D214 Capstone

Modeling Inflation Adusted Recessionary Lumber Prices

July 1981 - November 1982 Recession

Eric Yarger

Import Packages

In [1]:
```
# Import Initial Libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats
from statsmodels.tsa.stattools import adfuller
import statsmodels
import datetime
import platform
from pmdarima.arima import ndiffs
from statsmodels.tsa.seasonal import seasonal_decompose
from pylab import rcParams
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.graphics.tsaplots import plot_pacf
import warnings
from scipy import signal
from pmdarima.arima import StepwiseContext
from pmdarima.arima import auto_arima
from pmdarima.model_selection import train_test_split
```

Environment

In [2]:
```
# Windows 10, Anaconda, JupyterLab, JupyterNotebook
# Jupyter environment version
!jupyter --version
```

```
Selected Jupyter core packages...
IPython          : 7.31.1
ipykernel        : 6.15.2
ipywidgets       : not installed
jupyter_client   : 7.3.5
jupyter_core     : 4.10.0
jupyter_server   : 1.18.1
jupyterlab       : 3.4.4
nbclient         : 0.5.13
nbconvert        : 6.4.4
nbformat         : 5.5.0
notebook         : 6.4.12
qtconsole        : not installed
traitlets        : 5.1.1
```

In [3]:
```
# Python Version
print(platform.python_version())
```

```
3.7.13
```

In [4]:
```
#Load Medical Dataset
df = pd.read_csv('C:/Users/ericy/Desktop/lumber_trading_days_adj.csv')
```

Data Selection for Analysis

In [5]:
```
#------ Select Data Set for Recession
df = df[1664:2525]
```

In [6]:
```
df
```

Out[6]:

	Date	Trading Days	2022_Value	Value
1664	1979-07-05	1665	949.520	228.8
1665	1979-07-06	1666	938.730	226.2
1666	1979-07-09	1667	934.580	225.2
1667	1979-07-10	1668	940.390	226.6
1668	1979-07-11	1669	952.840	229.6
...	...	...	...	...
2520	1982-11-24	2521	493.190	165.5
2521	1982-11-26	2522	507.792	170.4
2522	1982-11-29	2523	504.216	169.2
2523	1982-11-30	2524	497.362	166.9
2524	1982-12-01	2525	498.554	167.3

861 rows × 4 columns

D1: Exploratory Data Analysis

In [7]:
```
df = df[['Trading Days','2022_Value']]
```

In [8]:
```
df
```

Out[8]:

	Trading Days	2022_Value
1664	1665	949.520
1665	1666	938.730
1666	1667	934.580
1667	1668	940.390
1668	1669	952.840
...	...	...
2520	2521	493.190
2521	2522	507.792
2522	2523	504.216
2523	2524	497.362
2524	2525	498.554

861 rows × 2 columns

EDA

In [9]:
```
df.head()
```

Out[9]:

	Trading Days	2022_Value
1664	1665	949.52
1665	1666	938.73
1666	1667	934.58
1667	1668	940.39
1668	1669	952.84

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 861 entries, 1664 to 2524
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Trading Days  861 non-null    int64  
 1   2022_Value    861 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 13.6 KB
```

In [11]:
```
df.shape
```

Out[11]:
```
(861, 2)
```

In [12]:
```
df.describe()
```

Out[12]:

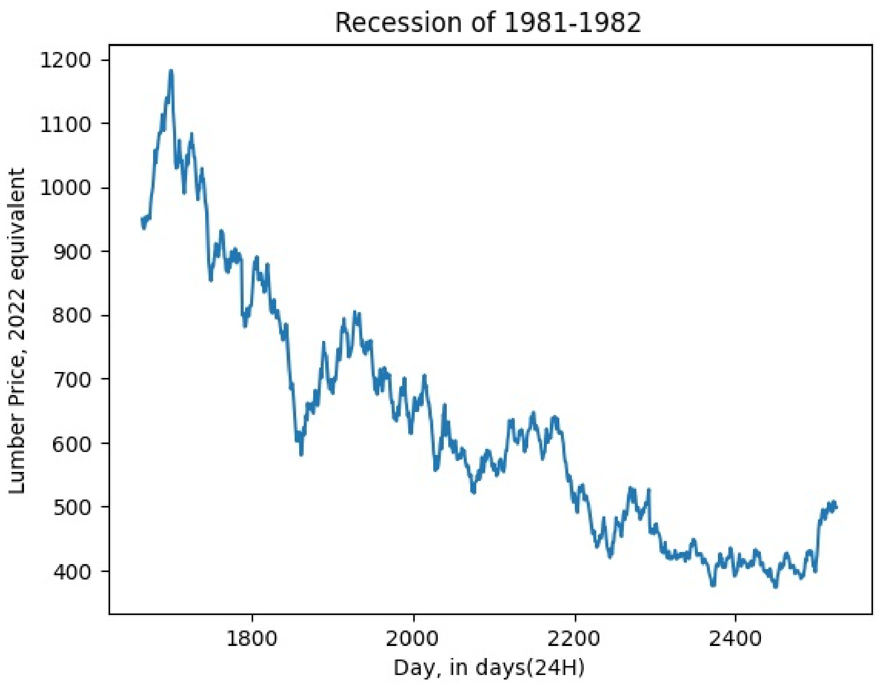
	Trading Days	2022_Value
count	861.000000	861.000000
mean	2095.000000	627.133584
std	248.693587	195.485514
min	1665.000000	373.394000
25%	1880.000000	453.556000
50%	2095.000000	599.438000
75%	2310.000000	740.778000
max	2525.000000	1182.335000

In [13]:
```
df.isnull().any()
```

Out[13]:
```
Trading Days    False
2022_Value      False
dtype: bool
```

Line Graph Visualization

In [14]:
```
#-------
plt.plot(df['Trading Days'],df['2022_Value'])
plt.title('Recession of 1981-1982')
plt.xlabel('Day, in days(24H)')
plt.ylabel('Lumber Price, 2022 equivalent')
plt.show()
```



Data Cleaning

In [15]:
```
# Drop any null columns
df = df.dropna()
```

D2: Time Step Formatting, Indexing

Set df['Trading Days'] to Index

In [16]:
```
# Day to datetime
df['Trading Days'] = pd.to_datetime(df['Trading Days'], unit='D')
```

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 861 entries, 1664 to 2524
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Trading Days  861 non-null    datetime64[ns]
 1   2022_Value    861 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 20.2 KB
```

In [18]:

```
# Set Day as Index
df.set_index('Trading Days',inplace=True)
```

In [19]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 861 entries, 1974-07-24 to 1976-11-30
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   2022_Value  861 non-null    float64
dtypes: float64(1)
memory usage: 13.5 KB
```

In [20]:

```
df
```

Out[20]:

Trading Days	2022_Value
1974-07-24	949.520
1974-07-25	938.730
1974-07-26	934.580
1974-07-27	940.390
1974-07-28	952.840
...	...
1976-11-26	493.190
1976-11-27	507.792
1976-11-28	504.216
1976-11-29	497.362
1976-11-30	498.554

861 rows × 1 columns

D3 Stationarity Analysis

Augmented Dickey Fuller (ADF) Test

Assess stationarity of dataset

In [21]:

```
# Code Reference (Making time series stationary | Python, n.d.)
dicky_fuller_test = adfuller(df)
```

In [22]:

```
dicky_fuller_test
```

Out[22]:

```
(-1.5443808173621811,
 0.5114538150396029,
 1,
 859,
 {'1%': -3.437985563419411,
  '5%': -2.864910527649228,
  '10%': -2.5685647265209433},
 6527.844611650443)
```

In [23]:

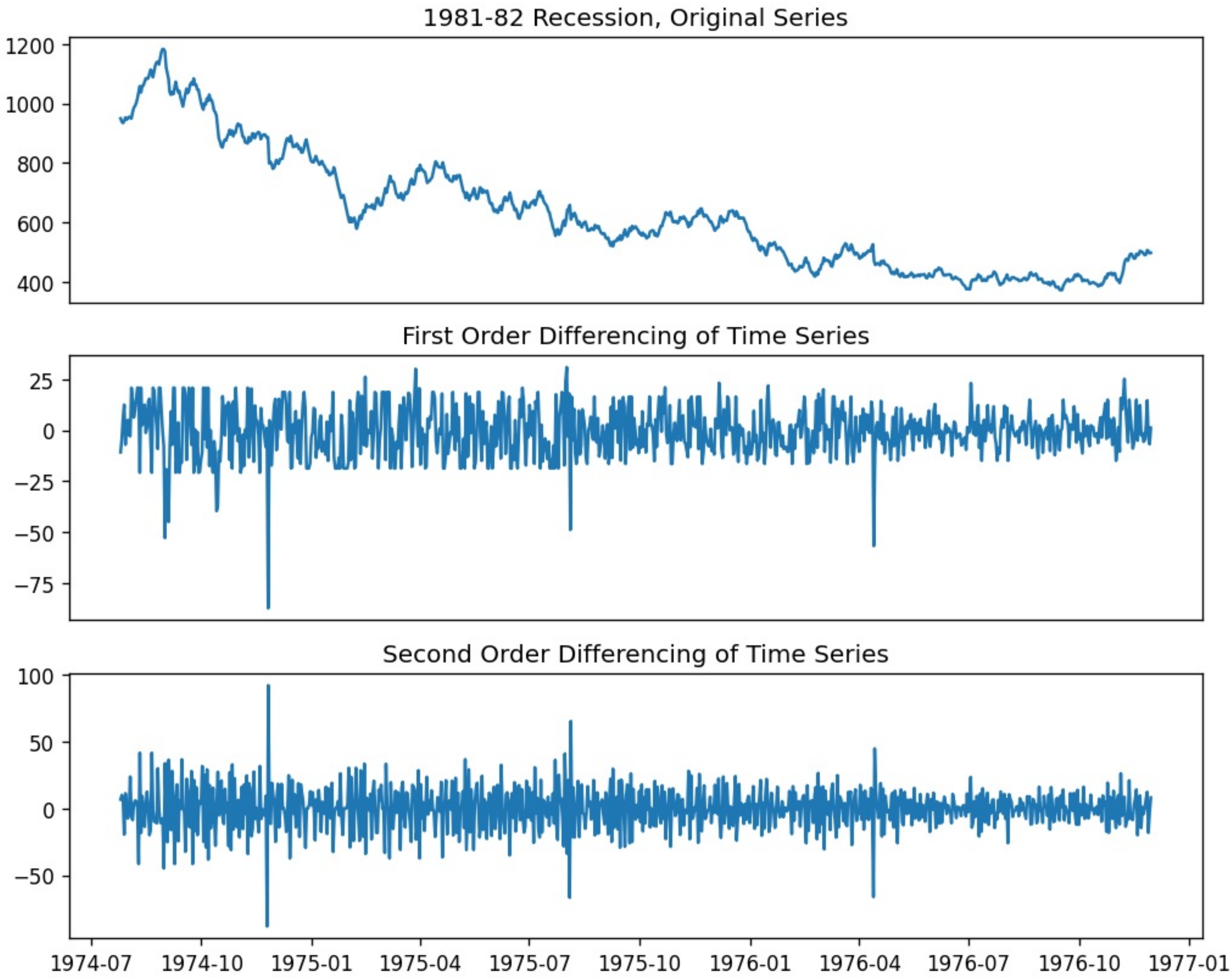
```
# Results show p = .51145
# Data does not reject null hypothesis at p < .05
# Therefore, Time series is determined to be non-stationary
```

D4 Differencing

1st and 2nd order Differencing

finding 'd' for ARIMA model

In [24]:
```
# Set plot parameters for multi-ax subplots
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':120})

# Establish that there are three subplots
fig, (ax1, ax2, ax3) = plt.subplots(3)

# Plot the original dataset
ax1.plot(df); ax1.set_title('1981-82 Recession, Original Series'); ax1.axes.xaxis.set_visible(False)

# First Order differencing of Time Series
ax2.plot(df.diff()); ax2.set_title('First Order Differencing of Time Series'); ax2.axes.xaxis.set_visible(False

# Second Order Differencing of Time Series
ax3.plot(df.diff().diff()); ax3.set_title('Second Order Differencing of Time Series')

# Plot all three graphs
plt.show()
```



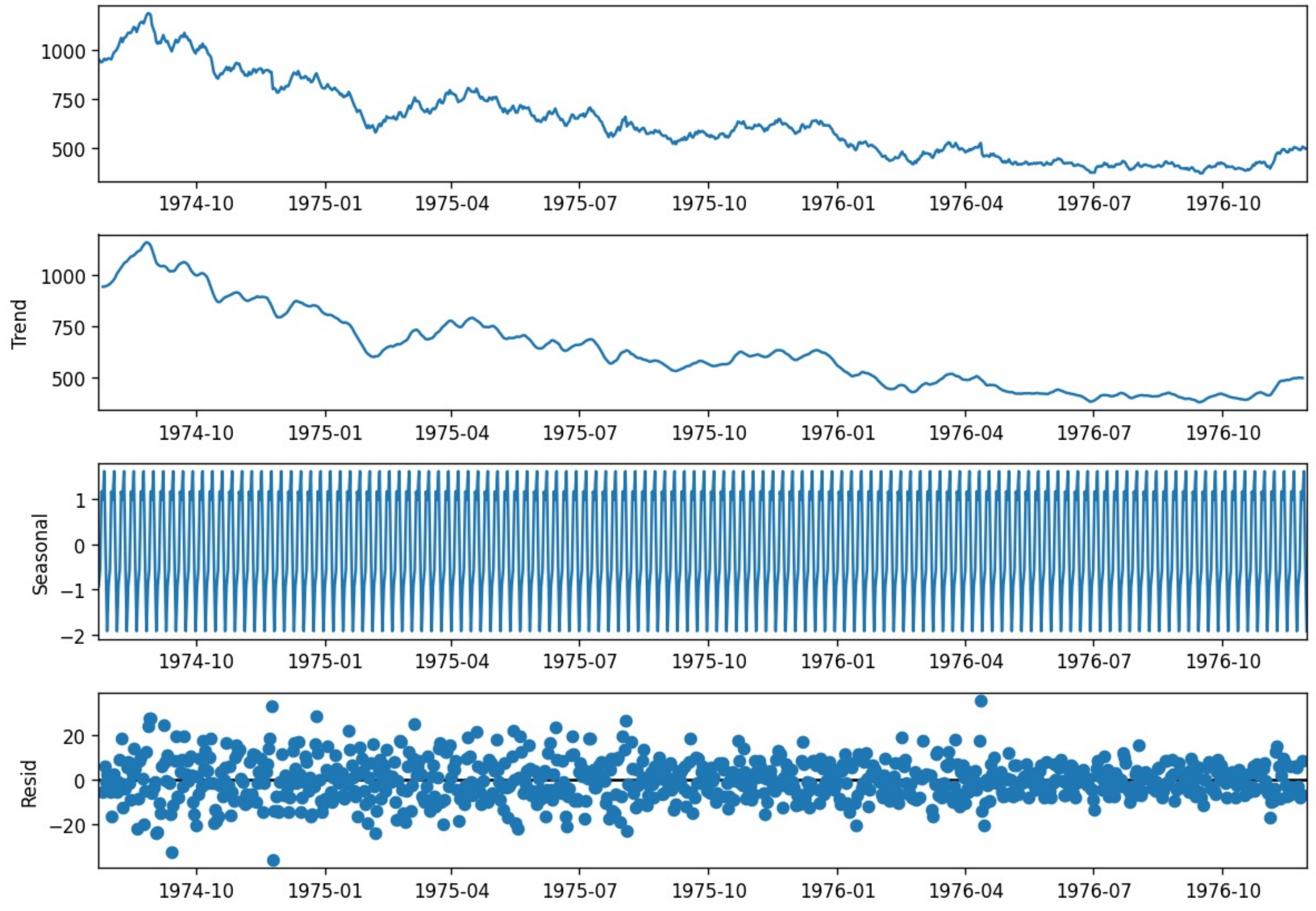
In [25]:
```
# Using pmdarima's ndiffs to find differencing term
# Code reference (Verma, 2021)

kpss_diffs = ndiffs(df, alpha=0.05, test='kpss', max_d=6)
adf_diffs = ndiffs(df, alpha=0.05, test='adf', max_d=6)
n_diffs = max(adf_diffs, kpss_diffs)

print(f"Estimated differencing term: {n_diffs}")
```

```
Estimated differencing term: 1
```

D5 Seasonality Analysis

In [26]:
```
# Code Reference (Boston, 2020)
result = seasonal_decompose(df)
```

In [27]:
```
# plotting the result of our seasonal decomposition from the step above
rcParams['figure.figsize'] = 10,7
result.plot();
```



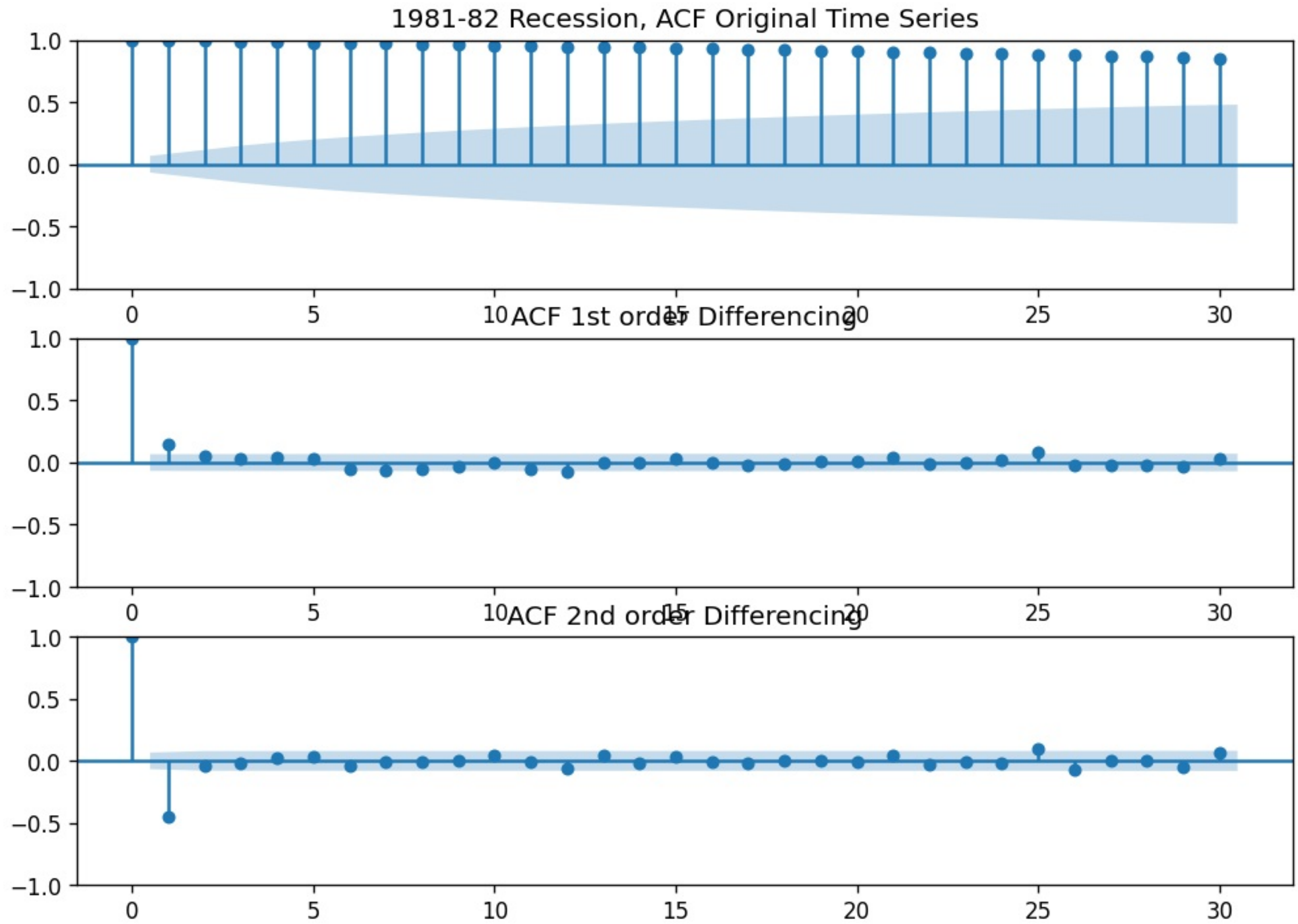
D6 ACF and PACF

Finding order of MA term 'q'

Using Autocorrelation function (ACF)

In [28]:

```
fig, (ax1, ax2, ax3) = plt.subplots(3)
plot_acf(df, ax=ax1, title='1981-82 Recession, ACF Original Time Series');
plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```

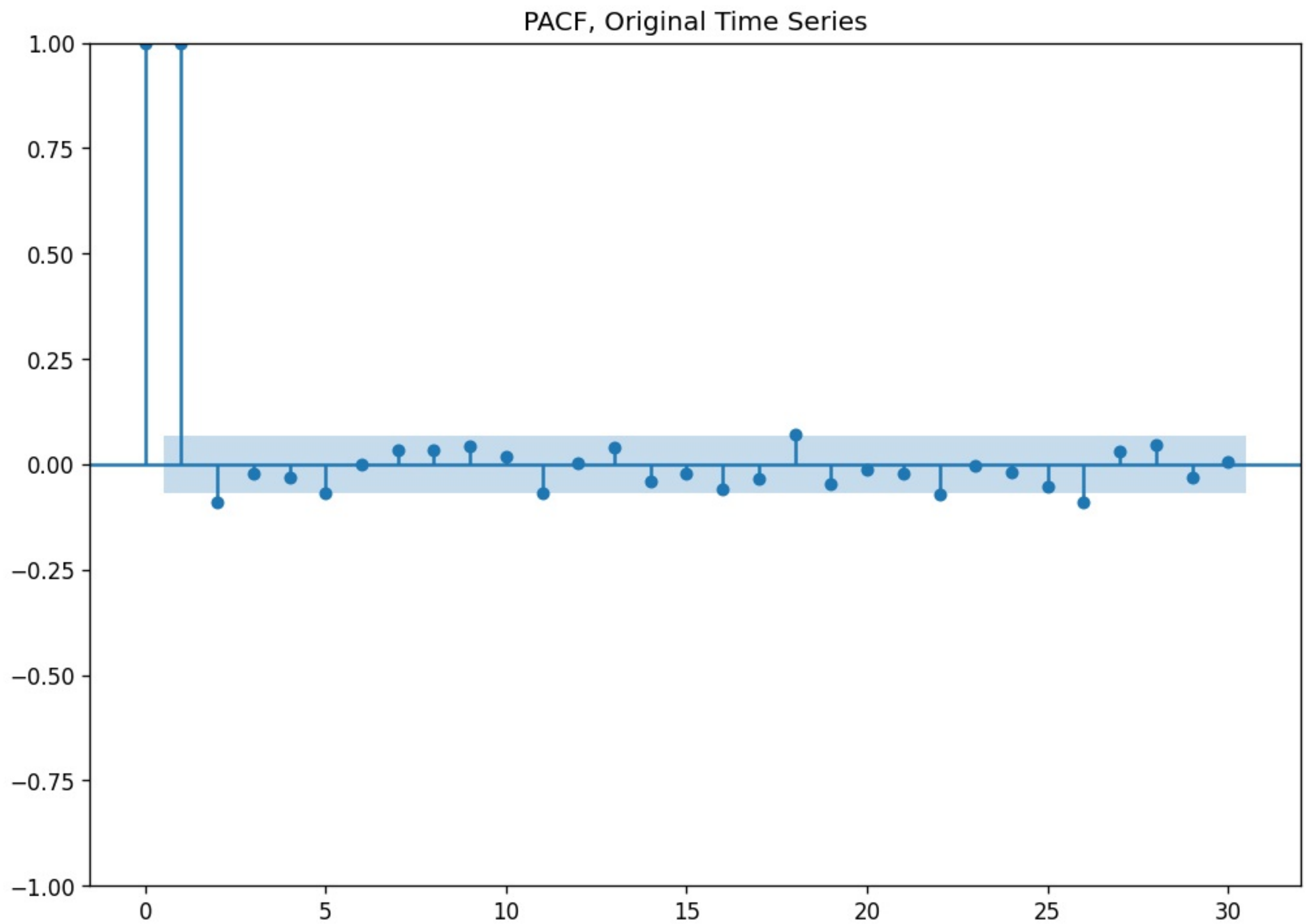


Finding order of AR term 'p'

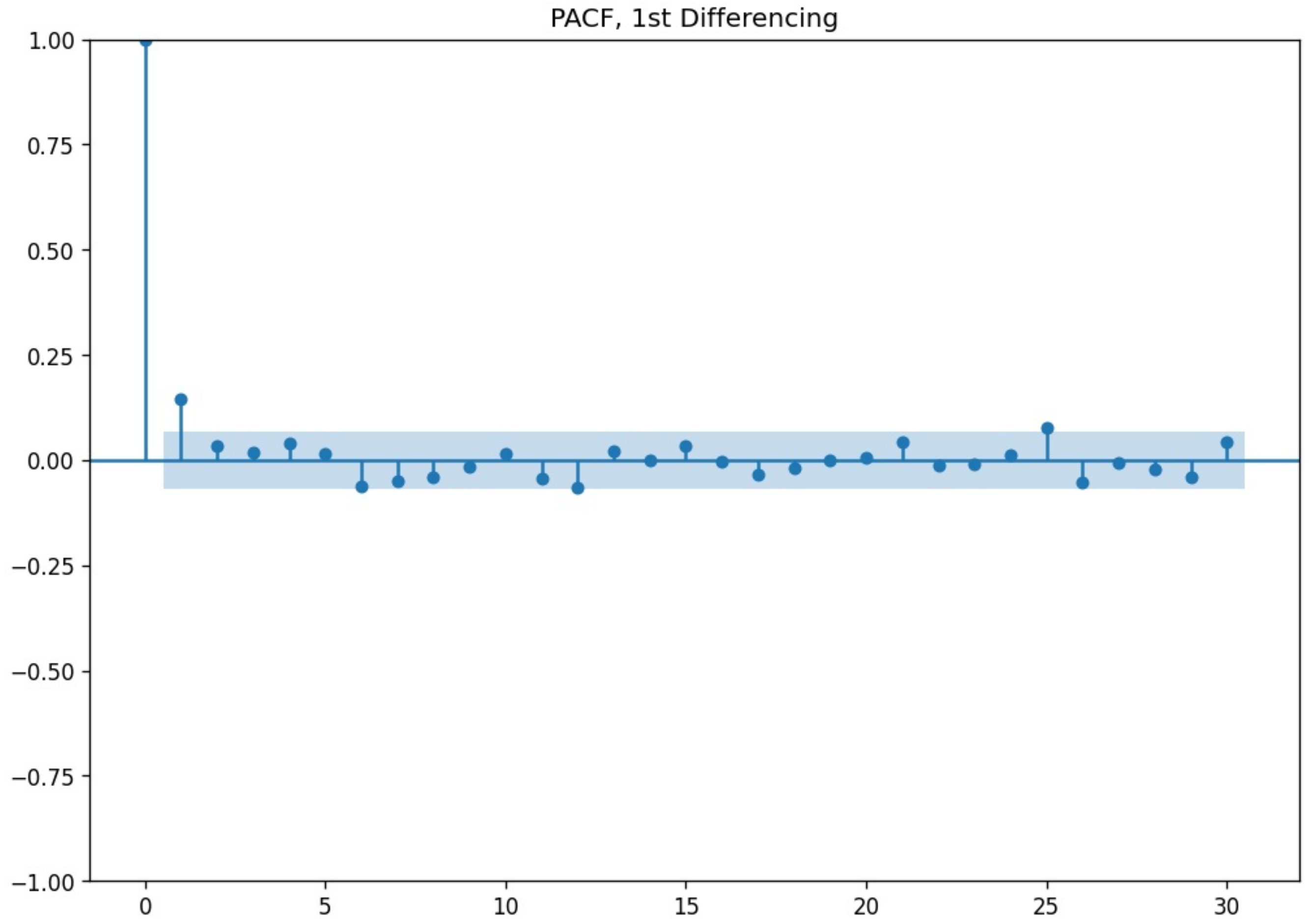
Using Partial autocorrelation (PACF)

In [29]:

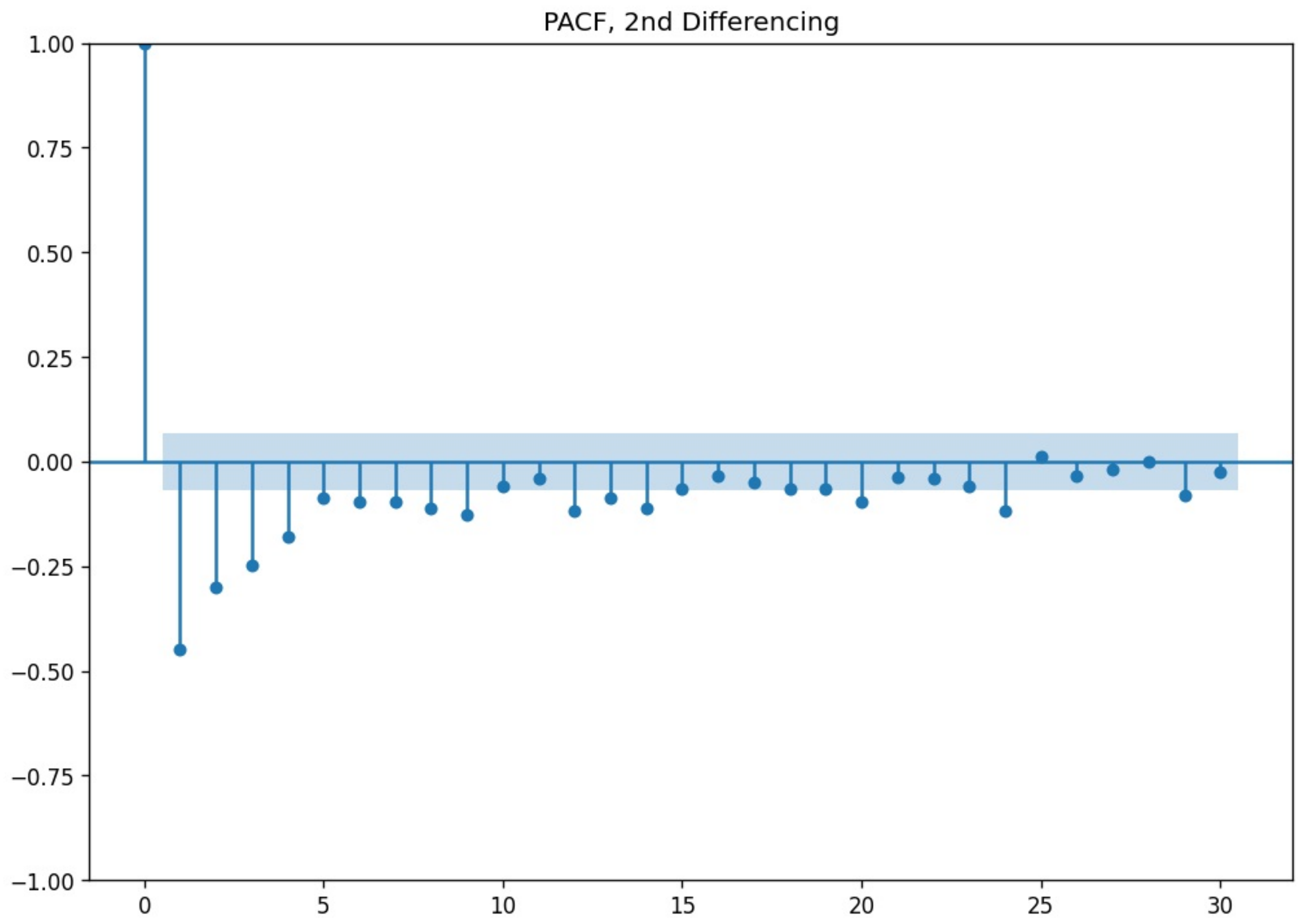
```
warnings.filterwarnings("ignore")
plot_pacf(df.dropna(), title='PACF, Original Time Series');
```



In [30]:
```
plot_pacf(df.diff().dropna(), title='PACF, 1st Differencing');
```

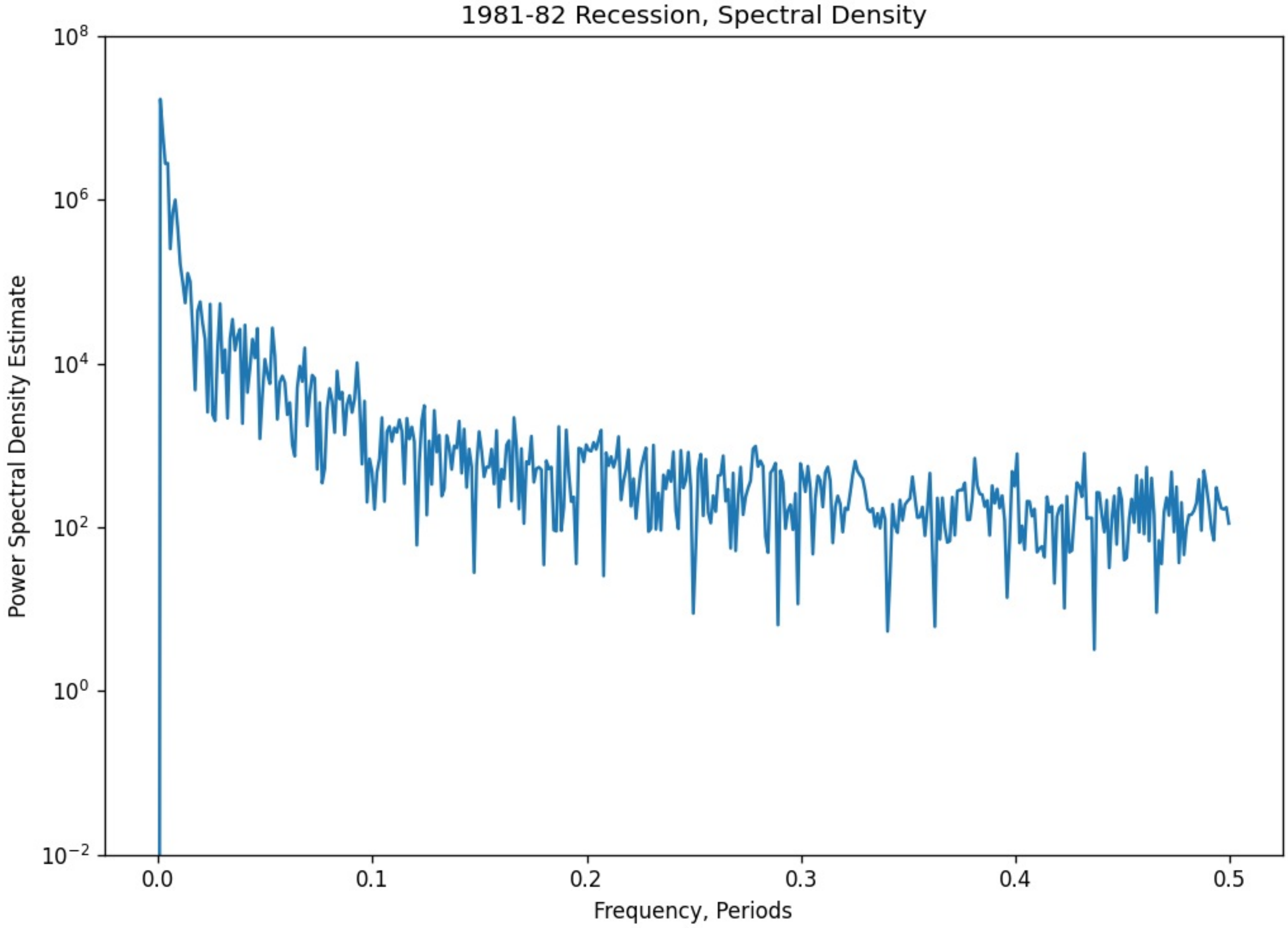


In [31]:
```
plot_pacf(df.diff().diff().dropna(), title='PACF, 2nd Differencing');
```



D7 Spectral Density

In [32]:
```
# Code Reference (Festus, 2022)

# signal periodogram
f, Pxx_den = signal.periodogram(df['2022_Value'])

# plotting semilogy - pyplot module used to make a plot with log scaling on the y-axis
plt.semilogy(f, Pxx_den)

# Setting coordinate values and titles for Spectral Density Graph
# setting y-axis min and max value
plt.ylim(1e-2, 1e8)

# Graph Title
plt.title('1981-82 Recession, Spectral Density')

# X label for Periods
plt.xlabel('Frequency, Periods')

# Y Label for SD Estimate
plt.ylabel('Power Spectral Density Estimate')
plt.show()
```



D8 Create Train/Test Datasets

Dataset Size = 861 cases

80/20 Train/Test Split

Split is 689 / 172

In [33]:
```
# ------Splitting data into Test and Train sets using pmdarima's train_test_split
# code reference (Smith, 2019)

train, test = train_test_split(df, train_size=689)
```

In [34]:
```
train
```

Out[34]:

Trading Days	2022_Value
1974-07-24	949.520
1974-07-25	938.730
1974-07-26	934.580
1974-07-27	940.390
1974-07-28	952.840
...	...
1976-06-07	444.318
1976-06-08	444.020
1976-06-09	432.994
1976-06-10	423.458
1976-06-11	422.564

689 rows × 1 columns

In [35]:
```
test
```

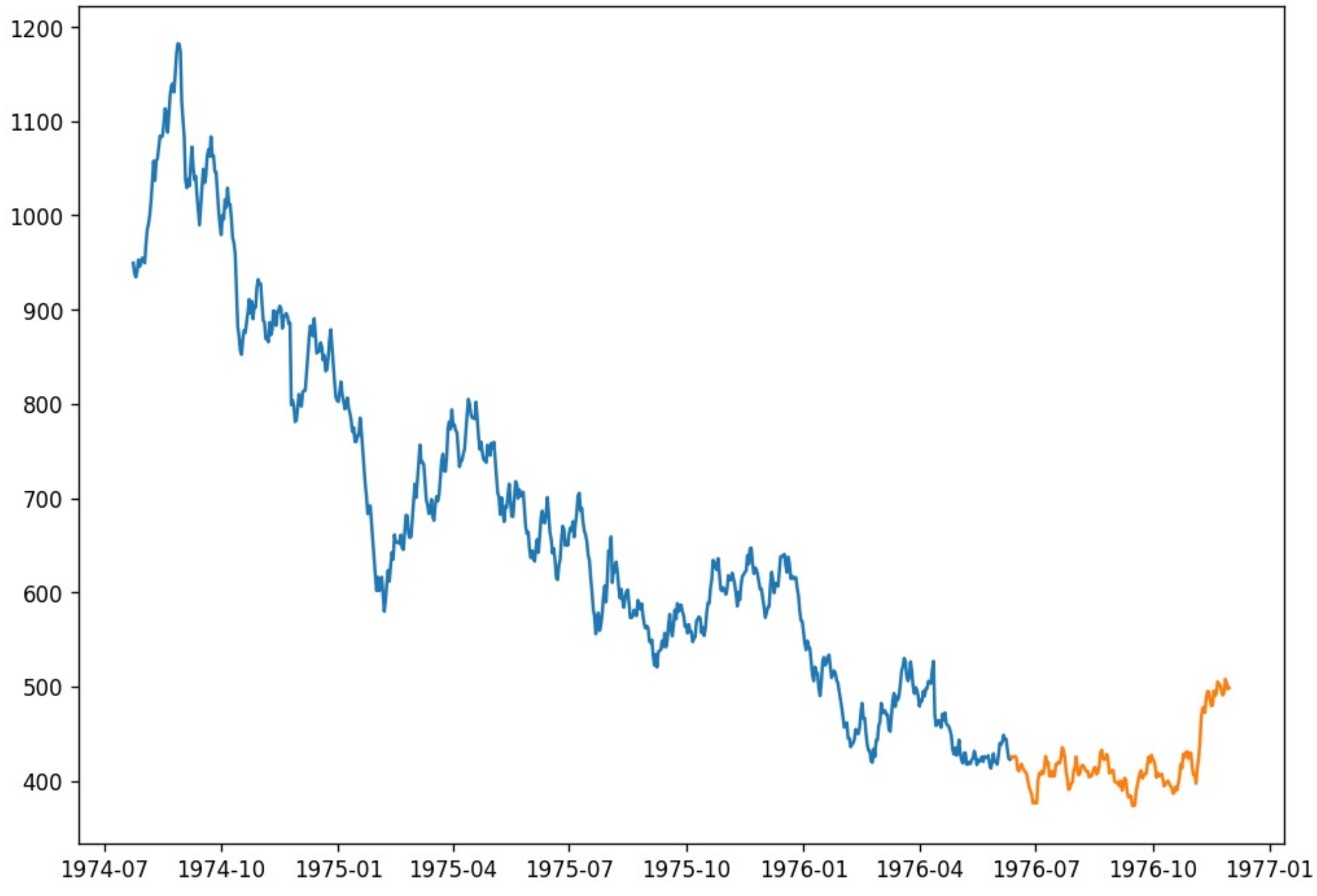
Out[35]:

Trading Days	2022_Value
1976-06-12	425.544
1976-06-13	424.352
1976-06-14	425.544
1976-06-15	426.140
1976-06-16	424.650
...	...
1976-11-26	493.190
1976-11-27	507.792
1976-11-28	504.216
1976-11-29	497.362
1976-11-30	498.554

172 rows × 1 columns

In [36]:
```
# Plot training data
plt.plot(train)

# Plot Test Data
plt.plot(test)
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x2464c2f4908>]
```



In [37]:
```
print(train.shape)
print(test.shape)
```

```
(689, 1)
(172, 1)
```

D9 Auto-arima ARIMA Modeling

Using pmdarima's auto_arima

In [38]:
```
# Fit the model using auto_arima
# Auto-arima code reference (6. Tips to using auto_arima — pmdarima 2.0.1 documentation, n.d.)
# Additional code reference (Pmdarima.arima.AutoARIMA — pmdarima 2.0.1 documentation, n.d)
# Auto-arima, initial parameter attempt
# Code Reference (Kosaka, 2021)

# Establish auto_arima to run ARIMA and take into account
# Any Seasonality of the data, and any trends found.
```

```python
model = auto_arima(train, start_p=1, start_q=1,
                   test='adf',
                   max_p=3,
                   max_q=3,
                   max_d=3,
                   seasonal=True,
                   stationarity=False,
                   seasonal_test='ocsb',
                   trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True,
                  trend='c')

# Print Summary of Best AIC Minimized SARIMAX Model
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=5462.961, Time=0.32 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=5473.627, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=5462.260, Time=0.14 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=5463.299, Time=0.25 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=5473.627, Time=0.04 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=5463.398, Time=0.25 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=5464.857, Time=0.29 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=5462.260, Time=0.15 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 1.482 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  689
Model:               SARIMAX(1, 1, 0)   Log Likelihood               -2728.130
Date:                Tue, 18 Oct 2022   AIC                           5462.260
Time:                        13:36:13   BIC                           5475.861
Sample:                    07-24-1974   HQIC                          5467.522
                         - 06-11-1976
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.6618      0.510     -1.298      0.194      -1.662       0.338
ar.L1          0.1387      0.035      3.959      0.000       0.070       0.207
sigma2       162.8281      5.731     28.410      0.000     151.595     174.061
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):               342.92
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               0.40   Skew:                            -0.68
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.18
===================================================================================

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

In [68]:

```python
model = auto_arima(train, trace=True)

# Print Summary of Best AIC Minimized SARIMAX Model
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=5463.210, Time=0.53 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=5473.627, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=5462.260, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=5463.299, Time=0.14 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=5474.054, Time=0.02 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=5463.398, Time=0.15 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=5462.961, Time=0.14 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=5464.857, Time=0.17 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=5462.103, Time=0.02 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=5463.118, Time=0.08 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=5462.566, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=5463.246, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=5464.447, Time=0.10 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 1.542 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  689
Model:               SARIMAX(1, 1, 0)   Log Likelihood               -2729.051
Date:                Tue, 18 Oct 2022   AIC                           5462.103
Time:                        21:10:31   BIC                           5471.170
Sample:                    07-24-1974   HQIC                          5465.610
                         - 06-11-1976
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1417      0.035      4.037      0.000       0.073       0.210
sigma2       163.3362      5.422     30.124      0.000     152.709     173.963
===================================================================================
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):               343.36
Prob(Q):                              0.84   Prob(JB):                         0.00
Heteroskedasticity (H):               0.41   Skew:                            -0.68
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.18
===================================================================================

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

In [40]:
```
model.conf_int()
```

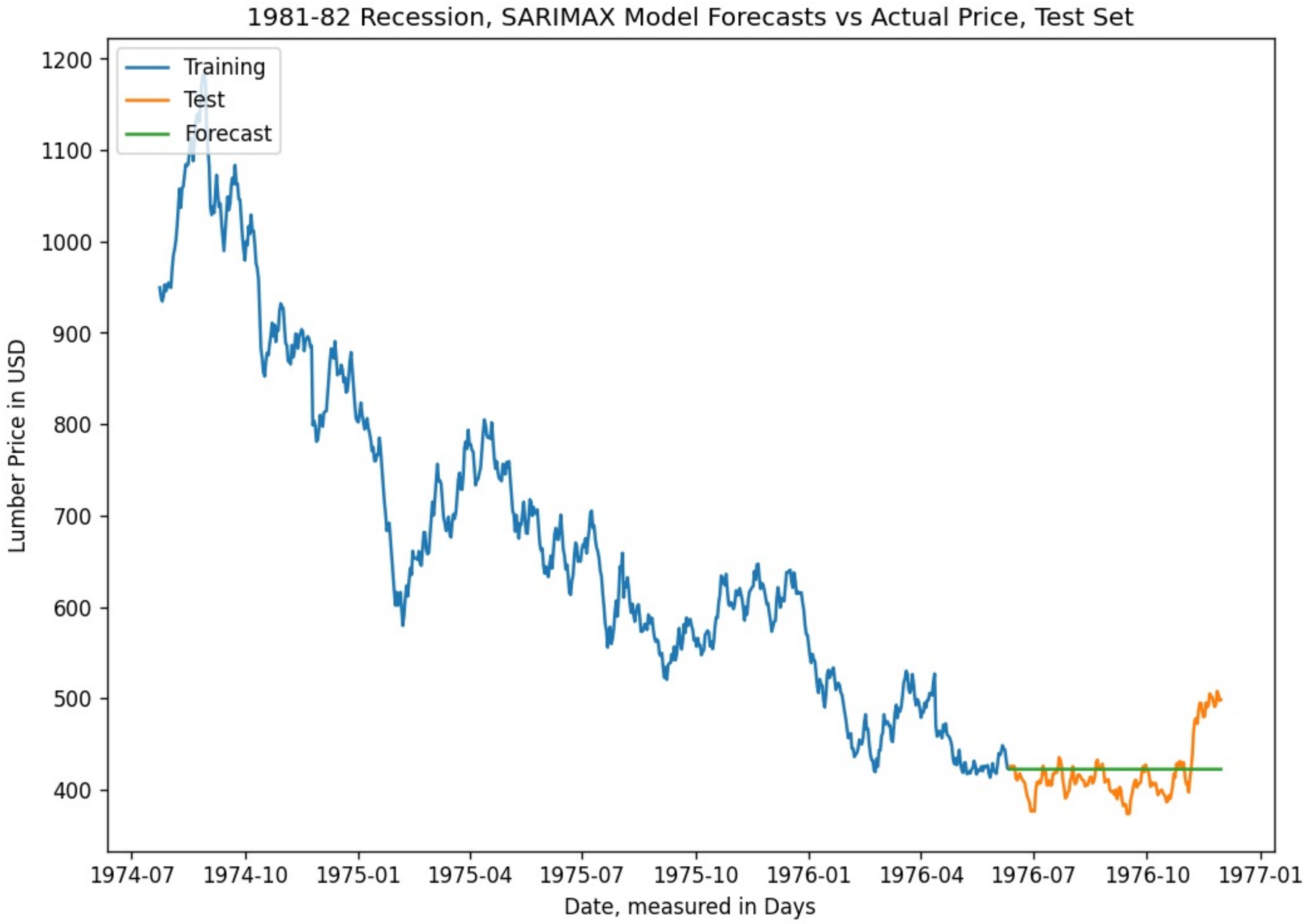
Out[40]:

	0	1
ar.L1	0.072884	0.210442
sigma2	152.709179	173.963304

Visualizing Model Results

In [41]:
```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation, n.d.)

# ------Creating varible with forecast values
forecast = pd.DataFrame(model.predict(n_periods = 172))

# Naming forecast_revenue column in forecast variable
forecast.columns = ['forecast_prices']

# Establish plot parameters for Forecast

# Plot figure size
plt.figure(figsize=(10,7))

# Training data
plt.plot(train,label="Training")

# Annotate X-axis label
plt.xlabel('Date, measured in Days')

# Annotate Y-axis label
plt.ylabel('Lumber Price in USD')

# Annotate Plot Title
plt.title('1981-82 Recession, SARIMAX Model Forecasts vs Actual Price, Test Set')

# Plot Test Data
plt.plot(test,label="Test")

# Plot Forecast Data
plt.plot(forecast,label="Forecast")

# Plot legend in upper lefthand corner
plt.legend(loc = 'upper left')
```

```
# Show Plot
plt.show()
```



In [42]:
```
forecast
```

Out[42]:

	forecast_prices
1976-06-12	422.437354
1976-06-13	422.419413
1976-06-14	422.416871
1976-06-15	422.416511
1976-06-16	422.416460
...	...
1976-11-26	422.416452
1976-11-27	422.416452
1976-11-28	422.416452
1976-11-29	422.416452
1976-11-30	422.416452

172 rows × 1 columns

D10 Accuracy Metrics for our forecast

In [43]:
```
# RMSE and MAE to test model accuracy
```

In [44]:
```
# Create array of actual Revenue values, stored in Test variable

test_array = test[['2022_Value']].to_numpy()
#test_array
```

In [45]:
```
test_array.shape
```

Out[45]:
```
(172, 1)
```

In [46]:
```
forecast
```

Out[46]:

	forecast_prices
1976-06-12	422.437354
1976-06-13	422.419413
1976-06-14	422.416871
1976-06-15	422.416511
1976-06-16	422.416460
...	...
1976-11-26	422.416452
1976-11-27	422.416452
1976-11-28	422.416452
1976-11-29	422.416452
1976-11-30	422.416452

172 rows × 1 columns

In [47]:

```
# Predictions to numpy array
predicted_array = forecast[['forecast_prices']].to_numpy()
```

In [48]:

```
predicted_array.shape
```

Out[48]:

```
(172, 1)
```

In [49]:

```
#RMSE Calculation

rmse = sqrt(mean_squared_error(test_array, predicted_array))
print ('RMSE = ' + str(rmse))
```

```
RMSE = 31.54283940539119
```

In [50]:

```
# MAE Calculation

def mae(y_true, predictions):
    y_true, predictions = np.array(y_true), np.array(predictions)
    return np.mean(np.abs(y_true - predictions))

true = test_array
predicted = predicted_array

print(mae(true, predicted))
```

```
23.29469340238698
```

D11 Visualizing Model Forecast Confidence Intervals at 20% CI

In [51]:

```
# Model Standard Error calculations, computed numerical Hessian

std_error = model.bse()
print(std_error)
```

```
ar.L1     0.035092
sigma2    5.422070
dtype: float64
```

In [52]:

```
# Generate Model confidence intervals

conf_int = model.conf_int()
```

In [53]:

```
# ------Generate Forecast Prediction Intervals at 90% Confidence

y_forec, conf_int = model.predict(172, return_conf_int=True, alpha=0.8)
print(conf_int)
```

```
[[419.1995     425.67520741]
 [417.50534342 427.33348193]
 [416.22840871 428.60533349]
 [415.16976069 429.66326142]
 [414.24678743 430.58613267]
 [413.41794716 431.41495849]
 [412.65925676 432.17364684]
 [411.95544626 432.87745705]
 [411.29609117 433.5368121 ]
 [410.67370067 434.15920259]
 [410.08267753 434.75022574]
 [409.51870892 435.31419435]
 [408.97838813 435.85451513]
 [408.45896855 436.37393472]
 [407.95819733 436.87470594]
 [407.47419943 437.35870384]
```

```
[407.00539442 437.82750885]
[406.5504355  438.28246777]
[406.10816382 438.72473945]
[405.67757369 439.15532958]
[405.25778565 439.57511762]
[404.84802535 439.98487791]
[404.44760678 440.38529648]
[404.05591872 440.77698454]
[403.67241385 441.16048941]
[403.29659976 441.53630351]
[402.9280315  441.90487176]
[402.56630547 442.2665978 ]
[402.21105418 442.62184908]
[401.86194192 442.97096135]
[401.518661   443.31424226]
[401.18092864 443.65197462]
[400.84848418 443.98441909]
[400.52108674 444.31181652]
[400.19851323 444.63439004]
[399.88055648 444.95234679]
[399.56702376 445.2658795 ]
[399.2577354  445.57516786]
[398.95252355 445.88037971]
[398.65123115 446.18167211]
[398.35371098 446.47919229]
[398.05982478 446.77307848]
[397.76944259 447.06346068]
[397.48244196 447.3504613 ]
[397.19870745 447.63419582]
[396.91813002 447.91477325]
[396.64060657 448.1922967 ]
[396.3660395  448.46686376]
[396.0943363  448.73856697]
[395.82540917 449.00749409]
[395.55917473 449.27372854]
[395.29555367 449.53734959]
[395.03447052 449.79843274]
[394.77585336 450.05704991]
[394.5196336  450.31326967]
[394.26574578 450.56715749]
[394.01412737 450.81877589]
[393.76471858 451.06818469]
[393.5174622  451.31544107]
[393.27230344 451.56059983]
[393.02918981 451.80371345]
[392.78807097 452.0448323 ]
[392.5488986  452.28400467]
[392.31162631 452.52127696]
[392.07620952 452.75669374]
[391.84260538 452.99029789]
[391.61077264 453.22213063]
[391.38067161 453.45223166]
[391.15226405 453.68063921]
[390.92551311 453.90739015]
[390.70038327 454.13251999]
[390.47684024 454.35606302]
[390.25485094 454.57805232]
[390.03438341 454.79851986]
[389.81540677 455.01749649]
[389.59789119 455.23501207]
[389.3818078  455.45109546]
[389.16712868 455.66577458]
[388.95382681 455.87907645]
[388.74187601 456.09102726]
[388.53125093 456.30165234]
[388.32192699 456.51097627]
[388.11388039 456.71902287]
[387.90708801 456.92581525]
[387.70152745 457.13137581]
[387.49717695 457.33572631]
[387.29401539 457.53888788]
[387.09202225 457.74088102]
[386.8911776  457.94172566]
[386.69146208 458.14144118]
[386.49285686 458.34004641]
[386.2953436  458.53755966]
[386.09890451 458.73399875]
[385.90352225 458.92938101]
[385.70917993 459.12372333]
[385.51586113 459.31704213]
[385.32354985 459.50935341]
[385.13223049 459.70067277]
[384.94188787 459.8910154 ]
[384.75250716 460.0803961 ]
[384.56407395 460.26882932]
[384.37657414 460.45632913]
[384.189994   460.64290927]
[384.00432013 460.82858314]
[383.81953945 461.01336382]
```

```
 [383.63563919 461.19726408]
 [383.45260689 461.38029638]
 [383.27043037 461.5624729 ]
 [383.08909774 461.74380553]
 [382.90859738 461.92430589]
 [382.72891793 462.10398533]
 [382.5500483  462.28285496]
 [382.37197763 462.46092563]
 [382.19469531 462.63820795]
 [382.01819096 462.8147123 ]
 [381.84245443 462.99044883]
 [381.66747579 463.16542747]
 [381.49324531 463.33965795]
 [381.31975348 463.51314978]
 [381.14699098 463.68591228]
 [380.97494869 463.85795457]
 [380.80361768 464.02928558]
 [380.6329892  464.19991407]
 [380.46305467 464.36984859]
 [380.29380571 464.53909756]
 [380.12523406 464.7076692 ]
 [379.95733168 464.87557158]
 [379.79009065 465.04281261]
 [379.62350322 465.20940004]
 [379.45756178 465.37534148]
 [379.29225888 465.54064439]
 [379.1275872  465.70531607]
 [378.96353956 465.86936371]
 [378.80010893 466.03279434]
 [378.63728838 466.19561488]
 [378.47507116 466.35783211]
 [378.31345058 466.51945268]
 [378.15242013 466.68048314]
 [377.99197337 466.84092989]
 [377.83210402 467.00079924]
 [377.67280588 467.16009739]
 [377.51407286 467.3188304 ]
 [377.35589901 467.47700425]
 [377.19827845 467.63462481]
 [377.04120541 467.79169785]
 [376.88467424 467.94822903]
 [376.72867935 468.10422392]
 [376.57321527 468.25968799]
 [376.41827663 468.41462663]
 [376.26385813 468.56904513]
 [376.10995457 468.72294869]
 [375.95656083 468.87634244]
 [375.80367187 469.02923139]
 [375.65128275 469.18162052]
 [375.49938859 469.33351467]
 [375.34798461 469.48491865]
 [375.19706609 469.63583718]
 [375.04662838 469.78627488]
 [374.89666693 469.93623633]
 [374.74717724 470.08572603]
 [374.59815487 470.23474839]
 [374.44959549 470.38330778]
 [374.30149479 470.53140847]
 [374.15384856 470.67905471]
 [374.00665264 470.82625063]
 [373.85990292 470.97300034]
 [373.71359539 471.11930787]
 [373.56772607 471.2651772 ]
 [373.42229103 471.41061223]
 [373.27728643 471.55561683]
 [373.13270847 471.70019479]
 [372.9885534  471.84434986]]
```

In [54]:

```
# Assign Predictions to pandas DataFrame

conf_pd = pd.DataFrame(conf_int, columns =['Low_Prediction','High_Prediction'])

#Assign Low predictions to variable
low_prediction = conf_pd['Low_Prediction']

#Assign High predictions to variable
high_prediction = conf_pd['High_Prediction']
```

In [ ]:

In [55]:

```
# Read out Test and Train sets to csv file
# Open csv files in Google Sheets, Add Day Column
# Dates align with 'test' variable, which contains actual revenue figures

low_prediction.to_csv('C:/Users/ericy/Desktop/Low_Prediction.csv')
high_prediction.to_csv('C:/Users/ericy/Desktop/High_Prediction.csv')
```

In [56]:

```
#-----Load predictions, date column added
```

```
low_pred = pd.read_csv('C:/Users/ericy/Desktop/1981-1982_Low_Prediction.csv')
high_pred = pd.read_csv('C:/Users/ericy/Desktop/1981-1982_High_Prediction.csv')
```

In [57]:
```
# Variable exploration to ensure compatability with 'test' datetime timeframe
low_pred
```

Out[57]:

	Date	Low_Prediction
0	1976-06-12	419.199500
1	1976-06-13	417.505343
2	1976-06-14	416.228409
3	1976-06-15	415.169761
4	1976-06-16	414.246787
...	...	...
167	1976-11-26	373.567726
168	1976-11-27	373.422291
169	1976-11-28	373.277286
170	1976-11-29	373.132708
171	1976-11-30	372.988553

172 rows × 2 columns

In [58]:
```
# Variable exploration to ensure compatability with 'test' datetime timeframe
high_pred
```

Out[58]:

	Date	High_Prediction
0	1976-06-12	425.675207
1	1976-06-13	427.333482
2	1976-06-14	428.605333
3	1976-06-15	429.663261
4	1976-06-16	430.586133
...	...	...
167	1976-11-26	471.265177
168	1976-11-27	471.410612
169	1976-11-28	471.555617
170	1976-11-29	471.700195
171	1976-11-30	471.844350

172 rows × 2 columns

Convert Low and High Prediction 'Day' column to datetime and index

In [59]:
```
# Lower Predictions, Set Day as Index
low_pred['Date'] = pd.to_datetime(low_pred['Date'])
```

In [60]:
```
low_pred.set_index('Date',inplace=True)
```

In [61]:
```
# High Predictions, Day to datetime
high_pred['Date'] = pd.to_datetime(high_pred['Date'])
```

In [62]:
```
# High Predictions, Set Day as Index
high_pred.set_index('Date',inplace=True)
```

In [63]:
```
low_pred
```

Out[63]:

Date	Low_Prediction
1976-06-12	419.199500
1976-06-13	417.505343
1976-06-14	416.228409
1976-06-15	415.169761
1976-06-16	414.246787
...	...
1976-11-26	373.567726
1976-11-27	373.422291
1976-11-28	373.277286
1976-11-29	373.132708
1976-11-30	372.988553

172 rows × 1 columns

In [64]:

```
high_pred
```

Out[64]:

Date	High_Prediction
1976-06-12	425.675207
1976-06-13	427.333482
1976-06-14	428.605333
1976-06-15	429.663261
1976-06-16	430.586133
...	...
1976-11-26	471.265177
1976-11-27	471.410612
1976-11-28	471.555617
1976-11-29	471.700195
1976-11-30	471.844350

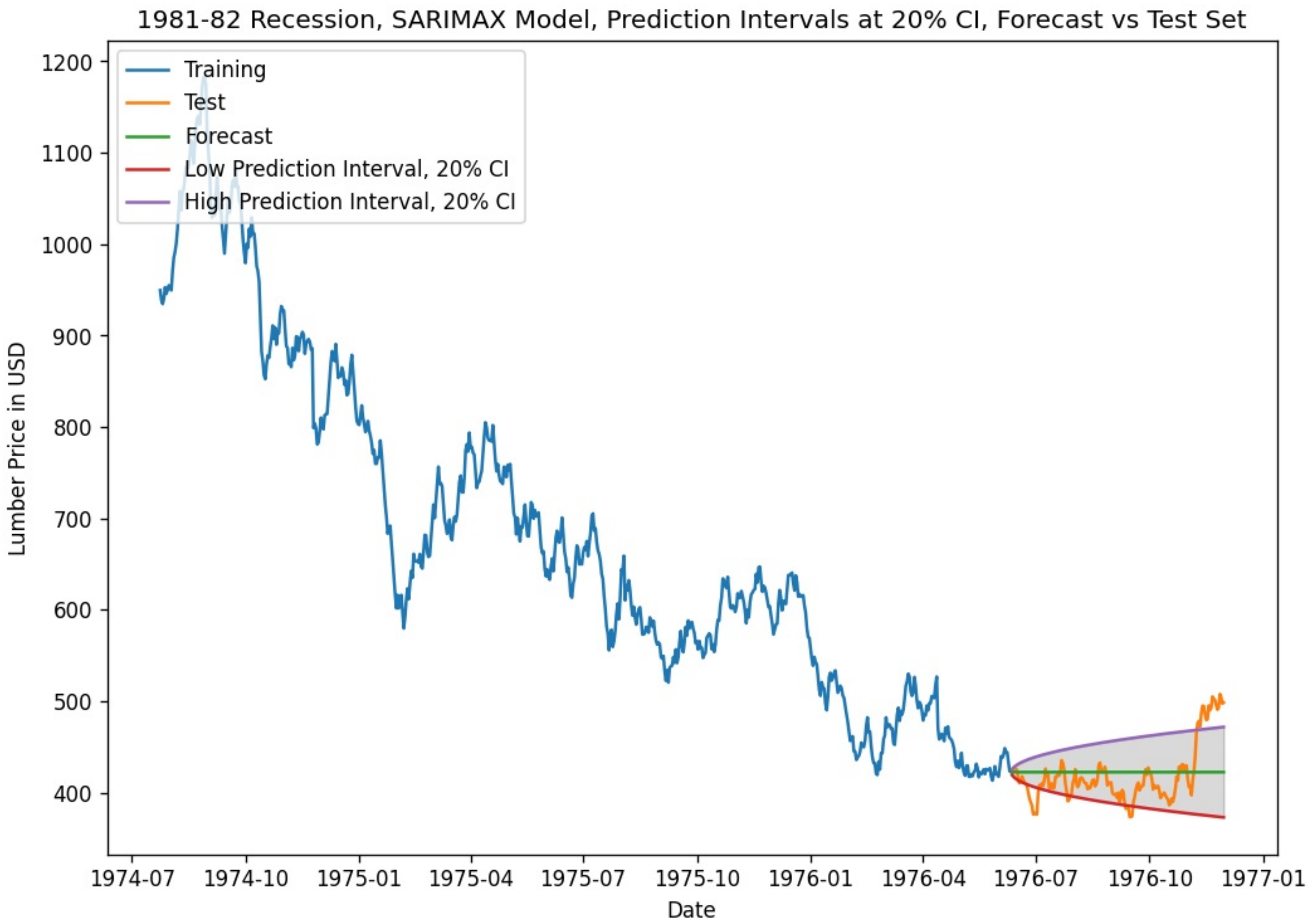
172 rows × 1 columns

SARIMAX Model Forecast, With Confidence Interval = 20%, Vs Test Set

In [65]:

```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation, n.d.)

# -----Creating varible with forecast values
forecast = pd.DataFrame(model.predict(n_periods = 172),index=test.index)

# Naming forecast_revenue column in forecast variable
forecast.columns = ['forecast_prices']

# Establish plot parameters for Forecast

# Plot figure size
plt.figure(figsize=(10,7))

# Training data
plt.plot(train,label="Training")

# Annotate X-axis label
plt.xlabel('Date')

# Annotate Y-axis label
plt.ylabel('Lumber Price in USD')

# Annotate Plot Title
plt.title('1981-82 Recession, SARIMAX Model, Prediction Intervals at 20% CI, Forecast vs Test Set')

# Plot Test Data
plt.plot(test,label="Test")
```

```python
# Plot Forecast Data
plt.plot(forecast,label="Forecast")

# Add Prediction Interval at 95% CI
plt.plot(low_pred,label='Low Prediction Interval, 20% CI')
plt.plot(high_pred,label='High Prediction Interval, 20% CI')
plt.fill_between(low_pred.index, low_pred['Low_Prediction'], high_pred['High_Prediction'], color='k', alpha=.15

# Plot legend in upper lefthand corner
plt.legend(loc = 'upper left')


# Show Plot
plt.show()
```



Is the null hypothesis Accepted or Rejected?

In [66]:
```python
# Accept or reject the Null Hypothesis
# 1981-1982 Recession we Accept the Null Hypothesis
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js