**PA TASK 1: Time Series Modeling**

**D213 – Advanced Data Analytics**

Michael Lawson

Western Governors University

D213 – Advanced Data Analytics

PA Task 1: Time Series Modeling

Festus Elleh

09/20/2022

Michael C. Lawson | Student ID: #001479557 | Program: MS Data Analytics (5/1/21) |

Program Mentor: Emil Stoica | 859-265-1888 (EST) | [mlaws77@wgu.edu](mailto:mlaws77@wgu.edu)

**Part I:**

A.

1.  Question:

What should the revenue forecast for the hospital chain in the upcoming year?

2.  Goal:

The goal of the analysis will be to create a predictive model forecasting revenue in the future by analyzing the last two years to find seasonality, trends and other factors.

**Part II:**

B.

Time series models assume the data is stationary. Stationary means that the model has a normal distribution, and the mean, variance and autocorrelation is constant over the period in the model. The model also assumes there are no outliers and that residuals are not autocorrelated. The error term is randomly distributed and assumed to be uncorrelated. Time series models assume that over time, the distribution of the data does not change. Stationarity means that the data is not growing or shrinking; therefore, the data has no trends. How each value is related in the time series to its neighbors stay the same (Datacamp, n.d). Autocorrelation measures correlation between current values in a time series against past values and is used to predict future values as a linear function of past values.

**Part III:**

C.

1.  Line graph:

plt.figure(figsize=(16,4))

plt.plot(data['Revenue'])

plt.title('Revenue Chart')

plt.xlabel('Day')

plt.ylabel('Revenue in million dollars')

plt.grid(True)

plt.show()

Chart, line chart

Description automatically generated

2.

No gaps exist in the ‘Day’ column. A new column for date converts the ‘Day’ columns to a time series beginning 9/1/2019.

# Convert 'Day' to time series DateTime format

data['Date'] = (pd.date\_range(start=pd.datetime(2019, 1, 1),

periods=data.shape[0], freq='24H'))

# Set the Date as an index

data.set\_index('Date',inplace=True)

data

Graphical user interface, text

Description automatically generated

# Check index dtype

data.index

Text

Description automatically generated

3.  Stationarity:

# Run ADFuller test to check if data is stationary

result = adfuller(data['Revenue'])

print("Test statistics: ", result[0])

print("p-value: ", result[1])

print("Critical values: ", result[4])



if result[1]<= 0.05:

print("The time series is stationary. Reject the null hypothesis.")

else:

print("The time series is non-stationary. Fail to reject null hypothesis.")



# Make the data stationary

df\_stationary = data.diff().dropna()

# Run ADFuller test on stationary data to check that changes worked

print("test statistics: ", st\_result[0])

print("p-value: ", st\_result[1])

print("critical values: ", st\_result[4])



st\_result = adfuller(df\_stationary['Revenue'])

if st\_result[1]<= 0.05:

print("The time series is stationary. Reject the null hypothesis.")

else:

print("The time series is non-stationary.Fail to reject null hypothesis.")



4.  Steps to prepare the data:

1. # Import libraries/packages.

import pandas as pd

import numpy as np

import matplotlib as mpl

from statsmodels.tsa.arima\_model import ARMA

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.stattools import acf, pacf

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.graphics.tsaplots import plot\_pacf

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

1. #Import data

med\_file = r"C:\Users\mlaws\OneDrive - Western Governors University\Documents\WGU\D213\medical\_time\_series .csv"

data = pd.read\_csv(med\_file)

1. # Exploratory data analysis. Visualize data. Search for nulls.

data.shape



data.info()

Text

Description automatically generated

data.isnull().sum()

Text

Description automatically generated with medium confidence

data.head()

Graphical user interface, table

Description automatically generated with medium confidence

1. # Convert 'Day' to time series

data['Date'] = (pd.date\_range(start=pd.datetime(2019, 1, 1),

periods=data.shape[0], freq='24H'))

# Set index to Date

data.set\_index('Date',inplace=True)

data

Graphical user interface

Description automatically generated with medium confidence

# Check index dtype

data.index

Text, letter

Description automatically generated

1. # Split data into train/test sets

X\_train = data.loc[:'2020-09-30']

X\_test = data.loc['2020-10-01':]

print('X\_train Shape', X\_train.shape)

print('X\_test Shape', X\_test.shape)



The data is split into two groups, training and test data sets. The training data has been taken from all data before 9/30/2020 and the test data is any tuples after 10/01/2020. Splitting the data allows a model to be trained on one data set, while setting another data set aside to be tested. The model trained on the training set then can be tested by running the same model on the test set to verify that the prediction line matches the actual data in the test set. For this analysis, the data was split according to dates so that the prediction could be run as a future forecast of the training set, and the test set contains data that is in the future of the training set. 639 rows went into the training set and 92 rows went to the test data set.

1. # Export cleaned dataset.

pd.DataFrame(df\_stationary).to\_csv(r"C:\Users\mlaws\OneDrive - Western Governors University\Documents\WGU\D213\medical\_time\_series\_cleaned.csv")

**Part IV:**

D.  1.

* + 1. Cycles that repeat over time are referred to as seasonality. A cycle is considered seasonal if it repeats at the same frequency such as daily, weekly, monthly, yearly, etc… Viewing seasonality helps identify patterns in the data related to time periods. Seasonal patterns can be used to create predictions and forecast future revenue. Looking at the plot below, it is apparent that the pattern repeats approximately every 90 days, or three months.

# Plot the Seasonality

plt.title('Seasonality')

decomp.seasonal.plot()

A picture containing text, antenna

Description automatically generated

* + 1. The trend refers to the overall direction of the series, whether upward or downward. The visualization for trend shown below reveals that the time series has a slightly downward trend in revenue from beginning to end.

# Plot the trend

plt.title('Trend')

decomp.trend.plot()

Chart, line chart

Description automatically generated

* + 1. Auto correlation function (ACF) is a plot of autocorrelation of the time series, showing how data is correlated across different time periods. Positive values denote correlation, while negative values denote that the data is not correlated. Lag is a function used in the algorithm to offset the data for comparison between the data and the offset data. Partial auto correlation function (PACF) is the correlation between the lagged time series and the original time series after the correlation at smaller lags is subtracted (Datacamp, n.d). The plotted ACF inside the shaded area is statistically insignificant. Significant ACF shows up as the plots outside of the shaded area. The plot for ACF below rises and drops on each side of the line and stays near the shaded area, which is an indicator that the data is stationary. Non-stationary data would display as a gradual decline from beginning to end in the ACF plot. The points outside of the shaded area indicate significant Lag in ACF at 1 and geometric decay at Lag 20 in both the ACF and PACF models.

# Plot ACF on the stationary data

X = df\_stationary['Revenue']

y = df\_stationary['Day']

plot\_acf(X, lags=30, zero=False)

plt.show()

Chart, box and whisker chart

Description automatically generated

# Plot PACF

plot\_pacf(X, lags=30, zero=False)

plt.show()

Chart

Description automatically generated with low confidence

* + 1. Spectral density graphs the frequencies (number of observations before the seasonal pattern is repeated) related to autocovariance (the covariance of two elements in the series) time domain. The plot for spectral density below appears to be random, except for the large drop at the very beginning of the plot, which indicates a strong negative autocorrelation.

# Plot spectral density

f, Pxx\_den = signal.periodogram(df\_stationary['Revenue'])

plt.semilogy(f, Pxx\_den)

plt.ylim([1e-6, 1e2])

plt.title('Spectal Density')

plt.xlabel('Frequency')

plt.ylabel('Spectal Density')

plt.show()

Chart, line chart

Description automatically generated

* + 1. The decomposed time series provides a summary of seasonality, trend and residual in order to detect the presence or lack thereof of each component. The visualization below reveals that seasonality repeats steadily, and trend is the less predictable component, and the reason the revenue plot line doesn’t steadily repeat.

# Decompose Time Series

decomp = seasonal\_decompose(df\_stationary['Revenue'], period=90)

# Plot decomposition

decomp.plot()

# Check for seasonality in the data

plt.show()

Application

Description automatically generated with low confidence

* + 1. Residuals refer to the decomposed time series after the trend and seasonality have been removed. Time series forecasting assumes that the residuals are not correlated. The seasonal\_decompose method can be used to generate and plot the residuals. Residuals in the plot below appear to be evenly distributed with a mostly constant variance.

Chart, line chart

Description automatically generated

2.

# Run Auto-Arima to discover the best p, d & q fit for the model

model = auto\_arima(X\_train['Revenue'], start\_p=1, start\_q=1,

test='adf',

max\_p=5, max\_q=5,

m=1,

d=1,

seasonal=False,

start\_P=0,

D=None,

trace=True,

error\_action='ignore',

suppress\_warnings=True,

stepwise=True)

Text

Description automatically generated with low confidence

model.summary()

A screenshot of a computer

Description automatically generated with low confidence

3.

# Forecast

prediction, confint = model.predict(n\_periods=92, return\_conf\_int=True)

prediction

Table

Description automatically generated

cf= pd.DataFrame(confint)

prediction\_series = pd.Series(prediction,index=X\_test.index)

fig, ax = plt.subplots(1, 1, figsize=(15, 5))

ax.plot(data['Revenue'])

ax.plot(prediction\_series)

ax.fill\_between(prediction\_series.index,

cf[0],

cf[1],color='grey',alpha=.3)

Chart, line chart

Description automatically generated

model.plot\_diagnostics(figsize=(14,10))

plt.show()

Chart, line chart

Description automatically generated

4.

Text

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with low confidence

Table

Description automatically generated

# Forecast output and Calculations

cf= pd.DataFrame(confint)

prediction\_series = pd.Series(prediction,index=X\_test.index)

fig, ax = plt.subplots(1, 1, figsize=(15, 5))

ax.plot(data['Revenue'])

ax.plot(prediction\_series)

ax.fill\_between(prediction\_series.index,

cf[0],

cf[1],color='grey',alpha=.3)

Chart, line chart

Description automatically generated

# Prediction data zoomed in

prediction\_series = pd.Series(prediction,index=X\_test.index)

fig, ax = plt.subplots(1, 1, figsize=(15, 5))

ax.plot(data['Revenue']['2020-10-01':])

ax.plot(prediction\_series)

ax.fill\_between(prediction\_series.index,

cf[0],

cf[1],color='grey',alpha=.3)

Chart, line chart

Description automatically generated

# Create function for symmetric mean absolute percentage error (SMAPE)

def calcsmape(actual, forecast):

return 1/len(actual) \* np.sum(2 \* np.abs(forecast-actual) / (np.abs(actual) + np.abs(forecast)))

# Calculate SMAPE

smape=calcsmape(X\_test.Revenue,prediction)

print('Symetric Mean Absolute Percentage Error (SMAPE): ', smape)



5.  Code:

An HTML file is attached along with this document containing all the code. The HTML file is named ‘D213\_PA1.html’.

**Part V:  Data Summary and Implications**

E.

1.

1. The process for finding the best ARIMA model starts with generating the ACF and PACF. Various combinations of AR and MA are run to find the model with the lowest AIC score. Auto\_arima is a python algorithm used to quickly do this for you to find the best values for p, d & q. In this data set, the best values turned out to be (1,1,0).

# Run Auto-Arima to discover the best p,q & d fit for the model

model = auto\_arima(X\_train['Revenue'], start\_p=1, start\_q=1,

test='adf',

max\_p=5, max\_q=5,

m=1,

d=1,

seasonal=False,

start\_P=0,

D=None,

trace=True,

error\_action='ignore',

suppress\_warnings=True,

stepwise=True)

Text

Description automatically generated with low confidence

1. The prediction interval of the forecast is one day. The data set includes daily revenue for a two-year period, and the ARIMA model identifies the seasonality and correlations to predict revenue by a daily interval.
2. The data includes daily revenue for a two-year interval, so the forecast could only predict up to one year of future data. The earliest predictions are most accurate, as can be detected by the expanding grey field for prediction around the actual data in the model plot. Longer prediction periods would require more data from the past.
3. The function auto\_arima was used to find the suitable seasonal order. The Symmetric Mean Absolute Percentage Error (SMAPE) is an accuracy measure. It is based on the relative error. The absolute error divided by the magnitude of the exact value is how the relative error is calculated. The lower the SMAPE value, the higher its accuracy. The model had a SMAPE of .1029, which denotes a 90% accuracy.

2.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

3.

The company can use the model to forecast the revenue for the next six months. This will allow the stakeholders to budget knowing the expected revenues and implement plans toward reducing readmissions. The prediction is a steady line, meaning that the budget can be configured around the current revenue.

**References**

Elleh, Festus. June 19, 2022. *D213 T1 Webinar Video.* <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=efceba6c-e8ef-47a2-b859-aec400fe18e7>

Masum, PhD, Mohammad. August 13, 2020. *Time Series Analysis: Identifying AR and MA using ACF and PACF Plots.*[https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8](%20https:/towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8)

DOR. October 7,2021.  *Forecasting Time Series with Auto-Arima.* <https://www.alldatascience.com/time-series/forecasting-time-series-with-auto-arima/>