E Navaneet Kumar

Introduction

In the volatile world of financial markets, accurately predicting future price movements and understanding market behavior are critical for investors, traders, and financial analysts. The primary challenge lies in dealing with the inherent unpredictability and fluctuations in financial data. This project focuses on leveraging advanced time series analysis techniques to forecast financial data, specifically examining returns and volatility. By doing so, I aim to provide a robust framework for better decision-making and risk management.

The significance of this project is based in its ability to enhance forecasting accuracy by combining ARMA (AutoRegressive Moving Average) models with GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models. ARMA models are capable at capturing the linear patterns in time series data, while GARCH models excel at modeling volatility clustering in financial markets. By integrating these models, we can achieve a more comprehensive understanding of market dynamics which offers valuable insights that can aid in strategic planning, risk assessment, and optimizing investment strategies.

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 nonseasonal 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 nonnegative 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.

Shapiro Wilk Test

- The Shapiro-Wilk test is a statistical test that checks whether a dataset is normally distributed. It calculates a p-value; if this value is small (less than 0.05), it suggests the data does not follow a normal distribution.

Dataset

https://finance.yahoo.com/quote/DABUR.NS/history/?guccounter=1&period1=1325376000&period2=1672531200

In [1]:

```
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 [2]:

```
df = pd.read_csv('DABUR.csv')
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2012-01-02	99.300003	101.400002	99.050003	100.449997	89.762779	289745.0
1	2012-01-03	101.000000	101.500000	98.449997	100.900002	90.164902	369216.0
2	2012-01-04	101.199997	102.849998	100.599998	101.099998	90.343620	354686.0
3	2012-01-05	101.199997	101.849998	99.449997	100.050003	89.405334	444672.0
4	2012-01-06	101.199997	101.199997	98.500000	98.800003	88.288345	924778.0

In [3]:

```
df.describe()
```

Out[3]:

	Open	High	Low	Close	Adj Close	Volume
count	2713.000000	2713.000000	2713.000000	2713.000000	2713.000000	2.713000e+03
mean	340.821231	344.689219	336.640435	340.496572	324.676184	1.856691e+06
std	151.732568	153.116946	150.038014	151.497707	152.087223	1.560272e+06
min	93.800003	94.900002	92.000000	92.500000	82.658607	0.000000e+00

	Open	High	Low	Close	Adj Close	Volume
25%	225.000000	226.850006	221.250000	223.699997	206.195831	8.797650e+05
50%	307.799988	311.000000	303.500000	307.600006	287.562988	1.422545e+06
75%	477.700012	483.649994	471.049988	477.549988	458.896515	2.312806e+06
max	654.849976	658.950012	649.000000	653.950012	638.903931	2.079605e+07

In [4]:

```
df.isna().sum()
```

Out[4]:

```
Date          0
Open          6
High          6
Low           6
Close         6
Adj Close     6
Volume        6
dtype: int64
```

Found some unexpected values. Need to remove them

In [5]:

```
df.dropna(inplace=True)
```

In [6]:

```
df.isna().sum()
```

Out[6]:

```
Date          0
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

In [7]:

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

```
Min Date: 2012-01-02
Max Date: 2022-12-30
```

My data ranges form 2012 to 2022. We will be using 6 years of data for our analysis.

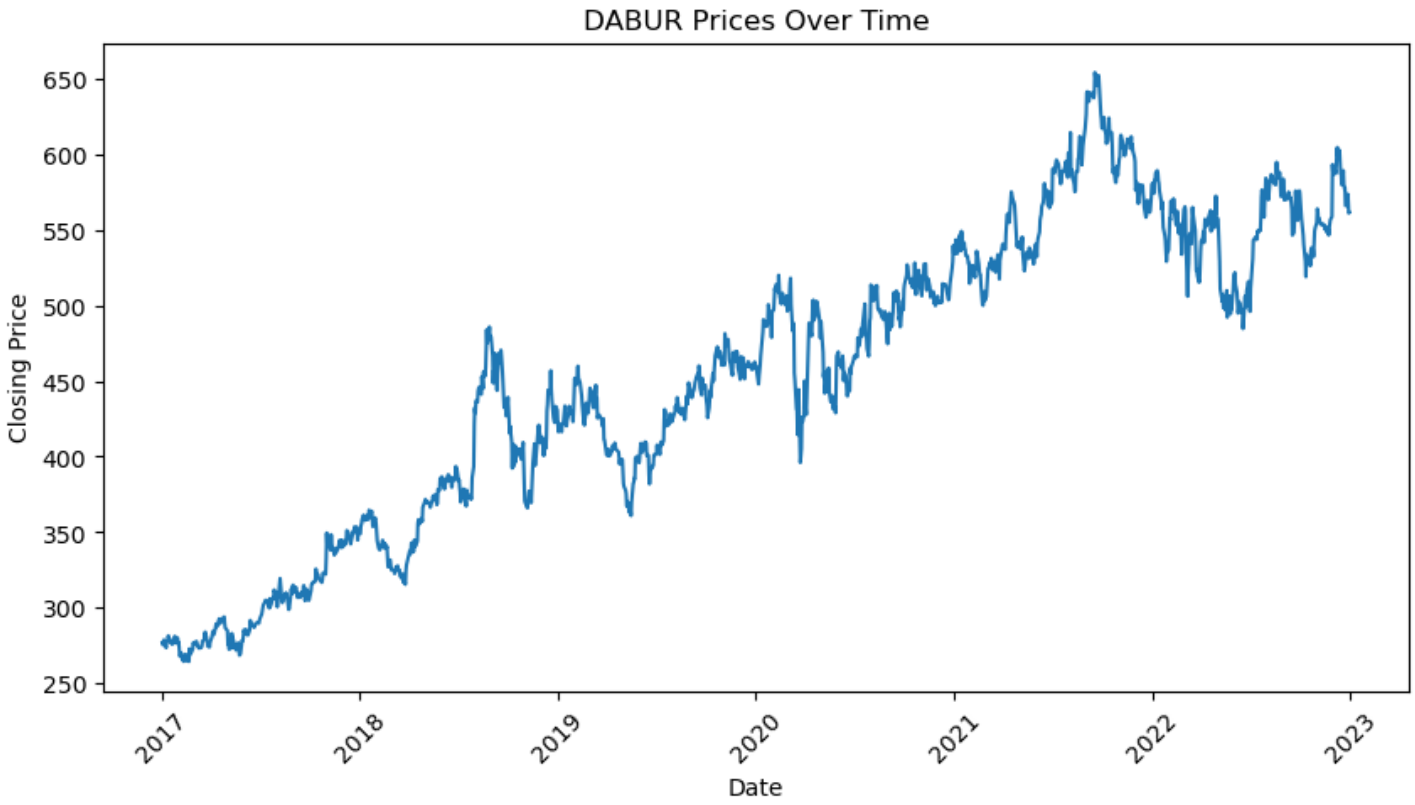
In [8]:

```
df = df[df['Date'] >= '2017-01-01']
```

In [9]:

```
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('DABUR Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Closing Price')
```

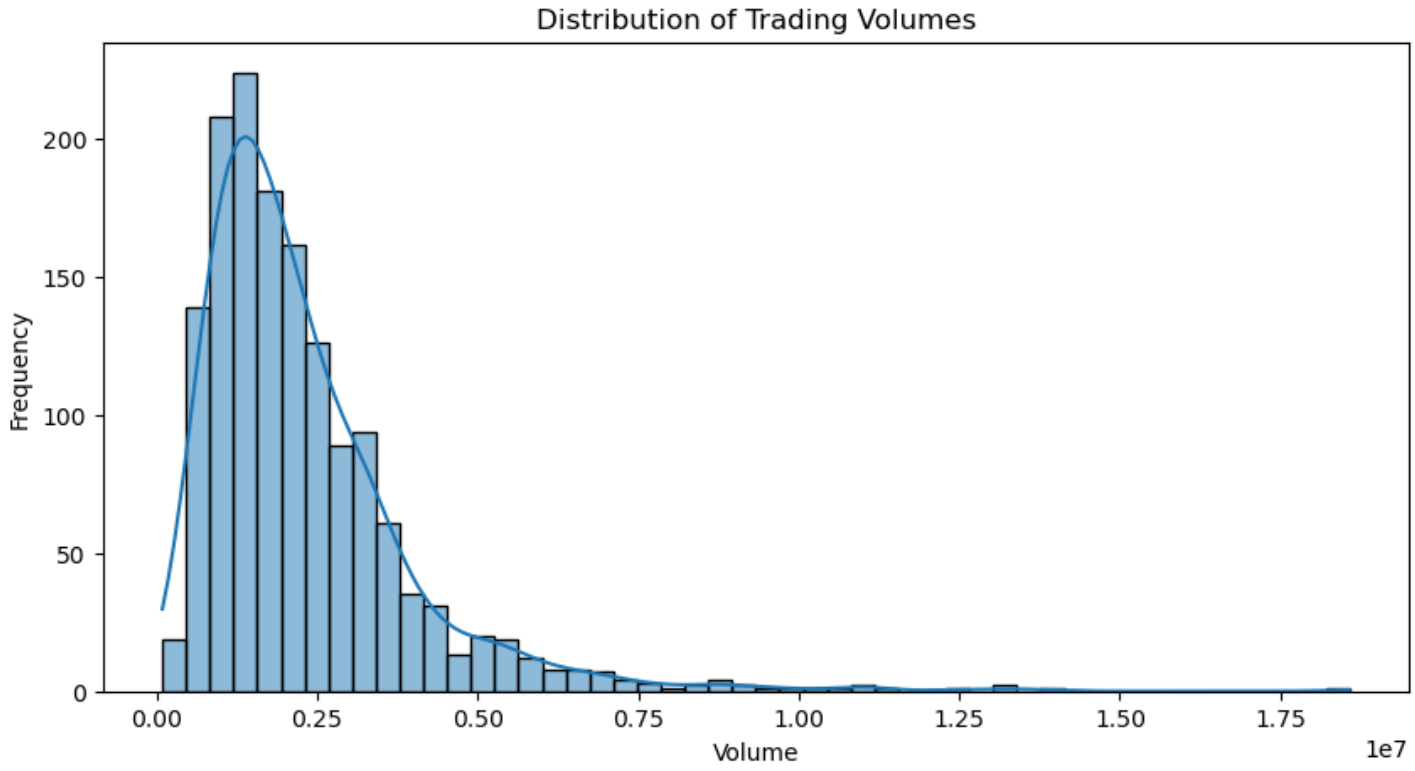
```
# Using YearLocator to set ticks at the start of each year
plt.gca().xaxis.set_major_locator(mdates.YearLocator(base=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xticks(rotation=45)
plt.show()
```



Visualizing the price of DABUR over a period of 6 years

In [10]:

```
plt.figure(figsize=(10, 5))
sns.histplot(df['Volume'], bins=50, 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, which is right-skewed rather than normally distributed. This indicates that most trading volumes are relatively low, with a few exceptionally high volumes. The distribution clusters around the lower end, with a long tail extending towards higher volumes which highlights that very large trades are less frequent.

In [11]:

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

In [12]:

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

Out[12]:

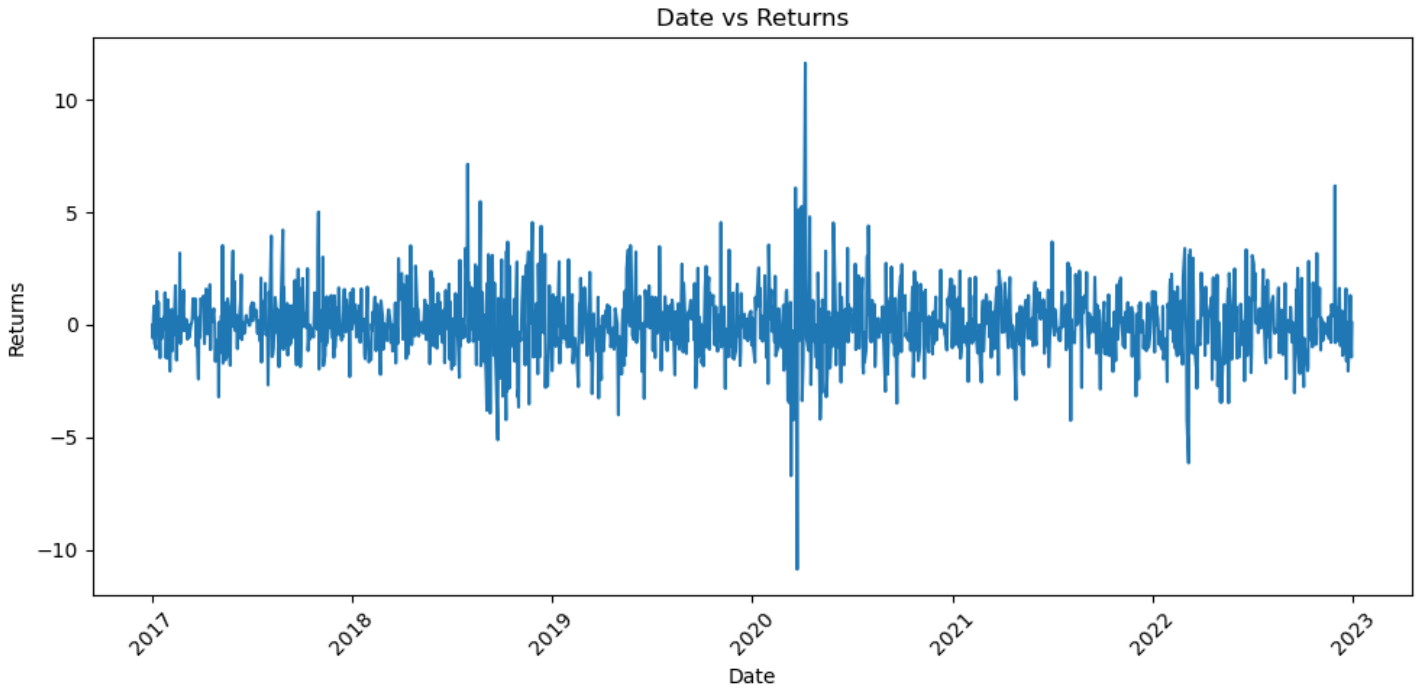
	Date	Open	High	Low	Close	Adj Close	Volume	Returns
1235	2017-01-02	278.000000	278.399994	274.399994	277.149994	259.739594	265516.0	0.000000
1236	2017-01-03	277.500000	278.000000	274.399994	275.500000	258.193268	450185.0	-0.595343
1237	2017-01-04	275.100006	278.200012	272.899994	276.549988	259.177338	2110365.0	0.381121
1238	2017-01-05	278.149994	279.950012	276.950012	278.850006	261.332886	508994.0	0.831683
1239	2017-01-06	279.250000	280.500000	275.500000	276.600006	259.224152	782100.0	-0.806885

In [ ]:

```

```

In [13]:

```
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()
```



The plot represents the variability or volatility in the returns from 2017 to 2023

Stationary Test

In [14]:

```
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}')

    # Interpretation of the ADF test
    print('\nHypothesis:')
    print('Null Hypothesis (H0): The series has a unit root (non-stationary).')
    print('Alternative Hypothesis (H1): The series has no unit root (stationary).')

    if result[1] > 0.05:
        print("\nFailed to reject the null hypothesis (H0), the data has a unit root and
is non-stationary.")
    else:
        print("\nReject 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: -12.8142
p-value: 0.0000
Critical Values:
 1%: -3.4348
 5%: -2.8635
 10%: -2.5678

Hypothesis:
Null Hypothesis (H0): The series has a unit root (non-stationary).
Alternative Hypothesis (H1): The series has no unit root (stationary).

Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
```

The highly negative ADF statistic of -12.8142, coupled with a practically zero p-value, solidly supports rejecting the null hypothesis. This tells us that the time series data is stationary, meaning it does not have a unit root and shows consistent statistical properties over time.

In [15]:

```
ts = pd.DataFrame(ts)
ts.head()
```

Out[15]:

Date	Returns
2017-01-02	0.000000
2017-01-03	-0.595343
2017-01-04	0.381121
2017-01-05	0.831683
2017-01-06	-0.806885

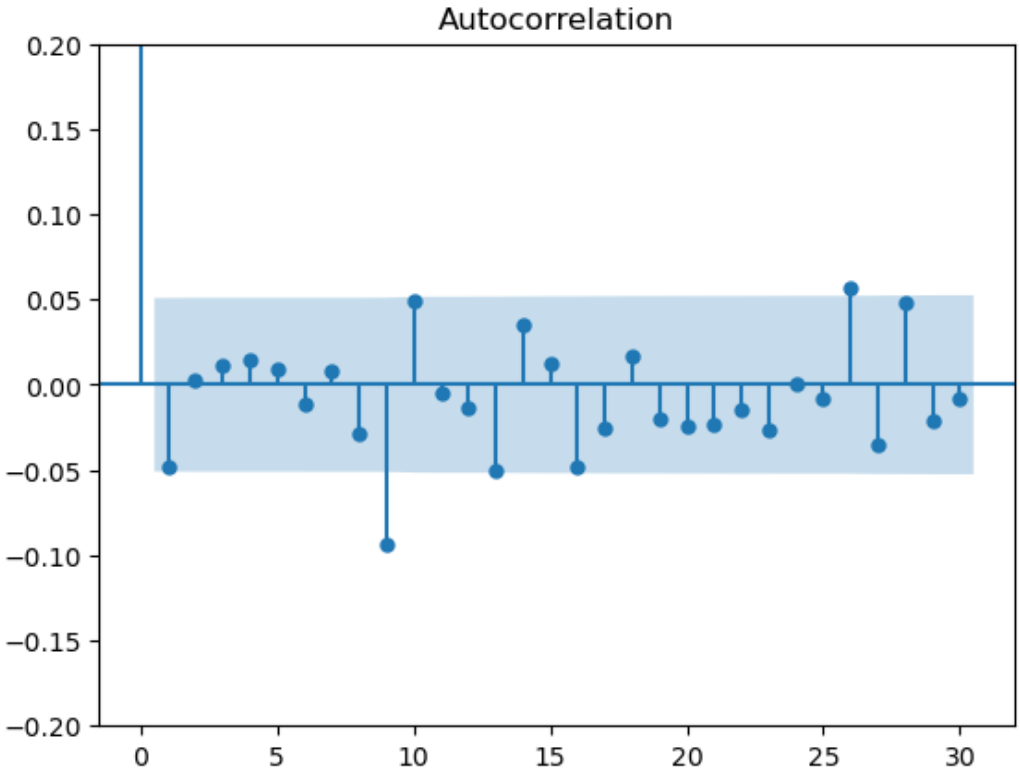
ACF and PACF

In [16]:

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

Out[16]:

```
(-0.2, 0.2)
```

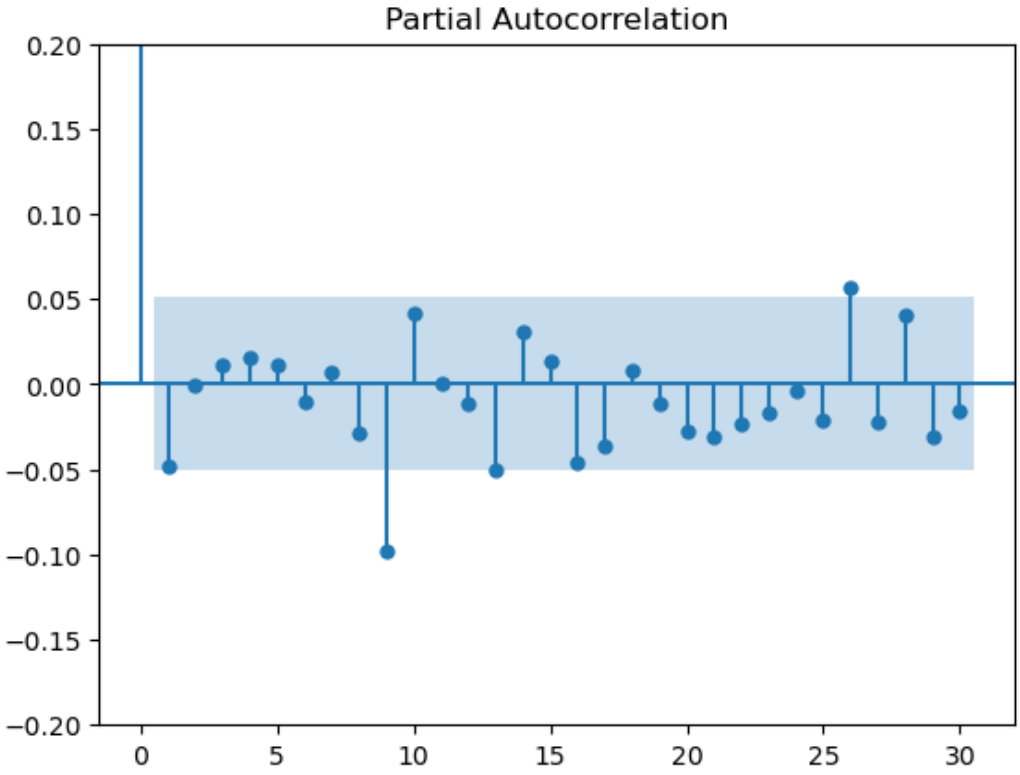


In [17]:

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

Out[17]:

```
(-0.2, 0.2)
```



Based on the ACF and PACF plots I can see the major significant lag is at 9th lag so based on that I will try some models

ARMA (6,1)

In [18]:

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

# Fit an ARIMA model

model = ARIMA(ts['Returns'], order=(6,0,1))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())

# Perform residual checks
residuals = model_fit.resid
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1484
Model:                 ARIMA(6, 0, 1)   Log Likelihood               -2696.950
Date:                Thu, 02 May 2024   AIC                           5411.901
Time:                        03:05:00   BIC                           5459.623
Sample:                             0   HQIC                          5429.689
                                - 1484
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0587      0.039      1.490      0.136      -0.019       0.136
ar.L1         -0.0243      2.088     -0.012      0.991      -4.118       4.069
ar.L2          0.0016      0.101      0.016      0.987      -0.197       0.200
ar.L3          0.0122      0.019      0.628      0.530      -0.026       0.050
ar.L4          0.0163      0.031      0.523      0.601      -0.045       0.077
ar.L5          0.0100      0.040      0.253      0.800      -0.068       0.088
ar.L6         -0.0104      0.031     -0.341      0.733      -0.070       0.049
ma.L1         -0.0240      2.091     -0.011      0.991      -4.123       4.075
sigma2         2.2185      0.049     45.016      0.000       2.122       2.315
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1816.79
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.83   Skew:                             0.25
Prob(H) (two-sided):                  0.04   Kurtosis:                         8.40
===================================================================================

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

ARMA(6,2)

In [19]:

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

# Fit an ARIMA model

model = ARIMA(ts['Returns'], order=(6,0,2))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())

# Perform residual checks
residuals = model_fit.resid
```

```
                               SARIMAX Results
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                    Returns   No. Observations:                 1484
Model:                 ARIMA(6, 0, 2)   Log Likelihood               -2691.386
Date:                Thu, 02 May 2024   AIC                           5402.772
Time:                        03:05:05   BIC                           5455.797
Sample:                             0   HQIC                          5422.537
                                - 1484
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0606      0.017      3.543      0.000       0.027       0.094
ar.L1          1.2719      0.605      2.101      0.036       0.085       2.458
ar.L2         -0.2732      0.561     -0.487      0.626      -1.373       0.826
ar.L3         -0.0046      0.049     -0.095      0.924      -0.100       0.091
ar.L4          0.0009      0.032      0.028      0.978      -0.061       0.063
ar.L5         -0.0066      0.037     -0.177      0.859      -0.080       0.067
ar.L6         -0.0181      0.037     -0.483      0.629      -0.091       0.055
ma.L1         -1.3278      0.604     -2.197      0.028      -2.512      -0.143
ma.L2          0.3402      0.594      0.572      0.567      -0.825       1.505
sigma2         2.2016      0.049     45.188      0.000       2.106       2.297
==========================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1739.54
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.84   Skew:                             0.20
Prob(H) (two-sided):                  0.05   Kurtosis:                         8.29
==========================================================================================

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

ARMA(6,5)

In [20]:

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

# Fit an ARIMA model

model = ARIMA(ts['Returns'], order=(6,0,5))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())

# Perform residual checks
residuals = model_fit.resid
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                    Returns   No. Observations:                 1484
Model:                 ARIMA(6, 0, 5)   Log Likelihood               -2690.823
Date:                Thu, 02 May 2024   AIC                           5407.646
Time:                        03:05:09   BIC                           5476.578
Sample:                             0   HQIC                          5433.339
                                - 1484
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0585      0.038      1.552      0.121      -0.015       0.132
ar.L1          0.2726      0.095      2.880      0.004       0.087       0.458
ar.L2          0.0394      0.076      0.516      0.606      -0.110       0.189
ar.L3         -0.0054      0.068     -0.079      0.937      -0.139       0.129
ar.L4          0.4194      0.065      6.454      0.000       0.292       0.547
ar.L5         -0.8463      0.082    -10.302      0.000      -1.007      -0.685
ar.L6         -0.0536      0.025     -2.128      0.033      -0.103      -0.004
ma.L1         -0.3173      0.093     -3.399      0.001      -0.500      -0.134
ma.L2         -0.0128      0.079     -0.162      0.871      -0.168       0.142
ma.L3          0.0330      0.072      0.460      0.646      -0.108       0.174
```

```
ma.L4          -0.4022      0.071     -5.688      0.000      -0.541      -0.264
ma.L5           0.8416      0.083     10.109      0.000       0.678       1.005
sigma2          2.1501      0.048     44.348      0.000       2.055       2.245
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):              1779.88
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               0.84   Skew:                             0.21
Prob(H) (two-sided):                  0.06   Kurtosis:                         8.35
===================================================================================

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

ARMA(7,2)

In [21]:

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

# Fit an ARIMA model

model = ARIMA(ts['Returns'], order=(7,0,2))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())

# Perform residual checks
residuals = model_fit.resid
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Returns   No. Observations:                 1484
Model:                 ARIMA(7, 0, 2)   Log Likelihood               -2689.510
Date:                Thu, 02 May 2024   AIC                           5401.020
Time:                        03:05:13   BIC                           5459.348
Sample:                             0   HQIC                          5422.761
                                - 1484
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0585      0.040      1.447      0.148      -0.021       0.138
ar.L1         -0.1683      0.060     -2.810      0.005      -0.286      -0.051
ar.L2         -0.8721      0.063    -13.905      0.000      -0.995      -0.749
ar.L3         -0.0307      0.029     -1.063      0.288      -0.087       0.026
ar.L4          0.0176      0.028      0.627      0.531      -0.037       0.073
ar.L5          0.0238      0.030      0.786      0.432      -0.036       0.083
ar.L6          0.0287      0.022      1.281      0.200      -0.015       0.073
ar.L7          0.0581      0.025      2.282      0.022       0.008       0.108
ma.L1          0.1201      0.055      2.172      0.030       0.012       0.228
ma.L2          0.8747      0.058     15.034      0.000       0.761       0.989
sigma2         2.1961      0.053     41.209      0.000       2.092       2.301
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1370.86
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               0.84   Skew:                             0.23
Prob(H) (two-sided):                  0.06   Kurtosis:                         7.69
===================================================================================

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

In [22]:

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

```python
for p in range(1, 15):
    for q in range(1, 15):
        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 = 5402.80234389899
ARIMA(1, 0, 2) : AIC = 5396.750865652479
ARIMA(1, 0, 3) : AIC = 5406.299832751347
ARIMA(1, 0, 4) : AIC = 5408.058980352547
ARIMA(1, 0, 5) : AIC = 5409.824994143258
ARIMA(1, 0, 6) : AIC = 5411.744783045005
ARIMA(1, 0, 7) : AIC = 5413.710019457894
ARIMA(1, 0, 8) : AIC = 5403.5864188165215
ARIMA(1, 0, 9) : AIC = 5400.591160948659
ARIMA(1, 0, 10) : AIC = 5402.271164369802
ARIMA(1, 0, 11) : AIC = 5395.173963708938
ARIMA(1, 0, 12) : AIC = 5397.011635015501
ARIMA(1, 0, 13) : AIC = 5403.347645193009
ARIMA(1, 0, 14) : AIC = 5404.7715348949205
ARIMA(2, 0, 1) : AIC = 5404.802334538377
ARIMA(2, 0, 2) : AIC = 5402.248045532302
ARIMA(2, 0, 3) : AIC = 5400.750577730124
ARIMA(2, 0, 4) : AIC = 5410.0347890804915
ARIMA(2, 0, 5) : AIC = 5404.667604331093
ARIMA(2, 0, 6) : AIC = 5406.408887022455
ARIMA(2, 0, 7) : AIC = 5400.835026748442
ARIMA(2, 0, 8) : AIC = 5401.945957885089
ARIMA(2, 0, 9) : AIC = 5398.49044484905
ARIMA(2, 0, 10) : AIC = 5400.489794372526
ARIMA(2, 0, 11) : AIC = 5400.488083602455
ARIMA(2, 0, 12) : AIC = 5402.351071754423
ARIMA(2, 0, 13) : AIC = 5400.325036864477
ARIMA(2, 0, 14) : AIC = 5402.309374017914
ARIMA(3, 0, 1) : AIC = 5406.377378871495
ARIMA(3, 0, 2) : AIC = 5402.855379323296
ARIMA(3, 0, 3) : AIC = 5403.481320146682
ARIMA(3, 0, 4) : AIC = 5405.988754466959
ARIMA(3, 0, 5) : AIC = 5410.151030802785
ARIMA(3, 0, 6) : AIC = 5403.621597457739
ARIMA(3, 0, 7) : AIC = 5402.535448530778
ARIMA(3, 0, 8) : AIC = 5393.061968070756
ARIMA(3, 0, 9) : AIC = 5398.697622743374
ARIMA(3, 0, 10) : AIC = 5397.122560664024
ARIMA(3, 0, 11) : AIC = 5399.086430930097
ARIMA(3, 0, 12) : AIC = 5403.742351728029
ARIMA(3, 0, 13) : AIC = 5400.850600780092
ARIMA(3, 0, 14) : AIC = 5399.483414550699
ARIMA(4, 0, 1) : AIC = 5408.1589023750175
ARIMA(4, 0, 2) : AIC = 5404.9153932984955
ARIMA(4, 0, 3) : AIC = 5405.310444903649
ARIMA(4, 0, 4) : AIC = 5400.748370931749
ARIMA(4, 0, 5) : AIC = 5397.642180173287
ARIMA(4, 0, 6) : AIC = 5402.802126106476
ARIMA(4, 0, 7) : AIC = 5404.1377752551925
ARIMA(4, 0, 8) : AIC = 5402.1898831714025
ARIMA(4, 0, 9) : AIC = 5390.388205356368
ARIMA(4, 0, 10) : AIC = 5391.681500111064
ARIMA(4, 0, 11) : AIC = 5393.389583070554
ARIMA(4, 0, 12) : AIC = 5395.581397340809
ARIMA(4, 0, 13) : AIC = 5397.12843165296
ARIMA(4, 0, 14) : AIC = 5399.007912655397
```

```
ARIMA(5, 0, 1) : AIC = 5409.940242156227
ARIMA(5, 0, 2) : AIC = 5404.551390209662
ARIMA(5, 0, 3) : AIC = 5406.199406234916
ARIMA(5, 0, 4) : AIC = 5403.203374298048
ARIMA(5, 0, 5) : AIC = 5402.629861531252
ARIMA(5, 0, 6) : AIC = 5404.36840773771
ARIMA(5, 0, 7) : AIC = 5408.119822953384
ARIMA(5, 0, 8) : AIC = 5404.284291880551
ARIMA(5, 0, 9) : AIC = 5392.5192072128675
ARIMA(5, 0, 10) : AIC = 5393.961313107182
ARIMA(5, 0, 11) : AIC = 5395.149904617383
ARIMA(5, 0, 12) : AIC = 5397.598601195038
ARIMA(5, 0, 13) : AIC = 5399.123981301581
ARIMA(5, 0, 14) : AIC = 5398.533569119739
ARIMA(6, 0, 1) : AIC = 5411.900797821638
ARIMA(6, 0, 2) : AIC = 5402.772287403706
ARIMA(6, 0, 3) : AIC = 5403.3235112440925
ARIMA(6, 0, 4) : AIC = 5407.836240948345
ARIMA(6, 0, 5) : AIC = 5407.645866007526
ARIMA(6, 0, 6) : AIC = 5406.092646966501
ARIMA(6, 0, 7) : AIC = 5410.668540076749
ARIMA(6, 0, 8) : AIC = 5404.236784277759
ARIMA(6, 0, 9) : AIC = 5393.502109978785
ARIMA(6, 0, 10) : AIC = 5395.226844255676
ARIMA(6, 0, 11) : AIC = 5397.985779449053
ARIMA(6, 0, 12) : AIC = 5399.183703554676
ARIMA(6, 0, 13) : AIC = 5394.949630898898
ARIMA(6, 0, 14) : AIC = 5394.526947253731
ARIMA(7, 0, 1) : AIC = 5413.777071093309
ARIMA(7, 0, 2) : AIC = 5401.020212336962
ARIMA(7, 0, 3) : AIC = 5401.370877779882
ARIMA(7, 0, 4) : AIC = 5414.752772768743
ARIMA(7, 0, 5) : AIC = 5404.280382496303
ARIMA(7, 0, 6) : AIC = 5403.461414978077
ARIMA(7, 0, 7) : AIC = 5408.947474066128
ARIMA(7, 0, 8) : AIC = 5402.756994394405
ARIMA(7, 0, 9) : AIC = 5388.684198923897
ARIMA(7, 0, 10) : AIC = 5390.750203209266
ARIMA(7, 0, 11) : AIC = 5395.169902889234
ARIMA(7, 0, 12) : AIC = 5396.503524972659
ARIMA(7, 0, 13) : AIC = 5394.565911554795
ARIMA(7, 0, 14) : AIC = 5395.355678805094
ARIMA(8, 0, 1) : AIC = 5403.893728890186
ARIMA(8, 0, 2) : AIC = 5401.816761218195
ARIMA(8, 0, 3) : AIC = 5392.835001380319
ARIMA(8, 0, 4) : AIC = 5399.645592072828
ARIMA(8, 0, 5) : AIC = 5407.5883622452275
ARIMA(8, 0, 6) : AIC = 5413.8806516615605
ARIMA(8, 0, 7) : AIC = 5405.717104003339
ARIMA(8, 0, 8) : AIC = 5404.459638363558
ARIMA(8, 0, 9) : AIC = 5396.9470350476195
ARIMA(8, 0, 10) : AIC = 5390.817023426845
ARIMA(8, 0, 11) : AIC = 5394.146624665469
ARIMA(8, 0, 12) : AIC = 5397.604061110516
ARIMA(8, 0, 13) : AIC = 5402.86963343356
ARIMA(8, 0, 14) : AIC = 5398.26828553105
ARIMA(9, 0, 1) : AIC = 5400.293735111799
ARIMA(9, 0, 2) : AIC = 5398.549009715448
ARIMA(9, 0, 3) : AIC = 5397.2667209587735
ARIMA(9, 0, 4) : AIC = 5391.027809960479
ARIMA(9, 0, 5) : AIC = 5393.179272745818
ARIMA(9, 0, 6) : AIC = 5396.223343965018
ARIMA(9, 0, 7) : AIC = 5390.866465419775
ARIMA(9, 0, 8) : AIC = 5396.633763682458
ARIMA(9, 0, 9) : AIC = 5398.4814382524455
ARIMA(9, 0, 10) : AIC = 5391.636900062089
ARIMA(9, 0, 11) : AIC = 5395.811560903115
ARIMA(9, 0, 12) : AIC = 5397.04035670869
ARIMA(9, 0, 13) : AIC = 5396.304868756625
ARIMA(9, 0, 14) : AIC = 5397.202459144968
ARIMA(10, 0, 1) : AIC = 5402.080377265192
ARIMA(10, 0, 2) : AIC = 5400.444867568017
```

```
ARIMA(10, 0, 3) : AIC = 5398.374464575117
ARIMA(10, 0, 4) : AIC = 5390.829398326914
ARIMA(10, 0, 5) : AIC = 5393.408421374298
ARIMA(10, 0, 6) : AIC = 5398.436256174442
ARIMA(10, 0, 7) : AIC = 5393.86791008232
ARIMA(10, 0, 8) : AIC = 5398.716093066838
ARIMA(10, 0, 9) : AIC = 5400.682441467256
ARIMA(10, 0, 10) : AIC = 5401.7544151966185
ARIMA(10, 0, 11) : AIC = 5404.7996917978035
ARIMA(10, 0, 12) : AIC = 5405.061742544686
ARIMA(10, 0, 13) : AIC = 5396.902239550642
ARIMA(10, 0, 14) : AIC = 5399.826581722281
ARIMA(11, 0, 1) : AIC = 5404.079173596253
ARIMA(11, 0, 2) : AIC = 5399.870689617061
ARIMA(11, 0, 3) : AIC = 5399.979936451011
ARIMA(11, 0, 4) : AIC = 5392.676654943945
ARIMA(11, 0, 5) : AIC = 5395.2017435721955
ARIMA(11, 0, 6) : AIC = 5397.897032776667
ARIMA(11, 0, 7) : AIC = 5397.960135470588
ARIMA(11, 0, 8) : AIC = 5401.14267563801
ARIMA(11, 0, 9) : AIC = 5404.120865181905
ARIMA(11, 0, 10) : AIC = 5404.649489205936
ARIMA(11, 0, 11) : AIC = 5419.8356786522045
ARIMA(11, 0, 12) : AIC = 5406.149346825049
ARIMA(11, 0, 13) : AIC = 5411.243895880277
ARIMA(11, 0, 14) : AIC = 5400.840581201006
ARIMA(12, 0, 1) : AIC = 5396.879784635505
ARIMA(12, 0, 2) : AIC = 5401.868274305048
ARIMA(12, 0, 3) : AIC = 5401.493622139562
ARIMA(12, 0, 4) : AIC = 5395.257109020221
ARIMA(12, 0, 5) : AIC = 5397.082365723956
ARIMA(12, 0, 6) : AIC = 5401.706640794462
ARIMA(12, 0, 7) : AIC = 5396.091895418908
ARIMA(12, 0, 8) : AIC = 5415.354451530724
ARIMA(12, 0, 9) : AIC = 5403.964852502593
ARIMA(12, 0, 10) : AIC = 5404.899922601615
ARIMA(12, 0, 11) : AIC = 5419.32619468408
ARIMA(12, 0, 12) : AIC = 5403.153121313948
ARIMA(12, 0, 13) : AIC = 5403.027366166516
ARIMA(12, 0, 14) : AIC = 5402.881756845849
ARIMA(13, 0, 1) : AIC = 5403.545844438279
ARIMA(13, 0, 2) : AIC = 5401.49906971969
ARIMA(13, 0, 3) : AIC = 5399.446363427149
ARIMA(13, 0, 4) : AIC = 5397.06682176508
ARIMA(13, 0, 5) : AIC = 5399.635837241296
ARIMA(13, 0, 6) : AIC = 5402.126102380351
ARIMA(13, 0, 7) : AIC = 5397.296602427205
ARIMA(13, 0, 8) : AIC = 5402.559013827829
ARIMA(13, 0, 9) : AIC = 5405.924457814676
ARIMA(13, 0, 10) : AIC = 5407.582702050658
ARIMA(13, 0, 11) : AIC = 5409.518720616096
ARIMA(13, 0, 12) : AIC = 5407.666277647457
ARIMA(13, 0, 13) : AIC = 5409.331550413796
ARIMA(13, 0, 14) : AIC = 5404.84766351795
ARIMA(14, 0, 1) : AIC = 5404.780132348459
ARIMA(14, 0, 2) : AIC = 5402.81819110992
ARIMA(14, 0, 3) : AIC = 5404.742052922873
ARIMA(14, 0, 4) : AIC = 5400.973832916306
ARIMA(14, 0, 5) : AIC = 5400.7325787648715
ARIMA(14, 0, 6) : AIC = 5405.922767816388
ARIMA(14, 0, 7) : AIC = 5401.361873813972
ARIMA(14, 0, 8) : AIC = 5404.632041836703
ARIMA(14, 0, 9) : AIC = 5407.123364320816
ARIMA(14, 0, 10) : AIC = 5409.46811485563
ARIMA(14, 0, 11) : AIC = 5423.361861390662
ARIMA(14, 0, 12) : AIC = 5412.483109112428
ARIMA(14, 0, 13) : AIC = 5413.2632928699195
ARIMA(14, 0, 14) : AIC = 5409.54329416076
```

In [23]:

```
# 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 5388.684198923897 for ARMA(7, 9)
```

After some hassle I got the best ARMA model as ARMA(7,9) based on the lowest AIC.

In [24]:

```
# 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:                 1484
Model:                 ARIMA(7, 0, 9)   Log Likelihood               -2676.342
Date:                Thu, 02 May 2024   AIC                           5388.684
Time:                        03:29:25   BIC                           5484.129
Sample:                             0   HQIC                          5424.260
                                - 1484
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0612      0.016      3.788      0.000       0.030       0.093
ar.L1         -0.2166      0.205     -1.059      0.290      -0.617       0.184
ar.L2         -0.0020      0.187     -0.011      0.992      -0.369       0.365
ar.L3         -0.1051      0.176     -0.598      0.550      -0.450       0.240
ar.L4          0.5497      0.098      5.633      0.000       0.358       0.741
ar.L5          0.2036      0.172      1.186      0.236      -0.133       0.540
ar.L6         -0.1629      0.176     -0.928      0.354      -0.507       0.181
ar.L7          0.4698      0.176      2.664      0.008       0.124       0.815
ma.L1          0.1667      0.205      0.813      0.416      -0.235       0.569
ma.L2         -0.0113      0.181     -0.062      0.950      -0.367       0.344
ma.L3          0.1160      0.172      0.673      0.501      -0.222       0.454
ma.L4         -0.5512      0.089     -6.186      0.000      -0.726      -0.377
ma.L5         -0.1791      0.170     -1.051      0.293      -0.513       0.155
ma.L6          0.1851      0.174      1.065      0.287      -0.156       0.526
ma.L7         -0.5086      0.179     -2.845      0.004      -0.859      -0.158
ma.L8         -0.0173      0.028     -0.617      0.537      -0.072       0.038
ma.L9         -0.0976      0.027     -3.626      0.000      -0.150      -0.045
sigma2         2.1566      0.050     43.224      0.000       2.059       2.254
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):              1360.09
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               0.86   Skew:                             0.10
Prob(H) (two-sided):                  0.09   Kurtosis:                         7.69
===================================================================================

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

Residual Analysis

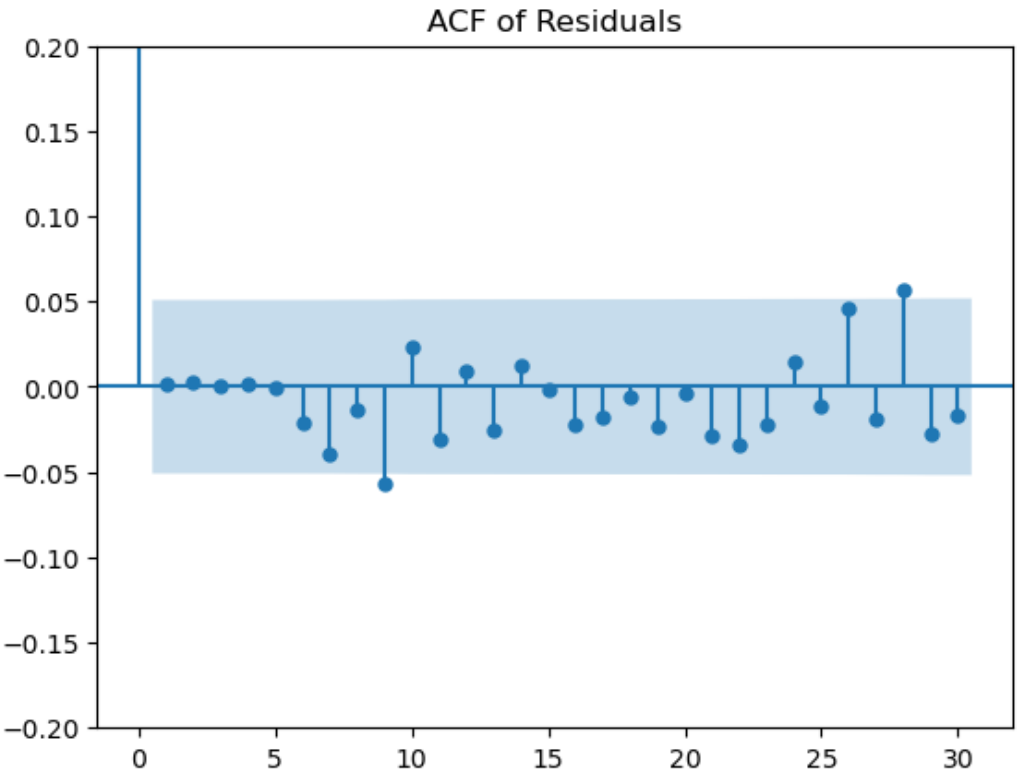
In [25]:

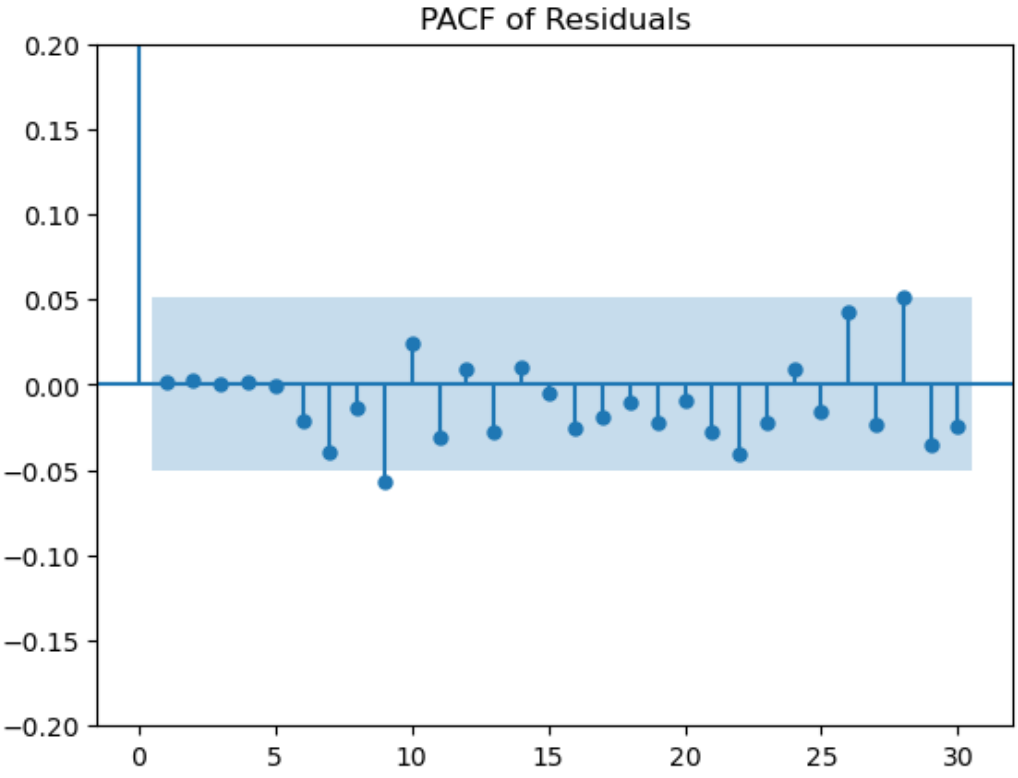
```
# ACF and PACF plots for the residuals
plot_acf(residuals, lags = 30);
plt.ylim(-0.2,0.2)
plt.title('ACF of Residuals')


plot_pacf(residuals, lags=30);
```

```
plt.ylim(-0.2,0.2)
plt.title('PACF of Residuals')
```

Out[25]:

```
Text(0.5, 1.0, 'PACF of Residuals')
```



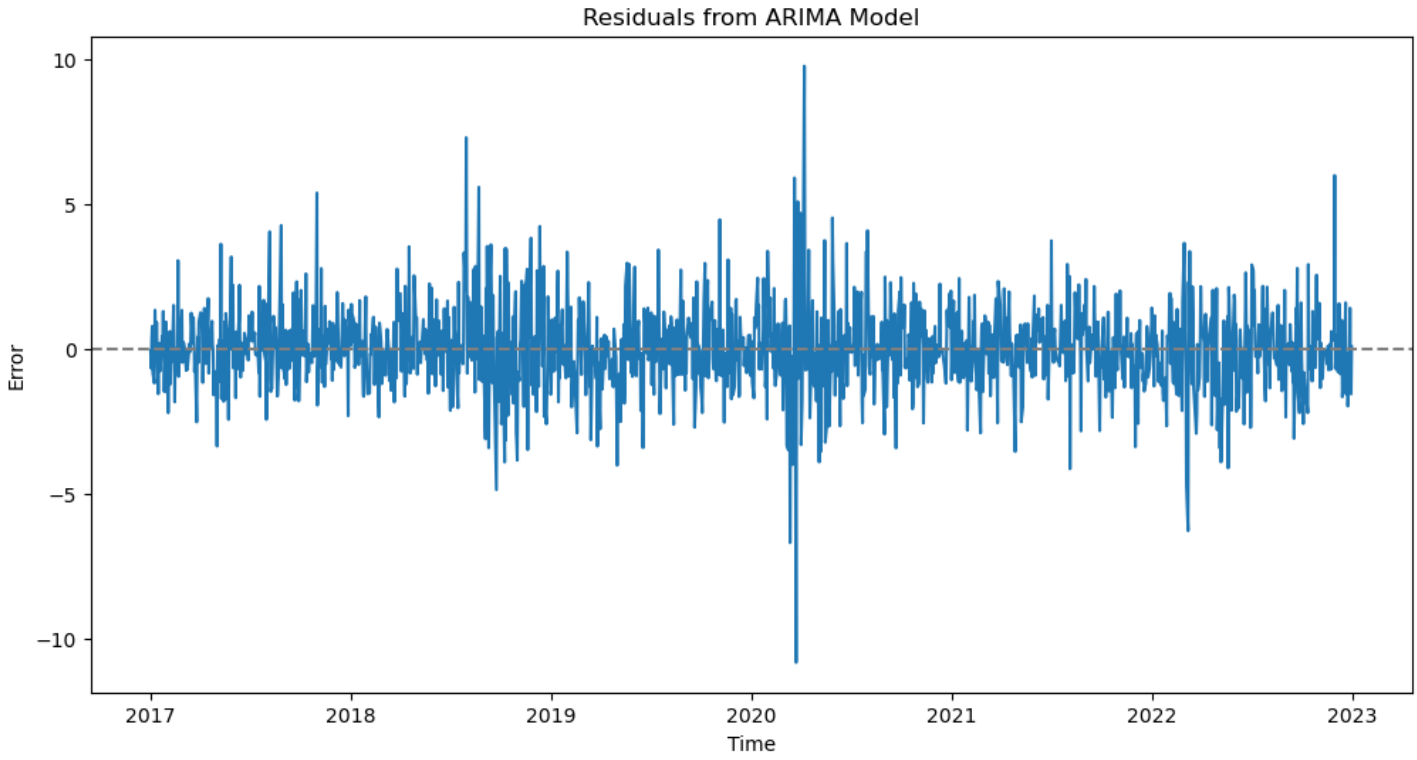


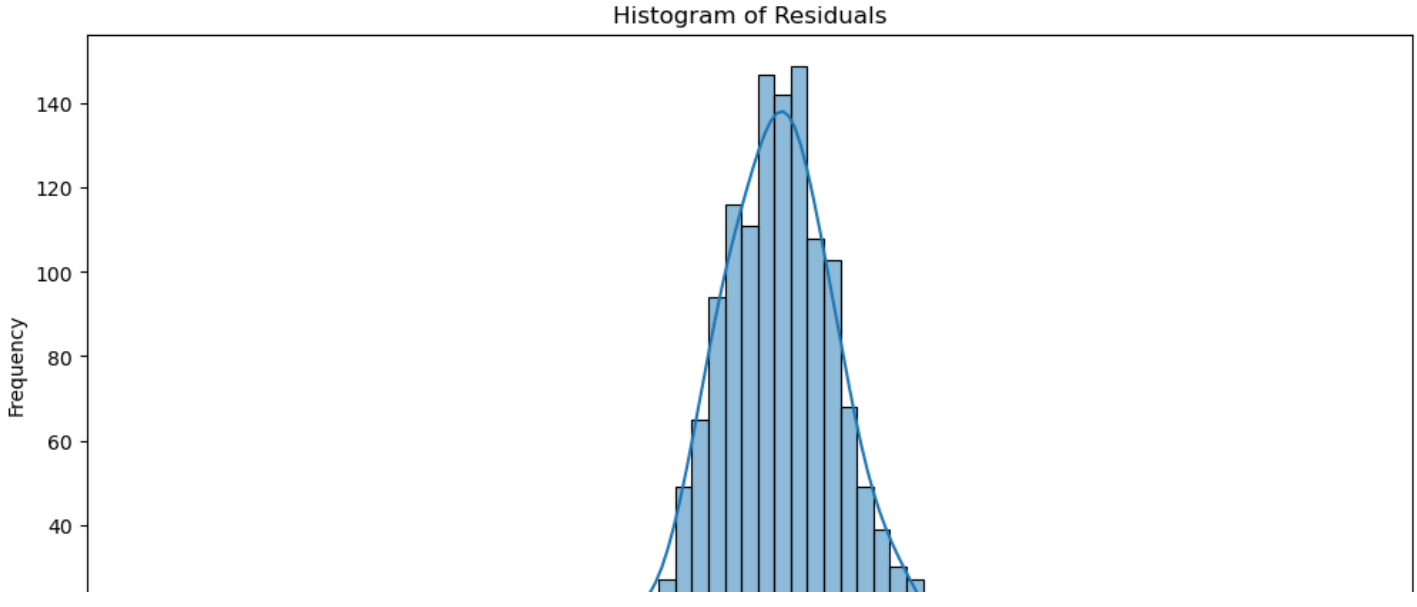
Almost all lags are inside confidence interval

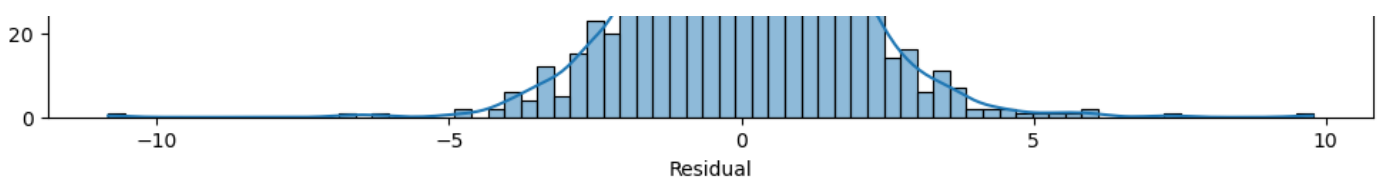
In [26]:

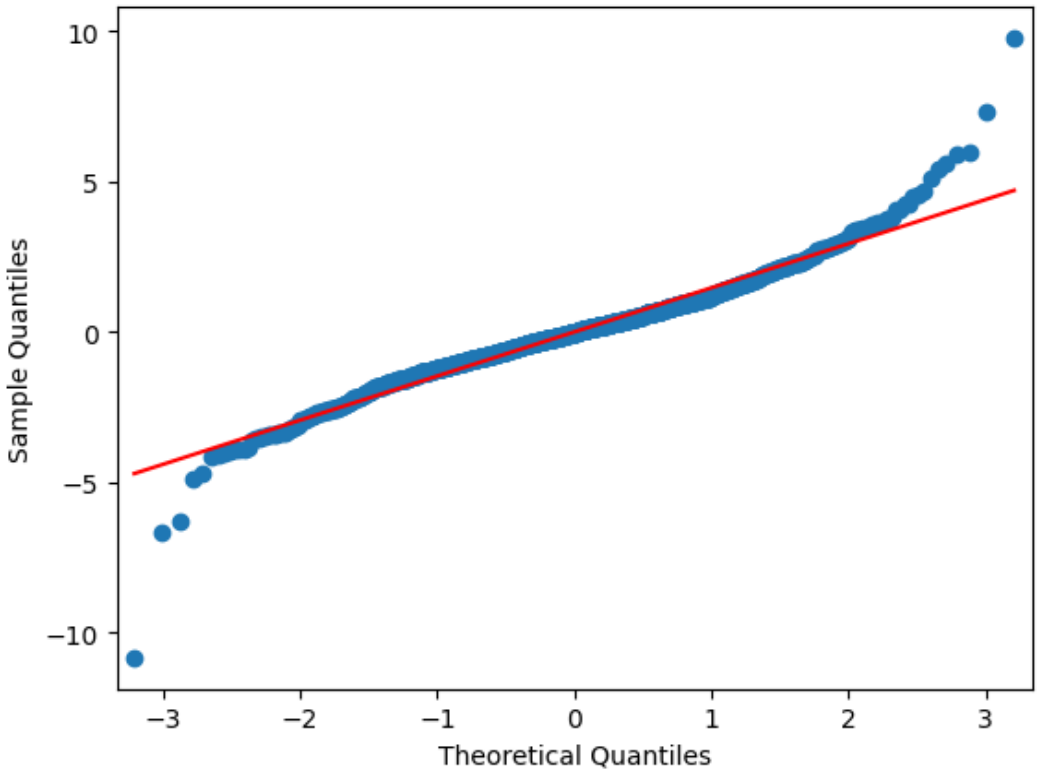
```
import statsmodels.api as sm
from scipy import stats
import matplotlib.pyplot as plt


new_order = (7, 0, 9)
```

```
model_refined = ARIMA(ts, order=new_order)
results_refined = model_refined.fit()

# Model Diagnostics
residuals = results_refined.resid

# Plot residual errors
plt.figure(figsize=(12, 6))
plt.plot(residuals)
plt.title('Residuals from ARIMA Model')
plt.xlabel('Time')
plt.ylabel('Error')
plt.axhline(0, linestyle='--', color='grey')
plt.show()

# Histogram of the residuals
plt.figure(figsize=(12, 6))
sns.histplot(residuals, kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.show()

# Q-Q plot
sm.qqplot(residuals, line='s')
plt.show()
```









From the residual plot I can say that there are some periods where the residuals exhibit clustering, indicating phases of higher or lower volatility and also there are a few noticeable spikes, particularly around 2020 indicating outliers.

The Histogram almost looks normaly distributed.

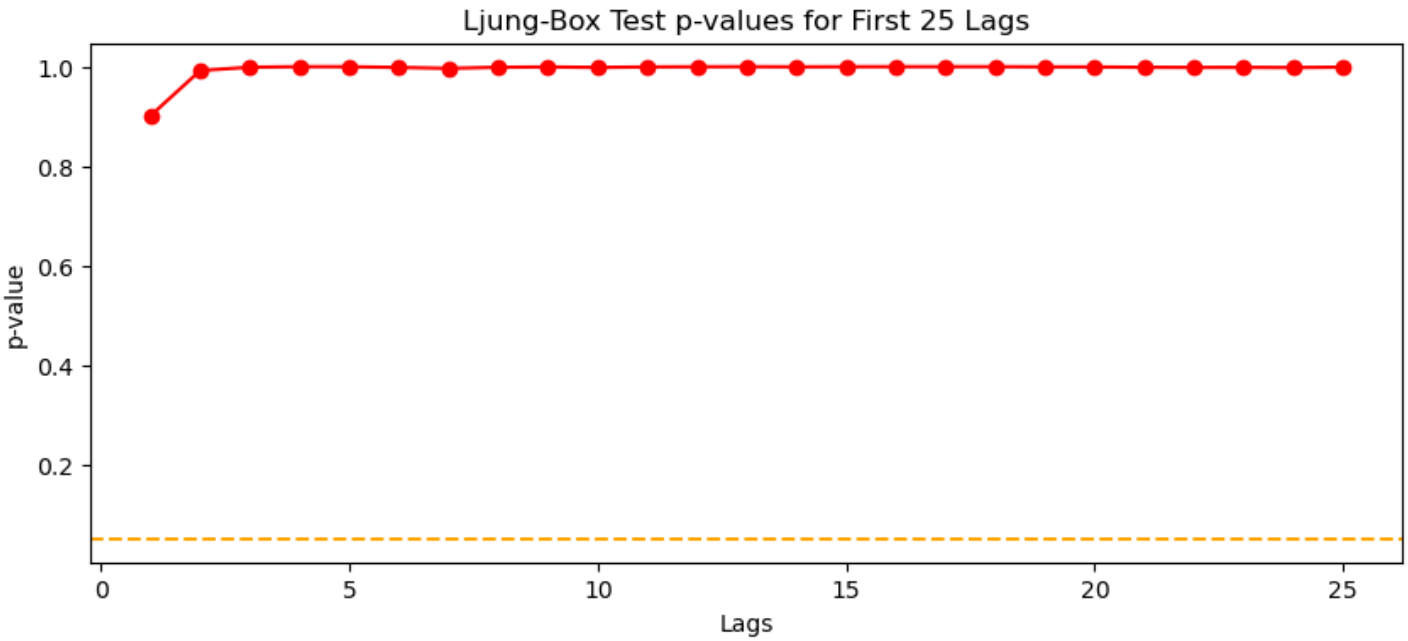
For the Q-Q plot the residuals roughly follow the red line in the middle range but deviate at the tails.This indicates that the residuals are not perfectly normally distributed especially in the low and high values.The points deviate significantly from the line at both ends which means the presence of fat tails.

In [27]:

```
from statsmodels.stats.diagnostic import acorr_ljungbox
import matplotlib.pyplot as plt

ljung_box_results = acorr_ljungbox(residuals, lags=25, return_df=True)

# Print the results
print(ljung_box_results)

# Plotting the Ljung-Box test p-values
plt.figure(figsize=(10, 4))
plt.plot(ljung_box_results['lb_pvalue'], marker='o', linestyle='-', color='red')
plt.title('Ljung-Box Test p-values for First 25 Lags')
plt.xlabel('Lags')
plt.ylabel('p-value')
plt.axhline(y=0.05, color='orange', linestyle='--')  # significance line at p = 0.05
plt.show()
```

```
     lb_stat  lb_pvalue
1   0.015407   0.901216
2   0.015550   0.992255
3   0.026794   0.998843
4   0.027181   0.999908
5   0.104061   0.999821
```

```
6   0.445925   0.998436
7   0.881197   0.996524
8   0.908812   0.998761
9   1.008531   0.999418
10  1.637731   0.998438
11  1.641909   0.999410
12  1.750939   0.999703
13  1.756485   0.999892
14  2.663878   0.999535
15  2.792380   0.999743
16  2.836102   0.999884
17  3.186230   0.999893
18  3.695923   0.999867
19  4.876459   0.999526
20  5.566965   0.999369
21  6.641558   0.998746
22  7.389885   0.998444
23  7.880760   0.998556
24  8.676179   0.998210
25  8.682269   0.998967
```



The Ljung-Box test results show that all the p-values are well above the 0.05 significance level across all 25 lags, indicating that there is no autocorrelation that can be seen amongst the residuals.

In [28]:

```
from scipy import stats

# Shapiro-Wilk test for normality
shapiro_test = stats.shapiro(residuals)

# Print results and hypothesis information
print(f'Shapiro-Wilk test statistic: {shapiro_test[0]:.4f}, p-value: {shapiro_test[1]:.4f
}')

# Hypotheses
print("\nHypotheses:")
print("Null Hypothesis (H0): The data is normally distributed.")
print("Alternative Hypothesis (Ha): The data is not normally distributed.")

# Interpretation
if shapiro_test[1] < 0.05:
    print("\nReject the null hypothesis (H0), suggesting the data is not normally distrib
uted.")
else:
    print("\nFailed to reject the null hypothesis (H0), suggesting the data is normally d
```

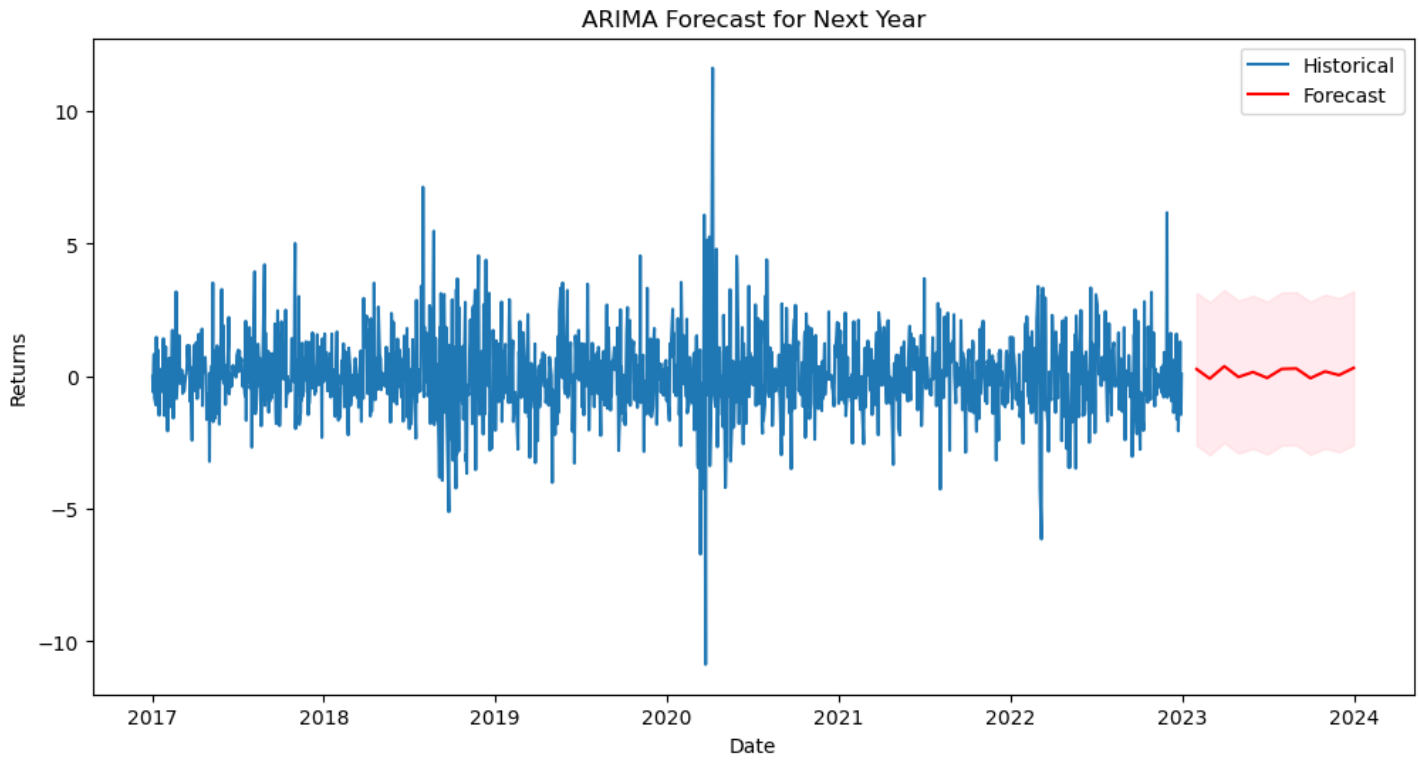
```
istributed.")
```

```
Shapiro-Wilk test statistic: 0.9601, p-value: 0.0000

Hypotheses:
Null Hypothesis (H0): The data is normally distributed.
Alternative Hypothesis (Ha): The data is not normally distributed.

Reject the null hypothesis (H0), suggesting the data is not normally distributed.
```

Forecasting

In [29]:

```
# Forecast the next 12 periods ahead in the future
forecast_steps = 12
forecast = results_refined.get_forecast(steps=forecast_steps)
forecast_mean = forecast.predicted_mean
forecast_conf_int = forecast.conf_int()


last_date = ts.index[-1]

forecast_dates = pd.date_range(start=last_date, periods=forecast_steps + 1, freq='M')[1:
]

# Assign the new date range as the index of the forecasted data
forecast_mean.index = forecast_dates
forecast_conf_int.index = forecast_dates

# Plot the forecast alongside the historical data
plt.figure(figsize=(12, 6))
plt.plot(ts.index, ts['Returns'], label='Historical')
plt.plot(forecast_mean.index, forecast_mean, color='red', label='Forecast')
plt.fill_between(forecast_conf_int.index,
                 forecast_conf_int.iloc[:, 0],
                 forecast_conf_int.iloc[:, 1], color='pink', alpha=0.3)
plt.title('ARIMA Forecast for Next Year')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.legend()
plt.show()
```



After performing forecasts using the ARIMA model, I plan to further enhance our analysis by applying a GARCH model to the residuals. This approach will allow us to investigate any potential volatility clustering in the time series data, providing a more nuanced understanding of variability in the model's errors over time.

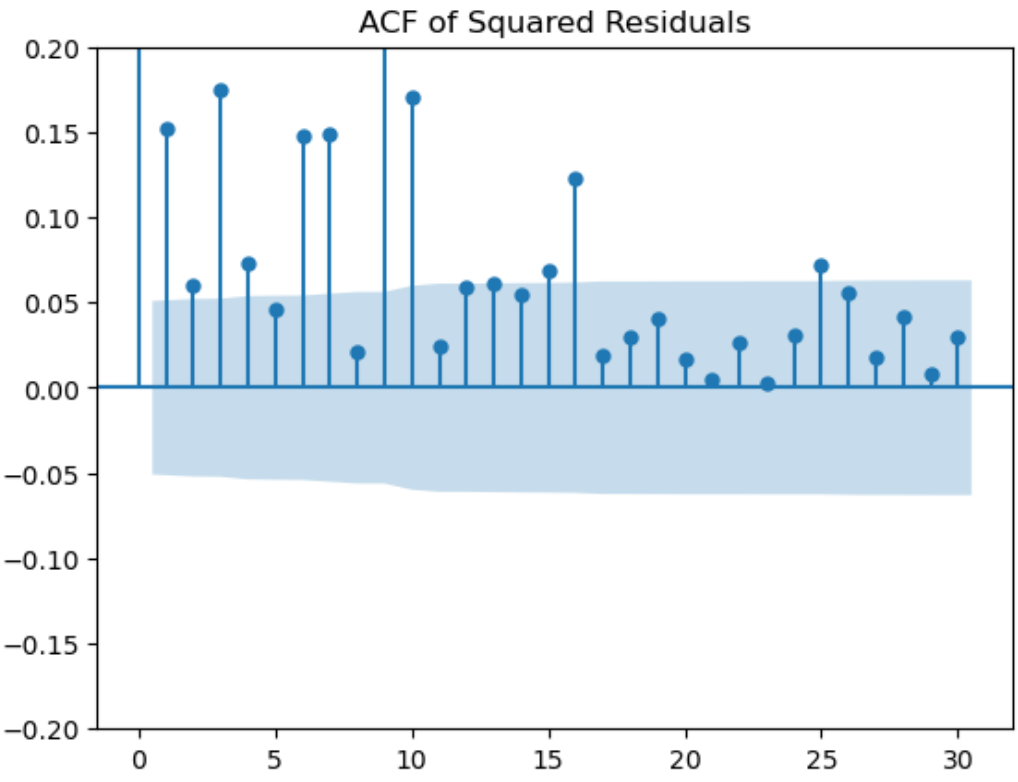
GARCH model

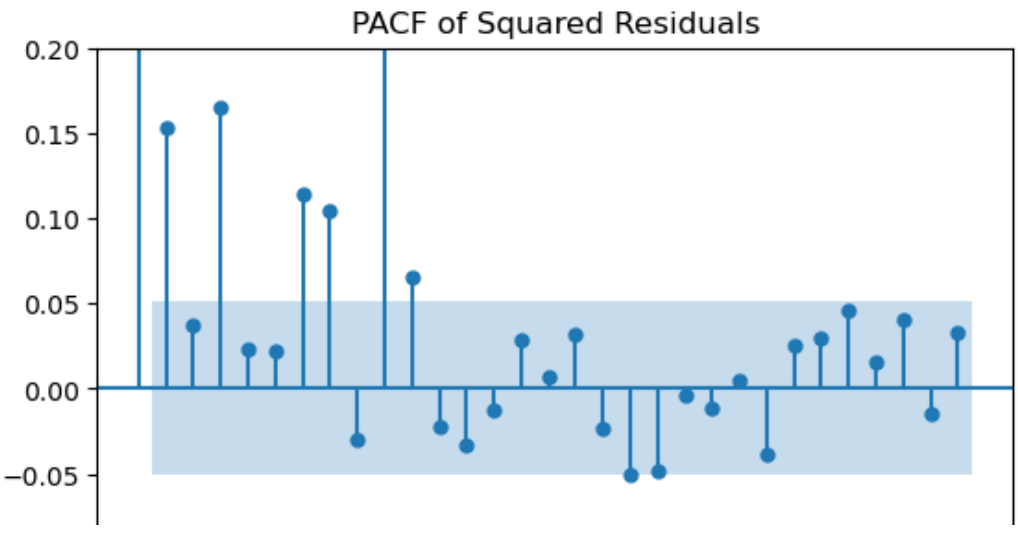
In [30]:

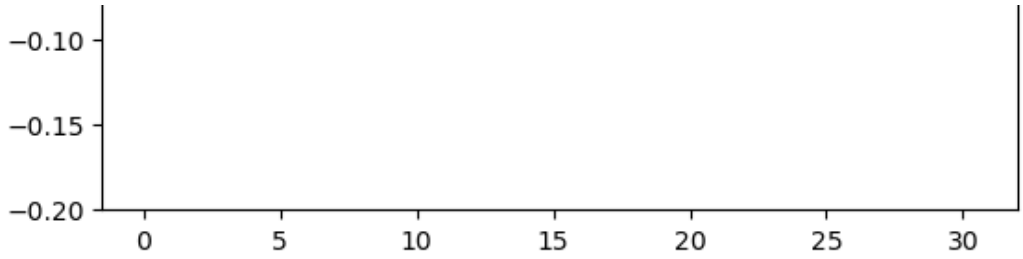
```
from statsmodels.graphics.tsaplots import plot_acf


squared_residuals = residuals**2

plot_acf(squared_residuals, alpha=0.05, lags = 30);
plt.ylim(-0.2,0.2)
plt.title('ACF of Squared Residuals')


plot_pacf(squared_residuals, alpha=0.05, lags = 30);
plt.ylim(-0.2,0.2)
plt.title('PACF of Squared Residuals')
```

Out[30]:

```
Text(0.5, 1.0, 'PACF of Squared Residuals')
```







Best GARCH model

In [31]:

```
from arch import arch_model
import numpy as np

squared_residuals = residuals ** 2  # Using squared residuals

# Define the range of p and q values to explore
p_range = range(1, 6)
q_range = range(1, 6)

best_aic = np.inf
best_order = None
best_model = None
aic_values = []  # List to store AIC values and corresponding orders

# Grid search over different combinations of p and q for the GARCH model
for p in p_range:
    for q in q_range:
        try:
            # Specify the GARCH model using the squared residuals
            model = arch_model(squared_residuals, mean='Zero', vol='GARCH', p=p, q=q)
            # Fit the model
            model_fit = model.fit(disp='off', show_warning=False)
            # Save the AIC and order
            aic_values.append((p, q, model_fit.aic))
            # Compare AIC
            if model_fit.aic < best_aic:
                best_aic = model_fit.aic
                best_order = (p, q)
                best_model = model_fit
        except Exception as e:
            print(f'Error fitting GARCH({p},{q}):', str(e))

# Output all AIC values with their corresponding models
for order in aic_values:
    print(f'GARCH({order[0]},{order[1]}) : AIC = {order[2]}')

# Output the results of the grid search
if best_order:
    print(f'\nBest GARCH model order: p={best_order[0]}, q={best_order[1]} with AIC: {bes
t_aic}')
    print(best_model.summary())
else:
    print("\nNo suitable GARCH model was found.")
```

```
GARCH(1,1) : AIC = 8406.712273976093
GARCH(1,2) : AIC = 8408.71219033534
GARCH(1,3) : AIC = 8409.012994147526
GARCH(1,4) : AIC = 8402.184983991296
GARCH(1,5) : AIC = 8397.95020857262
GARCH(2,1) : AIC = 8408.712273423294
GARCH(2,2) : AIC = 8410.662853402247
GARCH(2,3) : AIC = 8406.185959364768
GARCH(2,4) : AIC = 8390.969075917903
GARCH(2,5) : AIC = 8392.27674980971
GARCH(3,1) : AIC = 8410.712273486826
GARCH(3,2) : AIC = 8373.673070934725
GARCH(3,3) : AIC = 8375.547143550306
```

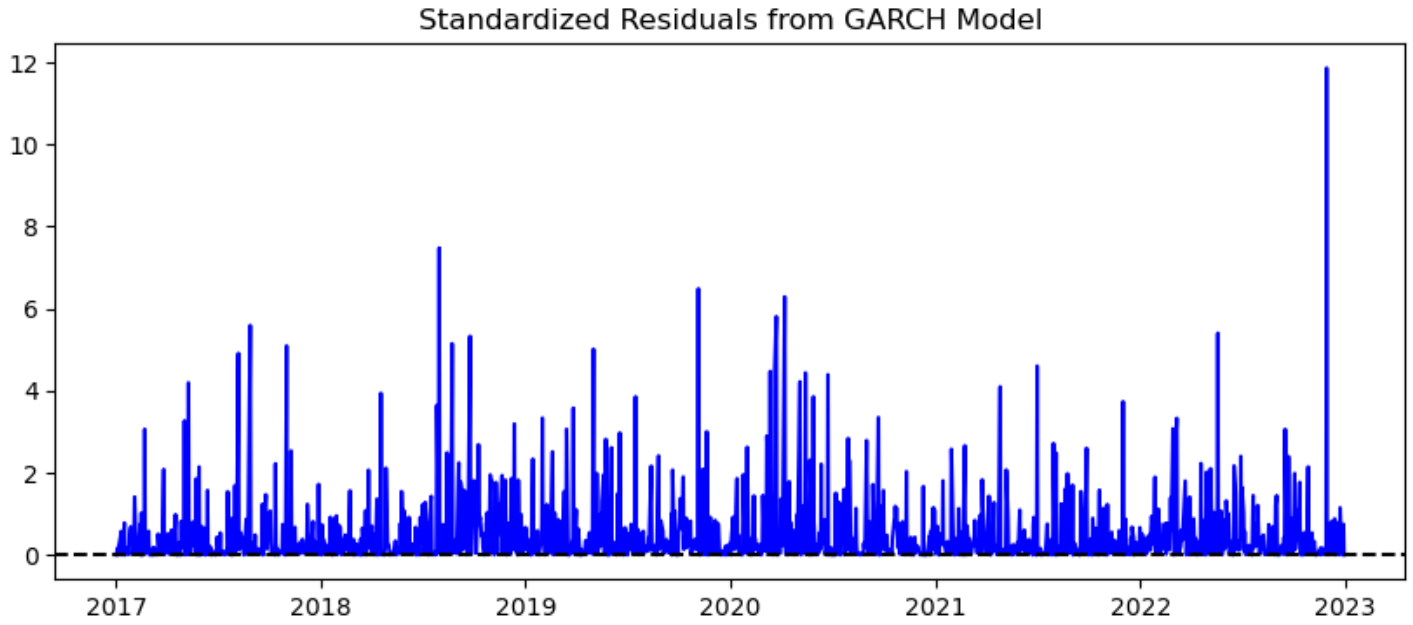
```
GARCH(3,4) : AIC = 8382.546269503458
GARCH(3,5) : AIC = 8384.546269424083
GARCH(4,1) : AIC = 8373.67307104133
GARCH(4,2) : AIC = 8375.67307063462
GARCH(4,3) : AIC = 8377.547143435753
GARCH(4,4) : AIC = 8384.54626913178
GARCH(4,5) : AIC = 8381.547145275976
GARCH(5,1) : AIC = 8414.71227355095
GARCH(5,2) : AIC = 8377.673070525067
GARCH(5,3) : AIC = 8379.439969170055
GARCH(5,4) : AIC = 8382.624636696317
GARCH(5,5) : AIC = 8384.624638253168

Best GARCH model order: p=3, q=2 with AIC: 8373.673070934725
                      Zero Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    None   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.001
Vol Model:                      GARCH   Log-Likelihood:               -4180.84
Distribution:                  Normal   AIC:                           8373.67
Method:            Maximum Likelihood   BIC:                           8405.49
                                        No. Observations:                 1484
Date:                Thu, May 02 2024   Df Residuals:                     1484
Time:                        03:30:00   Df Model:                            0
                            Volatility Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
omega          9.2001      1.661      5.537  3.070e-08      [  5.944, 12.457]
alpha[1]       0.2724      0.129      2.116  3.432e-02    [2.012e-02,  0.525]
alpha[2]       0.2974      0.181      1.645  9.999e-02   [-5.697e-02,  0.652]
alpha[3]       0.3759      0.194      1.933  5.325e-02   [-5.270e-03,  0.757]
beta[1]    2.1844e-10  4.794e-02  4.557e-09      1.000 [-9.395e-02,9.395e-02]
beta[2]    2.1370e-10  6.082e-02  3.514e-09      1.000      [ -0.119,  0.119]
==============================================================================

Covariance estimator: robust
```

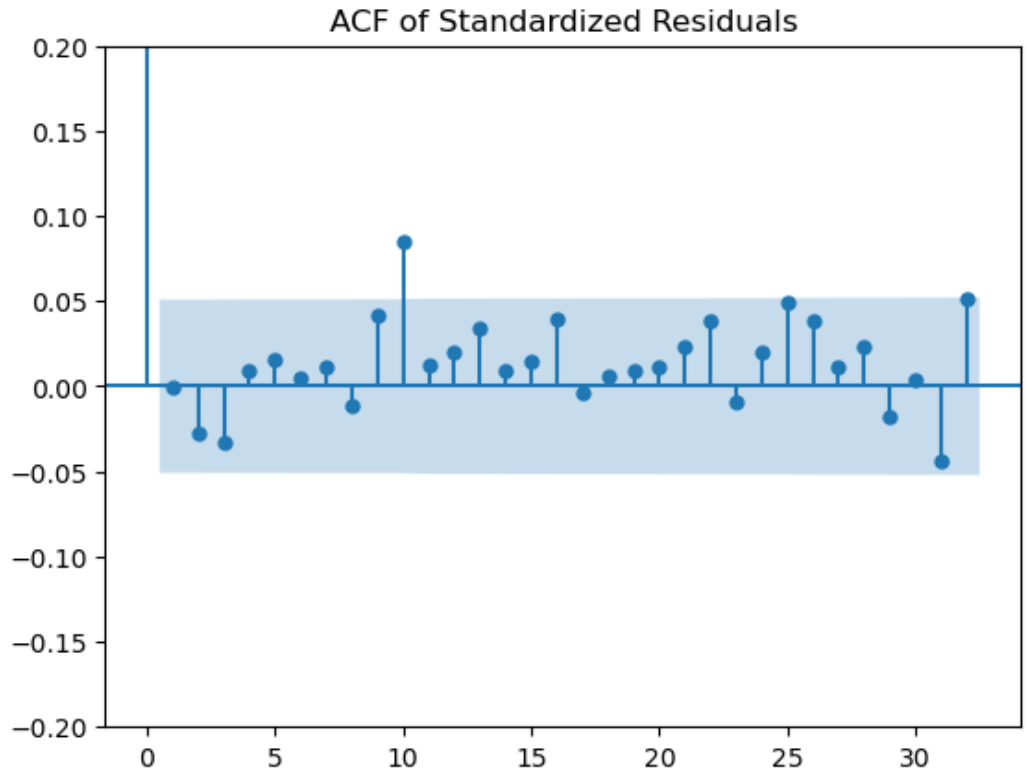
Residual Analysis using GARCH

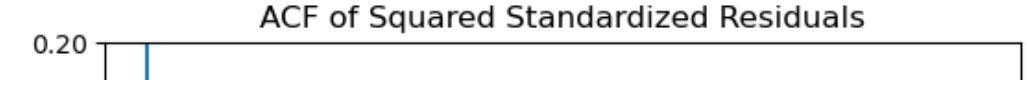
In [43]:

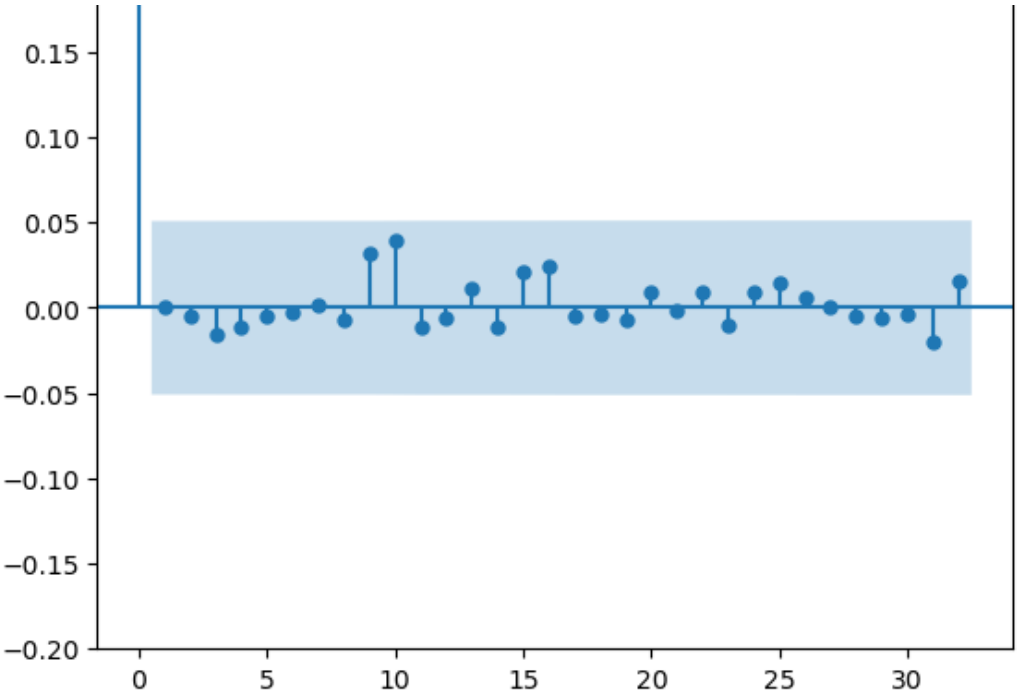
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf

# Get standardized residuals from the GARCH model
std_resid = best_model.resid / best_model.conditional_volatility

# Get squared standardized residuals
squared_std_resid = std_resid**2

# Plot standardized residuals
plt.figure(figsize=(10, 4))
plt.plot(std_resid, color='blue')
plt.title('Standardized Residuals from GARCH Model')
plt.axhline(0, linestyle='--', color='black')
plt.show()

# Plot ACF of standardized residuals
plot_acf(std_resid, alpha=0.05, title='ACF of Standardized Residuals')
plt.ylim(-0.2,0.2)
plt.show()

# Plot ACF of squared standardized residuals
plot_acf(squared_std_resid, alpha=0.05, title='ACF of Squared Standardized Residuals')
plt.ylim(-0.2,0.2)
plt.show()

# Perform Shapiro-Wilk test of normality on the standardized residuals
from scipy import stats
```

```
# Shapiro-Wilk test for normality
shapiro_test = stats.shapiro(std_resid)

# Print results and hypothesis information
print(f'Shapiro-Wilk test statistic: {shapiro_test[0]:.4f}, p-value: {shapiro_test[1]:.4f
}')

# Hypotheses
print("\nHypotheses:")
print("Null Hypothesis (H0): The data is normally distributed.")
print("Alternative Hypothesis (Ha): The data is not normally distributed.")

# Interpretation
if shapiro_test[1] < 0.05:
    print("\nReject the null hypothesis (H0), suggesting the data is not normally distrib
uted.")
else:
    print("\nFailed to reject the null hypothesis (H0), suggesting the data is normally d
istributed.")
```







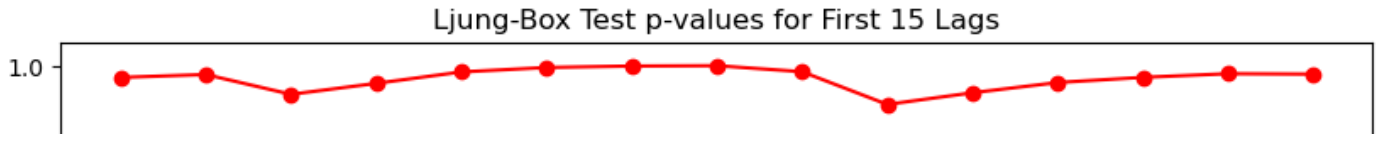


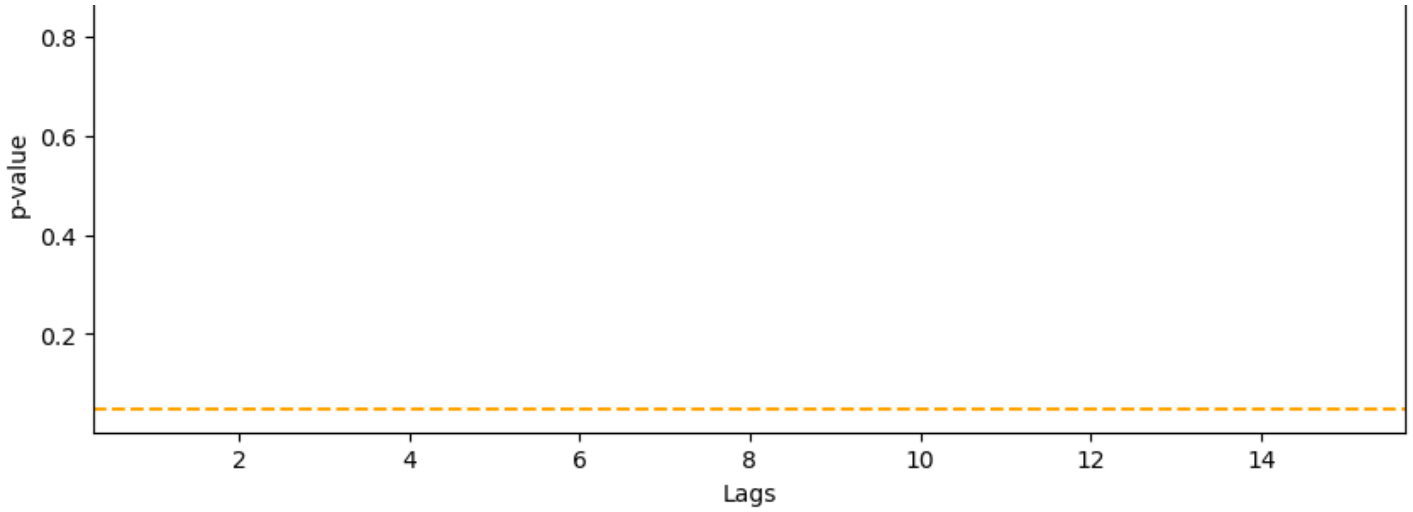

```
Shapiro-Wilk test statistic: 0.5532, p-value: 0.0000

Hypotheses:
Null Hypothesis (H0): The data is normally distributed.
Alternative Hypothesis (Ha): The data is not normally distributed.

Reject the null hypothesis (H0), suggesting the data is not normally distributed.
```

In [44]:

```
from statsmodels.stats.diagnostic import acorr_ljungbox
import matplotlib.pyplot as plt

ljung_box_results = acorr_ljungbox(squared_std_resid, lags=15, return_df=True)

# Print the results
print(ljung_box_results)

# Plotting the Ljung-Box test p-values
plt.figure(figsize=(10, 4))
plt.plot(ljung_box_results['lb_pvalue'], marker='o', linestyle='-', color='red')
plt.title('Ljung-Box Test p-values for First 15 Lags')
plt.xlabel('Lags')
plt.ylabel('p-value')
plt.axhline(y=0.05, color='orange', linestyle='--')  # significance line at p = 0.05
plt.show()
```

```
     lb_stat  lb_pvalue
1   0.000921   0.975785
2   0.036925   0.981707
3   0.392294   0.941830
4   0.592917   0.963848
5   0.624372   0.986860
6   0.633985   0.995806
7   0.639435   0.998759
8   0.721465   0.999470
9   2.226329   0.987362
10  4.508122   0.921528
11  4.696556   0.944967
12  4.757892   0.965580
13  4.964263   0.975971
14  5.166501   0.983358
15  5.826536   0.982516
```



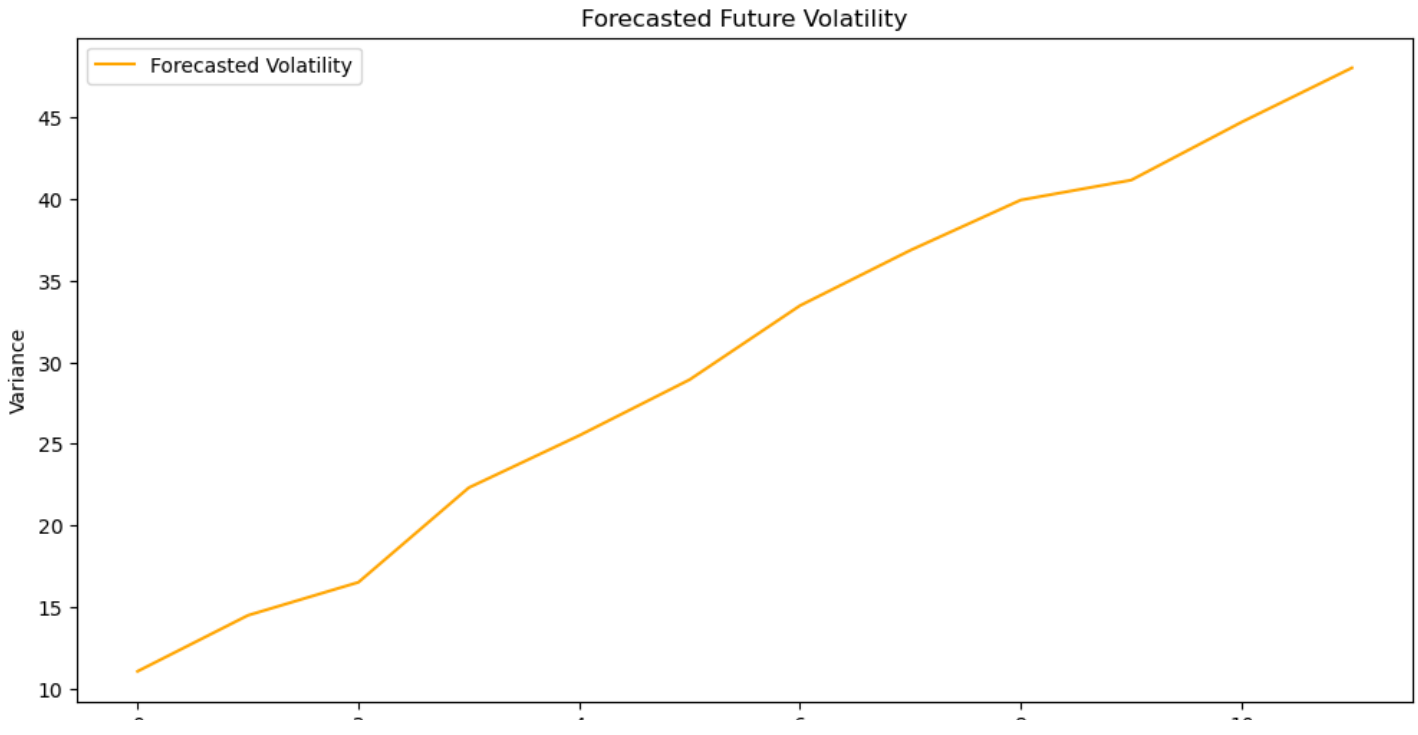


The Ljung-Box test results above suggest that there is no significant autocorrelation in the squared standardized residuals for the first 15 lags, as all p-values are well above the 0.05 significance level.

Forecasting the Volatility

In [37]:

```
# Forecast future volatility
forecasts = best_model.forecast(horizon=forecast_steps, start=None, method='simulation',
simulations=1000)

# Extract the forecasted variance
future_volatility = forecasts.variance[-1:]

# Plot the forecasted volatility
plt.figure(figsize=(12, 6))
plt.plot(future_volatility.values.flatten(), color='orange', label='Forecasted Volatility
')
plt.title('Forecasted Future Volatility')
plt.xlabel('Time')
plt.ylabel('Variance')
plt.legend()
plt.show()

# Display the last forecasted variance
print("Forecasted Variance:")
print(future_volatility)
```




```
Forecasted Variance:
                h.01       h.02       h.03       h.04       h.05       h.06  \
Date
2022-12-30  11.063933  14.490616  16.513971  22.308658  25.501871  28.922981

                h.07       h.08       h.09       h.10       h.11     h.12
Date
2022-12-30  33.458469  36.848685  39.925753  41.144098  44.696202  48.0197
```

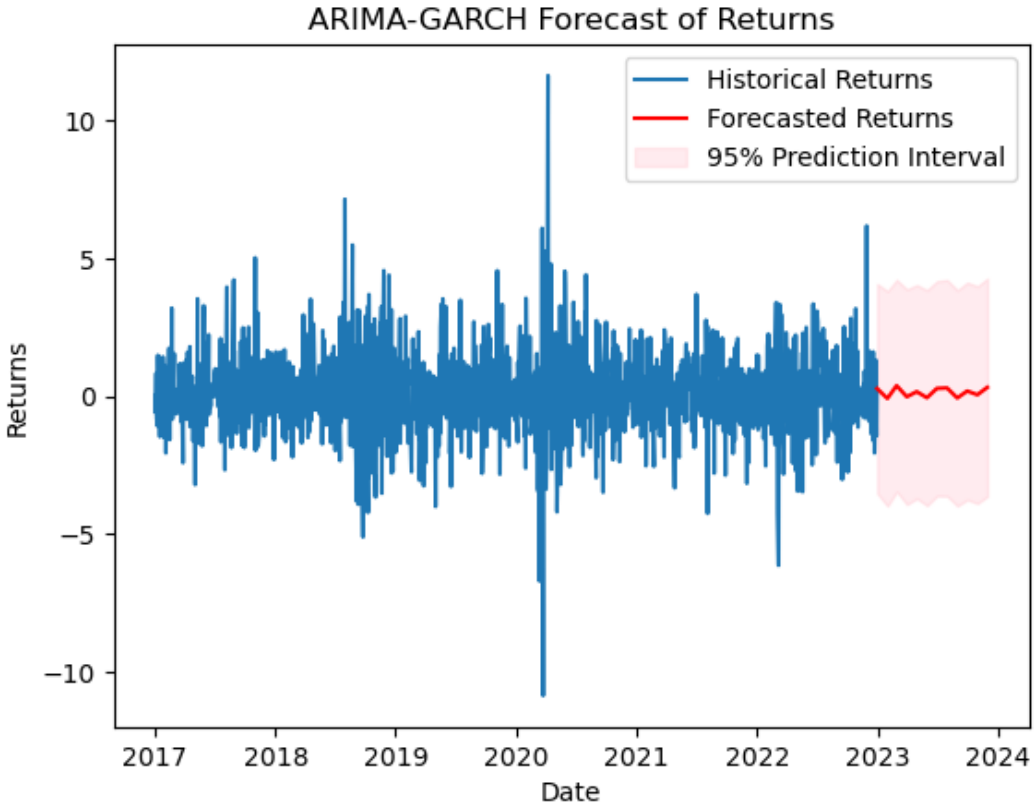
The forecast shows that the expected future changes or volatility in the data are likely to increase over time. This means we might see bigger ups and downs in the future. Understanding this helps us prepare for more unpredictable changes and manage risks better.

Forecating using ARIMA-GARCH

In [38]:

```
from arch import arch_model
import pandas as pd
import matplotlib.pyplot as plt

# Fit the ARIMA model
arima_model = ARIMA(ts, order=(7,0,9))
arima_results = arima_model.fit()

# Forecast future values with ARIMA
arima_forecast = arima_results.get_forecast(steps=forecast_steps)
forecast_mean = arima_forecast.predicted_mean
forecast_std = arima_forecast.se_mean

# Fit the GARCH model
garch_model = arch_model(arima_results.resid, p=3, q=2)
garch_results = garch_model.fit(update_freq=10)

# Forecast future volatility with GARCH
garch_forecast = garch_results.forecast(horizon=forecast_steps)
forecast_var = garch_forecast.variance.values[-1]

# Combine the ARIMA and GARCH forecasts
forecast_combined_var = forecast_std**2 + forecast_var

# Now, forecast_combined_var contains the forecasted variance for the ARIMA-GARCH model
lower_bound = forecast_mean - 1.96 * np.sqrt(forecast_combined_var)
upper_bound = forecast_mean + 1.96 * np.sqrt(forecast_combined_var)

# Generate future dates
future_dates = pd.date_range(start=ts.index[-1], periods=forecast_steps, freq='M')

# Plot the historical returns
plt.plot(ts.index, ts, label='Historical Returns')

# Plot the forecasted returns
plt.plot(future_dates, forecast_mean, color='red', label='Forecasted Returns')

# Plot the forecasted volatility (95% prediction interval)
plt.fill_between(future_dates, lower_bound, upper_bound, color='pink', alpha=0.3, label=
'95% Prediction Interval')

# Show the plot
plt.legend()
plt.title('ARIMA-GARCH Forecast of Returns')
plt.xlabel('Date')
plt.ylabel('Returns')
plt.show()
```

```
Iteration:     10,   Func. Count:     93,   Neg. LLF: 2591.1412996279223
Iteration:     20,   Func. Count:    178,   Neg. LLF: 2591.0733425361486
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2590.949167266409
            Iterations: 25
            Function evaluations: 221
            Gradient evaluations: 25
```



Conclusion

In this project, we utilized advanced forecasting techniques to analyze financial data, identifying the best GARCH model with parameters (3,2) for predicting future volatility. Simultaneously, we selected an ARMA model with parameters (7,9) for the returns data. By integrating these models into an ARIMA-GARCH framework, we effectively captured both the trends and variability in the data, offering a comprehensive view of the market dynamics.

This combined approach allowed us to generate detailed predictions that included confidence intervals, illustrating the potential future values and the associated uncertainties. The successful application of these techniques provides valuable insights for financial planning and risk management, demonstrating the critical role of appropriate model selection in understanding complex financial behaviors.

Introduction

In the realm of data analysis, accurately forecasting future values of time series data is critical for decision-making across various industries, from finance to weather prediction. My primary motivation for this project was to tackle the complexities associated with seasonal time series data which often exhibit patterns that repeat over time. By developing a robust model capable of accounting for these seasonal variations.

this project involves analyzing a dataset with evident seasonal trends, doing a thorough examination and transformation to achieve stationarity which is prerequisite for many time series modeling techniques. I started on this project to explore and implement advanced modeling techniques, particularly the Seasonal Autoregressive Integrated Moving Average (SARIMA) model, which is well-suited for handling such data. My goal is to identify the most effective SARIMA model parameters that could accurately capture the underlying patterns of the data, leading to improved forecast accuracy and insightful analysis.

Dataset

https://www.kaggle.com/datasets/vanvalkenberg/historicalweatherdataforindiancities/Chennai_1990_2022_Madra

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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("Chennai.csv")
df_temp.head()
```

Out[2]:

	time	tavg	tmin	tmax	prcp
0	01-01-1990	25.2	22.8	28.4	0.5
1	02-01-1990	24.9	21.7	29.1	0.0
2	03-01-1990	25.6	21.4	29.8	0.0
3	04-01-1990	25.7	NaN	28.7	0.0
4	05-01-1990	25.5	20.7	28.4	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	25.2
1	1990-01-02	24.9
2	1990-01-03	25.6
3	1990-01-04	25.7
4	1990-01-05	25.5

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	24.21
1	1990	2	26.83
2	1990	3	28.66
3	1990	4	30.38
4	1990	5	29.85

We will calculate the monthly average temperatures and use these figures for our analysis.

In [6]:

```
df.isna().sum()
```

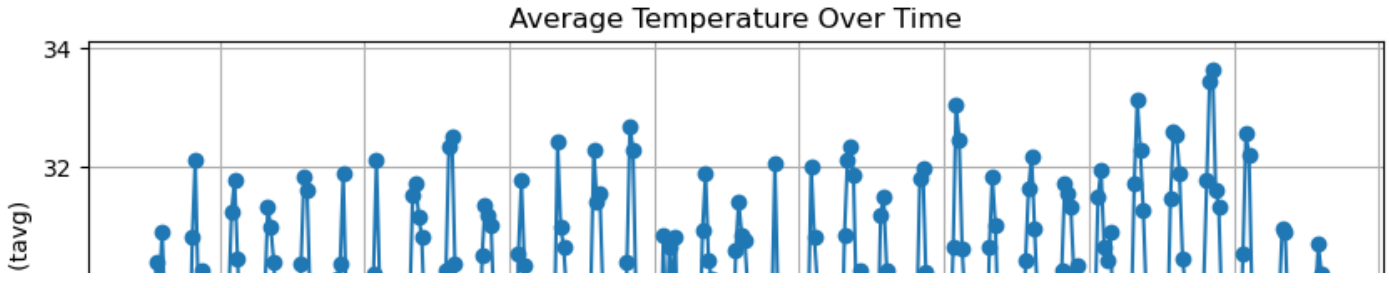
Out[6]:

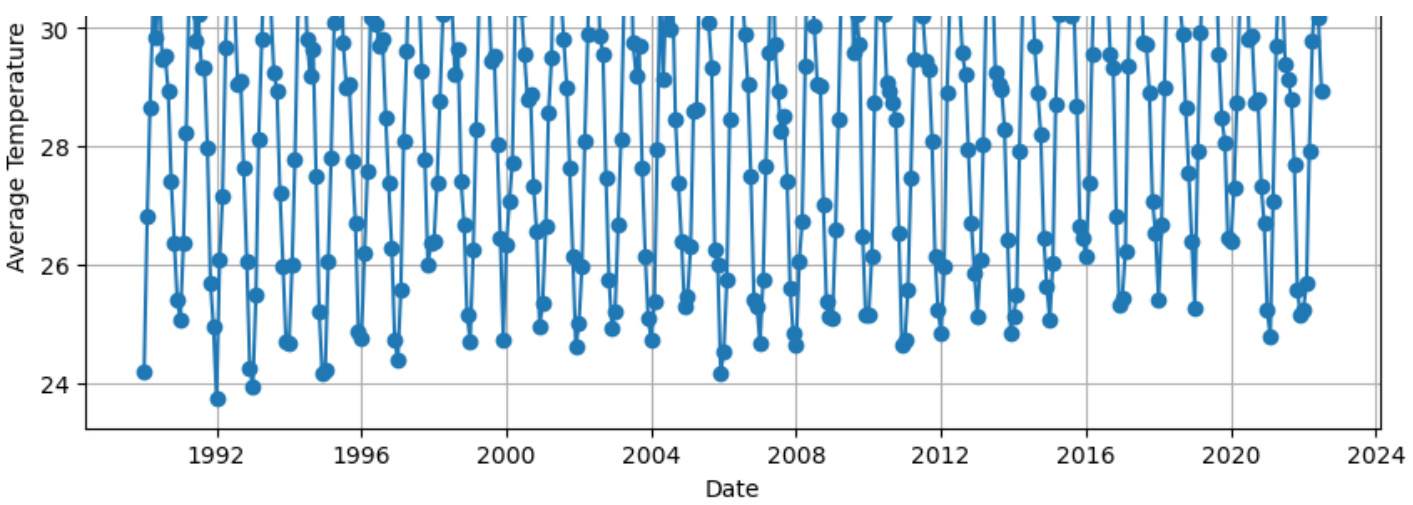
```
Year     0
Month    0
tavg     0
dtype: int64
```

Time Series plot by average temperature by month

In [7]:

```
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()
```





Visualizing the Time series plot

In [8]:

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

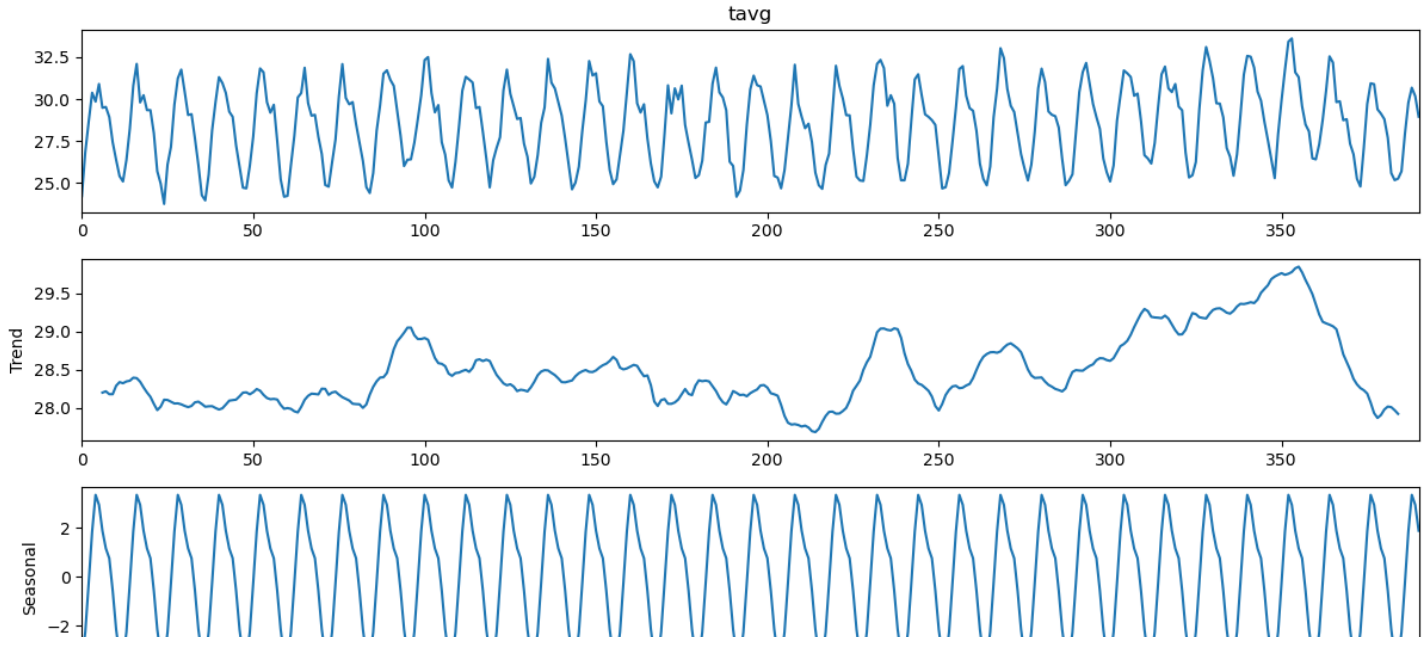
Out[8]:

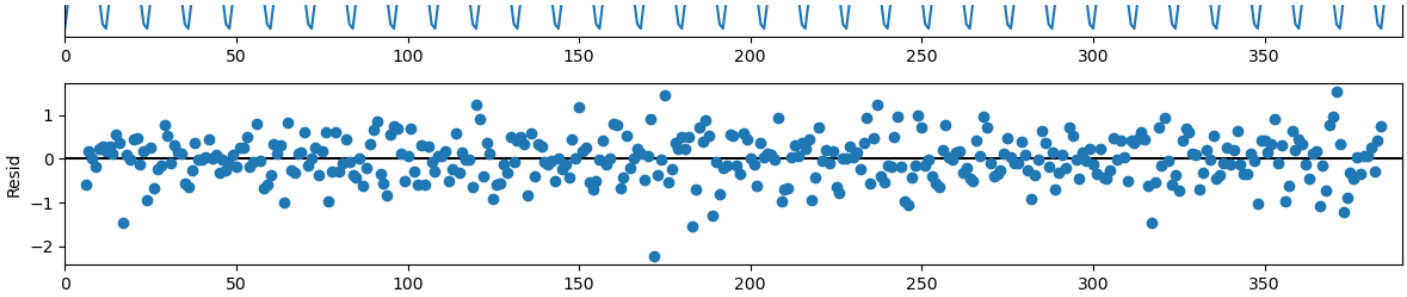
	Year	Month	tavg
0	**1990**	**1**	**24.21**
1	**1990**	**2**	**26.83**
2	**1990**	**3**	**28.66**
3	**1990**	**4**	**30.38**
4	**1990**	**5**	**29.85**

Sesonality

In [9]:

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

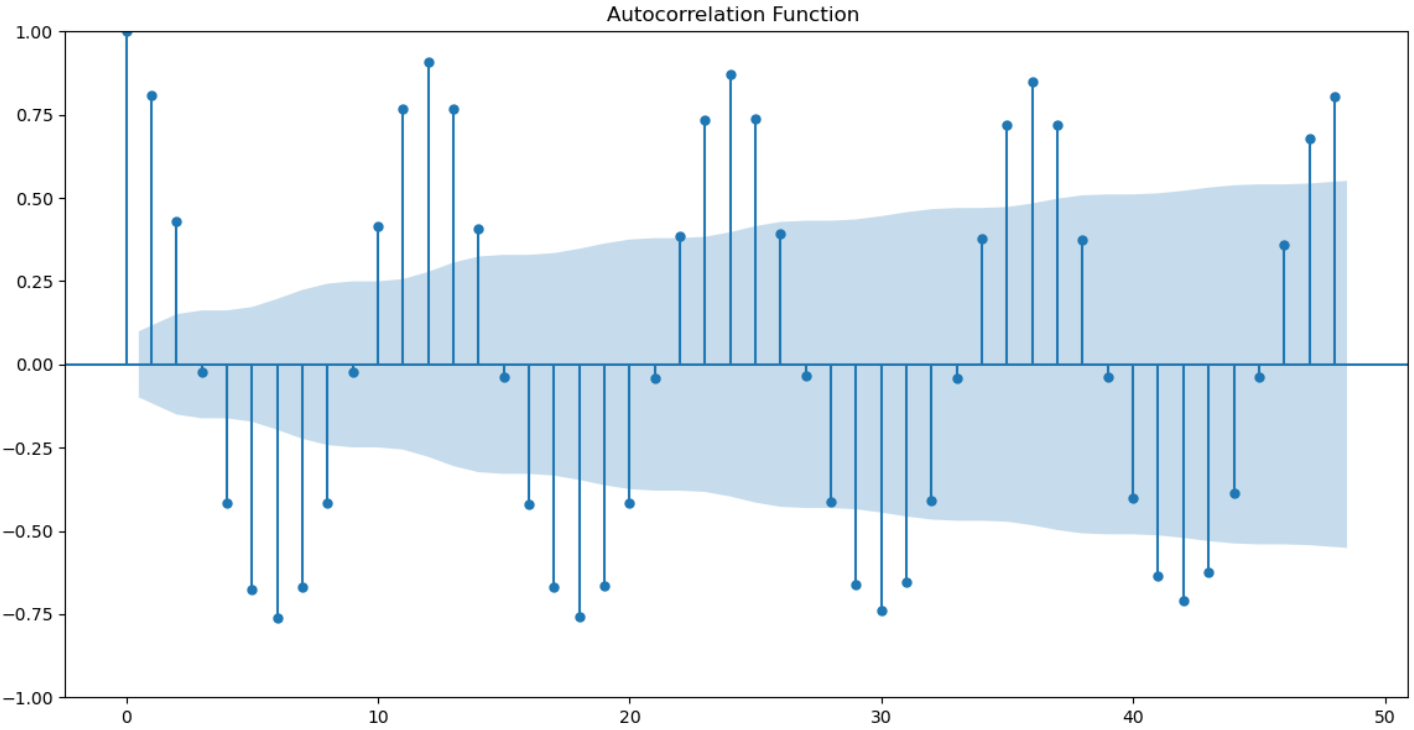


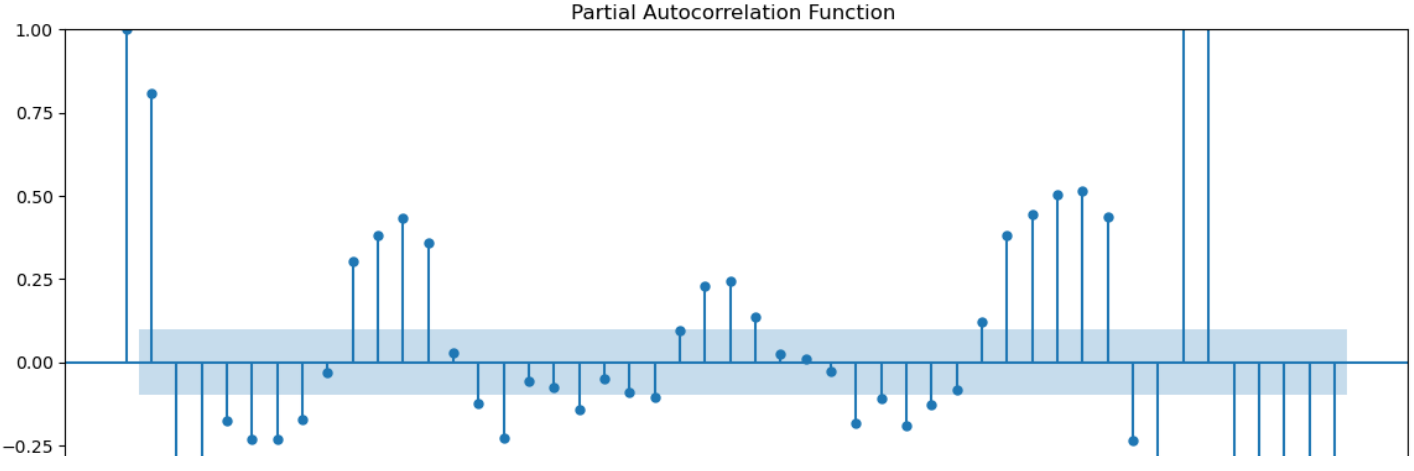


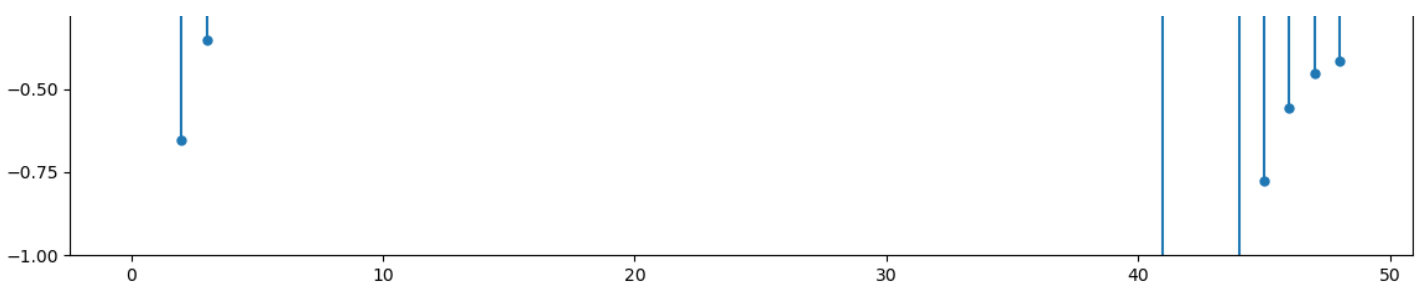
The decomposition of the original time series clearly reveals a seasonal component in the data, affirming its cyclical nature throughout the year. This analysis confirms the presence of seasonality in the dataset.

In [10]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Plotting the ACF for the differenced series
plt.figure(figsize=(14, 7))
plot_acf(df['tavg'], lags=48, ax=plt.gca())
plt.title('Autocorrelation Function ')
plt.show()

# Plotting the PACF for the differenced series
plt.figure(figsize=(14, 7))
plot_pacf(df['tavg'], lags=48, ax=plt.gca())
plt.title('Partial Autocorrelation Function ')
plt.show()
```







In [ ]:

```
```

Stationarity Check

In [11]:

```
from statsmodels.tsa.stattools import adfuller

# Perform the Augmented Dickey-Fuller test
adf_result = adfuller(df['tavg'])

print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
print('Critical Values:')
for key, value in adf_result[4].items():
    print('\t%s: %.3f' % (key, value))

# Interpretation
if adf_result[1] < 0.05:
    print("Reject the null hypothesis (H0), the data does not have a unit root and is sta
tionary.")
else:
    print("Fail to reject the null hypothesis (H0), the data has a unit root and is non-s
tationary.")
```

```
ADF Statistic: -2.660023
p-value: 0.081215
Critical Values:
 1%: -3.448
 5%: -2.869
 10%: -2.571
Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.
```

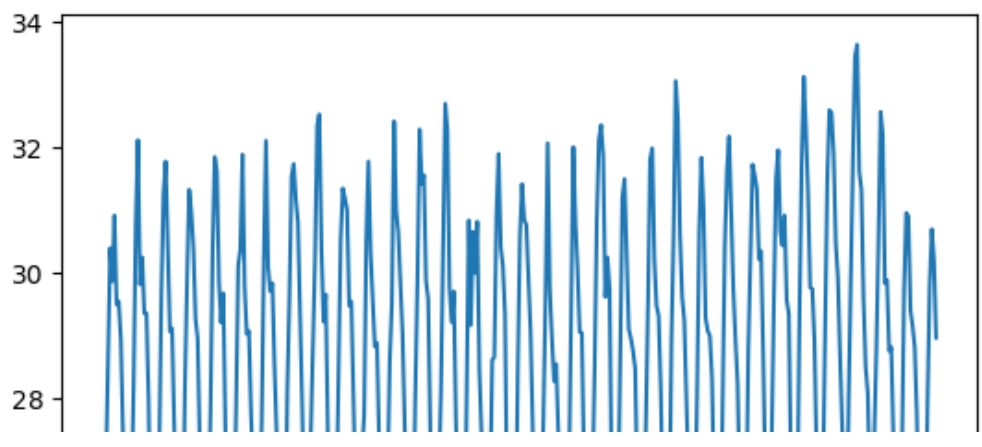
Since the series is not stationary I will do diffrencing

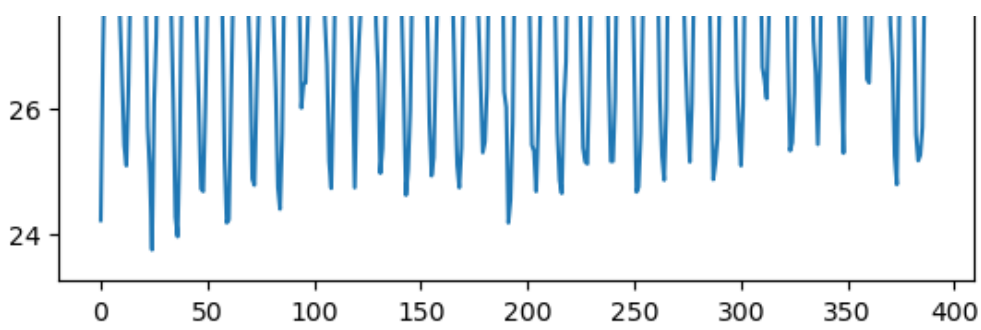
In [12]:

```
plt.plot(df['tavg'])
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x1fa1e4eaf40>]
```





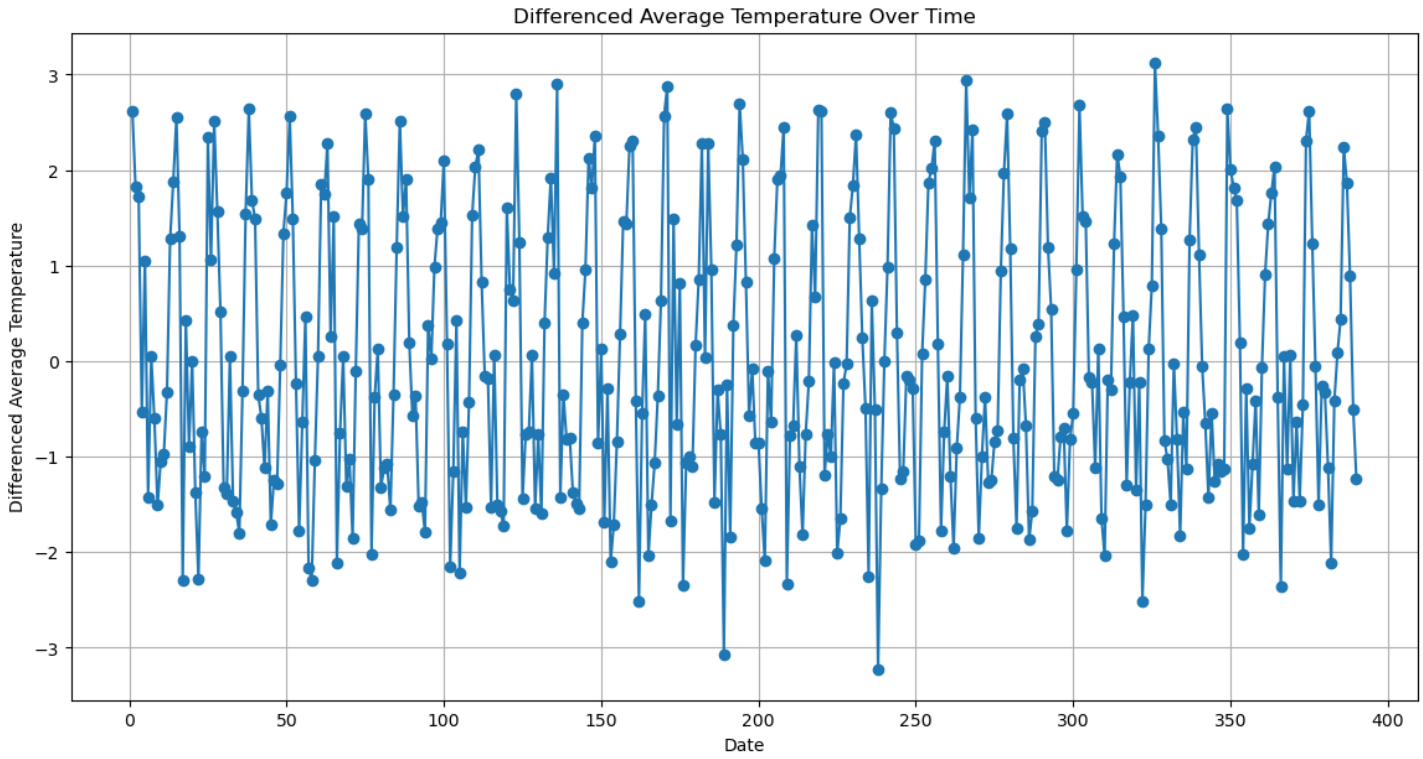
Differencing

In [13]:

```
# Apply differencing to the 'tavg' column
df['tavg_diff'] = df['tavg'].diff()

# Drop the NA values that are the result of differencing
df = df.dropna()
```

In [14]:

```
# Plot the differenced data
plt.figure(figsize=(14, 7))
plt.plot(df.index, df['tavg_diff'], marker='o')
plt.title('Differenced Average Temperature Over Time')
plt.xlabel('Date')
plt.ylabel('Differenced Average Temperature')
plt.grid(True)
plt.show()
```



Visualizng the Differenced Time Series plot

In [15]:

```
from statsmodels.tsa.stattools import adfuller

# Perform the Augmented Dickey-Fuller test
adf_result = adfuller(df['tavg_diff'])

print('ADF Statistic: %f' % adf_result[0])
```

```
print('p-value: %f' % adf_result[1])
print('Critical Values:')
for key, value in adf_result[4].items():
    print('\t%s: %.3f' % (key, value))

# Interpretation
if adf_result[1] < 0.05:
    print("Reject the null hypothesis (H0), the data does not have a unit root and is sta
tionary.")
else:
    print("Fail to reject the null hypothesis (H0), the data has a unit root and is non-s
tationary.")
```

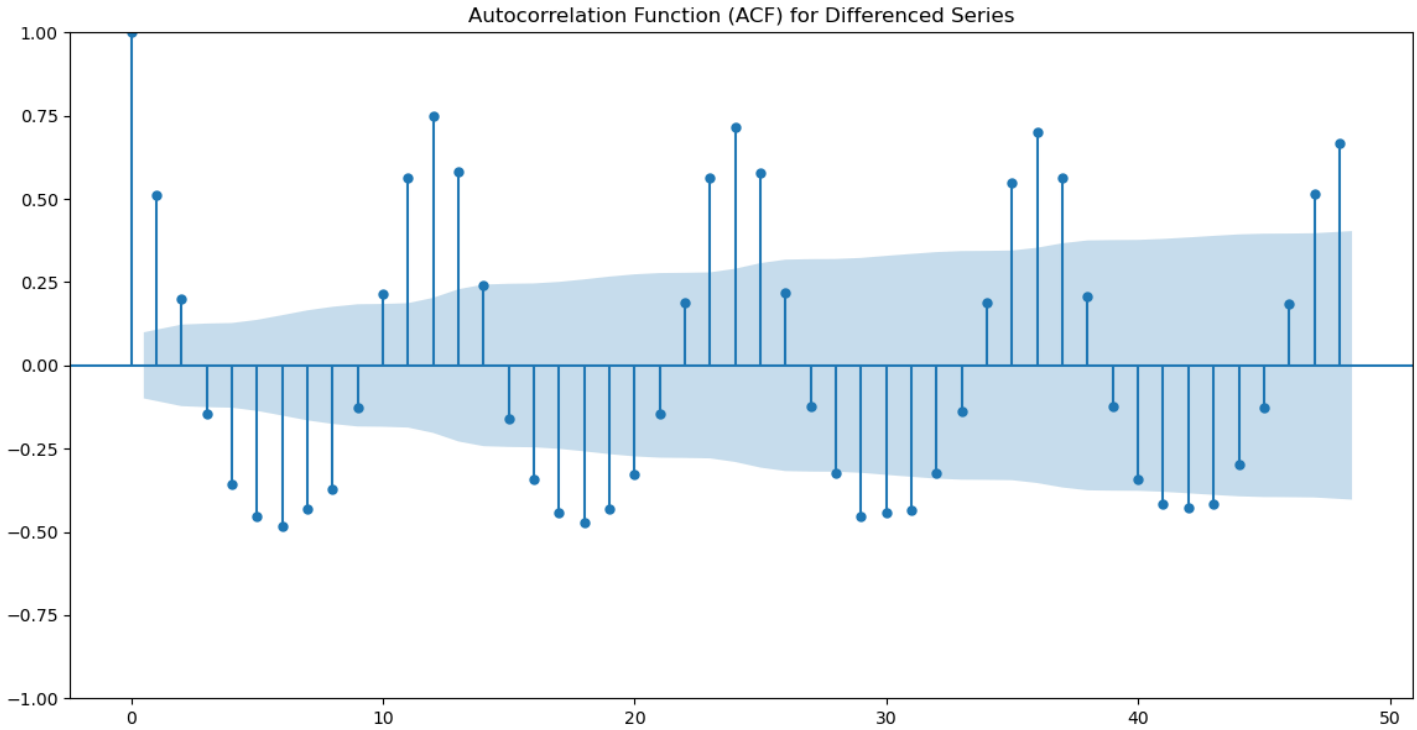
```
ADF Statistic: -7.303885
p-value: 0.000000
Critical Values:
 1%: -3.448
 5%: -2.869
 10%: -2.571
Reject the null hypothesis (H0), the data does not have a unit root and is stationary.
```

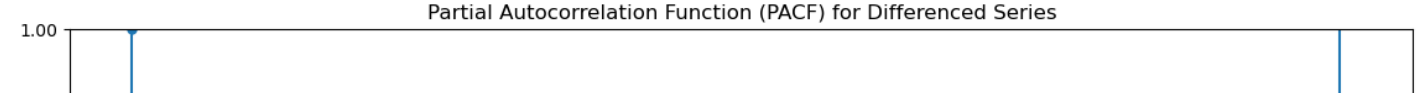
After performing the differencing operation we got the ADF test result as stationary.

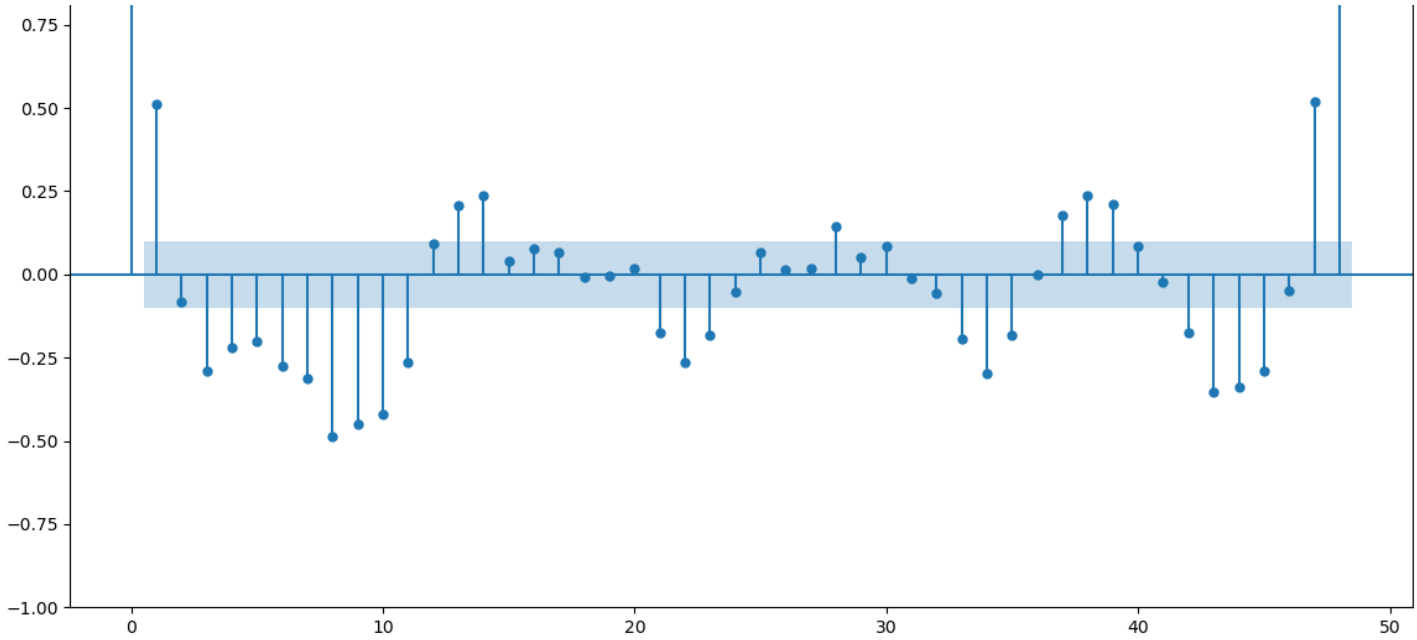
ACF and PACF plots

In [16]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Plotting the ACF for the differenced series
plt.figure(figsize=(14, 7))
plot_acf(df['tavg_diff'], lags=48, ax=plt.gca())
plt.title('Autocorrelation Function (ACF) for Differenced Series')
plt.show()

# Plotting the PACF for the differenced series
plt.figure(figsize=(14, 7))
plot_pacf(df['tavg_diff'], lags=48, ax=plt.gca())
plt.title('Partial Autocorrelation Function (PACF) for Differenced Series')
plt.show()
```







For ACF We observe 3 or 4 significant lags for non seasonal component and about 3 to 4 lags in seasonal component and for PACF 1 significant lags for non seasonal component and around 4 significant lags for seasonal component.

Train Test Split

In [17]:

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

Best SARIMA Model

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. Since we observed 4 significant lags in seasonal component and 3 to 4 significant lags in non seasonal component

p=1 to 4

q=1 to 4

P= 1 to 4

Q=1 to 4

S=12

In [18]:

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

# Loop over the range of parameters
for p in range(1, 4):
    for d in [1]:  # fixed depth as 1 since differencing of 1 was sufficient to make the
series stationary
```

```
        for q in range(1, 4):
            for P in range(1, 4):
                for D in [1]:
                    for Q in range(1, 4):
                        loop_counter += 1  # Increment the loop counter
                        try:
                            # Define and fit the model
                            model = SARIMAX(df_train['tavg_diff'],
                                            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)

                        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}")

# After all iterations, print the best AIC and parameters
print(f"Best Parameters: {best_params}")
print(f"Best AIC: {best_aic}")
print(f"Total loops completed: {loop_counter}")
```

```
Best Parameters: (1, 1, 2, 2, 1, 3)
Best AIC: 582.2106606807444
Total loops completed: 81
```

After 81 different combinations from the given range we got the best parameters as (1, 1, 2, 2, 1, 3) with an AIC 582.2106606807444

In [19]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

p, d, q, P, D, Q, s = 1, 1, 2, 2, 1, 3, 12

# Define the model with the adjusted parameters
adjusted_model = SARIMAX(df_train['tavg_diff'],
                         order=(p, d, q),
                         seasonal_order=(P, D, Q, s),
                         enforce_stationarity=False,
                         enforce_invertibility=False)

# Fit the model
adjusted_results = adjusted_model.fit()

# Conduct diagnostic checks on the adjusted model
print(adjusted_results.summary())
```

```
                                     SARIMAX Results

==========================================================================================
=========
Dep. Variable:                                  tavg_diff   No. Observations:
360
Model:             SARIMAX(1, 1, 2)x(2, 1, [1, 2, 3], 12)   Log Likelihood
-282.105
Date:                                   Thu, 02 May 2024   AIC
582.211
Time:                                           23:01:45   BIC
615.782
Sample:                                                0   HQIC
595.634
                                                  - 360
```

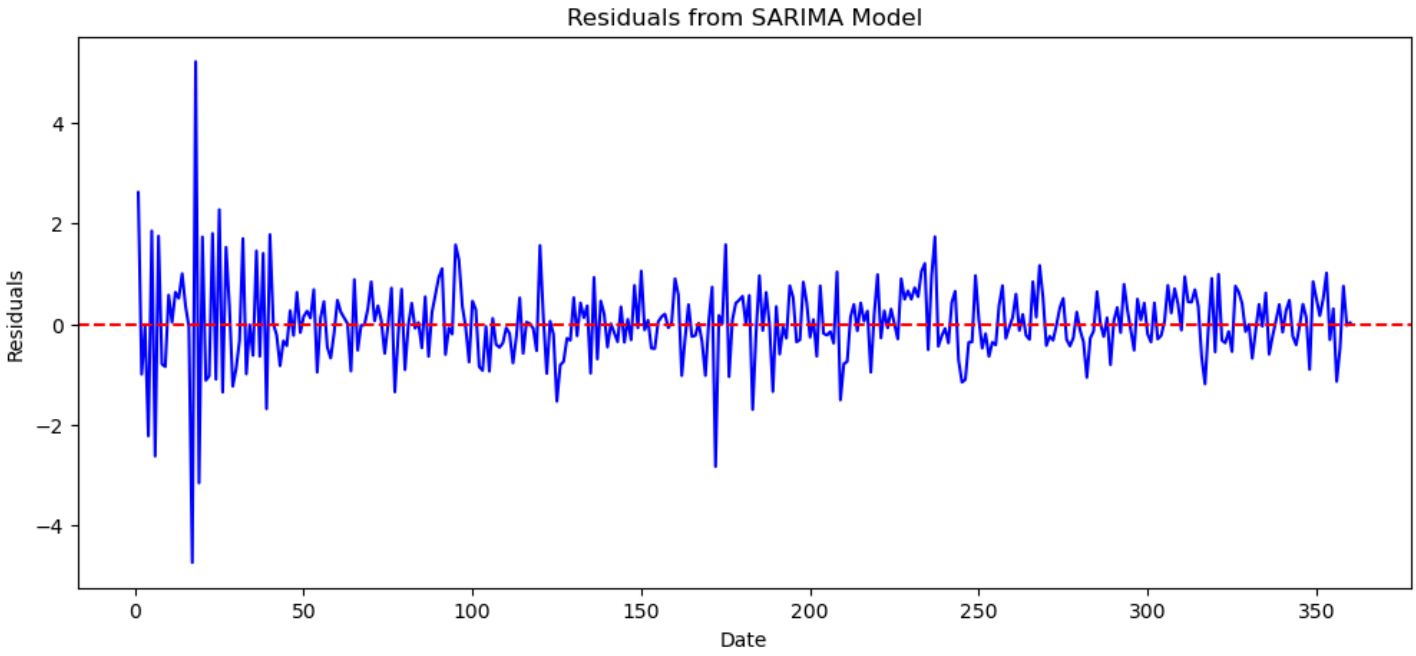
```
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2395      0.069      3.468      0.001       0.104       0.375
ma.L1         -1.9132      0.025    -75.106      0.000      -1.963      -1.863
ma.L2          0.9142      0.025     36.461      0.000       0.865       0.963
ar.S.L12      -0.8839      0.229     -3.854      0.000      -1.333      -0.434
ar.S.L24      -0.4441      0.150     -2.957      0.003      -0.738      -0.150
ma.S.L12      -0.1500      1.440     -0.104      0.917      -2.972       2.672
ma.S.L24      -0.5018      1.148     -0.437      0.662      -2.751       1.748
ma.S.L36      -0.3531      0.495     -0.714      0.476      -1.323       0.617
sigma2         0.3309      0.458      0.723      0.470      -0.566       1.228
==============================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                29.08
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.80   Skew:                            -0.31
Prob(H) (two-sided):                  0.26   Kurtosis:                         4.38
==============================================================================

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

Residual Analysis

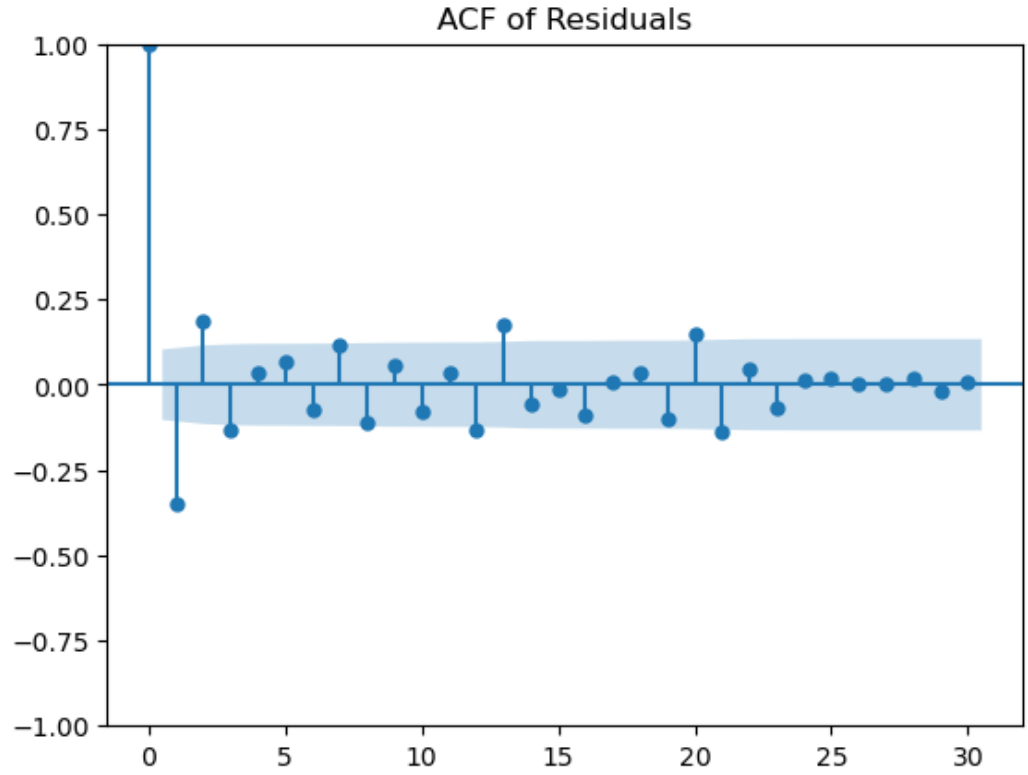
In [20]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import shapiro, probplot

# Extract the residuals
residuals = adjusted_results.resid

# Plotting the residuals
plt.figure(figsize=(12, 5))
plt.plot(residuals, color='blue')
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals from SARIMA Model')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.show()

# Plotting the ACF of residuals
plt.figure(figsize=(12, 5))
plot_acf(residuals, lags=30)
plt.title('ACF of Residuals')
plt.show()

# Plotting the PACF of residuals
plt.figure(figsize=(12, 5))
plot_pacf(residuals, lags=30)
plt.title('PACF of Residuals')
plt.show()

# Normal Q-Q plot
plt.figure(figsize=(12, 5))
probplot(residuals, dist="norm", plot=plt)
plt.title('Normal Q-Q Plot')
plt.show()

# Histogram of residuals
plt.figure(figsize=(12, 5))
sns.histplot(residuals, kde=True)
plt.title('Histogram of Residuals')
plt.show()
```

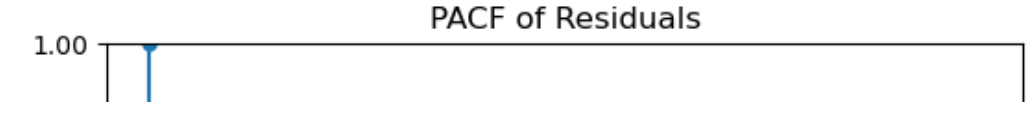
```
# Calculate the Shapiro-Wilk test on the residuals
shapiro_test = shapiro(residuals)
shapiro_stat, shapiro_p_value = shapiro_test

# Interpret the results
alpha = 0.05
if shapiro_p_value > alpha:
    conclusion = "Fail to reject the null hypothesis (H0), the data is normally distribut
ed."
else:
    conclusion = "Reject the null hypothesis (H0), the data is not normally distributed."

# Print the results
print(f"Shapiro-Wilk Test: Statistics={shapiro_stat}, p-value={shapiro_p_value}")
print(conclusion)
```

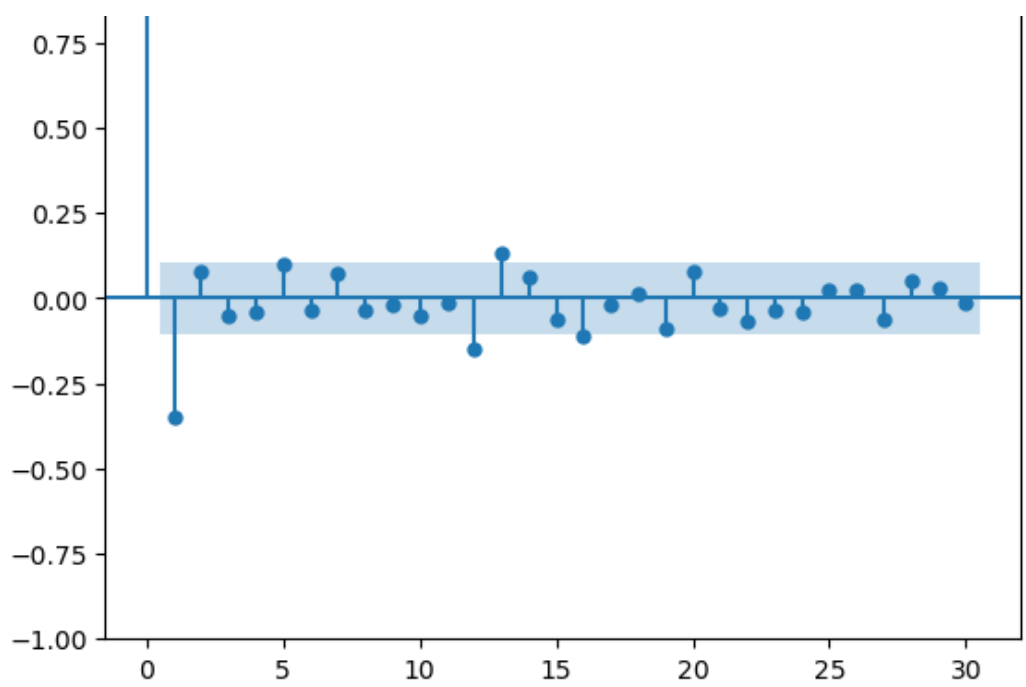


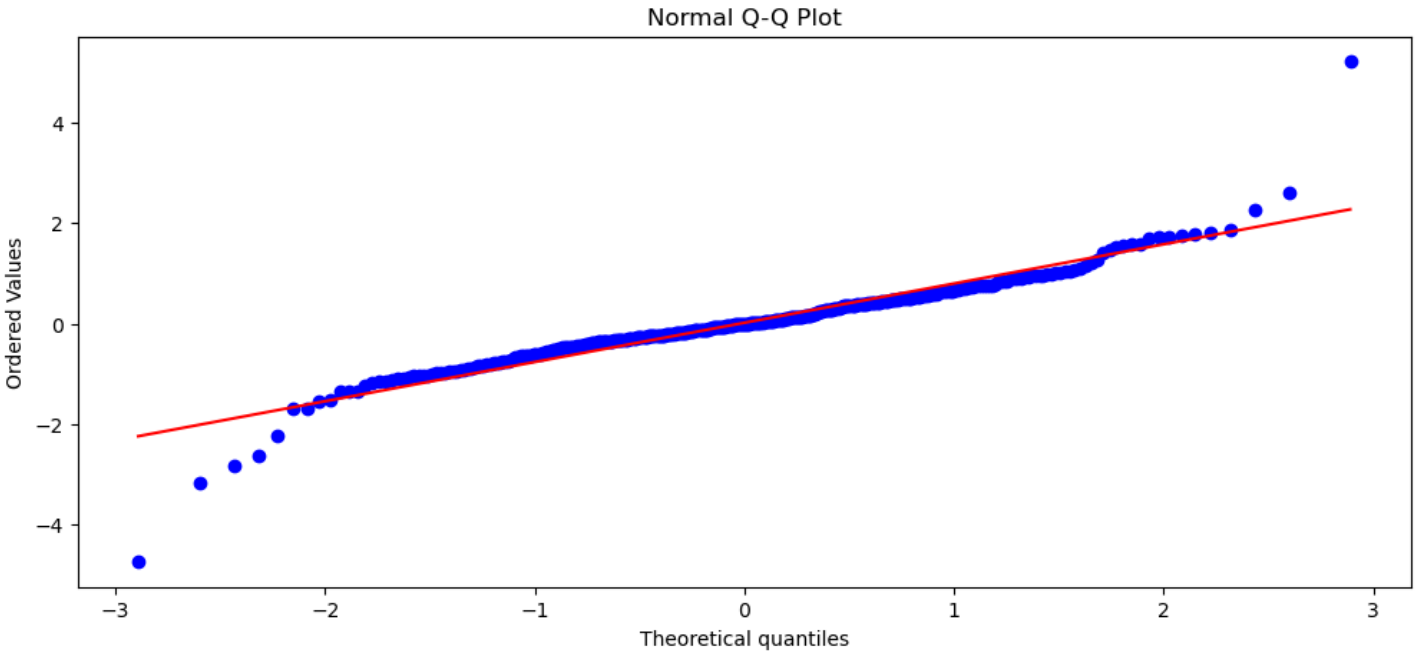

```
<Figure size 1200x500 with 0 Axes>
```

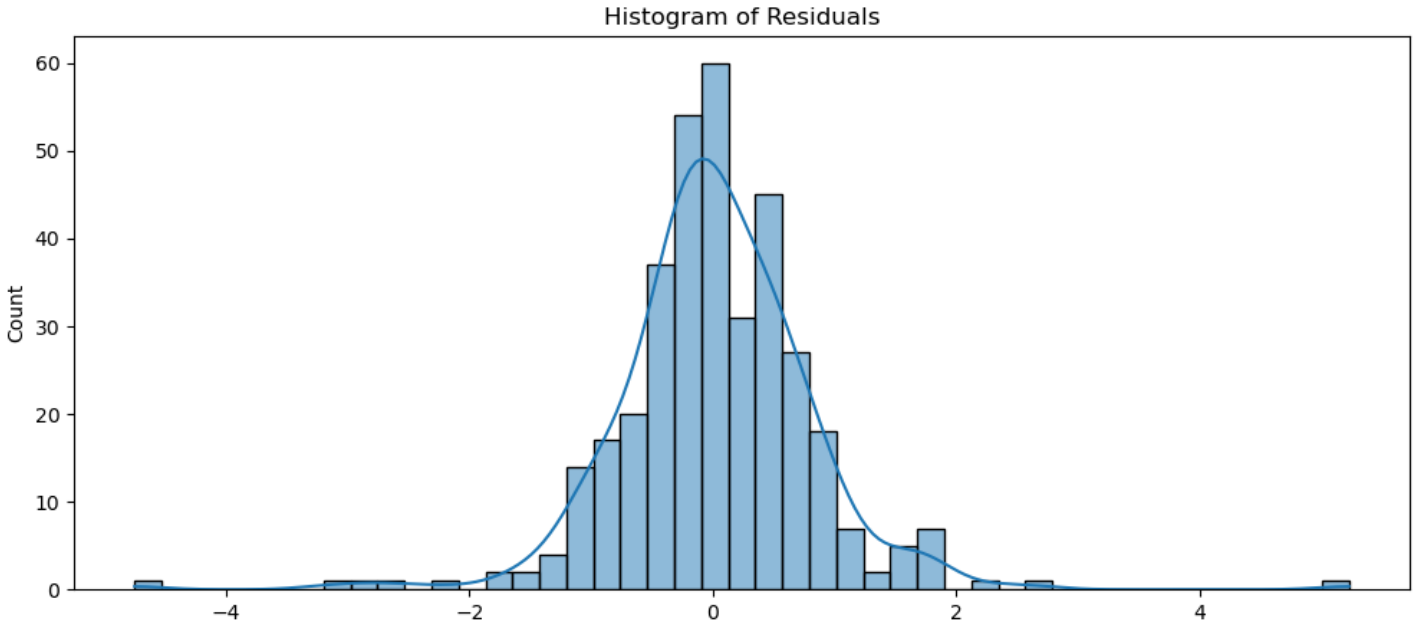



```
<Figure size 1200x500 with 0 Axes>
```










```
Shapiro-Wilk Test: Statistics=0.914070188999176, p-value=1.8269673968118738e-13
Reject the null hypothesis (H0), the data is not normally distributed.
```

The residuals appear to be randomly distributed without any obvious patterns

The residuals appear to be randomly distributed without any obvious patterns

For ACF and PACF plots of residuals I can see some significant lags.

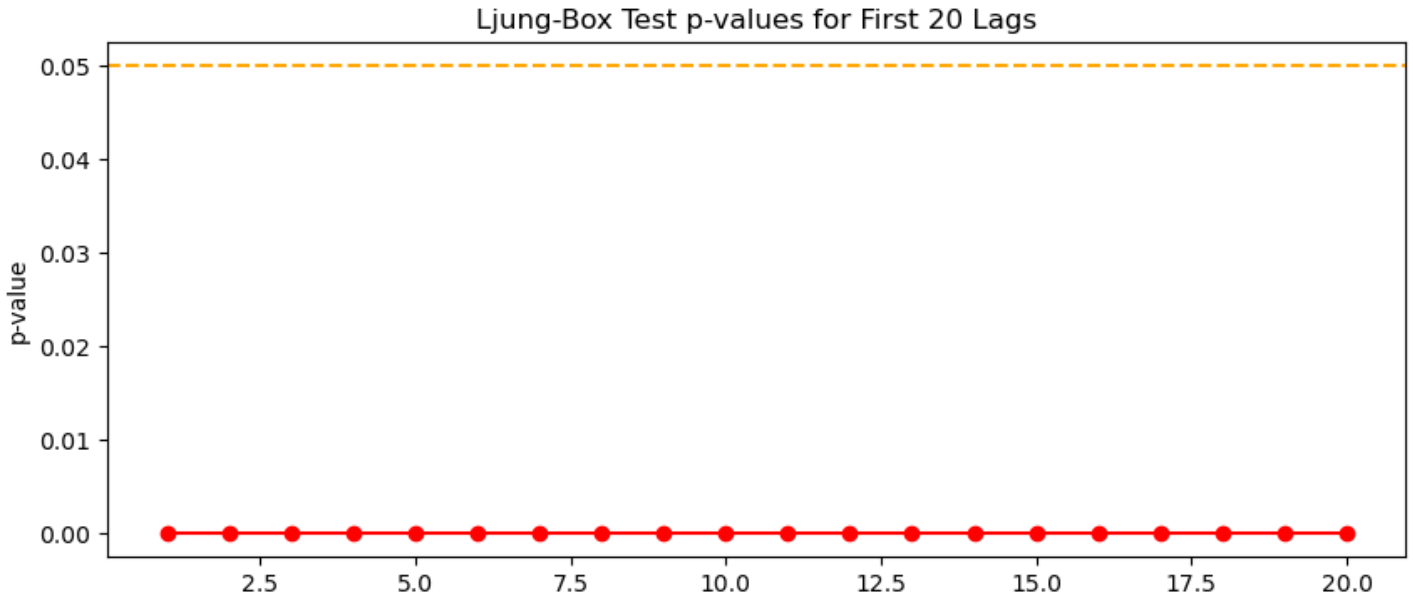
The residuals lie mostly along the red line indicating that the residuals are approximately normally distributed and there are slight deviations at the tails on both ends which tells some outliers are present.

The histogram is almost normally distibuted but bit of right-skewed.

In [24]:

```
from statsmodels.stats.diagnostic import acorr_ljungbox
import matplotlib.pyplot as plt
import statsmodels.api as sm

ljungbox_results = acorr_ljungbox(residuals, lags=20, return_df=True)
# Print the results
print(ljungbox_results)

# Plotting the Ljung-Box test p-values
plt.figure(figsize=(10, 4))
plt.plot(ljungbox_results['lb_pvalue'], marker='o', linestyle='-', color='red')
plt.title('Ljung-Box Test p-values for First 20 Lags')
plt.xlabel('Lags')
plt.ylabel('p-value')
plt.axhline(y=0.05, color='orange', linestyle='--')  # significance line at p = 0.05
plt.show()
```

```
       lb_stat     lb_pvalue
1    44.076244  3.158308e-11
2    57.093365  4.002519e-13
3    63.554092  1.022339e-13
4    64.066916  4.045759e-13
5    65.704860  8.003139e-13
6    67.672295  1.226128e-12
7    72.490677  4.629615e-13
8    76.972649  1.984447e-13
9    78.126777  3.803387e-13
10   80.376128  4.236468e-13
11   80.845047  1.012746e-12
12   87.332755  1.617815e-13
13   99.147526  2.426364e-15
14  100.299547  4.155092e-15
15  100.382218  1.104202e-14
16  103.309079  8.276395e-15
17  103.331810  2.138702e-14
18  103.846576  4.348491e-14
19  107.708232  2.104262e-14
20  116.163701  1.457291e-15
```



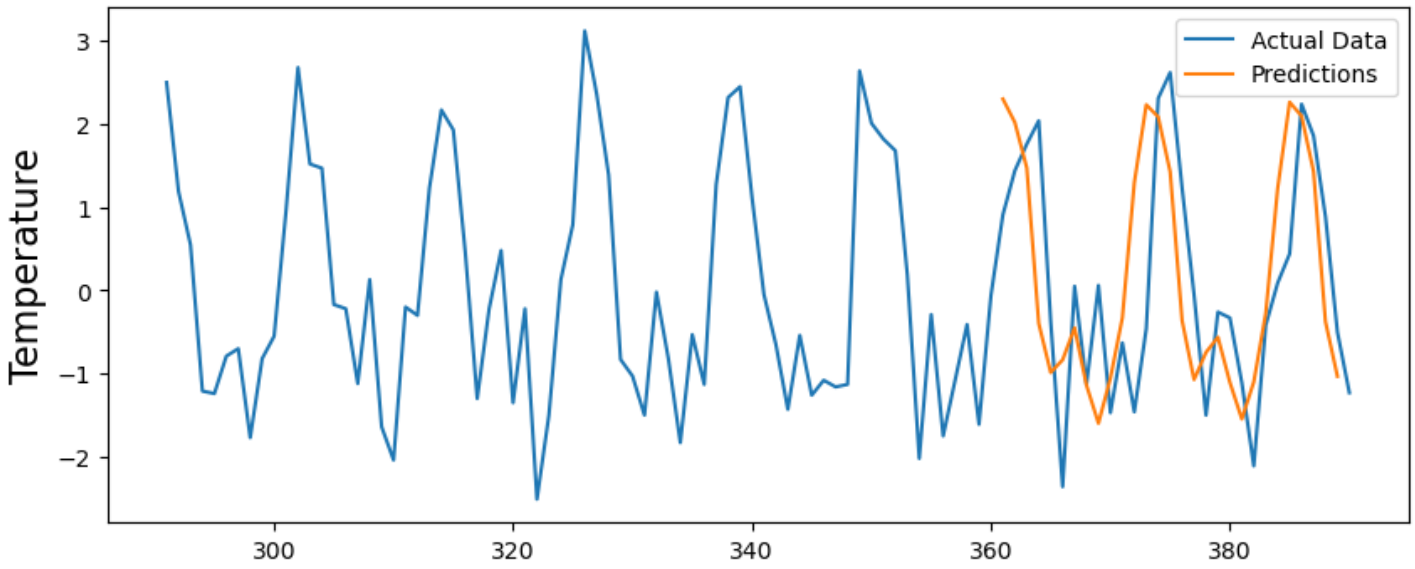
Lags

The Ljung-Box test results with extremely low p-values across all lags strongly suggest that the residuals are not independent and show significant autocorrelation.

Forecasting

In [33]:

```
# Forecast future values based on the length of the test dataset
predictions = results.forecast(len(df_test))

predictions = pd.Series(predictions, index=df_test.index)
# Compute residuals as the difference between actual and predicted values
residuals = df_test['tavg_diff'] - predictions
plt.figure(figsize=(10, 4))
# Plot the last 100 points of actual data for comparison
plt.plot(df['tavg_diff'][len(df)-100:len(df)], label='Actual Data')
# Plot the forecasted values
plt.plot(predictions, label='Predictions')
plt.legend(fontsize=10)
plt.ylabel('Temperature', fontsize=16)
plt.show()
```



The orange line representing the predictions closely follows the blue line of the actual data, indicating that the SARIMA model is effectively capturing the trend and seasonality in the data.

Conclusion

In this comprehensive time series analysis project, we successfully addressed the challenges posed by seasonaldata. 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 (1, 1, 2, 2, 1, 3) as the most effective, evidenced by its optimal AIC score of 582.211. This model not only outperformed others in terms of fit but also demonstrated its efficiancy in forecasting.

In conclusion, this project not only showcased the effectiveness in dealing with seasonal time series data but also underscored the importance of selecting appropriate models

based on comprehensive evaluation criteria.

In [ ]: