greatlearning
Power Ahead

A Short Monograph on Time Series Forecasting

To serve as a Refresher for PGP-DSBA

Index

Contents

List of Figures

List of Tables

1. Introduction

Forecast is a statistical method to predict an attribute using historical patterns in the data. All businesses rely on forecast and reap major benefits from accurate forecasted values. Forecasts can be made for several years in advance (e.g., capital investments, strategic planning) or for the next few minutes (for telecommunication routing). Every business and organization applies different methods of forecasting in different situations. Therefore, it is imperative to identify what to forecast and which method of forecasting should be utilized in different scenarios so that the risk of forecasting is minimized.

Data can be classified into three major groups through their temporal nature, such as:

I. ***Cross sectional data***: Data is collected at a single point in time on one or more variables. Here, data is not sequential and usually, data points are independent of one another. Regression, random forest or neural network methods have been applied widely on these data.

II. ***Time series data***: Univariate or multivariate data is observed across time in a sequential manner at pre-determined and equally-spaced time intervals (such as yearly, monthly, quarterly or hourly). Ordering among data points is important and cannot be destroyed.

III. ***Combination of cross sectional and time series data***: This is a complex study design where information on same variables are collected over various points of time. Many survey samplings make use of panel data.

In this monograph we have focused only on the forecasting methods appropriate for time series data observed at regular intervals.

Prediction for cross-sectional data has been the topic of discussion for other predictive models such as linear regression, logistic regression, CART, RF, ANN etc.

2. What is a time series?

2.1. Definition of Time series

Formal definition of time series: A collection of observations that has been observed at regular time intervals for a certain variable over a given duration is called a time series.

The regular time intervals can be ***daily*** (stock market prices), ***weekly*** (product manufactured), ***monthly*** (unemployment rate), ***quarterly*** (sales of product), ***annual*** (GDP), ***quinquennial***, that is in every 5 years (Census of manufactures), or ***decennial*** (census of population).

Time series can be applied in various fields such as economic forecasting, sales forecasting, budgetary analysis, stock market analysis, yield projection, inventory studies, workload projections, utility studies, census analysis, process and quality control and many more.

Time series data has several characteristics that make it unique. These characteristics can be stated below as: -

- **All observations are dependent**: In time series data, each observation is expected to depend on the past observations.
- **Missing data must be imputed**: Because all data points are sequential in time series, if any data point is missing, it must be imputed before the actual analysis process commences; otherwise the proper ordering is not preserved.
- **Two different types of intervals cannot be mixed**: Time series data is observed on the same variable over a given period of time with fixed and regular time intervals. Though data can be collected at various intervals such as yearly, monthly, weekly, daily, hourly (e.g. temperature) and/or any specific time-interval, the interval must remain the same throughout the entire range; e.g. yearly series cannot be combined with quarterly or monthly series.

Objective of Time Series Forecasting: Time series forecasting is applied to extract information from historical series and is used to predict future behaviour of the same series based on past pattern.

Approaches used for Time Series Forecasting: The following are two major approaches to time series forecasting.

I. **Decomposition**: This method is based on extraction of individual components of time series.
II. **Regression**: This method is based on regression on past observations

The two approaches are completely different. Both approaches are discussed with illustration.

By no means these are the only two approaches of time series forecasting. A few other methods are referred to in Section 7.

3. Decomposition Method

3.1. Components of Time Series

There are various forces that may affect the observations in a time series. The three important components are:

I. **Trend** (Long term movement)
II. **Seasonal component**: Intra-year stable fluctuations repeatable over the entire length of the series
III. **Irregular component** (Random movements)

Trend (T_t): When the series increases (or decreases) over the entire length of time. For example, the price of a share may increase or decrease linearly over a period of time., the sales of a new product may increase exponentially (or non-linearly). Figure 1 shows increasing linear trend of US GDP growth over a period of time.

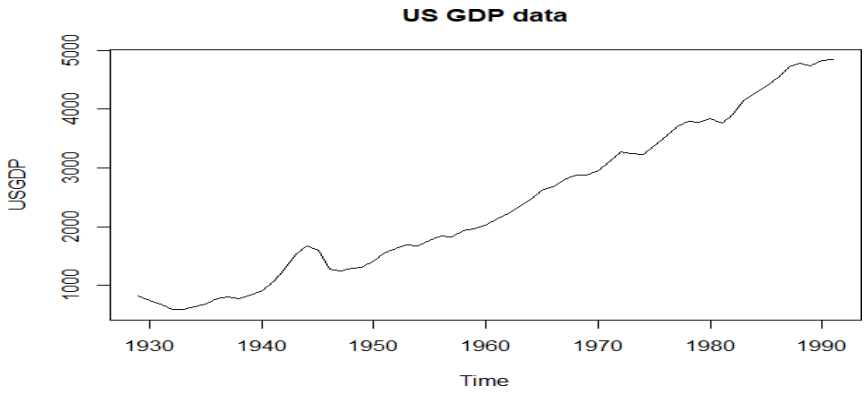


Figure 1: US GDP growth

Seasonality (S_t): When a series is observed with more frequently than a year (quarterly or monthly for example), the series is subject to rhythmic fluctuations which are stable and repeatable each year. For example, sales of umbrella increase in rainy season whereas sales of AC increase in summers and sales of woolen clothes increase in winters. This intra-year fluctuation is known as seasonal fluctuation. Figure 2 contains monthly average temperature that oscillates in a regular pattern over the given period of time.

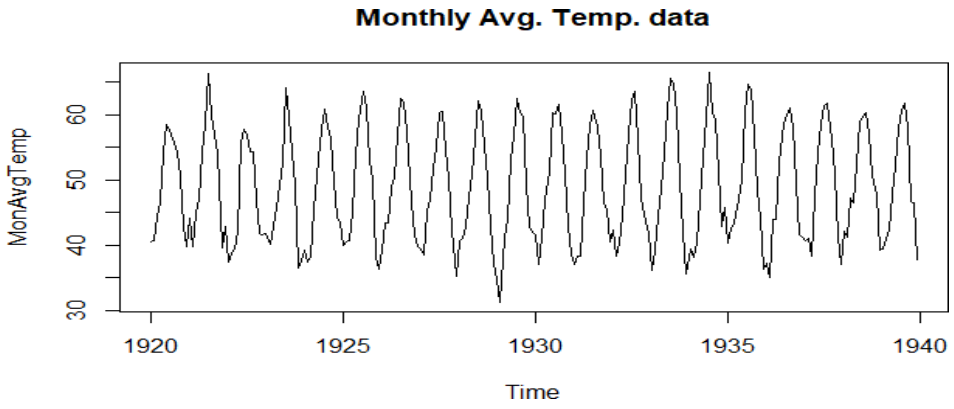


Figure 2: Monthly average temperature data

Irregularity (I_t): These fluctuations are purely random, erratic, unforeseen, and unpredictable. This is the random component of time series.

Trend and seasonal components are part of systematic components of time series.

Additive Model: $Y_t = T_t + S_t + I_t$ is considered when the resultant series is the sum of the components.

Multiplicative Model: $Y_t = T_t * S_t * I_t$ is considered when the resultant time series is the product of the components.

A series may be considered multiplicative series when the seasonal fluctuations increase as trend increases. A multiplicative time series can be transformed into an additive series by taking log transformation i.e.

$$\log(Y_t) = \log(T_t) + \log(S_t) + \log(I_t)$$

Decomposition of a time series leads to identification and extraction of the individual components.

Primary objective of decomposition is to study the components of the time series, NOT forecasting. However, forecasting models can be built on top of the decomposed series.

Case Study 1

A company ABC selling tractors has to forecast its sales for the next 24 months. It has 12 years of past sales data on monthly basis. The data may contain trend, seasonality or both. Objective is to provide a reasonable forecast for future sales.

The following steps are to be followed:

Part.I. Checking the pattern of the tractor sales series.
Part.II. Identification of the components of the series
Part.III. Proposing best model for the series

Solution:

Part I: To check the pattern of the Tractor sale series


```
## Import all the required libraries

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from pylab import rcParams
from statsmodels.graphics.tsaplots import month_plot,plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose,STL
from statsmodels.tsa.api import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
```

```
## Upload the data file and plot the same

df = pd.read_csv('Tractor-Sales.csv')
timestamp = pd.date_range(start='2003-01-01',end='2015-01-01',freq ='M')
df['Time_Stamp'] = timestamp
df.drop(labels='Month-Year',axis=1,inplace=True)
df.set_index(keys='Time_Stamp',drop=True,inplace=True)
rcParams['figure.figsize'] = 15,8
df.plot(grid=True);
```

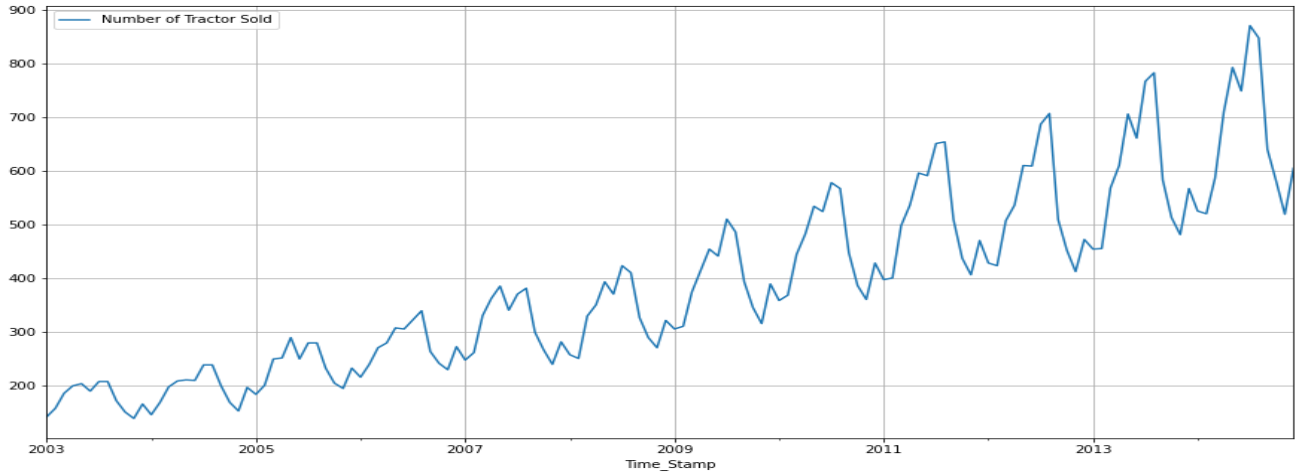


Figure 3: Tractor sales data

Following are the observations obtained from Figure 3.:

I. Data values are stored in correct time order and no data is missing.
II. The sales are increasing in numbers, implying presence of trend component.
III. Intra-year stable fluctuations are indicative of seasonal component. As trend increases, fluctuations are also increasing. This is indicative of multiplicative seasonality.

- **Note:**

All the versions of the libraries used are given in the appendix of the monograph given in page 57.

Following two plots will help us in identifying the seasonal fluctuations better.

```
## Plot 1: Seasonal plot Year-wise
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'Au
gust', 'September', 'October','November', 'December']

yearly_sales_across_years = pd.pivot_table(df, values = 'Number of Tractor So
ld', columns = df.index.year,index = df.index.month_name())

yearly_sales_across_years = yearly_sales_across_years.reindex(index = months)

yearly_sales_across_years.plot()
plt.grid()
plt.legend(loc='best');
```

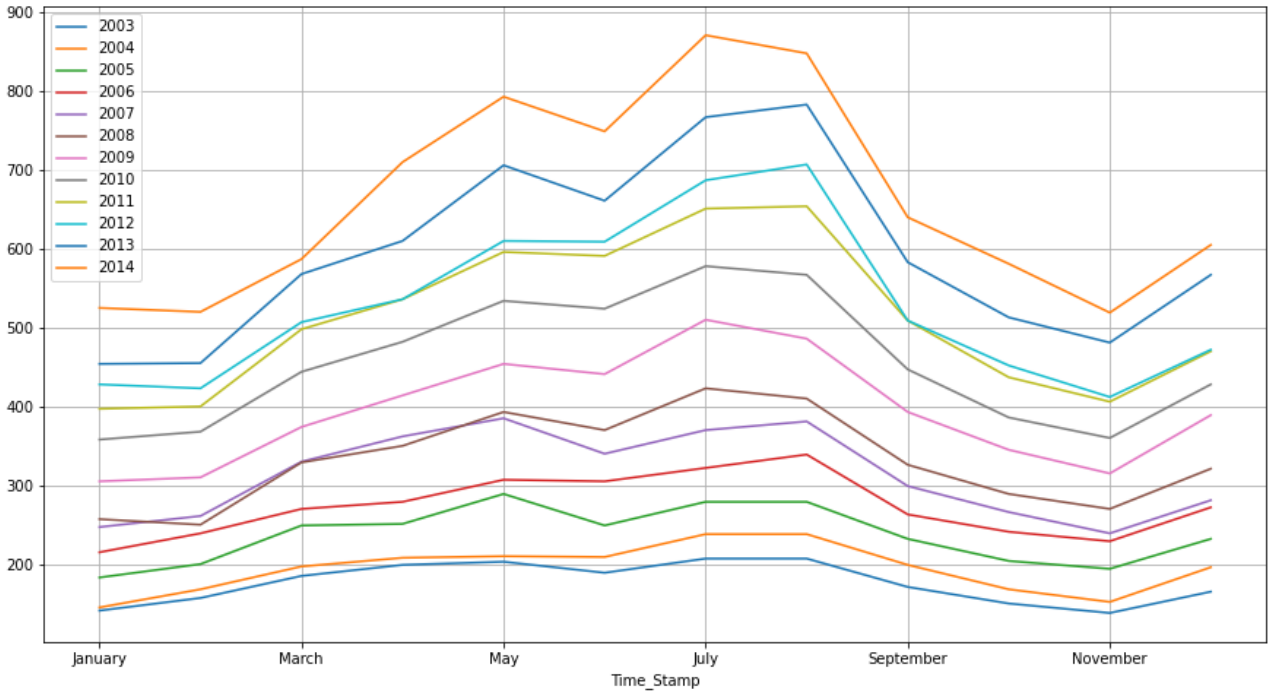


Figure 4: Seasonal plot year-month wise

```
## Plot 2: Seasonal plot Month-wise

month_plot(df,ylabel='TractorSalesTS')
plt.grid();
```

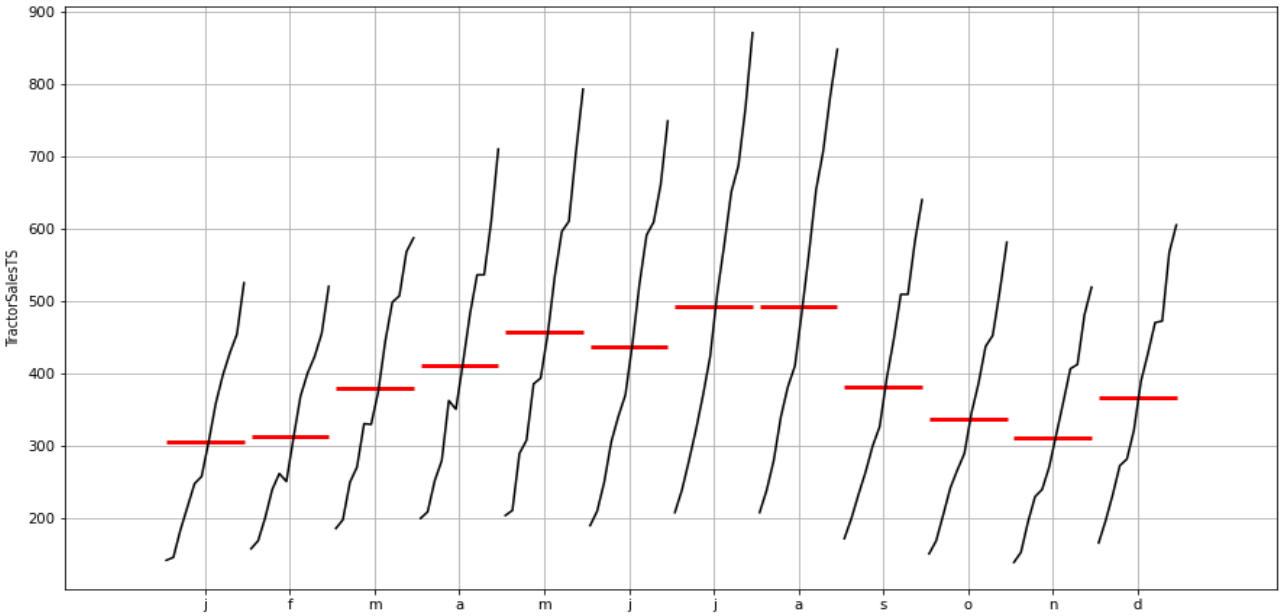



Figure 5: Seasonal plot month-wise

Figure 4 shows that the sales of tractors are increasing every year in number.
Note in Figure 5 that, the vertical lines represent monthly sales and the horizontal lines represent average sales of the given month. Here, it can be observed that average sales are higher in July and August as compared to other months
In all these above plots the increasing lines that represent sales have seasonal fluctuations along with a trend. Thus, we can confirm that it is multiplicative seasonality.

Part II: To Identify the components of the given Tractor sales data.

Now decomposition method is applied to identify and separate out the three components (i.e. trend, seasonality and irregular components) from the given series to observe their independent properties.

3.2. Two alternative methods for decomposition

3.2.1. Using moving average decomposition

Case Study continued.

```
## Decomposition of the TS using the moving average decomposition


decomposition = seasonal_decompose(df['Number of Tractor Sold'],
model='multiplicative')
decomposition.plot();
```

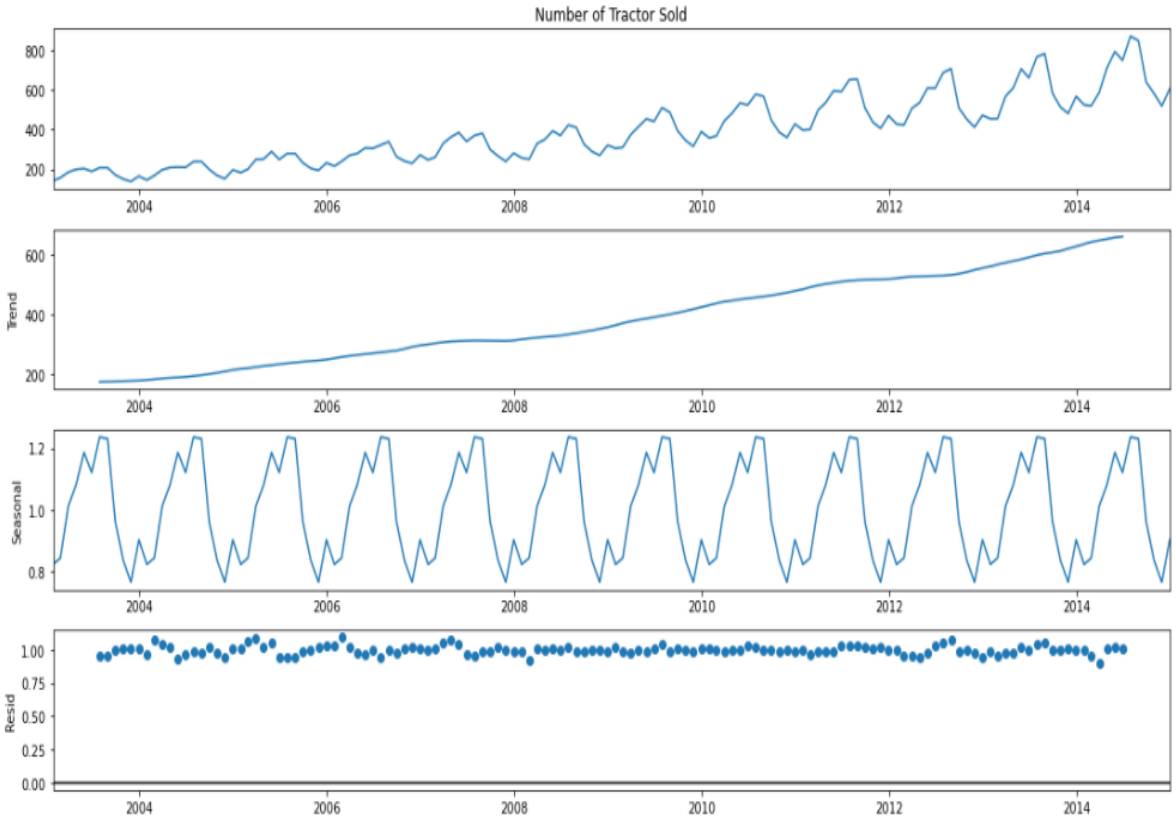


Figure 6: Decomposed tractor time series into components using moving average decomposition


```
## Observing the seasonal indices

Seasonal_Ind = pd.DataFrame({'Jan':round(decomposition.seasonal.head(12),2).values[0],
                             'Feb':round(decomposition.seasonal.head(12),2).values[1],
                             'Mar':round(decomposition.seasonal.head(12),2).values[2],
                             'Apr':round(decomposition.seasonal.head(12),2).values[3],
                             'May':round(decomposition.seasonal.head(12),2).values[4],
                             'Jun':round(decomposition.seasonal.head(12),2).values[5],
                             'Jul':round(decomposition.seasonal.head(12),2).values[6],
                             'Aug':round(decomposition.seasonal.head(12),2).values[7],
                             'Sep':round(decomposition.seasonal.head(12),2).values[8],
                             'Oct':round(decomposition.seasonal.head(12),2).values[9],
                             'Nov':round(decomposition.seasonal.head(12),2).values[10],
                             'Dec':round(decomposition.seasonal.head(12),2).values[11]},
                                index=range(1,2))
Seasonal_Ind
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	0.82	0.84	1.01	1.08	1.19	1.12	1.24	1.23	0.96	0.84	0.77	0.9

Figure 6 indicates that trend is increasing linearly. Since this is monthly data, there are 12 seasonal indices. Sum of the monthly indices must be 12. In July tractor sales is the highest among all months in the same year, as borne by the highest value of the seasonal component whereas in November (lowest value of the seasonality) tractor sales is the lowest.

3.2.2. Using Seasonal and Trend decomposition by Loess

Owing to some limitations in the moving average decomposition, Loess decomposition has been proposed. This is more versatile but does not admit multiplicative seasonality. Hence log transformation is used to convert multiplicative seasonality into additive seasonality.

```
## Decomposition of the TS using Loess Trend and Seasonality decomposition

decomposition = STL(np.log10(df['Number of Tractor Sold']),
                    seasonal=13).fit()
decomposition.plot();
```

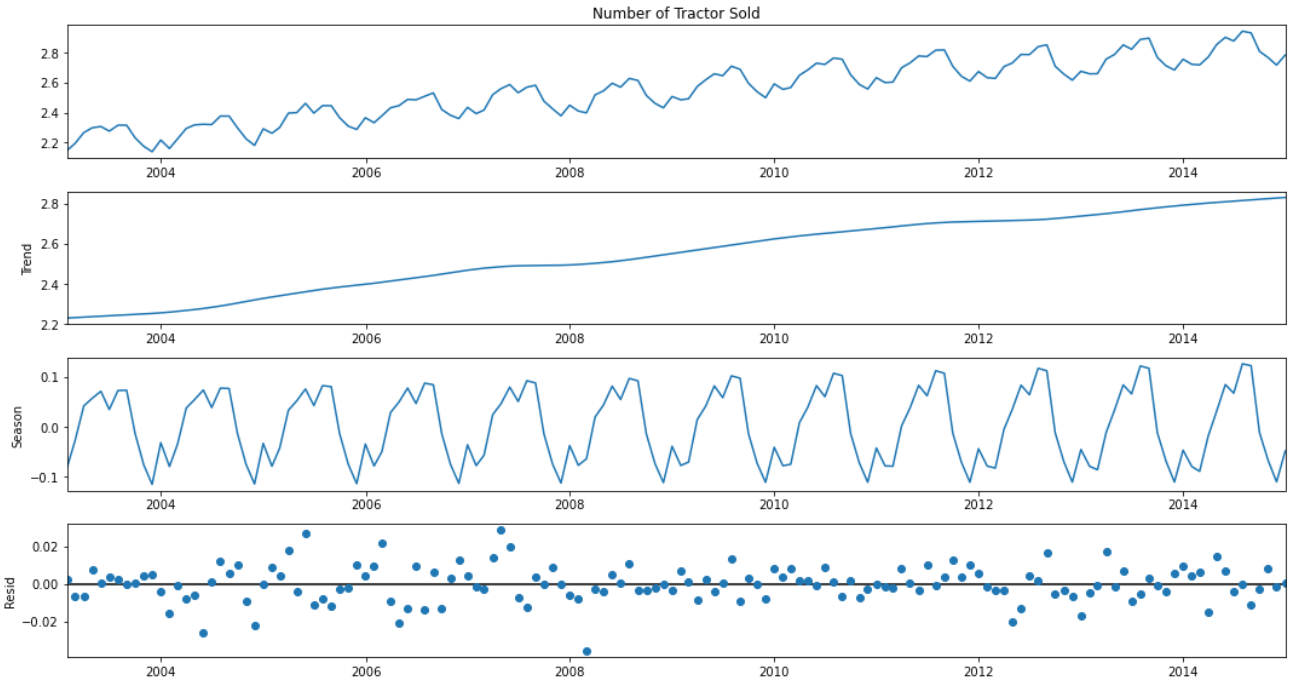



Figure 7: Decomposed tractor time series into components using Loess decomposition

4. Data split and its significance

Before a forecast method is proposed, the method needs to be validated. For that purpose, data has to be split into two sets i.e. training and testing. Training data helps in identifying and fitting right model(s) and test data is used to validate the same.

In case of time series data, the test data is the most recent part of the series so that the ordering in the data is preserved.

Part III: To Propose best model for the Tractor sales data

4.1. Measures of Forecast Accuracy

Forecasting accuracy measures compare the predicted values against the observed values to quantify the predictive power of the proposed model. Mathematically, it can be defined as

Forecast error e_t for period t is given by: $e_t = \hat{Y}_t - Y_t$
Where
$\hat{Y}_t$ = forecast value for time period t
Y_t = actual value in time period t
n = No. of observations

Various measures of forecasting errors can be defined as: -

Mean Absolute Deviation (MAD): $\frac{1}{n}\sum_{t=1}^{n}|e_t|$

Mean Absolute Percentage Error (MAPE): This method is used extensively in time series because it acts as a unit free criteria; therefore, performance of forecasted values can be easily compared. $MAPE=\frac{1}{n}\sum_{t=1}^{n}\frac{|e_t|}{Y_t} * 100$

MAPE is usually expressed as a percentage.

Mean Square Error (MSE): $MSE=\frac{1}{n}\sum_{t=1}^{n}e_t^2$

Root Mean Square Error (RMSE): $RMSE=\sqrt{\frac{1}{n}\sum_{t=1}^{n}e_t^2}$

Case Study continued.

For Tractor Sales series, the first 10 years of data is used for training purpose and last 2 years of data is for testing purpose.

```
## Step 4: Splitting data into training and test data sets

TS_Train = df[df.index.year <= 2012]
TS_Test = df[df.index.year > 2012]
```

```
TS_Train['Number of Tractor Sold'].plot()
TS_Test['Number of Tractor Sold'].plot()
plt.grid()
plt.title('Tractor Sales Traning and Test data')
plt.xlabel('Year')
plt.ylabel('Sales')
plt.legend(['Training Data','Test Data'],title='Forecast');
```

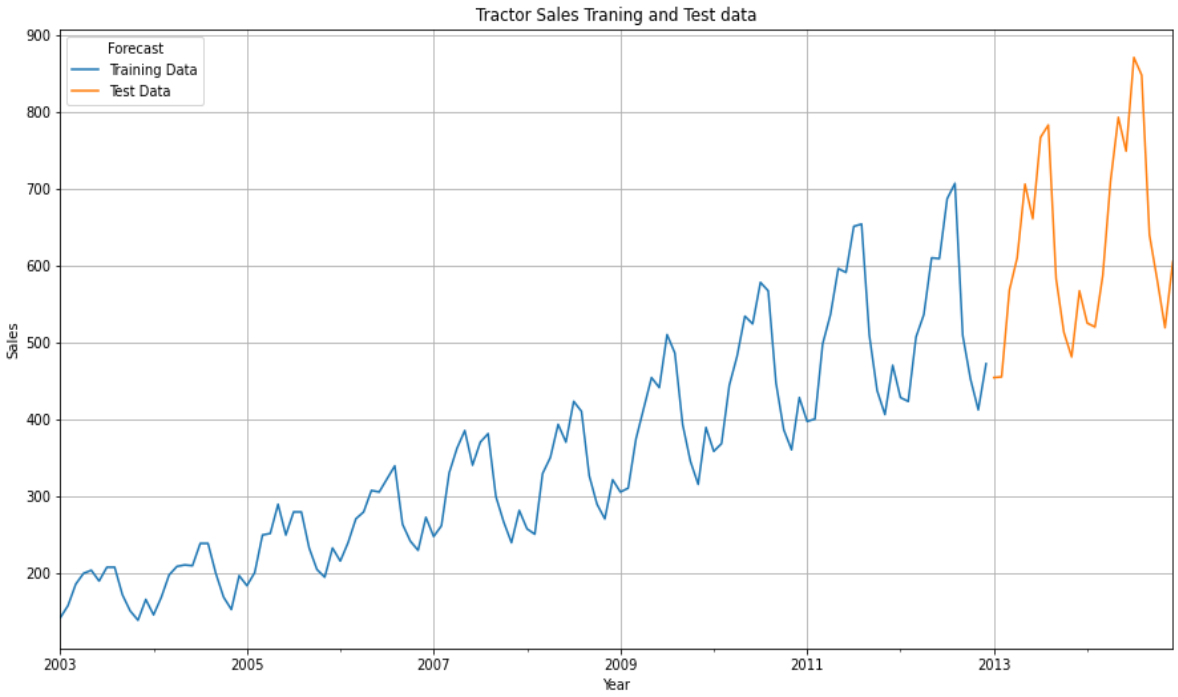



Figure 8: Data (split into Training and testing purpose)

5. Forecasting Methods

5.1. Exponential Smoothing Method:

This is an extension of moving (rolling) average method where more recent observations get higher weight.

5.1.1. Simple smoothing method:

SES or one-parameter exponential smoothing is applicable to time series which do not contain either of trend or seasonality. Forecast by SES is given by:

$\hat{Y}_{t+1} = \alpha Yt + \alpha(1-\alpha)Yt-1 + \alpha(1-\alpha)2Yt-1 + \cdots$, $0 < \alpha < 1$

where, α is the smoothing parameter for the level. In reality such a series is hard to find. This is a one-step-ahead forecast where all the forecast values are identical.

5.1.2. Holt's Method (Double Exponential Smoothing):

This method is an extension of SES method, proposed by Holt in 1957. This method is applicable where trend is present in the data but no seasonality.
The forecast values are given as:

Forecast equation	: $\hat{Y}_{t+1} = l_t + b_t$	
Level Equation	: $l_t = \alpha Y_t + \alpha(1-\alpha)Y_{t-1}$,	$0 < \alpha < 1$
Trend Equation	: $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$,	$0 < \beta < 1$

where, l_t is the estimate of level and b_t is the trend estimate.
α is the smoothing parameter for the level and β is the smoothing parameter for trend.

5.1.3. Holt-Winter's method (Triple Exponential Smoothing):

This is an extension of Holt's method when seasonality is found in the data.

Forecast equation: $Y_{t+1} = l_t + b_t + s_{t-m(k+1)}$
Level Equation: $l_t = \alpha(Y_t - s_{t-m}) + \alpha(1-\alpha)Y_{t-1}$, $0 < \alpha < 1$
Trend Equation: $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$, $0 < \beta < 1$
Seasonal Equation: $\gamma(Y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$, $0 < \gamma < 1$

This is also known as three parameters exponential or triple exponential because of the three smoothing parameters α, β and γ. This is a general method and a true multi-step ahead forecast.

Case Study continued.

Tractor sales series is to be forecasted using Holt-Winters' method since it contains both trend and seasonality.

```
TS_Train_HW = ExponentialSmoothing(TS_Train,seasonal='multiplicative',trend='additive',freq='M')
TS_Train_HW_autofit = TS_Train_HW.fit(optimized=True)
```

```
TS_Train_HW_autofit.params_formatted
```

	name	param	optimized
smoothing_level	alpha	3.694208e-01	True
smoothing_trend	beta	1.996233e-07	True
smoothing_seasonal	gamma	6.305791e-01	True
initial_level	l.0	2.602306e+02	True
initial_trend	b.0	5.669954e+00	True
initial_seasons.0	s.0	5.302682e-01	True
initial_seasons.1	s.1	5.873613e-01	True
initial_seasons.2	s.2	6.780100e-01	True
initial_seasons.3	s.3	7.100880e-01	True
initial_seasons.4	s.4	7.149382e-01	True
initial_seasons.5	s.5	6.547088e-01	True
initial_seasons.6	s.6	7.127422e-01	True
initial_seasons.7	s.7	7.027975e-01	True
initial_seasons.8	s.8	5.765927e-01	True
initial_seasons.9	s.9	5.047207e-01	True
initial_seasons.10	s.10	4.719774e-01	True
initial_seasons.11	s.11	5.705769e-01	True

A user may also choose values of α, β *and* γ, and can observe the differences in the model.


```
TES_pred = TS_Train_HW_autofit.forecast(steps=len(TS_Test))
TS_Train.plot()
TES_pred.plot()
plt.grid()
plt.legend(['Training Data','Forecasted Values']);
```

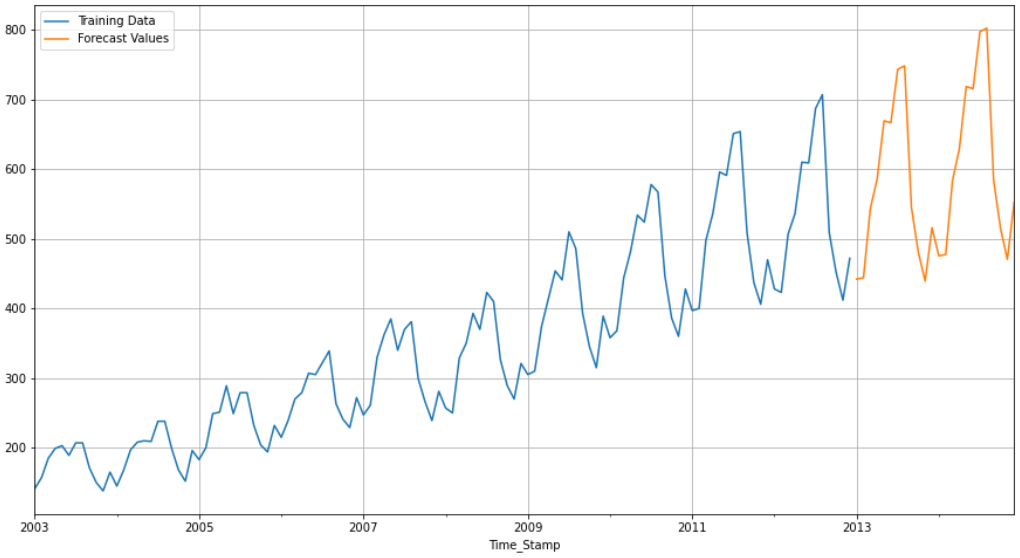


Figure 9: Forecasted sales using Holt Winter's method

Accuracy value using Holt-Winter method

```
## Plotting only the forecast and the test data

TS_Test['Number of Tractor Sold'].plot()
TES_pred.plot()
plt.grid()
plt.title('Tractor Sales: Actual vs Forecast')
plt.xlabel('Time')
plt.legend(['Actual Data','Forecasted Data']);
```

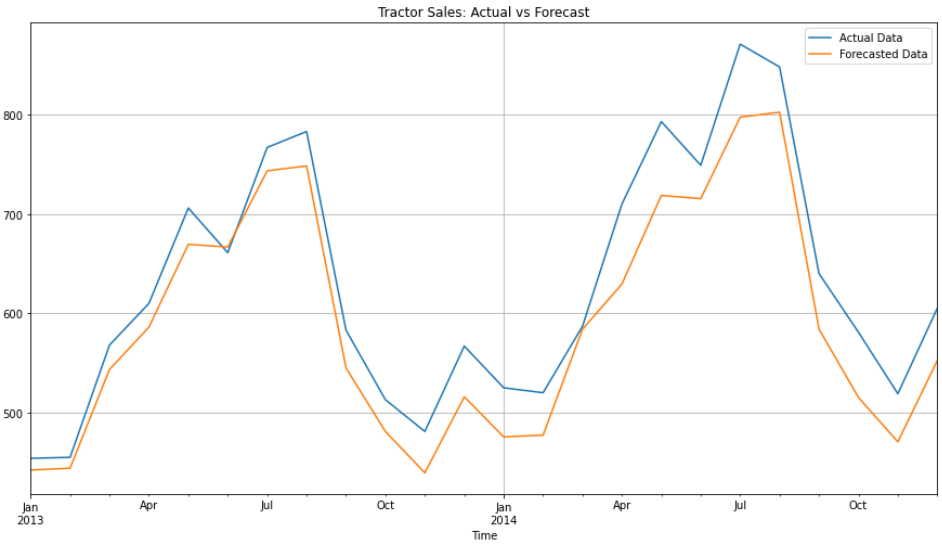


Figure 10: Plot Actual vs. Forecasted sales using HW's method for 2013-2014 years

The blue line shows the actual observations and the red shows forecasted values.
The latter is below the actual values.

```
## Mean Absolute Percentage Error (MAPE) - Function Definition

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean((np.abs(y_true-y_pred))/(y_true))*100

RMSE = mean_squared_error(TS_Test,TES_pred,squared=False)
MAPE = mean_absolute_percentage_error(TS_Test['Number of Tractor Sold'],T
ES_pred)

resultsDf = pd.DataFrame({'Test RMSE': [RMSE],'Test MAPE':[MAPE]}
                         ,index=['TripleExponentialSmoothing'])
resultsDf
```

	Test RMSE	Test MAPE
TripleExponentialSmoothing	45.097549	6.350421

Table 1 represents a summary table of decomposition methods stating its conditions, advantages and disadvantages.

Table 1: Summary Table for Exponential Smoothing Models

Method	When to consider a particular method?		Advantages	Disadvantages
	There is trend in data	There is seasonality in data		
Decomposition Method	May/May not be	May/may not be	Helps in understanding pattern of time series components individually.	Does not generate values for a few initial and last data points
Simple Exponential method	No	No	Suitable when data has no clear presence of trend and seasonality	Takes time in calibrating value of parameter α
Holt's Method	Yes	No	Adjusts level and trend both	Takes time in calibrating value of parameter α, β.
Holt Winter's Method	Yes	Yes	Adjust level, trend and seasonality simultaneously	Takes time in calibrating value of parameter α, β, γ.

5.1.4. Two custom seasonalities (in Python)

The *data_range* function within the *Pandas* library helps adjust seasonality according to requirement. A few examples of various custom seasonality is given below. This list is not exhaustive and various more refined seasonality or multiple seasonality may also be defined for time series.

The following parameters are important for generating time stamps according to frequency (seasonality) using the *date_range* function in *Pandas*.

- 'start'- This defines the time-stamp to indicate the first instance of the data.
- 'period' – This specifies the number of date-time observations to generate.
- 'freq' – This can be used to change the seasonality of the time stamps.
- 'end' – Instead of the 'period' parameter, 'end' can be used to specify the last instance of the observation

Frequency	Code	Output
Business Day (Monday to Friday)	import pandas as pd pd.date_range(start='2020/12/1', periods=12,freq='B')	DatetimeIndex(['2020-12-01', '2020-12-02', '2020-12-03', '2020-12-04', '2020-12-07', '2020-12-08', '2020-12-09', '2020-12-10', '2020-12-11', '2020-12-14', '2020-12-15', '2020-12-16'], dtype='datetime64[ns]', freq='B') Time stamps have been generated in accordance to the business days.
Only Weekends (Saturday and Sunday)	import pandas as pd pd.bdate_range(start='2020/12/1',periods=12,freq='C',weekmask='Sat Sun')	DatetimeIndex(['2020-12-05', '2020-12-06', '2020-12-12', '2020-12-13', '2020-12-19', '2020-12-20', '2020-12-26', '2020-12-27', '2021-01-02', '2021-01-03', '2021-01-09', '2021-01-10'], dtype='datetime64[ns]', freq='C') Time stamps have been generated in accordance to the weekends only.

After the time stamps are generated, they may be used to index data to make it an appropriate time series.

The following links of the *Pandas* library is useful for various other custom date ranges.

1. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.date_range.html
2. https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#timeseries-offset-aliases
3. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.tseries.offsets.CustomBusinessDay.html#pandas.tseries.offsets.CustomBusinessDay

6. Concept of ARIMA

Auto Regressive Integrated Moving Average (ARIMA) models are applied on time series data when the current value is assumed to be correlated to past values and past prediction errors. Therefore, these models are used in defining current value as a linear combination of past values and past prediction errors.

Here, we have defined a few terms that would be useful in understanding ARIMA models in detail.

ARMA models can only be applied only on stationary time series data.

6.1. Stationary Series and Non-stationary series

Stationary process: A process is said to be stationary if its mean and variance are constant over a period of time and, the correlation between the two time periods depends only on the distance or lag between the two periods. Mathematically, let Y_t be a time series with these properties:

Mean: $E(Y_t)=\mu$
Variance: $Var(Y_t)=E(Y_t-\mu)^2=\sigma^2$
Correlation: $\rho_k=E[(Y_t-\mu)(Y_{t+k}-\mu)/(\sigma_t\sigma_{t+k})]$

Where ρ_k is the correlation (or auto-correlation) at lag k between the values of Y_t and Y_{t+k}

So, if mean, variance and correlation (or auto-correlation) of time series data is constant (at different lags) no matter at what point of time it is measured; i.e. if they are time invariant, the series is called a ***stationary time series***. A series not possessing these properties is termed as a ***non-stationary time series***.

6.1.1. Stationarization of a non-stationary series

Since ARIMA model requires a stationary series, a formal stationarity test needs to be applied to the time series under consideration.

Augmented Dickey-Fuller Test: A formal test to check whether time series data follows stationary process.

H_0: Time series is non-stationary

H_1: Time series is stationary

Case Study continued.

To check whether Tractor Sales series is stationary or not.


```
dftest = adfuller(TS_Train,regression='ct',autolag=None,maxlag=24)
print('DF test statistic is %3.3f' %dftest[0])
print('DF test p-value is' ,dftest[1])
print('Number of lags used' ,dftest[2])
```

```
DF test statistic is -1.890
DF test p-value is 0.659709723365856
Number of lags used 24
```

Tractor sales data is not stationary.

Now Log transformed data is put to Augmented Dickey-Fuller test to check for stationarity.

```
TS_Train_log = np.log10(TS_Train)

dftest = adfuller(TS_Train_log,regression='ct',autolag=None,maxlag=24)
print('DF test statistic is %3.3f' %dftest[0])
print('DF test p-value is' ,dftest[1])
print('Number of lags used' ,dftest[2])
```

```
DF test statistic is -1.232
DF test p-value is 0.90383399629346
Number of lags used 24
```

Neither original nor log-transformed series is stationary. Hence, a stationarization is necessary. Often differencing a non-stationary time series leads to a stationary series.

First difference of a series is defined as $D_1=Y_t-Y_{t-1}$

Autocorrelation Function (ACF): Autocorrelation of order p is the correlation between Y_t and Y_{t+k} for all values of k=0,1,…, $-1\leq ACF\leq 1$ and $ACF(0)=1$. ACF measures strength of dependency of current observations on past observations.

Partial Autocorrelation Function (PACF): PACF of order k is the autocorrelation between Y_t and Y_{t+k} adjusting for all the intervening periods i.e. it provides the correlation value between current and k - lagged series by removing the influence of all other observations that exist in between.

- ACF and PACF used together to identify the order of the ARMA.
- Seasonal ACF and PACF examines correlations for seasonal data

Case Study continued

-

```
# ACF and PACF after taking the logarithmic transformation

f,a =  plt.subplots(1,2,sharex=True,sharey=False,squeeze=False)

#Plotting the ACF and the PACF

plot_0 = plot_acf(TS_Train_log,title='ACF Tractor Sales',ax=a[0][0])

plot_1 = plot_pacf(TS_Train_log,title='PACF Tractor Sales',zero= False,ax=a[
0][1]);
```

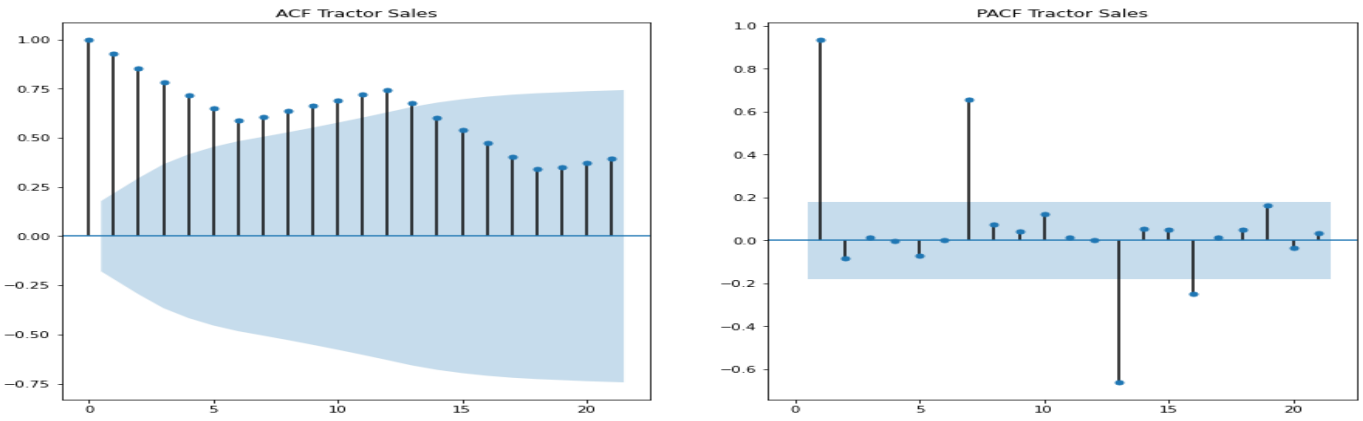


Figure 11: ACF and PACF of Tractor Sales after log transformation

6.2. ARIMA (p, d, q) Model

$ARIMA(p,d,q)$ Model: ARIMA is defined by 3 parameters
p : No of autoregressive terms
d **:** No of differencing to stationarize the series
q**:** No of moving average terms

General form of an ARMA model is: (assuming Y_t is a stationary series)

$$Y_t=\beta_1 Y_{t-1}+\beta_2 Y_{t-2}+\beta_3 Y_{t-3}+\cdots+\beta_p Y_{t-p}+\varepsilon_t+\alpha_1\varepsilon_{t-1}+\alpha_2\varepsilon_{t-2}+\cdots+\alpha_q\varepsilon_{t-q}$$

$\beta_1,\beta_2,\ldots,\beta_p$: Auto-regression coefficients
$\alpha_1,\alpha_2,\ldots,\alpha_q$: Moving average parameters
ε_t : white noise

6.2.1. AR(p) Model

When the current value of variable can be expressed as a linear function of its past values then, it is known as an *auto-regression* process

Autoregressive Process (AR(p)): An autoregressive process of order p is a sequence of a random variable $\boldsymbol{Y_t}$ defined by the rule:

$$\boldsymbol{Y_t=\beta_1 Y_{t-1}+\beta_2 Y_{t-2}+\beta_3 Y_{t-3}+\cdots+\beta_p Y_{t-p}+\varepsilon_t}$$

Several AR processes have simpler interpretation.

$\boldsymbol{AR(1)}$ can be written as: $\boldsymbol{Y_t=\beta_1 Y_{t-1}+\varepsilon}$, and
$\boldsymbol{AR}(2)$ can be written as: $\boldsymbol{Y_t=\beta_1 Y_{t-1}+\beta_2 Y_{t-2}+\varepsilon_t}$
Thus, these models attempts to explain the current values of Y as a linear combination of past values with some additionally externally generated variations ($\boldsymbol{\varepsilon_t}$).

PACF is used for identifying the value of p.

6.2.2. MA(q) Model

When the current value of the series is a function of past forecast errors this model is known as a moving average model

A moving average process of order q i.e. MA(q), is a sequence Y_t defined by the rule:

$$\boldsymbol{Y_t=\mu+\varepsilon_t+\alpha_1\varepsilon_{t-1}+\alpha_2\varepsilon_{t-2}+\cdots+\alpha_q\varepsilon_{t-q}}$$

ACF is used for identifying the value of q.

6.2.3. ARMA Model

ARMA model is a combination of two basic processes i.e. AR and MA. The defining equation of ARMA (p, q) process is:

$$\boldsymbol{Y_t=\beta_1 Y_{t-1}+\beta_2 Y_{t-2}+\beta_3 Y_{t-3}+\cdots+\beta_p Y_{t-p}+\varepsilon_t+\alpha_1\varepsilon_{t-1}+\alpha_2\varepsilon_{t-2}+\cdots+\alpha_q\varepsilon_{t-q}}$$

6.2.4. ARIMA Model

$\boldsymbol{ARIMA\ (p,d,q)}$**:** ARIMA model is an advance version of ARMA model where $d > 0$ indicates that the original series is non-stationary and d differencing is required to make is stationary. We will also define the models with centered Y_t, centering is done by subtracting the mean of the stationary series.

So far we have discussed non-seasonal ARIMA model only.

6.3. SARIMA(p,d,q)(P,D,Q)[m]:

Seasonal ARIMA model with seasonal frequency m

Seasonal ARIMA models are more complex models with seasonal adjustments. These models are used when time series data has significant seasonality. The most general form of seasonal ARIMA is $ARIMA(p,d,q)*ARIMA(P,D,Q)$[m], where P, D, Q are defined as seasonal AR component, seasonal difference and seasonal MA component respectively. And, 'm' represents the frequency (time interval) at which the data is observed. For example, a monthly series will have $m = 12$.

Seasonal ACF and PACF may be used to understand seasonality.

Case Study continued:

Figure (11) indicates the presence of seasonality using the ACF and PACF plots of tractor sales data as the past values are significantly correlated. It is also clear that at every multiple of 12 the ACF swings higher than its neighbouring values.

```
## Here we have taken the range of p,q,P and Q to be within 0 to 2. We can chan
ge this if need be.

import itertools
p = range(0, 3)
q = range(0, 3)
d = range(1, 2)
pdq = list(itertools.product(p, d, q))
model_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

## Defining an empty data frame to store the parameter values along with th
e model AIC
SARIMA_AIC = pd.DataFrame(columns=['param','seasonal', 'AIC'])

for param in pdq:
    for param_seasonal in model_pdq:
        try:
            SARIMA_model = sm.tsa.statespace.SARIMAX(TS_Train_log['Number o
f Tractor Sold'].values,
                                                     order=param,
                                                     seasonal_order=param_season
al)

            results_SARIMA = SARIMA_model.fit()
        except:
            continue
        print('SARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, result
s_SARIMA.aic))
        SARIMA_AIC = SARIMA_AIC.append({'param':param,'seasonal':param_seas
onal ,'AIC': results_SARIMA.aic}, ignore_index=True)
```



```
## Sorting the parameters of the SARIMA models to get the parameters which
give us the lowest AIC value

SARIMA_AIC.sort_values(by=['AIC']).head()
```

	param	seasonal	AIC
10	(0, 1, 1)	(0, 1, 1, 12)	-568.069929
28	(1, 1, 0)	(0, 1, 1, 12)	-567.883349
11	(0, 1, 1)	(0, 1, 2, 12)	-566.380580
13	(0, 1, 1)	(1, 1, 1, 12)	-566.359756
19	(0, 1, 2)	(0, 1, 1, 12)	-566.101192

```
## Running the SARIMA model and getting the model diagnostics

## SARIMA(0, 1, 1)(0, 1, 1, 12)
TS_AutoARIMA = sm.tsa.statespace.SARIMAX(TS_Train_log,
                                         order=(0,1,1),
                                         seasonal_order=(0, 1, 1, 12))
TS_AutoARIMA = TS_AutoARIMA.fit()
print(TS_AutoARIMA.summary())
```

```
                                 SARIMAX Results
==========================================================================================
Dep. Variable:              Number of Tractor Sold   No. Observations:                  120
Model:             SARIMAX(0, 1, 1)x(0, 1, 1, 12)   Log Likelihood                 287.030
Date:                            Tue, 20 Oct 2020   AIC                           -568.059
Time:                                    16:27:33   BIC                           -560.041
Sample:                                01-31-2003   HQIC                          -564.809
                                     - 12-31-2012
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3706      0.087     -4.285      0.000      -0.540      -0.201
ma.S.L12      -0.5280      0.105     -5.024      0.000      -0.734      -0.322
sigma2         0.0003   3.31e-05      7.968      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):                 1.26
Prob(Q):                              0.77   Prob(JB):                         0.53
Heteroskedasticity (H):               0.38   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.45
===================================================================================
```


According to the result, *ARIMA*(0,1,1)(0,1,1)[12] is the indicated model for Tractor Sales. with AIC= (-568.059) and BIC= (-560.041).

```
## Checking the behaviour of the residuals of the model

TS_AutoARIMA.plot_diagnostics();
```

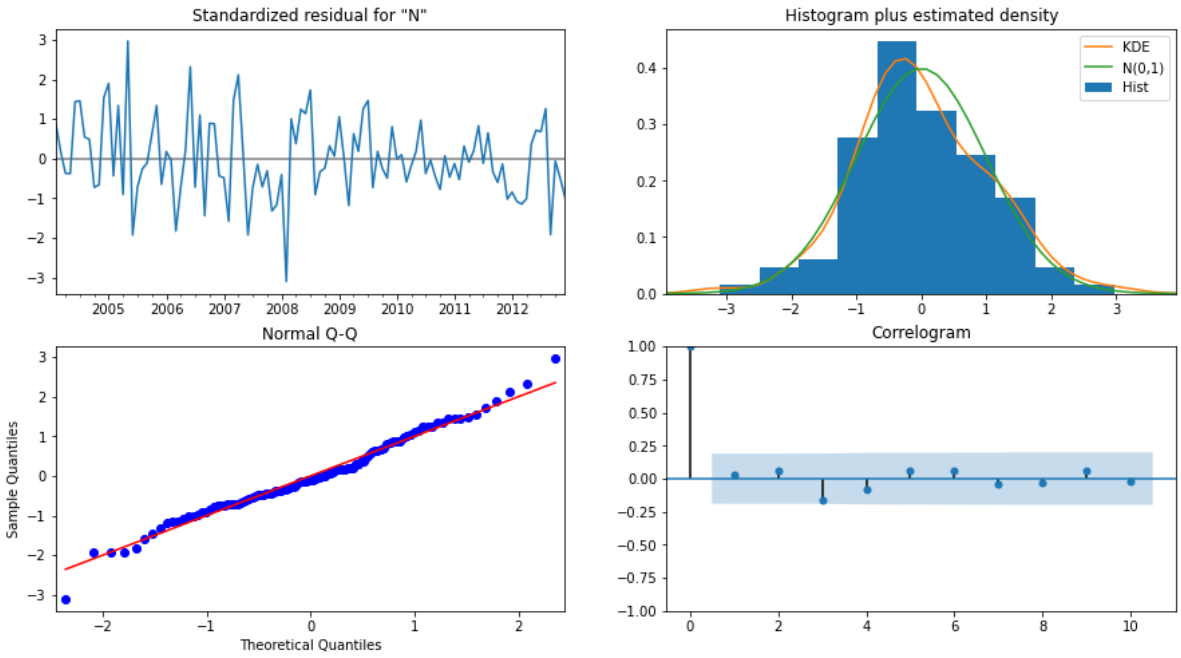


Figure 12: Residual Diagnostics of the (0,1,1)(0,1,1)12 SARIMA Model

Alternatively, one might investigate other suitable model(s) for a time series using ACF and PACF for the differenced series.

```
# Plot the first difference series, ACF and PACF of Log(Tractor Sales)

TS_Train_log.diff().plot()
plt.grid()
# ACF and PACF after taking the differenced logarithmic transformation

f,a =  plt.subplots(1,2,sharex=True,sharey=False,squeeze=False)

#Plotting the ACF and the PACF

plot_0 = plot_acf(TS_Train_log.diff().dropna(),title='ACF of differenced Log
Tractor Sales',ax=a[0][0])

plot_1 = plot_pacf(TS_Train_log.diff().dropna(),title='PACF of differenced Log Tractor Sales',zero =
False,ax=a[0][1]);
```

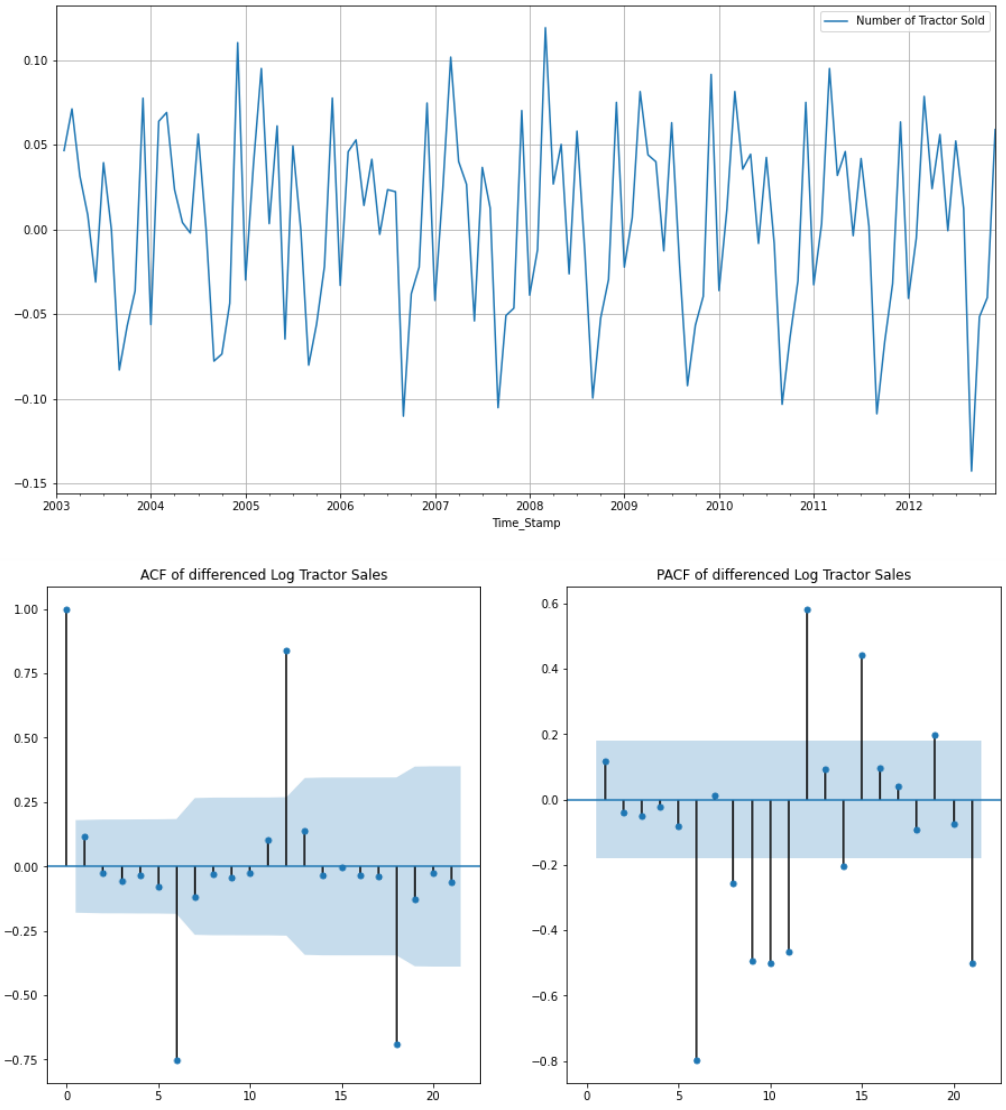
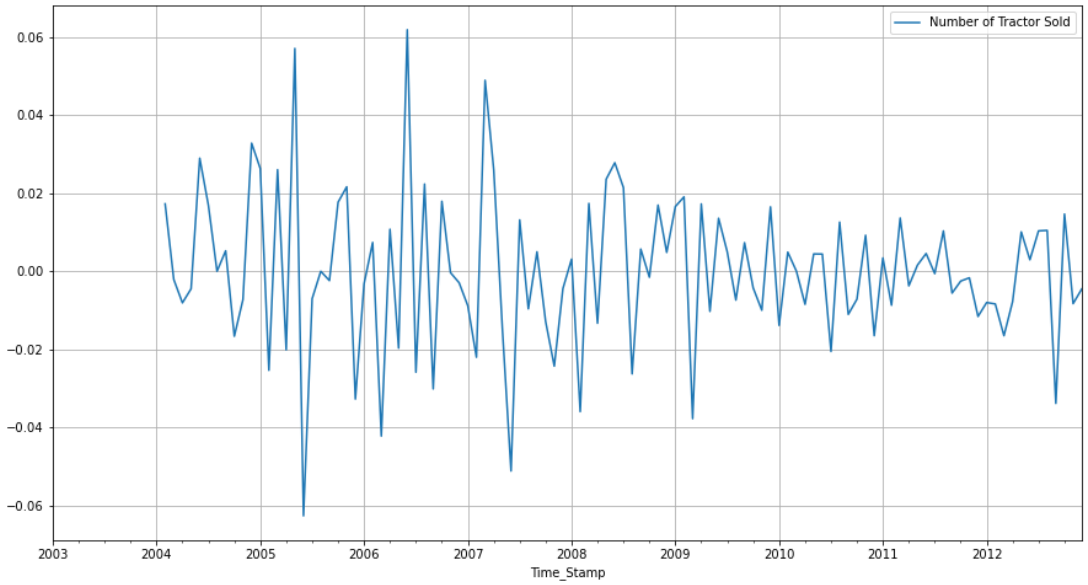



Figure 13: Plot of first difference series, ACF and PACF of Log(Tractor Sales)

The plots of ACF and PACF indicate possible values for p to be 1 and q to be 0.

```
##Plot of first difference and seasonal first difference series, ACF and PACF of Log(Tractor Sales)

TS Train log.diff(12).diff().plot()
plt.grid()

# ACF and PACF after taking the differenced logarithmic transformation --> Seasonal Series Plot

f,a =  plt.subplots(1,2,sharex=True,sharey=False,squeeze=False)

#Plotting the ACF and the PACF

plot_0 = plot_acf(TS_Train_log.diff(12).diff().dropna(),title='ACF of differenced Log Seasonal Tractor Sales',ax=a[0][0])

plot_1 = plot_pacf(TS_Train_log.diff(12).diff().dropna(),title='PACF of differenced Log Seasonal Tractor Sales',ax=a[0][1]);
```

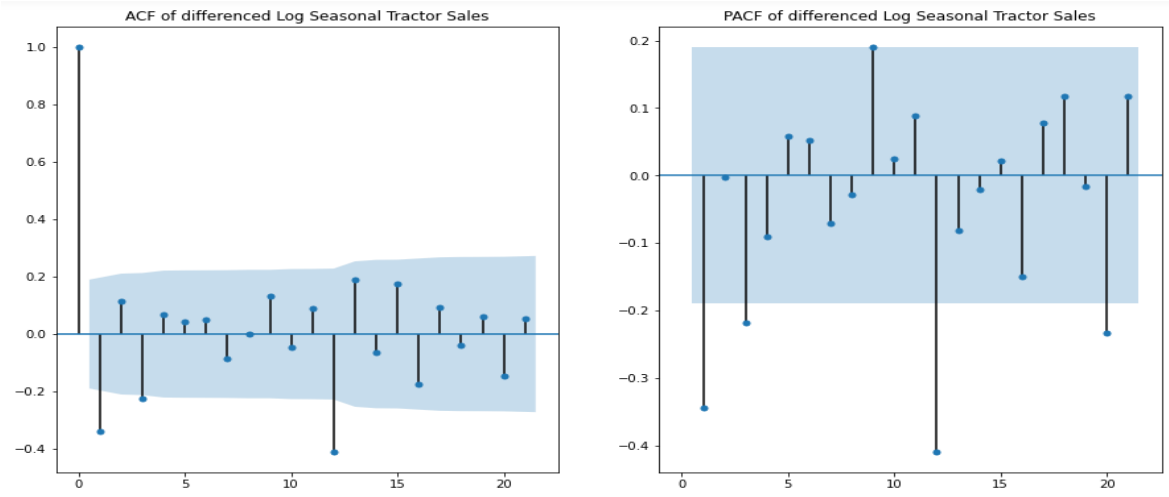


Figure 14: Plot of first difference and seasonal first difference series, ACF and PACF of Log(Tractor Sales)

Both ACF and PACF plots indicate possible value of P to be 1 and Q to be 1.
A user may choose different values of p,d,q ($or\ P,D\ and\ Q$) and compare AIC, BIC values and select the 'best' model accordingly.

For example, we have chosen $ARIMA(0,1,1)(1,1,1)[12]$ as an alternative model and got lower values for AIC and BIC compared to results obtained from using an automated version of the ARIMA model.


```
## Changing the p,q,P and Q values by looking at the ACF and PACF

## SARIMA(0, 1, 1)(1, 1, 1, 12)
TS_f = sm.tsa.statespace.SARIMAX(TS_Train_log,
                                 order=(0,1,1),
                                 seasonal_order=(1, 1, 1, 12))
TS_f = TS_f.fit()
print(TS_f.summary())
```

```
                                 SARIMAX Results
==========================================================================================
Dep. Variable:             Number of Tractor Sold   No. Observations:                  120
Model:             SARIMAX(0, 1, 1)x(1, 1, 1, 12)   Log Likelihood                 287.179
Date:                            Tue, 20 Oct 2020   AIC                           -566.359
Time:                                    18:29:29   BIC                           -555.667
Sample:                                01-31-2003   HQIC                          -562.024
                                     - 12-31-2012
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3541      0.087     -4.093      0.000      -0.524      -0.185
ar.S.L12      -0.0817      0.232     -0.352      0.725      -0.537       0.374
ma.S.L12      -0.4751      0.261     -1.820      0.069      -0.987       0.036
sigma2         0.0003   3.28e-05      8.003      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 1.11
Prob(Q):                              0.95   Prob(JB):                         0.58
Heteroskedasticity (H):               0.38   Skew:                             0.12
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.44
===================================================================================

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

```
## Checking the residual behaviour of the above model
TS_f.plot_diagnostics();
```

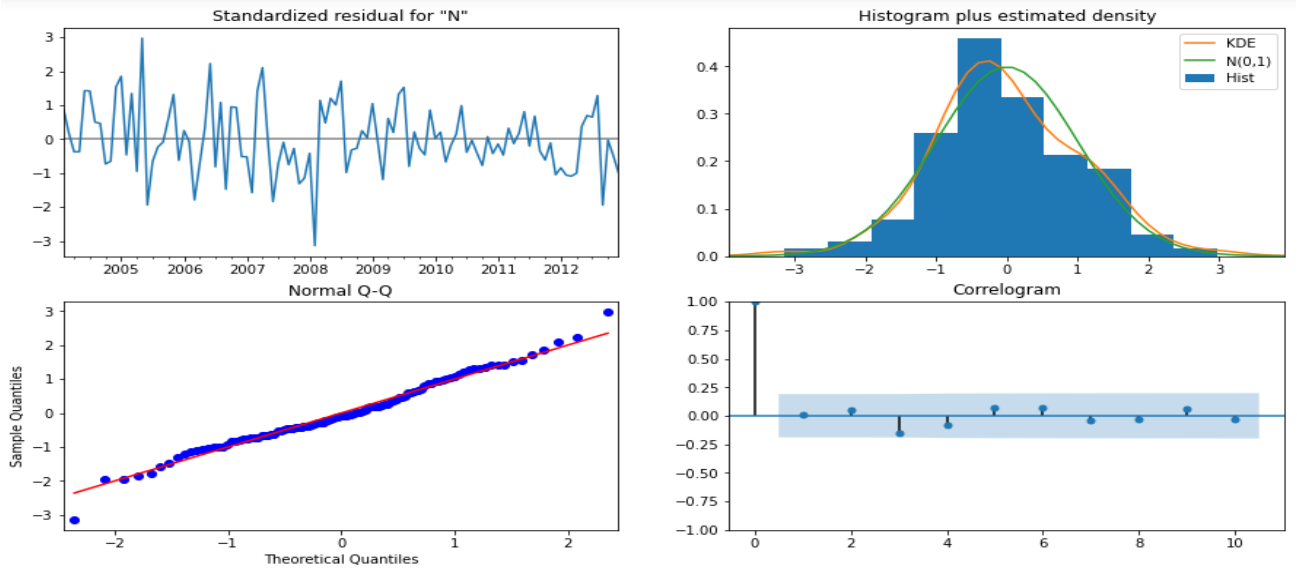



Figure 15: Residual Diagnostics of the (0,1,1)(1,1,1)12 SARIMA Model

Model Validation: Seasonal ARIMA model with seasonal frequency m

```
# Forecast for the test set duration using the automated SARIMA model with
95% confidence intervals

pred_AutoARIMA = TS_AutoARIMA.get_forecast(steps=len(TS_Test))

# plot the forecast along with the confidence band

axis = TS_Train_log.plot()
pred_AutoARIMA.summary_frame(alpha=0.05)['mean'].plot(ax=axis, label='Forec
ast', alpha=0.7)
axis.fill_between(pred_AutoARIMA.summary_frame(alpha=0.05).index, pred_Auto
ARIMA.summary_frame(alpha=0.05)['mean_ci_lower'],
                  pred_AutoARIMA.summary_frame(alpha=0.05)['mean_ci_upper']
, color='k', alpha=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('Number of Tractor Sold')
plt.legend(loc='best')
plt.title('Tractor sales data forecast using SARIMA (0,1,1)(0,1,1)[12]')
plt.grid();
```

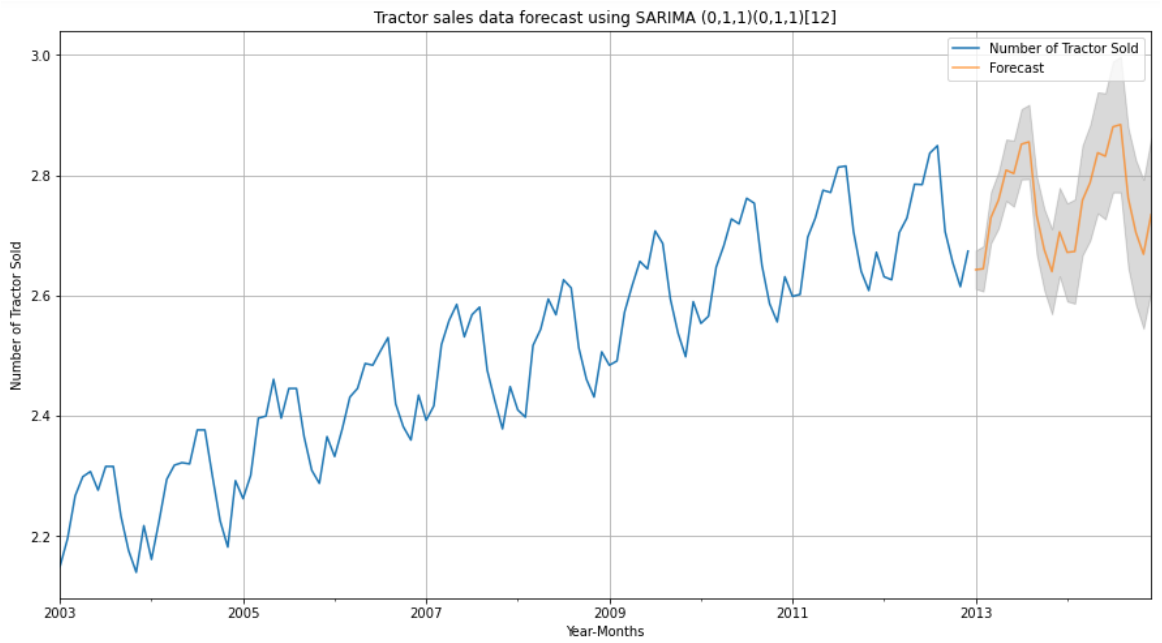



Figure 16: Tractor sales data forecast using SARIMA (0,1,1)*(0,1,1)[12]

Forecasting made using SARIMA model (0,1,1)*(1,1,1) is as given below.

```
# Forecast for the test set duration using the manual SARIMA model with 95%
confidence intervals

pred_f = TS_f.get_forecast(steps=len(TS_Test))

# plot the forecast along with the confidence band

axis = TS_Train_log.plot()
pred_f.summary_frame(alpha=0.05)['mean'].plot(ax=axis, label='Forecast', al
pha=0.7)
axis.fill_between(pred_f.summary_frame(alpha=0.05).index, pred_f.summary_fr
ame(alpha=0.05)['mean_ci_lower'],
                  pred_f.summary_frame(alpha=0.05)['mean_ci_upper'], color=
'k', alpha=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('Number of Tractor Sold')
plt.legend(loc='best')
plt.title('Tractor sales data forecast using SARIMA (0,1,1)(1,1,1)[12]')
plt.grid();
```

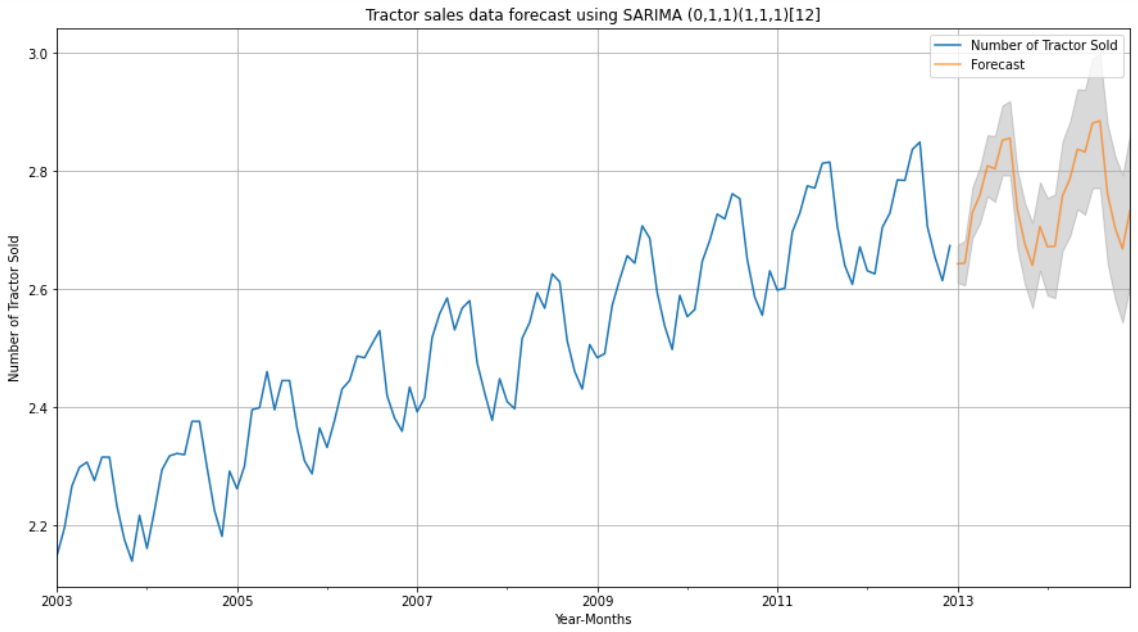



Figure 17: Tractor sales data forecast using ARIMA (0,1,1)(1,1,1)[12]

Accuracy measures: SARIMA (0,1,1)*(0,1,1)[12]

```
## Plot Actual vs. Forecasted sales using SARIMA(0,1,1)*(0,1,1)[12] for 2013-2014 years

TS_Test.plot()
np.power(10,pred_AutoARIMA.summary_frame(alpha=0.05)['mean']).plot()
plt.legend(['Actual Data','Forecasted Data']);
plt.title('Plot of Actual vs. Forecasted sales using SARIMA(0,1,1)*(0,1,1)[12]
for 2013-2014 years')
plt.grid();
```

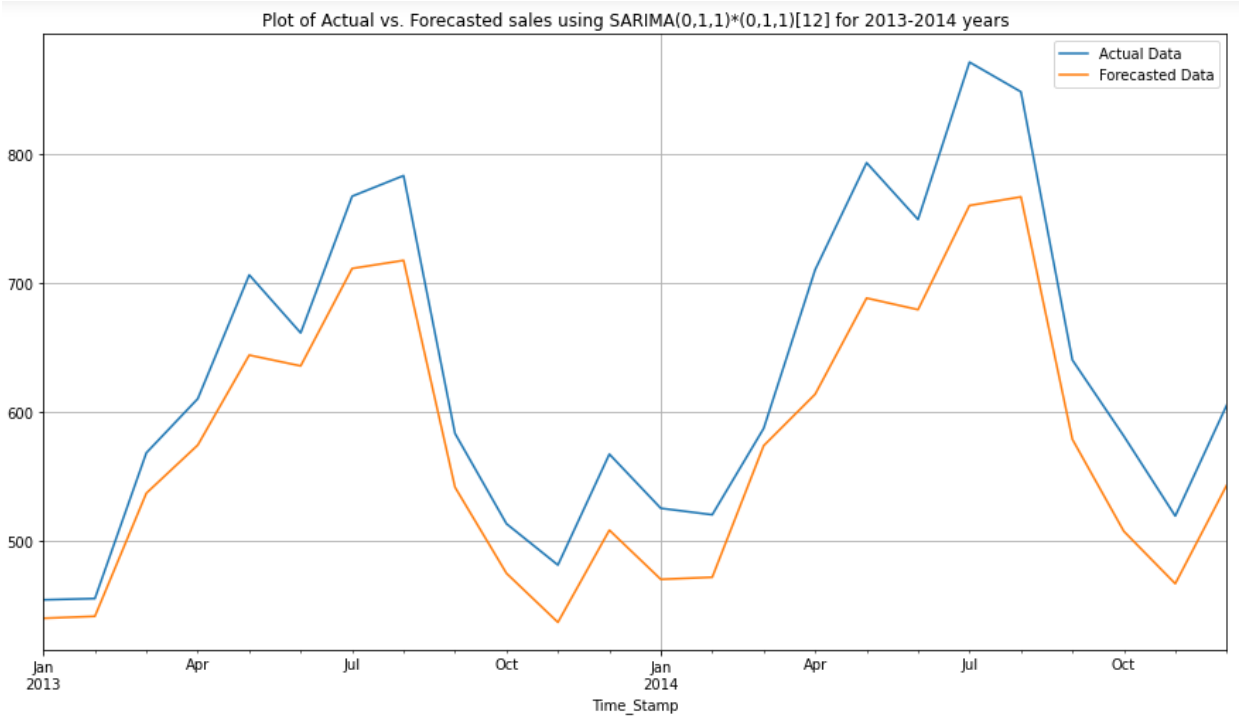



Figure 18: Plot Actual vs. Forecasted sales using SARIMA(0,1,1)*(0,1,1)[12] for 2013-2014 years

```
## Accuracy Measures for SARIMA(0,1,1)(0,1,1)12

RMSE1 = mean_squared_error(TS_Test.values,np.power(10,pred_AutoARIMA.summar
y_frame()['mean']).values,squared=False)
MAPE1 = mean_absolute_percentage_error(TS_Test,np.power(10,pred_AutoARIMA.s
ummary_frame()['mean']))

print("Accuracy Measures: RMSE:", RMSE1, "and MAPE:", MAPE1,'%')
```

```
Accuracy Measures: RMSE: 60.873021905862245 and MAPE: 20.88415799584785 %
```

Accuracy measures: SARIMA (0,1,1)*(1,1,1)[12]

```
## Plot Actual vs. Forecasted sales using SARIMA(0,1,1)*(1,1,1)[12] for 201
3-2014 years

TS_Test.plot()
np.power(10,pred_f.summary_frame(alpha=0.05)['mean']).plot()
plt.legend(['Actual Data','Forecasted Data']);
plt.title('Plot of Actual vs. Forecasted sales using SARIMA(0,1,1)*(1,1,1)[
12] for 2013-2014 years')
plt.grid();
```

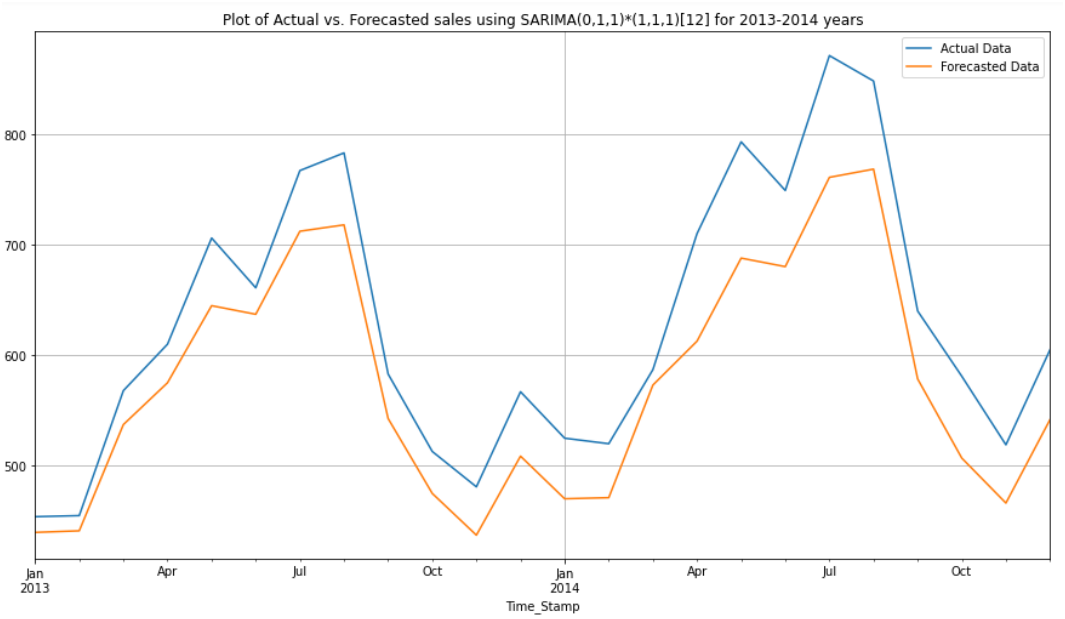



Figure 19: Plot Actual vs. Forecasted sales using ARIMA (0,1,1)*(1,1,1)[12] for 2013-2014 years

```
## Accuracy Measures for SARIMA(0,1,1)(1,1,1)12

RMSE2 = mean_squared_error(TS_Test.values,np.power(10,pred_f.summary_frame(
)['mean']).values,squared=False)
MAPE2 = mean_absolute_percentage_error(TS_Test.values,np.power(10,pred_f.su
mmary_frame()['mean']).values)

print("Accuracy Measures: RMSE:", RMSE2, "and MAPE:", MAPE2,'%')
```

```
Accuracy Measures: RMSE: 60.479384812908954 and MAPE: 20.90140784023054 %
```

It seems that ARIMA(0, 1, 1)(1, 1, 1)[12] not only has smaller AIC and BIC values compared to ARIMA(0, 1, 1)(0, 1, 1)[12] recommended by the automated ARIMA, it provides a smaller value of RMSE, albeit marginal.

Table 2, represents a summary for ARIMA (or SARIMA) models as discussed above.

Table 2: Summary Table for ARIMA models

Method	Advantages	Disadvantages
$AR(p)$	Inculcates effect of past observations of the current data	Interpretation become more complicated when it is dependent on its past values more than 1.
$MA(q)$	Managing current values highly impacted by random errors.	Interpretability issue arises when dependency on past values increases more than 1.
$ARIMA\ (p,d,q)$ (non-seasonal data) Or $ARIMA\ (p,d,q)\ (P,D,Q)$ (seasonal data)	It is a generalized model, which considers effect of past values and random values along with (seasonality if present).	Complicated in calibrating right combination of p, d and q and (P, D, Q if seasonality is present).

A flow chart has been defined for understating ARIMA/ SARIMA models below: -

Case Study continued

Table 3: Comparison of result

	Test RMSE	Test MAPE
TripleExponentialSmoothing	45.097549	6.350421
SARIMA(0, 1, 1)(0, 1, 1)12	60.873022	20.884158
SARIMA(0, 1, 1)(1, 1, 1)12	60.479385	20.901408

Final Forecasts

Once a model is chosen and validated, forecasts into the future to be determined. Tractor sales to be forecasted for 24 months: 2015 Jan – 2016 Dec.

Going strictly by MAPE, recommended model is Triple Exponential Smoothing (Holt Winter's Model).


```
TS_df_HW = ExponentialSmoothing(df,seasonal='mul',trend='additive',freq='M')
TS_df_HW_autofit = TS_df_HW.fit(smoothing_level=3.694208e-01,smoothing_trend=1.996233e-07,smoothing_seasonal=6.305791e-01)
```

```
TS_df_HW_autofit.params_formatted
```

	name	param	optimized
smoothing_level	alpha	3.694208e-01	False
smoothing_trend	beta	1.996233e-07	False
smoothing_seasonal	gamma	6.305791e-01	False
initial_level	l.0	2.678225e+02	True
initial_trend	b.0	6.407265e+00	True
initial_seasons.0	s.0	5.141684e-01	True
initial_seasons.1	s.1	5.692835e-01	True
initial_seasons.2	s.2	6.566287e-01	True
initial_seasons.3	s.3	6.873381e-01	True
initial_seasons.4	s.4	6.917788e-01	True
initial_seasons.5	s.5	6.335173e-01	True
initial_seasons.6	s.6	6.889339e-01	True
initial_seasons.7	s.7	6.791171e-01	True
initial_seasons.8	s.8	5.578601e-01	True
initial_seasons.9	s.9	4.888158e-01	True
initial_seasons.10	s.10	4.572424e-01	True
initial_seasons.11	s.11	5.517919e-01	True

```
# Plot Actual vs. Forecasted sales using HW method for 2015-2016 years

df.plot()
TS_df_HW_autofit.forecast(steps=24).plot()
plt.legend(['Actual','Forecast'])
plt.title('Forecast from the Holt-Winters Multiplicative Method')
plt.grid();
```

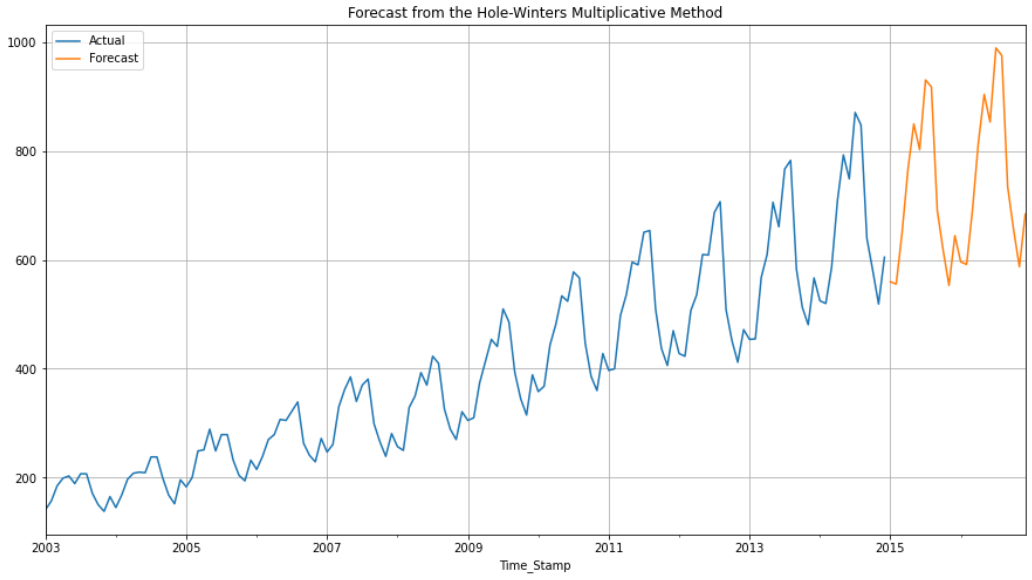


Figure 20: Plot Actual vs. Forecasted sales using Triple Exponential Smoothing method for 2015-2016 years

For illustrative purpose the codes and output from the model Holt-Winters (given above) and the chosen ARIMA model are also given.

```
## Actual forecast by SARIMA (0,1,1)(1,1,1)[12]


TS_final_arima = sm.tsa.statespace.SARIMAX(np.log10(df),
                                   order=(0,1,1),
                                   seasonal_order=(1, 1, 1, 12))
TS_final_arima = TS_final_arima.fit()

pred_final_arima = TS_final_arima.get_forecast(steps=24)

# plot the forecast along with the confidence band

axis = np.log10(df).plot()
pred_final_arima.summary_frame(alpha=0.05)['mean'].plot(ax=axis, lab
el='Forecast', alpha=0.7)
axis.fill_between(pred_final_arima.summary_frame(alpha=0.05).index,
                  pred_final_arima.summary_frame(alpha=0.05)['mean_c
i_lower'],
                  pred_final_arima.summary_frame(alpha=0.05)['mean_c
i_upper'], color='k', alpha=.15)
axis.set_xlabel('Year-Months')
axis.set_ylabel('Number of Tractor Sold')
plt.legend(loc='best')
plt.title('Forecast from SARIMA(0,1,1)(1,1,1)[12]')
plt.grid();
```

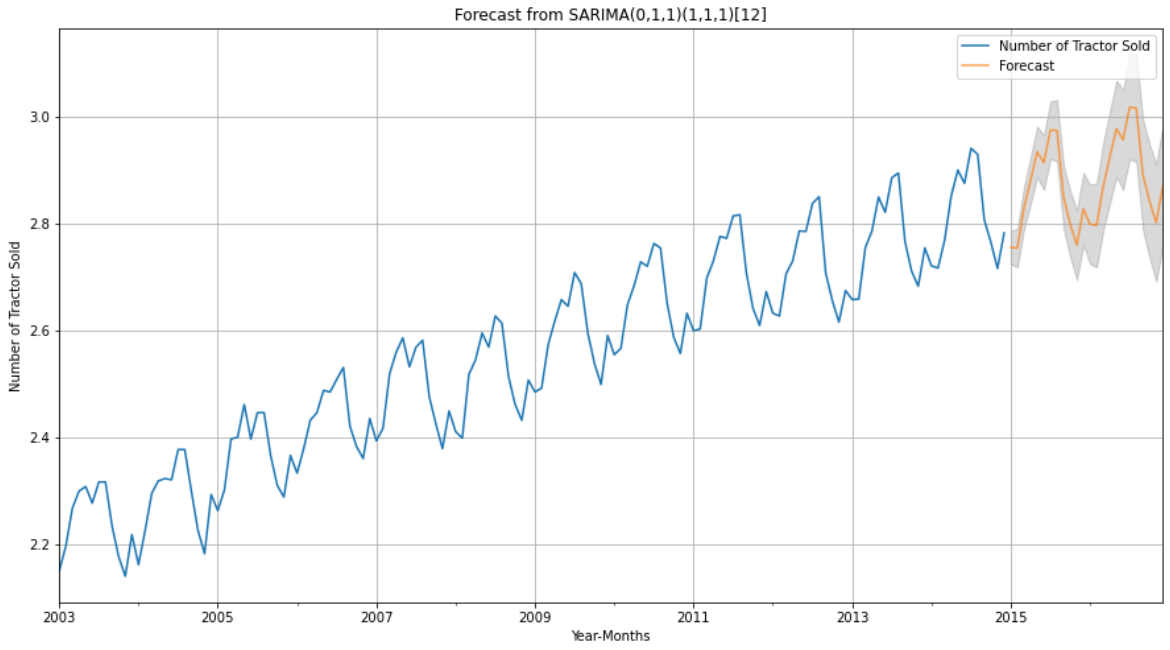


Figure 21: Plot Actual vs. Forecasted sales using SARIMA Model for 2015-2016 years

6.4. **ARIMAX** Model

Often a times series is influenced by one or more exogenous variables, i.e. a time series may have two independent components. One component is directly influenced by the part observations, but another comment works like a multiple linear regression. Such a time series model is known as an ARIMAX model and may be treated as an extension of ARIMA model. The 'X' in ARIMAX (or SARIMAX) stands for the independent predictors in the model.

General form of an ARIMAX model is: (assuming Y_t is a stationary series)

$$Y_t=\beta_1 Y_{t-1}+\beta_2 Y_{t-2}+\beta_3 Y_{t-3}+\cdots+\beta_p Y_{t-p}+\varepsilon_t+\alpha_1\varepsilon_{t-1}+\alpha_2\varepsilon_{t-2}+\cdots+\alpha_q\varepsilon_{t-q}+\gamma Xt$$

$\beta_1,\beta_2,\ldots,\beta_p$: Auto-regression coefficients
$\alpha_1,\alpha_2,\ldots,\alpha_q$: Moving average parameters
γ : Regression coefficient
ε_t : white noise

It is possible that X is a vector, like a set of typical multiple linear regression predictor. X may vary over time or not. For example, if price of a commodity is modelled as a time series, X may be price index, which is a time dependent variable. Alternatively, X may be a categorical variable, such as geographic location.

There is one major difference with multiple linear regression and inclusion of a set of predictors in an ARIMA model. Interpretation of the regression coefficients are not straight forward. The estimated coefficient of X is not an estimate of the increase in Y_t for a unit increase in X, because Y_t includes the lag variables.

A second case study is given below as an example of ARIMAX model fitting.

Case Study with an Exogenous Variable

The dataset Pollution_Data.csv contains average weekly value of several polluting particles in one pollution monitoring station. The main parameter for monitoring ambient air quality is PM2.5. The data points are weekly average from 2013 – 2017.

Fig 22 shows the data pattern. Note that the frequency of the time stamps is weekly.


```
## Upload the data file and plot the same

df = pd.read_csv('Pollution_Data.csv')
daterange = pd.date_range(start='2013-03-03',periods=len(df),freq='W')
df['Time_Stamp'] = daterange
df.set_index(keys='Time_Stamp',inplace=True)
rcParams['figure.figsize'] = 15,8
df['PM2.5'].plot(grid=True);
```

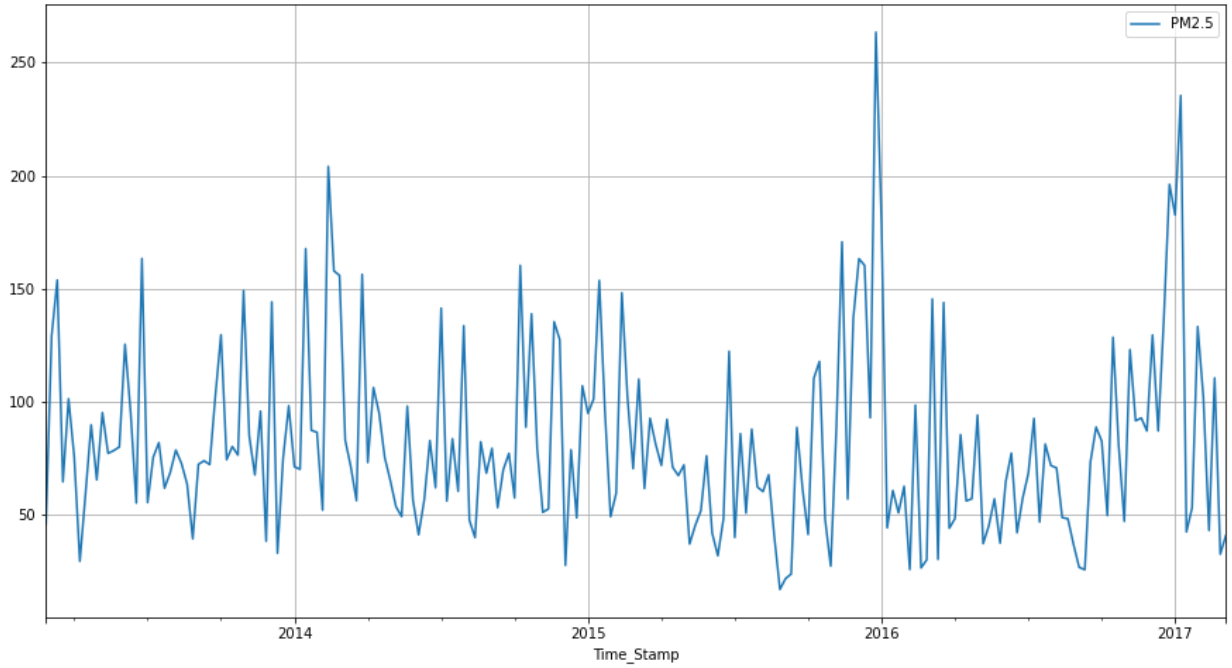


Figure 22: PM2.5 Pollution data

Following are the observations obtained from Figure 22.:

I. Data values are stored in correct time order and no data is missing.
II. The PM2.5 values are randomly fluctuating.

Since we have already discussed the theory behind building ARIMA models using the lowest Akaike Information Criteron, we will directly go ahead and apply those concepts over here.
But before that we need to split the data into training and test and then go on to check for stationarity of the Training Data.

6.5. Data Split and Plot

```
## Spliting data into training and test data sets
TS_Train = df[df.index.year <= 2015]
TS_Test = df[df.index.year > 2015]
```



```
TS_Train['PM2.5'].plot()
TS_Test['PM2.5'].plot()
plt.grid()
plt.title('PM2.5 Plot')
plt.xlabel('Year')
plt.ylabel('PM2.5')
plt.legend(['Training Data','Test Data'],title='Data Split');
```

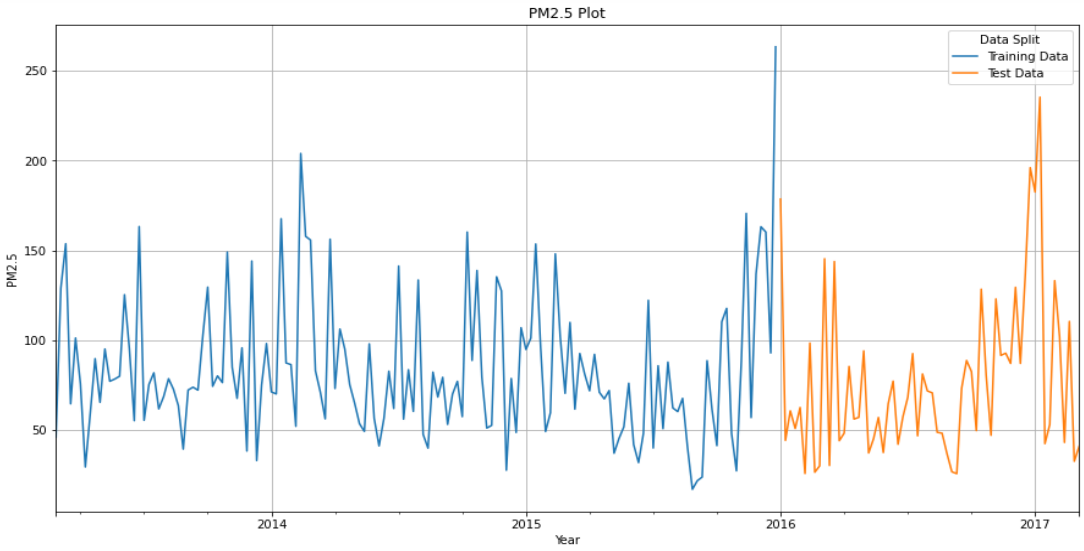


Figure 23: PM2.5 Data (split into Training and testing purpose)

Let us check the ACF plot to understand the exact nature of the seasonality in the data.

```
## Plotting the ACF plot

plot_acf(TS_Train['PM2.5'],lags=60);
```

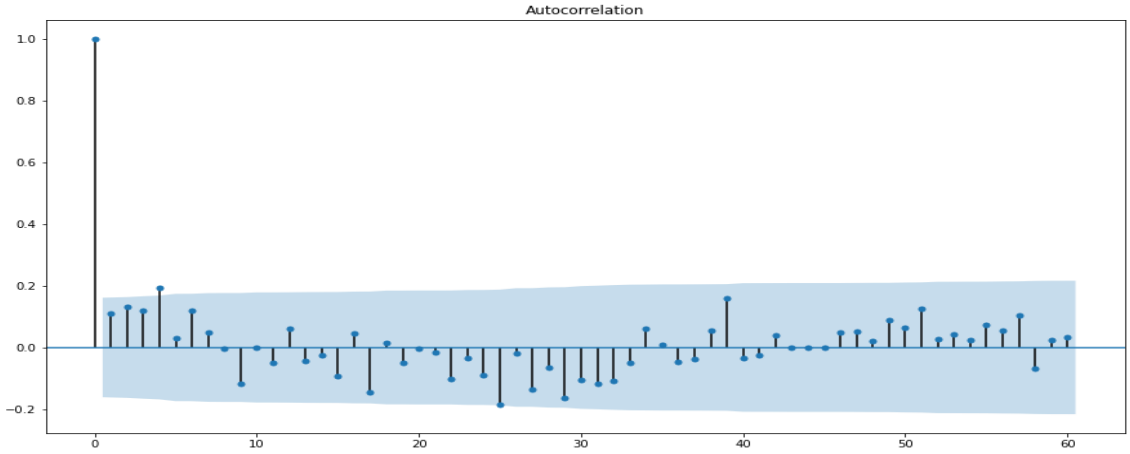


Figure 24: PM2.5 Data ACF Plot

Notice that there is no clear indicative seasonality in the data and thus we opt for ARIMA over here and not SARIMA.

Now, we will check whether the Training data is stationary using the Augmented Dickey-Fuller Test.

```
dftest = adfuller(TS_Train['PM2.5'],regression='ct')
print('DF test statistic is %9.9f' %dftest[0])
print('DF test p-value is' ,dftest[1])
print('Number of lags used' ,dftest[2])
```

```
DF test statistic is -2.814812125
DF test p-value is 0.19155731152974392
Number of lags used 3
```

The data is not stationary at 95% confidence interval. Now, we will take a first order difference of the data and then check for stationarity.

```
dftest = adfuller(TS_Train['PM2.5'].diff().dropna(),regression='ct')
print('DF test statistic is %9.9f' %dftest[0])
print('DF test p-value is' ,dftest[1])
print('Number of lags used' ,dftest[2])
```

```
DF test statistic is -8.494372102
DF test p-value is 5.168558877586937e-12
Number of lags used 4
```

After taking a first order differencing we see that the data has indeed become stationary at 95% confidence level.

Let us plot the differenced Time Series once and check the plots of the ACF and the PACF.


```
TS_Train['PM2.5'].diff().dropna().plot(grid=True)

# ACF and PACF

f,a =  plt.subplots(1,2,sharex=True,sharey=False,squeeze=False)

#Plotting the ACF and the PACF

plot_0 = plot_acf(TS_Train['PM2.5'].diff(),ax=a[0][0],missing='drop')

plot_1 = plot_pacf(TS_Train['PM2.5'].diff().dropna(),ax=a[0][1],zero=False);
```

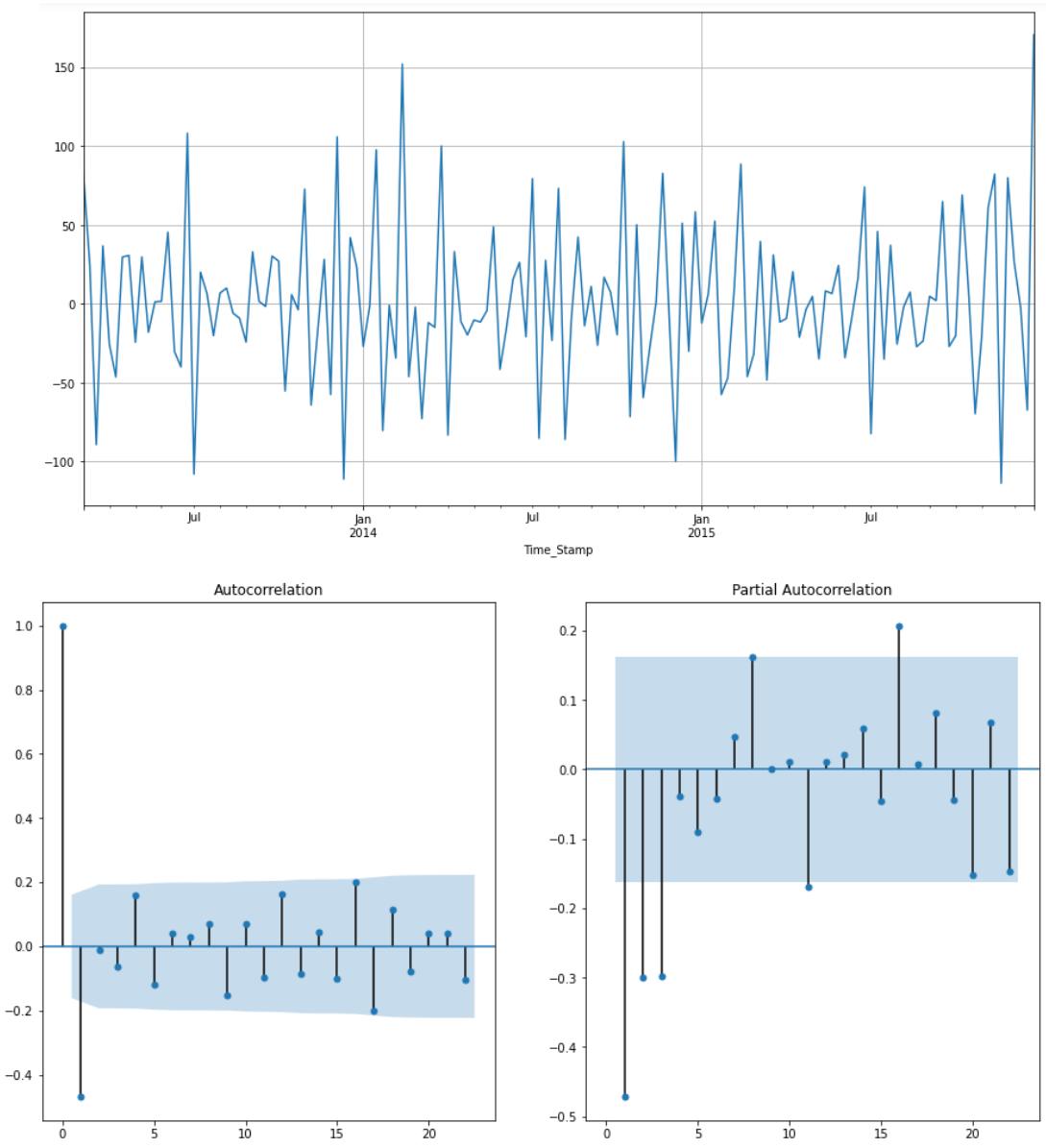


Figure 25: Plot of first difference series, ACF and PACF of PM2.5

We are going to fit the differenced series in the ARIMA(p,d,q) model.

ARIMA(p,d,q):

```
## The following loop helps us in getting a combination of different parame
ters of p and q in the range of 0 and 3
## We have kept the value of d as 1 as we need to take a difference of the
series to make it stationary.

import itertools
p = q = range(0, 4)
d= range(1,2)
pdq = list(itertools.product(p, d, q))

# Creating an empty Dataframe with column names only
ARIMA_AIC = pd.DataFrame(columns=['param', 'AIC'])

for param in pdq:# running a loop within the pdq parameters defined by iter
tools
    ARIMA_model = sm.tsa.statespace.SARIMAX(TS_Train['PM2.5'].values,order=
param).fit()#fitting the ARIMA model using the parameters from the loop
    ARIMA_AIC = ARIMA_AIC.append({'param':param, 'AIC': ARIMA_model.aic}, i
gnore_index=True)
    #appending the AIC values and the model parameters to the previously cr
eated data frame
    #for easier understanding and sorting of the AIC values

## Sort the above AIC values in the ascending order to get the parameters f
or the minimum AIC value

ARIMA_AIC.sort_values(by='AIC',ascending=True)
```

	param	AIC
1	(0, 1, 1)	1500.212988
6	(1, 1, 2)	1500.965614
2	(0, 1, 2)	1501.225767
5	(1, 1, 1)	1501.411728
14	(3, 1, 2)	1501.981607
3	(0, 1, 3)	1502.400824
10	(2, 1, 2)	1502.933616
13	(3, 1, 1)	1503.042612
15	(3, 1, 3)	1503.864485
7	(1, 1, 3)	1504.297660
11	(2, 1, 3)	1504.302200
12	(3, 1, 0)	1504.427366
9	(2, 1, 1)	1506.640148
8	(2, 1, 0)	1519.174140
4	(1, 1, 0)	1534.418781
0	(0, 1, 0)	1572.833556


```
## Running the SARIMA model and getting the model diagnostics

TS_AutoARIMA = sm.tsa.statespace.SARIMAX(endog=TS_Train['PM2.5'],order=(0,1
,1))

TS_AutoARIMA = TS_AutoARIMA.fit()
print(TS_AutoARIMA.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      PM2.5   No. Observations:                  148
Model:                 SARIMAX(0, 1, 1)   Log Likelihood                -748.106
Date:                  Wed, 21 Oct 2020   AIC                           1500.213
Time:                          16:08:33   BIC                           1506.194
Sample:                      03-03-2013   HQIC                          1502.643
                           - 12-27-2015
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.8065      0.053    -15.113      0.000      -0.911      -0.702
sigma2      1530.4319    151.526     10.100      0.000    1233.446    1827.418
===================================================================================
Ljung-Box (L1) (Q):                   0.36   Jarque-Bera (JB):                31.52
Prob(Q):                              0.55   Prob(JB):                         0.00
Heteroskedasticity (H):               1.43   Skew:                             1.02
Prob(H) (two-sided):                  0.22   Kurtosis:                         4.00
===================================================================================
```

According to the result, *SARIMA*(0,1,1) is the indicated model for the PM2.5 data with AIC= (1500.213) and BIC= (1506.194).

```
## Checking the behaviour of the residuals of the model

TS_AutoARIMA.plot_diagnostics();
```

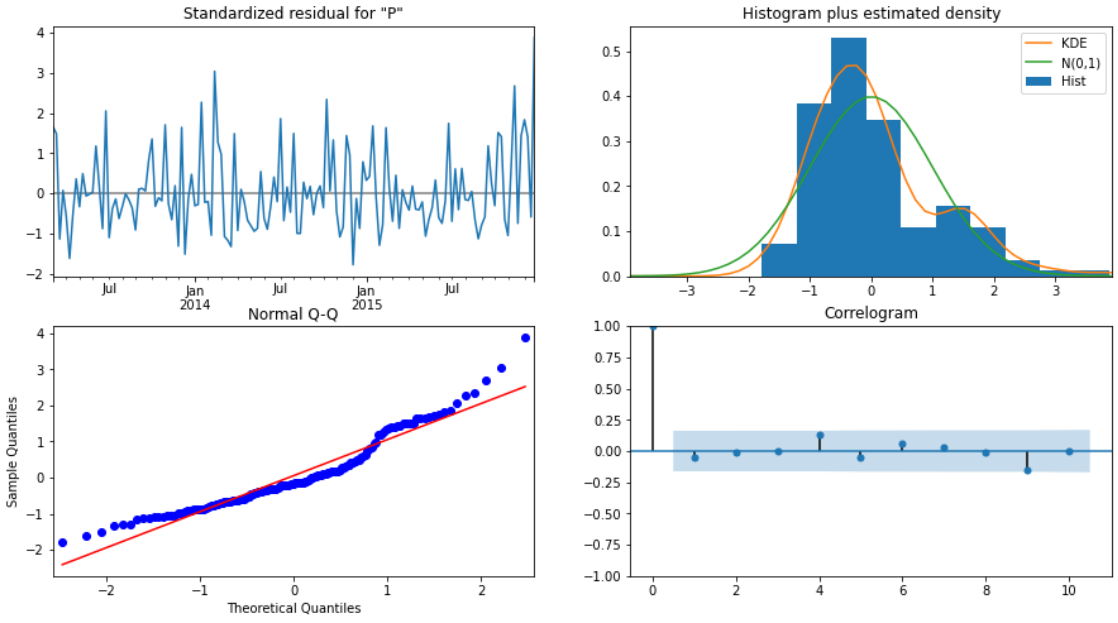


Figure 26: Residual Diagnostics of the (0,1,1) ARIMA Model

Model Validation: ARIMA model

```
# Forecast for the test set duration using the automated ARIMA model with 95% confidence intervals

pred_AutoARIMA = TS_AutoARIMA.get_forecast(steps=len(TS_Test))

# plot the forecast along with the confidence band
axis = TS_Train['PM2.5'].plot()
pred_AutoARIMA.summary_frame(alpha=0.05)['mean'].plot(ax=axis, label='Forecast', alpha=0.7)
axis.fill_between(pred_AutoARIMA.summary_frame(alpha=0.05).index, pred_AutoARIMA.summary_frame(alpha=0.05)['mean_ci_lower'],
                  pred_AutoARIMA.summary_frame(alpha=0.05)['mean_ci_upper'], color='k', alpha=.15)
axis.set_xlabel('Time')
axis.set_ylabel('PM2.5')
plt.legend(loc='best')
plt.title('PM2.5 data forecast using ARIMA (0,1,1)')
plt.grid();
```

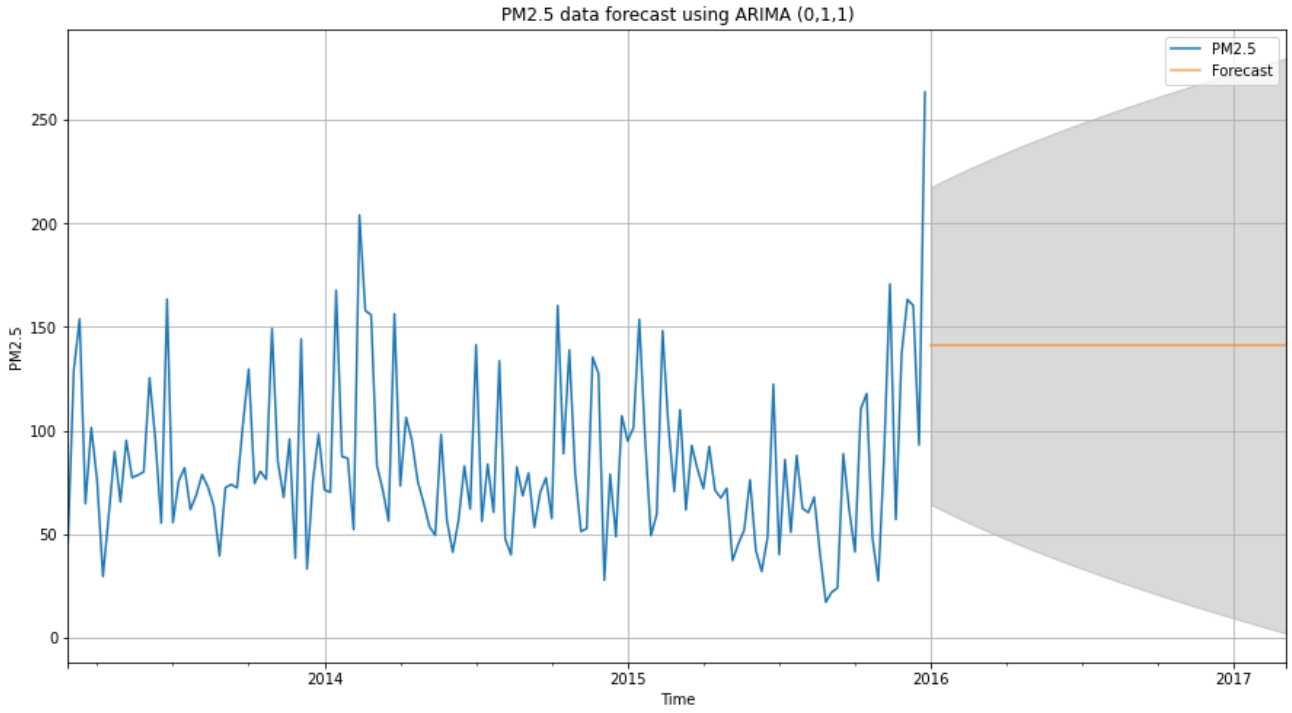


Figure 27: PM2.5 data forecast using ARIMA (0,1,1)

Accuracy measures: ARIMA (0,1,1)

```
## Plotting only the forecast and the test data

TS_Test['PM2.5'].plot()
pred_AutoARIMA.summary_frame()['mean'].plot()
plt.grid()
plt.title('PM2.5: Actual vs Forecast - SARIMA Model')
plt.xlabel('Time')
plt.legend(['Actual Data','Forecasted Data']);
```

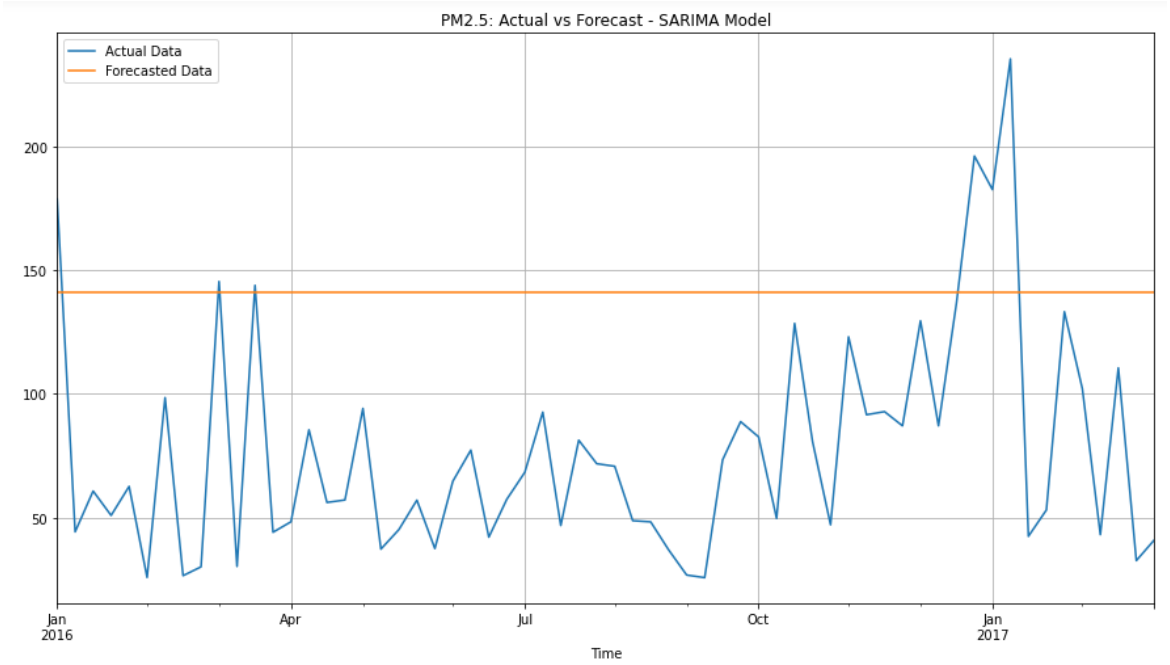


Figure 28: Plot Actual vs. Forecasted PM2.5 data using ARIMA(0,1,1) for 2016-2017 years

```
## Accuracy Measures for ARIMA(0,1,1)

RMSE = mean_squared_error(TS_Test['PM2.5'], pred_AutoARIMA.predicted_mean,s
quared=False)
MAPE = mean_absolute_percentage_error(TS_Test['PM2.5'], pred_AutoARIMA.pred
icted_mean)

print("Accuracy Measures: RMSE:", RMSE, "and MAPE:", MAPE,'%')
```

```
Accuracy Measures: RMSE: 77.9299323862566 and MAPE: 146.7665624636378 %
```

Here, we have used the AIC to select the best ‘p’ and ‘q’ values for the ARIMA models. We can also decide the ‘p’ and ‘q’ values based on lags where the ACF and the PACF cuts-off at a particular confidence level (usually 95% confidence level is taken).

It is clear that ARIMA model does not provide good approximation to the data. An effort will be made to fit an ARIMAX model using a set of covariates, temperature, dew point, rainfall amount and wind speed to improve accuracy of the model.

6.6. **ARIMAX(p,d,q)** Model

Case Study continued:

```
## Here we have taken the range of p and q to be within 0 to 3. We can change this if need
be. The range of p and q has been defined in the last loop and we will be
using the same parameter values

ARIMAX_AIC = pd.DataFrame(columns=['param', 'AIC'])
ARIMAX_AIC
```

```
for param in pdq:#running a loop within the pdq parameters defined by itertools
    ARIMAX_model = sm.tsa.statespace.SARIMAX(endog=TS_Train['PM2.5'].values,
order=param,exog=TS_Train[['TEMP','DEWP','RAIN','WSPM']]).fit()#fitting the ARIMA
model using the parameters from the loop
    ARIMAX_AIC = ARIMAX_AIC.append({'param':param, 'AIC': ARIMAX_model.aic}, igno
re_index=True)
    #appending the AIC values and the model parameters to the previously created
data frame
    #for easier understanding and sorting of the AIC values
```

```
## Sorting the parameters of the ARIMAX models to get the parameters which give u
s the lowest AIC value
```

```
ARIMAX_AIC.sort_values(by=['AIC']).head()
```

	param	AIC
3	(0, 1, 3)	1443.447369
9	(2, 1, 1)	1444.277646
1	(0, 1, 1)	1445.652262
13	(3, 1, 1)	1445.996375
7	(1, 1, 3)	1446.022486


```
TS_ARIMAX = sm.tsa.statespace.SARIMAX(endog=TS_Train['PM2.5'],
exog=TS_Train[['TEMP','DEWP','RAIN','WSPM']],order=(0,1,3))

TS_ARIMAX = TS_ARIMAX.fit()
print(TS_ARIMAX.summary())
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                      PM2.5   No. Observations:                  148
Model:                 SARIMAX(0, 1, 3)   Log Likelihood                -713.724
Date:                  Wed, 21 Oct 2020   AIC                           1443.447
Time:                          16:09:27   BIC                           1467.371
Sample:                      03-03-2013   HQIC                          1453.168
                           - 12-27-2015
Covariance Type:                    opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
TEMP          -6.1775      1.211     -5.103      0.000      -8.550      -3.805
DEWP           4.8216      1.200      4.017      0.000       2.469       7.174
RAIN         -85.0729     32.607     -2.609      0.009    -148.981     -21.165
WSPM         -23.6595      8.693     -2.722      0.006     -40.698      -6.621
ma.L1         -0.8043      0.082     -9.822      0.000      -0.965      -0.644
ma.L2         -0.1442      0.105     -1.370      0.171      -0.350       0.062
ma.L3          0.2269      0.088      2.573      0.010       0.054       0.400
sigma2       948.6626    101.910      9.309      0.000     748.922    1148.403
==========================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                14.30
Prob(Q):                              0.89   Prob(JB):                         0.00
Heteroskedasticity (H):               2.29   Skew:                             0.71
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.54
==========================================================================================

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

According to the result, *ARIMAX* (0,1,3) is the indicated model for the PM2.5 data with AIC= (1443.447) and BIC= (1467.371).

```
## Checking the residual behaviour of the above model

TS_ARIMAX.plot_diagnostics();
```

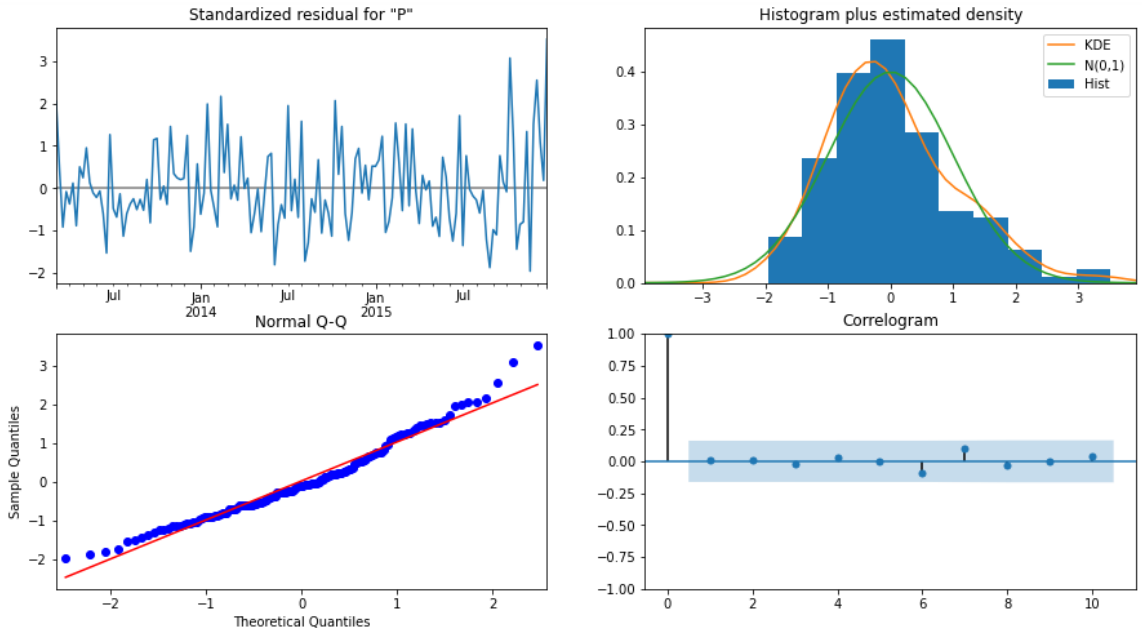



Figure 29: Residual Diagnostics of the (0,1,3) ARIMAX Model

Model Validation: ARIMAX model

```
# Forecast for the test set duration using the automated ARIMA model with 95% confidence intervals

pred = TS_ARIMAX.get_forecast(steps=len(TS_Test),exog=TS_Test[['TEMP','DEWP','RAIN','WSPM']])


# plot the forecast along with the confidence band

axis = TS_Train['PM2.5'].plot()
pred.summary_frame(alpha=0.05)['mean'].plot(ax=axis, label='Forecast', alpha=0.7)
axis.fill_between(pred.summary_frame(alpha=0.05).index, pred.summary_frame(alpha=0.05)['mean_ci_lower'],
                  pred.summary_frame(alpha=0.05)['mean_ci_upper'], color='k', alpha=.15)
axis.set_xlabel('Time')
axis.set_ylabel('PM2.5')
plt.legend(loc='best')
plt.title('PM2.5 data forecast using ARIMAX (0,1,3)')
plt.grid();
```

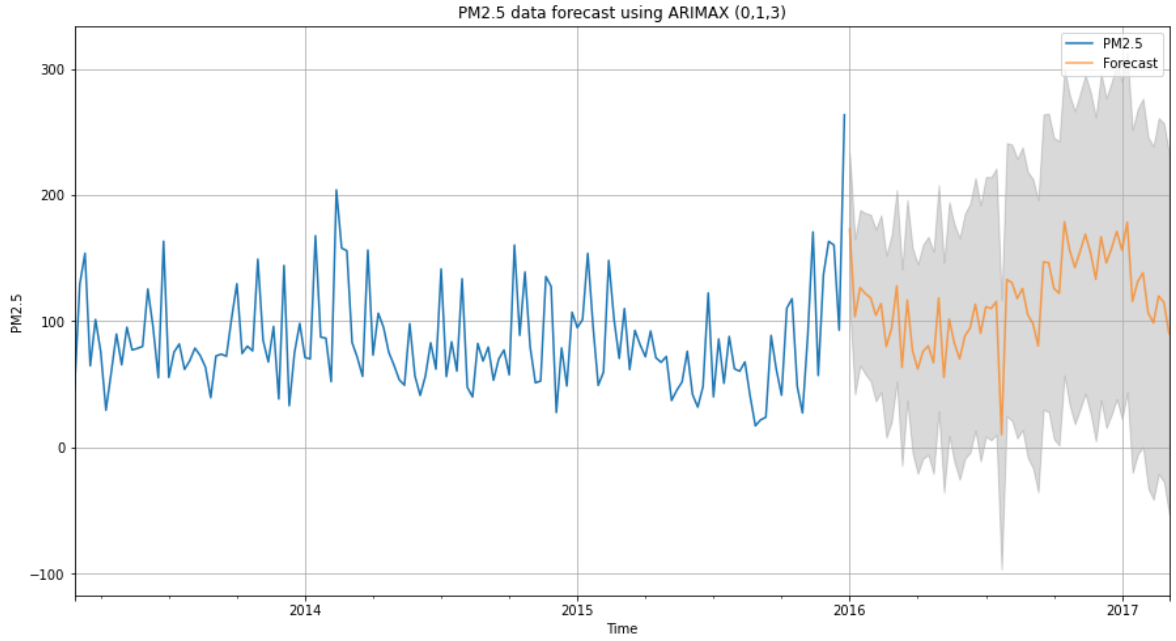



Figure 30: PM2.5 data forecast using ARIMAX (0,1,3)

Accuracy measures: ARIMAX (0,1,3)

```
## Plotting only the forecast and the test data

TS_Test['PM2.5'].plot()
pred.summary_frame()['mean'].plot()
plt.grid()
plt.title('PM2.5: Actual vs Forecast - ARIMAX Model (0,1,3)')
plt.xlabel('Time')
plt.legend(['Actual Data','Forecasted Data']);
```

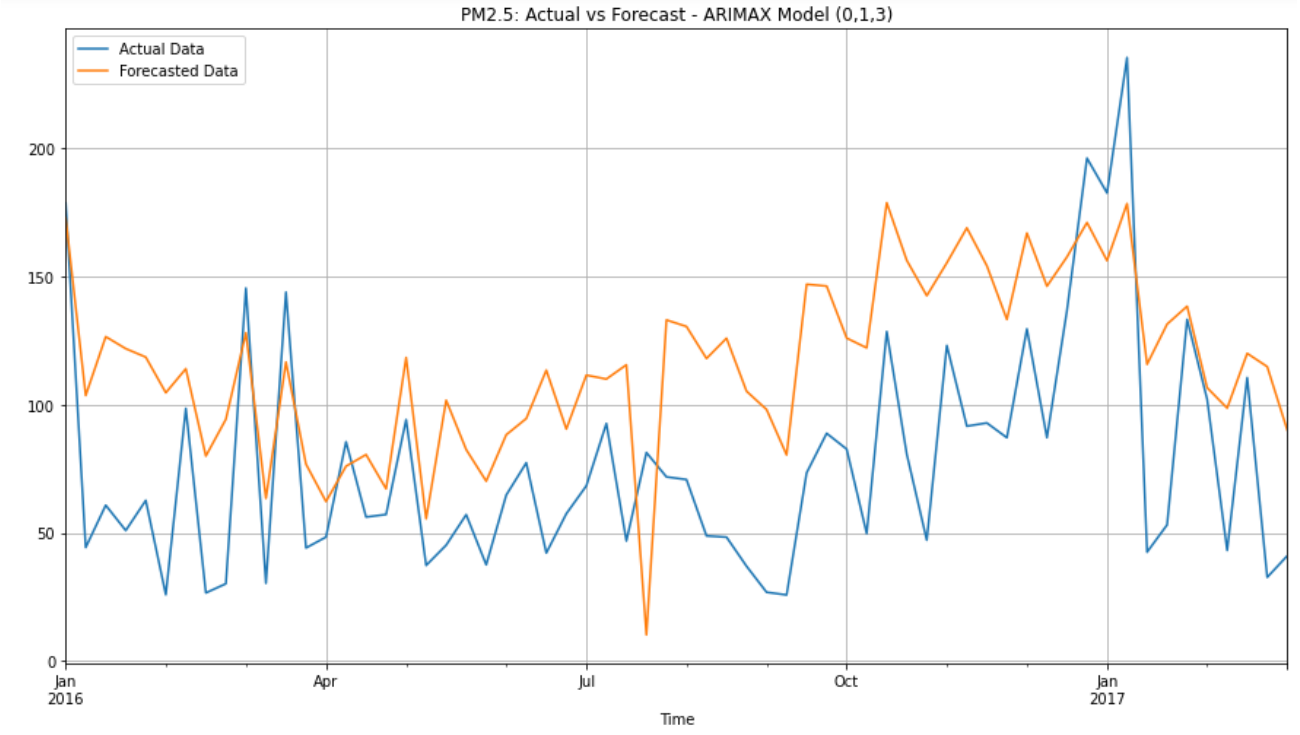


Figure 31: Plot Actual vs. Forecasted PM2.5 data using ARIMAX (0,1,3) for 2016-2017 years


```
## Accuracy Measures for ARIMA(0,1,3)

RMSE = mean_squared_error(TS_Test['PM2.5'],pred.predicted_mean,squared=False)
MAPE = mean_absolute_percentage_error(TS_Test['PM2.5'],pred.predicted_mean)

print("Accuracy Measures: RMSE:", RMSE, "and MAPE:", MAPE,'%')
```

```
Accuracy Measures: RMSE: 52.21777001299539 and MAPE: 88.92919488274941 %
```

Here, we have chosen the model parameters (p and q) using the lowest AIC. But we can definitely go back and investigate the lags at which the ACF and the PACF cuts-off and take p and q values accordingly for the ARIMAX model.

Table 4: Comparison Table

	Test RMSE	Test MAPE
ARIMA(0,1,1)	77.929932	146.766562
ARIMAX(0,1,3)	52.217770	88.929195

Note that none of the above models show small MAPE. However, it is to be noted that with the addition of an exogenous variable, there is a reduction of 40% in the MAPE.

Since no future data for exogenous variables is available, forecast into the future for ARIMAX models (beyond the time stamps of the test set) is not possible. The seasonal parameters (with the appropriate seasonal frequency) may be added to the ARIMAX model to make it a SARIMAX model.

7. Further Study

Time series is a vast area and quite complex models may be applicable here. In the previous sections only two elementary but useful methods have been discussed. In this section a few more complex variations are mentioned.

7.1. Cyclical Component

While decomposing a time series three components are mentioned: trend, seasonality and irregularity. There is a fourth component called cyclical component. This may be taken as a seasonality whose frequency is more than one year. There is a special method of analysis called spectral decomposition to address cyclicality in a time series. The tool used is known as periodogram.

7.2. ARCH and GARCH Models

ARIMA models assume that the variances are constant over the entire duration of a time series. An ARCH (Auto Regressive Conditionally Heteroscedastic) model takes into account changing variance in a series. Although usually an increasing variance is the case for applicability of ARCH model, it can also be used to describe a situation in which there may be short periods of high volatility. In fact, gradually increasing variance connected to a gradually increasing mean level might be better handled by transforming the variable. ARCH models have applicability in the context of econometric and financial time series, such as amount of investments or stock prices. Often the residuals obtained after fitting an ARIMA model may show heteroscedasticity, thereby indicating that an ARCH model may be more appropriate for such series. A further extension of ARCH model is known as a GARCH model.

7.3. Other Variations

Other variations in time series modeling includes time series regression when both response and predictors are measured as a time series and the errors are assumed to follow an AR structure. This is often easier to interpret than an ARIMAX model.

The same concept can be extended to a lagged regression where lags of response is regressed on lags on predictors. The cross-correlation function (CCF) is used to identify possible models.

In the present day context of Artificial Neural Networks, various Deep Neural Network models are also used to forecast time series data. Especially the Recurrent Neural Network (RNN) and Long

Short Term Memory (LSTM) models are finding use in varied fields of Time Series. These models give us a better accuracy in a majority of the cases but these being black-box models, interpretative power is not very high.

7.4. Multivariate Time Series

Recall that so far only univariate time series has been forecasted. In reality there may be many series which are inter-dependent. Many such applications can be thought of in application of stock market, GDP or other growth series. These are known as multivariate time series. Methods such as Vector Auto Regressive Integrated Moving Average (VARIMA) have been developed to address multivariate time series. In case of a bivariate time series, the second time-series is used to facilitate the prediction of the first time series. Concepts such as the causality (e.g., Granger Causality) is introduced on the top of the VARIMA to understand if there is a causal relationship between the two time-series in question.

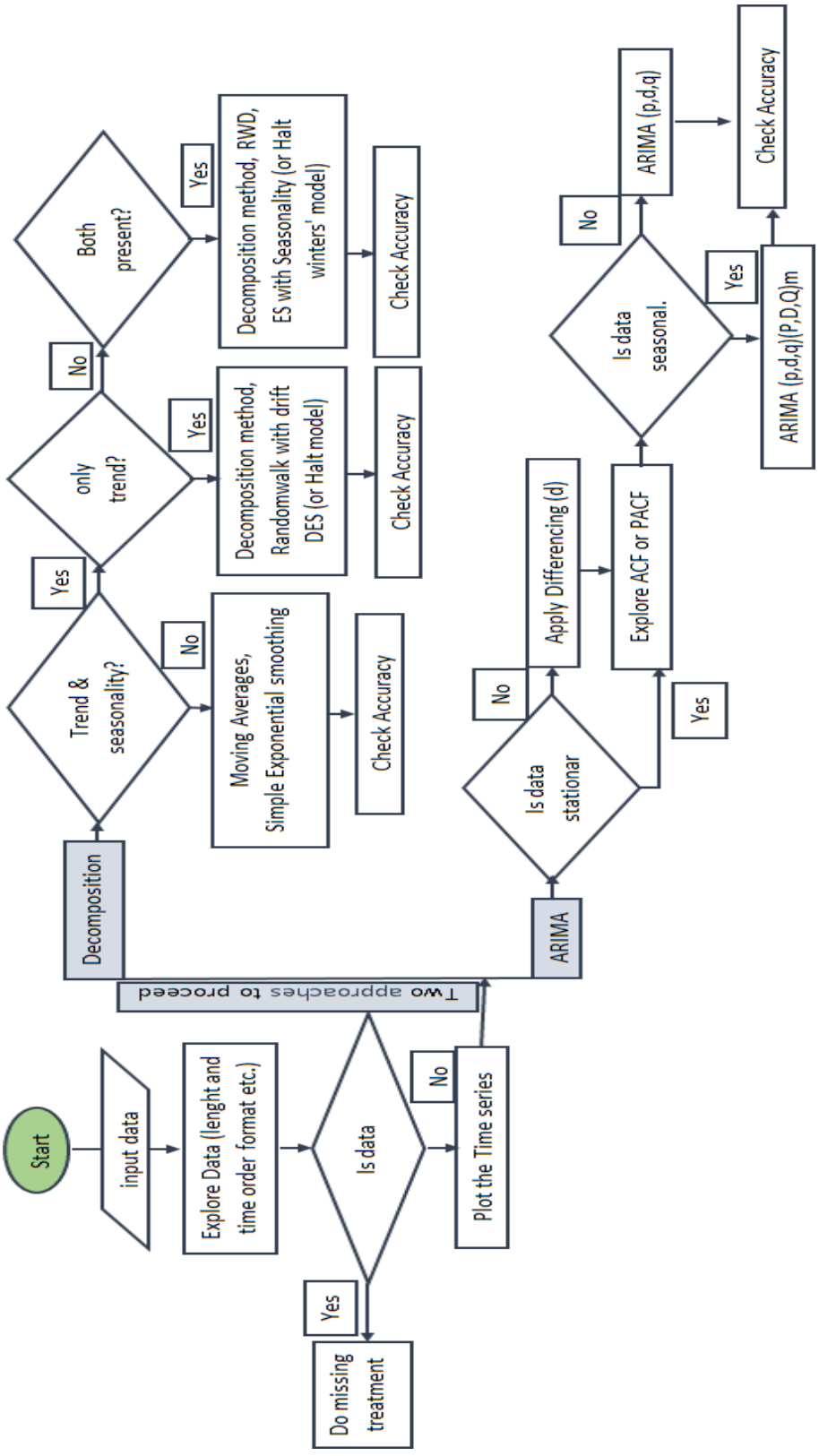


Figure 32: Flow chart of Time Series

Appendix:

Libraries	**Versions**
Pandas	1.0.5
Numpy	1.19.0
Matplotlib	3.2.1
Seaborn	0.10.1
Statsmodels	0.12.0
Sklearn	0.23.1

References:

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts. Accessed Link: https://otexts.org/fpp2/

Tsay, R. S. (2005). *Analysis of financial time series* (Vol. 543). John Wiley & Sons.

Mills, T. C., & Patterson, K. D. (2015). Modelling the trend: the historical origins of some modern methods and ideas. *Journal of Economic Surveys*, *29*(3), 527-548.

Klein, J. L., & Klein, D. (1997). Statistical visions in time: a history of time series analysis, 1662-1938. Cambridge University Press.

Nau, R., (2018), Statistical forecasting: notes on regression and time series analysis, Fuqua School of Business, Duke University, Accessed link: http://people.duke.edu/~rnau/411home.htm

Hyndman, R., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). Forecasting with exponential smoothing: the state space approach. Springer Science & Business Media.

Coghlan, A. (2015). A little book of R for time series. Disponível em: https://media. readthedocs. org/pdf/a-little-book-of-r-for-time-series/latest/a-little-book-of-r-for-time-series. pdf>. Acesso em, 10.

Tsay, R. S. (2014). An introduction to analysis of financial data with R. John Wiley & Sons.

Dagum, E. B., & Bianconcini, S. (2016). Seasonal adjustment methods and real time trend-cycle estimation. Springer.

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time series analysis: Forecasting and control (5th ed). Hoboken, New Jersey: John Wiley & Sons.

