Eric Yarger

Modeling Recessionary USA Lumber Prices: A Time Series Analyis

Executive Summary

and

Data Analytics Report

Outline

Executive Summary

*****Note***** Seven Supplemental PDFs are included as separate attachments with this analysis. These are copies of the Jupyter Notebook coding environment used for each Recessionary period.

Executive Summary

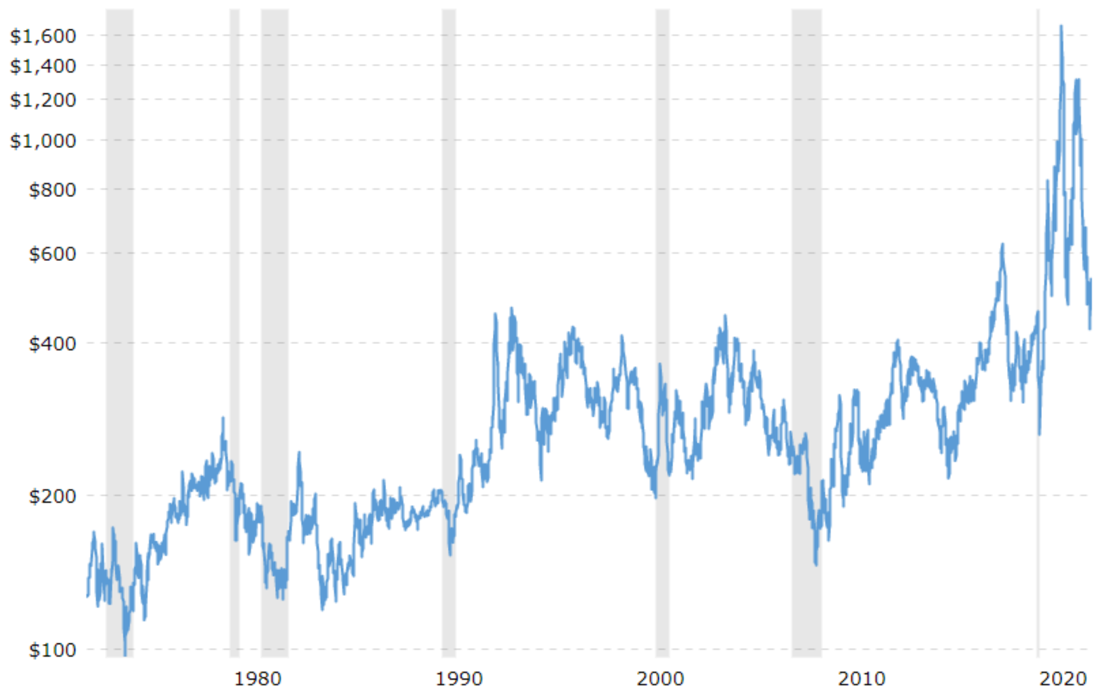
Project Summary

This project uses data on historic US Lumber prices to create predictive pricing models for US Recessionary periods. The dataset used for this analysis includes Lumber prices from 1972 through 2022. During this time period there were seven economic recessions:

Recession Name	Length of Recession In Months	Recession Beginning	Recession Ending
1973-1975 Recession	16	November 1973	March 1975
1980 Recession	6	January 1980	July 1980
1981-1982 Recession	16	July 1981	November 1982
Early 1990's Recession	8	July 1990	March 1991
Early 2000's Recession	8	March 2001	November 2001
Great Recession	18	December 2007	June 2009
COVID-19 Recession	2	February 2020	April 2020

Source: (List of recessions in the United States, n.d.)

The recession periods are visualized as the shaded gray time periods in the graph showing lumber prices from 1972 to 2022 below.



Time Series forecasting will be used for this analysis, specifically the creation of ARIMA models for each recession period. This results in seven separate model development environments for this analysis. The generated forecast lumber prices are then compared to actual prices for the same period. The goal of each model is to forecast recession pricing within a 20% confidence interval. All of the model environments, with all calculations and outputs, are included as attached PDFs.

Project Results

Project results were outlined through defining a null and alternative hypothesis.   The null hypothesis for this analysis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval cannot be made from the historical US Lumber pricing dataset. The alternative hypothesis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval can be made from the historical US Lumber pricing dataset.

Through this lens of interpreting the project's results, every model failed to reject the Null Hypothesis. This results in the author recommending against solely relying on ARIMA models for forecasting Lumber prices during Recession periods.

Data Analytics Report

A: Research Question

Capstone Project Name

Time Series Analysis of Recessionary US Lumber Prices Data Set.

Project Topic

ARIMA Model for Prediction of US Lumber Prices.

Research Question

The research question for this analysis is: Can an ARIMA model be constructed that accurately forecasts Recessionary US Lumber Prices?

Context

The US market size for lumber wholesaling in the United States in 2022 is estimated to be $200.8 billion dollars. The annualized average market size growth from 2017 to 2022 is 14.2% (*Industry market research, reports, and statistics*, n.d.). Lumber is traded on the commodity market in the same manner as oil, metals, agriculture products, and energy. Market lumber prices are measured in USD per 1000 board feet of lumber. For example, the price of lumber on October 14th, 2022 closed at 494.00. This represents the price of 1000 board feet of lumber on this specific day.

Lumber pricing is volatile over time, and regularly sees large swings both up and down in pricing. Provided below are the annual percentage change in lumber pricing for the past 11 years.

Year	Annual % Change
2022 (Year to Date, October 14th)	**-56.91%**
2021	**52.90%**
2020	**77.28%**
2019	**25.56%**
2018	**-24.15%**
2017	**35.58%**
2016	**27.47%**
2015	**-22.01%**
2014	**-10.47%**

2013	**-2.04%**
2012	**45.13%**

Data source: (Lumber prices - 50 year historical chart, n.d.)

This volatility in pricing affects not only the lumber market, but the markets that lumber is used in. Lumber and overall economic growth in the US are tightly connected. One example of this is the US housing market. One third of all US lumber is used in the construction of residential homes and apartment buildings (Farmer, 2021).

Justification for this question stems from a desire to be able to accurately model and predict lumber pricing, specifically during periods of US recession. This is because lumber price prediction has the potential to provide value to investors, lumber harvesters, manufacturers of lumber products, residential and commercial builders, and ultimately the end consumer of lumber products. Historical lumber pricing provides the date and daily closing price, and is well suited for Time Series modeling.

Hypotheses

The Null Hypothesis for this analysis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval cannot be made from the historical US Lumber pricing dataset.

The Alternate Hypothesis for this analysis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval can be made from the historical US Lumber pricing dataset.

A 20% confidence interval was selected to align the hypotheses with the research question. A common phrase heard in the investment community is "past performance is no guarantee of future results". In fact, it's required by the U.S. Securities and Exchange Commission that mutual funds inform their investors of this (Brown, 2016). Providing a confidence interval for prediction success, as compared to providing exact prediction values, helps align the results of ths analysis with the uncertainty that comes with forecasting lumber's price from past data.

B: Data Collection

The data collection process involved finding an accurate and publicly available data source for historic lumber pricing. This was performed through manually searching various financial news sites, such as Google Finance, Yahoo Finance, and Macrotrends until a data source was found. The data collected for this analysis came from one source, www.macrotrends.net. The data collected was downloaded as a .csv file that contains two variables:

Variable Name	**Type**	**# of Cases**
Date	Continuous	12,558
Value	Continuous	12,558

Macrotrends.net is a research platform for investors. They have data available on stocks, market indices, precious metals, interest rates, and commodities. Specifically for this analysis, macrotrends.net has historical data for the past 50 years of lumber pricing for trading days that is freely available for public and academic use to download (Lumber prices - 50 year historical chart, n.d.). To download the unprepared data, navigate to the lumber prices home page and click the 'Download Historical Data' button to download a .csv file containing historical lumber prices from 1972 to 2022.

One advantage of the methodology used is the level of ease in acquiring the data. Being able to download a .csv file with historical lumber prices provides a straight-forward starting point for this analysis. It is much more forward than having to utilize an API call or web scraper to access or collect the required data.

One disadvantage of the methodology used was the amount of manual searching required to find a source of data on lumber pricing that was suitable for this analysis. There are many different websites that populate a search for lumber pricing, but most do not have a downloadable dataset.

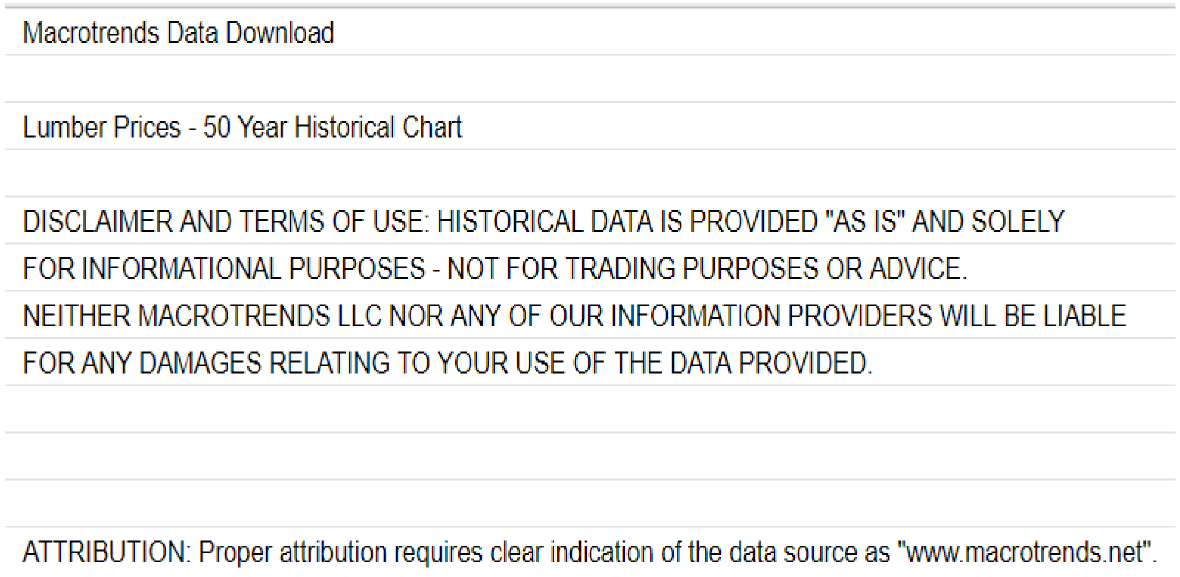
A challenge in the process of data collection was that It was a long manual process of looking through different websites for a downloadable source providing lumber pricing that was free to use, free of cost, and suitable for this analysis. This was overcome by allocating an appropriate amount of time for manually searching for the right dataset in the project planning stage of this analysis. There's no substitution for putting in the time during the search for a dataset. Having enough time to go through dozens of websites until this dataset was found was the critical component in overcoming this data collection challenge.

C: Data Extraction and Preparation

C1: Preparation of data in Google Sheets

The original dataset, sourced from www.macrotrends.net, is uploaded into Google Sheets for initial preparation. The dataset is in .csv format, with no special extraction methods necessary. Google Sheets is a spreadsheet program that is compatible with a number of file formats, including .csv and .xlsx.

Advantages of Google Sheets are that it's a free, fully functional web-based spreadsheet program on par with Microsoft's Excel. A disadvantage of Google Sheets compared to Excel is that it doesn't have as wide of a range of data visualization options (Simplilearn, 2022). This was not an issue for the analysis, as no visualization options were utilized in the spreadsheet program. The uploaded .csv file contains www.macrotrends.net's disclaimer, terms of use, and attribution, as shown in the screenshot below.



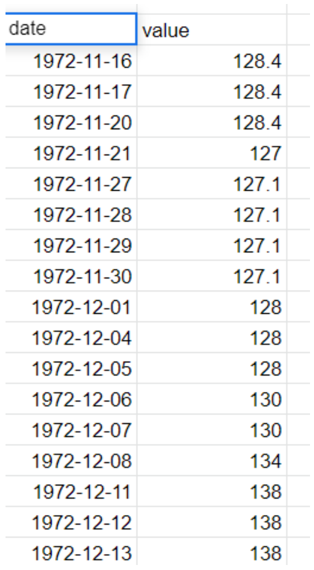
Macrotrends Data Download

Lumber Prices - 50 Year Historical Chart

DISCLAIMER AND TERMS OF USE: HISTORICAL DATA IS PROVIDED "AS IS" AND SOLELY
FOR INFORMATIONAL PURPOSES - NOT FOR TRADING PURPOSES OR ADVICE.
NEITHER MACROTRENDS LLC NOR ANY OF OUR INFORMATION PROVIDERS WILL BE LIABLE
FOR ANY DAMAGES RELATING TO YOUR USE OF THE DATA PROVIDED.

ATTRIBUTION: Proper attribution requires clear indication of the data source as "www.macrotrends.net".

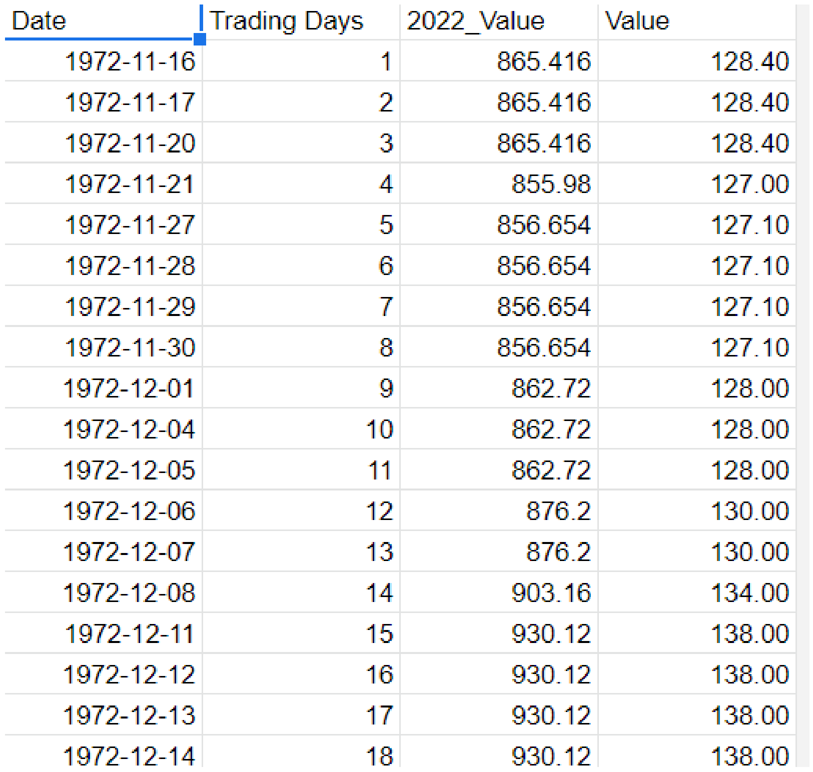
The rows containing the title, disclaimer, terms of use, and attribution information are deleted from the dataset in preparation for analysis. The dataset now contains two columns, 'date' and 'value'. An example of the formatting of the original variables is provided below.



date	value
1972-11-16	128.4
1972-11-17	128.4
1972-11-20	128.4
1972-11-21	127
1972-11-27	127.1
1972-11-28	127.1
1972-11-29	127.1
1972-11-30	127.1
1972-12-01	128
1972-12-04	128
1972-12-05	128
1972-12-06	130
1972-12-07	130
1972-12-08	134
1972-12-11	138
1972-12-12	138
1972-12-13	138

Variables 'date' and 'value' names are changed to 'Date' and 'Value'. Additionally, two variables are added to the dataset: '2022_Value' and 'Trading Days'. '2022_Value' contains inflation-adjusted lumber values. These were calculated by taking the 'Value' for each day, and multiplying it with the corresponding annual Consumer Price Index (USA historical consumer price index (CPI) - 1913 to 2022, n.d.) . This effectively provides a variable that has all lumber values, regardless of the year, equitable to 2022 lumber prices. This positively affects the stationarity of the data for time series analysis, and allows for direct comparisons to be made between different recessionary period lumber prices.

The data set does not contain any null values. However, the ‘Date’ column only has values for trading days. This means that weekend and holiday dates are not included in the dataset. Column ‘Trading Days’ is added to address this discontinuity in the dataset. ‘Trading Days’ will be used as the datetime index to provide continuity for time series analysis. The data contains only Quantitative variables. Provided below is a screenshot of the prepared dataset, with the changes made from above.



Date	Trading Days	2022_Value	Value
1972-11-16	1	865.416	128.40
1972-11-17	2	865.416	128.40
1972-11-20	3	865.416	128.40
1972-11-21	4	855.98	127.00
1972-11-27	5	856.654	127.10
1972-11-28	6	856.654	127.10
1972-11-29	7	856.654	127.10
1972-11-30	8	856.654	127.10
1972-12-01	9	862.72	128.00
1972-12-04	10	862.72	128.00
1972-12-05	11	862.72	128.00
1972-12-06	12	876.2	130.00
1972-12-07	13	876.2	130.00
1972-12-08	14	903.16	134.00
1972-12-11	15	930.12	138.00
1972-12-12	16	930.12	138.00
1972-12-13	17	930.12	138.00
1972-12-14	18	930.12	138.00

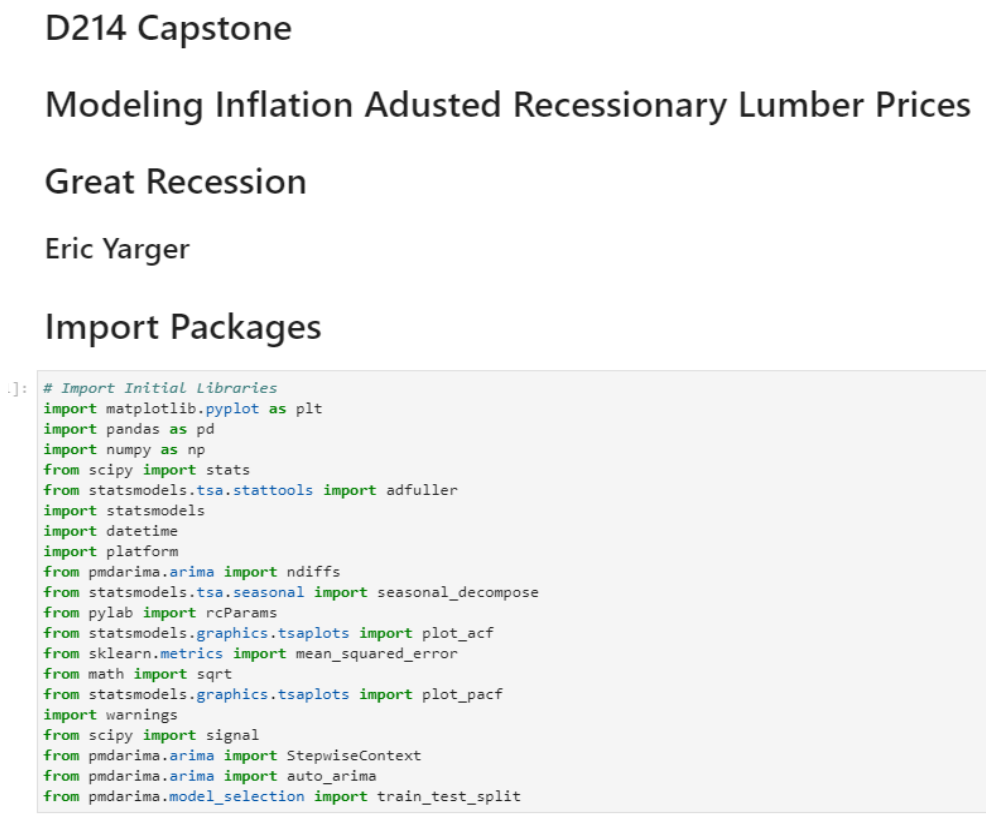
C2: Preparation of data in Jupyter Notebook, using Python

The dataset is saved as a .csv file, and uploaded into a Jupyter Notebook, accessed through IDE JupyterLab for further preparation and analysis. Python is the language selected for the remaining data preparation and analysis. Python is appropriate due to the author's familiarity with the language, coupled with Python's ability to clean and perform regressive Time Series analysis and modeling. In comparison to using R for this analysis, Python offers comparable auto_arima packages, and is advantageous as it is cited as being faster in ARIMA modeling execution (Manokhin, 2021). In comparison to using SAS for this analysis, Python has pmdarima's auto_arima module available free of charge. Comparable SAS auto_arima modeling tools, PROC HPFDIAGNOSE+PROC HPFENGINE (Li, 2019), would need to be licensed through SAS Forecast Server, which is only available to use for a fee.

A disadvantage of Python is that visualizations can be more complicated and less of a strength than in R (Python vs. R: What's the difference?, 2021). With this in mind, it is important to utilize a language that allows various packages and libraries to be installed in order to execute the intended analysis. In reviewing the option between Python, SAS, and R for this analysis, Python is the only language that meets all the above criteria for the author.

Jupyter Notebook Setup

The data file is loaded into a Jupyter Notebook, and is ready for analysis. There will be seven models created, one for each recession period. A Jupyter Notebook is created for each recession period. Each notebook is named after the recession period the model will forecast. The packages used for the analysis are imported into the notebooks. The environment and language version are identified, and the dataset file is read into the environment.



D214 Capstone

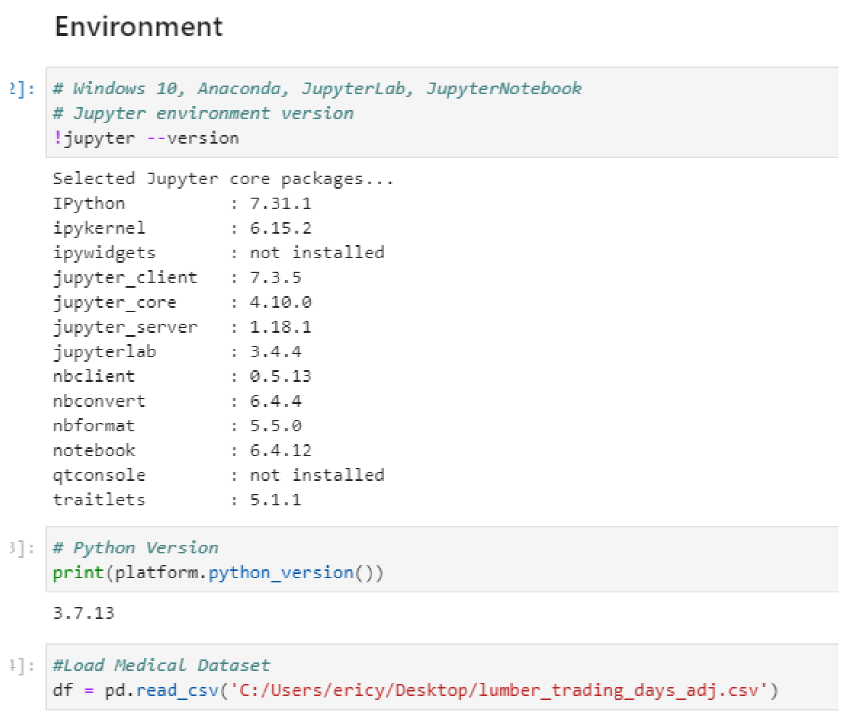
Modeling Inflation Adusted Recessionary Lumber Prices

Great Recession

Eric Yarger

Import Packages

```
# 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

```
# 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
```

```
# Python Version
print(platform.python_version())
```

```
3.7.13
```

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

Separating Datasets for each recession period

Next the dataset is separated into the individual set used for the forecasting of each specific recession period. There were seven total recessions during the time the dataset covers:

Recession Name	Beginning	Ended	Length of Recession In Months
1973 - 1975 Recession	November 1973	March 1975	16
1980 Recession	January 1980	July 1980	6
1981 - 1982 Recession	July 1981	November 1982	16
Early 1990's Recession	July 1990	March 1991	8
Early 2000's Recession	March 2001	November 2001	8
Great Recession	December 2007	June 2009	18
COVID-19 Recession	February 2020	April 2020	2

The recession beginning and end dates correspond with the dataset cases as:

Recession Name	Beginning Case	Ending Case	Length of Recession In Daily Cases
1973 - 1975 Recession	320	591	271

1980 Recession	1790	1937	147
1981 - 1982 Recession	2167	2525	358
Early 1990's Recession	4695	4864	169
Early 2000's Recession	7135	7324	189
Great Recession	8831	9221	390
COVID-19 Recession	11879	11941	62

Now that the recession periods have been determined in the dataset, an appropriate sample size must be determined for each. There is no universal rule for sample size selection, but a general rule of thumb in time series forecasting is that the larger the sample period, the better. However, if the selected time series goes too far in the past, the risk is run that the data is no longer indicative of the prices that are going to be predicted (Kolassa, 2016). To provide an adequate number of samples for each recession period, the sample size was decided to be equal to two years of daily prices before the beginning of the recession in addition to the recession time frame. This works well for all periods with exception of the 1973-1975 recession. The 1973-1975 recession was too close to the beginning of the dataset to have a full two years. Provided below is a breakdown of the dataset selected for each recessionary period.

Recession Name	**Total Dataset Size Cases**	**Pre-Recession Data Cases**	**In-Recession Data Cases**	**Forecast Length In Days**
1973-1975 Recession	591	249	342	191
1980 Recession	649	520	130	130
1981-1982 Recession	861	520	341	172
Early 1990's Recession	926	520	406	185
Early 2000's Recession	694	520	174	138
Great Recession	889	520	369	178
COVID-19 Recession	582	520	62	113

After timeframe selection for each Recession period, the datasets are now prepared for analysis.

D: Analysis

The analysis is broken down into 11 steps. The sections of this analysis are:

- D1: Exploratory Data Analysis
- D2: Time Step Formatting and Indexing
- D3: Stationarity Analysis
- D4: Differencing
- D5: Seasonal Decomposition
- D6: ACF and PACF
- D7: Spectral Density
- D8: Create Training and Test Datasets
- D9: ARIMA Modeling
- D10: Accuracy Metrics For The Forecasts
- D11: Visualizing Model Forecast Confidence Intervals at 20% CI

The analytical techniques implemented for each recession period are identical. Due to this, to increase the readability and functionality of this analysis only screenshots from the Great Recession period will be visualized in this report where screenshots of calculations and output are necessary. The exception to this is section D11, where the final model is provided for each Recession period.

The analysis results for each recession period will be summarized for each step. For full consideration of each Recession period, PDF copies of the coding environment are included with this submission. Each PDF is named after the recession it models.

D1: Exploratory Data Analysis

- Techniques used.
    - Df.head() provides a short preview of the dataset features and case examples.
    - Df.info() provides information on Non-Null counts for each feature and datatype.
    - Df.shape() prints the number of cases and features that comprise the dataset.
    - Df.describe() prints dataset descriptive statistics: count, mean standard deviation, minimum value per feature, maximum value per feature, and quartile values.

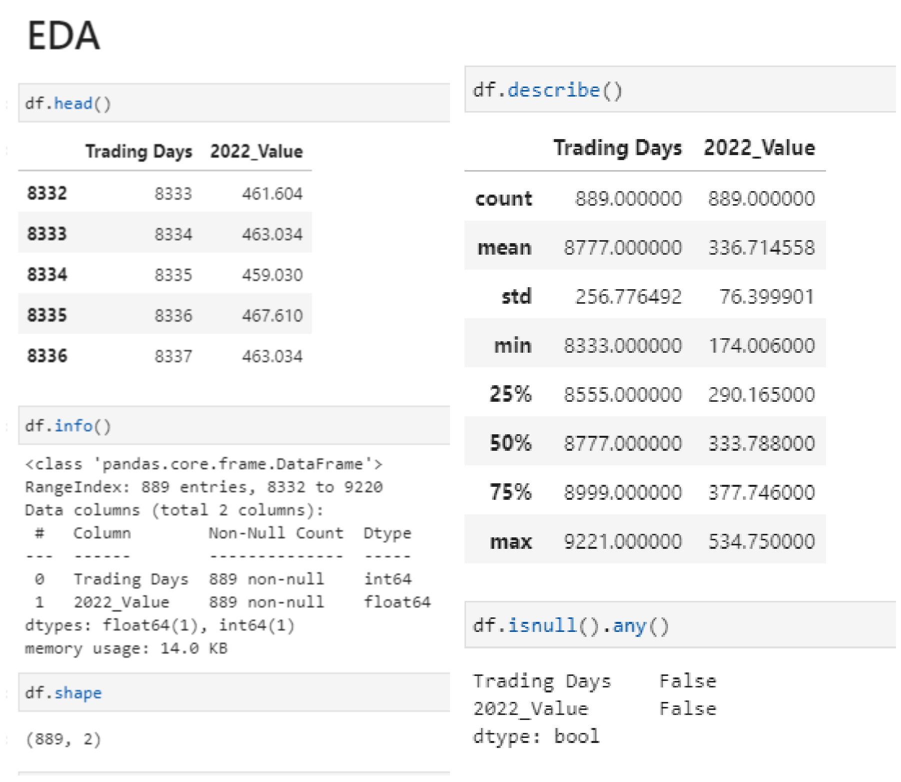
- Matplotlib.pyplot is used to make a line graph of the dataset.
- Df.dropna() drops null values from the dataset. For all datasets in this analysis, there were no null values to drop.

- Justification of technique.
  - EDA is the investigation of the data to explore, visualize, and summarize key statistical insights. This gives the analyst a basic starting point for understanding the data's distribution, null value count, and shape.
  - One advantage of performing EDA as a first step in data analysis is that it gathers insights that help the analyst gain a better understanding of the data. This knowledge helps to shape the remainder of the analysis.
  - One disadvantage of the techniques used is that Exploratory Data Analysis is a very broad category, with many tools and techniques. Correct technique selection for specific analyses comes from both research and analyst experience. If the wrong techniques are implemented the dataset can be explored insufficiently for the analysis.
- Results. From the line graph visualization it can be seen that all recessionary periods exhibit downward trends in price over time. All recessionary periods have no missing values. Descriptive statistics for each period are as follows:

Recession Name	**Total Dataset Size**	**2022_Value Mean**	**2022_Value Min**	**2022_Value Max**	**2022_Value Standard Deviation**
1973-1975 Recession	591	848.87	588.65	1111.32	119.66
1980 Recession	649	905.57	579.64	1182.34	106.34
1981-1982 Recession	861	627.13	373.39	1182.34	195.49
Early 1990's Recession	926	413.59	316.92	508.39	29.82
Early 2000's Recession	694	475.44	305.02	684.52	93.31
Great Recession	889	336.71	174.00	534.75	76.40

COVID-19 Recession	582	445.87	275.39	709.29	91.04

- Calculations and Outcomes. Provided below are the calculations and outputs for this section with examples of the EDA performed.



EDA

```
df.head()
```

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

```
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
```

```
df.shape
```

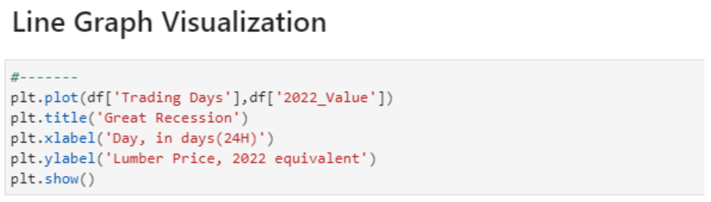
```
(889, 2)
```

```
df.describe()
```

```
       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
```

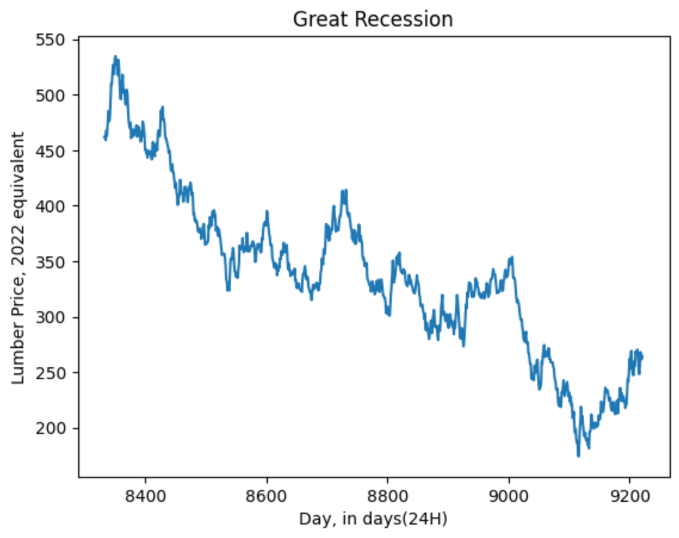
```
df.isnull().any()
```

```
Trading Days    False
2022_Value      False
dtype: bool
```



Line Graph Visualization

```
#-------
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()
```



D2: Time Step Formatting, Indexing

- Techniques used.
    - Pandas.to_datetime was used to convert feature 'Trading Days' to datetime with the unit specified as 'D' for days.
    - Pandas.set_index was used to set 'Trading Days' as the dataset index.
- Justification of technique.
    - Preparation for ARIMA modeling in Python requires the data to be formatted in a specific manner. There must be a feature with datetime datatype. This feature must be assigned as the dataset index.
    - One advantage of these techniques is that it provides a quick and effective method for time step formatting. pd.to_datetime does a good job of inferring datetime format automatically. This prepares the data for further analysis.
    - One disadvantage of this technique is that pd.to_datetime can be burdensome to get the correct dates to output. This means that the dates going in do not match the actual date periods at this point in the analysis. This is addressed in the delimitations from the analyst's capstone proposal, and is noted in the coding environment.
- Results, Calculations, and Outcomes. The method for each recession period for this section is identical. This results in each recession having a dataset that is prepared for Time Series modeling. Provided below are the calculations and outputs for the Great Recession analysis.

D2: Time Step Formatting, Indexing

Set df['Trading Days'] to Index

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

```
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
```

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

```
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

```
df
```

	2022_Value
Trading Days	
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

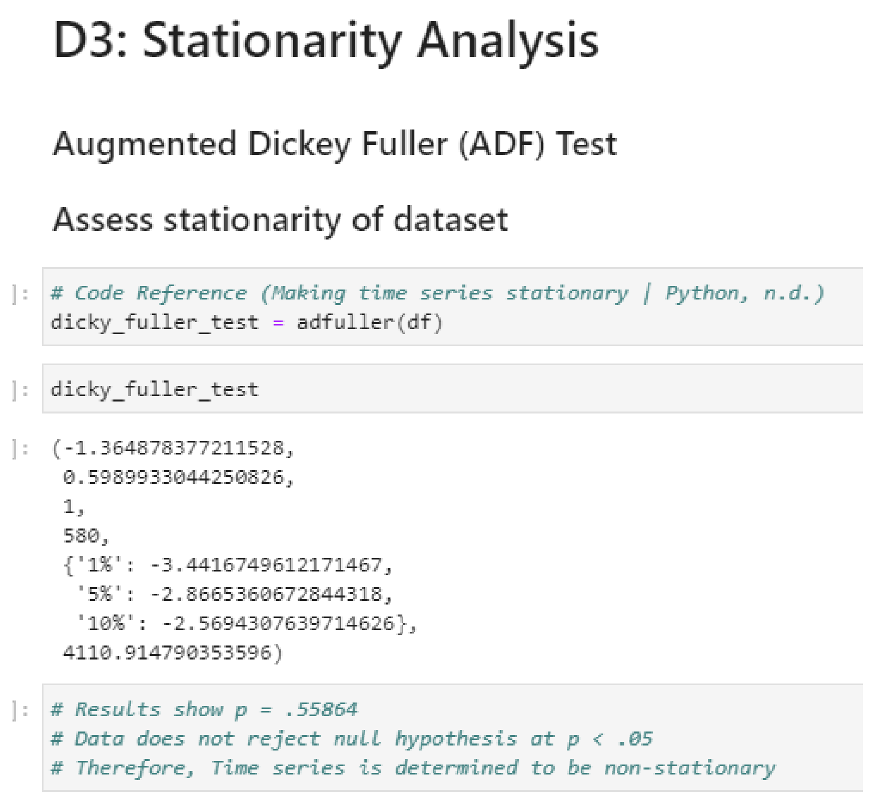
889 rows × 1 columns

D3: Stationarity Analysis

- Technique used.
  - Statsmodels.tsa.stattools_adfuller for performing an Augmented Dickey-Fuller (ADF) Test.
- Justification of technique.
  - ADF Test is used widely in statistics and econometrics for testing the null hypothesis that a unit root is present in a time series sample. The alternative hypothesis is that the data exhibits stationarity (Augmented Dickey–Fuller test, n.d.). This technique helps identify the stationarity of the data, and if differencing will be required. The more negative the ADF statistic, the stronger the rejection of the null hypothesis. The technique used in this analysis also returns the p-value. If the p-value is < .05, the null hypothesis is rejected.
  - One advantage of the ADF test is that it is a proven statistical test for determining the stationarity of data (Augmented Dickey–Fuller test, n.d.).
  - One disadvantage of the ADF test is that it has low statistical power when distinguishing true unit root processes and near root processes (Augmented Dickey-Fuller (ADF) test, n.d.).
- Results, Calculations and Outcomes. Provided below is a table for each recession with the results of this section.

Recession Name	**ADF statistic**	**P-Value**	**Accept or Reject Null Hypothesis**	**Result**
1973-1975 Recession	-1.83	.37	Accept	Data is not stationary
1980 Recession	-1.81	.37	Accept	Data is not stationary
1981-1982 Recession	-1.54	.51	Accept	Data is not stationary
Early 1990's Recession	-2.31	.17	Accept	Data is not stationary
Early 2000's Recession	-1.11	.71	Accept	Data is not stationary
Great Recession	-1.55	.51	Accept	Data is not stationary
COVID-19 Recession	-1.36	.60	Accept	Data is not stationary

Provided below are the calculations and outputs for this section.



D3: Stationarity Analysis

Augmented Dickey Fuller (ADF) Test

Assess stationarity of dataset

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

```
]: dicky_fuller_test
```

```
]: (-1.364878377211528,
    0.5989933044250826,
    1,
    580,
    {'1%': -3.4416749612171467,
     '5%': -2.8665360672844318,
     '10%': -2.5694307639714626},
    4110.914790353596)
```

```
]: # Results show p = .55864
   # Data does not reject null hypothesis at p < .05
   # Therefore, Time series is determined to be non-stationary
```

D4: Differencing

- Technique used.
  - Pandas.df.diff() is used for first and second order differencing of dataset.
  - Matplotlib.pyplot() is used to visualize the differencing results.
  - Pmdarima's ndiffs function is used to double check the results of differencing.
- Justification of technique.
  - Stationarity is required for appropriate time series analysis. It was established the data for each recession period is not stationary in section D3. Pmdarima's auto_arima will automatically select the optimal differencing order for the time series during modeling (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation, n.d.). However, for the purpose of full analysis of the time series, differencing is examined and analyzed before modeling. This provides the analyst with important information that can be useful for fine-tuning the model, if necessary.

- One way of addressing non-stationarity is through analyzing the effect differencing has on the time series. Differencing is a method of transforming a non-stationary time series into a stationary one (*Identifying the order of differencing in ARIMA models*, n.d.). Pandas df.diff() calculates the difference of the data elements compared with the element in the previous row. This data is differenced once (first order differencing) and again (second order differencing). The results of differencing are best determined visually. Pyplot is used to create the original and differenced visualizations.
- Pmdarmia's ndiffs accepts the data as input and automatically determines the optimal differencing order 'd' value. In the case of the Great Recession it returns 1 for the best 'd'. This is useful for providing a statistically sound decision making process for identifying the optimal differencing requirement of the ARIMA model.
- One advantage of pandas.diff() is that it is a simple calculation that can make a large difference on the stationarity of the data. It is advantageous to use two techniques to verify results, as was performed in this analysis by using ndiffs as well.
- One disadvantage of pandas.diff() is that it relies on the analyst to visually interpret the results of differencing and choose the appropriate value. This introduces human error into the analysis.

- Results of differencing analysis

Recession Name	Estimated Differencing Order
1973-1975 Recession	1
1980 Recession	1
1981-1982 Recession	1
Early 1990's Recession	1
Early 2000's Recession	1
Great Recession	1
COVID-19 Recession	1

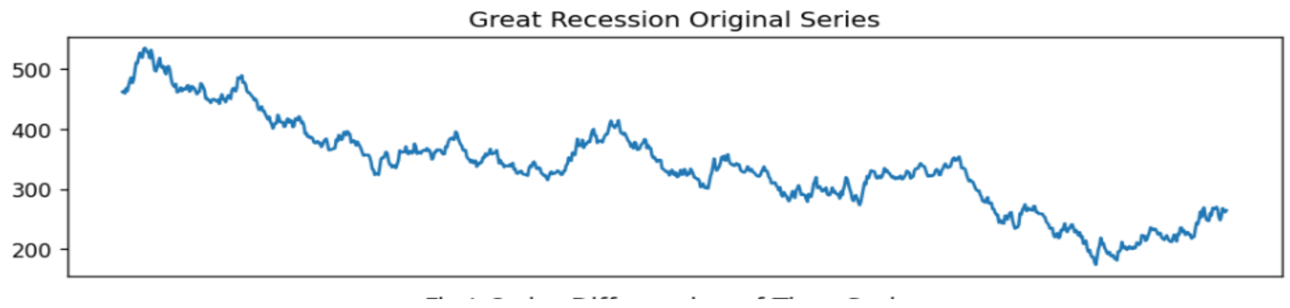
- Calculations and Outcomes. Provided below are the calculations and outputs for this section.

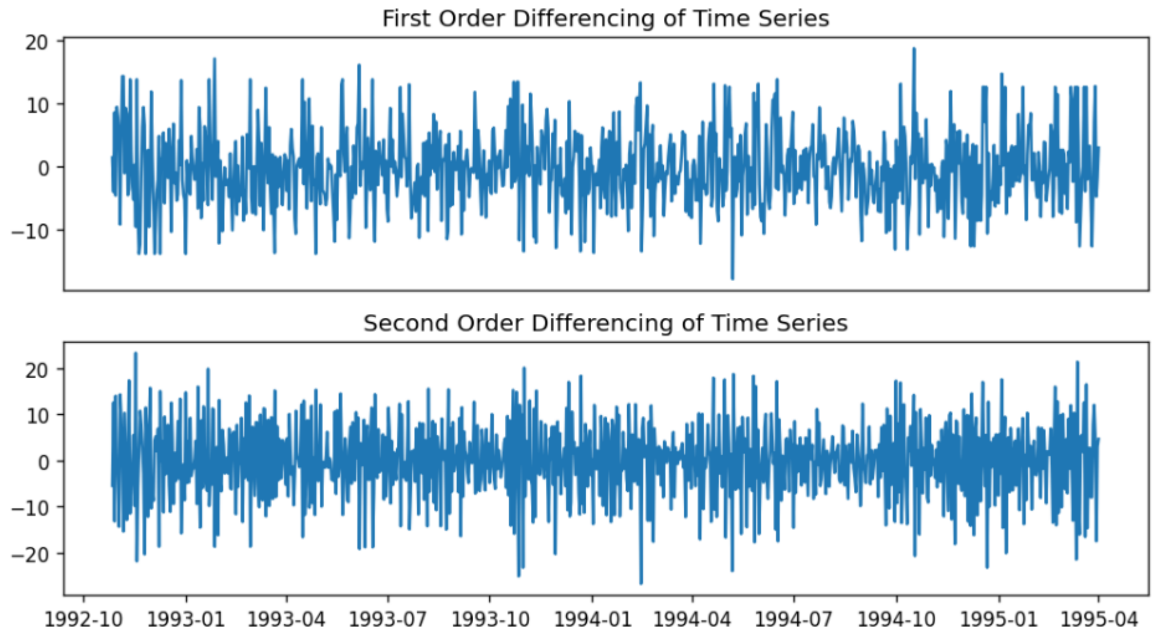
D4 Differencing

1st and 2nd order Differencing

finding 'd' for ARIMA model

```
# 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()
```





```
# 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: Seasonal_decomposition Analysis

- Technique used.
    - Statsmodels.tsa.seasonal's seasonal_decompose is used for seasonal decomposition.
- Justification of technique.
    - Seasonal_decompose breaks down time series data into its components. This makes it easier to determine the trend, seasonality, and residual distribution of the time series. The trend captures the slowly moving overall level of the series. The seasonal component captures patterns that repeat with seasonal frequency. The residuals are what is left over after capturing trend and seasonality of the time series. Using seasonal_decompose provides the analyst with a more thorough lens to analyze the time series in preparation for modeling.
    - One advantage of seasonal_decompision is that it functions seamlessly with matplotlib pyplot for visualizing the resulting seasonal decomposition. It also provides the analyst with valuable information regarding the seasonality and residual distribution of the time series.
    - One disadvantage of statsmodel's seasonal_decompose is that it is described as naive in the documentation (Statsmodels.tsa.seasonal.seasonal_decompose — statsmodels, n.d.). This is because it uses simple convolution and averages to separate out the seasonal structure.
- Results.
    - The resulting plots from this step provide visual indication of the time series trend, seasonality, and residuals over time.
        - The overall trend is noted for each time series.
        - Seasonality can come in many forms: daily, weekly, monthly, yearly, and in many other time period formations. It can be difficult to determine the correct seasonality from visualizations. For each Recession period seasonality is noted as indeterminable by the author. Pmdarima's auto_arima is designed to perform statistical tests to determine seasonality in time series (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation, n.d.), and will be utilized to do so for each recession period in this analysis.

- Residuals should optimally resemble white noise-like distribution for ARIMA modeling (Brownlee, 2020). Volatility in time series over time can affect this. Recessions exhibit more volatility on commodity pricing (Vasishtha, 2022). This means that the time series used for this analysis are more likely to exhibit imperfectly distributed residuals. To note this, the author will give a rating of 'Excellent'', 'Fair', or 'Poor' to describe the residual distribution of each time series.

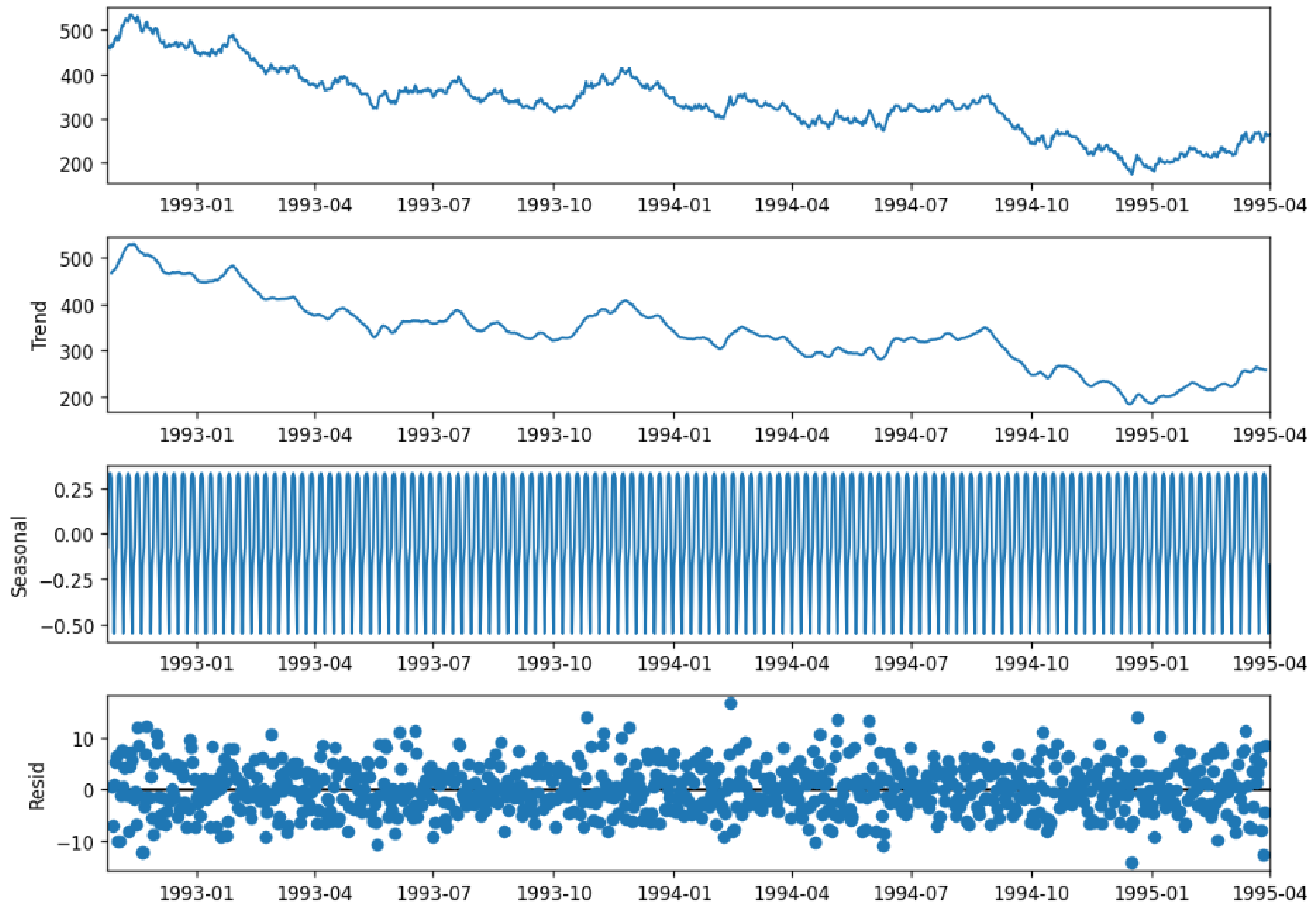
Recession Name	**Trend**	**Seasonality**	**Residuals**
1973-1975 Recession	Downward	Undetermined	Fair
1980 Recession	Downward	Undetermined	Fair
1981-1982 Recession	Downward	Undetermined	Poor
Early 1990's Recession	Stationary	Undetermined	Poor
Early 2000's Recession	Downward	Undetermined	Fair
Great Recession	Downward	Undetermined	Fair
COVID-19 Recession	Downward	Undetermined	Fair

- Calculations and Outputs

D5 Seasonality Analysis

```
# Code Reference (Boston, 2020)
result = seasonal_decompose(df)
```

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



D6: ACF and PACF

- Technique used.
    - Statsmodels.graphics.tsaplots plot_acf for plotting autocorrelation function.
    - Statsmodels.graphics.tsaplots plot_pacf for plotting partial autocorrelation function.
- Justification of technique.
    - After the time series has been made stationary by differencing, the next step in fitting an ARIMA model is to decide whether AR and MA terms will be needed. AR and MA terms will be necessary if there is any autocorrelation that remains in the time series that needs to be

accounted for and corrected (*Identifying the orders of AR and MA terms in an ARIMA model*, n.d.). Autocorrelation function (ACF) is plotted to visualize the time series to determine if an MA term is necessary. Partial autocorrelation function (PACF) is plotted to visualize the time series to determine if an AR term is necessary. Plot_acf and plot_pacf are calculated and plotted for the original time series, first-order differentiated series, and second-order differentiated series.

  - The advantage of plotting plot_acf is that it visualizes the coefficients of correlation between a time series and lags of itself. This makes it easier to determine if there is autocorrelation in the time series. The same can be said for plotting plot_pacf, but for the partial correlation in the time series .
  - One disadvantage of techniques that check autocorrelation is when autocorrelation is small, they may not be powerful enough to reject the hypothesis that the time series is free of autocorrelation (Levich, Rizzo, 1998). A second disadvantage is that selecting AR and MA terms from analyzing ACF and PACF plots invites user error into term selection. This means that the selection can be incorrect for the time series. This is suboptimal compared to auto_arima, which automatically selects the optimal terms for the ARIMA model through testing and conducting differencing tests (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation, n.d.).

- Results: Analyst estimates for MA and AR term from ACF and PACF

Recession Name	Estimated MA Term (ACF)	Estimated AR Term (PACF)
1973-1975 Recession	1	1
1980 Recession	2	1
1981-1982 Recession	1	1
Early 1990's Recession	1	1
Early 2000's Recession	1	2
Great Recession	1	1
COVID-19 Recession	1	0

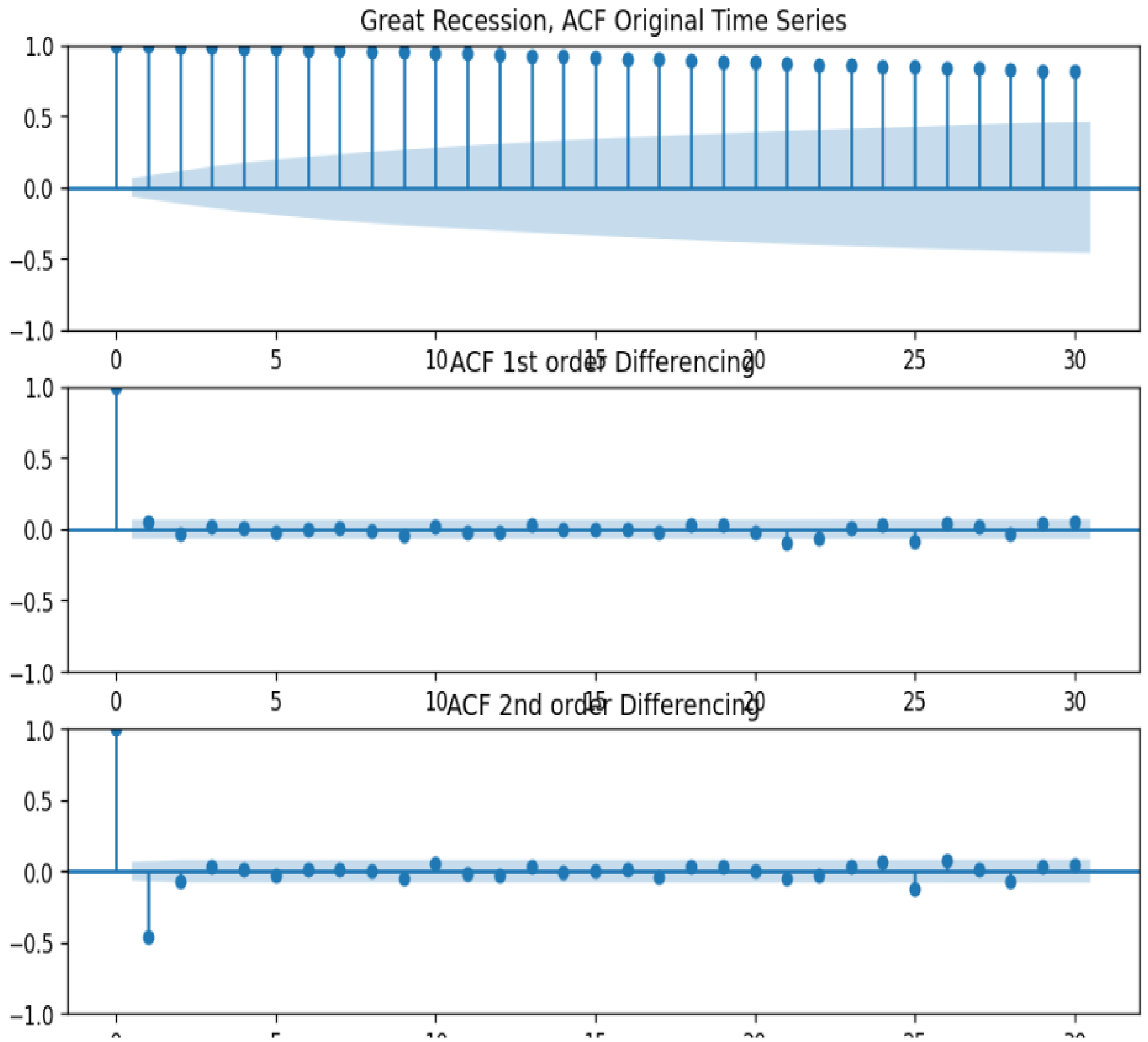
- Calculations and Outcomes. Provided below are the calculations and outputs for this section.

D6 ACF and PACF

Finding order of MA term 'q'

Using Autocorrelation function (ACF)

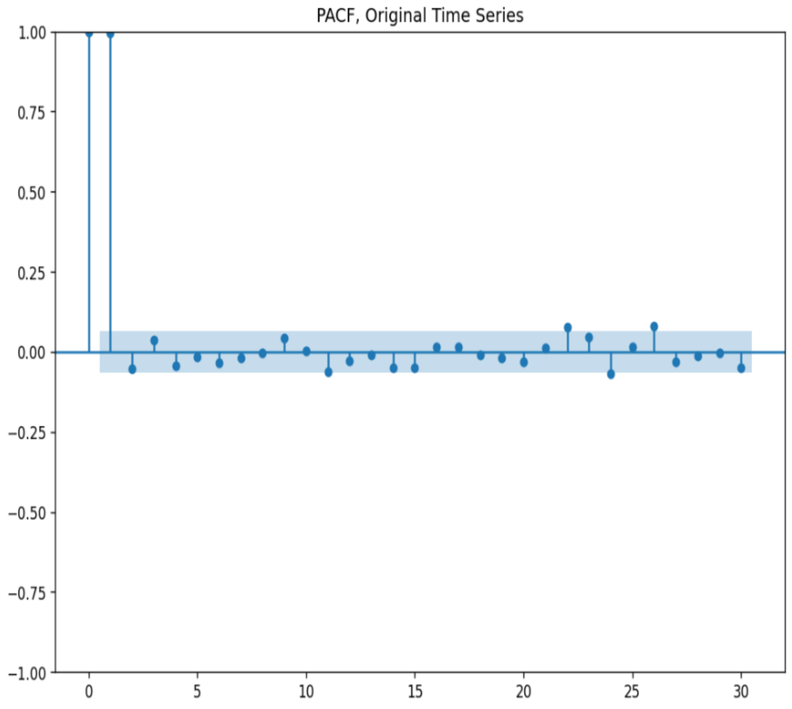
```
]: fig, (ax1, ax2, ax3) = plt.subplots(3)
   plot_acf(df, ax=ax1, title='Great Recession, ACF Original Time Series');
   plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
   plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```



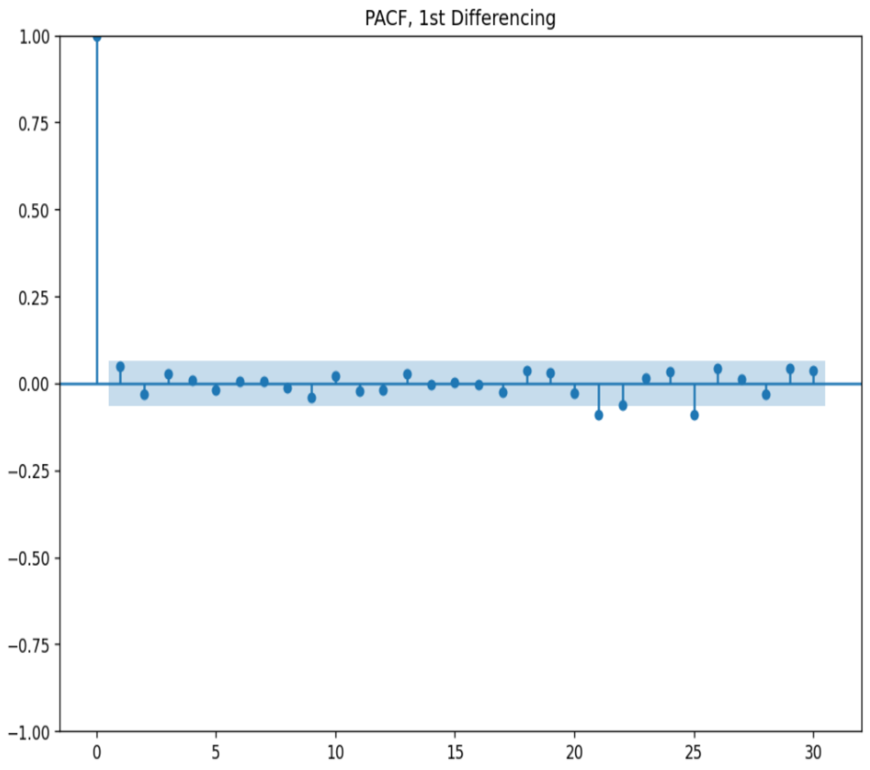
Finding order of AR term 'p'

Using Partial autocorrelation (PACF)

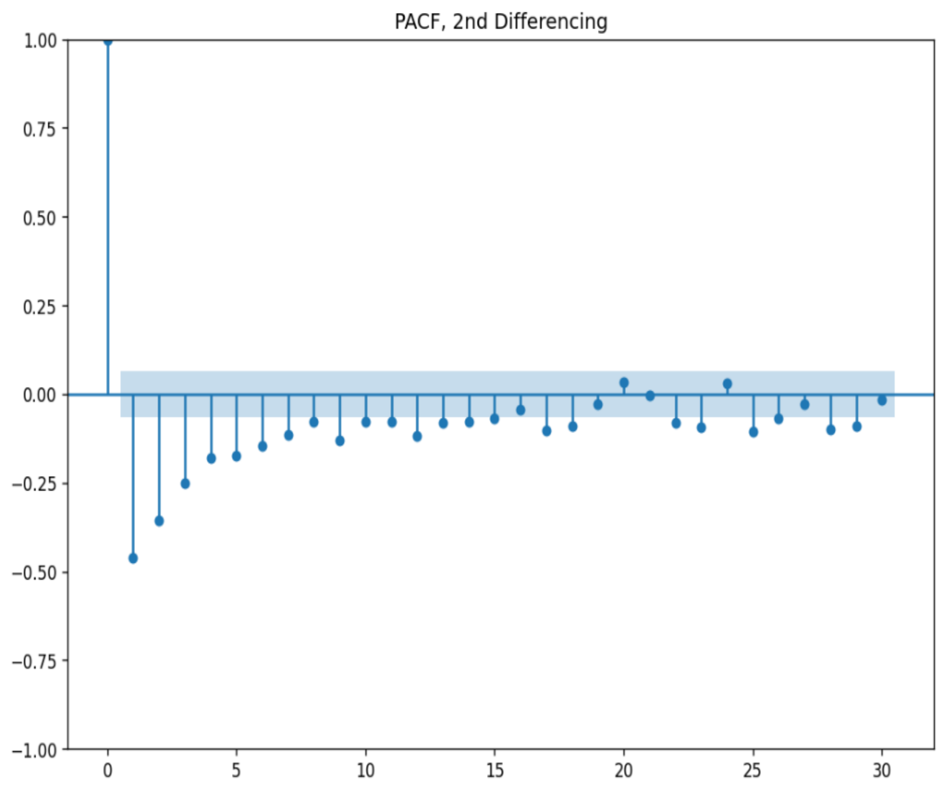
```
warnings.filterwarnings("ignore")
plot_pacf(df.dropna(), title='PACF, Original Time Series');
```



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



```
plot_pacf(df.diff().diff().dropna(), title='PACF, 2nd Differencing');
```



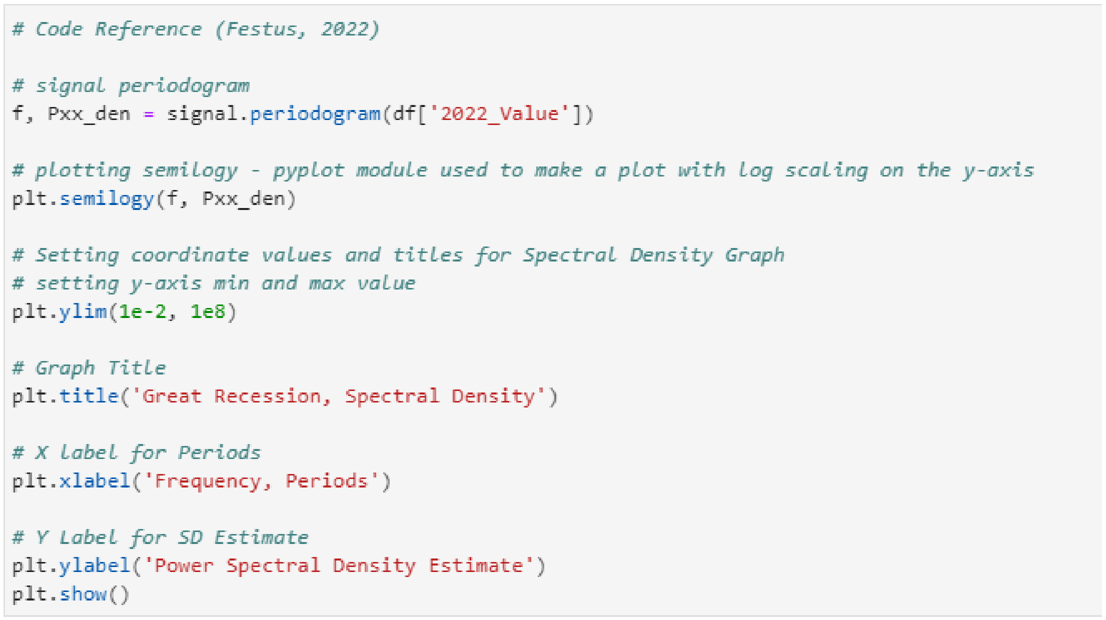
D7: Spectral Density

- Technique used.
    - Scipy's signal.periodogram() is used to plot a periodogram of the time series.
- Justification of technique.
    - This technique takes a time series of values and estimates power spectral density. This is used to help identify the dominant periods of a time series. This is helpful for identifying the dominant cyclical frequencies, or seasonality, in the series.
    - One advantage of using a periodogram is that it provides an additional technique for identifying cyclical trends in the time series (Periodogram, n.d.).
    - One disadvantage of periodograms is that the variance at a given frequency doesn't decrease, even as the number of samples used increases. This can lead to it not providing the averaging needed to analyze noise like signals (Periodogram, n.d.).
- Results. Spectral Density for each recession period was determined to provide inconclusive evidence of specific seasonality. To compensate for possible seasonality, argument 'seasonal' in pmd_arima's auto_arima is left with the default 'True'. This allows auto_arima to test for seasonality using the

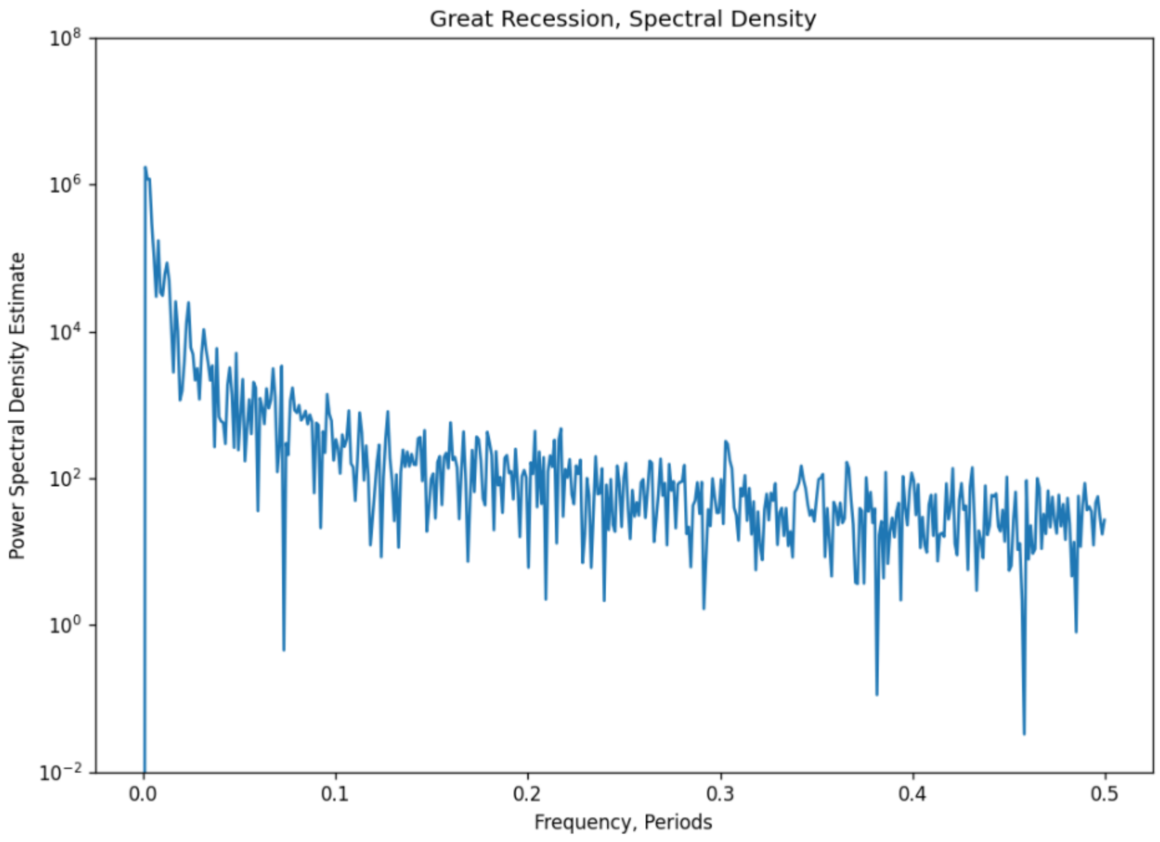
Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation. (n.d.).

- Calculations and Outcomes. Provided below are the calculations and outputs for this section.

D7 Spectral Density

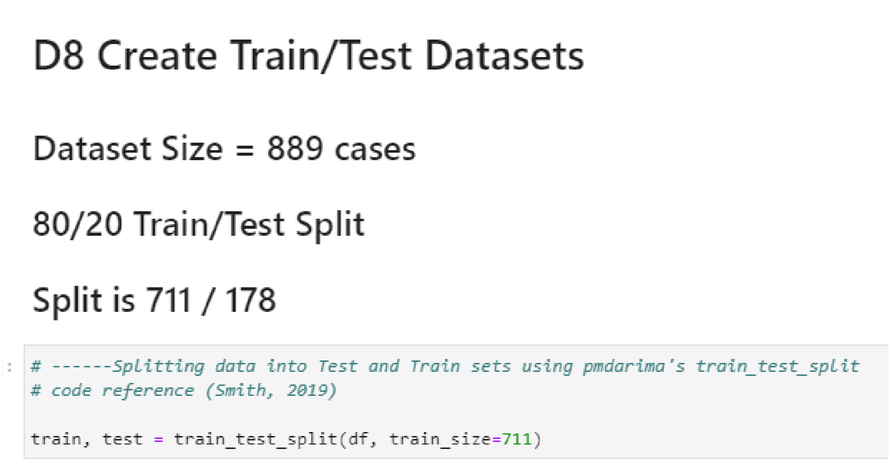


```
# Code Reference (Festus, 2022)

# 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 Training and Testing Sets

- Technique used.
    - pmdarima.model_selection's train_test_split is used to split the data into training and test sets.
- Justification of technique.
    - train_test_split provides an easy to use function for splitting data into testing and training sets. The time series data is split into 80% training and 20% testing sets for each recession analysis.
    - One advantage of using train_test_split is that it's an elegant method for splitting data into training and testing sets. It does in one line of code what it would take two lines if done manually.
    - One disadvantage of using train_test_split over manually splitting the data is that the shape of the test set needs to be printed to determine the size, in cases, if not known beforehand.
- Result. The time series is split into training and test sets. The training set comprises 80% of the time series length, the test set comprises 20% of the time series length.
- Calculations and Outcomes. Provided below are the calculations and outputs for this section.



D8 Create Train/Test Datasets

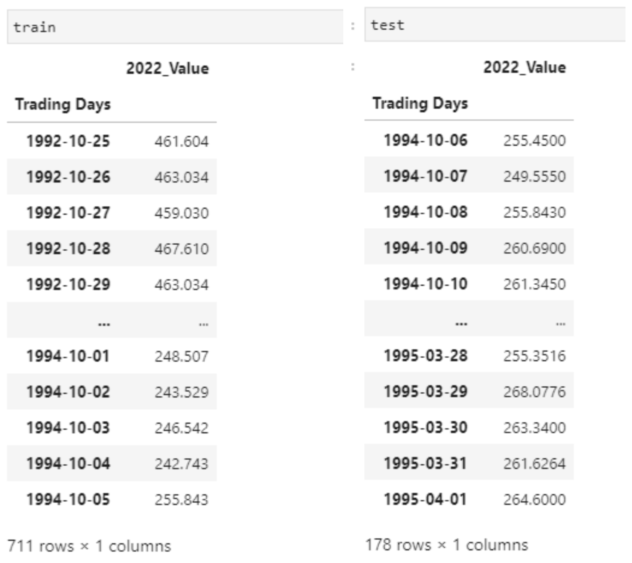
Dataset Size = 889 cases

80/20 Train/Test Split

Split is 711 / 178

```
# ------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)
```

Training and Test example and shape.



```
train
```

Trading Days	2022_Value
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

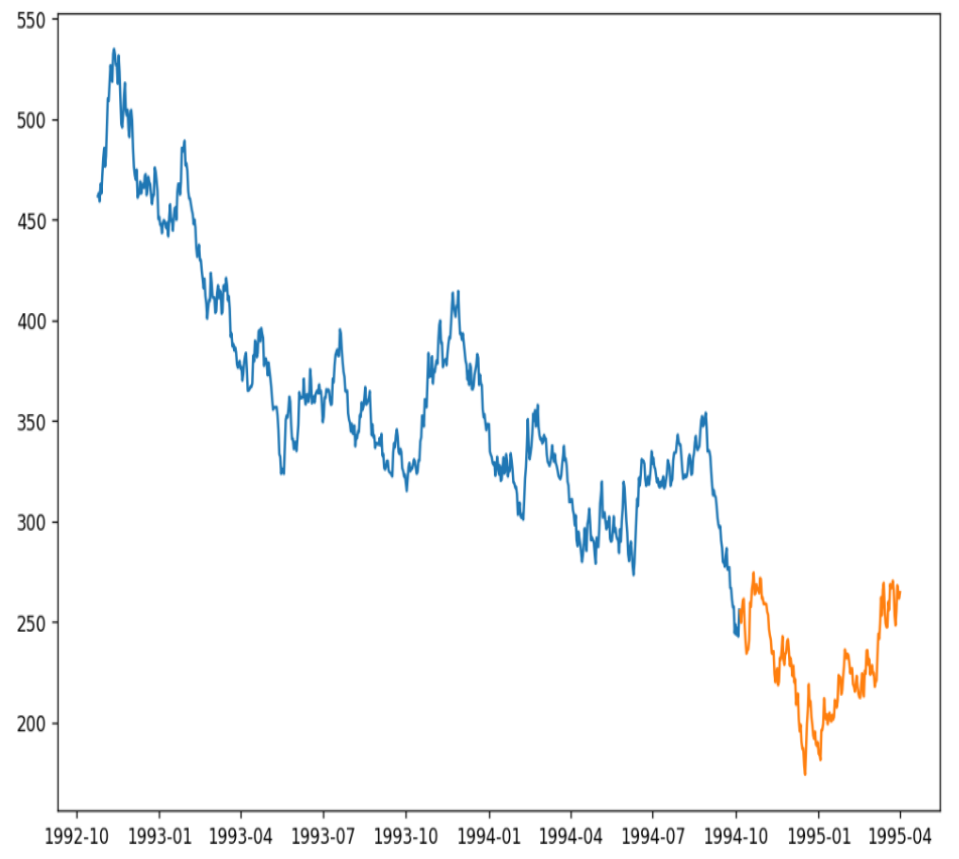
711 rows × 1 columns

```
test
```

Trading Days	2022_Value
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

Training and Test sets graphed for Great Recession. Training set is the blue line, Test set is the orange line.



D9: ARIMA Modeling

- Technique used.
  - Pmdarima.arima.auto_arima is used to model the data.
- Justification of technique.
  - pmdarima's auto_arima automatically discovers the optimal order for ARIMA models. It does so by seeking to identify the best parameters, and ultimately ends in settling on a single ARIMA model fitted to the training data. This provides justification in selecting the optimal model for the training data by fitting multiple models, and selecting the best one. Model selection criteria for auto_arima is specified under argument 'information_criterion', which is set to select the model with the lowest Akaike Information Criteria (AIC) score by default. AIC score is a good way to measure the goodness of fit of the model, while simultaneously penalizing the model for overfitting of the data (*The Akaike information criterion,* n.d.). AIC is the information_criterion used for model selection for every Recession period model.
  - One advantage of using auto_arima over a manual ARIMA modeling technique is that the auto_arima comes with many built in differencing and statistical tests that are performed. This aids in selecting the optimal ARIMA model for the time series (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation. (n.d.).
  - One disadvantage of auto_arima is that stationarity issues in the time series can cause auto_arima to not find a suitable model, resulting in the inability to converge (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation. (n.d.).
- Calculations and Outcomes.
  - The author was skeptical of letting auto_arima select the optimal model, and as such auto_arima was run twice and a model generated twice for every recession period.
    - The first model is trained with auto_arima arguments that are specified by the author. These specifications are informed by the analysis steps performed prior to modeling. Seasonal is set to 'True' to force auto_arima to assess seasonality. Stationarity is set to 'False', which allows auto_arima to assess different differencing terms for the model. Trace is set to 'True' to print the status on ARIMA model fits. Max values for terms p, d,

and q are set to 3 to avoid the model running longer than necessary, but to allow for a wide number of models to be tested. Seasonal_test is set to 'ocsb' to perform the Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots. 'Stepwise' is assigned for the selection algorithm.

- The second model allows auto_arima to have total control of the results. Argument 'trace' is set to 'True' to provide feedback on model fits, and all else is left to auto_arima.
- The results between the 'specified' model and the 'automatic' model are clear: the automatic model provides a better outcome for every recession period but the recession of 1980. This is assessed by the model's Akaike Information Criterion (AIC) score. The lower the AIC score, the more accurately the model models the data. The second 'automatic' model had a lower AIC score for 6 of the 7 recessions.

- Results. The table below outlines the results of both models, for all recessionary periods.

Recession Name	Specified Model AIC Score	Automatic Model AIC Score
1973-1975 Recession	4078.620	4076.950
1980 Recession	4214.481	4215.885
1981-1982 Recession	5462.260	5462.103
Early 1990's Recession	4221.106	4219.164
Early 2000's Recession	3861.975	3860.037
Great Recession	4541.481	4541.477
COVID-19 Recession	3402.640	3400.710

- Auto_arima also runs a Jarque-Bera test. This tests whether the dataset has the skewness and kurtosis that matches a normal distribution, effectively testing the normality of the data. The score is always a positive number, and the closer it is to 0, the more the data can be described as having a normal distribution. Provided below are the Jarque-Bera scores and the analyst's interpretation of it for each recession period.

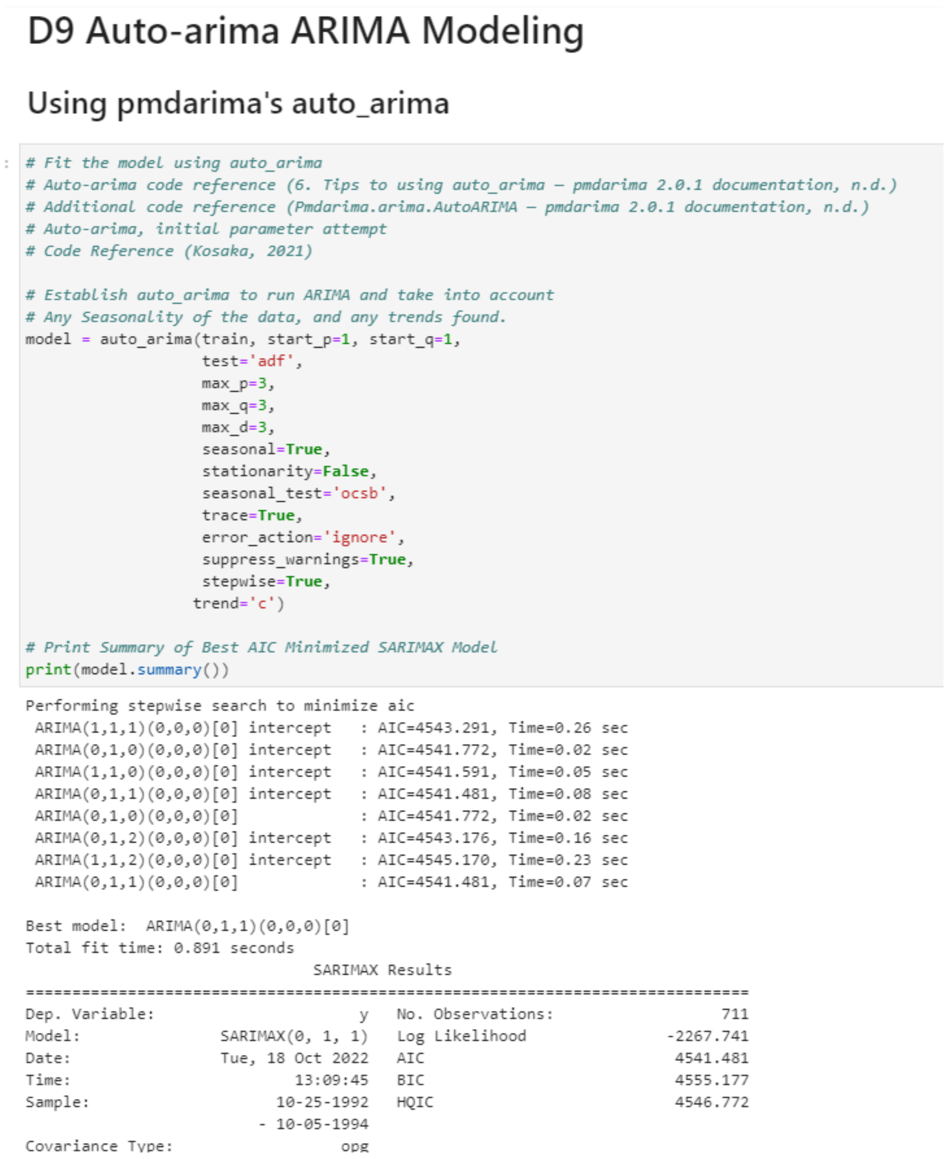
Jarque-Bera Score, test for normality for each Recession period

Recession Name	Jarque-Bera Score	Is the data normally

		distributed?
1973-1975 Recession	4.19	Approaching a normal distribution
1980 Recession	582.52	No
1981-1982 Recession	343.36	No
Early 1990's Recession	10487.19	No
Early 2000's Recession	2.21	Approaching a normal distribution
Great Recession	3.63	Approaching a normal distribution
COVID-19 Recession	8.23	Approaching a normal distribution

- Provided below is are screenshots of this process

Specified Model, Great Recession



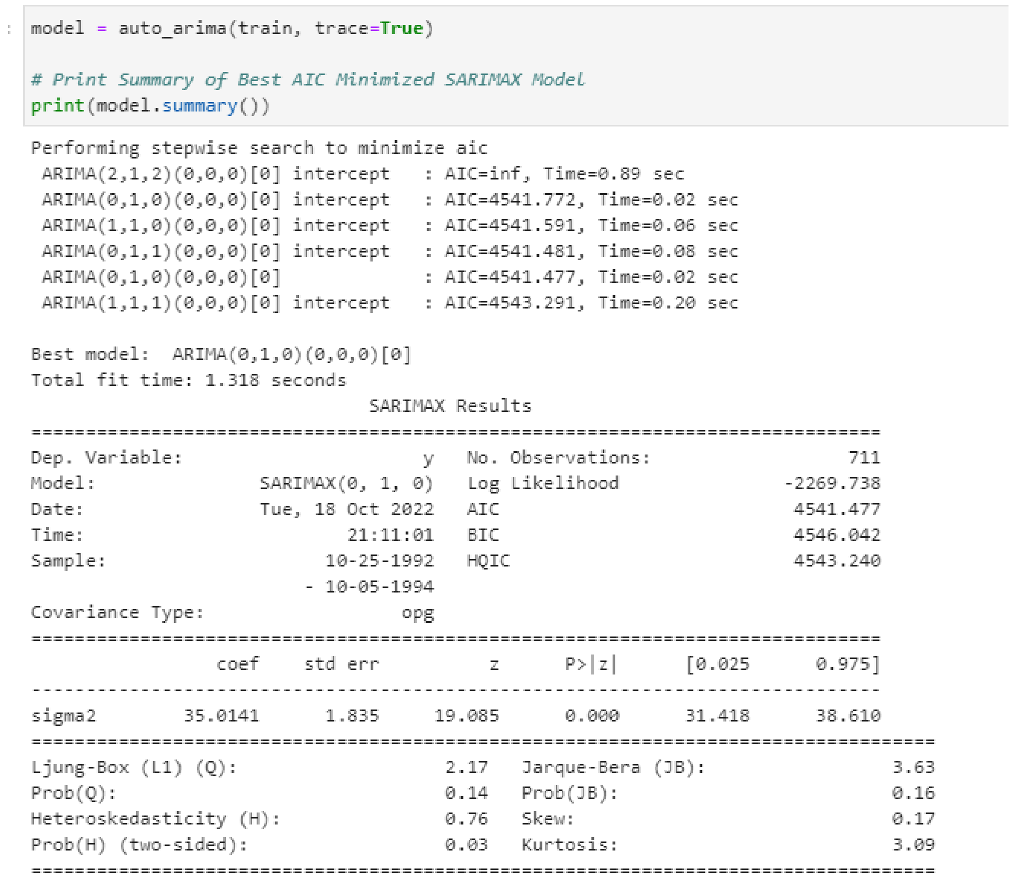
D9 Auto-arima ARIMA Modeling

Using pmdarima's auto_arima

```
# 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.
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
```

Automatic Model, Great Recession



```
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
===================================================================================
```

D10: Accuracy Metrics for Forecast

- Techniques used.
  - Root-mean-square-error (RMSE)
  - Mean absolute error (MAE)
- Justification of technique.
  - Calculating root-mean-square error calculates square root of the differences between predicted and observed values. The difference between the two products are called residuals, or in the case of estimated out-of-sample estimates, prediction errors (Root-mean-square deviation, n.d.). RMSE is always non-negative, and the closer the value is to 0 the better the forecasted values of the models compare to actual predictions. RMSE was selected as an accuracy metric for this analysis for the purpose of quantifying the model's forecasts into an industry relevant metric for model evaluation.

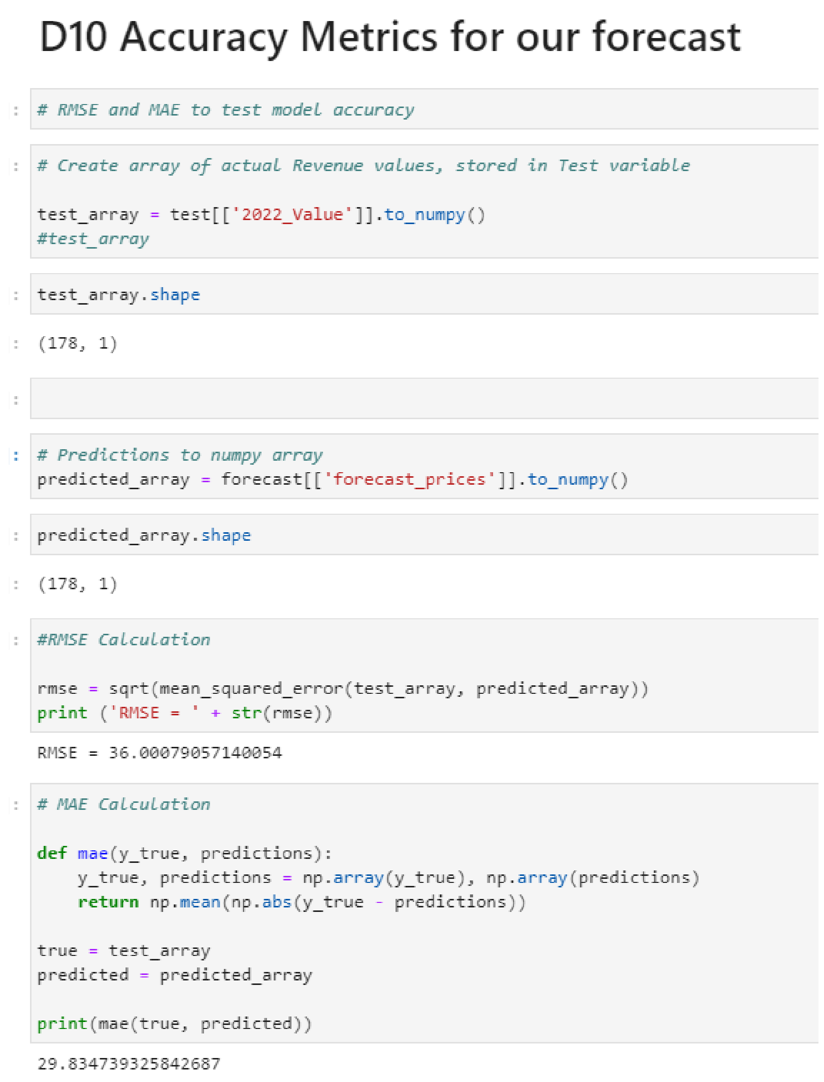
- MAE is calculated as the sum of absolute errors divided by the sample size. This returns the average absolute distance between the forecasted data point and the actual data point. The lower the MAE value for the model, the lower the absolute overall forecast error of the model is in its predictions (Mean absolute error, n.d.). This metric was used in conjunction with RMSE to provide two separate evaluation metrics for statistically analyzing the ARIMA models.
- One advantage of RMSE and MAE is that both are well known and widely used measures for evaluating the performance of time series model predictions (Calzone, 2022).
- One disadvantage of using RMSE as an evaluation metric is that the resulting RMSE value can be sensitive to being skewed by outlier values in the data (Padhma, 2021). This can result in an inaccurate evaluation metric to assess the model by.

- Results for both metrics are provided below. Mean Lumber Value is taken from the descriptive statistics provided in section D1 of this analysis. This provides a point of comparison for RMSE and MAE for each Recession period. The 1981-1982 Recession model exhibits the best RMSE and MAE compared to Lumber prices for the period. The RMSE and MAE for the remaining Recession periods compared to the mean value for the period indicates that the models do a poor job of predicting lumber prices compared to the actual price.

RMSE and MAE for each Recession period

Recession Name	Mean 2022_Value For the Period	RMSE	MAE
1973-1975 Recession	848.87	78.19	70.57
1980 Recession	905.57	220.77	204.95
1981-1982 Recession	627.13	31.54	23.29
Early 1990's Recession	413.59	69.21	62.12
Early 2000's Recession	475.44	95.71	75.17
Great Recession	336.71	36.00	29.83
COVID-19 Recession	445.87	69.06	46.66

- Calculations and Outcomes. Provided below are the calculations and outputs for this section.



D10 Accuracy Metrics for our forecast

```
# RMSE and MAE to test model accuracy
```

```
# Create array of actual Revenue values, stored in Test variable

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

```
test_array.shape
```

```
(178, 1)
```

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

```
predicted_array.shape
```

```
(178, 1)
```

```
#RMSE Calculation

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

```
RMSE = 36.00079057140054
```

```
# 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 Interval of 20%

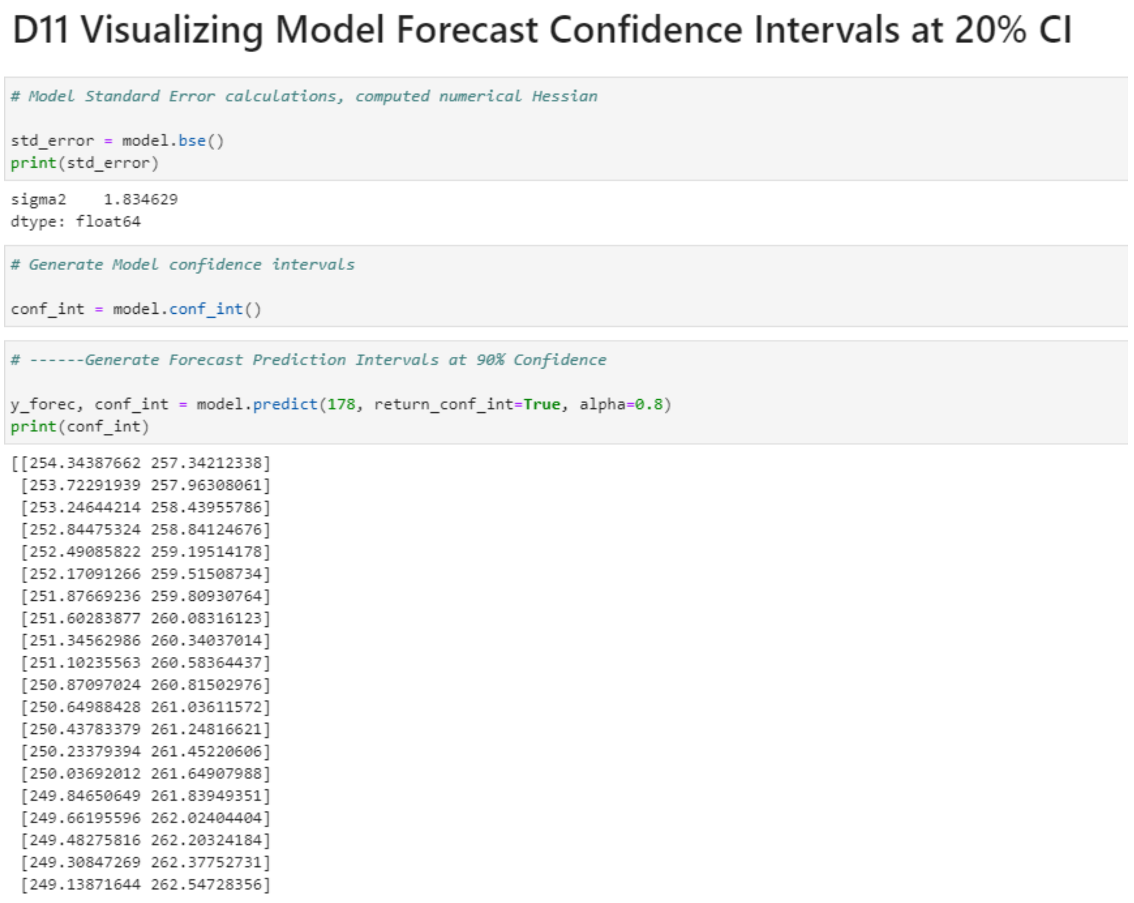
- Technique used.
  - Model.predict().
  - Google Sheets.
  - Matplotlib pyplot plot.
  - Pandas.to_datetime().
  - Df.set_index().
- Justification of technique.
  - This method accurately visualizes the specified confidence interval and all necessary data, creating a visually appealing and informative line graph for analyzing the results of the analysis. Model.predict is utilized to return the confidence interval values at 20% CI for model predictions.

The returned values are assigned to variables for the low and high predictions representing the upper and lower bounds of the confidence interval. The low and high prediction variables are read out to a .csv file, which is loaded into Google Sheets to add column 'Date' to both predictions. 'Date' aligns with the date for the prediction period for each Recession period. The files are saved and read back into Jupyter Notebook.

  - The low and high prediction variables are then formatted as time series. Pandas.to_datetime assigns 'Date' as datetime, df.set_index sets 'Date' to the index for both variables. These variables are then plotted as the upper and lower bounds for the 20% confidence interval alongside the test data, actual values, and model predictions for each Recession period. This creates a line graph that has all of the data and predictive results one one graph.
  - One advantage of visualizing the results in this manner is that it is intuitive for technical and non-technical individuals to assess how well the model performed. Being able to see everything together as a cohesive model allows the viewer to assess the model fit.
  - One disadvantage of this method is that it doesn't return the percentage of predictions that were accurate and inaccurate in predicting the value. This method of visualizing the results relies on the end user accurately interpreting the results of the analysis. For this analysis specifically this wasn't a drawback, as the results are visually obvious for each Recession period.

- Results, Calculations and Outcomes. Provided below are the calculations used for generation of model forecasts and the code necessary for creating the model visualization. Below this are the resulting plotted ARIMA model forecasts compared to the actual time series for each recession period.

Generating Low and High Confidence interval values from model predictions



D11 Visualizing Model Forecast Confidence Intervals at 20% CI

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

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

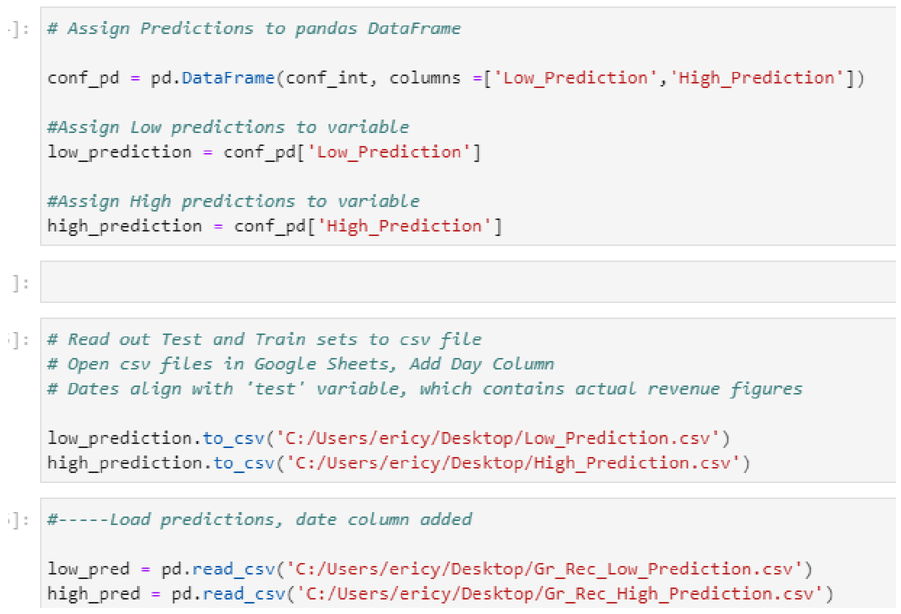
```
sigma2    1.834629
dtype: float64
```

```
# Generate Model confidence intervals

conf_int = model.conf_int()
```

```
# ------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]
```

Low and high prediction values assigned to variables. Read out to be edited in Google Sheets, read back in for further analysis in IDE.

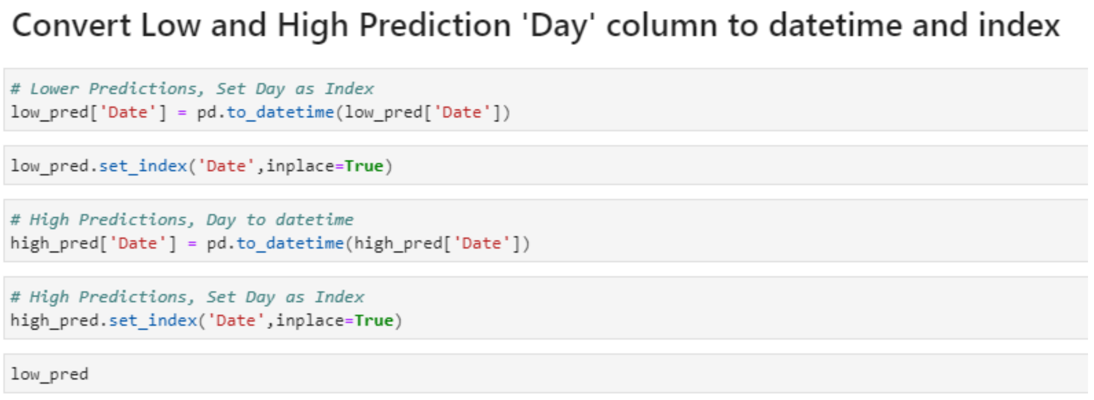


```
# Assign Predictions to pandas DataFrame

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']
```

```
# 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')
```

```
#-----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')
```

Low and High prediction variables converted to datetime, indexed.



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

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

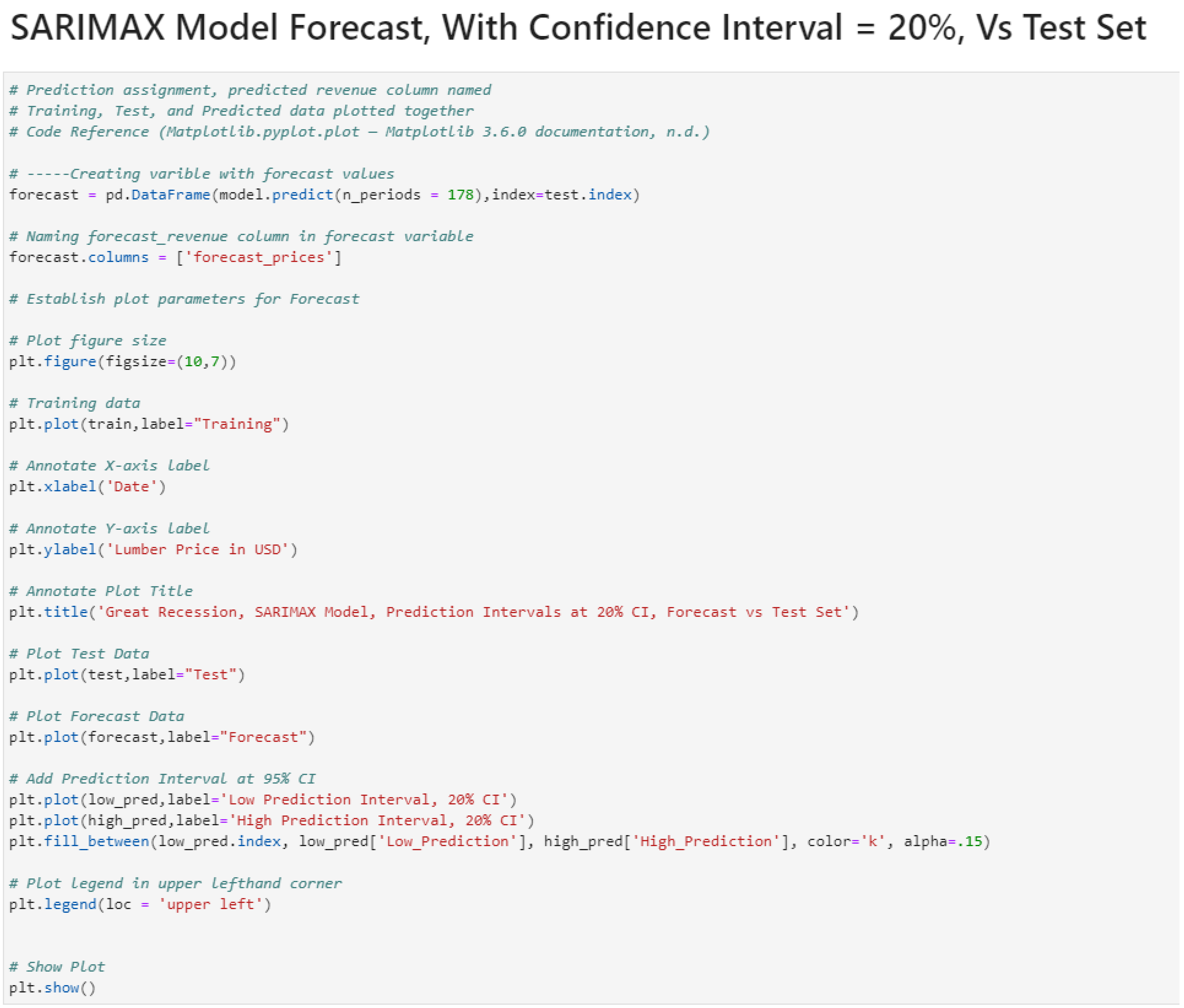
```
low_pred.set_index('Date',inplace=True)
```

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

```
# High Predictions, Set Day as Index
high_pred.set_index('Date',inplace=True)
```

```
low_pred
```

Code for plotting the training data, actual values, model forecast, low predictions, and high predictions.

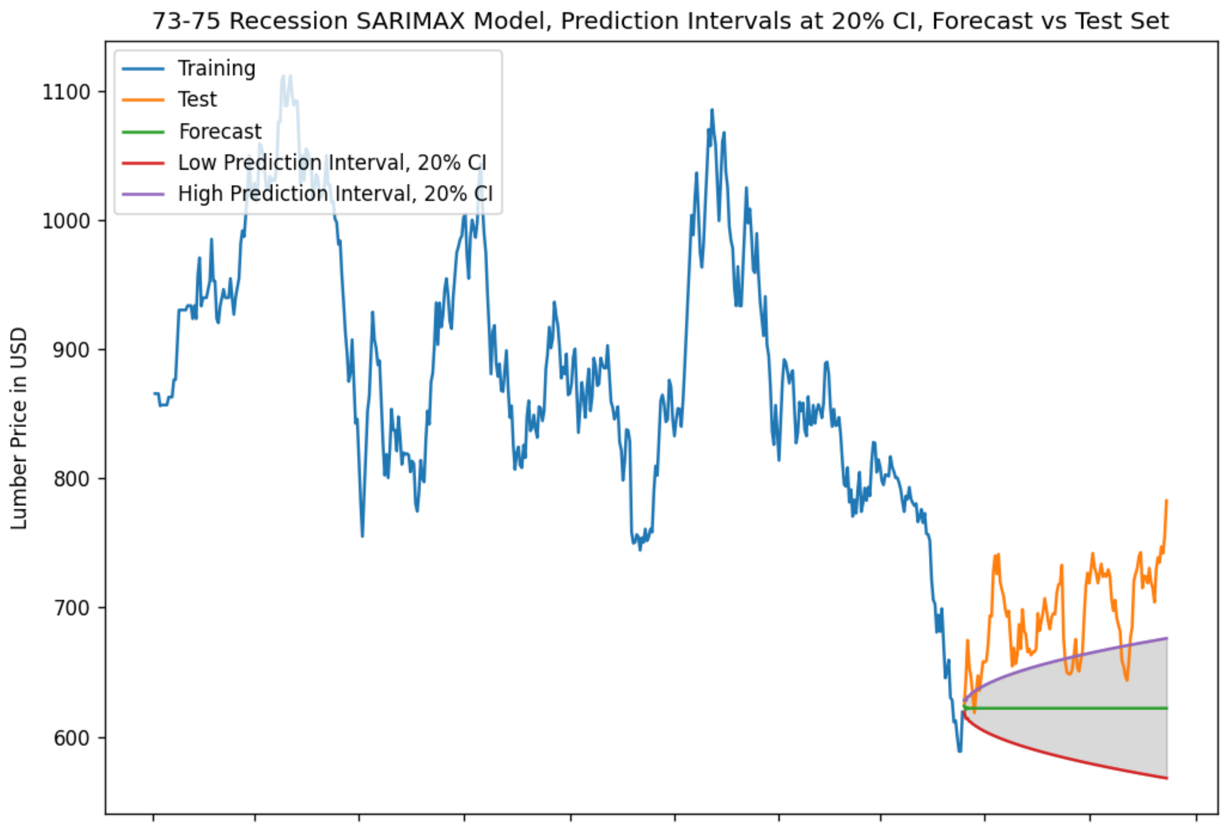


SARIMAX Model Forecast, With Confidence Interval = 20%, Vs Test Set

```
# 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")

# 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()
```

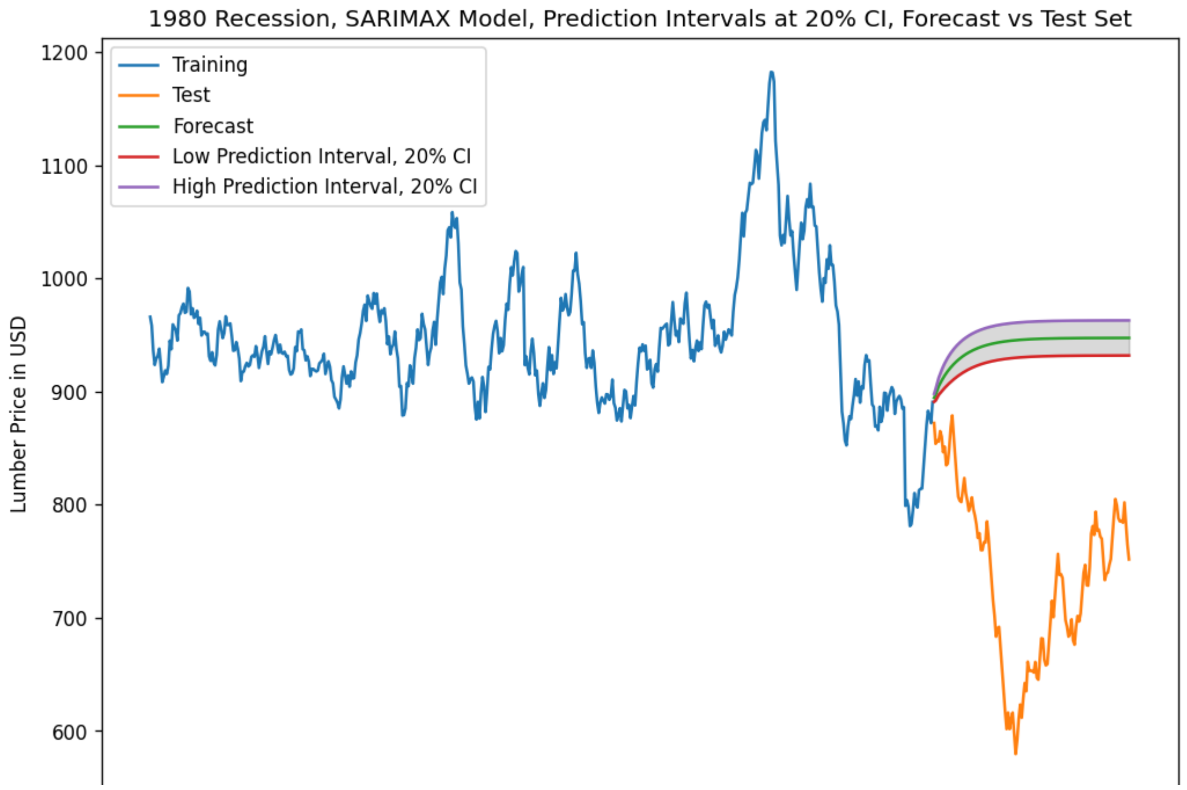
1973-1975 Recession

Training Period: 11/16/1972 - 06/25/1974 . Test & Forecast Period: 06/26/1974 - 3/31/1975.



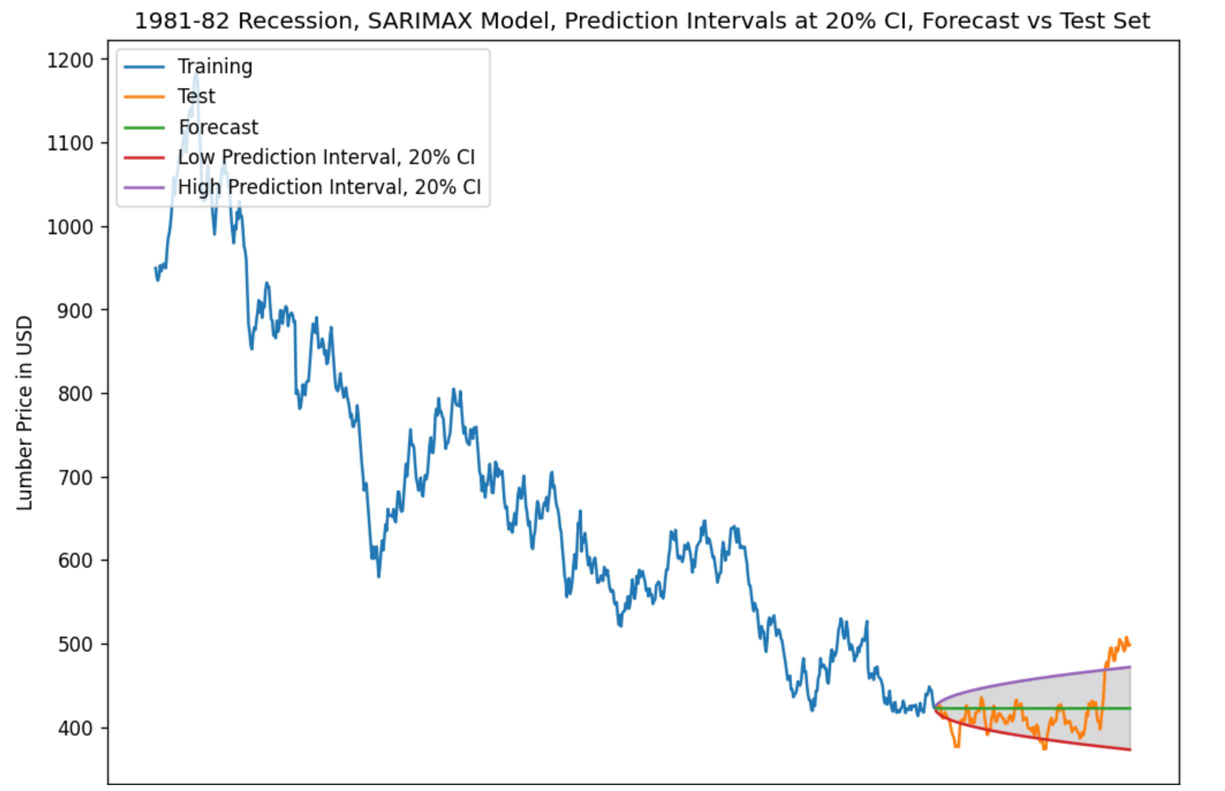
1980 Recession

Training Period: 01/03/1978 - 01/24/1980 . Test & Forecast Period: 01/25/1980 - 7/31/1980.



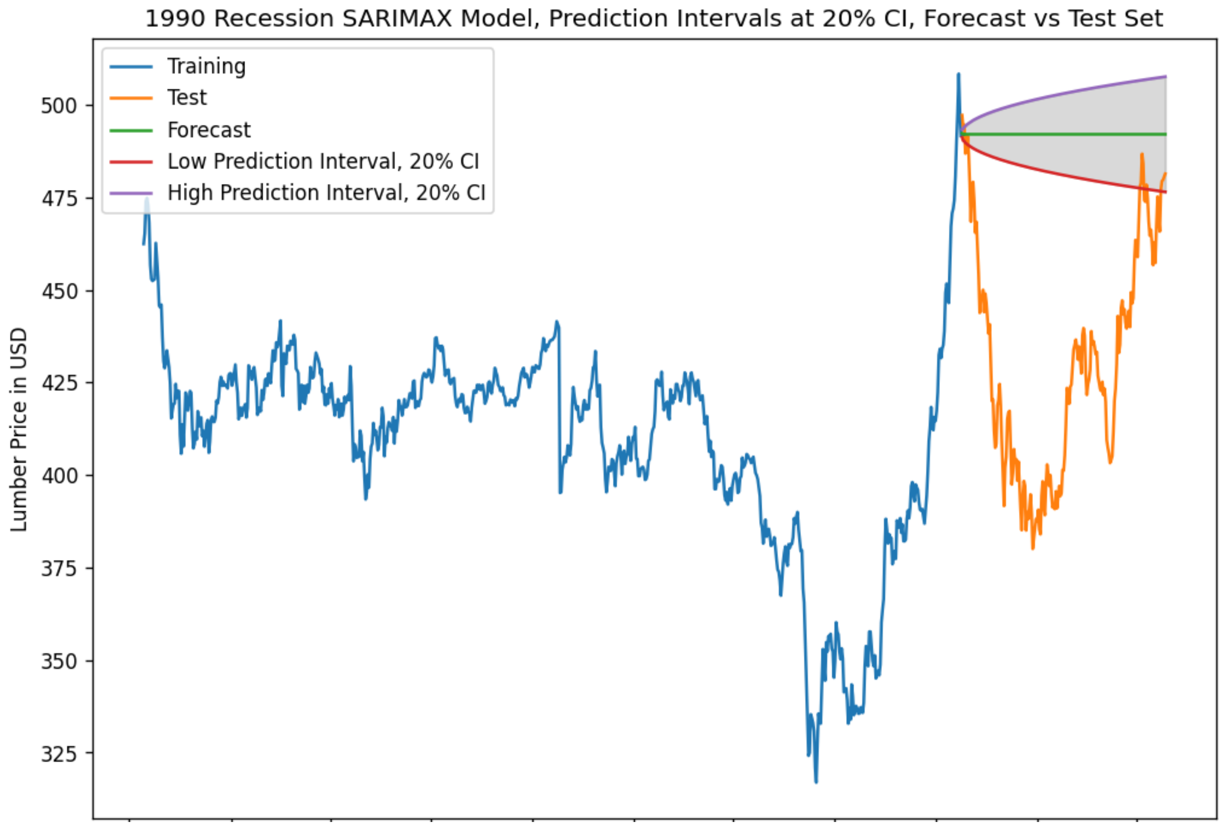
1981-1982 Recession

Training Period: 07/02/1979 - 03/25/1982 . Test & Forecast Period: 03/26/1982 - 11/30/1982.



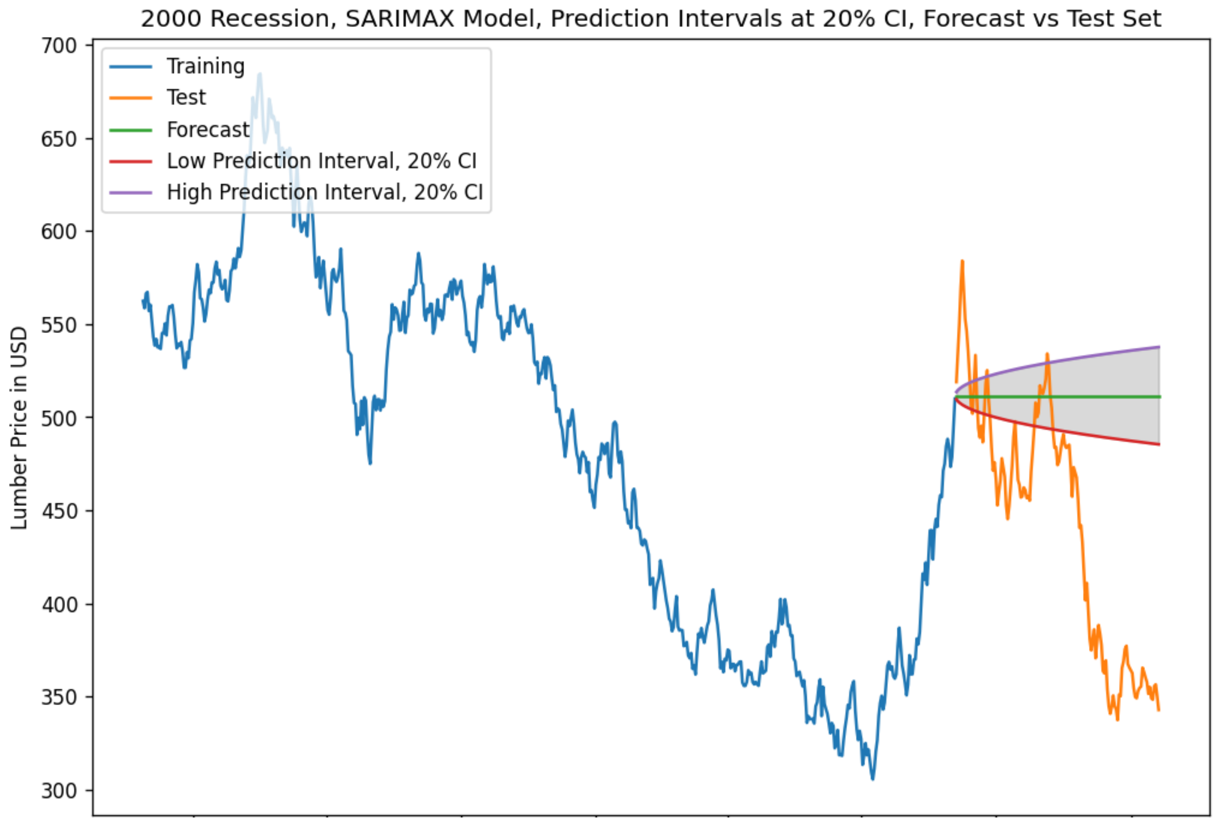
1990 Recession

Training Period: 07/01/1988 - 06/06/1990 . Test & Forecast Period: 06/07/1990 - 03/01/1991.



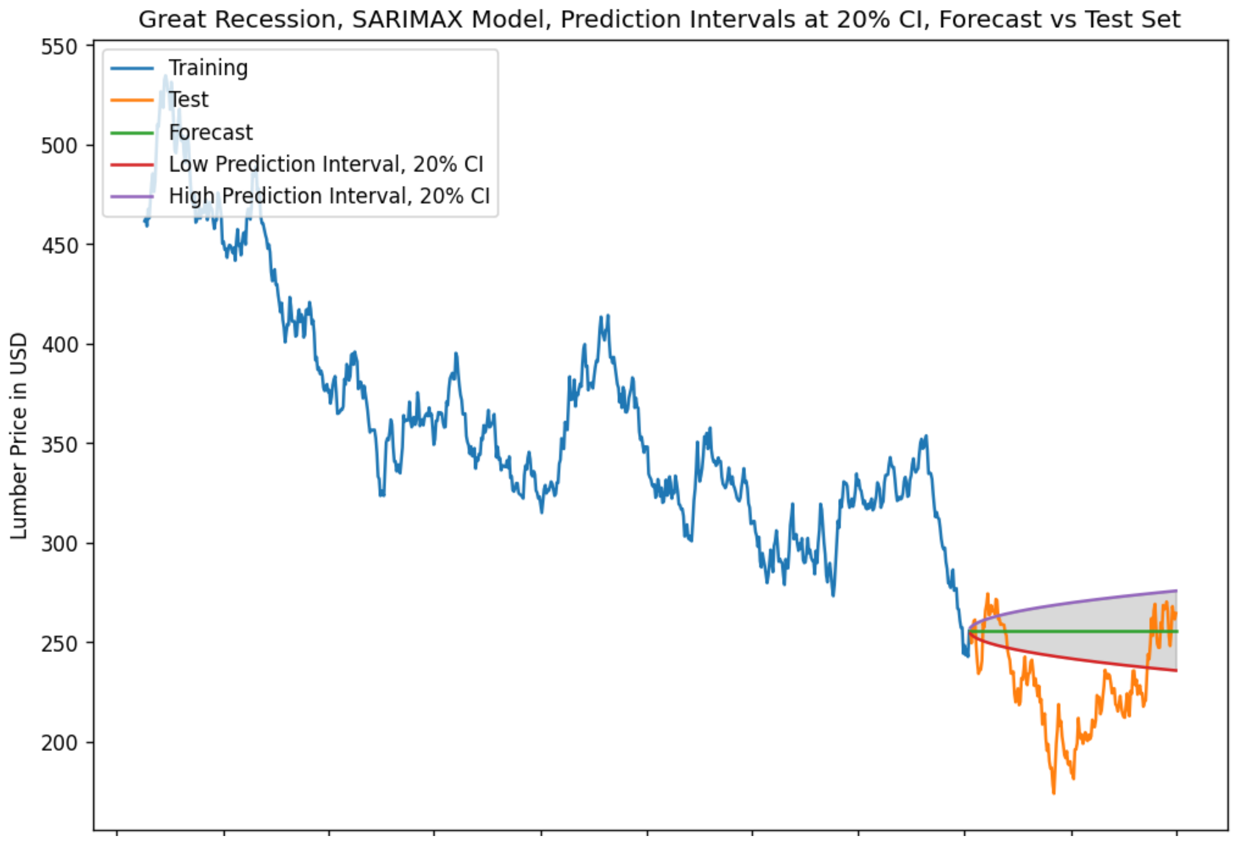
2000 Recession

Training Period: 03/01/1999 - 05/03/2001. Test & Forecast Period: 05/04/2001 - 11/30/2001.



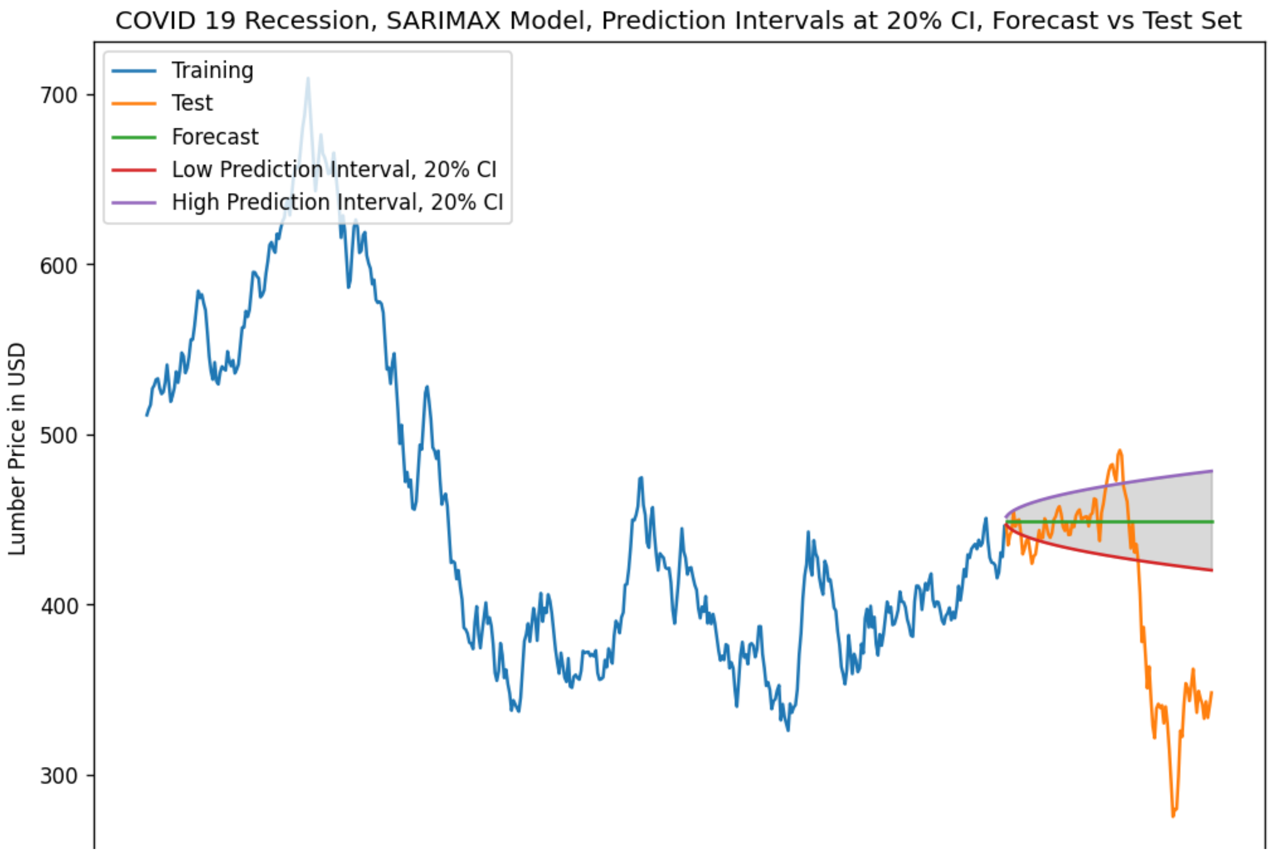
Great Recession

Training Period: 12/2/2005 - 10/09/2008. Test & Forecast Period: 10/10/2008 - 06/30/2009.



COVID-19 Recession

Training Period: 02/01/2018 - 11/14/2019. Test & Forecast Period: 11/15/2019 - 4/30/2020.



E: Data Summary and Implications

To revisit the foundation of this analysis, the question that was set out to answer was: Can an ARIMA model be constructed that accurately forecasts Recessionary US Lumber Prices? The null hypothesis for this analysis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval cannot be made from the historical US Lumber pricing dataset. The alternative hypothesis is: A Time Series model that accurately predicts Recession lumber pricing within a 20% confidence interval can be made from the historical US Lumber pricing dataset.

It can be concluded from this analysis that for every recessionary time period analyzed, an ARIMA model cannot be created that predicts lumber pricing within a 20% confidence interval. The plotted results of model predictions vs. actual values, with a 20% confidence interval, visualizes this. The evaluation metrics used to assess model prediction performance, RMSE and MAE, also indicate the models predicted lumber price poorly for every period analyzed. For each recessionary period the Null Hypothesis is accepted.

Recession Name	**Accept or Reject Null Hypothesis**
1973-1975 Recession	Accept Null Hypothesis
1980 Recession	Accept Null Hypothesis
1981-1982 Recession	Accept Null Hypothesis
Early 1990's Recession	Accept Null Hypothesis
Early 2000's Recession	Accept Null Hypothesis
Great Recession	Accept Null Hypothesis
COVID-19 Recession	Accept Null Hypothesis

A limitation of the analysis was in choosing to only implement ARIMA models for forecasting the time series during recessionary periods. Recessionary periods historically cause commodity price volatility. There are multiple causes for this, including global macroeconomic shocks, demand weakness during recessions, and supply chain disruptions (Vasishtha, 2022). As this analysis progressed, it was seen that this volatility leads to the price swings that are seen in each of the recessionary periods modeled. This affects the stationarity, normality, and residual distribution of the data, ultimately affecting model performance in forecasting future prices.

One future approach for studying lumber prices in the future would be to select a different statistical method for modeling the time series. One recommendation is to utilize an autoregressive conditional heteroskedasticity (ARCH) model for future modeling of Lumber pricing. ARCH models for time series data function by describing the variance of the current error term as a function of the actual size from the previous time period error term. This type of model is commonly used in modeling financial time series when the data exhibits volatility over time (Autoregressive conditional heteroskedasticity, n.d.). The recession periods analyzed all exhibited price volatility over time, and as such would benefit from a statistical technique that addresses volatility more adequately than an ARIMA model.

A second approach for studying lumber prices in the future would be to look at modeling different spans of time than selected for this analysis. After much searching for the optimal length of time a time series should be for forecasting, it seems that there is as much art as there is science in proper length selection. For shorter time series periods a common issue that arises is that, when AIC is selected for model selection, AIC suggests simpler models. This is because anything with more than one, perhaps two, parameters begins to produce poor forecasts due to estimation errors (Hyndman, Athanasopoulos, 2018). Longer time periods for time series analysis can suffer from not being flexible enough to work over the whole period of data, leading to poor results (Hyndman, Athanasopoulos, 2018).

The author recommends against using only ARIMA models for predicting the price of Lumber during recessions or periods of price volatility. This analysis analyzes seven separate recessionary periods, and ARIMA was found to be an inadequate method for accurately forecasting lumber prices for all of them. The results of this analysis would not provide the level of certainty that investors, and stakeholders interested in lumber prices throughout the supply chain, would require to make informed business decisions with. Beginning further analysis with an ARCH model would be a plausible course of action, given the information learned from this analysis.

It is recommended that further analysis utilizing different statistical methods be undertaken. It is also recommended that the price of lumber be assessed over varying periods of time. This allows for more control over time periods in the time series, which brings both the art and the science of time length selection into the control of the analyst. Shortening the time frame for prediction would be a plausible course of action to yield more desirable predictions, given the experience from this analysis.

F1: Sources For Written Analysis

Augmented Dickey-Fuller (ADF) test. (n.d.). REAL-TIME MATH Overview. https://rtmath.net/assets/docs/finmath/html/93a7b7b9-e3c3-4f19-8a57-49c3938d607d.htm

Autoregressive conditional heteroskedasticity. (n.d.). Wikipedia, the free encyclopedia. Retrieved October 19, 2022, from https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity

Brown, J. (2016, September 29). Past Performance Is Not Indicative Of Future Results. Forbes. https://www.forbes.com/sites/johnbrown/2016/09/29/past-performance-is-not-indicative-of-future-results/?sh=51fcf7373bf5

Brownlee, J. (2020, August 14). White Noise Time Series with Python. Machine Learning Mastery. https://machinelearningmastery.com/white-noise-time-series-python/

Calzone, O. (2022, April 7). Mae, mse, rmse, and F1 score in time series forecasting. Medium. https://medium.com/@ottaviocalzone/mae-mse-rmse-and-f1-score-in-time-series-forecasting-d04021ffa7ce

Farmer, S. (2021, August 2). Projecting lumber demand in the U.S. and abroad. Retrieved from https://www.usda.gov/media/blog/2018/09/07/projecting-lumber-demand-us-and-abroad

Hyndman, R., Athanasopoulos, G. (2018). 12.7 Very Long and very short time series. In *Forecasting: principles and practice, 2nd edition*, OTexts: Melbourne, Australia. OTexts.com/fpp2.

Identifying the orders of AR and MA terms in an ARIMA model . (n.d.). people.duke.edu. https://people.duke.edu/~rnau/411arim3.htm

Identifying the order of differencing in ARIMA models. (n.d.). people.duke.edu. https://people.duke.edu/~rnau/411arim2.htm

Industry market research, reports, and statistics. (n.d.). IBISWorld - Industry Market Research, Reports, & Statistics. https://www.ibisworld.com/industry-statistics/market-size/lumber-wholesaling-united-states/

Kolassa, S. (2016, February 1). Sample size for best forecasting ARIMA model. Cross Validated. https://stats.stackexchange.com/questions/193550/sample-size-for-best-forecasting-arima-model

Levich, R., & Rizzo, R. (1998, December). Welcome to Pages at the Stern School of Business, New York University. https://pages.stern.nyu.edu/~rlevich/wp/LR1.pdf

Li, Y. (2019, May 29). "auto.arima" in SAS? Retrieved from https://stackoverflow.com/questions/11620445/auto-arima-in-sas

List of recessions in the United States. (n.d.). Wikipedia, the free encyclopedia. Retrieved October 16, 2022, from https://en.wikipedia.org/wiki/List_of_recessions_in_the_United_States

Lumber prices - 50 year historical chart. (n.d.). https://www.macrotrends.net/2637/lumber-prices-historical-chart-data

Manokhin, V., & CQF. (2021, November 28). Python vs R for time-series forecasting. Retrieved from https://valeman.medium.com/python-vs-r-for-time-series-forecasting-395390432598

Mean absolute error. (n.d.). Wikipedia, the free encyclopedia. Retrieved October 19, 2022, from https://en.wikipedia.org/wiki/Mean_absolute_error

Padhma, M. (2021, October 28). Evaluation metric for regression models. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/

Periodogram. (n.d.). Wikipedia, the free encyclopedia. Retrieved October 18, 2022, from https://en.wikipedia.org/wiki/Periodogram

Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation. (n.d.). https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

Python vs. R: What's the difference? (2021, March 23). IBM - United States. https://www.ibm.com/cloud/blog/python-vs-r

Root-mean-square deviation. (n.d.). Wikipedia, the free encyclopedia. Retrieved October 19, 2022, from https://en.wikipedia.org/wiki/Root-mean-square_deviation

Simplilearn. (2022, July 22). What is the difference between Excel and Google sheets? | Simplilearn. Simplilearn.com. https://www.simplilearn.com/tutorials/excel-tutorial/google-sheets-vs-excel

Statsmodels.tsa.seasonal.seasonal_decompose — statsmodels. (n.d.). https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html

The Akaike information criterion. (n.d.). Time Series Analysis, Regression and Forecasting. https://timeseriesreasoning.com/contents/akaike-information-criterion/

USA historical consumer price index (CPI) - 1913 to 2022. (n.d.). Retrieved from https://www.rateinflation.com/consumer-price-index/usa-historical-cpi/

Vasishtha, G. (2022, January 24). Commodity price cycles: Causes and consequences. World Bank Blogs. https://blogs.worldbank.org/developmenttalk/commodity-price-cycles-causes-and-consequences

F2: Sources For 3rd Party Code

6. Tips to using auto_arima — pmdarima 2.0.1 documentation. (n.d.). Retrieved from https://alkaline-ml.com/pmdarima/tips_and_tricks.html#period

Boston, S. (2020, October 31). ValueError: You must specify a period or X must be a pandas object with a DatetimeIndex with a freq not set to none. Retrieved from https://stackoverflow.com/questions/64617482/valueerror-you-must-specify-a-period-or-x-must-be-a-pandas-object-with-a-dateti

Kosaka, M. (2021, March 3). Efficient time-series using Python's Pmdarima library. Retrieved from https://towardsdatascience.com/efficient-time-series-using-pythons-pmdarima-library-f6825407b7f0

Making time series stationary | Python. (n.d.). Retrieved from https://campus.datacamp.com/courses/arima-models-in-python/chapter-1-arma-models?ex=5

Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation. (n.d.). Retrieved from https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Pmdarima.arima.AutoARIMA — pmdarima 2.0.1 documentation. (n.d.). Retrieved from https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.AutoARIMA.html#pmdarima.arima.AutoARIMA

Smith, T. G. (2019, December 18). Forecasting the stock market with pmdarima. Retrieved from https://alkaline-ml.com/2019-12-18-pmdarima-1-5-2/

Verma, Y. (2021). Complete guide to SARIMAX in Python for time series modeling. Retrieved from https://analyticsindiamag.com/complete-guide-to-sarimax-in-python-for-time-series-modeling/