In [ ]:

D214 Capstone

Modeling Inflation Adusted Recessionary Lumber Prices

Great 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[8332:9221]
```

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

Out[6]:

	Date	Trading Days	2022_Value	Value
8332	2005-12-07	8333	461.6040	322.80
8333	2005-12-08	8334	463.0340	323.80
8334	2005-12-09	8335	459.0300	321.00
8335	2005-12-12	8336	467.6100	327.00
8336	2005-12-13	8337	463.0340	323.80
...	...	...	...	...
9216	2009-06-25	9217	255.3516	202.66
9217	2009-06-26	9218	268.0776	212.76
9218	2009-06-29	9219	263.3400	209.00
9219	2009-06-30	9220	261.6264	207.64
9220	2009-07-01	9221	264.6000	210.00

889 rows × 4 columns

D1: Exploratory Data Analysis

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

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

Out[8]:

	Trading Days	2022_Value
8332	8333	461.6040
8333	8334	463.0340
8334	8335	459.0300
8335	8336	467.6100
8336	8337	463.0340
...	...	...
9216	9217	255.3516
9217	9218	268.0776
9218	9219	263.3400
9219	9220	261.6264
9220	9221	264.6000

889 rows × 2 columns

EDA

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

Out[9]:

	Trading Days	2022_Value
8332	8333	461.604
8333	8334	463.034
8334	8335	459.030
8335	8336	467.610
8336	8337	463.034

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 889 entries, 8332 to 9220
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Trading Days  889 non-null    int64  
 1   2022_Value    889 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 14.0 KB
```

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

Out[11]:
```
(889, 2)
```

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

Out[12]:

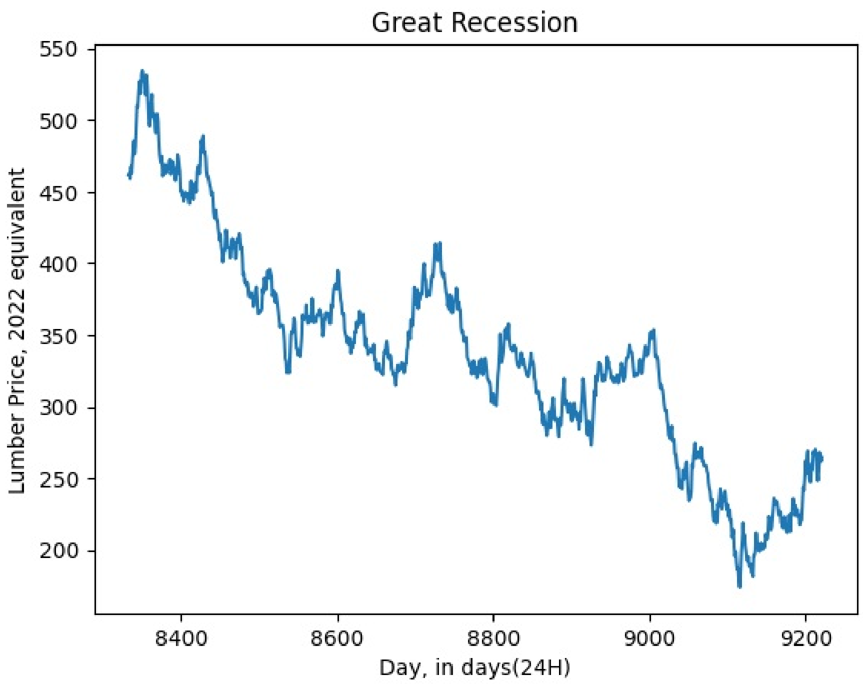
	Trading Days	2022_Value
count	889.000000	889.000000
mean	8777.000000	336.714558
std	256.776492	76.399901
min	8333.000000	174.006000
25%	8555.000000	290.165000
50%	8777.000000	333.788000
75%	8999.000000	377.746000
max	9221.000000	534.750000

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('Great 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: 889 entries, 8332 to 9220
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Trading Days  889 non-null    datetime64[ns]
 1   2022_Value    889 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 20.8 KB
```

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

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 889 entries, 1992-10-25 to 1995-04-01
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   2022_Value  889 non-null    float64
dtypes: float64(1)
memory usage: 13.9 KB
```

Date discrepency noted, per proposal delimitations

Dataset is for Great Recession Period, December 2005 - July 2009

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

Out[20]:

Trading Days	2022_Value
1992-10-25	461.6040
1992-10-26	463.0340
1992-10-27	459.0300
1992-10-28	467.6100
1992-10-29	463.0340
...	...
1995-03-28	255.3516
1995-03-29	268.0776
1995-03-30	263.3400
1995-03-31	261.6264
1995-04-01	264.6000

889 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]:
```
(-1.5523723714020112,
 0.5074629515436003,
 0,
 888,
 {'1%': -3.4377354773501243,
  '5%': -2.8648002689134535,
  '10%': -2.5685059946940183},
 5551.09052170876)
```

In [23]:
```
# Results show p = .50746
# 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('Great 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}")
```

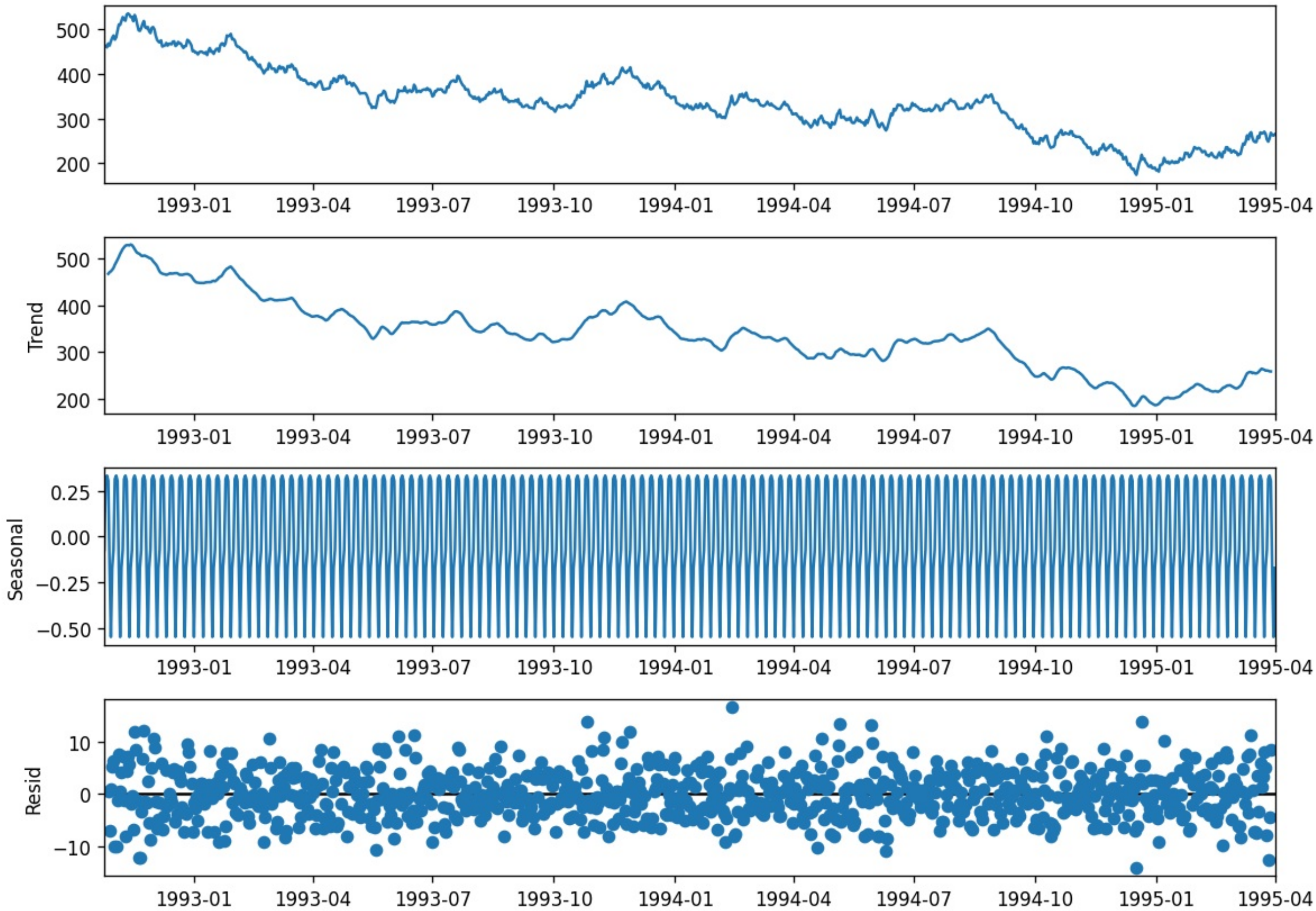
```
Estimated differencing term: 1
```

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();
```



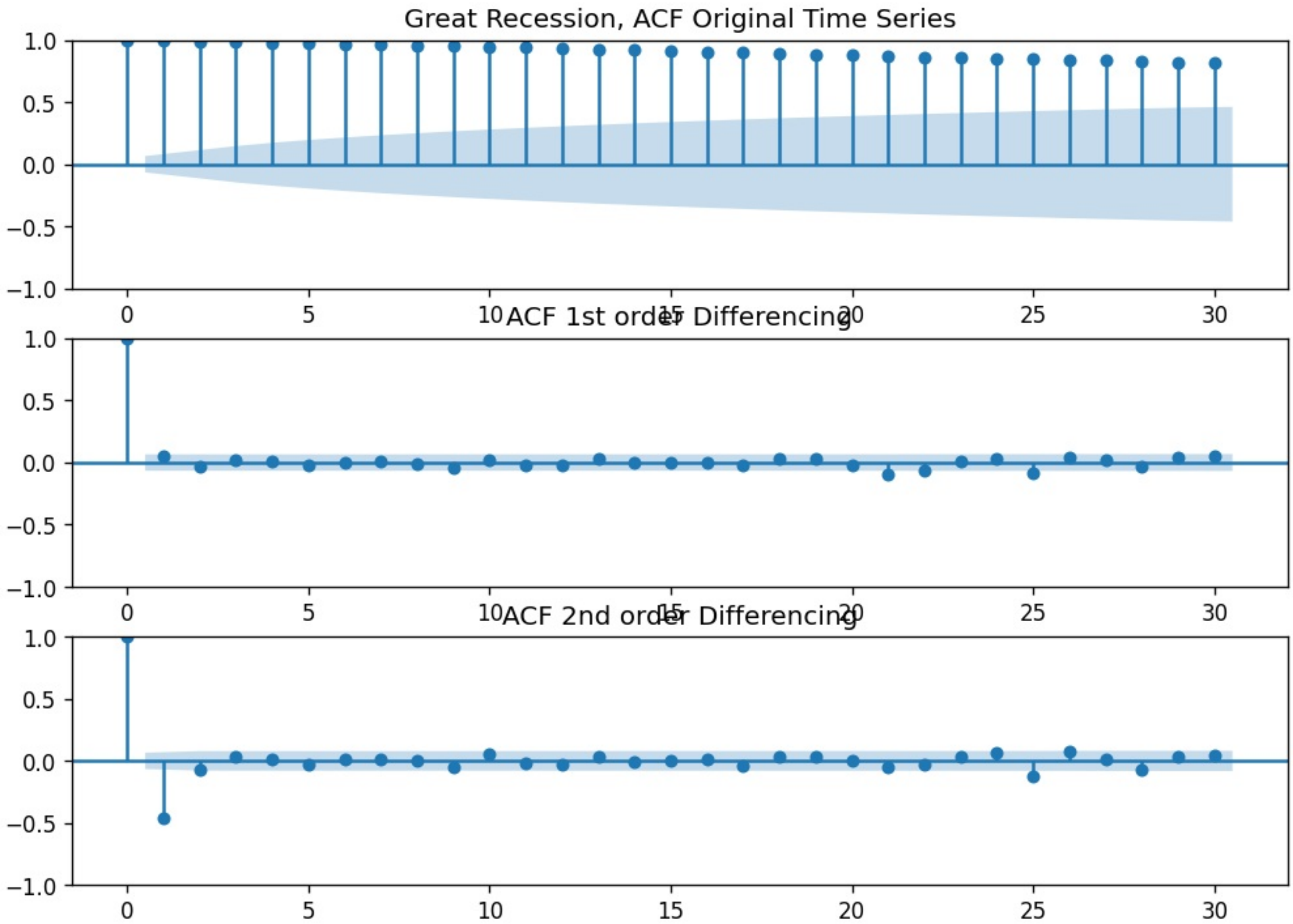
D6 ACF and PACF

Finding order of MA term 'q'

Using Autocorrelation function (ACF)

In [28]:

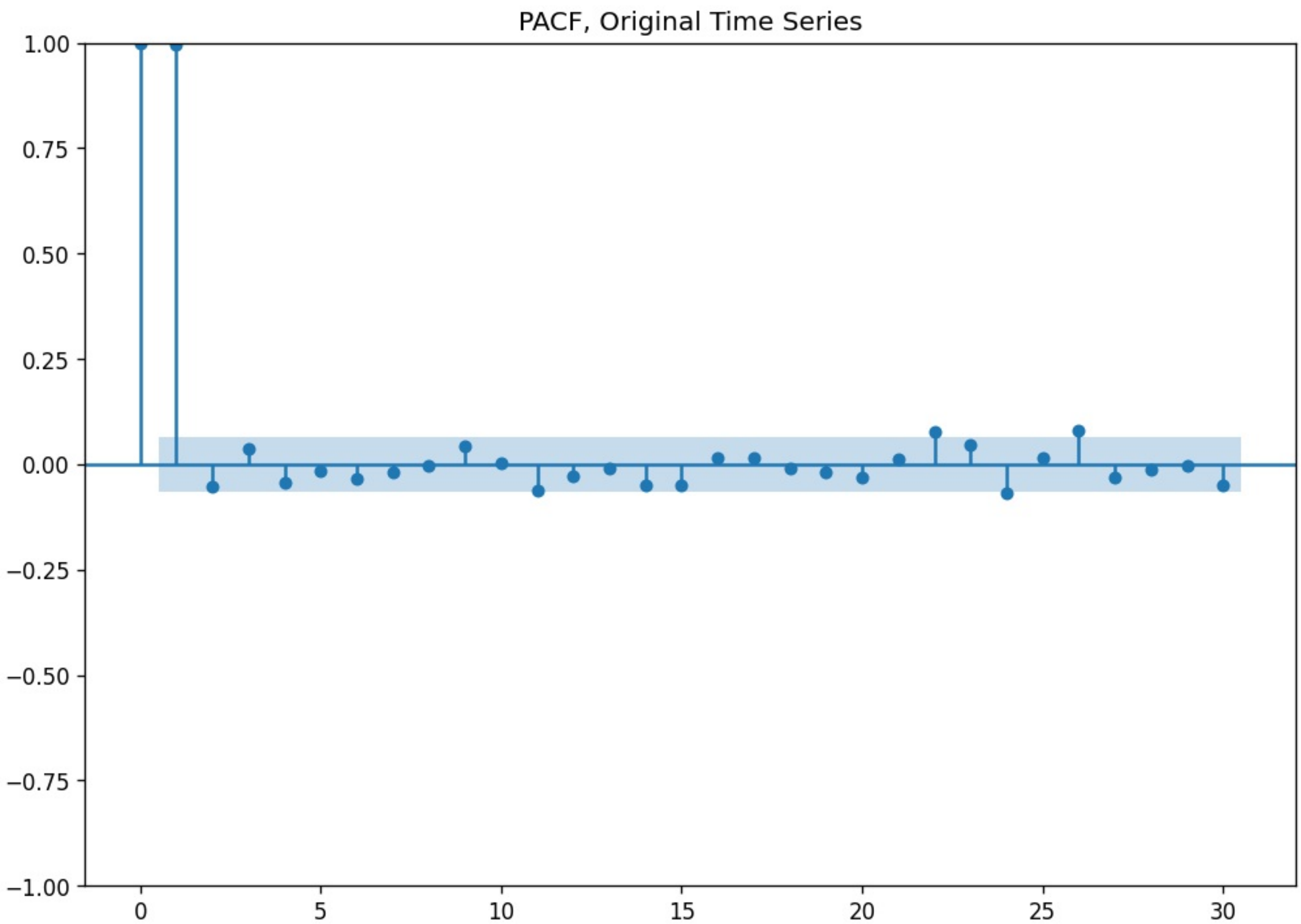
```
fig, (ax1, ax2, ax3) = plt.subplots(3)
plot_acf(df, ax=ax1, title='Great Recession, ACF Original Time Series');
plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```



Finding order of AR term 'p'

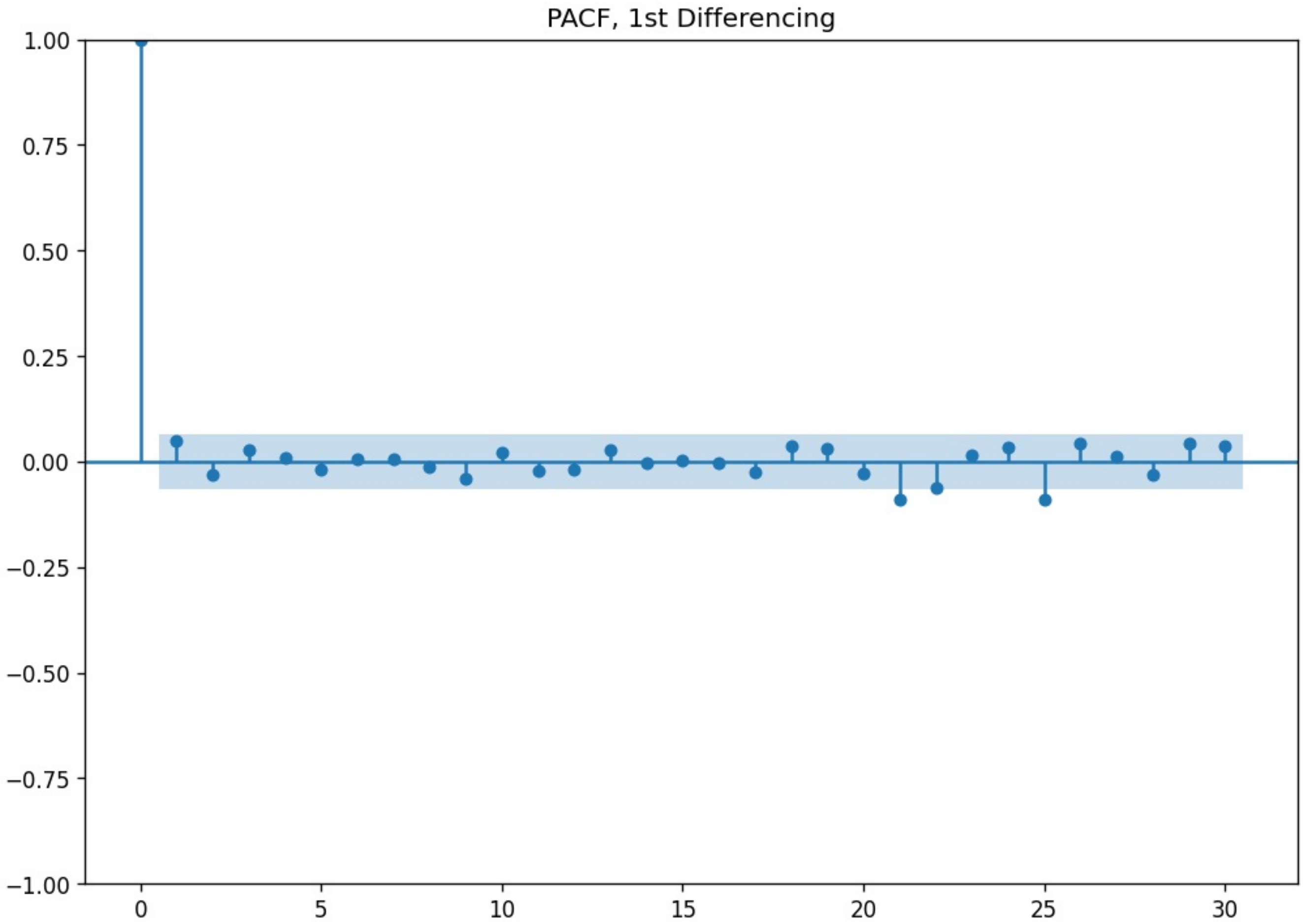
Using Partial autocorrelation (PACF)

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



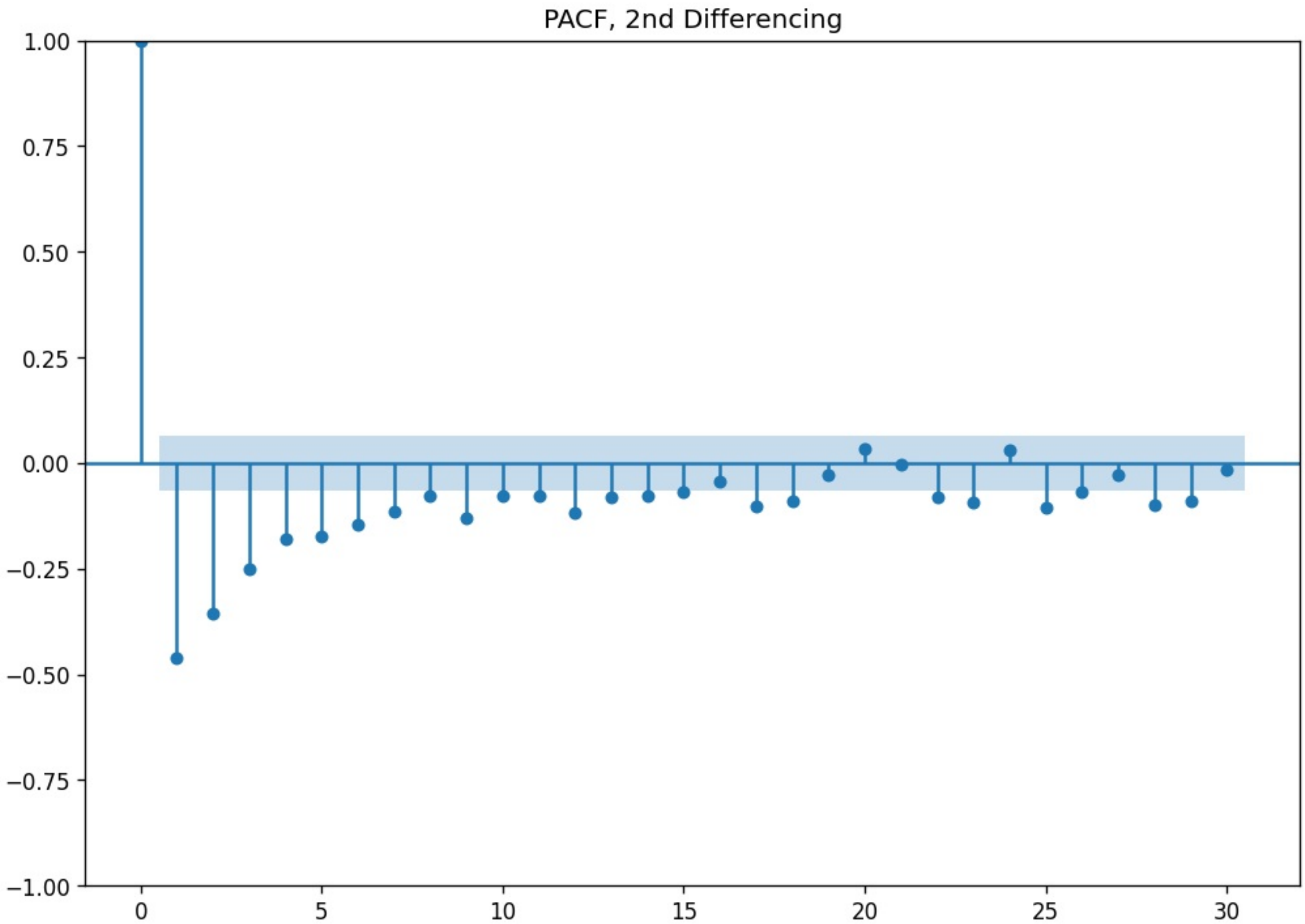
In [30]:

```
plot_pacf(df.diff().dropna(), title='PACF, 1st Differencing');
```



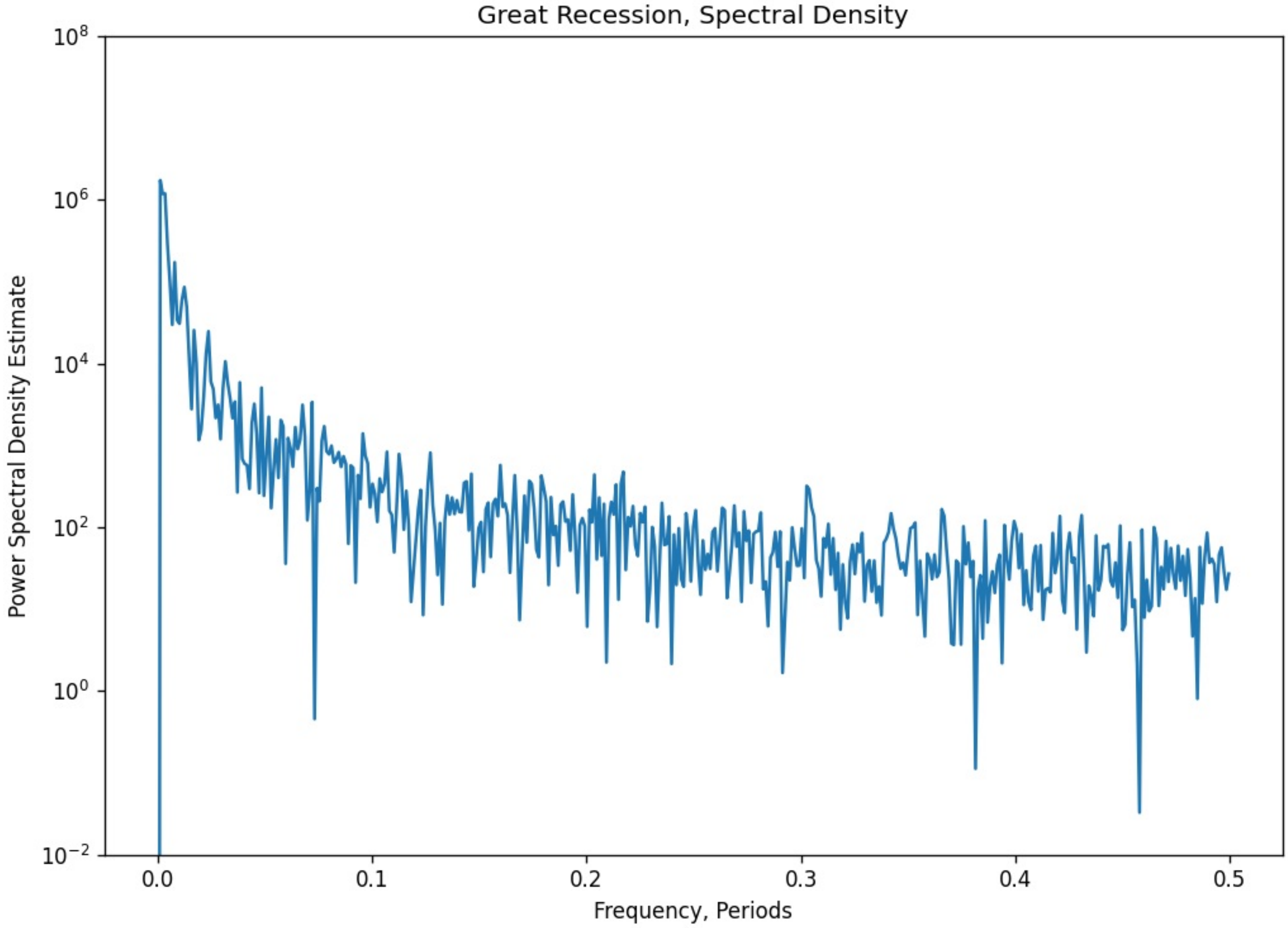
In [31]:

```
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('Great 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 = 889 cases

80/20 Train/Test Split

Split is 711 / 178

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=711)
```

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

Out[34]:

	2022_Value
Trading Days	
1992-10-25	461.604
1992-10-26	463.034
1992-10-27	459.030
1992-10-28	467.610
1992-10-29	463.034
...	...
1994-10-01	248.507
1994-10-02	243.529
1994-10-03	246.542
1994-10-04	242.743
1994-10-05	255.843

711 rows × 1 columns

In [35]:

```
test
```

Out[35]:

	2022_Value
Trading Days	
1994-10-06	255.4500
1994-10-07	249.5550
1994-10-08	255.8430
1994-10-09	260.6900
1994-10-10	261.3450
...	...
1995-03-28	255.3516
1995-03-29	268.0776
1995-03-30	263.3400
1995-03-31	261.6264
1995-04-01	264.6000

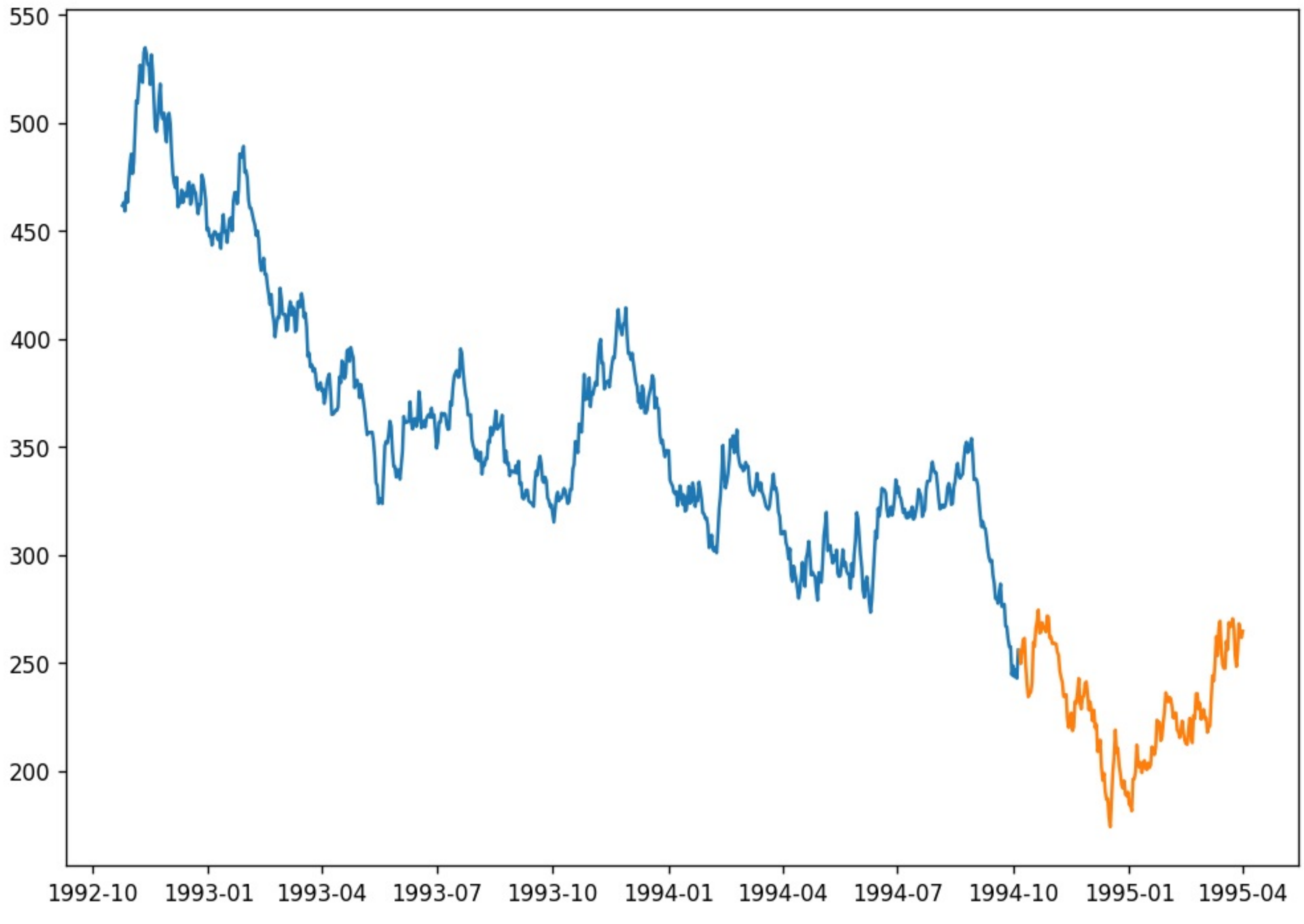
178 rows × 1 columns

In [36]:

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

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

Out[36]:

```
[<matplotlib.lines.Line2D at 0x1b1827fa388>]
```



In [37]:

```
print(train.shape)
print(test.shape)
```

```
(711, 1)
(178, 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)
```

```python
# Establish auto_arima to run ARIMA and take into account
# 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=4543.291, Time=0.26 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=4541.772, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=4541.591, Time=0.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=4541.481, Time=0.08 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4541.772, Time=0.02 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=4543.176, Time=0.16 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=4545.170, Time=0.23 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=4541.481, Time=0.07 sec

Best model:  ARIMA(0,1,1)(0,0,0)[0]
Total fit time: 0.891 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  711
Model:               SARIMAX(0, 1, 1)   Log Likelihood               -2267.741
Date:                Tue, 18 Oct 2022   AIC                           4541.481
Time:                        13:09:45   BIC                           4555.177
Sample:                    10-25-1992   HQIC                          4546.772
                         - 10-05-1994
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.2885      0.238     -1.213      0.225      -0.755       0.178
ma.L1          0.0583      0.038      1.551      0.121      -0.015       0.132
sigma2        34.8176      1.827     19.055      0.000      31.236      38.399
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 2.56
Prob(Q):                              0.97   Prob(JB):                         0.28
Heteroskedasticity (H):               0.76   Skew:                             0.14
Prob(H) (two-sided):                  0.03   Kurtosis:                         3.07
===================================================================================

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=inf, Time=0.89 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=4541.772, Time=0.02 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=4541.591, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=4541.481, Time=0.08 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4541.477, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=4543.291, Time=0.20 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 1.318 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  711
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -2269.738
Date:                Tue, 18 Oct 2022   AIC                           4541.477
Time:                        21:11:01   BIC                           4546.042
Sample:                    10-25-1992   HQIC                          4543.240
                         - 10-05-1994
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2        35.0141      1.835     19.085      0.000      31.418      38.610
===================================================================================
Ljung-Box (L1) (Q):                   2.17   Jarque-Bera (JB):                 3.63
Prob(Q):                              0.14   Prob(JB):                         0.16
Heteroskedasticity (H):               0.76   Skew:                             0.17
Prob(H) (two-sided):                  0.03   Kurtosis:                         3.09
===================================================================================

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

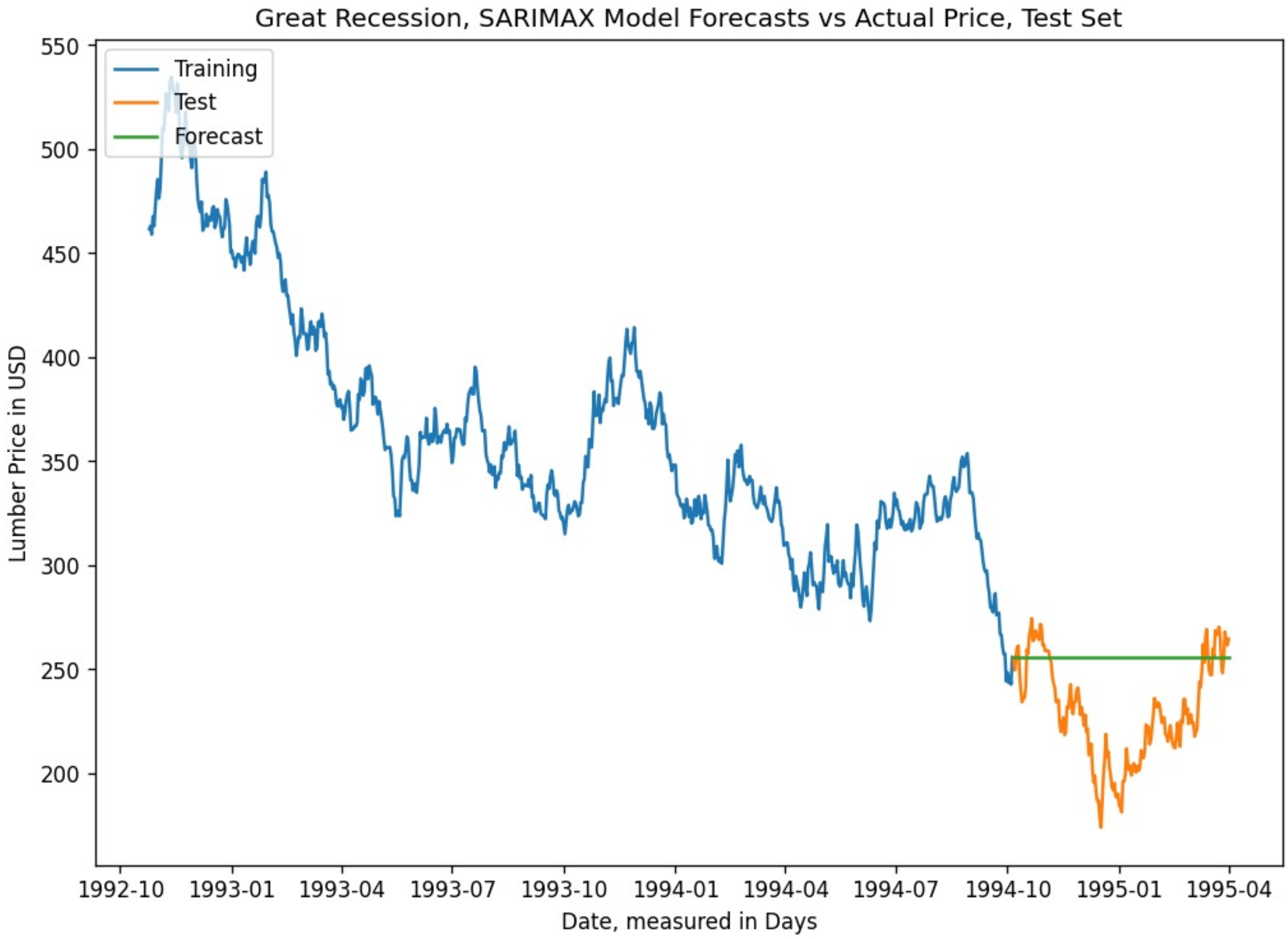
In [40]:
```
model.conf_int()
```

Out[40]:

	0	1
sigma2	31.418285	38.609899

Plotting 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 = 178))

# 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('Great 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
1994-10-06	255.843
1994-10-07	255.843
1994-10-08	255.843
1994-10-09	255.843
1994-10-10	255.843
...	...
1995-03-28	255.843
1995-03-29	255.843
1995-03-30	255.843
1995-03-31	255.843
1995-04-01	255.843

178 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]:
```
(178, 1)
```

In [ ]:
```

```

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

In [48]:
```
predicted_array.shape
```
Out[48]:
```
(178, 1)
```

In [49]:
```
#RMSE Calculation

rmse = sqrt(mean_squared_error(test_array, predicted_array))
print ('RMSE = ' + str(rmse))
```
```
RMSE = 36.00079057140054
```

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))
```
```
29.834739325842687
```

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)
```
```
sigma2    1.834629
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(178, return_conf_int=True, alpha=0.8)
print(conf_int)
```
```
[[254.34387662 257.34212338]
 [253.72291939 257.96308061]
 [253.24644214 258.43955786]
 [252.84475324 258.84124676]
 [252.49085822 259.19514178]
 [252.17091266 259.51508734]
 [251.87669236 259.80930764]
 [251.60283877 260.08316123]
 [251.34562986 260.34037014]
 [251.10235563 260.58364437]
 [250.87097024 260.81502976]
 [250.64988428 261.03611572]
 [250.43783379 261.24816621]
 [250.23379394 261.45220606]
 [250.03692012 261.64907988]
 [249.84650649 261.83949351]
 [249.66195596 262.02404404]
 [249.48275816 262.20324184]
 [249.30847269 262.37752731]
 [249.13871644 262.54728356]
 [248.97315364 262.71284636]
 [248.81148808 262.87451192]
 [248.65345684 263.03254316]
 [248.49882532 263.18717468]
 [248.34738311 263.33861689]
 [248.19894064 263.48705936]
 [248.05332642 263.63267358]
 [247.91038471 263.77561529]
 [247.76997354 263.91602646]
 [247.63196309 264.05403691]
 [247.49623428 264.18976572]
 [247.36267755 264.32332245]
 [247.23119184 264.45480816]
 [247.10168369 264.58431631]
 [246.97406649 264.71193351]
 [246.84825973 264.83774027]
 [246.72418848 264.96181152]
 [246.60178285 265.08421715]
 [246.4809775  265.2050225 ]
 [246.36171126 265.32428874]
 [246.24392676 265.44207324]
 [246.12757011 265.55842989]
 [246.0125906  265.6734094 ]
 [245.89894048 265.78705952]
```

```
[245.78657466 265.89942534]
[245.67545056 266.01054944]
[245.56552791 266.12047209]
[245.45676857 266.22923143]
[245.34913635 266.33686365]
[245.24259693 266.44340307]
[245.13711769 266.54888231]
[245.03266758 266.65333242]
[244.92921707 266.75678293]
[244.82673798 266.85926202]
[244.72520347 266.96079653]
[244.62458787 267.06141213]
[244.52486669 267.16113331]
[244.42601649 267.25998351]
[244.32801483 267.35798517]
[244.23084024 267.45515976]
[244.13447212 267.55152788]
[244.03889071 267.64710929]
[243.94407707 267.74192293]
[243.85001297 267.83598703]
[243.75668092 267.92931908]
[243.6640641  268.0219359 ]
[243.5721463  268.1138537 ]
[243.48091193 268.20508807]
[243.39034597 268.29565403]
[243.30043394 268.38556606]
[243.21116188 268.47483812]
[243.12251632 268.56348368]
[243.03448424 268.65151576]
[242.94705308 268.73894692]
[242.86021071 268.82578929]
[242.77394538 268.91205462]
[242.68824574 268.99775426]
[242.60310081 269.08289919]
[242.51849996 269.16750004]
[242.43443287 269.25156713]
[242.35088959 269.33511041]
[242.26786044 269.41813956]
[242.18533603 269.50066397]
[242.10330728 269.58269272]
[242.02176536 269.66423464]
[241.94070171 269.74529829]
[241.860108   269.825892  ]
[241.77997616 269.90602384]
[241.70029833 269.98570167]
[241.62106689 270.06493311]
[241.54227441 270.14372559]
[241.46391369 270.22208631]
[241.38597769 270.30002231]
[241.30845959 270.37754041]
[241.23135273 270.45464727]
[241.15465064 270.53134936]
[241.07834702 270.60765298]
[241.0024357  270.6835643 ]
[240.92691072 270.75908928]
[240.85176621 270.83423379]
[240.7769965  270.9090035 ]
[240.70259603 270.98340397]
[240.62855939 271.05744061]
[240.55488128 271.13111872]
[240.48155655 271.20444345]
[240.40858016 271.27741984]
[240.33594719 271.35005281]
[240.26365285 271.42234715]
[240.19169243 271.49430757]
[240.12006136 271.56593864]
[240.04875515 271.63724485]
[239.97776942 271.70823058]
[239.90709989 271.77890011]
[239.83674238 271.84925762]
[239.76669277 271.91930723]
[239.69694708 271.98905292]
[239.62750137 272.05849863]
[239.55835181 272.12764819]
[239.48949464 272.19650536]
[239.42092618 272.26507382]
[239.35264283 272.33335717]
[239.28464107 272.40135893]
[239.21691744 272.46908256]
[239.14946855 272.53653145]
[239.08229109 272.60370891]
[239.01538181 272.67061819]
[238.94873752 272.73726248]
[238.88235509 272.80364491]
[238.81623147 272.86976853]
[238.75036364 272.93563636]
[238.68474868 273.00125132]
[238.61938367 273.06661633]
[238.5542658  273.1317342 ]
```

```
 [238.48939227 273.19660773]
 [238.42476036 273.26123964]
 [238.36036739 273.32563261]
 [238.29621072 273.38978928]
 [238.23228778 273.45371222]
 [238.16859603 273.51740397]
 [238.10513297 273.58086703]
 [238.04189617 273.64410383]
 [237.97888322 273.70711678]
 [237.91609175 273.76990825]
 [237.85351946 273.83248054]
 [237.79116405 273.89483595]
 [237.7290233  273.9569767 ]
 [237.66709499 274.01890501]
 [237.60537697 274.08062303]
 [237.5438671  274.1421329 ]
 [237.4825633  274.2034367 ]
 [237.42146351 274.26453649]
 [237.36056571 274.32543429]
 [237.29986789 274.38613211]
 [237.23936812 274.44663188]
 [237.17906445 274.50693555]
 [237.118955   274.567045  ]
 [237.0590379  274.6269621 ]
 [236.99931132 274.68668868]
 [236.93977346 274.74622654]
 [236.88042252 274.80557748]
 [236.82125677 274.86474323]
 [236.76227448 274.92372552]
 [236.70347395 274.98252605]
 [236.64485352 275.04114648]
 [236.58641154 275.09958846]
 [236.52814639 275.15785361]
 [236.47005647 275.21594353]
 [236.41214022 275.27385978]
 [236.35439608 275.33160392]
 [236.29682253 275.38917747]
 [236.23941807 275.44658193]
 [236.18218121 275.50381879]
 [236.1251105  275.5608895 ]
 [236.06820449 275.61779551]
 [236.01146178 275.67453822]
 [235.95488095 275.73111905]
 [235.89846065 275.78753935]
 [235.84219949 275.84380051]]
```

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/Gr_Rec_Low_Prediction.csv')
high_pred = pd.read_csv('C:/Users/ericy/Desktop/Gr_Rec_High_Prediction.csv')
```

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

Out[57]:

	Date	Low_Prediction
0	1994-10-06	254.343877
1	1994-10-07	253.722919
2	1994-10-08	253.246442
3	1994-10-09	252.844753
4	1994-10-10	252.490858
...	...	...
173	1995-03-28	236.068205
174	1995-03-29	236.011462
175	1995-03-30	235.954881
176	1995-03-31	235.898461
177	1995-04-01	235.842199

178 rows × 2 columns

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

Out[58]:

	Date	High_Prediction
0	1994-10-06	257.342123
1	1994-10-07	257.963081
2	1994-10-08	258.439558
3	1994-10-09	258.841247
4	1994-10-10	259.195142
...	...	...
173	1995-03-28	275.617795
174	1995-03-29	275.674538
175	1995-03-30	275.731119
176	1995-03-31	275.787539
177	1995-04-01	275.843800

178 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
1994-10-06	254.343877
1994-10-07	253.722919
1994-10-08	253.246442
1994-10-09	252.844753
1994-10-10	252.490858
...	...
1995-03-28	236.068205
1995-03-29	236.011462
1995-03-30	235.954881
1995-03-31	235.898461
1995-04-01	235.842199

178 rows × 1 columns

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

Out[64]:

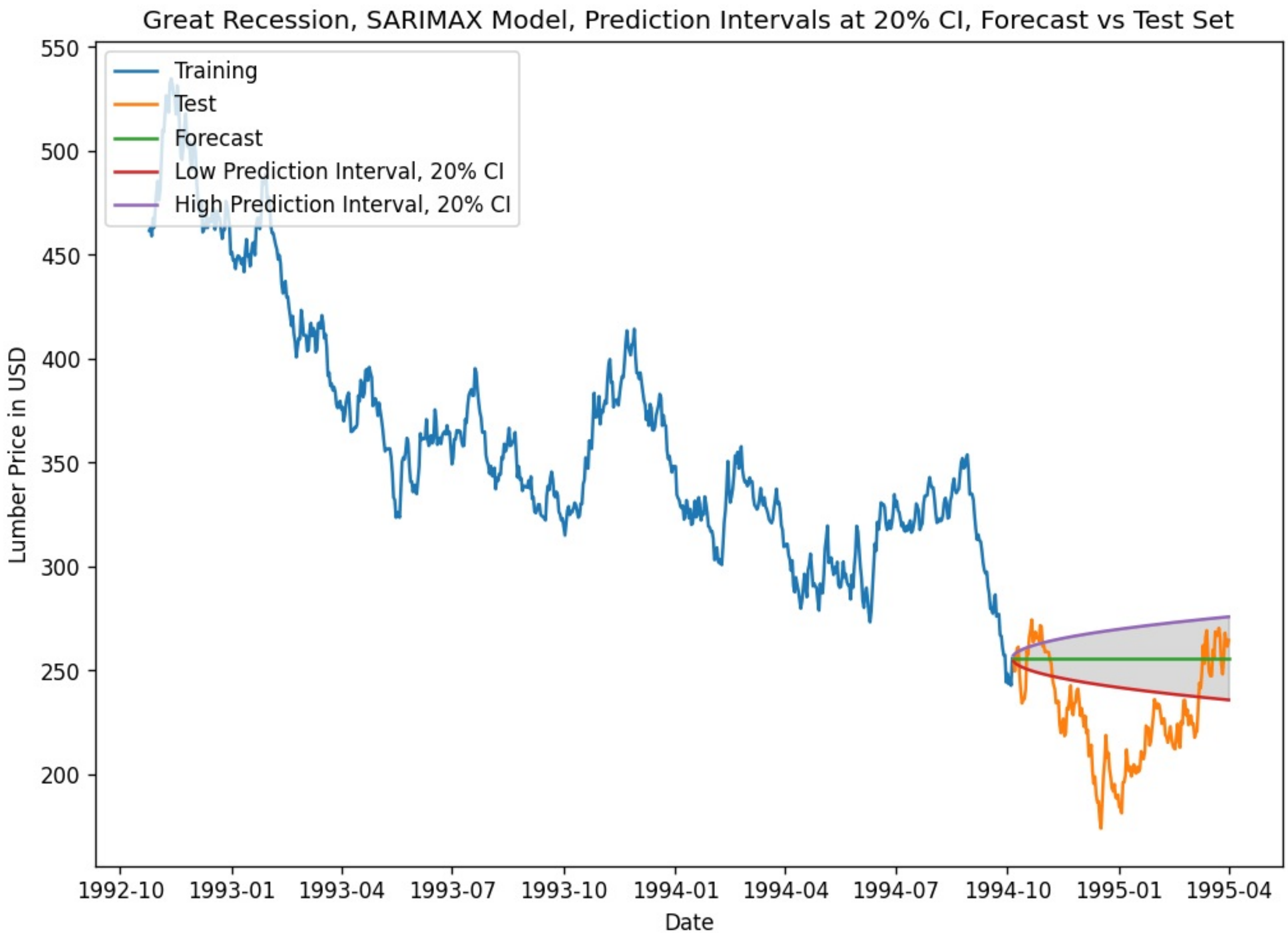
Date	High_Prediction
1994-10-06	257.342123
1994-10-07	257.963081
1994-10-08	258.439558
1994-10-09	258.841247
1994-10-10	259.195142
...	...
1995-03-28	275.617795
1995-03-29	275.674538
1995-03-30	275.731119
1995-03-31	275.787539
1995-04-01	275.843800

178 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 = 178),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('Great 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
# Great Recession we Accept the Null Hypothesis
```

In [ ]:

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