Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2022, 9(11):110-122

Research Article ISSN: 2394 - 658X

Performance Evaluation using Time Series Modeling

Kailash Alle

Sr. Software Engineer, Comscore Inc
kailashalle@gmail.com

ABSTRACT

In this paper, we present an investigation into hidden patterns within customer data for a telecommunications company. Utilizing time-series analysis, we aim to uncover critical revenue trends for more accurate forecasting. Our findings will provide stakeholders with a deeper understanding of organizational revenue trends, enabling more informed decision-making and strategic planning.

Keywords: Time Series, Modeling, Analysis

INTRODUCTION

Understanding customers is one of the most important aspects of customer relationship management that directly influences a company's long-term success. When a corporation has a greater understanding of its consumers' traits, this could good target promotion and advertising campaigns for them, resulting in higher long-term earnings.

You operate as an investigator for a telecoms business that wants to understand more about its hidden patterns in the data. You've been given with conducting time-series research on customer research to discover critical revenue trends for forecasting, it will help the stakeholders for better understand the organizational revenue trend.

Analytical Question

Which customers are likely to churn? What are the company's most important revenue trends that assist stakeholders to forecast?

The Time series technique will be used to solve this analysis.

Goals and Objectives

Everybody in the organization will profit from recognizing, with some degree of certainty, which customers will be able to churn since it will give importance to hidden revenue patterns. This data analysis' purpose is to give numerical information to company stakeholders to assist them in better understand their customers and forecasting future trends.

JUSTIFICATION FOR THE METHOD

Assumptions Summary: Time Series Modelling

For forecasting a time-series, we will use ARIMA modeling is one of the greatest forecasting strategies.

It's an expansion of the ARIMA method that considers seasonality in data. ARIMA is a time-series data forecasting statistical model. The ARIMA model is a regression type equation with lags of the dependent variable and/or lags of the forecast errors as independent variables. In this regression we'll be predicting whether a customer will churn or not, the ARIMA model equations

$$y'(t) = c + \phi 1 * y'(t-1) + \cdots + \phi p * y'(t-p) + \theta 1 * \varepsilon(t-1) + \cdots + \theta q * \varepsilon(t-q) + \varepsilon$$

In the equation there are three terms:

AR: Auto Regression

The preceding elements of the time series are regressed, i.e., y(t-1), y(t-2), and so on. The lag order is indicated by the letter p.

I: Integration:

Differentiation is used to maintain the time series stationary. The difference's order is represented by the letter d.

MA: Moving average

The time series is transformed using residuals from previous observations, such as error(t-1), error(t-2), and so on. The error delay sequence is symbolized by the letter q.

y′t is the conditional variance series, 1 is the first AR term's coefficient, p is the AR term's order, 1 is the first MA term's coefficient, q is the MA term's order, and t is the error in the above equation.

The initial number of ARIMA models is determined using the autocorrelation function (ACF) and partial autocorrelation (PACF) graphs.

We must confirm that the time series is stationary before using ARIMA or any of its extensions.

We can only investigate the behavior of the time series for that period if it is not stationary. If the time series is not stationary, each period will have its own specific behavior, making it impossible to anticipate or predict in future time periods.

A white noise series is a random series with the same distribution of independent variables. For a white noise series, it is also considered that the mean is zero. A series that does not vary over time is called a stationary series. The series' variance is consistent over time; it may fluctuate, but it will behave similarly from one period to the next. Because a time series is a collection of data in chronological order, there is a presumption of normalcy for all-time series data types. Some time series are auto correlated, implying that each observation is linked to the one before it. The higher the autocorrelation, the more predictable the data set is.

ADVANTAGES/BENEFIT OF THE TOOL

Tools will be used

For this assessment, I'll use Python because the study will be supported by Jupyter notebooks in Python and I Python. Python includes many established data science and machine learning tools, straightforward, and extensible programming style, and grammar. Python is cross-platform, so it will function whether the analysis is viewed on a Windows PC or a MacBook laptop. When compared to other programming languages such as R or MATLAB, it is quick (Massaron, p. 8). In addition, Python is often regarded in popular media as the most widely used programming language for data science and media (CBTNuggets, p. 1).

NumPy used to work with arrays,
Pandas used to load datasets,
Matplotlib used to plot charts,
ARIMA import from statsmodels
Scikit-learn used for machine learning model classes,
SciPy used for mathematical problems, specifically linear algebra transformations, and
Seaborn used for a high-level interface and appealing visualizations.

Using the Pandas library and its accompanying "read csv" function to transform our data as a dataframe is a quick, exact example of loading a dataset and constructing a variable efficiently:
imported pandas as pd, df(dataframe) = pd.read csv('ChurnData.csv')

DATA OBJECTIVES

Line Graph Visualization using Time-series:

- Using Pandas' read csv command, read the data collection into python programming.
- Using the info () and description () methods, evaluate the data for a better understanding of the input data.
- Using the variable "teleco df" to name the dataset, and "df" to name the data frame's subsequent usable slices.
- Check for misspellings, strange variable names, and data that is missing.
- Draw the line graph visualizations.
- First step in line graph time series analysis import the data into data frame using python, the "teleco_df" csv as teo coloumns "Day" and "Revenue"

Include standard imports all the required references:

```
# Standard data science imports
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
# Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# Scikit-learn
import sklearn
from sklearn import datasets
from sklearn import preprocessing
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

Change font and color of the Matplotlib:

```
In [2]: # Change color of Matplotlib font
        import matplotlib as mpl

        COLOR = 'white'
        mpl.rcParams['text.color'] = COLOR
        mpl.rcParams['axes.labelcolor'] = COLOR
        mpl.rcParams['xtick.color'] = COLOR
        mpl.rcParams['ytick.color'] = COLOR
```

Increase display cell-width

```
In [3]: # Increase Jupyter display cell-width
        from IPython.core.display import display, HTML
        display(HTML("<style>.container { width:75% !important; }</style>"))
```

Ignore warning codes

```
In [4]: # Ignore Warning Code
        import warnings
        warnings.filterwarnings('ignore')
```

Dataset

```
# Load data set into Pandas dataframe
teleco_df = pd.read_csv('C:/Kailash/Rekha/D213/data/teleco_time_series.csv', header=0,names =['Day','Revenue'],index_col = 0)
```

Dataset size

```
In [24]: teleco_df.shape

Out[24]: (731, 1)
```

Data frame Info

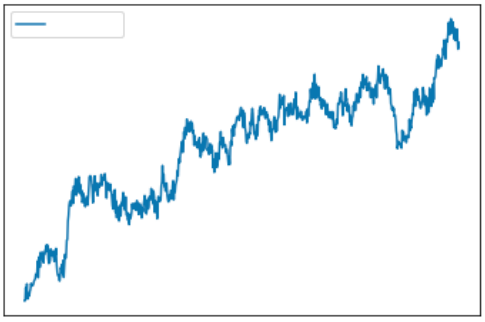
```
In [29]: teleco_df.info

Out[29]: <bound method DataFrame.info of         Revenue
         Day
         1      0.000000
         2      0.000793
         3      0.825542
         4      0.320332
         5      1.082554
         ..          ...
         727   16.931559
         728   17.490666
         729   16.803638
         730   16.194813
         731   16.620798

         [731 rows x 1 columns]>
```

Line graph created using pyplot:

```
# Visualizing time series in python
# 1. Line graph with matplotlib pyplot module
plt.figure(figsize=(12,8))
teleco_df.plot()
plt.title('Time series Analysis')
plt.ylabel('Revenue')
plt.xlabel('Days')
```

```
Text(0.5, 0, 'Days')
```

```
<Figure size 864x576 with 0 Axes>
```



Shared axis plot with line graph:

```
# 2. Shared axis plot
# Getting the cumulative total of teleco_df
# The new object inherits the original index
cumsum_churn = np.cumsum(teleco_df)
```

Cumulative total of dataset:

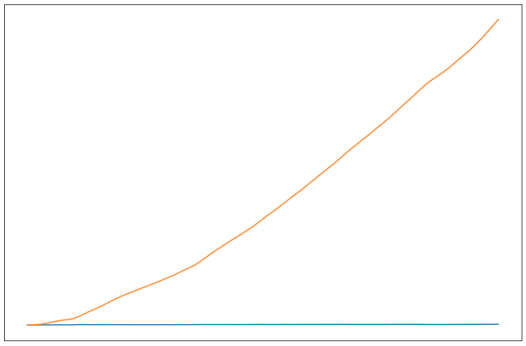
```
cumsum_churn.head()
```

Day	Revenue
1	0.000000
2	0.000793
3	0.826335
4	1.146667
5	2.229221

Plotting two-line graphs on the same axes:

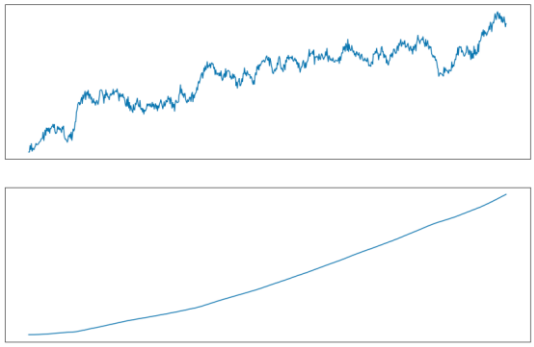
```
# Plotting two series on the same axes
plt.figure(figsize=(12,8))
plt.plot(teleco_df)
plt.plot(cumsum_churn)
plt.title('Time series Analysis')
plt.ylabel('Revenue')
plt.xlabel('Days')
```

```
Text(0.5, 0, 'Days')
```



Two plots in the same figure with subplots:

```
# 3. Two plot in the same figure with subplots
plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.plot(teleco_df)
plt.title('Time series')

plt.subplot(2,1,2)
plt.plot(cumsum_churn)
plt.title('cumsum Time series')
plt.tight_layout()
```



TIME STEP WITH MISSING MEASUREMENTS

A basic approach in time series analysis is to consider the observed series as a realization of the process. In the "teleco" data set, the data doesn't have any gaps in it. The data is continuously collected, and everyday steps are done to achieve the highest level of granularity possible. The sequence takes 731 days, or two years, to complete. Because the number for the rows corresponded to the days in the data set's time sequence, the original data set's integrity was preserved.

No anomalies are missing values observed in the teleco data set.

Time Series Stationarity:

The statistical features of a process that generates a time series do not change over time, which is known as stationarity. Stats properties in this case contain variance, mean and the autocorrelation.

Most analytics procedures in time series analysis require data to be stationary or at least the procedure itself, meaning the model function itself will make the data stationary during the model setup

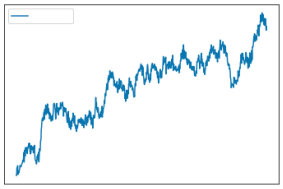
Lot of time series do have a trend in them, the mean changes over time because of trend, Predictions tend to be underestimated because of this. we are evaluating teleco data set using stationarity test based on the Augmented Dickey-Fuller test. Then we need to import the adfuller test from statsmodels.tsa.stattools

First step, we tell our function to perform the Dickey-Fuller test

```
# Test for stationarity

def stationarity_test(timeseries):
    """Augmented Dickey-Fuller test
    A test for stationarity"""
    from statsmodels.tsa.stattools import adfuller
    print("Result of Dickey-Fuller Test:")
    df_test = adfuller(timeseries, autolag = "AIC")
    df_output = pd.Series(df_test[0:4],index = ["Test statistic", "p-value", "Number of lags used", "Number of observations used"])
    print(df_output)
```

```
teleco_df.plot()
```

```
<AxesSubplot:xlabel='Day'>
```



Applying test on Teleco_df data set to see stationary or non-stationary

```
# Applying the test on three different datasets
# 1. The Teleco_df dataset
stationarity_test(teleco_df)
```

```
Result of Dickey-Fuller Test:
Test statistic                  -1.924612
p-value                          0.320573
Number of lags used              1.000000
Number of observations used    729.000000
dtype: float64
```

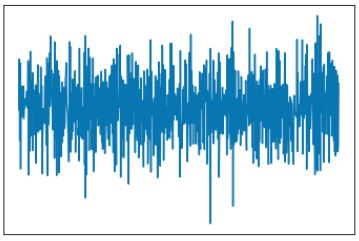
There is insufficient evidence to reject the null hypothesis because the p-value is higher than 0.03. As a result, the non-stationary nature of the time series can be deduced. This appears to be the case because the first third of the line graph shows a considerable increasing trend before leveling out after day 200.
Differentiation will be used to correct the non-stationarity. The differences between successive observations are used in this method.

Applying differencing on Teleco_df data set to convert non-stationary to stationary

```
# Differencing meaning y(t) = y(t) - y(t-1)
teleco_df['Revenue_diff'] = teleco_df['Revenue'] - teleco_df['Revenue'].shift(1)
```

```
teleco_df['Revenue_diff'].dropna().plot()
```

```
<AxesSubplot:xlabel='Day'>
```



Split of Training /Testing Sets

It is not advisable to employ sequential, random sampling. As a result, the data will be divided into two groups, with the training set using the first 60% of the time series and the testing set using the remaining 40%. Both sets of data, as well as the data that has been cleaned, will be saved to a file.

Applying split method to split the data set

```
# splitting method & metrics from sklearn
from sklearn.model_selection import train_test_split
```

```
X = new_df['Revenue_diff']
y = new_df['Revenue']
```

```
# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6)
```

```
new_df.to_csv('C:/Kailash/Rekha/D213/data/teleco_cleaned.csv')
X_train,y_train.to_csv('C:/Kailash/Rekha/D213/data/train.csv')
X_test,y_test.to_csv('C:/Kailash/Rekha/D213/data/test.csv')
```

We created the Cleaned dataset, Training data set and Test data set

```
new_df.to_csv('C:/Kailash/Rekha/D213/data/teleco_cleaned.csv')
X_train,y_train.to_csv('C:/Kailash/Rekha/D213/data/train.csv')
X_test,y_test.to_csv('C:/Kailash/Rekha/D213/data/test.csv')
```

ANALYSIS/IDENTIFICATION OF MODEL

Report annotated findings together with data visuals:

In this time-series analysis annotations are based the data sets to forecast the model, in the given data set is a trend as well as seasonal component, but after differencing, the trend appears to be insignificant. Decomposition reveals the presence of these substances:

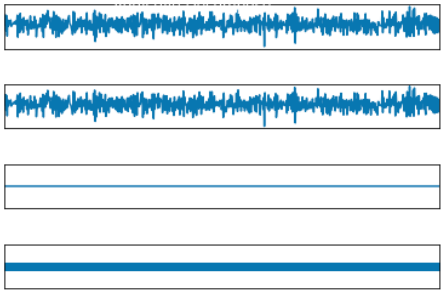
- Lack of Seasonal Component: When the trend increases at a certain interval, we see seasonal component. In our analysis we see Lack of seasonal component as trend increasing in an upward and there are no certain intervals in the trend. “Revenue” increasing based on increasing days but no in proper intervals due to lack of seasonal component.
- Trends: Trend is the general direction of our time series, if you look at teleco_df, the “Revenue” trend increasing as the number of days increasing.so there is not linear increase, it’s a kind of exponential increase. So, in our analysis the trend is exponential as it increases upward.so the time-series is non-stationarity in trend.
- Auto Correlation: It describes the correlation between the values of an ordered series at different time points. It is a key statistic as soon as we work with the sort of data, the previous observations influence the recent ones, so the steps on the time scale is lags. we can identify the autocorrelation present in our data set

or not by looking at trend and seasonality. Trend is an indicator of auto correlation, if the trend is upward there is observation at time point t will be higher than at the previous time point t minus 1.in our time series analysis we see upward trend which means auto correlation at different time points.

- Spectral density: It describes how the time series distributed with frequency. A periodogram is used to identify the frequencies of a time series. It will be helpful to identifying cyclical behavior in a series, especially when cycles are not related to the seasonality. In our analysis the data shows upward trend, not much seasonality.
- Decomposition: It's a technique that divides the time series multiple components, each component represents a trend and patterns or seasonality. Decomposition has two types additive and multiplicative. Addictive time series includes base level data, trend, seasonality, and Error whereas Multiplicative includes base level, trend, seasonality, and Error.

```
#Decomposition of addictive time series
from statsmodels.tsa.seasonal import seasonal_decompose

add_result = seasonal_decompose(new_df['Revenue_diff'],model='addictive',period=1)

plt.figure(figsize=(12,8))
add_result.plot().suptitle('\n Addictive Decompose', fontsize = 12)
```

```
Text(0.5, 0.98, '\n Addictive Decompose')

<Figure size 864x576 with 0 Axes>
```



The data does show some seasonality and trend, but not to a large extent. A model like ARIMA can capture this with the afc() function, autocorrelation.

ARIMA (Autoregressive Integrated Moving Average) model:

ARIMA stands for Autoregressive Integrated Moving Average and it specified by these 3 order parameters: (p,d,q). The process of fitting an ARIMA model is sometimes referred to as the Box-Jenkins method. AR component is represented by parameter p and I component represented by d and MA component represented by q. An auto regressive (AR(p)) component referring to the use of past values in the regression equation for the series Y, the auto regressive parameter p specifies the number of lags used in the model. The d represents the degree of differencing in the integrated(I(d)) component. Differencing involves simply subtracting its current and previous values d times. Differencing is used to stabilize the series when the stationarity assumption is not met. A moving average (MA(q)) components represents the error of the model as a combination of previous error terms et. The order q determines the number of terms to include in the model.

To evaluate the quality of our model will use (AIC) Akaike Information criterion and (BIC)Bayesian Information criterion Below are the ARIMA code and graphics.

```
#ARIMA model
import numpy as np
import pandas as pd
%matplotlib inline

#Load specific forecasting tools
from statsmodels.tsa.arima_model import ARMA,ARMAResults,ARIMA,ARIMAResults
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from pmdarima import auto_arima

#Loading non-stationary data set
new_df = pd.read_csv('C:/Kailash/Rekha/D213/data/teleco_time_series.csv',index_col='Day')
new_df.index.freq ='Revenue'
```

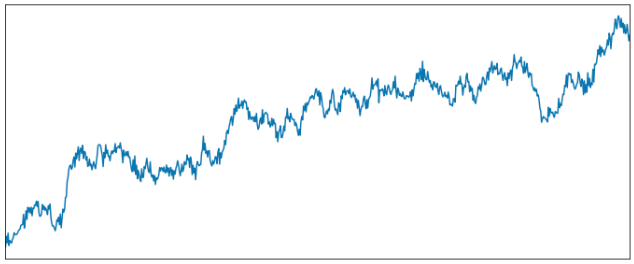
```
new_df.head()
```

Day	Revenue
1	0.000000
2	0.000793
3	0.825542
4	0.320332
5	1.082554

```
new_df.tail()
```

Day	Revenue
727	16.931559
728	17.490666
729	16.803638
730	16.194813
731	16.620798

```
new_df['Revenue'].plot(figsize=(12,5)).autoscale(axis = 'x',tight = True)
```



```
#pmdarima Auto-ARIMA
!pip install pmdarima
from pmdarima import auto_arima

#Ignore harmless warnings
import warnings
warnings.filterwarnings("ignore")
```

```
Stepwise_fit = auto_arima(new_df['Revenue'], start_p=1,start_q=1,
                          max_p=3,max_q=3,m=12,
                          start_P=0,seasonal=True,
                          d=None,D=1,trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)
Stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=1.89 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2367.159, Time=0.03 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=1419.537, Time=0.52 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=1969.738, Time=0.38 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2399.547, Time=0.03 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=1568.311, Time=0.17 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=1320.755, Time=1.68 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=6.25 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.71 sec
 ARIMA(0,0,0)(2,1,0)[12] intercept   : AIC=2339.965, Time=0.91 sec
 ARIMA(2,0,0)(2,1,0)[12] intercept   : AIC=1147.041, Time=2.94 sec
 ARIMA(2,0,0)(1,1,0)[12] intercept   : AIC=1256.245, Time=0.86 sec
 ARIMA(2,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=6.59 sec
 ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.69 sec
 ARIMA(3,0,0)(2,1,0)[12] intercept   : AIC=1148.348, Time=3.23 sec
 ARIMA(2,0,1)(2,1,0)[12] intercept   : AIC=1148.544, Time=3.29 sec
 ARIMA(1,0,1)(2,1,0)[12] intercept   : AIC=1195.703, Time=2.62 sec
 ARIMA(3,0,1)(2,1,0)[12] intercept   : AIC=1132.917, Time=8.64 sec
 ARIMA(3,0,1)(1,1,0)[12] intercept   : AIC=1235.930, Time=2.88 sec
 ARIMA(3,0,1)(2,1,1)[12] intercept   : AIC=inf, Time=8.03 sec
 ARIMA(3,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=2.95 sec
 ARIMA(3,0,2)(2,1,0)[12] intercept   : AIC=1132.894, Time=8.40 sec
 ARIMA(3,0,2)(1,1,0)[12] intercept   : AIC=1236.059, Time=3.36 sec
 ARIMA(3,0,2)(2,1,1)[12] intercept   : AIC=inf, Time=9.93 sec
 ARIMA(3,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=4.05 sec
 ARIMA(2,0,2)(2,1,0)[12] intercept   : AIC=1142.858, Time=3.86 sec
 ARIMA(3,0,3)(2,1,0)[12] intercept   : AIC=1136.533, Time=11.08 sec
 ARIMA(2,0,3)(2,1,0)[12] intercept   : AIC=1138.666, Time=8.55 sec
 ARIMA(3,0,2)(2,1,0)[12]             : AIC=1135.369, Time=2.89 sec

Best model:  ARIMA(3,0,2)(2,1,0)[12] intercept
Total fit time: 109.459 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(3, 0, 2)x(2, 1, [], 12)	**Log Likelihood**	-557.447
Date:	Thu, 21 Apr 2022	**AIC**	1132.894
Time:	16:47:01	**BIC**	1174.095
Sample:	0	**HQIC**	1148.801
	- 731		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0062	0.004	1.727	0.084	-0.001	0.013
ar.L1	1.4301	0.077	18.490	0.000	1.278	1.582
ar.L2	-0.0278	0.127	-0.219	0.827	-0.276	0.221
ar.L3	-0.4135	0.073	-5.702	0.000	-0.556	-0.271
ma.L1	-0.9605	0.086	-11.138	0.000	-1.130	-0.791
ma.L2	0.0972	0.079	1.226	0.220	-0.058	0.253
ar.S.L12	-0.7078	0.038	-18.445	0.000	-0.783	-0.633
ar.S.L24	-0.3826	0.039	-9.729	0.000	-0.460	-0.305
sigma2	0.2719	0.015	17.663	0.000	0.242	0.302

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.89
Prob(Q):	0.98	**Prob(JB):**	0.39
Heteroskedasticity (H):	1.06	**Skew:**	0.01
Prob(H) (two-sided):	0.63	**Kurtosis:**	2.75

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

ACF and PCF are generated from differenced Time series

```
from statsmodels.tsa.stattools import acf,pacf
acf(new_df['Revenue'])
```

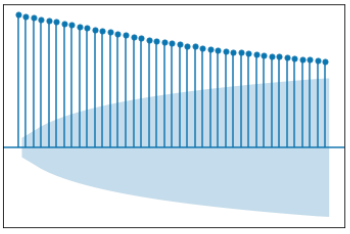
```
array([1.        , 0.98375067, 0.97679376, 0.96584953, 0.95545834,
       0.94515237, 0.93469998, 0.92356074, 0.91259674, 0.9007179 ,
       0.88986482, 0.87820542, 0.86783127, 0.85669648, 0.84568014,
       0.83433728, 0.82369773, 0.81266219, 0.80261913, 0.79240124,
       0.78364098, 0.77521764, 0.76563762, 0.75985141, 0.75066964,
       0.742678  , 0.73537013, 0.72746968, 0.72092685, 0.71435789,
       0.70827958, 0.70240237, 0.69647315, 0.69006167, 0.68432411,
       0.67889965, 0.6730078 , 0.66649597, 0.66008687, 0.65416797,
       0.64929254])
```

```
pacf(new_df['Revenue'])
```

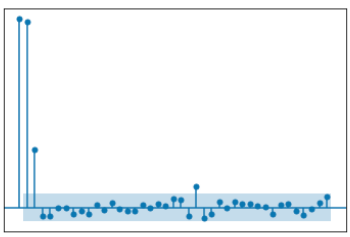
```
array([ 1.        ,  0.98509827,  0.30610361, -0.04616345, -0.04390678,
       -0.00207786, -0.00482952, -0.03490847, -0.01623674, -0.03656166,
        0.01388844, -0.01405953,  0.02666325, -0.00859463, -0.01686665,
       -0.01933449,  0.01418875, -0.00512425,  0.02007584,  0.00773943,
        0.04540836,  0.04006931, -0.04430379,  0.10987468, -0.05488396,
       -0.0364612 ,  0.03075863, -0.00279951,  0.02882068,  0.01883929,
        0.01886735,  0.00836569,  0.0036029 , -0.03488273,  0.01402038,
        0.01890911, -0.01835112, -0.03911919, -0.0086121 ,  0.02305523,
        0.05929618])
```

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

```
title ='Autocorrelation:Teleco'
lags = 40
plot_acf(new_df['Revenue'],title=title,lags=lags);
```



```
plot_pacf(new_df['Revenue'],title='Partial Autocorrelation:Teleco',lags=40);
```



When looking for a strong seasonal trend in the plots, none is found. To find the proper arima model parameters, the ACF and PACF are examined. Because the ACF fades and the PACF disappears, an AR (1) model is created, and residual analysis is carried out:

FORECAST USING ARIMA MODEL

Predict the values of test set than will compare the values with prediction, what is the error in each prediction.to predict values will use "Predict" function which contains start index of our test set, end index contains end of test record, type is linear, linear type predict the values. After that will rename the values

The predict () function used to create a forecast with testing and training sets.

```
#Set 60% of training set
train = new_df.iloc[:403]
test = new_df.iloc[403:]
```

```
from statsmodels.tsa.arima_model import ARMA,ARMAResults,ARIMA,ARIMAResults
```

```
model = ARIMA(train['Revenue'],order=(1,1,0))
results = model.fit()
results.summary()
```

ARIMA Model Results

Dep. Variable:	D.Revenue	**No. Observations:**	402
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-267.262
Method:	css-mle	**S.D. of innovations**	0.470
Date:	Fri, 22 Apr 2022	**AIC**	540.525
Time:	15:27:10	**BIC**	552.514
Sample:	1	**HQIC**	545.272

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0302	0.016	1.880	0.060	-0.001	0.062
ar.L1.D.Revenue	-0.4611	0.044	-10.441	0.000	-0.548	-0.375

Roots

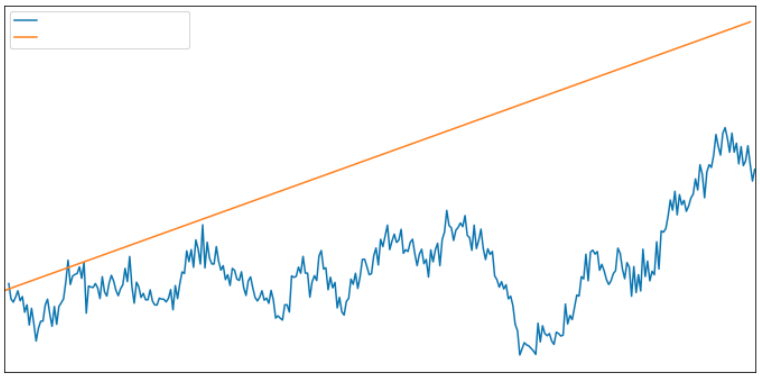
	Real	Imaginary	Modulus	Frequency
AR.1	-2.1688	+0.0000j	2.1688	0.5000

```
# Obtain Predicted Values

start = len(train)
end = len(train) + len(test)-1
predictions = results.predict(start=start,end=end,dynamic=False,typ='levels').rename('ARIMA(1,1,0) predictions')
```

```
# compare prediction to expected values
for i in predictions:
    for j in test['Revenue']:
        print(f"predicted = {i}, expected = {j} ")
```

```
predicted = 12.202312620701385, expected = 12.42338133
predicted = 12.202312620701385, expected = 11.85932698
predicted = 12.202312620701385, expected = 11.73912658
predicted = 12.202312620701385, expected = 11.92291
predicted = 12.202312620701385, expected = 12.157616
predicted = 12.202312620701385, expected = 11.79346872
predicted = 12.202312620701385, expected = 11.9430778
predicted = 12.202312620701385, expected = 11.36953639
predicted = 12.202312620701385, expected = 11.64293102
predicted = 12.202312620701385, expected = 10.89965094
predicted = 12.202312620701385, expected = 11.51358095
predicted = 12.202312620701385, expected = 10.95912682
predicted = 12.202312620701385, expected = 10.31231737
predicted = 12.202312620701385, expected = 10.77773788
predicted = 12.202312620701385, expected = 11.03581856
predicted = 12.202312620701385, expected = 11.04543669
predicted = 12.202312620701385, expected = 11.63735871
predicted = 12.202312620701385, expected = 11.84817774
predicted = 12.202312620701385, expected = 11.30317125
```

```
# plot predictions aganist the known values

ax = test['Revenue'].plot(legend=True,figsize=(12,6))

predictions.plot(legend=True)

ax.autoscale(axis='x',tight=True)
```



```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(test['Revenue'],predictions)
print(f'ARIMA(1,1,0) MSE Error: {error:11.10}')
```

```
ARIMA(1,1,0) MSE Error: 22.79172739
```

```
from statsmodels.tools.eval_measures import rmse

error = rmse(test['Revenue'],predictions)
print(f'ARIMA(1,1,1) RMSE Error: {error:11.10}')
```

```
ARIMA(1,1,1) RMSE Error: 4.774068223
```

```
#Retrain the model on the full data and forecast the future
model = ARIMA(new_df['Revenue'],order=(1,1,0))
results = model.fit()

fcast = results.predict(len(new_df),len(new_df)+60*10,typ='levels').rename('ARIMA(1,1,0) Forecast')
```

```
# plot predictions aganist known values

ax = new_df['Revenue'].plot(legend=True,figsize=(12,6))
fcast.plot(legend=True)
ax.autoscale(axis='x',tight=True)
```

Do the following to describe your data analysis:

ARIMA Model

The model's final goal is to forecast future time series movement by studying disparities between series values rather than actual values. When data displays signs of non-stationarity, ARIMA models are used. Non-stationary data is always converted to stationary data in time series analysis.

The trend and seasonal components are the most common sources of non-stationary data in time series. The differencing step can be used to convert non-stationary data to stationary data. To get rid of the trend component in the data, you can use one or more times of differencing steps.

We can break down the model into smaller components based on the name:

- AR: an AutoregRegressive
- MA: a Moving Average
- I: Integrated

AR: an Autoregressive

This model represent a random process. The model output is linearly reliant on its own previous value. Which can be a set of lagged data points or a set of historical observations.

MA: a Moving Average

a Moving Average model whose output is linearly dependent on current and previous observations of a stochastic factor

I: Integrated

The differencing phase used to construct stationary time series data, i.e. removing seasonal and trend components, is referred to as integrated here.

There is a trend and seasonal component in our dataset, we differenced the data, although after differencing, the trend appears to be insignificant. Decomposition reveals the presence of these substances in our data set. To find the proper arima model parameters, the ACF and PACF are examined and at lags 1, 2, and 6, autocorrelation may be significant. To see if there's any evidence of non-zero connection at these time lags. The periodogram depicts the density of the spectrum at a specific place. Lower frequencies tend to have the highest density. The residuals appear to be random, and the autocorrelation appears to be minor. The “FORECAST” we used predict () function to forecast with testing and training sets.

After determining the ARIMA model, seasonality and trends are present with order p, d, and q of 1, 0, 0 and seasonal coefficients of 1, 1, 0 was adopted. Because of the lack of stationarity in the data, differencing was required even after cleaning. For the selected ARIMA model, this combination resulted in an AIC. The forecast length is sufficient for planning; but, when the forecast is extended beyond 90 days, the confidence interval widens significantly, making prediction difficult.

The chosen model is used to generate a daily interval forecast with +/- 1 and 2 prediction error boundaries to compare performance to retained test data. The line represents test data, implying that the model is consistent with it. The error is computed.

FORECAST VISUALIZATION

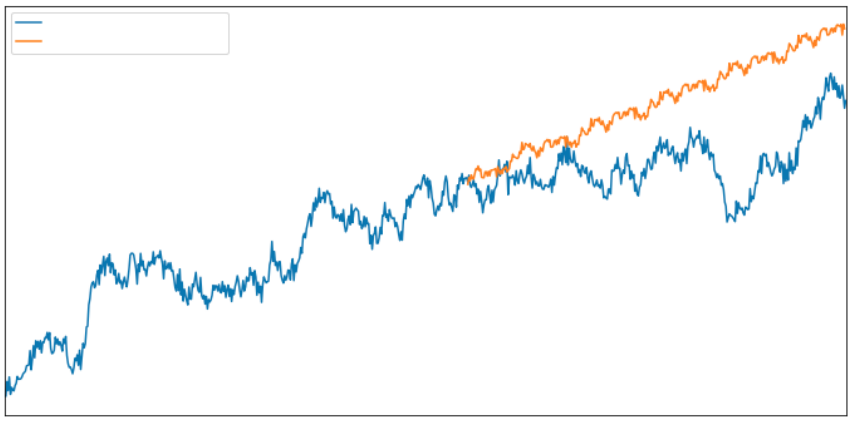
These are the predictions for the ARIMA model

```
# obtain predicted values
start = len(train)
end = len(train) + len(test)-1
predictions = results.predict(start=start,end=end,dynamic=False,typ='levels').rename('SARIMAX(1,1,0) predictions'
```

```
for i in predictions:
    for j in test['Revenue']:
        print(f"predicted = {i}, expected = {j} ")
```

```
predicted = 11.917428045113075, expected = 12.42338133
predicted = 11.917428045113075, expected = 11.85932698
predicted = 11.917428045113075, expected = 11.73912658
predicted = 11.917428045113075, expected = 11.92291
predicted = 11.917428045113075, expected = 12.157616
predicted = 11.917428045113075, expected = 11.79346872
predicted = 11.917428045113075, expected = 11.9430778
predicted = 11.917428045113075, expected = 11.36953639
predicted = 11.917428045113075, expected = 11.64293102
predicted = 11.917428045113075, expected = 10.89965094
predicted = 11.917428045113075, expected = 11.51358095
predicted = 11.917428045113075, expected = 10.95912682
predicted = 11.917428045113075, expected = 10.31231737
predicted = 11.917428045113075, expected = 10.77773788
predicted = 11.917428045113075, expected = 11.03581856
predicted = 11.917428045113075, expected = 11.04543669
predicted = 11.917428045113075, expected = 11.63735871
predicted = 11.917428045113075, expected = 11.84817774
predicted = 11.917428045113075, expected = 11.30317125
```

```
# plot predictions aganist known values

ax = new_df['Revenue'].plot(legend=True,figsize=(12,6))
predictions.plot(legend=True)
ax.autoscale(axis='x',tight=True)
```



```
from sklearn.metrics import mean_squared_error

error = mean_squared_error(test['Revenue'],predictions)
print(f'SARIMAX(1,1,0) MSE Error: {error}')
```

```
SARIMAX(1,1,0) MSE Error: 16.963412002751568
```

```
from statsmodels.tools.eval_measures import rmse

error = rmse(test['Revenue'],predictions)
print(f'ARIMA(1,1,1) RMSE Error: {error:11.10}')
```

```
ARIMA(1,1,1) RMSE Error: 4.118666289
```

PLAN OF ACTION

The model that was created was unable to achieve a high level of accuracy. This could indicate that the data has a substantial amount of unpredictability. However, there were two separate trends in the data, one from the first year and the other from the second. If the change in churn was linked to revenue, it could reveal information about the customers churn. More data beyond the second year could result in a more accurate model.

CONCLUSION

In conclusion, understanding customers plays a vital role in customer relationship management and directly impacts a company's long-term success. By gaining a deeper understanding of customer traits, companies can more effectively target their promotions and advertising campaigns, leading to increased long-term earnings.

Our investigation for a telecommunications business aimed to uncover hidden patterns in customer data through time-series analysis. This research has successfully identified critical revenue trends, providing valuable insights for more accurate forecasting. These findings equip stakeholders with a clearer understanding of the organization's revenue trends, enabling more informed and strategic decision-making. This enhanced understanding ultimately supports the company's goal of achieving sustained long-term success.

REFERENCES

[1]. Author: Massaron, L. & Boschetti, A. Date: 2016; Title: Regression Analysis with Python. Packt Publishing.

[2]. Author: CBTNuggets Date: September 20th,2018.; Title: Why Data Scientists Love Python https://www.cbtnuggets.com/blog/technology/data/why-data-scientists-love-python