Time Series Analysis of Telecom Data

Mike Mattinson

Western Governors University

Advanced Data Analytics – D213

Task 1: Time Series Analysis

Dr. Festus Elleh

July 22, 2022

Revision 3

Abstract

Times series data for fictious Telecom company will be analyzed and modeled using ARIMA methods to forecast the next time steps in the series.

Keywords: Time Series Analysis. ARIMA. Forecast. Prediction.

Part I: Research Question

A. Describe the purpose of this data analysis by doing the following:

A1. Summarize one research question that is relevant to a real-world organizational situation captured in the selected data set and that you will answer using time series modeling techniques.

1. What is the company's revenue forecast for the upcoming quarter?

A2. Define the objectives or goals of the data analysis. Ensure that your objectives or goals are reasonable within the scope of the scenario and are represented in the available data.

Use ARIMA time series analysis to model and predict the next 90 values in the given time series. Calculate and report the accuracy of the model.

Part II: Method Justification

B. Summarize the assumptions of a time series model including stationarity and autocorrelated data.

Two (2) assumptions related to forecasting time series data include data stationarity and autocorrelation. Stationarity is a key part of time series analysis. Simply put, stationarity means that the way time series data changes is constant. A stationary time series will not have any trends or seasonal patterns. You should check for stationarity because it not only makes modeling time series easier, but it is an underlying assumption in many time series methods. Specifically, stationarity is assumed for a wide variety of time series forecasting methods including autoregressive moving average (ARMA), ARIMA and Seasonal ARIMA (SARIMA). (Pierre, 2021)

Checking for **autocorrelation** in time series data is another important part of the analytic process. This is a measure of how correlated time series data is at a given point in time with past values, which has huge implications across many industries. For example, if our passenger data has strong autocorrelation, we can assume that high passenger numbers today suggest a strong likelihood that they will be high tomorrow as well. (Pierre, 2021)

Part III: Data Preparation

C. Summarize the data cleaning process by doing the following:

C1. Provide a line graph visualizing the realization of the time series.

Figure 1 shows the visualization of the time series data created using Python matplotlib. The revenue data for the company spans 730 days (2 years) starting from datatime(2020,1,1). The following code was used to load and visualize the data. The plot can be visually useful to determine if the series is stationary or trending.

```
# visualize raw revenue data
x = pd.Series(df.index.values) # if using date
x2 = pd.Series(range(df.shape[0])) # if using date index
fig, ax = plt.subplots(2,1, sharex=True, sharey=True)
ax[0].plot(x, df.Revenue, 'r-', label='Revenue')
ax[1].plot(x, df.Revenue, 'r-', label='Revenue')
ax[0].plot(x,f(x2),"b", label='Poly fit (deg=' + str(n_deg) + ')')
ax[0].legend()
ax[0].set_title('Revenue ($M)')
ax[1].plot(x,df['rolling_mean'], "b-.",
         label=str(n_days) + '-d Roll Mean')
ax[1].plot(x,df['rolling_std'], "g",
         label=str(n_days) + '-d Roll Std')
ax[1].set_title('Revenue ($M) - 30-d Rolling Mean')
ax[1].legend()
import matplotlib.dates as mdates
ax[1].xaxis.set_major_locator(mdates.YearLocator())
ax[1].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[1].xaxis.set_minor_locator(mdates.MonthLocator())
ax[1].xaxis.set_minor_formatter(mdates.DateFormatter('\n%b'))
fig.supxlabel('Date') # common x label
fig.supylabel('Revenue ($M)') # common y label
#plt.gcf().text(0, -.1, "${}$".format(eq_latex), fontsize=14)
title = 'Revenue ($M)'
save_fig(title)
```

C2. Describe the time step formatting of the realization, including any gaps in measurement and the length of the sequence.

The raw data is a .CSV file with two (2) columns, ‘Day’ and ‘Revenue’. Day is an integer ranging from 1 to 731 with no gaps. Revenue is a float. Table 1 show the data and the Table 2 shows the description of the numerical data.

The Day column will be used as the initial dataframe index. Once the data is read into Python, a new index will be created called ‘Date’ which will be based on a specific date value of Jan 1, 2020.

The first day (Day=1) has Revenue zero, which will cause errors if using the log function. There is no meaningful reason to keep the initial zero value, so it will be dropped.

C3. Evaluate the stationarity of the time series.

Autocorrelation (ACF) and partial autocorrelation (PACF) plots are shown in Figure 2. The Dickey-Fuller test for raw data (Table 3) and for differenced data (Table 4). Figure 4 shows the additive decomposition plot. The auto-ARIMA was also used to confirm these observations (Table 5). The following steps are used to determine stationarity.

1. Visually look at raw data (Figure 1) for trends and seasonality. The plot appears to have an increase to the right. The poly-fit regression line included with the raw data is increasing to the right.
2. Use Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots (Figure 3) to look for steady decrease in ACF and sudden drop in PACF. The ACF has a steady decrease and the PACF plot show a sudden drop after lag = 3, indicators that the differenced data is now stationary and ready to be modeled.

3. Use Dickey-Fuller test, the null hypothesis (H0) is that the data is stationary, the alternative hypothesis (H1) is that it is not. If the P value is less than 0.05, then fail to reject null hypothesis and conclude support that data is stationary.
4. Use decomposition analysis to look for periodic trends.
5. As the original data is non-stationary, difference the data and use Dickey-Fuller to re-test. The results of the re-test are shown in Table 4 showing a p-value of 0.00. The differenced data is stationary and ready to be modeled.
6. Auto-ARIMA results (Table 5) found the best model as (1,1,0). The data becomes stationary at the first-differnce.

C4. Explain the steps used to prepare the data for analysis, including the training and test set split.

The following steps were used to prepare the data for the time series analysis:

1. Read in data from CSV file. Use customer function to read in time data and convert the time series data to datetime format.
2. Determine the appropriate index.
3. Mitigate missing data.
4. Create additional fields of 30-d rolling averages.
5. Run tests to determine if data is stationary.
    a. If data is not stationary, then difference the data and repeat stationary tests until the data is stationary.
6. Use pandas iloc as a simple means to split time series data keeping the testing data as the last 30 time values, and the training data everything up to that point.

C5. Provide a copy of the cleaned dataset.

The cleaned data is exported as .CSV file and attached to submission. In addition, the stationary data is also exported as .CSV file using similar code and included with submission.

```
In [204]: # export cleaned data to file
          df.to_csv('tables\cleaned.csv', index=True, header=True)
```

Part IV: Model Identification and Analysis

D. Analyze the time series dataset by doing the following:

D1. Report the annotated findings with visualizations of your data analysis, including the following elements:

Seasonal Component. There is no seasonality trend in the data using the decomposition analysis (Figure 4) and spectral density (Figure 2).

Trends. There are no trends in the data according to the decomposition analysis (Figure 4).

Autocorrelation. Autocorrelation and partial autocorrelation (Figure 3) was generated for raw and differenced data. The raw data is not stationary. The first-differenced data is stationary.

Spectral Density. The spectral density analysis shows no indication of seasonal trend (Figure 2).

Decomposed Time Series. The decomposition analysis showed a light positive trend with raw data and no seasonal trend (Figure 4).

Residuals. The decomposition showed no residuals (Figure 4).

D2. Identify an autoregressive integrated moving average (ARIMA) model that considers the observed trend and seasonality of the time series data.

The best model was found using the auto-ARIMA function as ARIMA(1,1,0)(0,0,0)[0] (Table 5),

D3. Perform a forecast using the derived ARIMA model.

The following is a forecast using the final model for a date outside of the sample data:

```
In [46]:  # make forecast outside of sample
          results.forecast(30)

Out[46]:  2021-12-31    16.421560
          2022-01-01    16.514746
          2022-01-02    16.471162
          2022-01-03    16.491547
          2022-01-04    16.482012
          2022-01-05    16.486472
          2022-01-06    16.484386
          2022-01-07    16.485362
          2022-01-08    16.484905
          2022-01-09    16.485119
          2022-01-10    16.485019
          2022-01-11    16.485066
          2022-01-12    16.485044
          2022-01-13    16.485054
          2022-01-14    16.485049
          2022-01-15    16.485051
          2022-01-16    16.485050
          2022-01-17    16.485051
          2022-01-18    16.485051
          2022-01-19    16.485051
          2022-01-20    16.485051
          2022-01-21    16.485051
          2022-01-22    16.485051
          2022-01-23    16.485051
          2022-01-24    16.485051
          2022-01-25    16.485051
          2022-01-26    16.485051
          2022-01-27    16.485051
          2022-01-28    16.485051
          2022-01-29    16.485051
          Freq: D, Name: predicted_mean, dtype: float64
```

D4. Provide the output and calculations of the analysis you performed.

All output included in the attached Jupyter notebook.

D5. Provide the code used to support the implementation of the time series model.

All of the code is included in the attached Jupyter notebook. In addition, the Python code is included in Appendix A.

Part V: Data Summary and Implications

E. Summarize your findings and assumptions, including the following points:

E1. Discuss the results of your data analysis, including the following:

- the selection of an ARIMA model. The final model was selected based on the results of the auto-ARIMA in conjunction with the individual decomposition and auto-correlation analysis. The final model used was based on the first-differenced data.
- the prediction interval of the forecast. The final model can be used within the 720 days of the sample using the .predict method or outside of the sample using the .forecast method. Over time, the model forecast accuracy will decrease, but should be relatively effective within the first 180 days.

The final model summary is shown in Table 6.

E2. Provide an annotated visualization of the forecast of the final model compared to the test set.

Figure 6 shows the final visualization of the forecast using the final model compared to the test set.

E3. Recommend a course of action based on your results.

The out-of-sample forecast data for Jan 2022 indicates a revenue value of approximately $16.48M, with a slight downward trend. Recommend configuring company operations during the 2Q/FY23 for baseline revenue of $16.48M.

Part VI: Reporting

F. Create your report from part E using an industry-relevant interactive development environment (e.g., a Jupyter Notebook). Include a PDF or HTML document of your executed notebook presentation.

All of the Python code was executed using a local Jupyter server. A .PDF copy of the Jupyter notebook is attached to the submission. A Python .PY file for the associated notebook was created using the Jupyter notebook "Download as Python (PY)" feature and is included in this document in Appendix A.

G. List the web sources used to acquire data or segments of third-party code to support the application.

See References.

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

See References.

I. Demonstrate professional communication in the content and presentation of your submission.

References

Brownlee, J. (2016, December 29). *How to Check if Time Series Data is Stationary with Python*. Retrieved June 13, 2022, from How to Check if Time Series Data is Stationary with Python: https://machinelearningmastery.com/time-series-data-stationary-python/

Brownlee, J. (2016, July 20). *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. Retrieved June 5, 2022, from Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Brownlee, J. (2017, April 27). *Dropout with LSTM Networks for Time Series Forecasting*. Retrieved June 5, 2022, from Dropout with LSTM Networks for Time Series Forecasting: https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/

Brownlee, J. (2017, January 29). *How to Decompose Time Series Data into Trend and Seasonality*. Retrieved June 10, 2022, from How to Decompose Time Series Data into Trend and Seasonality: https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/

hackdeploy. (2018, November 20). *Augmented Dickey-Fuller Test in Python*. Retrieved June 16, 2022, from Augmented Dickey-Fuller Test in Python: https://www.hackdeploy.com/augmented-dickey-fuller-test-in-python/

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practive.* Retrieved June 27, 2022, from https://otexts.com/fpp2/index.html

Khodadadi, A. (2021). *Keras documentation: Traffic forecasting using graph neural networks and LSTM*. Retrieved June 5, 2022, from Keras documentation: Traffic forecasting using graph neural networks and LSTM: https://keras.io/examples/timeseries/timeseries_traffic_forecasting/

Kumar, J. (2021, December 16). *How to Check if Time Series Data is Stationary with Python?* Retrieved June 13, 2022, from How to Check if Time Series Data is Stationary with Python?: https://www.geeksforgeeks.org/how-to-check-if-time-series-data-is-stationary-with-python/

Palachy, S. (2019, November 12). *Detecting stationarity in time series data*. Retrieved June 13, 2022, from Detecting stationarity in time series data: https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e638

Pierre, S. (2021, July 16). A Guide to TIme Series Analysis in Python | Built In. *A Guide to TIme Series Analysis in Python | Built In*. Retrieved May 28, 2022, from https://builtin.com/data-science/time-series-python

Prabhakaran, S. (2019, November 2). *Augmented Dickey-Fuller (ADF) Test - Must Read Guide - ML+*. Retrieved June 19, 2022, from Augmented Dickey-Fuller (ADF) Test - Must Read Guide - ML+: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/

Prabhakaran, S. (2019, February). Time Series Analysis in Python - A Comprehensive Guide with Examples - ML+. *Time Series Analysis in Python - A Comprehensive Guide with Examples - ML+*. Retrieved May 29, 2022, from https://www.machinelearningplus.com/time-series/time-series-analysis-python/

Rajbhoj, A. (2019, October 4). *ARIMA simplified.* Retrieved June 9, 2022, from ARIMA simplified.: https://towardsdatascience.com/arima-simplified-b63315f27cbc

Verma, Y. (2021, August 18). *Complete Guide To Dickey-Fuller Test In Time-Series Analysis*. Retrieved June 16, 2022, from Complete Guide To Dickey-Fuller Test In Time-Series Analysis: https://analyticsindiamag.com/complete-guide-to-dickey-fuller-test-in-time-series-analysis/

Tables

Table 1

Raw data

```
In [6]:  # read time series data from CSV file
         from datetime import datetime
         df =  read_time_series(
             file='data/teleco_time_series.csv',
             index='Day', freq='d',
             start_date=datetime(2020,1,1)
         )
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 730 entries, 2020-01-01 to 2021-12-30
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Revenue  730 non-null    float64
 1   Year     730 non-null    int64
 2   Month    730 non-null    int64
dtypes: float64(1), int64(2)
memory usage: 22.8 KB
None
(730, 3)
              Revenue  Year  Month
Date
2020-07-15   5.328601  2020      7
2020-07-06   5.816199  2020      7
2020-01-15   1.085547  2020      1
2020-02-01   2.442888  2020      2
2021-01-25  11.234359  2021      1
```

Notes. Raw data showing 730 data values. The original column ‘Day’ is converted to a datetime field and is used to index the dataframe based on a given start date of Jan 1, 2020.

Table 2

Descriptive Statistics

In [9]:
```
# descripe numerical data
df.describe()
```

Out[9]:

	Revenue	Year	Month
count	729.000000	729.000000	729.000000
mean	9.849038	2020.499314	6.519890
std	3.825359	0.500343	3.444609
min	0.000793	2020.000000	1.000000
25%	6.916245	2020.000000	4.000000
50%	10.804153	2020.000000	7.000000
75%	12.568041	2021.000000	10.000000
max	18.154769	2021.000000	12.000000

Notes. After removing the initial zero value, there are 729 non-zero data points. Continuous data ranging from a minimum value of 0 to a maximum value of 18.2 with a mean value of 9.8.

Table 3

Augmented Dickey-Fuller Test on raw data

```
In [16]:  # augmented dickey-fuller
          dickey_fuller(df['Revenue'].values, stats=True)

          ADF Statistic: -1.787175
          p-value: 0.386847
          Critical Values:
                  1%: -3.439
                  5%: -2.866
                  10%: -2.569
          Accept H0, data is non-stationary.

Out[16]:  0.3868472463695337
```

Notes. The p-value of 39% indicate the data is not stationary.

Table 4

Augmented Dickey-Fuller Test on differenced data

```
In [18]:  # augmented dickey-fuller
          dickey_fuller(df_stationary['Revenue'].values,
                  stats=True)

          ADF Statistic: -43.503317
          p-value: 0.000000
          Critical Values:
                  1%: -3.440
                  5%: -2.866
                  10%: -2.569
          Reject H0, data is stationary.

Out[18]:  0.0
```

Notes. The p-value of 0.000 indicates the data is stationary.

Table 5

Auto-ARIMA Results – on training data

```
In [77]:  # use auto arima to find best p,d,q
          from pmdarima import auto_arima
          import warnings
          warnings.filterwarnings('ignore')
          pdq = auto_arima(train['Revenue'],
                  trace=True, supress_warings=True)
          #pdq.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=940.179, Time=0.14 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1111.572, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=935.885, Time=0.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=974.705, Time=0.05 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1110.709, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=937.557, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=937.606, Time=0.07 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=938.427, Time=0.21 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=937.012, Time=0.02 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 0.654 seconds
```

Notes. The auto-ARIMA results found the best model is ARIMA(1,1,0)(0,0,0)[0]. This is confirmation of the other stationary analysis that stationarity is achieved in the first-differenced data.

Table 6

Final Model Summary

In [35]:
```
# create final model
model = ARIMA(df['Revenue'], order=(1,1,0))
results = model.fit()
results.summary()
```

Out[35]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	729
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-490.019
Date:	Fri, 22 Jul 2022	**AIC**	984.039
Time:	18:06:43	**BIC**	993.219
Sample:	01-02-2020	**HQIC**	987.581
	- 12-30-2021		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-0.4677	0.033	-14.214	0.000	-0.532	-0.403
sigma2	0.2249	0.013	17.706	0.000	0.200	0.250

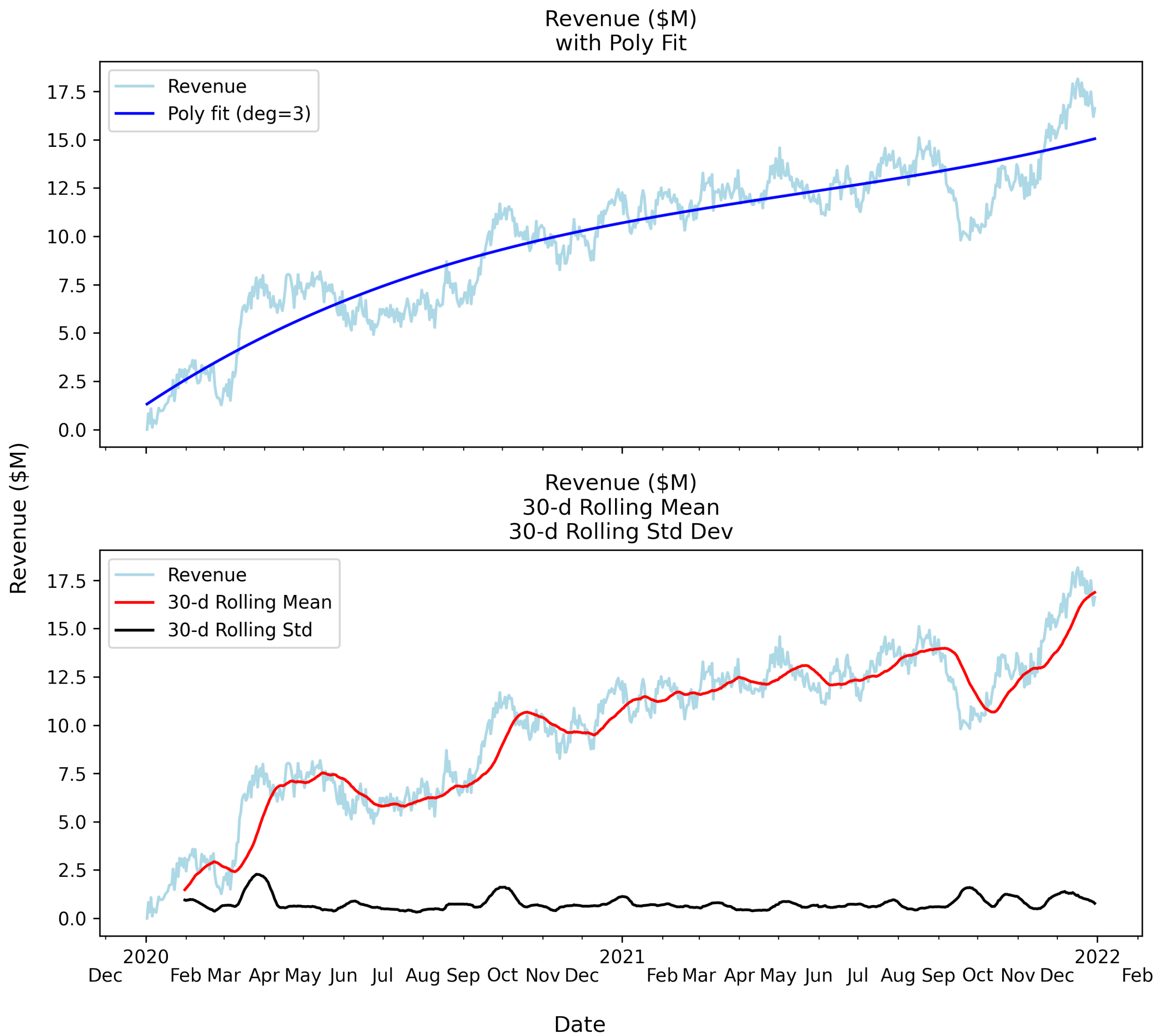
Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.22
Prob(Q):	0.99	**Prob(JB):**	0.33
Heteroskedasticity (H):	1.02	**Skew:**	-0.02
Prob(H) (two-sided):	0.89	**Kurtosis:**	2.73

Notes.

Figures

Figure 1

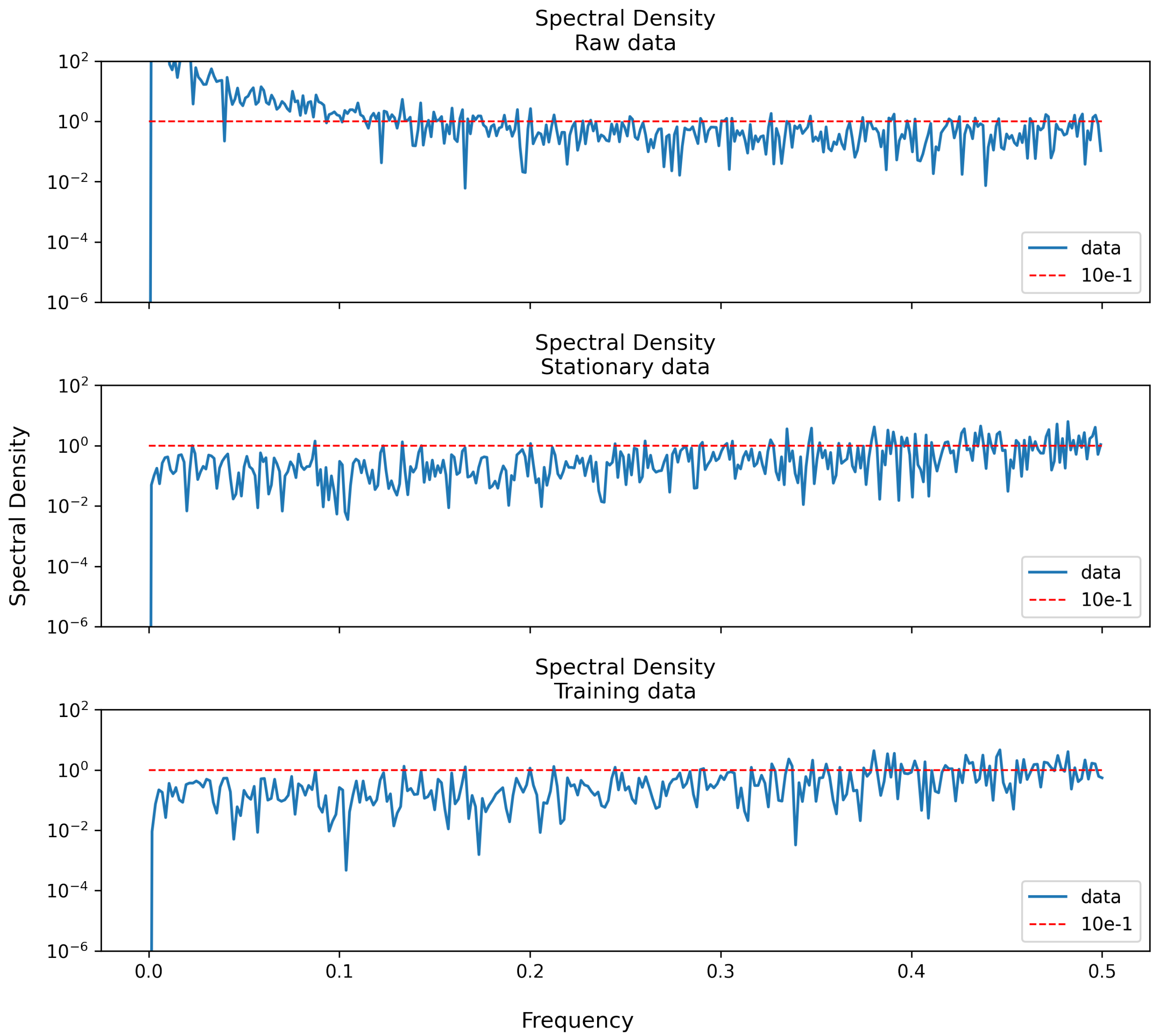
Revenue ($M)



Notes. (top) Generally, trending up towards the right side, not stationary. Also, does not appear to have seasonality. (bottom) Shows original data along with calculated 30-day rolling average. Source: Telecom revenue data (2020).

Figure 2

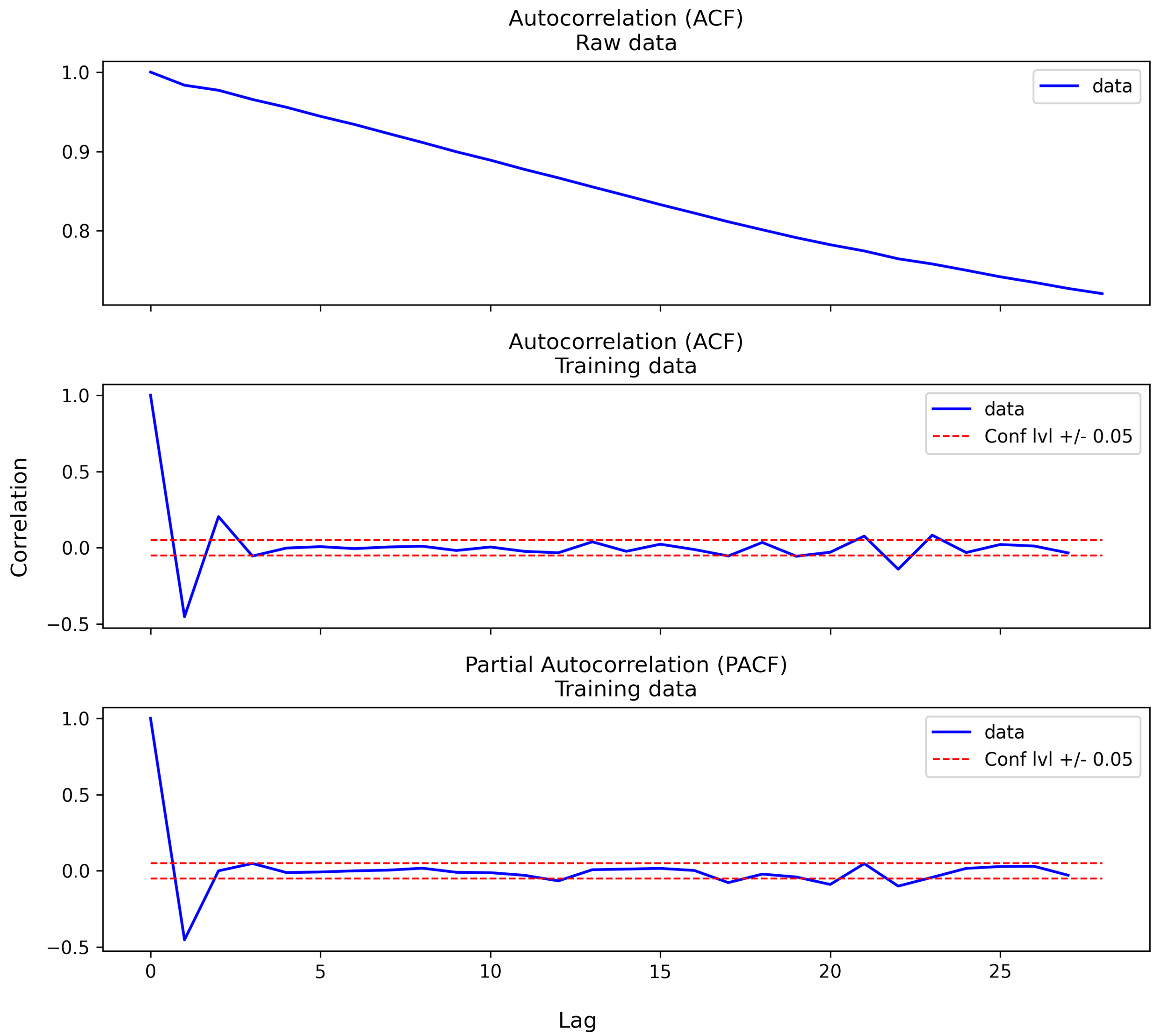
Spectral Density Plots



Notes.

Figure 3

Autocorrelation (ACP) & Partial Autocorrelation (PACF) Plots

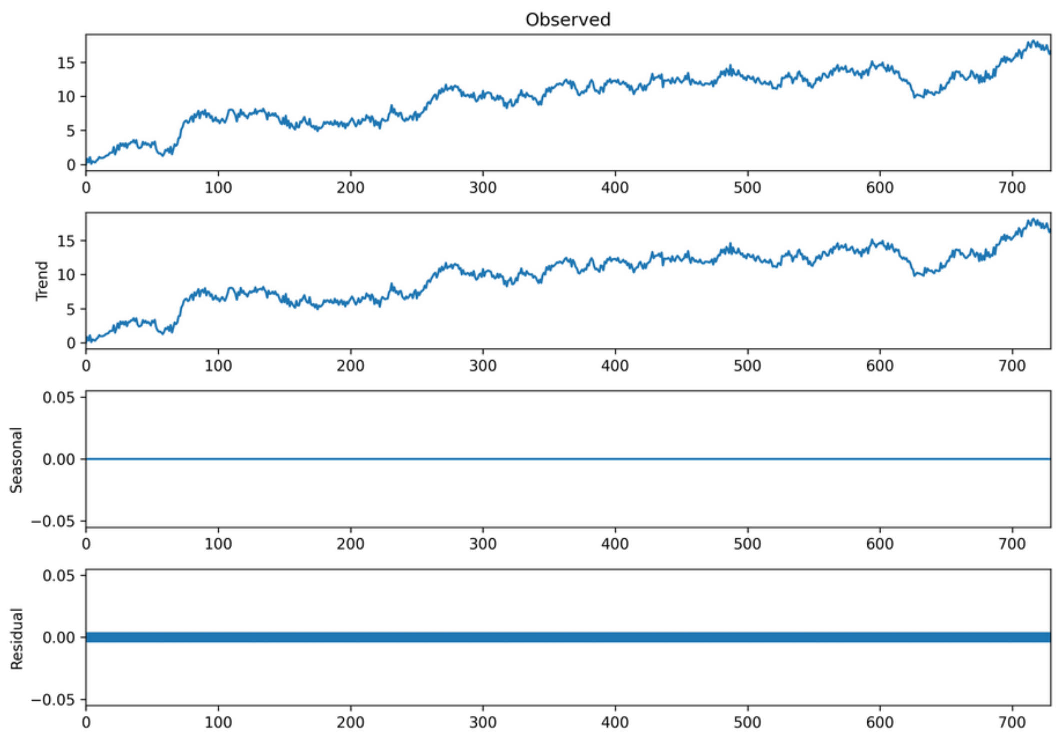


Notes. Steady decrease downward on the ACF and PACF is "exponentially decaying" or "tapering" to the right, these indicators suggest that the time series data is in AR(1) format.

Figure 4

Decomposition Summary - additive

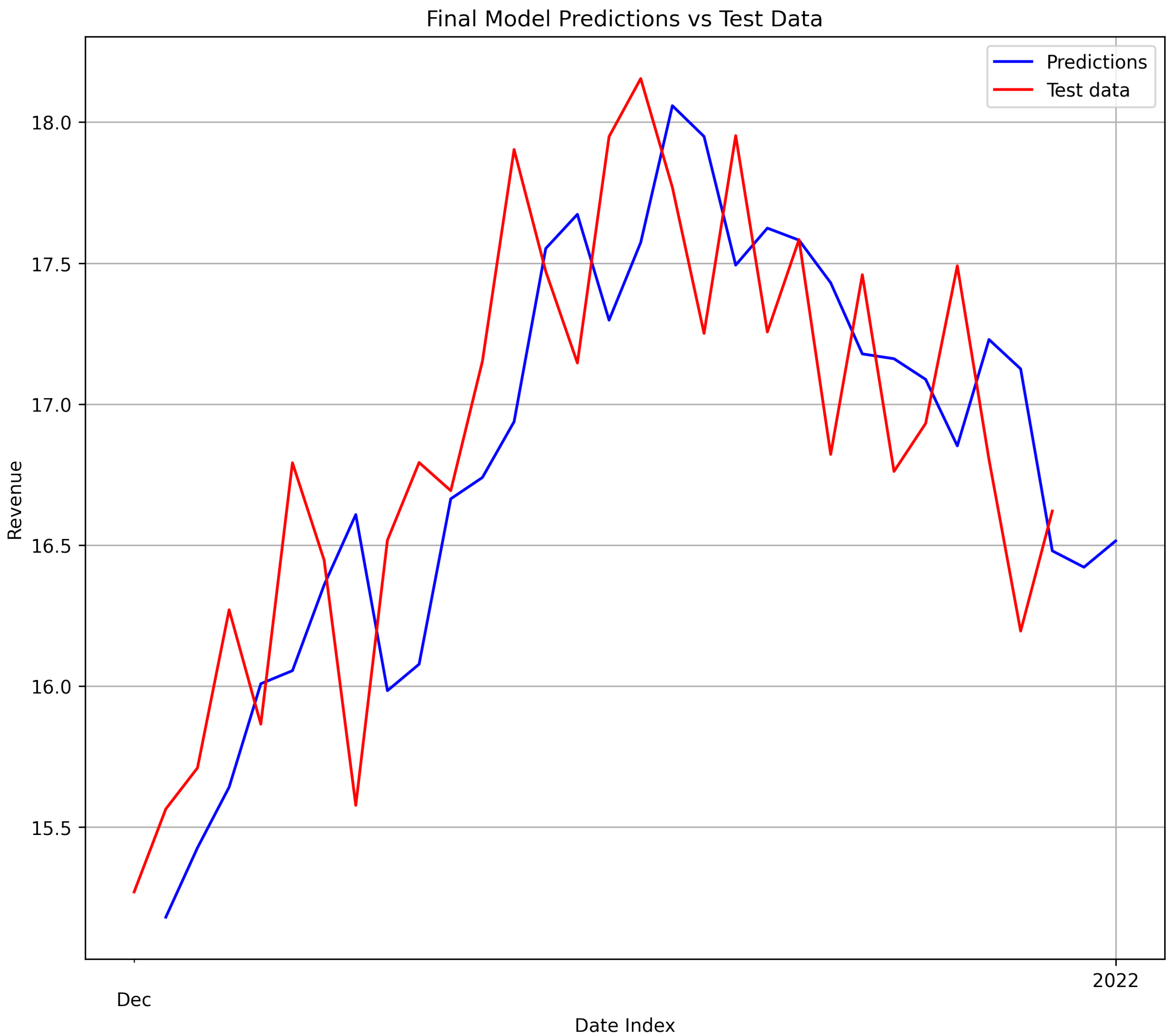
```
In [133]: # decompose revenue data - additive
          # adapted from https://machinelearningmastery.com/decompose-time-series-data-tre
          from matplotlib import pyplot
          from statsmodels.tsa.seasonal import seasonal_decompose
          result = seasonal_decompose(revenue, model='additive', period=1)
          result.plot()
          pyplot.show()
```



Notes. No seasonal trend.

Figure 5

Final Model Predictions vs Test Data



Notes. The final model appears to align with the test data, a good indication that the model is sound.

Appendix A Python Code

```
#!/usr/bin/env python
# coding: utf-8

# # D213 Task 1 Rev 3 - Mattinson

# ## Update & install
# pip install pmdarima
!pip install pmdarima
# ## import packages & read data

# ### import packages

# In[1]:


#import basic libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from scipy import signal


# In[2]:


# import and configure matplotlib
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
get_ipython().run_line_magic('matplotlib', 'inline')
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300


# In[3]:


# import required model libraries
from statsmodels.tsa.stattools import acf, pacf
#from statsmodels.tsa.arima.model import ARIMA
import statsmodels.tsa.stattools as ts
#from statsmodels.tsa.arima_model import ARIMA2
from statsmodels.tsa.arima.model import ARIMA


# In[4]:


# Where to save figures and model diagrams
# adapted code (Geron, 2019)
import os
IMAGES_PATH = os.path.join(".", "figures")
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
```

```
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print('Saving figure: {}'.format(fig_id))
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension,
        dpi=resolution, bbox_inches = "tight")

MODEL_PATH = os.path.join(".", "models")
os.makedirs(MODEL_PATH, exist_ok=True)

TABLE_PATH = os.path.join(".", "tables")
os.makedirs(TABLE_PATH, exist_ok=True)

DATA_PATH = os.path.join(".", "data")
os.makedirs(DATA_PATH, exist_ok=True)


# ### read time data

# In[5]:


def read_time_series(file: str, index: str, start_date=None, freq='d') ->
pd.DataFrame():
    """create dataframe of time series data
    Author: Mike Mattinson
    Date: June 22, 2022

    Parameters
    ----------
    file: str
       filename of time series data
    index: str
       column name of date index
    start_date: datetime
       (optional) if using specific start date
    freq: str
       (default) '24H' 24-hour increments

    Returns
    -------
    tsdf: pd.DataFrame()
       time series dataframe

    """

    # read and initialize index
    tsdf =  pd.read_csv(file)
    tsdf.set_index(index, inplace=True)

    # re-index on specific optional start_date
    index_label = 'Date'
    if(start_date is not None):
        tsdf[index_label] = (pd.date_range(
            start=start_date,
            periods=tsdf.shape[0],
            freq=freq))
```

```
        tsdf.set_index(index_label, inplace=True)
        tsdf['Year'] = tsdf.index.year
        tsdf['Month'] = tsdf.index.month
        #tsdf['Weekday Name'] = tsdf.index.weekday_name

    # print out summary
    print(tsdf.info())
    print(tsdf.shape)
    print(tsdf.sample(5, random_state=0))

    return tsdf # time series dataframe


# In[6]:


# read time series data from CSV file
from datetime import datetime
df =  read_time_series(
    file='data/teleco_time_series.csv',
    index='Day', freq='d',
    start_date=datetime(2020,1,1)
)


# ## clean & explore data

# In[7]:


# show sample from dataframe
n_rows=10
df.sample(n_rows, random_state=0)


# In[8]:


# drop zero values
df= df[df['Revenue'] != 0]


# In[9]:


# descripe numerical data
df.describe()


# In[10]:


#find rolling mean of previous n periods
n_days = 30
df['rolling_mean'] = df['Revenue'].rolling(window=n_days).mean()
df['rolling_std'] = df['Revenue'].rolling(window=n_days).std()
```

```
# In[11]:


#check missing data
df.isnull().any()


# ### export cleaned data

# In[12]:


# export cleaned data to file
df.to_csv('tables\cleaned.csv', index=True, header=True)
print(df.info())
print(df.shape)


# ### revenue plot with polyfit regression

# https://stackoverflow.com/questions/39801403/how-to-derive-equation-from-
numpys-polyfit
!pip install sympy
# In[13]:


# equation of poly fit
from sympy import S, symbols, printing
x = pd.Series(range(df.shape[0]))
y = df['Revenue'].values
n_deg = 3
p = np.polyfit(x, y, deg=n_deg)
f = np.poly1d(p)
e = symbols("x")
poly = sum(S("{:6.7f}".format(v))*e**i for i, v in enumerate(p[::-1]))
eq_latex = printing.latex(poly)
print(p)
print(poly) # won't include zero terms


# In[14]:


# visualize raw revenue data
x = pd.Series(df.index.values) # if using date
x2 = pd.Series(range(df.shape[0])) # if using date index
fig, ax = plt.subplots(2,1, figsize = (9, 8), sharex=True, sharey=True)
ax[0].plot(x, df.Revenue, 'lightblue', label='Revenue')
ax[1].plot(x, df.Revenue, 'lightblue', label='Revenue')
ax[0].plot(x,f(x2),"b", label='Poly fit (deg=' + str(n_deg) + ')')
ax[0].legend()
ax[0].set_title('Revenue ($M)\nwith Poly Fit')
ax[1].plot(x,df['rolling_mean'], "red",
         label=str(n_days) + '-d Rolling Mean')
ax[1].plot(x,df['rolling_std'], "black",
         label=str(n_days) + '-d Rolling Std')
```

```
ax[1].set_title('Revenue ($M)\n30-d Rolling Mean\n30-d Rolling Std Dev')
ax[1].legend()
import matplotlib.dates as mdates
ax[1].xaxis.set_major_locator(mdates.YearLocator())
ax[1].xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax[1].xaxis.set_minor_locator(mdates.MonthLocator())
ax[1].xaxis.set_minor_formatter(mdates.DateFormatter('\n%b'))
fig.supxlabel('Date') # common x label
fig.supylabel('Revenue ($M)') # common y label
#plt.gcf().text(0, -.1, "${}$".format(eq_latex), fontsize=14)
title = 'Revenue ($M)'
save_fig(title)


# Generally, trending up and not stationary. Also, does not appear to have
seasonality.

# ## diff data - make stationary

# ### dickey-fuller - on raw data, non-stationary data

#
https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.
html
#
# https://machinelearningmastery.com/time-series-data-stationary-python/
#
# https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-
test/
#
# https://www.quora.com/What-is-an-Augmented-Dickey-Fuller-test

# In[15]:


import statsmodels.tsa.stattools as ts
def dickey_fuller(
    array: np.array,
    critical=0.05,
    stats=False) -> float:
    """return p-value of augmented dickey-fullter test
    Author: Mike Mattinson
    Date: June 29, 2022

    Parameters
    ----------
    array: np.array # array-like
       array of values to be evaluated
    critical: float (default=0.05)
       critical value
    stats: bool (default=False)
        include stats is output or not

    Returns
    -------
    pvalue: float
        p-value
```

```
    """
    result = ts.adfuller(array, autolag='AIC')
    pvalue = result[1]

    if(stats):
        print('ADF Statistic: %f' % result[0])
        print('p-value: %f' % pvalue)
        print('Critical Values:')
        for key, value in result[4].items():
            print('\t%s: %.3f' % (key, value))

    if pvalue <= critical:
        print('Reject H0, data is stationary.')
    else:
        print('Accept H0, data is non-stationary.')

    return pvalue


# In[16]:


# augmented dickey-fuller
dickey_fuller(df['Revenue'].values, stats=True)


# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.diff.html

# In[17]:


"""Calculates difference of Dataframe element compared with another
element in the Dataframe (default is element in previous row)."""
df_stationary = df.diff(periods=1,axis=0).dropna()
print(df_stationary.info())
print(df_stationary.shape)
#print(df_stationary.describe())


# ### dickey-fuller - on differenced data

# In[18]:


# augmented dickey-fuller
dickey_fuller(df_stationary['Revenue'].values,
        stats=True)


# ### export stationary data

# In[19]:


# export stationary data to file
df_stationary.to_csv('tables\stationary.csv', index=True, header=True)
```

```
print(df_stationary.info())
print(df_stationary.shape)


# ## train test split

# https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.h
tml
#setup training and test data 80/20
test_size = int(.20 * df_stationary.shape[0])  # last 20%
train, test = train_test_split(df_stationary,
            test_size=test_size, shuffle=False)
print('training: {}'.format(train.shape))
print('testing: {}'.format(test.shape))
# In[20]:


# use last 30 days for testing
train = df.iloc[:-30]
test = df.iloc[-30:]
print('training: {}'.format(train.shape))
print('testing: {}'.format(test.shape))


# In[21]:


test.info()


# In[22]:


test.describe()


# In[23]:


train.info()


# In[24]:


train.describe()


# ## spectral density

#
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.periodogram
.html
#
# https://www.geeksforgeeks.org/plot-the-power-spectral-density-using-
matplotlib-python/
```

```
#
# https://online.stat.psu.edu/stat510/lesson/12/12.1
#
# https://web.stanford.edu/class/earthsys214/notes/series.html
#

# In[25]:


from scipy import signal
def sd_plot(data, target, ax, i: int, title: str) -> None:
    f, Pxx = signal.periodogram(data[target])
    ax[i].semilogy(f, Pxx, label='data')
    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].set_ylim([1e-6, 1e2])
    ax[i].legend()
    return None


# In[26]:


# plot spectral density
fig, ax = plt.subplots(3,1, figsize = (9, 8), sharex=True, sharey=True)
sd_plot(data=df, target='Revenue', ax=ax, i=0,
        title='Spectral Density\nRaw data')
sd_plot(data=df_stationary, target='Revenue', ax=ax, i=1,
        title='Spectral Density\nStationary data')
sd_plot(data=train, target='Revenue', ax=ax, i=2,
        title='Spectral Density\nTraining data')
title = 'Spectral Density'
fig.supxlabel('Frequency') # common x label
fig.supylabel('Spectral Density') # common y label
save_fig(title)


# ## acf & pacf plots

# In[27]:


from statsmodels.tsa.stattools import acf
def acf_plot(data, target, ax, i: int, conf: bool, title: str) -> None:

    acf_values = acf((data[target].values))
    acf_df = pd.DataFrame([acf_values]).T
    acf_df.columns = ['ACF']
    ax[i].plot(acf_df.ACF, 'b-', label='data')
    if(conf):
        ax[i].hlines(y=0.05, xmin=0, xmax=len(acf_values), lw=1,
                     linestyles='--', color='r', label='Conf lvl +/- 0.05')
        ax[i].hlines(y=-0.05, xmin=0, xmax=len(acf_values), lw=1,
                     linestyles='--', color='r')
    ax[i].set_title(title)
    ax[i].legend()
```

```
    return None


# In[28]:


from statsmodels.tsa.stattools import pacf
def pacf_plot(data, target, ax, i: int, conf: bool, title: str) -> None:

    pacf_values = pacf((data[target].values))
    pacf_df = pd.DataFrame([pacf_values]).T
    pacf_df.columns = ['PACF']
    ax[i].plot(pacf_df.PACF, 'b-', label='data')
    if(conf):
        ax[i].hlines(y=0.05, xmin=0, xmax=len(pacf_values), lw=1,
                 linestyles='--', color='r', label='Conf lvl +/- 0.05')
        ax[i].hlines(y=-0.05, xmin=0, xmax=len(pacf_values), lw=1,
                 linestyles='--', color='r')
    ax[i].set_title(title)
    ax[i].legend()

    return None


# In[29]:


# autocorrelation/partial autocorrleation
fig, ax = plt.subplots(3,1, figsize = (9, 8), sharex=True, sharey=False)
acf_plot(data=df, target='Revenue', ax=ax, i=0, conf=False,
         title='Autocorrelation (ACF)\nRaw data')
acf_plot(data=train, target='Revenue', ax=ax, i=1, conf=True,
         title='Autocorrelation (ACF)\nTraining data')
pacf_plot(data=train, target='Revenue', ax=ax, i=2, conf=True,
          title='Partial Autocorrelation (PACF)\nTraining data')
fig.supxlabel('Lag') # common x label
fig.supylabel('Correlation') # common y label
title = 'Autocorrelation - Partial Autocorrelation Plots'
save_fig(title)


# ## decompose cleaned data

# https://machinelearningmastery.com/decompose-time-series-data-trend-
seasonality/

# In[30]:


# decompose cleaned data - additive
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df['Revenue'].values,
            model='additive', period=1)
result.plot()
title = 'Decomposition on cleaned data'
save_fig(title)
```

```
# decompose log data
result = seasonal_decompose(lnrevenue, model='additive', period=1)
result.plot()
pyplot.show()# decompose revenue data - multiplicative
# adapted from https://machinelearningmastery.com/decompose-time-series-data-
trend-seasonality/
from matplotlib import pyplot
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(revenue, model='multiplicative', period=1)
result.plot()
pyplot.show()
# ## auto find p,d,q values

# In[31]:


# use auto arima to find best p,d,q
from pmdarima import auto_arima
import warnings
warnings.filterwarnings('ignore')
pdq = auto_arima(train['Revenue'],
        trace=True, supress_warings=True)
#pdq.summary()


# ## final model

# ### model (1,1,0) on original data

# In[35]:


# create ARIMA model (1,1,0) on training data
model = ARIMA(df['Revenue'], order=(1,1,0))
results = model.fit()
results.summary()


# ### make a forecast outside of sample data

# In[46]:


# make forecast outside of sample
results.forecast(30)


# ## plot forecast of final model (30-day) compared to the test data

# In[36]:


df.tail(30)


# In[37]:
```

```
# prediction for last 30-days
predictions = results.predict(start=700, end=730, type='levels')
print(predictions)


# In[43]:


fig, ax = plt.subplots(1,1, figsize = (9, 8))
pred = plt.plot(predictions, "b", label='Predictions')
plt.plot(test['Revenue'], "r", label='Test data')
plt.xlabel("Date Index")
plt.ylabel("Revenue")
title = 'Final Model Predictions vs Test Data'
plt.legend()
plt.grid()
import matplotlib.dates as mdates
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax.xaxis.set_minor_locator(mdates.MonthLocator())
ax.xaxis.set_minor_formatter(mdates.DateFormatter('\n%b'))
plt.title(title)
save_fig(title)


# In[ ]:
```