.4124537139105
Iteration:      8,   Func. Count:     84,   Neg. LLF: 2683.6256998880135
Iteration:      9,   Func. Count:     94,   Neg. LLF: 2670.935636726553
Iteration:     10,   Func. Count:    103,   Neg. LLF: 2679.268851982642
Iteration:     11,   Func. Count:    113,   Neg. LLF: 2679.3897209504307
Iteration:     12,   Func. Count:    123,   Neg. LLF: 2669.9161470539852
Iteration:     13,   Func. Count:    132,   Neg. LLF: 2670.7179626014636
Iteration:     14,   Func. Count:    142,   Neg. LLF: 2669.6997935165864
Iteration:     15,   Func. Count:    151,   Neg. LLF: 2669.6985639881977
Iteration:     16,   Func. Count:    160,   Neg. LLF: 2669.698195519432
Iteration:     17,   Func. Count:    169,   Neg. LLF: 2669.6980500933423
Iteration:     18,   Func. Count:    178,   Neg. LLF: 2669.6980477312604
Iteration:     19,   Func. Count:    187,   Neg. LLF: 36194.970782440316
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2669.6980476862673
            Iterations: 20
            Function evaluations: 193
            Gradient evaluations: 19
```

In [136…

```
results.summary()
```

Out[136]:

Constant Mean - GARCH Model Results

Dep. Variable:	Returns	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-2669.70
Distribution:	Normal	**AIC:**	5355.40
Method:	Maximum Likelihood	**BIC:**	5397.54
		No. Observations:	1433
Date:	Tue, Dec 12 2023	**Df Residuals:**	1432
Time:	15:19:54	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-3.7376e-03	3.721e-02	-0.100	0.920	[-7.666e-02,6.919e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0525	4.889e-02	1.075	0.282	[-4.328e-02, 0.148]
alpha[1]	0.0522	3.906e-02	1.337	0.181	[-2.435e-02, 0.129]
alpha[2]	0.0490	4.773e-02	1.026	0.305	[-4.459e-02, 0.143]
beta[1]	0.1670	0.839	0.199	0.842	[-1.477, 1.811]
beta[2]	1.4456e-17	0.967	1.495e-17	1.000	[-1.896, 1.896]
beta[3]	1.6311e-17	0.847	1.925e-17	1.000	[-1.661, 1.661]
beta[4]	0.7146	0.658	1.087	0.277	[-0.574, 2.004]

Covariance estimator: robust

In [137…

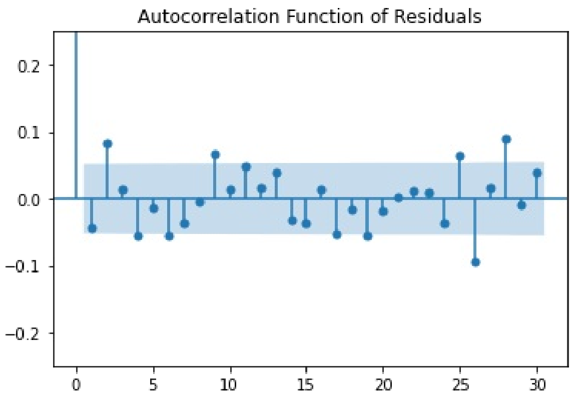
```
sm.stats.acorr_ljungbox(results.resid)
```

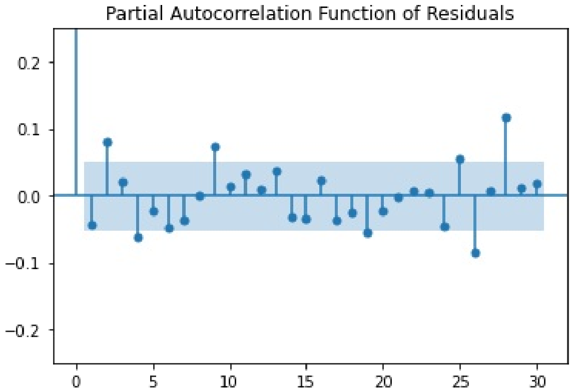
Out[137]:

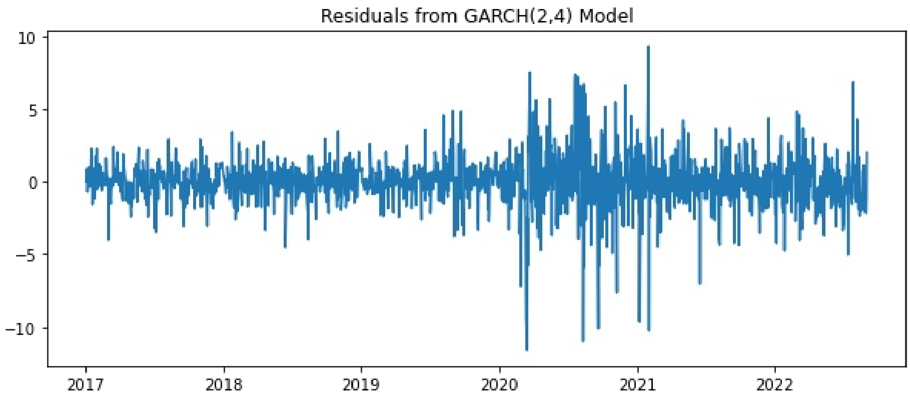
	lb_stat	lb_pvalue
1	2.717971	0.099224
2	12.344554	0.002086
3	12.601261	0.005583
4	17.063141	0.001879
5	17.359655	0.003866
6	21.875849	0.001275
7	23.766131	0.001252
8	23.782137	0.002493
9	30.066951	0.000427
10	30.305192	0.000763

In [138…

```
ljung_box_result, p_value = sm.stats.acorr_ljungbox(results.resid, lags=[30], return_df=False)

plot_acf(results.resid, lags=30)
plt.title('Autocorrelation Function of Residuals')
plt.ylim(-0.25,0.25)
plt.show()

plot_pacf(results.resid, lags=30)
plt.title('Partial Autocorrelation Function of Residuals')
plt.ylim(-0.25,0.25)
plt.show()

plt.figure(figsize=(10,4))
plt.plot(results.resid)
plt.title('Residuals from GARCH(2,4) Model')
plt.show()

print('Ljung-Box test statistic:', ljung_box_result)
print('Ljung-Box test p-value:', p_value)
```








```
Ljung-Box test statistic: [84.84549845]
Ljung-Box test p-value: [3.88274732e-07]
```

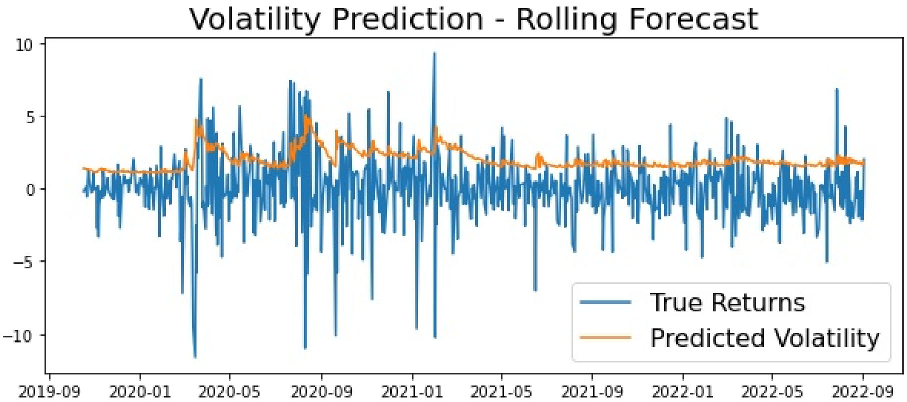
Since we got a better AIC score with a GARCH model when compared to the ARMA model we will be using our best model for forecasting purposes.

Forecasting

In [144…
```
rolling_predictions = []
test_size = 365*2
for i in range(test_size):
    train = ts[:-(test_size-i)]
    model = arch.arch_model(train['Returns'], p=2, q=4)
    model_fit = model.fit(disp='off')
    pred = model_fit.forecast(horizon=1)
    rolling_predictions.append(np.sqrt(pred.variance.values[-1,:][0]))
```

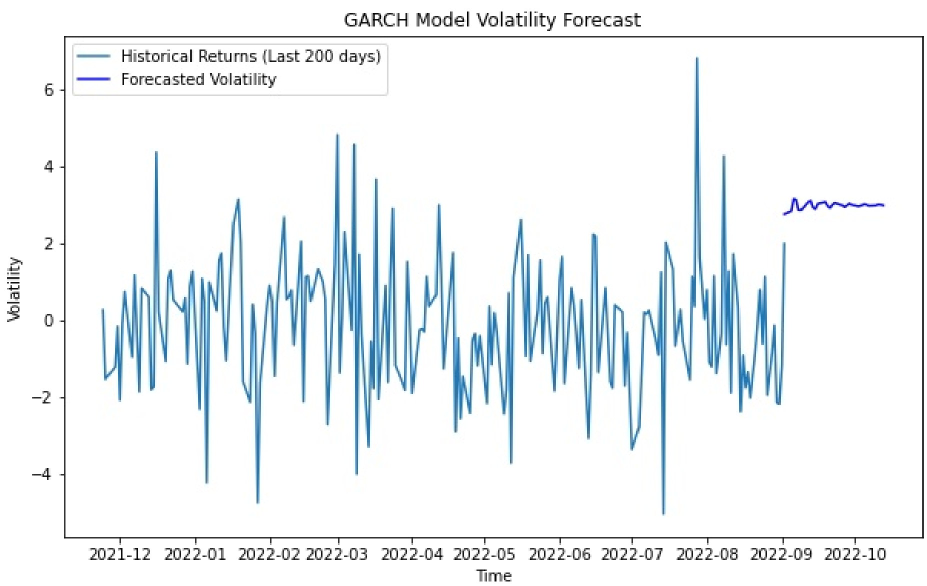
In [145…
```
rolling_predictions = pd.Series(rolling_predictions, index=ts.index[-365*2:])
plt.figure(figsize=(10,4))
true, = plt.plot(ts[-365*2:])
preds, = plt.plot(rolling_predictions)
plt.title('Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

Out[145]:
```
<matplotlib.legend.Legend at 0x19f02e8a8b0>
```



In [150…

```
garch_model = arch.arch_model(ts['Returns'], p=lowest_aic_p, q=lowest_aic_q)
garch_fit = garch_model.fit(disp='off')

# Forecast the next 30 days for volatility
garch_forecast = garch_fit.forecast(horizon=30)
volatility_forecast = garch_forecast.variance.iloc[-1]

# Preparing the plot
forecast_horizon = 30
last_200_days = ts['Returns'].iloc[-200:]  # Select the last 200 days
volatility_forecast = np.array(volatility_forecast).reshape(-1)
forecast_index = pd.date_range(start=ts.index[-1], periods=forecast_horizon, freq='B')  # Adjust the frequency 

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(last_200_days.index, last_200_days, label='Historical Returns (Last 200 days)')
plt.plot(forecast_index, volatility_forecast, color='blue', label='Forecasted Volatility')

plt.title('GARCH Model Volatility Forecast')
plt.xlabel('Time')
plt.ylabel('Volatility')
plt.legend()
plt.show()
```



Conclusion

In this comprehensive project, we successfully applied advanced time series analysis techniques to forecast silver price data, a challenging domain characterized by significant volatility. Our rigorous analysis led us to identify the GARCH model with parameters (2,4) as the optimal forecasting tool, evidenced by its AIC score of 5355.39. This score notably outperformed the ARMA model, which registered an AIC of 5736, underscoring the GARCH model's superior capability in handling the intricacies of volatile financial time series.

The project involved a series of crucial steps, starting with the verification of data stationarity, a foundational

The project involved a series of crucial steps, starting with the verification of data stationarity, a foundational aspect of time series analysis. This was followed by the application and comparative evaluation of various ARIMA and GARCH models. Our focus was not only on identifying the best model but also understanding the underlying dynamics of the silver market.

The choice of the GARCH model was pivotal, considering its proficiency in capturing and modeling the volatility clustering—a common characteristic of financial markets like silver trading. This feature provided us with a more nuanced and accurate forecasting ability, setting it apart from other models such as ARMA.

In conclusion, the project was successful in achieving its objectives of conducting a detailed time series analysis and providing reliable forecasts for silver prices. The insights gained from employing ARIMA and GARCH models have proven invaluable, demonstrating the importance of selecting appropriate models based on the specific characteristics of the data and the context of the analysis. This endeavor has contributed significantly to our understanding of financial market dynamics, particularly in the context of precious metals such as silver.

In [ ]:

SEASONAL DATASET

Initial Steps

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.statespace.sarimax import SARIMAX
import itertools
import warnings
warnings.filterwarnings('ignore')
import statsmodels.api as sm
```

In [2]:
```
df_temp = pd.read_csv("Mumbai.csv")
df_temp.head()
```

Out[2]:

	time	tavg	tmin	tmax	prcp
0	01-01-1990	23.2	17.0	NaN	0.0
1	02-01-1990	22.2	16.5	29.9	0.0
2	03-01-1990	21.8	16.3	30.7	0.0
3	04-01-1990	25.4	17.9	31.8	0.0
4	05-01-1990	26.5	19.3	33.7	0.0

In [3]:
```
df_temp['time'] = pd.to_datetime(df_temp['time'], format='%d-%m-%Y')
```

In [4]:
```
df = df_temp[['time', 'tavg']]
df.head()
```

Out[4]:

	time	tavg
0	1990-01-01	23.2
1	1990-01-02	22.2
2	1990-01-03	21.8
3	1990-01-04	25.4
4	1990-01-05	26.5

In [5]:
```
df['time'] = pd.to_datetime(df['time'])
df['Year'] = df['time'].dt.year
df['Month'] = df['time'].dt.month
df = df.groupby(['Year', 'Month'])['tavg'].mean().round(2).reset_index()

df.head()
```

Out[5]:

	Year	Month	tavg
0	1990	1	24.84
1	1990	2	24.95
2	1990	3	25.65
3	1990	4	27.85
4	1990	5	29.85

We will be calculating the average temperature for every month and will be using that for our analysis

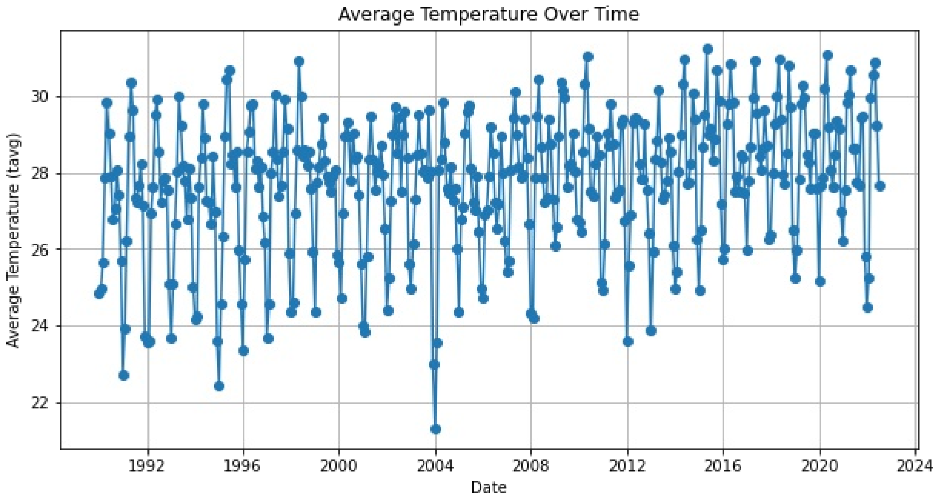
In [6]:
```
df.isna().sum()
```

Out[6]:
```
Year     0
Month    0
tavg     0
dtype: int64
```

Time Series Plot for tavg by month

In [17]:
```
df['Date'] = pd.to_datetime(df[['Year', 'Month']].assign(DAY=1))

# Plotting
```

```
plt.figure(figsize=(10, 5))
plt.plot(df['Date'], df['tavg'], marker='o')
plt.title('Average Temperature Over Time')
plt.xlabel('Date')
plt.ylabel('Average Temperature (tavg)')
plt.grid(True)
plt.show()
```



In [19]:

```
# Dropping the 'Date' column
df = df.drop(columns=['Date'])
df.head()
```

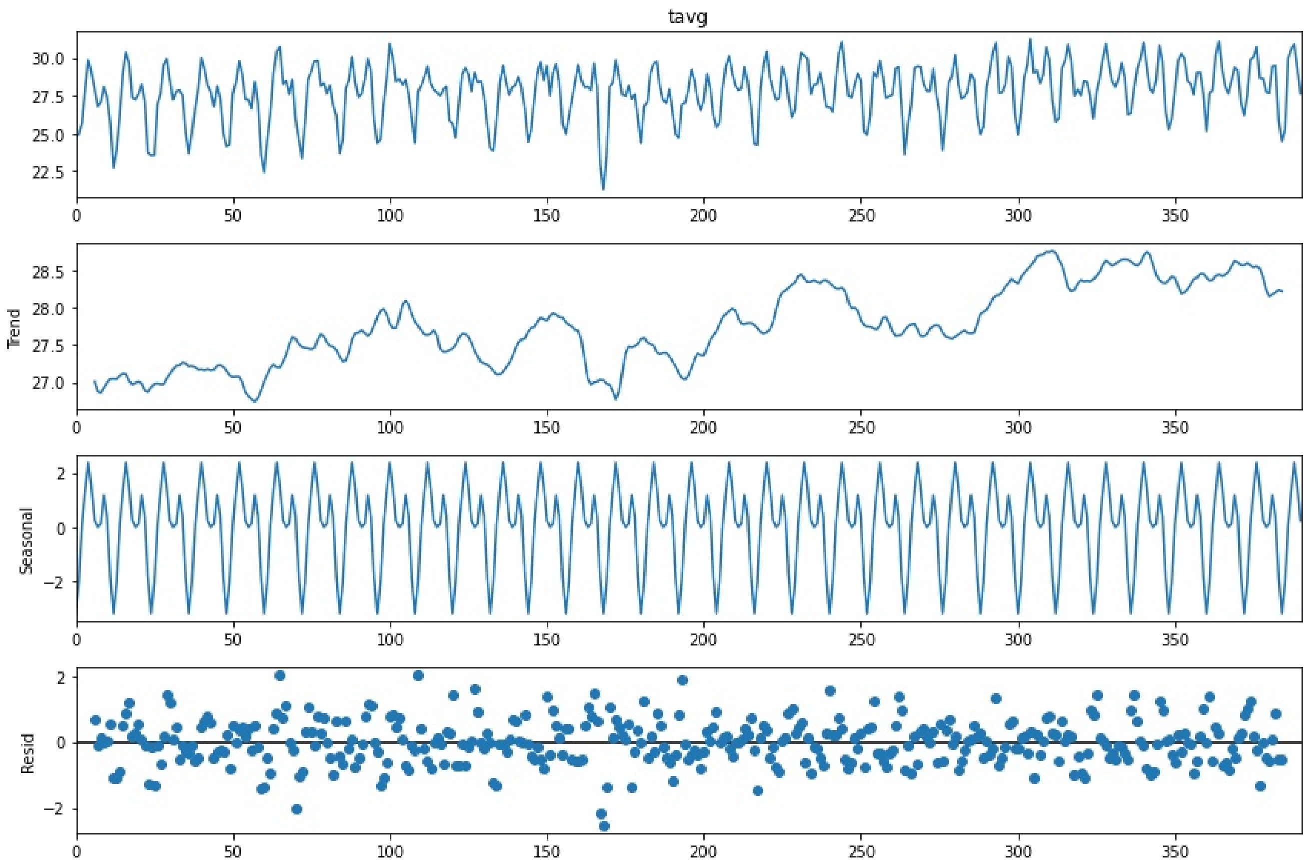
Out[19]:

	Year	Month	tavg
0	1990	1	24.84
1	1990	2	24.95
2	1990	3	25.65
3	1990	4	27.85
4	1990	5	29.85

Seasonality

In [21]:

```
result = seasonal_decompose(df['tavg'], model='additive', period=12)
fig = result.plot()
fig.set_size_inches((12, 8))
fig.tight_layout()
plt.show()
```



After decomposing the original time series we can clearly identify that there is a seasonal component within the dataset and can conclude that the data is indeed seasonal.

Stationarity

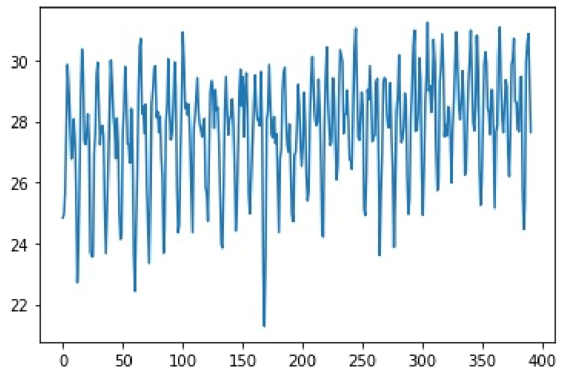
In [98]:
```
# Perform the ADF test
result_adf = adfuller(df['tavg'])

# Print the ADF test results
print('ADF Statistic:', result_adf[0])
print('p-value:', result_adf[1])
print('Critical Values:', result_adf[4])

# Check the p-value against a significance level (e.g., 0.05)
if result_adf[1] <= 0.05:
    print("Reject the null hypothesis; the time series is likely stationary.")
else:
    print("Fail to reject the null hypothesis; the time series may not be stationary.")
```

```
ADF Statistic: -2.1514576509422016
p-value: 0.22433255503804606
Critical Values: {'1%': -3.4478619826418817, '5%': -2.869257669826291, '10%': -2.570881358363513}
Fail to reject the null hypothesis; the time series may not be stationary.
```

ADF test tells us that the time series is not stationary

In [99]:
```
plt.plot(df['tavg'])
```

Out[99]:
```
[<matplotlib.lines.Line2D at 0x288827115e0>]
```



Plotting the time series before differencing it to see the difference

Differencing

In [100…
```
df['tavg'] = df['tavg'].diff()
df = df.dropna()
```

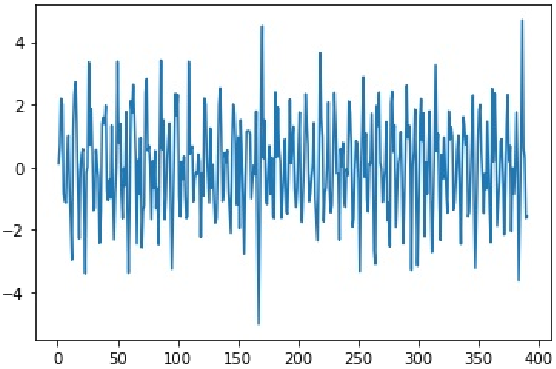
In [101…
```
# Perform the ADF test
result_adf = adfuller(df['tavg'])

# Print the ADF test results
print('ADF Statistic:', result_adf[0])
print('p-value:', result_adf[1])
print('Critical Values:', result_adf[4])

# Check the p-value against a significance level (e.g., 0.05)
if result_adf[1] <= 0.05:
    print("Reject the null hypothesis; the time series is likely stationary.")
else:
    print("Fail to reject the null hypothesis; the time series may not be stationary.")
```

```
ADF Statistic: -10.228642528228558
p-value: 5.0975102794413036e-18
Critical Values: {'1%': -3.4478619826418817, '5%': -2.869257669826291, '10%': -2.570881358363513}
Reject the null hypothesis; the time series is likely stationary.
```

After performing the differencing operation we got the ADF test result as stationary

In [102…
```
plt.plot(df['tavg'])
```

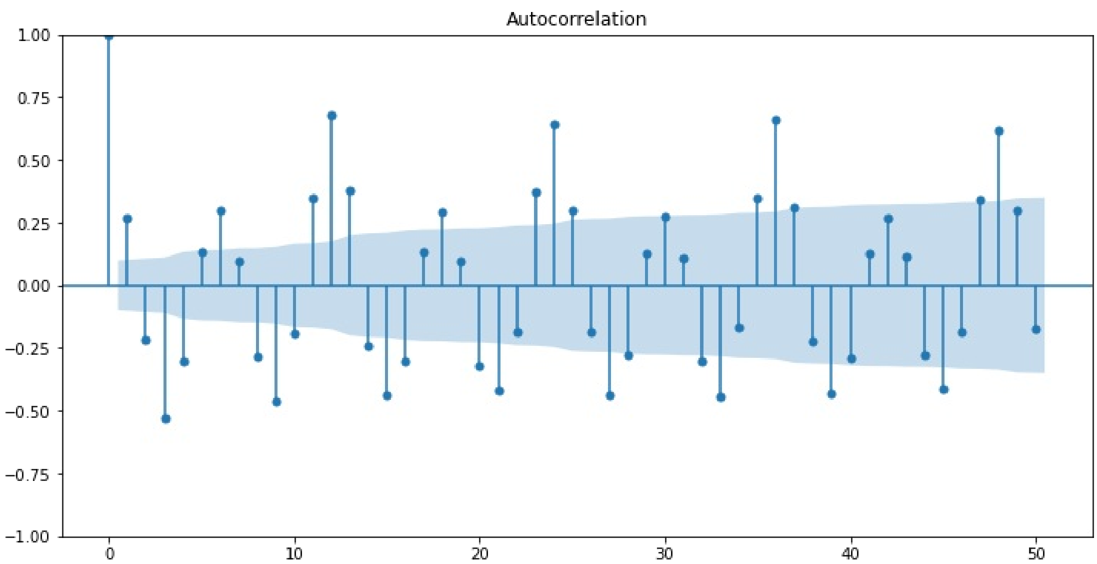
Out[102]:
```
[<matplotlib.lines.Line2D at 0x2888262c940>]
```



We can clearly see the difference in the time series plto

ACF & PACF

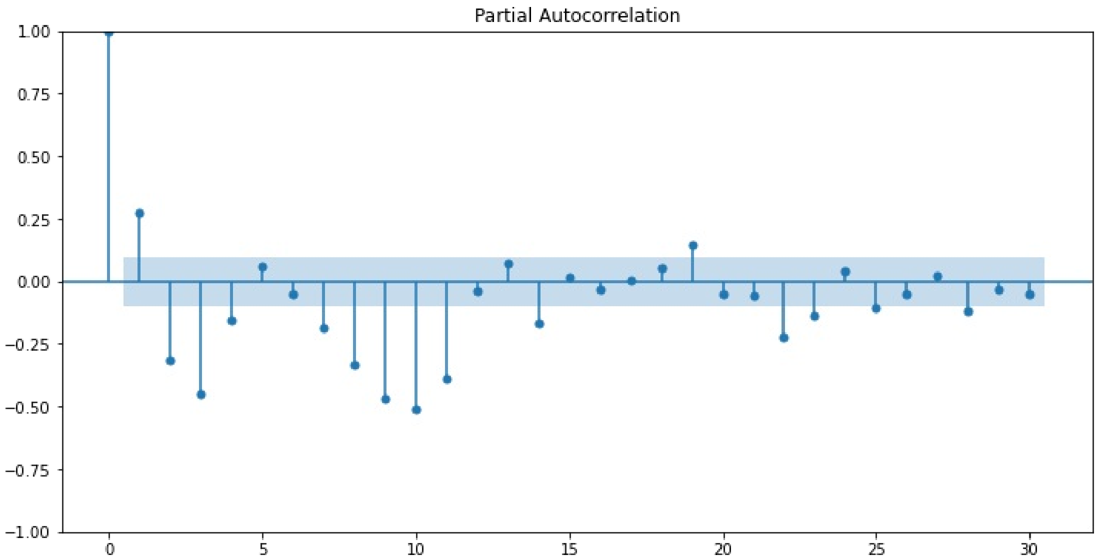
In [103…
```
# Plot ACF
plt.figure(figsize=(12, 6))
plot_acf(df['tavg'], ax=plt.gca(), lags=50)
plt.show()
```



We can observe 4 significant lags for non seasonal component and about 3 to 4 lags in seasonal component.

In [104…
```
plt.figure(figsize=(12, 6))
```

```
plot_pacf(df['tavg'], ax=plt.gca(), lags=30)
plt.show()
```



We can observe 4 significant lags for non seasonal component and around 1- 2 significant lags for seasonal component.

TRAIN TEST SPLIT

In [23]:

```
length = len(df)
df_train = df.head(length - 30)
df_test = df.tail(30)
```

SARIMA MODELS

Based on what we observed in the acf and pacf plot let us form a loop with a range of values for the parameters in the SARIMA model which will help us determine our best model with the lowest aic score

In [111…

```
best_aic = float('inf')
best_params = None
loop_counter = 0

# Loop over the range of parameters
for p in range(3, 6):
    for d in [1]:
        for q in range(3, 6):
            for P in range(1, 5):
                for D in [1]:
                    for Q in range(3, 7):
                        loop_counter += 1  # Increment the loop counter
                        try:
                            # Define and fit the model
                            model = SARIMAX(df_train['tavg'],
                                            order=(p, d, q),
                                            seasonal_order=(P, D, Q, 12),
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
                            results = model.fit()

                            # Check if the current model's AIC is better (lower)
                            if results.aic < best_aic:
                                best_aic = results.aic
                                best_params = (p, d, q, P, D, Q)

                            # Print the loop counter for tracking progress
                            print(f"Loop {loop_counter} completed for parameters {(p, d, q, P, D, Q)}")

                        except Exception as e:
                            # Catch exceptions, which are common in model fitting
                            print(f"An error occurred for parameters {(p, d, q, P, D, Q)}: {e}")

# Print the best parameters and corresponding AIC
print(f"Best Parameters: {best_params}")
print(f"Best AIC: {best_aic}")

# Print the total number of loops completed
print(f"Total loops completed: {loop_counter}")
```

```
Loop 1 completed for parameters (3, 1, 3, 1, 1, 3)
```

```
Loop 2 completed for parameters (3, 1, 3, 1, 1, 4)
Loop 3 completed for parameters (3, 1, 3, 1, 1, 5)
Loop 4 completed for parameters (3, 1, 3, 1, 1, 6)
Loop 5 completed for parameters (3, 1, 3, 2, 1, 3)
Loop 6 completed for parameters (3, 1, 3, 2, 1, 4)
Loop 7 completed for parameters (3, 1, 3, 2, 1, 5)
Loop 8 completed for parameters (3, 1, 3, 2, 1, 6)
Loop 9 completed for parameters (3, 1, 3, 3, 1, 3)
Loop 10 completed for parameters (3, 1, 3, 3, 1, 4)
Loop 11 completed for parameters (3, 1, 3, 3, 1, 5)
Loop 12 completed for parameters (3, 1, 3, 3, 1, 6)
Loop 13 completed for parameters (3, 1, 3, 4, 1, 3)
Loop 14 completed for parameters (3, 1, 3, 4, 1, 4)
Loop 15 completed for parameters (3, 1, 3, 4, 1, 5)
Loop 16 completed for parameters (3, 1, 3, 4, 1, 6)
Loop 17 completed for parameters (3, 1, 4, 1, 1, 3)
Loop 18 completed for parameters (3, 1, 4, 1, 1, 4)
Loop 19 completed for parameters (3, 1, 4, 1, 1, 5)
Loop 20 completed for parameters (3, 1, 4, 1, 1, 6)
Loop 21 completed for parameters (3, 1, 4, 2, 1, 3)
Loop 22 completed for parameters (3, 1, 4, 2, 1, 4)
Loop 23 completed for parameters (3, 1, 4, 2, 1, 5)
Loop 24 completed for parameters (3, 1, 4, 2, 1, 6)
Loop 25 completed for parameters (3, 1, 4, 3, 1, 3)
Loop 26 completed for parameters (3, 1, 4, 3, 1, 4)
Loop 27 completed for parameters (3, 1, 4, 3, 1, 5)
Loop 28 completed for parameters (3, 1, 4, 3, 1, 6)
Loop 29 completed for parameters (3, 1, 4, 4, 1, 3)
Loop 30 completed for parameters (3, 1, 4, 4, 1, 4)
Loop 31 completed for parameters (3, 1, 4, 4, 1, 5)
Loop 32 completed for parameters (3, 1, 4, 4, 1, 6)
Loop 33 completed for parameters (3, 1, 5, 1, 1, 3)
Loop 34 completed for parameters (3, 1, 5, 1, 1, 4)
Loop 35 completed for parameters (3, 1, 5, 1, 1, 5)
Loop 36 completed for parameters (3, 1, 5, 1, 1, 6)
Loop 37 completed for parameters (3, 1, 5, 2, 1, 3)
Loop 38 completed for parameters (3, 1, 5, 2, 1, 4)
Loop 39 completed for parameters (3, 1, 5, 2, 1, 5)
Loop 40 completed for parameters (3, 1, 5, 2, 1, 6)
Loop 41 completed for parameters (3, 1, 5, 3, 1, 3)
Loop 42 completed for parameters (3, 1, 5, 3, 1, 4)
Loop 43 completed for parameters (3, 1, 5, 3, 1, 5)
Loop 44 completed for parameters (3, 1, 5, 3, 1, 6)
Loop 45 completed for parameters (3, 1, 5, 4, 1, 3)
Loop 46 completed for parameters (3, 1, 5, 4, 1, 4)
Loop 47 completed for parameters (3, 1, 5, 4, 1, 5)
Loop 48 completed for parameters (3, 1, 5, 4, 1, 6)
Loop 49 completed for parameters (4, 1, 3, 1, 1, 3)
Loop 50 completed for parameters (4, 1, 3, 1, 1, 4)
Loop 51 completed for parameters (4, 1, 3, 1, 1, 5)
Loop 52 completed for parameters (4, 1, 3, 1, 1, 6)
Loop 53 completed for parameters (4, 1, 3, 2, 1, 3)
Loop 54 completed for parameters (4, 1, 3, 2, 1, 4)
Loop 55 completed for parameters (4, 1, 3, 2, 1, 5)
Loop 56 completed for parameters (4, 1, 3, 2, 1, 6)
Loop 57 completed for parameters (4, 1, 3, 3, 1, 3)
Loop 58 completed for parameters (4, 1, 3, 3, 1, 4)
Loop 59 completed for parameters (4, 1, 3, 3, 1, 5)
Loop 60 completed for parameters (4, 1, 3, 3, 1, 6)
Loop 61 completed for parameters (4, 1, 3, 4, 1, 3)
Loop 62 completed for parameters (4, 1, 3, 4, 1, 4)
Loop 63 completed for parameters (4, 1, 3, 4, 1, 5)
Loop 64 completed for parameters (4, 1, 3, 4, 1, 6)
Loop 65 completed for parameters (4, 1, 4, 1, 1, 3)
Loop 66 completed for parameters (4, 1, 4, 1, 1, 4)
Loop 67 completed for parameters (4, 1, 4, 1, 1, 5)
Loop 68 completed for parameters (4, 1, 4, 1, 1, 6)
Loop 69 completed for parameters (4, 1, 4, 2, 1, 3)
Loop 70 completed for parameters (4, 1, 4, 2, 1, 4)
Loop 71 completed for parameters (4, 1, 4, 2, 1, 5)
Loop 72 completed for parameters (4, 1, 4, 2, 1, 6)
Loop 73 completed for parameters (4, 1, 4, 3, 1, 3)
Loop 74 completed for parameters (4, 1, 4, 3, 1, 4)
Loop 75 completed for parameters (4, 1, 4, 3, 1, 5)
Loop 76 completed for parameters (4, 1, 4, 3, 1, 6)
Loop 77 completed for parameters (4, 1, 4, 4, 1, 3)
Loop 78 completed for parameters (4, 1, 4, 4, 1, 4)
Loop 79 completed for parameters (4, 1, 4, 4, 1, 5)
Loop 80 completed for parameters (4, 1, 4, 4, 1, 6)
Loop 81 completed for parameters (4, 1, 5, 1, 1, 3)
Loop 82 completed for parameters (4, 1, 5, 1, 1, 4)
Loop 83 completed for parameters (4, 1, 5, 1, 1, 5)
Loop 84 completed for parameters (4, 1, 5, 1, 1, 6)
Loop 85 completed for parameters (4, 1, 5, 2, 1, 3)
Loop 86 completed for parameters (4, 1, 5, 2, 1, 4)
Loop 87 completed for parameters (4, 1, 5, 2, 1, 5)
Loop 88 completed for parameters (4, 1, 5, 2, 1, 6)
Loop 89 completed for parameters (4, 1, 5, 3, 1, 3)
Loop 90 completed for parameters (4, 1, 5, 3, 1, 4)
```

```
Loop 91 completed for parameters (4, 1, 5, 3, 1, 5)
Loop 92 completed for parameters (4, 1, 5, 3, 1, 6)
Loop 93 completed for parameters (4, 1, 5, 4, 1, 3)
Loop 94 completed for parameters (4, 1, 5, 4, 1, 4)
Loop 95 completed for parameters (4, 1, 5, 4, 1, 5)
Loop 96 completed for parameters (4, 1, 5, 4, 1, 6)
Loop 97 completed for parameters (5, 1, 3, 1, 1, 3)
Loop 98 completed for parameters (5, 1, 3, 1, 1, 4)
Loop 99 completed for parameters (5, 1, 3, 1, 1, 5)
Loop 100 completed for parameters (5, 1, 3, 1, 1, 6)
Loop 101 completed for parameters (5, 1, 3, 2, 1, 3)
Loop 102 completed for parameters (5, 1, 3, 2, 1, 4)
Loop 103 completed for parameters (5, 1, 3, 2, 1, 5)
Loop 104 completed for parameters (5, 1, 3, 2, 1, 6)
Loop 105 completed for parameters (5, 1, 3, 3, 1, 3)
Loop 106 completed for parameters (5, 1, 3, 3, 1, 4)
Loop 107 completed for parameters (5, 1, 3, 3, 1, 5)
Loop 108 completed for parameters (5, 1, 3, 3, 1, 6)
Loop 109 completed for parameters (5, 1, 3, 4, 1, 3)
Loop 110 completed for parameters (5, 1, 3, 4, 1, 4)
Loop 111 completed for parameters (5, 1, 3, 4, 1, 5)
Loop 112 completed for parameters (5, 1, 3, 4, 1, 6)
Loop 113 completed for parameters (5, 1, 4, 1, 1, 3)
Loop 114 completed for parameters (5, 1, 4, 1, 1, 4)
Loop 115 completed for parameters (5, 1, 4, 1, 1, 5)
Loop 116 completed for parameters (5, 1, 4, 1, 1, 6)
Loop 117 completed for parameters (5, 1, 4, 2, 1, 3)
Loop 118 completed for parameters (5, 1, 4, 2, 1, 4)
Loop 119 completed for parameters (5, 1, 4, 2, 1, 5)
Loop 120 completed for parameters (5, 1, 4, 2, 1, 6)
Loop 121 completed for parameters (5, 1, 4, 3, 1, 3)
Loop 122 completed for parameters (5, 1, 4, 3, 1, 4)
Loop 123 completed for parameters (5, 1, 4, 3, 1, 5)
Loop 124 completed for parameters (5, 1, 4, 3, 1, 6)
Loop 125 completed for parameters (5, 1, 4, 4, 1, 3)
Loop 126 completed for parameters (5, 1, 4, 4, 1, 4)
Loop 127 completed for parameters (5, 1, 4, 4, 1, 5)
Loop 128 completed for parameters (5, 1, 4, 4, 1, 6)
Loop 129 completed for parameters (5, 1, 5, 1, 1, 3)
Loop 130 completed for parameters (5, 1, 5, 1, 1, 4)
Loop 131 completed for parameters (5, 1, 5, 1, 1, 5)
Loop 132 completed for parameters (5, 1, 5, 1, 1, 6)
Loop 133 completed for parameters (5, 1, 5, 2, 1, 3)
Loop 134 completed for parameters (5, 1, 5, 2, 1, 4)
Loop 135 completed for parameters (5, 1, 5, 2, 1, 5)
Loop 136 completed for parameters (5, 1, 5, 2, 1, 6)
Loop 137 completed for parameters (5, 1, 5, 3, 1, 3)
Loop 138 completed for parameters (5, 1, 5, 3, 1, 4)
Loop 139 completed for parameters (5, 1, 5, 3, 1, 5)
Loop 140 completed for parameters (5, 1, 5, 3, 1, 6)
Loop 141 completed for parameters (5, 1, 5, 4, 1, 3)
Loop 142 completed for parameters (5, 1, 5, 4, 1, 4)
Loop 143 completed for parameters (5, 1, 5, 4, 1, 5)
Loop 144 completed for parameters (5, 1, 5, 4, 1, 6)
Best Parameters: (3, 1, 5, 3, 1, 6)
Best AIC: 617.4339407694008
Total loops completed: 144
```

After 144 different combinations from the given range we got the best parameters as (3, 1, 5, 3, 1, 6) with an AIC of 617.4339407694008

Best Model

In [113…

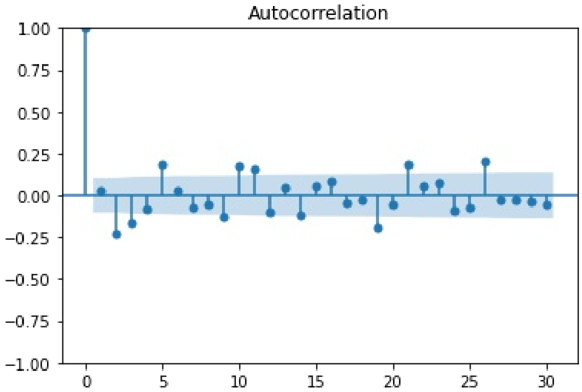
```
p, d, q, P, D, Q, s = 3, 1, 5, 3, 1, 6, 12
model = SARIMAX(df_train['tavg'],
                order=(p, d, q),
                seasonal_order=(P, D, Q, s),
                enforce_stationarity=False,
                enforce_invertibility=False)

results = model.fit()

# Show the results
print(results.summary())
```

```
                                           SARIMAX Results
=========================================================================================================
Dep. Variable:                                      tavg   No. Observations:                        360
Model:             SARIMAX(3, 1, 5)x(3, 1, [1, 2, 3, 4, 5, 6], 12)   Log Likelihood                  -290.717
Date:                               Tue, 12 Dec 2023   AIC                                      617.434
Time:                                       19:24:52   BIC                                      682.139
Sample:                                            0   HQIC                                     643.420
                                               - 360
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.9720      0.333     -2.919      0.004      -1.625      -0.319
ar.L2         -0.0973      0.233     -0.418      0.676      -0.553       0.359
ar.L3         -0.0449      0.233     -0.193      0.847      -0.502       0.412
ma.L1         -0.5706      0.465     -1.228      0.220      -1.482       0.340
ma.L2         -1.1147      0.394     -2.832      0.005      -1.886      -0.343
ma.L3          0.2064      0.390      0.530      0.596      -0.557       0.970
ma.L4          0.1278      0.392      0.326      0.745      -0.641       0.897
ma.L5          0.3500      0.207      1.693      0.090      -0.055       0.755
ar.S.L12      -0.3842      0.131     -2.930      0.003      -0.641      -0.127
ar.S.L24      -0.3210      0.108     -2.974      0.003      -0.533      -0.109
ar.S.L36      -0.5445      0.087     -6.267      0.000      -0.715      -0.374
ma.S.L12      -0.4939      0.166     -2.984      0.003      -0.818      -0.170
ma.S.L24      -0.0992      0.206     -0.481      0.630      -0.503       0.305
ma.S.L36       0.4639      0.154      3.020      0.003       0.163       0.765
ma.S.L48      -0.7375      0.141     -5.229      0.000      -1.014      -0.461
ma.S.L60      -0.0389      0.102     -0.380      0.704      -0.239       0.162
ma.S.L72       0.1236      0.090      1.366      0.172      -0.054       0.301
sigma2         0.4130      0.135      3.066      0.002       0.149       0.677
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 2.10
Prob(Q):                              0.96   Prob(JB):                         0.35
Heteroskedasticity (H):               0.57   Skew:                            -0.11
Prob(H) (two-sided):                  0.01   Kurtosis:                         3.37
===================================================================================

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

Residual Analysis

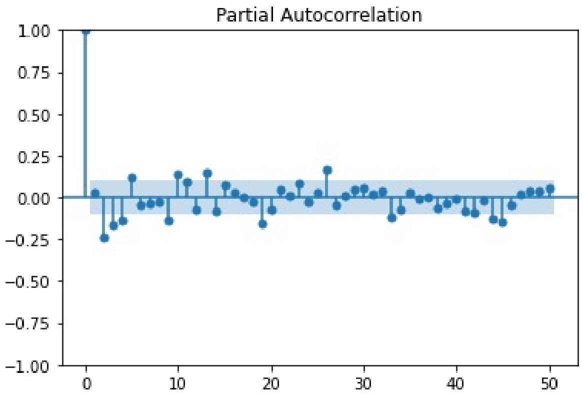
In [114…

```
plot_acf(results.resid, lags = 30);
```



In [115…

```
plot_pacf(results.resid, lags = 50);
```



In [116…

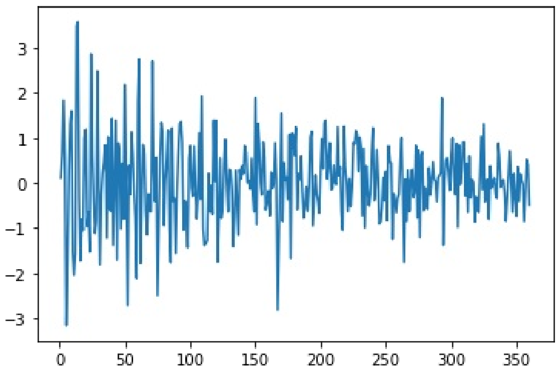
```
sm.stats.acorr_ljungbox(results.resid, lags = 20)
```

Out[116]:

	lb_stat	lb_pvalue
1	0.251719	6.158675e-01
2	19.831900	4.938074e-05
3	30.125658	1.298564e-06
4	32.336190	1.633080e-06
5	44.905053	1.516812e-08
6	45.263562	4.148416e-08
7	47.094797	5.349329e-08
8	48.223288	8.954389e-08
9	54.278668	1.670978e-08
10	65.831946	2.808869e-10
11	74.848793	1.449041e-11
12	78.481604	8.033447e-12
13	79.192878	1.564579e-11
14	84.433957	4.223748e-12
15	85.569211	6.601915e-12
16	88.446400	4.830625e-12
17	89.185961	8.581661e-12
18	89.396459	1.852826e-11
19	102.998050	1.528925e-13
20	103.988135	2.418961e-13

In [117…

```
fig = results.resid.plot()
```



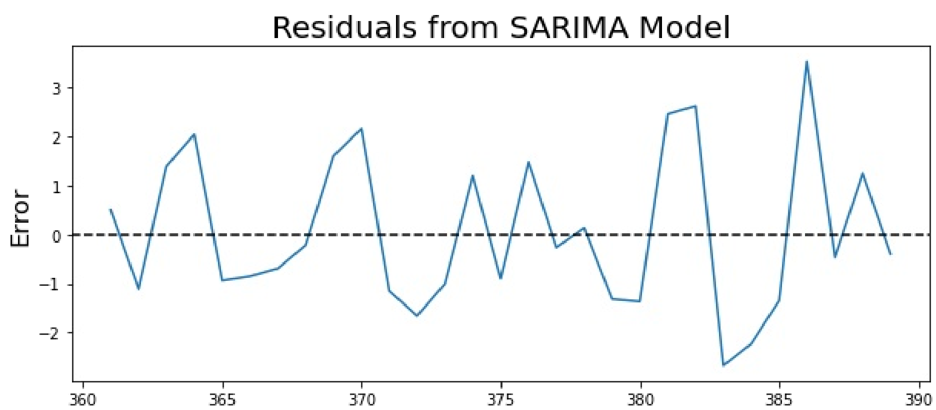
Residuals plot to observe the residuals that we got from our model

In [118…

```
predictions = results.forecast(len(df_test))
predictions = pd.Series(predictions, index=df_test.index)
residuals = df_test['tavg'] - predictions
plt.figure(figsize=(10,4))
plt.plot(residuals)
plt.axhline(0, linestyle='--', color='k')
plt.title('Residuals from SARIMA Model', fontsize=20)
plt.ylabel('Error', fontsize=16)
plt.figure(figsize=(10,4))
```

Out[118]:

```
<Figure size 720x288 with 0 Axes>
```



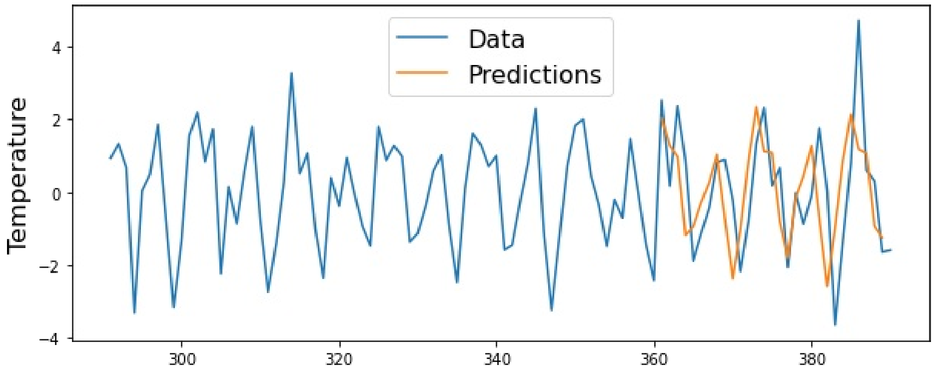

```
<Figure size 720x288 with 0 Axes>
```

This plot gives us a better understanding of how far off we are.

Forecast

In [119…
```
plt.figure(figsize=(10,4))

plt.plot(df['tavg'][len(df)-100:len(df)])
plt.plot(predictions)

plt.legend(('Data', 'Predictions'), fontsize=16, )
plt.ylabel('Temperature', fontsize=16)
```

Out[119]:
```
Text(0, 0.5, 'Temperature')
```



The orange line is the predicted value for our test data set using our best model.

In [120…
```
print('Root Mean Squared Error:', np.sqrt(np.mean(residuals**2)))
```

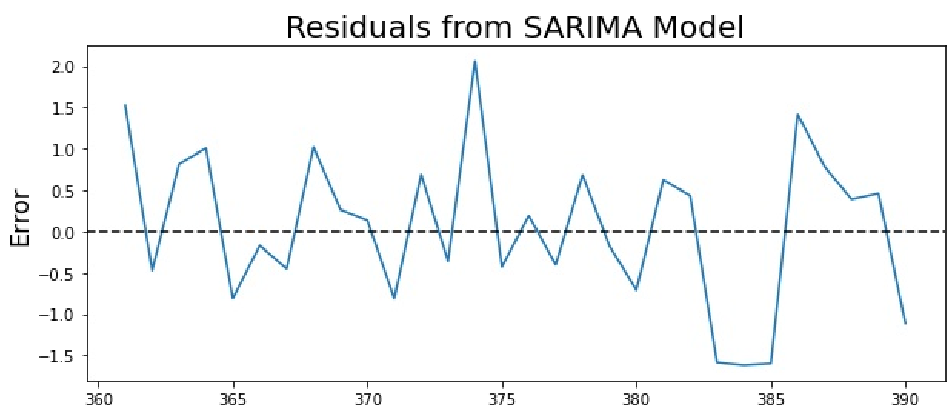
```
Root Mean Squared Error: 1.5681512378138633
```

Rolling Forecast

In [121…
```
rolling_predictions = []
len_test = len(df_test)
len_train = len(df_train)
for i in range(len_train, len_train + len_test):
    train = df[0:i]['tavg']
    model = SARIMAX(train, order = (1, 0, 1), seasonal_order = (1, 0, 0, 12), enforce_stationarity = False)
    model_fit = model.fit()
    pred = model_fit.forecast()
    rolling_predictions.append(pred.tolist()[0])
```

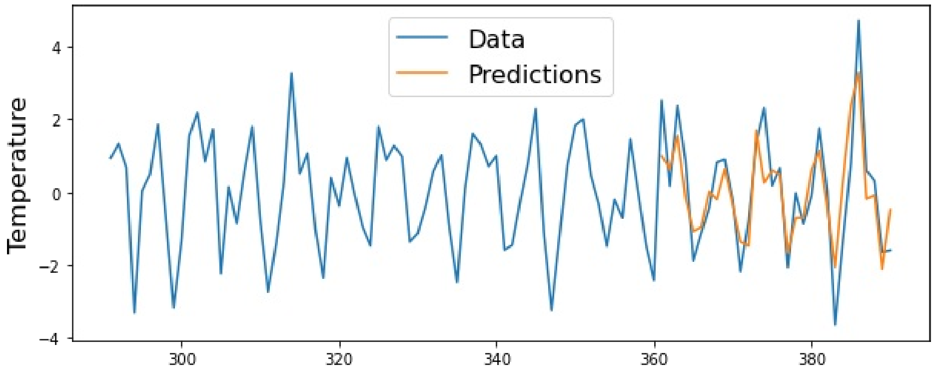
In [122…
```
rolling_predictions = pd.Series(rolling_predictions, index=df_test.index)
residuals = df_test['tavg'] - rolling_predictions
plt.figure(figsize=(10,4))
plt.plot(residuals)
plt.axhline(0, linestyle='--', color='k')
plt.title('Residuals from SARIMA Model', fontsize=20)
plt.ylabel('Error', fontsize=16)
```

Out[122]:
```
Text(0, 0.5, 'Error')
```



In [123…
```
plt.figure(figsize=(10,4))

plt.plot(df['tavg'][len(df)-100:len(df)])
plt.plot(rolling_predictions)

plt.legend(('Data', 'Predictions'), fontsize=16, )
plt.ylabel('Temperature', fontsize=16)
```

Out[123]: Text(0, 0.5, 'Temperature')



We were able to get a closer and better prediction using rolling forecast and it can be clearly seen in our plot.

In [124…

```
print('Root Mean Squared Error:', np.sqrt(np.mean(residuals**2)))
```

```
Root Mean Squared Error: 0.9241870002448656
```

Conclusion

In this comprehensive time series analysis project, we successfully addressed the challenges posed by seasonal data. Our approach involved several critical steps, starting with verifying the stationarity of the data. We achieved stationarity through appropriate differencing techniques, ensuring a robust foundation for further analysis.

Our exploration of various SARIMA models was a key aspect of this project. After thorough testing and evaluation, we identified the model with parameters (3, 1, 5, 3, 1, 6) as the most effective, evidenced by its optimal AIC score of 617.43. This model not only outperformed others in terms of fit but also demonstrated its efficacy in forecasting.

The highlight of our findings was the comparison between the Rolling Forecast and the Simple Forecast methods. The Rolling Forecast method proved to be superior, yielding predictions that closely aligned with the actual values. This was quantitatively supported by a significantly lower Root Mean Squared Error (RMSE) of 0.92, compared to the RMSE of 1.56 obtained using the Simple Forecast approach.

In conclusion, this project not only showcased the effectiveness of the Rolling Forecast method in dealing with seasonal time series data but also underscored the importance of selecting appropriate models based on comprehensive evaluation criteria like the AIC score and RMSE. The insights gained from this analysis are invaluable for accurate forecasting and model selection in similar time series analyses.

In [ ]: