Saemi Ramirez
011926418

D213 PA 1
Times Series Modeling

10/11/2024

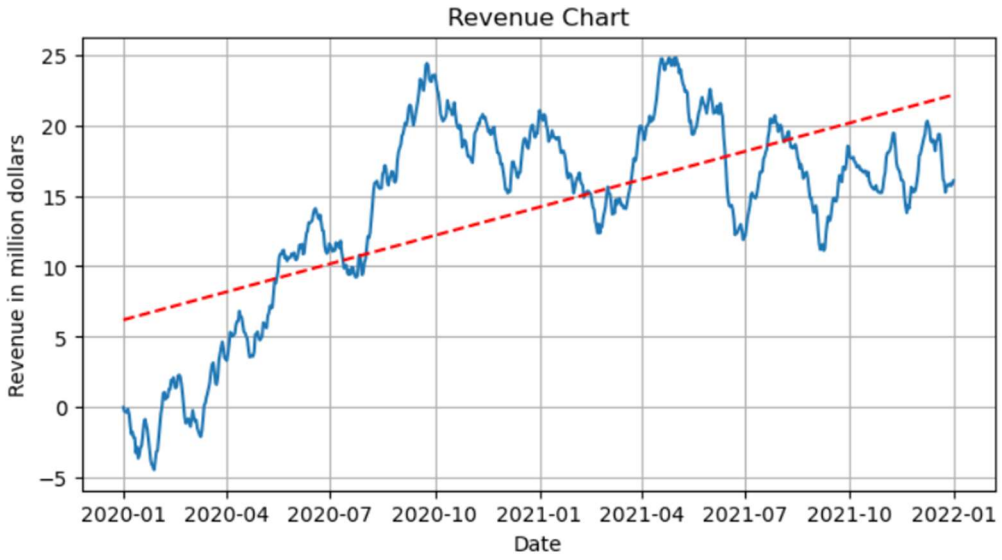
WGU

A. Purpose of this data analysis
    1. Research Question
       What is the WGU medical organization's revenue in the following quarter?
    2. Define objectives or goals
       The objective is to predict the future revenue of WGU medical organization using ARIMA time series modeling. ARIMA time series modeling can capture the patterns in the past revenue data. It also can help the management to make critical decisions about budgeting, resource allocation, and or financial goals by providing the accurate revenue forecasting.

B. Summarize assumptions of a time series modeling including stationarity and autocorrelated data
    1. The assumption for the stationary time series modeling implies that its statistical properties, such as mean, variance, and autocorrelation, remain constant over time. A stationary time series is one where the mean, variance, and autocorrelation structure remain consistent over time. This assumption is crucial because many time series analysis methods, particularly those used for forecasting, rely on the premise that the patterns and relationships in the data are stable over time. (The Stationary)
    2. Autocorrelation measures how related the current data points are to past values, which can have significant implications across various industries. For instance, if our passenger data exhibits strong autocorrelation, we can infer that high passenger numbers today indicate a strong likelihood of similarly high numbers tomorrow. (Pierre)

C. Data cleaning process
    1. Provide the line graph visualizing the realization of the time series
       Following screenshot is the plotting code and the line graph of the WGU medical revenue with the trend line starting January 1, 2020 to December 31, 2021.

```
#Visualize the data
plt.figure(figsize=(8,4))
plt.plot(cleaned_df.Revenue)
plt.title('Revenue Chart')
plt.xlabel('Date')
plt.ylabel('Revenue in million dollars')

#Generate trend line
x = mdates.date2num(cleaned_df.index)
y = cleaned_df.Revenue
z = np.polyfit(x,y,1)
p = np.poly1d(z)

plt.plot(x, p(x), "r--")
plt.grid(True)
plt.show()
```



2. Describe the time step formatting of the realization including any gaps in measurement and the length of the sequence

   The Day column in the provided medical data was indexed from 1 to 731 in index datatype. The new index was given with the column name 'Date' from 2020-01-01 ended on 2021-12-31. The data was for 2-year worth of information. There was no gap in both columns to fill in or to drop.

3. Evaluate the stationarity of the time series

   The given WGU medical dataset was not the stationarity of the time series. After running the adfuller (Augmented Dickey-Fuller) function on the cleaned data, the p-value was close to 0.2, so differentiated function was applied to lower the p-value. Then the p-value lowered to close to 0, which means the data is now stationary.

```
#Checking for stationarity using the Adfuller (Augmented Dickey-Fuller) Test
result = adfuller(cleaned_df['Revenue'])
print("Test statistics: ", result[0])

#p-value indicates the order of the lag or the difference btwn the present & past data
print("p-value: ", result[1])
print("Critical values: ", result[4])
```

```
Test statistics:  -2.2183190476089436
p-value:  0.199664006150644
Critical values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

```
#Check if it is stationary time series or not
if result[1] <= 0.05:
    print("Reject null hypothesis, the time series is stationary")
else:
    print("Fail to reject null hypothesis, the time series is non-stationary")
```

```
Fail to reject null hypothesis, the time series is non-stationary
```

```
#Make the time series stationary
#Differentiate the data here
df_stationary = cleaned_df.diff(periods=1,axis=0).dropna()
#View the data
df_stationary.head()
```

Date	Revenue
2020-01-02	-0.292356
2020-01-03	-0.035416
2020-01-04	-0.012215
2020-01-05	0.215100
2020-01-06	-0.366702

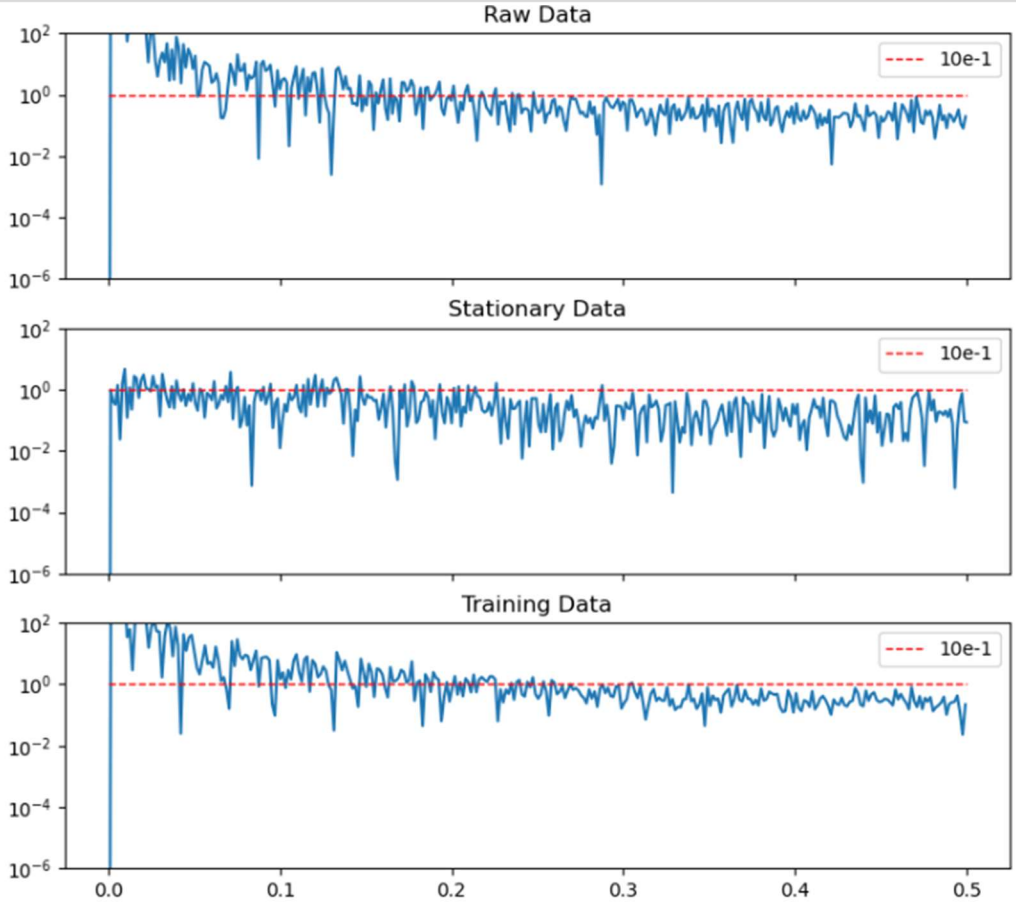
```
#Test for stationarity again
result = adfuller(df_stationary['Revenue'])
print("Test statistics: ", result[0])
print("p-value: ", result[1])
print("Critical values: ", result[4])

if result[1] <= 0.05:
    print("Reject null hypothesis, the time series is stationary")
else:
    print("Fail to reject null hypothesis, the time series is non-stationary")
```

```
Test statistics:  -17.374772303557066
p-value:  5.113206978840171e-30
Critical values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
Reject null hypothesis, the time series is stationary
```

The following visualizations are the Spectral Density graphs on the raw (cleaned_df), the differentiated (df_stationary), and the test set (test).

```python
#Create a definition for varieties of spectral density graphs
def spec_density(data, ax, i: int, title: str) -> None:
    f, Pxx = signal.periodogram(data['Revenue'])
    ax[i].semilogy(f, Pxx)
    ax[i].set_ylim([1e-6, 1e2])
    ax[i].set_title(title)
    ax[i].hlines(y=10e-1, xmin=0, xmax=0.5, lw=1, linestyles='--',
                 color='r', label='10e-1')
    ax[i].legend()
```

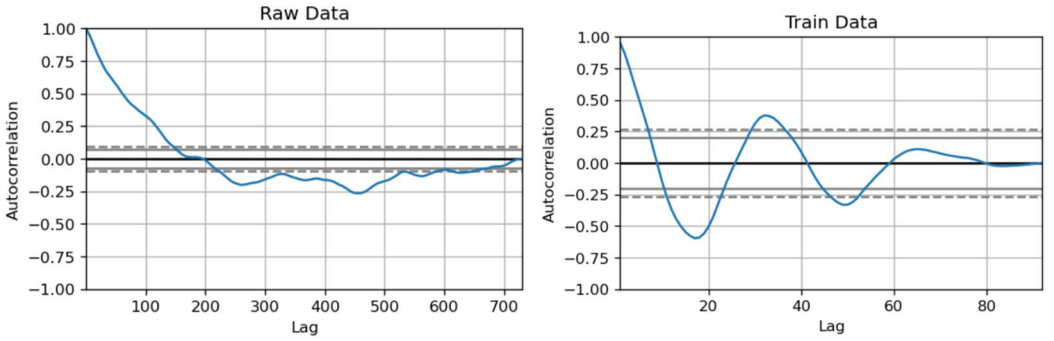
```python
#Displaying Power Spectral density using the stationarity data
fig, ax = plt.subplots(3,1, figsize=(9,8), sharex=True, sharey=True)
spec_density(data=cleaned_df, ax=ax, i=0, title='Raw Data')
spec_density(data=df_stationary, ax=ax, i=1, title='Stationary Data')
spec_density(data=train, ax=ax, i=2, title='Training Data')
```

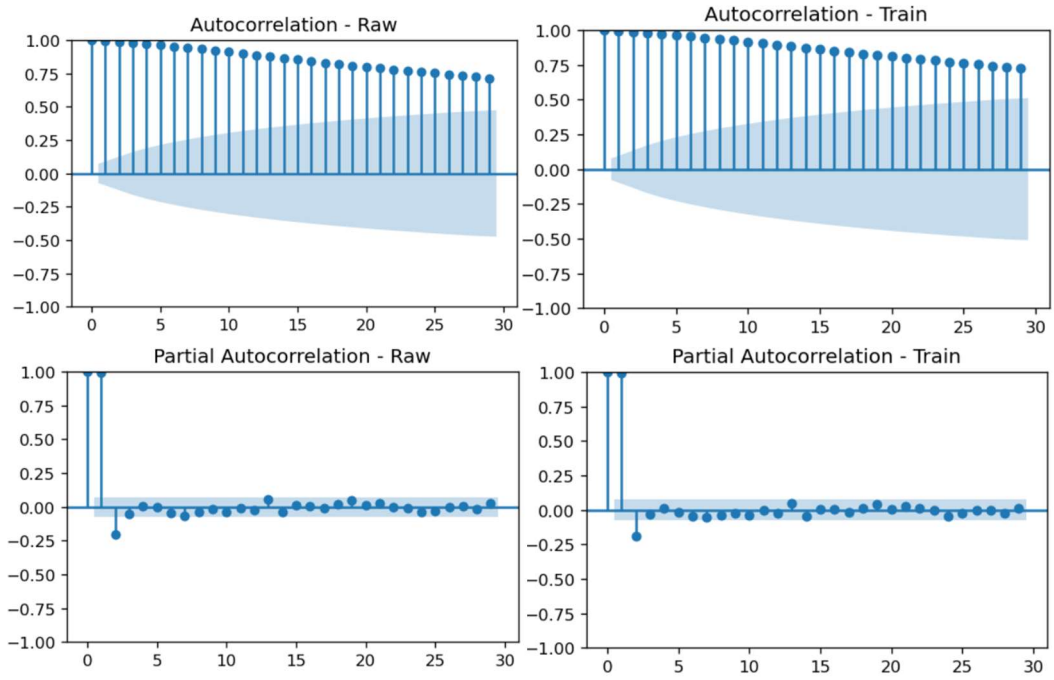


Autocorrelaton and partial autocorrelation visualizations are compared between the raw (cleaned_df, on the left) and the train data (on the right).

```
#ACF & PACF plots on raw (cleaned_df)
plot_acf(cleaned_df)
plot_pacf(cleaned_df)
plt.show()
```

```
#ACF & PACF plots on train set
plot_acf(train)
plot_pacf(train)
plt.show()
```





4. Explain the steps you used to prepare the data for analysis including the training and test set split
    i. Read the WGU medical dataset using read_csv function
    ii. Checked the shape, head, info, and any null values and dropped null values if there was any in the dataset
    iii. Checked the info and description again
    iv. Created the ‘Date’ column and added the date starting January 1, 2020 in yyyy-mm-dd format to the end
    v. Set the ‘Date’ column as index and dropped ‘Day’ column

vi. Ran the adfuller function if the dataset is stationary
vii. The p-value was over 0.05, so ran the differentiated function
viii. Utilized pandas loc function to separate the test and the training set

5. Copy of a cleaned data set
'cleaned_data.csv', 'stationary_data.csv', 'test_data.csv', and 'train_data.csv' are provided

D. Analyze the time series data set

1. Report the annotated finding with visualizations of your data analysis

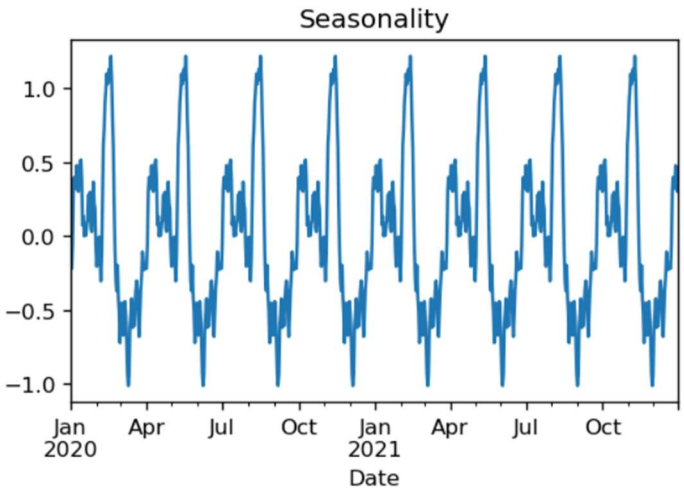
Presence or lack of a seasonal component

The following screenshot is a seasonal graph which shows that it repeats the same behavior approximately every 3-4 months. Raw (cleaned_df), which is non-stationary, was used.

i.

```
#Plot seasonality from the non-stationary data whose value is in the variable decomp
plt.title('Seasonality')
decomp.seasonal.plot()
```

```
<Axes: title={'center': 'Seasonality'}, xlabel='Date'>
```

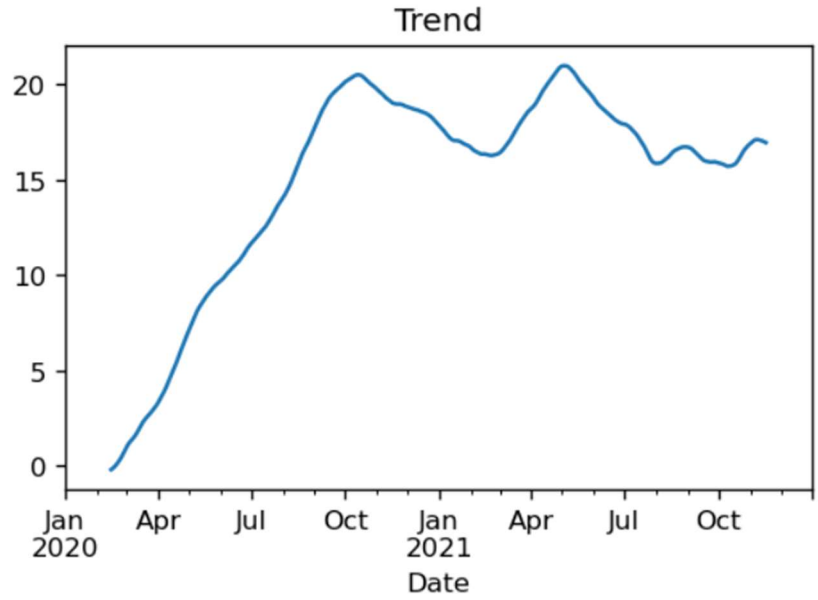


ii. Trends

Trend graphs shows that the revenue was reaching higher until the October of the first year. There are some ups and downs after October 2020. Raw (cleaned_df) was used.

```
plt.title('Trend')
decomp.trend.plot()
```

```
<Axes: title={'center': 'Trend'}, xlabel='Date'>
```

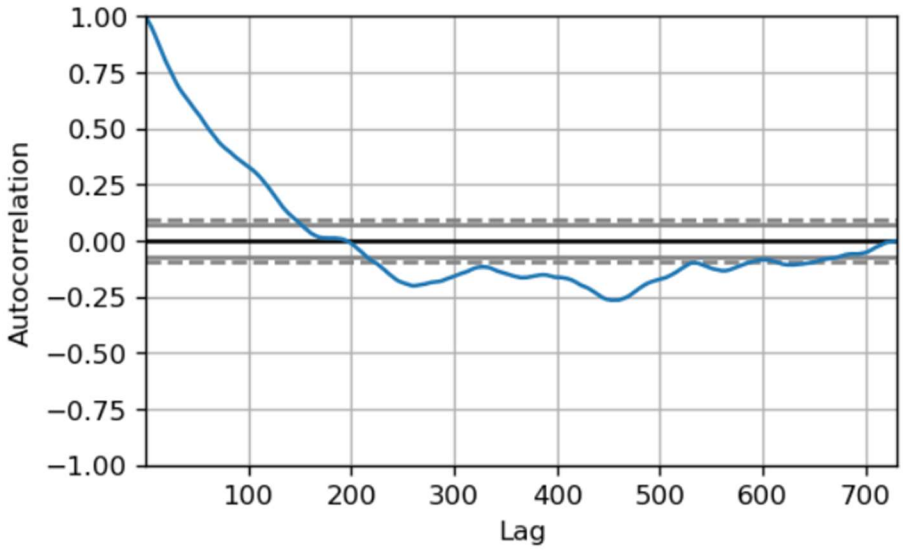


iii. Autocorrelation function

Following screenshot is the autocorrelation graph on raw (cleaned_df) data.

```
#Continue Looking for Seasonality data
#Autocorrelation plot of a non-stationary data
plt.rcParams.update({'figure.figsize':(5,3),
                     'figure.dpi':120})
pd.plotting.autocorrelation_plot(cleaned_df.Revenue.tolist())
```

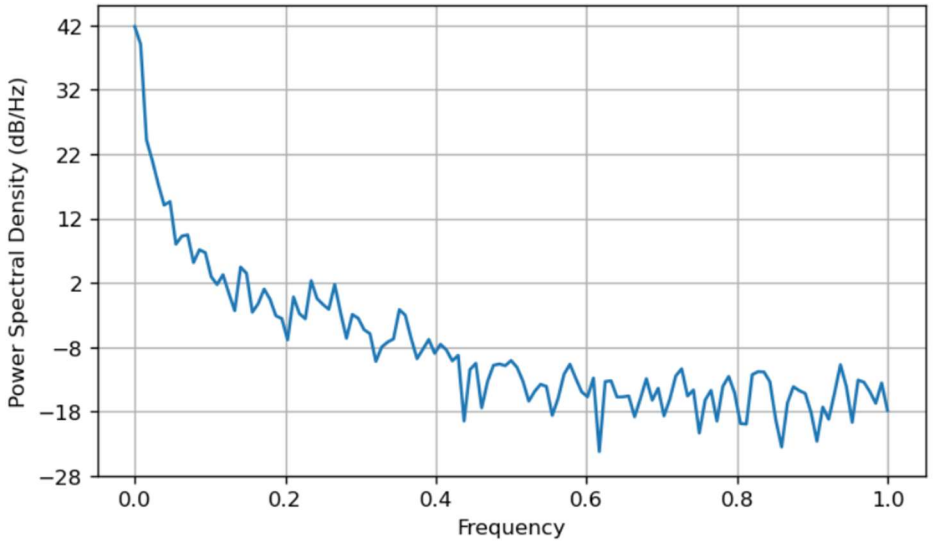
```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```



iv. Spectral density

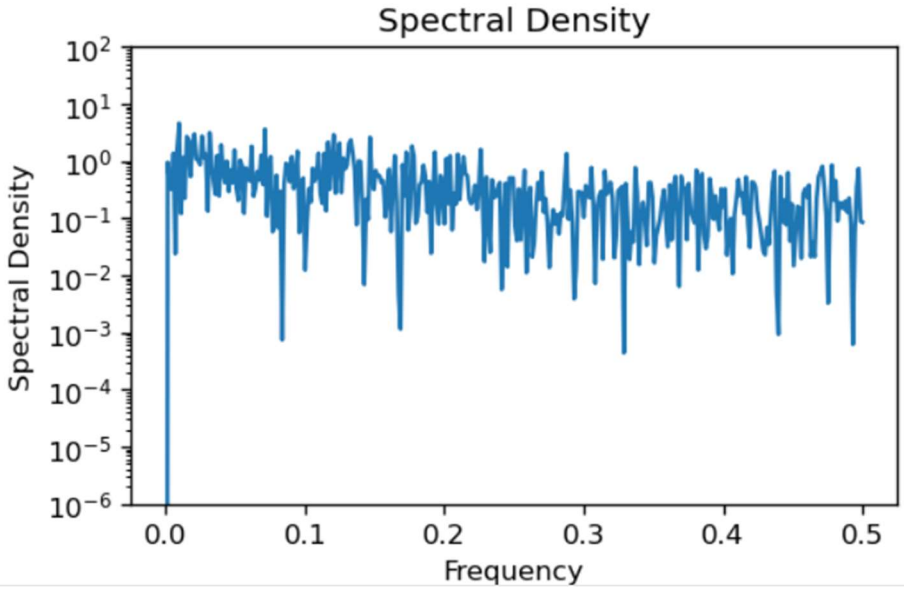
Following screenshot is a plot of power spectral density of the raw (cleaned_df) data.

```
#Example 1: displaying Power Spectral density using non-stationarity revenue data
plt.figure(figsize=(7,4), linewidth=3)
plt.psd(cleaned_df['Revenue'])
```



Following screenshot is a plot of power spectral density of the non-stationary (df_stationary) data.

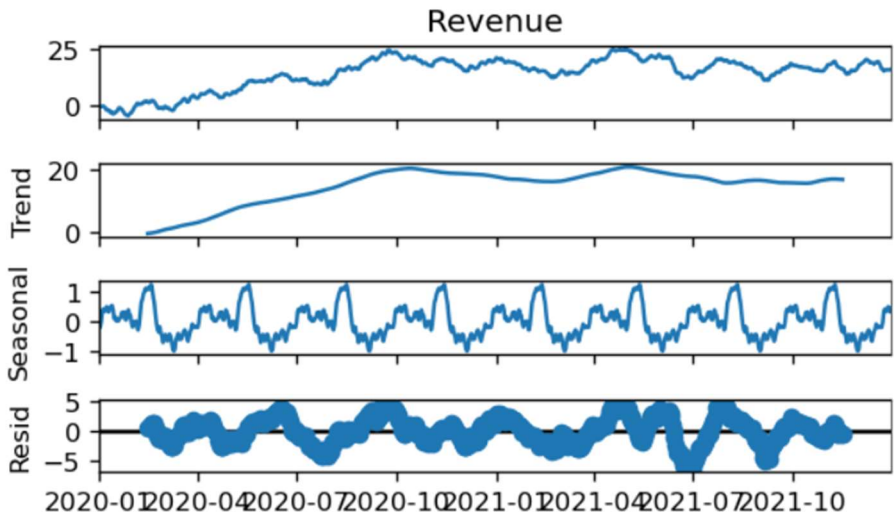
```
#Example 2: displaying Power Spectral density using the stationarity data
f, Pxx_den = signal.periodogram(df_stationary['Revenue'])
plt.semilogy(f, Pxx_den)
plt.ylim([1e-6, 1e2])
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectral Density')
plt.show()
```



v. Decomposed time series

The following screenshot is the full seasonal decomposition graph including the raw data, trendline, seasonal, and the residual of the cleaned_df non-stationary data.

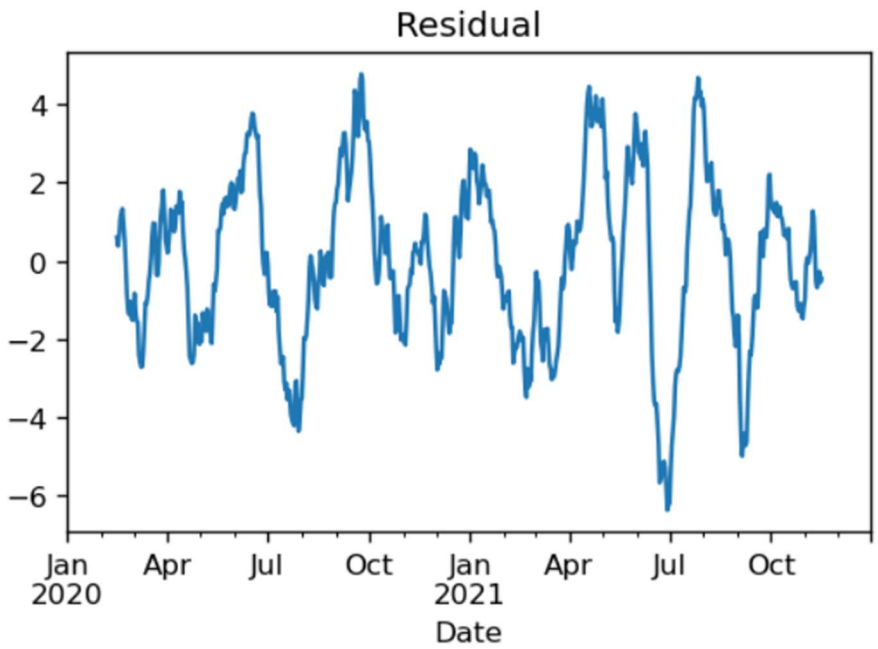
```
#plot decomposition
decomp.plot()
plt.show()
```



vi. Confirmation of the lack of trends in the residuals of the decomposed series Residual graph in the following screenshot of the non-stationary raw (cleaned_df) data

```
plt.title('Residual')
decomp.resid.plot()
```

```
<Axes: title={'center': 'Residual'}, xlabel='Date'>
```



2. Identify ARIMA (autoregressive integrated moving average) model that accounts for the observed trend and seasonality of the time series data

ARIMA function was utilized with train dataset with order of (1, 1, 0) as a best model fit which will be discussed in section E1i. Summary of the ARIMA shows as follows.

```
model = ARIMA(train, order=(1,1,0))
results = model.fit()
results.summary()
```

```
C:\Users\Saemi\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.
py:473: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  self._init_dates(dates, freq)
C:\Users\Saemi\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.
py:473: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  self._init_dates(dates, freq)
C:\Users\Saemi\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.
py:473: ValueWarning: No frequency information was provided, so inferred f
requency D will be used.
  self._init_dates(dates, freq)
```

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	639
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-387.933
Date:	Fri, 11 Oct 2024	**AIC**	779.867
Time:	20:34:55	**BIC**	788.784
Sample:	01-01-2020	**HQIC**	783.328
	- 09-30-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4100	0.037	11.194	0.000	0.338	0.482
sigma2	0.1975	0.012	16.543	0.000	0.174	0.221

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.10
Prob(Q):	0.94	**Prob(JB):**	0.35
Heteroskedasticity (H):	1.03	**Skew:**	-0.02
Prob(H) (two-sided):	0.82	**Kurtosis:**	2.72

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

3. Perform a forecast using the derived ARIMA model
   Added the forecast values for the next 92 days

```
#Make out of Sample Forecast
results.forecast(92)
```

```
2021-10-01    18.382233
2021-10-02    18.364138
2021-10-03    18.356718
2021-10-04    18.353677
2021-10-05    18.352430
2021-10-06    18.351918
2021-10-07    18.351709
2021-10-08    18.351623
2021-10-09    18.351587
2021-10-10    18.351573
2021-10-11    18.351567
2021-10-12    18.351565
2021-10-13    18.351564
2021-10-14    18.351563
2021-10-15    18.351563
2021-10-16    18.351563
```

Created the date list from 2021-10-01 to 2021-12-31

```
#Create future dates after the dataset
index_future_dates = pd.date_range(start='2021-10-01', end='2021-12-31')
print(index_future_dates)
```

```
DatetimeIndex(['2021-10-01', '2021-10-02', '2021-10-03', '2021-10-04',
               '2021-10-05', '2021-10-06', '2021-10-07', '2021-10-08',
               '2021-10-09', '2021-10-10', '2021-10-11', '2021-10-12',
               '2021-10-13', '2021-10-14', '2021-10-15', '2021-10-16',
               '2021-10-17', '2021-10-18', '2021-10-19', '2021-10-20',
               '2021-10-21', '2021-10-22', '2021-10-23', '2021-10-24',
               '2021-10-25', '2021-10-26', '2021-10-27', '2021-10-28',
               '2021-10-29', '2021-10-30', '2021-10-31', '2021-11-01',
```

Assigned the start and end positions with index

```
pred = results.predict(start=len(train), end=len(train)+91, typ='levels')
pred.index = index_future_dates
pred.head()
```

```
2021-10-01    18.382233
2021-10-02    18.364138
2021-10-03    18.356718
2021-10-04    18.353677
2021-10-05    18.352430
Freq: D, Name: predicted_mean, dtype: float64
```

Converted the Series to the DataFrame

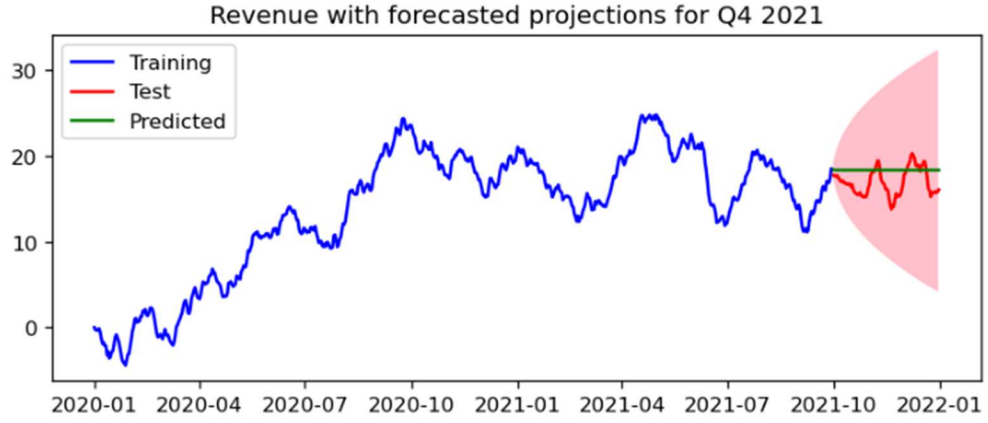
```
prediction = pd.DataFrame(pred, index=test.index)
prediction.columns=['Revenue']
prediction.head()
```

Date	Revenue
2021-10-01	18.382233
2021-10-02	18.364138
2021-10-03	18.356718
2021-10-04	18.353677
2021-10-05	18.352430

Plotted all 3 (train, test, and prediction) in one visualization

```
diff_forecast = results.get_forecast(steps=92)
mean_forecast = diff_forecast.predicted_mean
#Get confidence intervals of predictions
confidence_intervals = diff_forecast.conf_int()
#Select lower and upper confidence limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']

plt.figure(figsize=(8,3))
plt.plot(train,label="Training",color='b')
plt.plot(test,label="Test",color='r')
plt.plot(prediction,label="Predicted",color='g')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pin
plt.title('Revenue with forecasted projections for Q4 2021')
plt.legend(loc = 'upper left')
```

```
<matplotlib.legend.Legend at 0x1faf77d7cd0>
```



4. Provide the output and calculations of the analysis
   The root mean squared error of this forecasting of Q4 2021 is approximately 2.08

```
#Calculate root mean squared error of forecasted data against the observed data
rmse = mean_squared_error(test, prediction, squared=False)
print('The rmse of the prediction is ', rmse)
```

```
The rmse of the prediction is  2.0836449676173965
```

5. Provide the code
   'Saemi Ramirez D213 PA1 – Time Series Modeling.ipynp' is submitted

E. Summarize your findings and assumptions

1. Discuss the results of your data analysis

   i. Selection of an ARIMA model

   In order to find the best model for ARIMA, auto-ARIMA function was conducted using the train data. As it is indicated in the middle of the screenshots, the best ARIMA function model is (1,1,0)

```
#Find the best model using auto-ARIMA
model = auto_arima(train['Revenue'], trace=True, suppress_warnings=True)
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=782.828, Time=0.68 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=895.175, Time=0.12 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=780.955, Time=0.14 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=801.272, Time=0.13 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=895.419, Time=0.05 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=782.951, Time=0.16 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=782.952, Time=0.18 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=782.178, Time=0.54 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=779.867, Time=0.07 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=781.858, Time=0.11 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=781.860, Time=0.15 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=800.680, Time=0.06 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=781.053, Time=0.29 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 2.682 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  639
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -387.933
Date:                Fri, 11 Oct 2024   AIC                            779.867
Time:                        20:34:55   BIC                            788.784
Sample:                  01-01-2020   HQIC                           783.328
                       - 09-30-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4100      0.037     11.194      0.000       0.338       0.482
sigma2         0.1975      0.012     16.543      0.000       0.174       0.221
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 2.10
Prob(Q):                              0.94   Prob(JB):                         0.35
Heteroskedasticity (H):               1.03   Skew:                            -0.02
Prob(H) (two-sided):                  0.82   Kurtosis:                         2.72
===================================================================================

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

   ii. Prediction interval of the forecast and justification

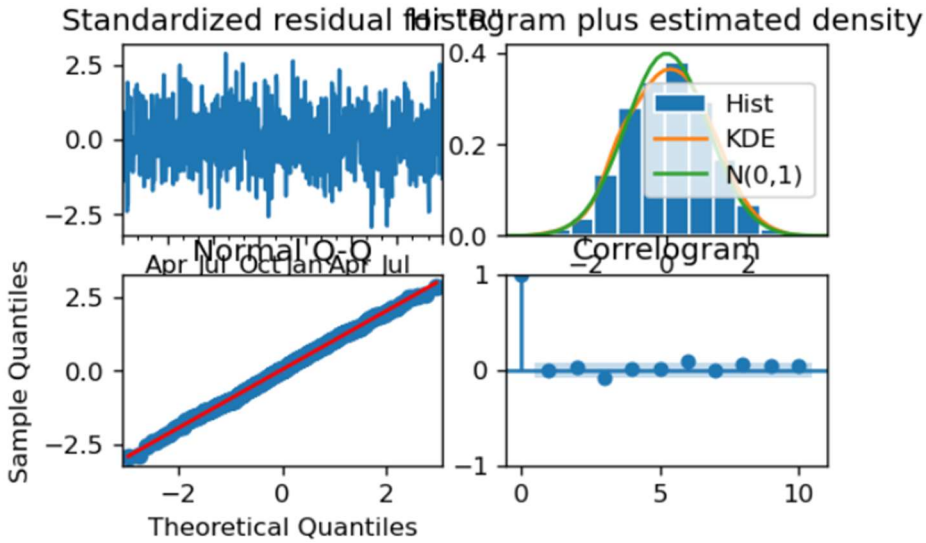
   The prediction interval of the forecast was set to 90 days using the forecast() method because it is a balance between meaningful foresight and reasonable prediction accuracy.

iii. Model evaluation procedure and error metric

```
#Create the 4 diagnostic plots
results.plot_diagnostics().show()
```

```
C:\Users\Saemi\AppData\Local\Temp\ipykernel_8272\792183438.py:3: UserWarni
ng: FigureCanvasAgg is non-interactive, and thus cannot be shown
  results.plot_diagnostics().show()
```

```
<Figure size 3600x3600 with 0 Axes>
```

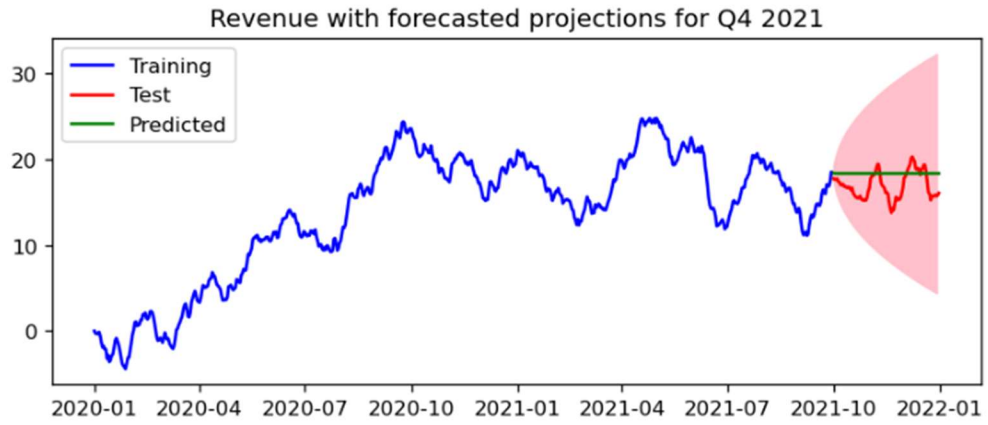


2. Provide an annotated visualization of the forecast of the final model compared to the test set

   The final forecast model compared to the test set is provided in the following screenshots.

```
diff_forecast = results.get_forecast(steps=92)
mean_forecast = diff_forecast.predicted_mean
#Get confidence intervals of predictions
confidence_intervals = diff_forecast.conf_int()
#Select lower and upper confidence limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']

plt.figure(figsize=(8,3))
plt.plot(train,label="Training",color='b')
plt.plot(test,label="Test",color='r')
plt.plot(prediction,label="Predicted",color='g')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pin
plt.title('Revenue with forecasted projections for Q4 2021')
plt.legend(loc = 'upper left')
```

```
<matplotlib.legend.Legend at 0x1faf77d7cd0>
```



3. Recommend a course of action
   With the given forecast of next 90 days, the organization can set the short-term financial and operational goals such as improving cash flow, managing operational costs, or optimizing resource allocation. They can make an adjustment to the quarterly budget as necessary with increase or decrease staffing levels or launching initiatives based on expected revenue.

F. Create your report using an industry-relevant interactive development environment
   'Saemi Ramirez (011926418) D213 PA 1 Time Series Modeling.pdf' is submitted

G. List 3rd party code used
   N/A

H. List 3rd party citations and references
   1. Pierre, Sandrach. *A Guide to Time Series Analysis in Python.* Builtin. (May 10, 2024). https://builtin.com/data-science/time-series-python.
   2. *The Stationary Data Assumption in Time Series Analysis*. Complete Dissertation. https://www.statisticssolutions.com/stationary-data-assumption-in-time-series-analysis.