D214 Capstone

Modeling Inflation Adusted Recessionary Lumber Prices

January 1980- July 1980 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')
```

November 16 1973 to March 31, 1975

In [5]:
```
#------ Select Data Set for Recession
df = df[1288:1937]
```

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

Out[6]:

	Date	Trading Days	2022_Value	Value
1288	1978-01-05	1289	965.967	216.1
1289	1978-01-06	1290	958.368	214.4
1290	1978-01-09	1291	936.018	209.4
1291	1978-01-10	1292	923.502	206.6
1292	1978-01-11	1293	928.866	207.8
...	...	...	...	...
1932	1980-07-28	1933	784.046	210.2
1933	1980-07-29	1934	801.950	215.0
1934	1980-07-30	1935	783.300	210.0
1935	1980-07-31	1936	764.650	205.0
1936	1980-08-01	1937	751.595	201.5

649 rows × 4 columns

D1: Exploratory Data Analysis

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

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

Out[8]:

	Trading Days	2022_Value
1288	1289	965.967
1289	1290	958.368
1290	1291	936.018
1291	1292	923.502
1292	1293	928.866
...	...	...
1932	1933	784.046
1933	1934	801.950
1934	1935	783.300
1935	1936	764.650
1936	1937	751.595

649 rows × 2 columns

EDA

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

Out[9]:

	Trading Days	2022_Value
1288	1289	965.967
1289	1290	958.368
1290	1291	936.018
1291	1292	923.502
1292	1293	928.866

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 1288 to 1936
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Trading Days  649 non-null    int64  
 1   2022_Value    649 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 10.3 KB
```

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

Out[11]:
```
(649, 2)
```

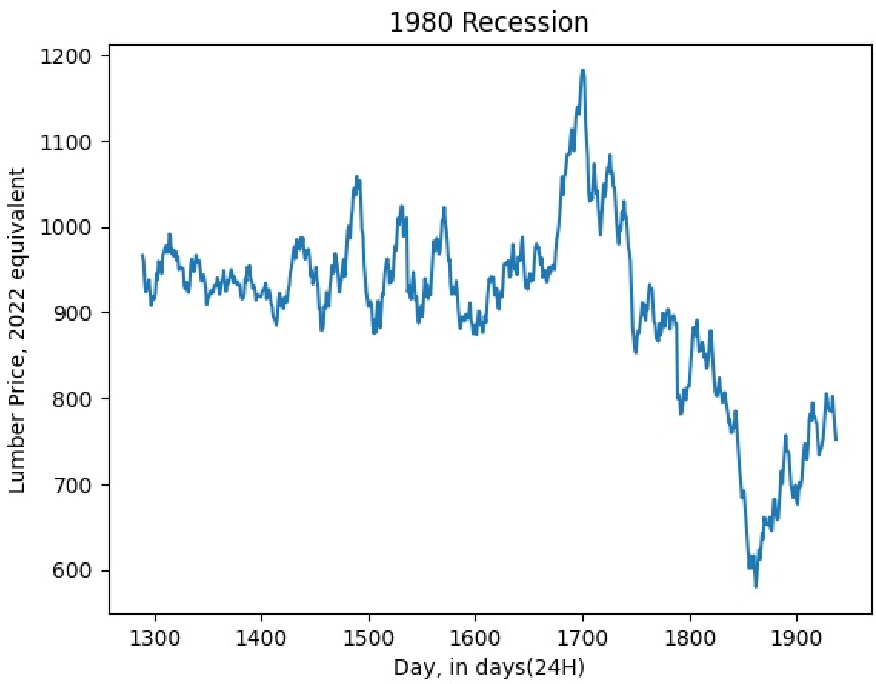
In [12]:
```
df.describe()
```
Out[12]:

	Trading Days	2022_Value
count	649.000000	649.000000
mean	1613.000000	905.570802
std	187.494444	106.342252
min	1289.000000	579.642000
25%	1451.000000	878.788000
50%	1613.000000	926.695000
75%	1775.000000	961.944000
max	1937.000000	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('1980 Recession')
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: 649 entries, 1288 to 1936
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Trading Days  649 non-null    datetime64[ns]
 1   2022_Value    649 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 15.2 KB
```

In [18]:
```
# Set Day as Index
df.set_index('Trading Days',inplace=True)
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 649 entries, 1973-07-13 to 1975-04-22
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   2022_Value  649 non-null    float64
dtypes: float64(1)
memory usage: 10.1 KB
```

In [20]:
```
df
```

Out[20]:

	2022_Value
Trading Days	
1973-07-13	965.967
1973-07-14	958.368
1973-07-15	936.018
1973-07-16	923.502
1973-07-17	928.866
...	...
1975-04-18	784.046
1975-04-19	801.950
1975-04-20	783.300
1975-04-21	764.650
1975-04-22	751.595

649 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]:
```
(-1.8123856973437957,
 0.3743301074911876,
 3,
 645,
 {'1%': -3.4405290941696722,
  '5%': -2.8660314117601575,
  '10%': -2.569161868277147},
 5094.404131203963)
```

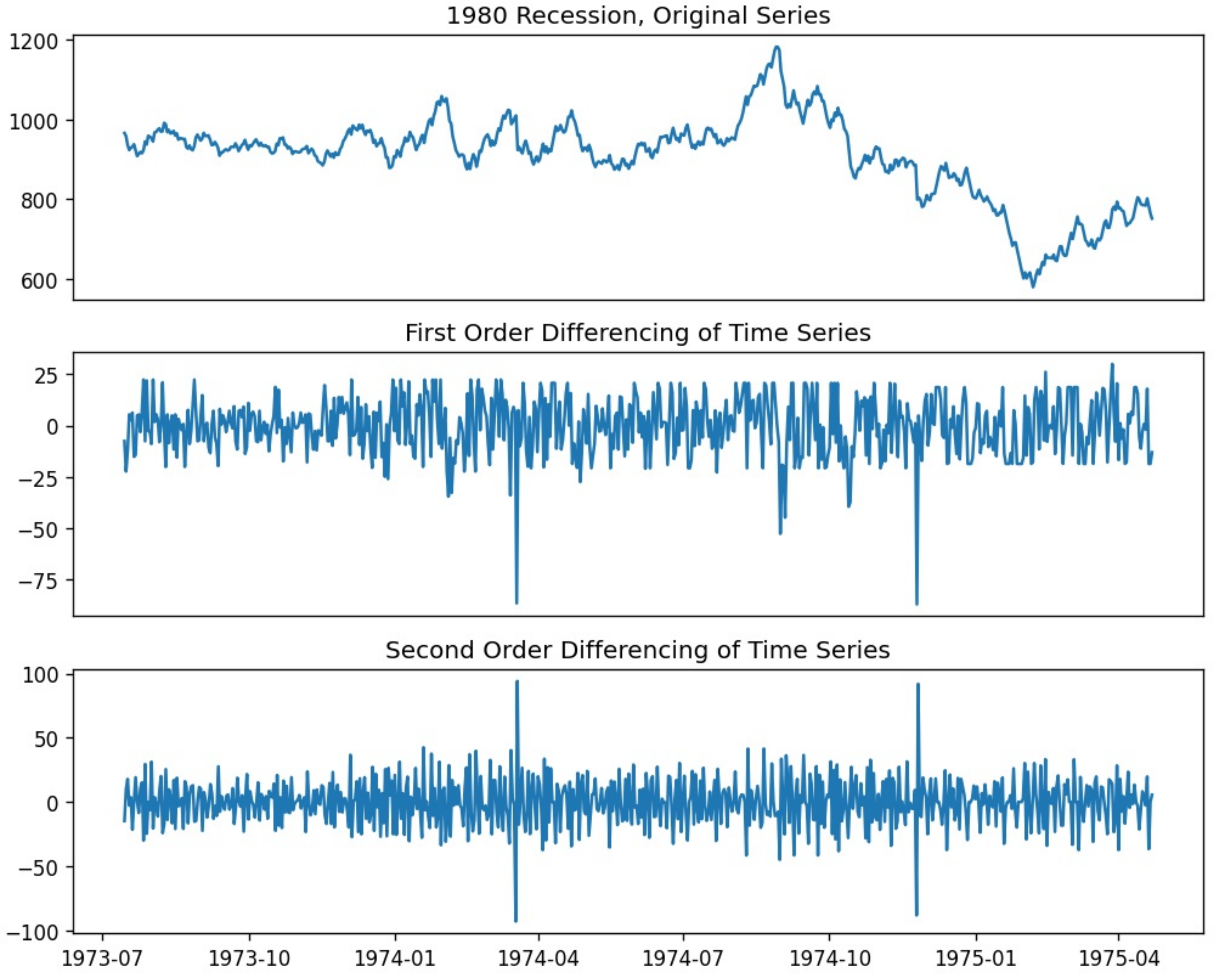
In [23]:
```
# Results show p = .37433
# Data does not reject null hypothesis at p < .05
# Therefore, Time series is determined to be non-stationary
```

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('1980 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()
```



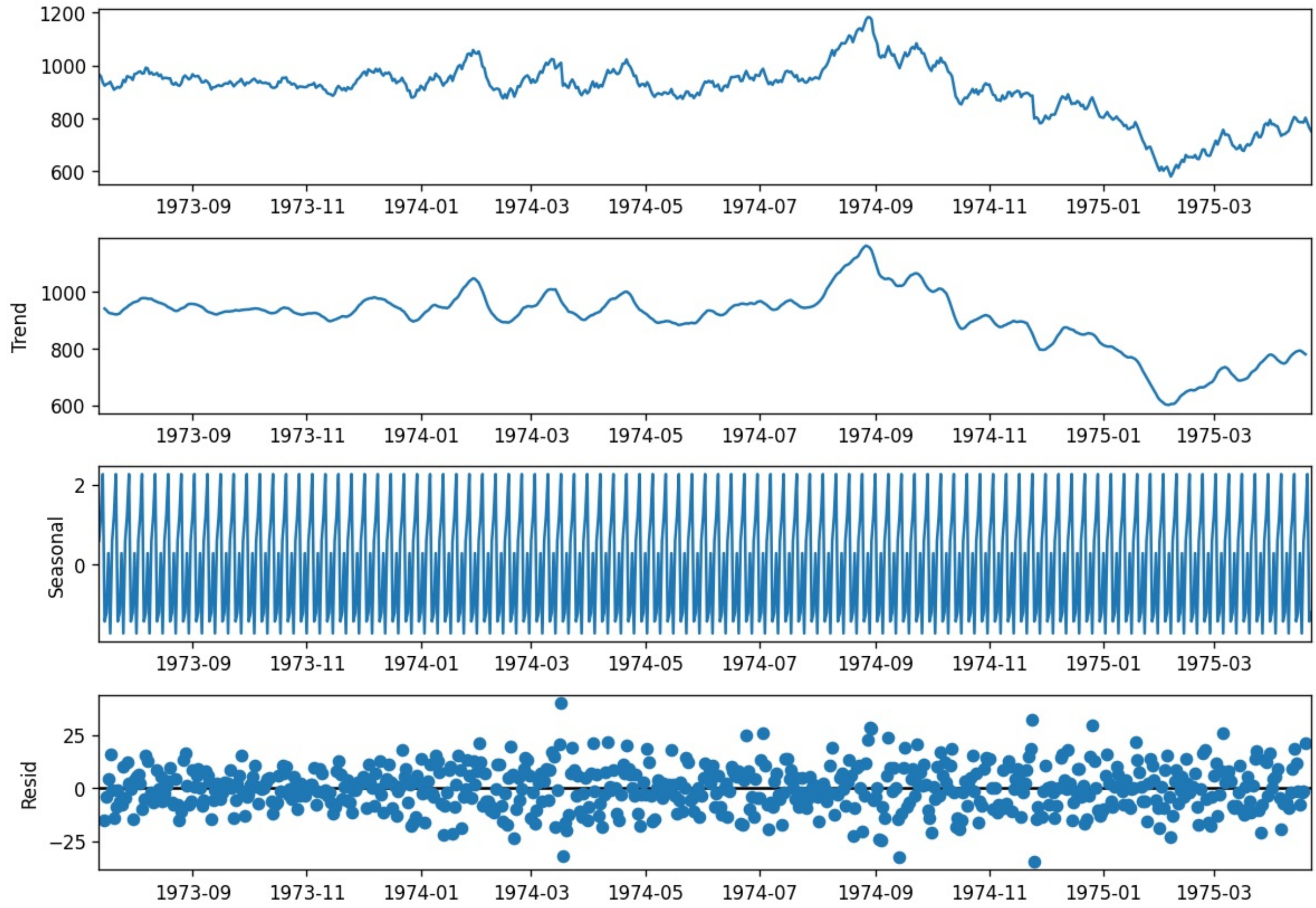
In [25]:
```
# Using pmdarima's ndiffs to find differencing term
# Code reference (Verma, 2021)

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();
```



In [ ]:

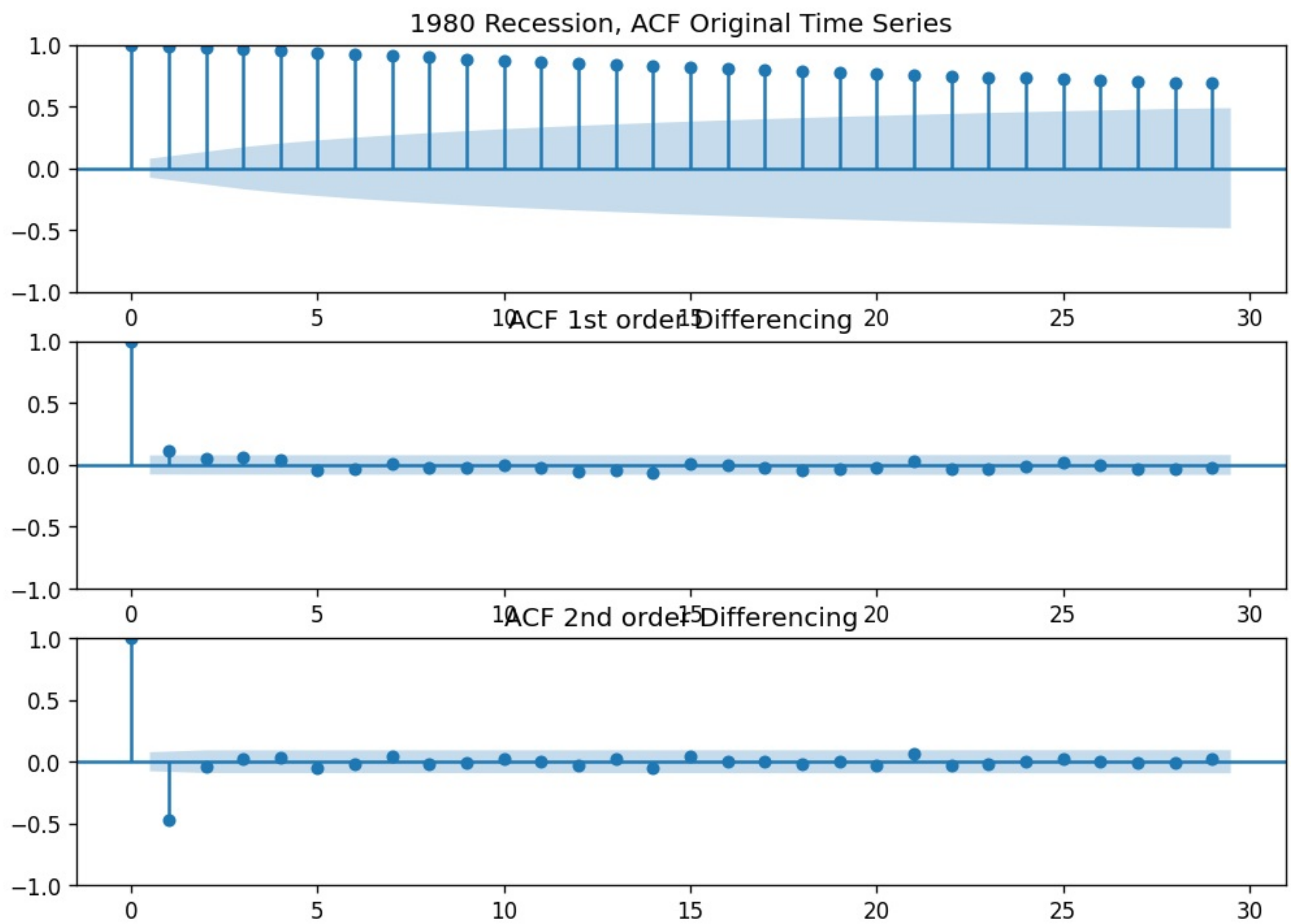
D6 ACF and PACF

Finding order of MA term 'q'

Using Autocorrelation function (ACF)

In [28]:

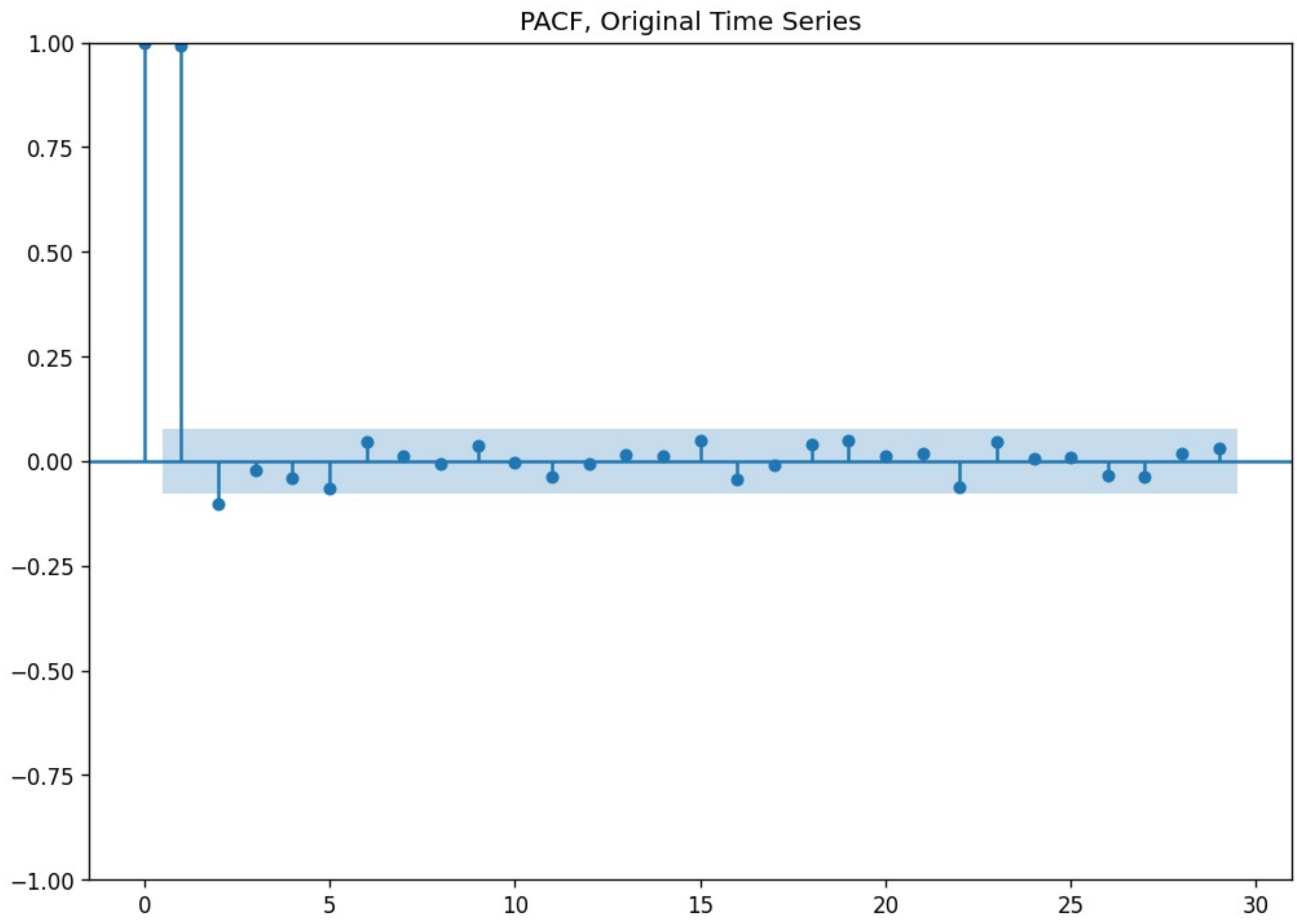
```
fig, (ax1, ax2, ax3) = plt.subplots(3)
plot_acf(df, ax=ax1, title='1980 Recession, ACF Original Time Series');
plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```



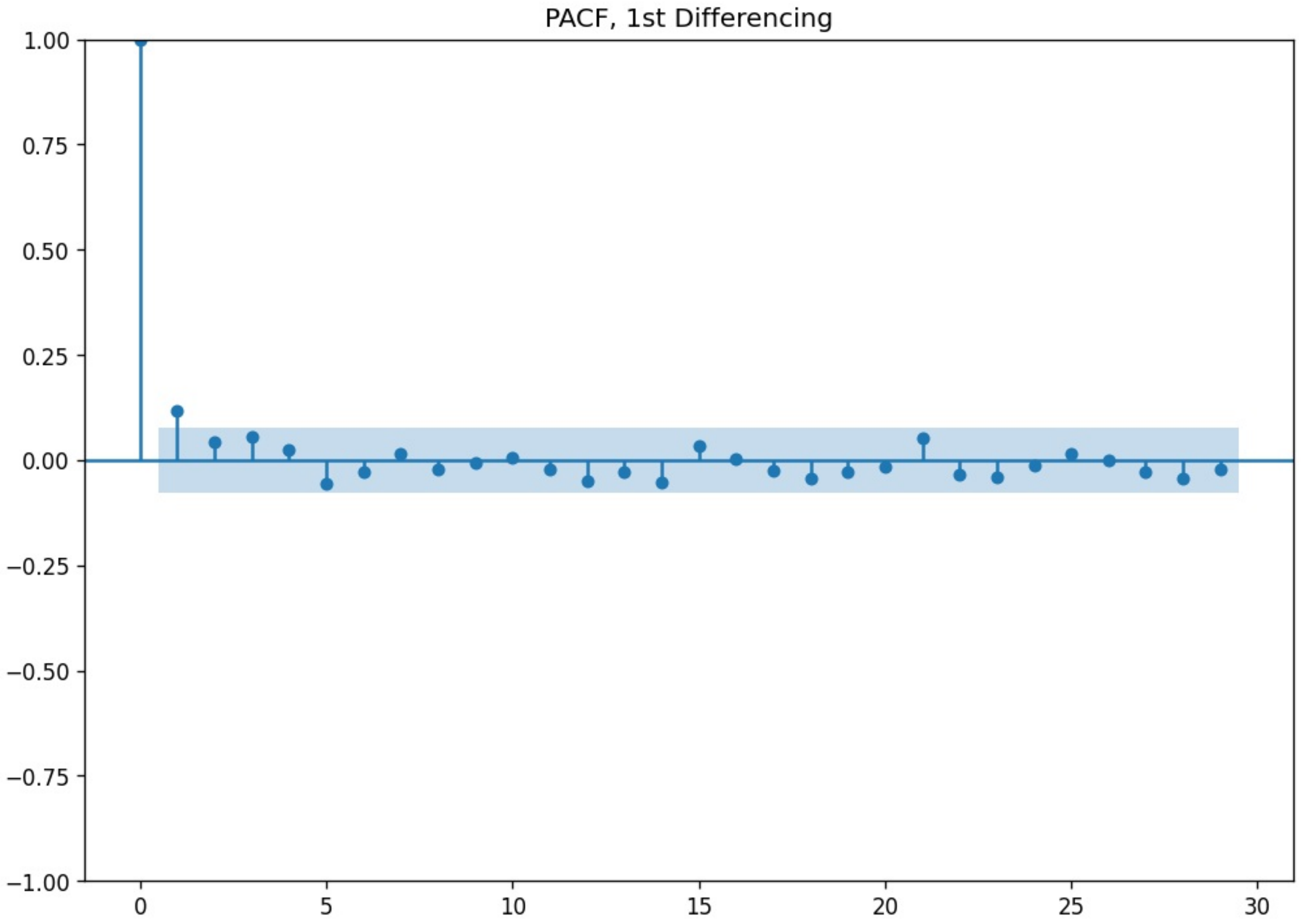
Finding order of AR term 'p'

Using Partial autocorrelation (PACF)

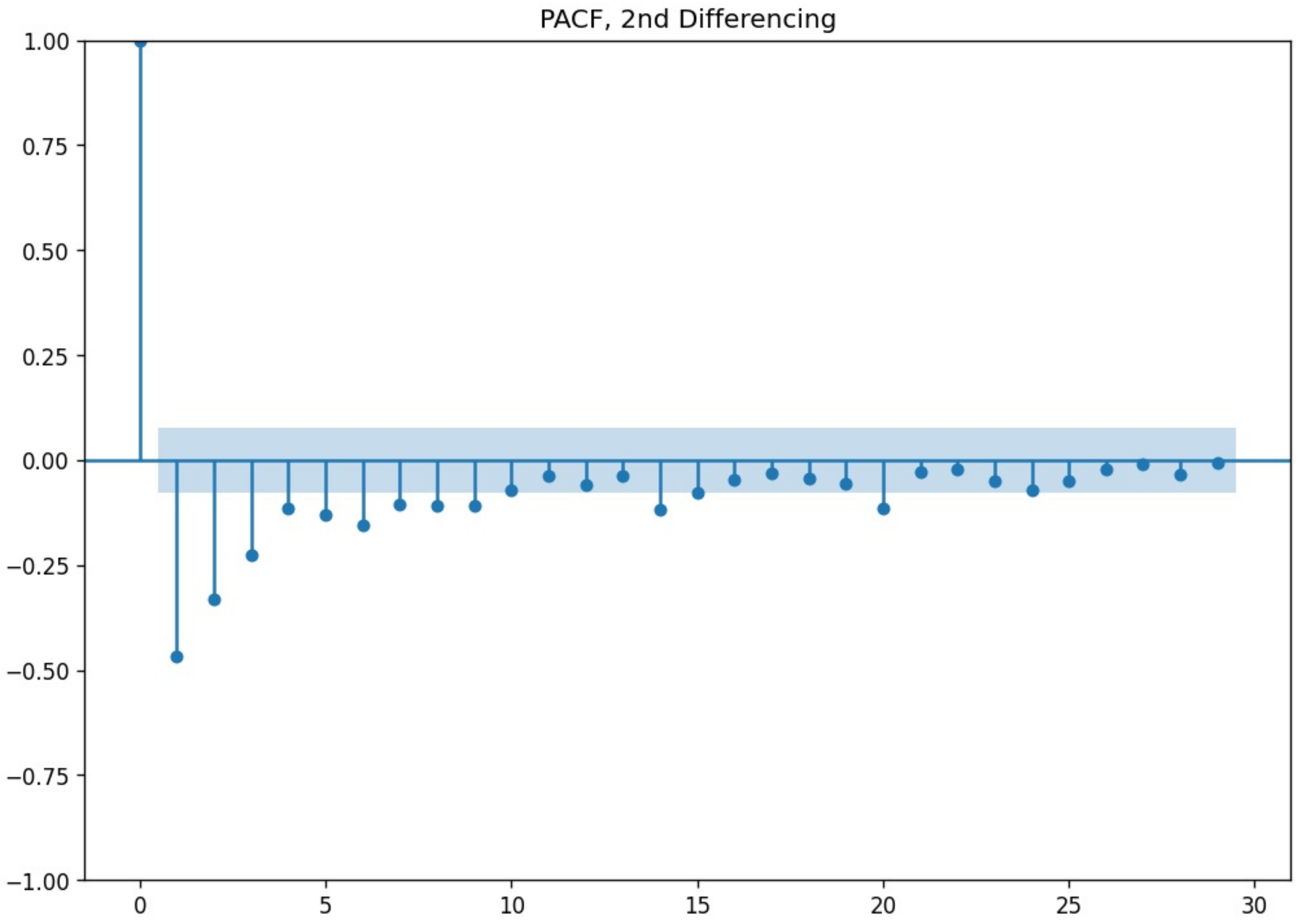
In [29]:
```
warnings.filterwarnings("ignore")
plot_pacf(df.dropna(), title='PACF, Original Time Series');
```



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

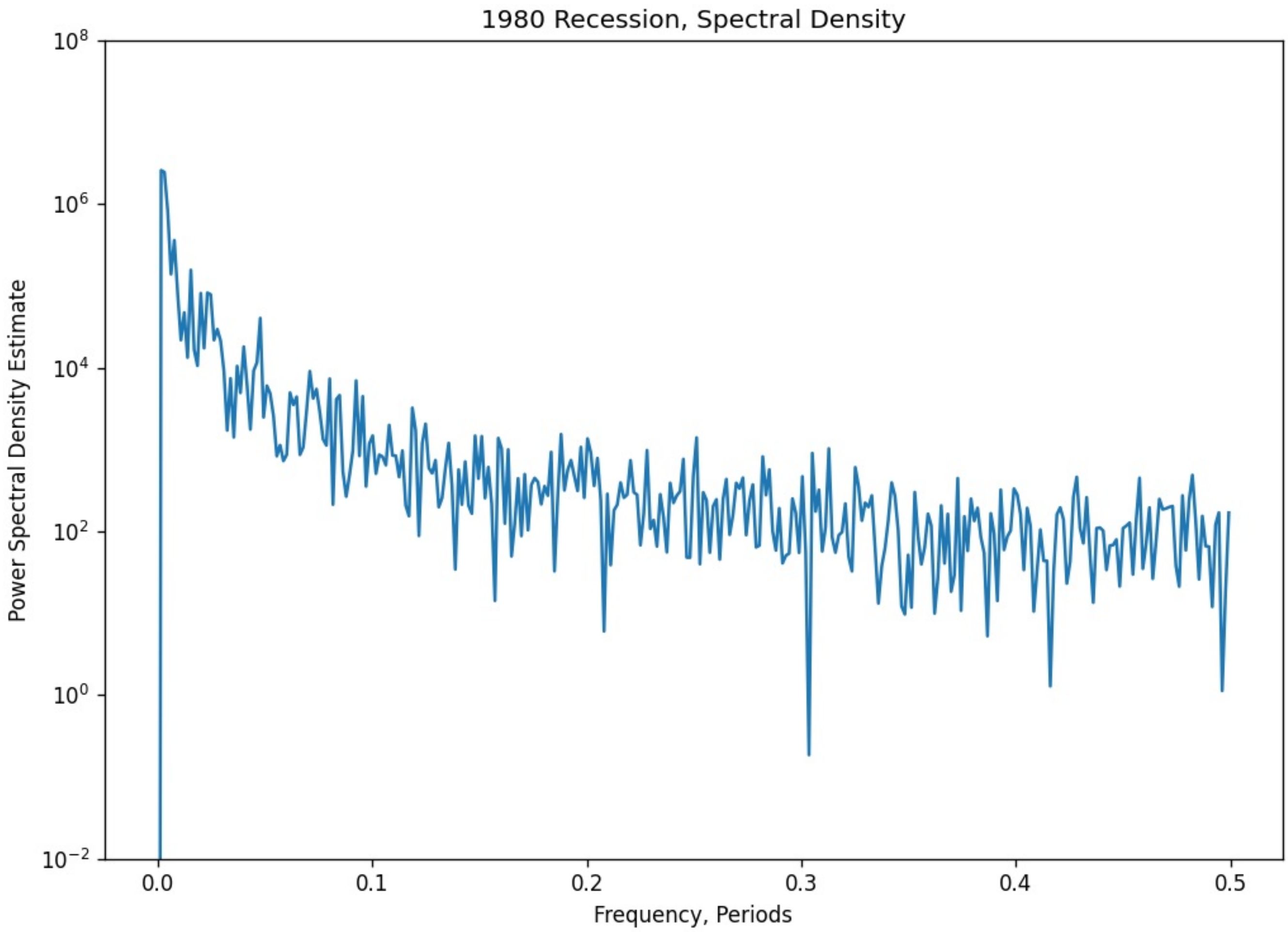


In [31]:
```
plot_pacf(df.diff().diff().dropna(), title='PACF, 2nd Differencing');
```



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('1980 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 = 649 cases

80/20 Train/Test Split

Split is 519 / 130

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=519)
```

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

Out[34]:

Trading Days	2022_Value
1973-07-13	965.967
1973-07-14	958.368
1973-07-15	936.018
1973-07-16	923.502
1973-07-17	928.866
...	...
1974-12-09	870.209
1974-12-10	882.891
1974-12-11	878.042
1974-12-12	872.074
1974-12-13	890.724

519 rows × 1 columns

In [35]:

```
test
```

Out[35]:

Trading Days	2022_Value
1974-12-14	872.074
1974-12-15	853.797
1974-12-16	856.781
1974-12-17	855.662
1974-12-18	864.987
...	...
1975-04-18	784.046
1975-04-19	801.950
1975-04-20	783.300
1975-04-21	764.650
1975-04-22	751.595

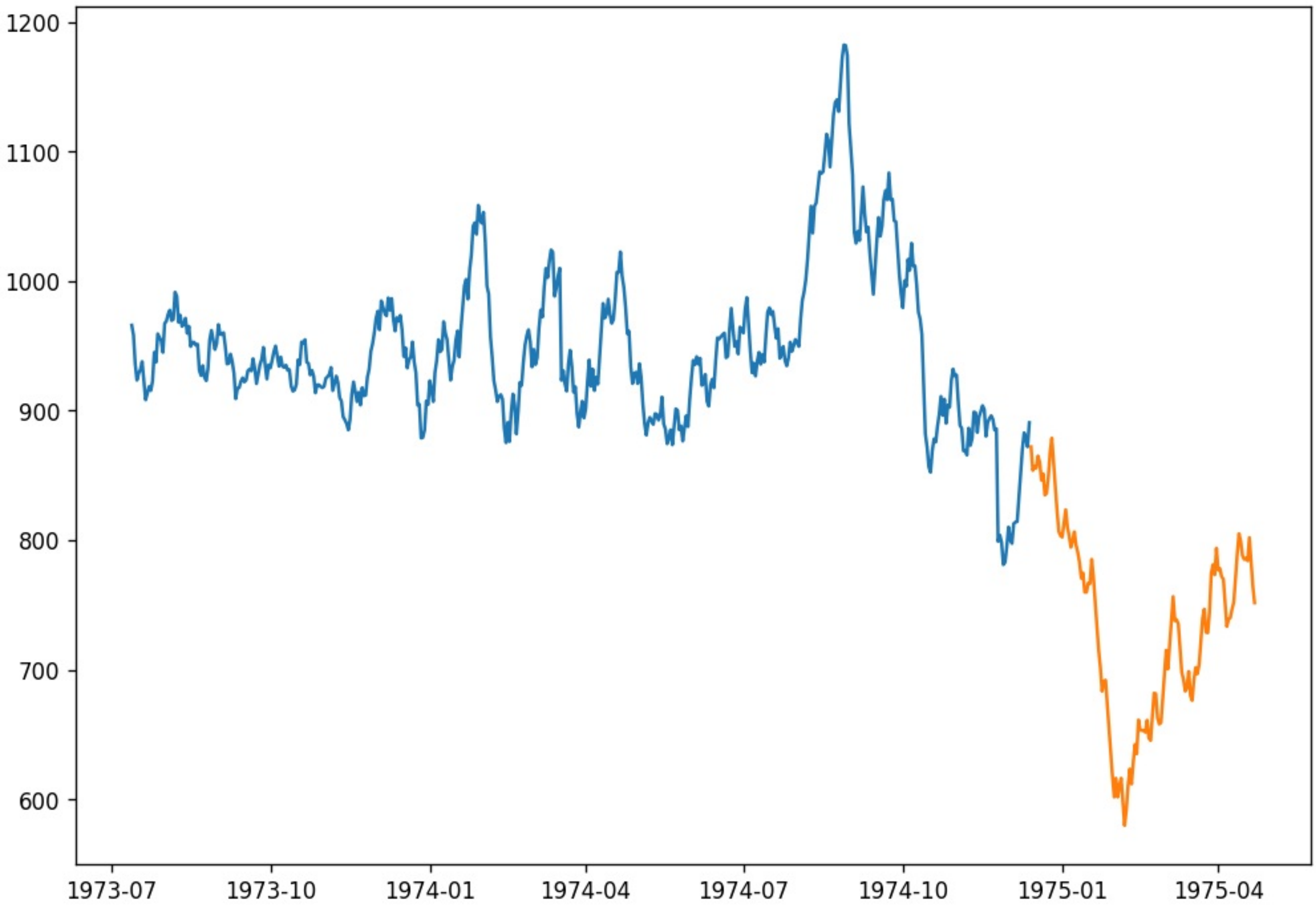
130 rows × 1 columns

In [36]:

```
# Plot training data
plt.plot(train)

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

Out[36]:

```
[<matplotlib.lines.Line2D at 0x1bdd8a76b88>]
```



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

```
(519, 1)
(130, 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
```

```python
# Any Seasonality of the data, and any trends found.
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=4214.746, Time=0.33 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=4217.906, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=4215.865, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=4216.276, Time=0.10 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4217.906, Time=0.01 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=4216.346, Time=0.42 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=4216.228, Time=0.20 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=4216.925, Time=0.21 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=4216.050, Time=0.14 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=4214.481, Time=0.70 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=4215.432, Time=0.49 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=4215.357, Time=0.54 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=4215.983, Time=0.30 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=4216.812, Time=0.20 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.86 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=4214.481, Time=0.63 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 5.199 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  519
Model:               SARIMAX(2, 1, 2)   Log Likelihood               -2101.241
Date:                Tue, 18 Oct 2022   AIC                           4214.481
Time:                        13:37:04   BIC                           4239.981
Sample:                    07-13-1973   HQIC                          4224.472
                         - 12-13-1974
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0885      0.410     -0.216      0.829      -0.893       0.716
ar.L1          1.1942      0.186      6.431      0.000       0.830       1.558
ar.L2         -0.7165      0.152     -4.699      0.000      -1.015      -0.418
ma.L1         -1.1477      0.171     -6.710      0.000      -1.483      -0.812
ma.L2          0.7522      0.133      5.671      0.000       0.492       1.012
sigma2       194.9795      7.341     26.561      0.000     180.592     209.367
===================================================================================
Ljung-Box (L1) (Q):                   0.33   Jarque-Bera (JB):               606.16
Prob(Q):                              0.56   Prob(JB):                         0.00
Heteroskedasticity (H):               2.39   Skew:                            -1.11
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.81
===================================================================================

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(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.78 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=5740.844, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=4223.447, Time=0.12 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=5176.914, Time=0.11 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=8591.962, Time=0.02 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=4219.902, Time=0.10 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=4218.827, Time=0.07 sec
 ARIMA(4,0,0)(0,0,0)[0] intercept   : AIC=4215.972, Time=0.20 sec
 ARIMA(5,0,0)(0,0,0)[0] intercept   : AIC=4215.885, Time=0.24 sec
 ARIMA(5,0,1)(0,0,0)[0] intercept   : AIC=4217.242, Time=0.82 sec
 ARIMA(4,0,1)(0,0,0)[0] intercept   : AIC=4217.384, Time=0.42 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=inf, Time=0.05 sec

Best model:  ARIMA(5,0,0)(0,0,0)[0] intercept
Total fit time: 2.961 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  519
Model:               SARIMAX(5, 0, 0)   Log Likelihood               -2100.943
Date:                Tue, 18 Oct 2022   AIC                           4215.885
Time:                        21:10:10   BIC                           4245.649
Sample:                    07-13-1973   HQIC                          4227.546
                         - 12-13-1974
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     34.8626      8.419      4.141      0.000      18.361      51.364
ar.L1          1.0519      0.047     22.185      0.000       0.959       1.145
ar.L2         -0.0212      0.071     -0.300      0.764      -0.159       0.117
ar.L3          0.0245      0.073      0.336      0.737      -0.118       0.167
ar.L4         -0.0296      0.061     -0.486      0.627      -0.149       0.090
ar.L5         -0.0624      0.038     -1.654      0.098      -0.136       0.012
sigma2       191.3392      7.241     26.426      0.000     177.148     205.530
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):               582.52
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               2.49   Skew:                            -1.00
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.79
===================================================================================

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

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

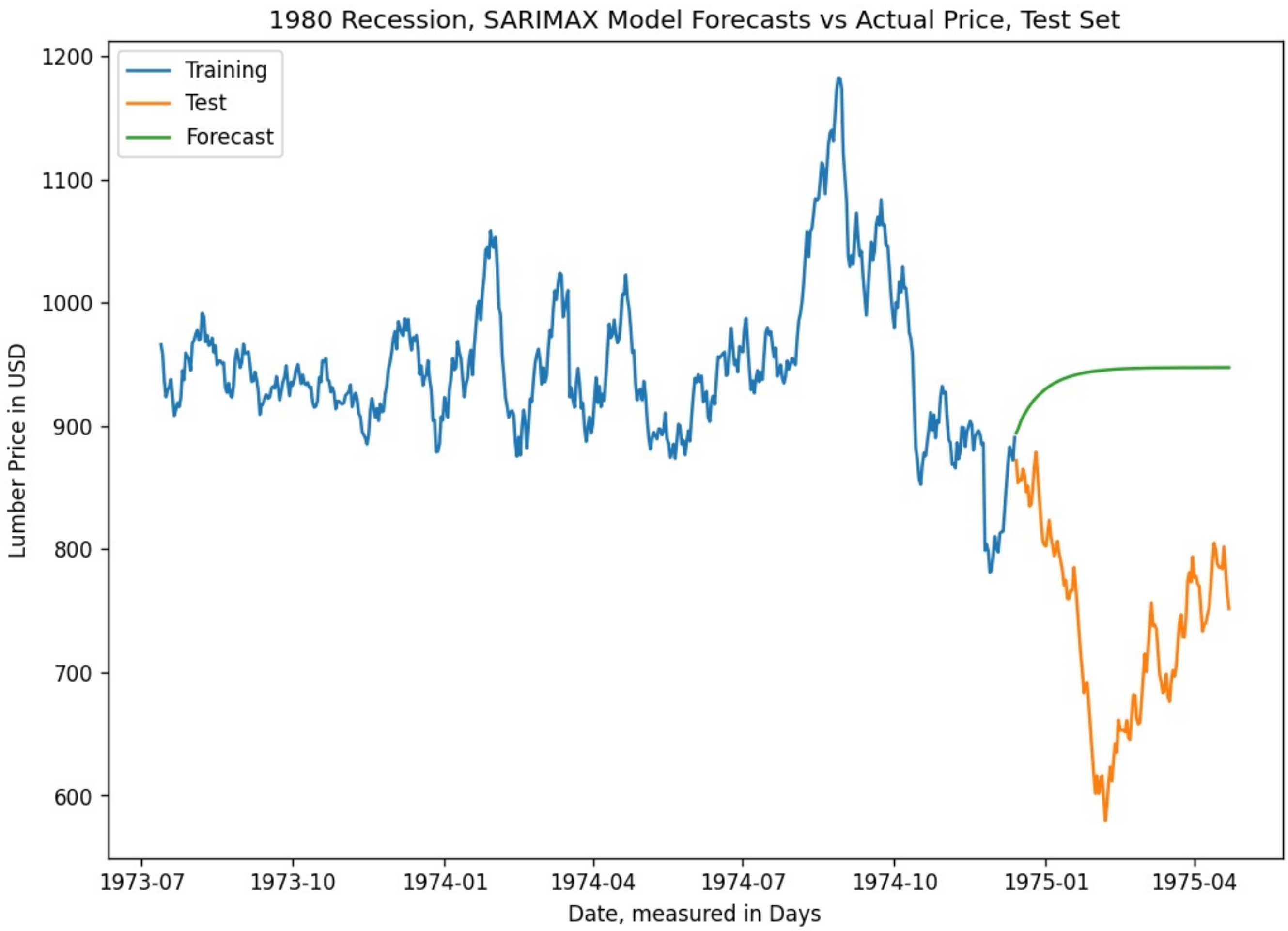
Out[40]:

	0	1
intercept	18.361481	51.363647
ar.L1	0.958992	1.144856
ar.L2	-0.159404	0.117056
ar.L3	-0.118257	0.167266
ar.L4	-0.148988	0.089742
ar.L5	-0.136433	0.011571
sigma2	177.148006	205.530352

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 = 130))

# 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')
```

```python
# Annotate Plot Title
plt.title('1980 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]:
```python
forecast
```

Out[42]:

	forecast_prices
1974-12-14	894.404920
1974-12-15	897.087718
1974-12-16	900.768410
1974-12-17	904.493727
1974-12-18	907.126901
...	...
1975-04-18	947.323270
1975-04-19	947.325035
1975-04-20	947.326699
1975-04-21	947.328267
1975-04-22	947.329745

130 rows × 1 columns

D10 Accuracy Metrics for our forecast

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

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

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]:
```
(130, 1)
```

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

Out[46]:

	forecast_prices
1974-12-14	894.404920
1974-12-15	897.087718
1974-12-16	900.768410
1974-12-17	904.493727
1974-12-18	907.126901
...	...
1975-04-18	947.323270
1975-04-19	947.325035
1975-04-20	947.326699
1975-04-21	947.328267
1975-04-22	947.329745

130 rows × 1 columns

In [47]:
```
# Predictions to numpy array
predicted_array = forecast[['forecast_prices']].to_numpy()
```

In [48]:
```
predicted_array.shape
```

Out[48]:
```
(130, 1)
```

In [49]:
```
#RMSE Calculation

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

```
RMSE = 220.77337542916757
```

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))
```

```
204.95296677080003
```

D11 Visualizing Model Forecast Confidence Intervals at 20% CI# Visualizing Model Forecasts

In [51]:
```
# Model Standard Error calculations, computed numerical Hessian

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

```
intercept    8.419074
ar.L1        0.047415
ar.L2        0.070527
ar.L3        0.072839
ar.L4        0.060901
ar.L5        0.037757
sigma2       7.240527
dtype: float64
```

In [52]:
```
# Generate Model confidence intervals

conf_int = model.conf_int()
```

In [53]:
```
# ------Generate Forecast Prediction Intervals at 90% Confidence
```

```python
y_forec, conf_int = model.predict(130, return_conf_int=True, alpha=0.8)
print(conf_int)
```

```
[[890.90048613 897.9093543 ]
 [892.00140503 902.17403169]
 [894.41718988 907.1196302 ]
 [896.983104   912.00434914]
 [898.55899009 915.69481269]
 [900.13579084 919.06188457]
 [901.72514751 922.19112259]
 [903.21139341 925.01255612]
 [904.59967074 927.55570808]
 [905.94311101 929.89701096]
 [907.23542315 932.05465043]
 [908.47195647 934.04332633]
 [909.65678252 935.88302379]
 [910.7927403  937.590286  ]
 [911.87983665 939.17679043]
 [912.91888375 940.6531547 ]
 [913.91117651 942.02899471]
 [914.85788783 943.31258319]
 [915.76019962 944.51116714]
 [916.61944688 945.63127751]
 [917.43704029 946.67879413]
 [918.21442592 947.65901904]
 [918.95308448 948.57676903]
 [919.65451934 949.43644172]
 [920.32023802 950.24206194]
 [920.95174028 950.99732389]
 [921.55050927 951.70562767]
 [922.11800341 952.37010928]
 [922.65564965 952.99366635]
 [923.16483856 953.57898057]
 [923.64692052 954.12853735]
 [924.10320288 954.64464307]
 [924.53494804 955.12944035]
 [924.9433722  955.58492165]
 [925.32964469 956.01294149]
 [925.69488785 956.4152274 ]
 [926.04017716 956.79338982]
 [926.36654185 957.14893107]
 [926.67496563 957.48325348]
 [926.96638769 957.79766682]
 [927.24170379 958.09339506]
 [927.50176756 958.37158257]
 [927.74739176 958.63329982]
 [927.97934973 958.87954855]
 [928.19837674 959.11126659]
 [928.4051715  959.32933226]
 [928.60039757 959.53456848]
 [928.78468482 959.72774647]
 [928.95863083 959.90958929]
 [929.12280235 960.080775  ]
 [929.27773662 960.24193965]
 [929.42394274 960.39368008]
 [929.56190299 960.53655644]
 [929.69207409 960.67109457]
 [929.8148884  960.79778823]
 [929.93075514 960.91710117]
 [930.04006153 961.02946898]
 [930.14317384 961.13530095]
 [930.24043851 961.23498169]
 [930.33218311 961.32887267]
 [930.41871731 961.41731369]
 [930.50033384 961.50062423]
 [930.57730933 961.57910467]
 [930.64990516 961.65303752]
 [930.7183683  961.72268847]
 [930.78293203 961.78830747]
 [930.84381668 961.85012964]
 [930.90123035 961.90837621]
 [930.95536951 961.96325535]
 [931.0064197  962.01496295]
 [931.05455607 962.06368339]
 [931.09994397 962.10959018]
 [931.14273945 962.15284667]
 [931.18308984 962.19360663]
 [931.22113413 962.23201481]
 [931.25700354 962.2682075 ]
 [931.29082186 962.30231303]
 [931.32270589 962.33445224]
 [931.35276585 962.36473891]
 [931.38110572 962.39328019]
 [931.40782359 962.420177  ]
 [931.43301199 962.44552439]
 [931.45675822 962.46941187]
 [931.47914462 962.49192375]
 [931.50024885 962.51313947]
 [931.52014417 962.53313384]
```

```
 [931.53889965 962.55197732]
 [931.55658045 962.56973631]
 [931.57324803 962.58647335]
 [931.58896033 962.60224737]
 [931.60377201 962.61711387]
 [931.61773459 962.63112517]
 [931.63089667 962.64433053]
 [931.64330408 962.65677638]
 [931.65500001 962.66850648]
 [931.66602522 962.67956204]
 [931.67641811 962.68998189]
 [931.68621491 962.69980265]
 [931.69544976 962.70905878]
 [931.70415487 962.7177828 ]
 [931.71236058 962.72600531]
 [931.72009553 962.73375519]
 [931.72738669 962.74105961]
 [931.73425951 962.74794421]
 [931.74073797 962.75443314]
 [931.74684468 962.76054914]
 [931.75260095 962.76631368]
 [931.75802689 962.77174696]
 [931.76314143 962.77686802]
 [931.76796244 962.78169483]
 [931.77250676 962.78624429]
 [931.77679025 962.79053236]
 [931.78082788 962.79457405]
 [931.78463376 962.79838354]
 [931.78822117 962.80197416]
 [931.79160266 962.8053585 ]
 [931.79479005 962.80854842]
 [931.79779446 962.81155508]
 [931.8006264  962.81438901]
 [931.80329576 962.81706016]
 [931.80581189 962.81957786]
 [931.80818356 962.82195093]
 [931.81041908 962.82418769]
 [931.81252625 962.82629597]
 [931.81451245 962.82828315]
 [931.81638461 962.83015619]
 [931.81814928 962.83192164]
 [931.81981264 962.83358568]
 [931.8213805  962.83515415]
 [931.82285833 962.83663253]]
```

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 [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/1980Low_Prediction.csv')
high_pred = pd.read_csv('C:/Users/ericy/Desktop/1980High_Prediction.csv')
```

In [57]:

```
# Variable exploration to ensure compatability with 'test' datetime timeframe
low_pred
```

Out[57]:

	Date	Low_Prediction
0	1974-12-14	890.900486
1	1974-12-15	892.001405
2	1974-12-16	894.417190
3	1974-12-17	896.983104
4	1974-12-18	898.558990
...	...	...
125	1975-04-18	931.816385
126	1975-04-19	931.818149
127	1975-04-20	931.819813
128	1975-04-21	931.821381
129	1975-04-22	931.822858

130 rows × 2 columns

In [58]:

```
# Variable exploration to ensure compatability with 'test' datetime timeframe
high_pred
```

Out[58]:

	Date	High_Prediction
0	1974-12-14	897.909354
1	1974-12-15	902.174032
2	1974-12-16	907.119630
3	1974-12-17	912.004349
4	1974-12-18	915.694813
...	...	...
125	1975-04-18	962.830156
126	1975-04-19	962.831922
127	1975-04-20	962.833586
128	1975-04-21	962.835154
129	1975-04-22	962.836632

130 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
1974-12-14	890.900486
1974-12-15	892.001405
1974-12-16	894.417190
1974-12-17	896.983104
1974-12-18	898.558990
...	...
1975-04-18	931.816385
1975-04-19	931.818149
1975-04-20	931.819813
1975-04-21	931.821381
1975-04-22	931.822858

130 rows × 1 columns

In [64]:
```
high_pred
```

Out[64]:

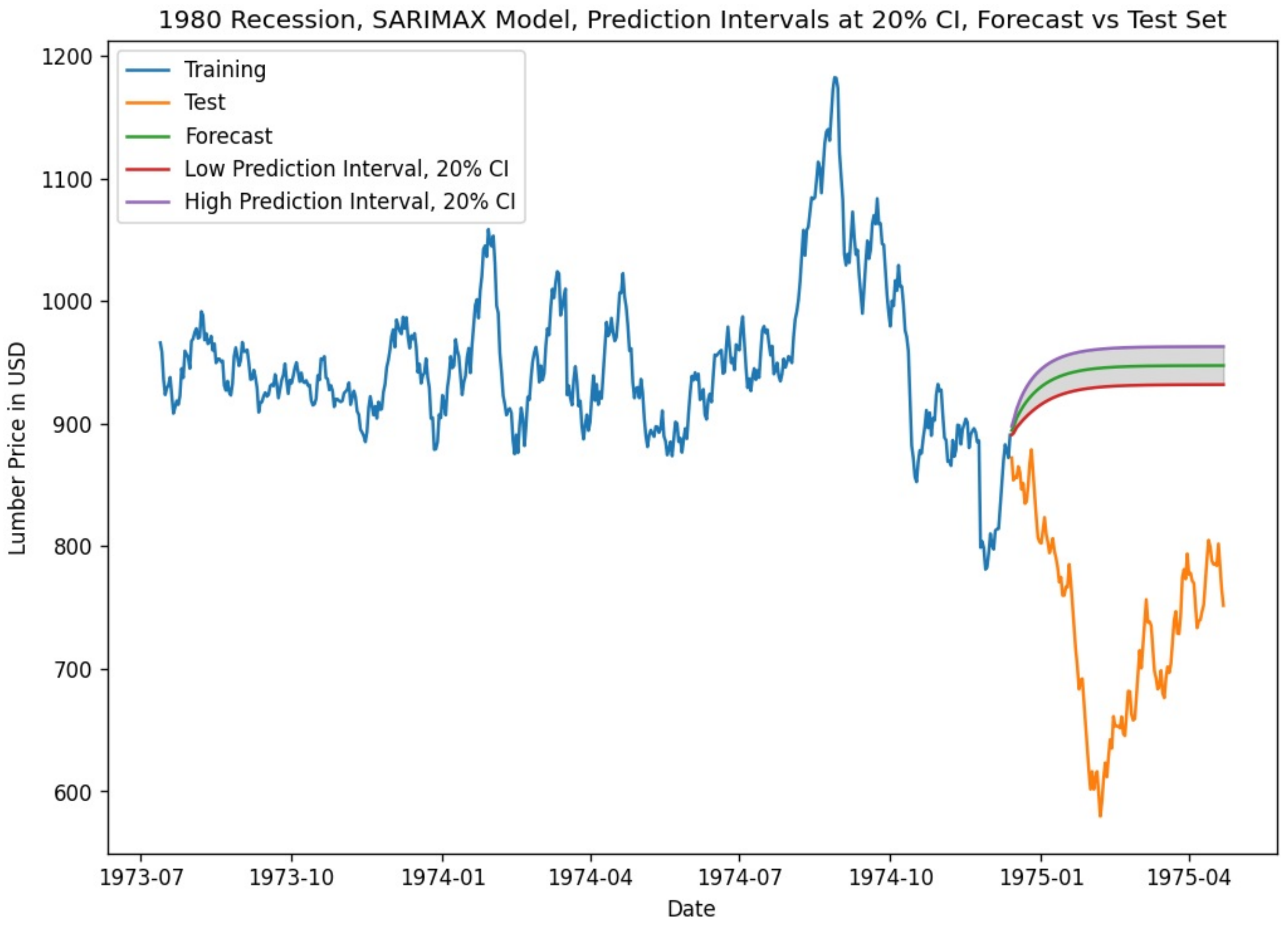
Date	High_Prediction
1974-12-14	897.909354
1974-12-15	902.174032
1974-12-16	907.119630
1974-12-17	912.004349
1974-12-18	915.694813
...	...
1975-04-18	962.830156
1975-04-19	962.831922
1975-04-20	962.833586
1975-04-21	962.835154
1975-04-22	962.836632

130 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 = 130),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('1980 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
# 1980 Recession we Accept the Null Hypothesis
```

In [ ]:

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