**What makes Time Series Special?**

As the name suggests, TS is a collection of data points collected at **constant time intervals**. These are analyzed to determine the long term trend so as to forecast the future or perform some other form of analysis. But what makes a TS different from say a regular regression problem? There are 2 things:

1. It is **time dependent**. So, the basic assumption of a linear regression model that the observations are independent doesn’t hold in this case.
2. Along with an increasing or decreasing trend, most TS have some form of **seasonality trends**, i.e. variations specific to a particular time frame. For example, if you see the sales of a woolen jacket over time, you will invariably find higher sales in winter seasons.

**How to Check Stationarity of a Time Series?**

A TS is said to be stationary if its **statistical properties** such as mean, variance remain **constant over time**. But why is it important? Most of the TS models work on the assumption that the TS is stationary. Intuitively, we can sat that if a TS has a particular behaviour over time, there is a very high probability that it will follow the same in the future. Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

Stationarity is defined using very strict criterion. However, for practical purposes we can assume the series to be stationary if it has constant statistical properties over time, ie. the following:

1. constant mean
2. constant variance
3. an autocovariance that does not depend on time.

# How to make a Time Series Stationary?

Though stationarity assumption is taken in many TS models, almost none of practical time series are stationary. So statisticians have figured out ways to make series stationary, which we’ll discuss now. Actually, its almost impossible to make a series perfectly stationary, but we try to take it as close as possible.

Lets understand what is making a TS non-stationary. There are 2 major reasons behind non-stationaruty of a TS:  
1. **Trend** – varying mean over time. For eg, in this case we saw that on average, the number of sales was growing over time.  
2. **Seasonality** – variations at specific time-frames. eg people might have a tendency to buy furniture in a particular month because of pay increment or festivals.

The underlying principle is to model or estimate the trend and seasonality in the series and remove those from the series to get a stationary series. Then statistical forecasting techniques can be implemented on this series. The final step would be to convert the forecasted values into the original scale by applying trend and seasonality constraints back.

### Eliminating Trend and Seasonality

The simple trend reduction techniques discussed before don’t work in all cases, particularly the ones with high seasonality. Lets discuss two ways of removing trend and seasonality:

1. **Differencing** – taking the differece with a particular time lag
2. **Decomposition** – modeling both trend and seasonality and removing them from the model.

#### Differencing

One of the most common methods of dealing with both trend and seasonality is differencing. In this technique, we take the difference of the observation at a particular instant with that at the previous instant. This mostly works well in improving stationarity.

#### Decomposing

In this approach, both trend and seasonality are modeled separately and the remaining part of the series is returned.

Project: Forecasting and Predicting the Furniture Sales

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data.

This dataset consists of daily sales data of various products at a superstore.

We will need to apply Time Series (Auto Regressive Integrated Moving Average - ARIMA) to build model to predict and forecast the sales of furniture for the next one year i.e., predict future values based on previously observed values. We have a 4-year furniture sales data.

# Data Preparation

We remove unwanted columns that is not needed and check missing values. Aggregate sales data by date and finally index it with the time series data.

# Feature Engineering

We check the stationarity of the data and decide the next step to be taken. Also decompose the data for further clarification and apply the time series model on the data.

# Model Comparison

We perform parameter selection to find optimal set of parameters that yields the best performance for the model.

# Model Selection

We compare predicted value to the real values and set the forecast from the start to the end of the data.

# ****Installing the required packages****

import warnings

import itertools

import pandas as pd

import numpy as np

import statsmodels.api as sm

import matplotlib

import matplotlib.pyplot as plt

warnings.filterwarnings("ignore")

plt.style.use('fivethirtyeight')

matplotlib.rcParams['axes.labelsize']=14

matplotlib.rcParams['xtick.labelsize']=12

matplotlib.rcParams['ytick.labelsize']=12

matplotlib.rcParams['text.color']='k'

# ****Reading the Time Series Data****

*#read the dataset*

furniture = pd.read\_csv("Furniture.csv")

*#run the 1st 6 rows and all columns of the dataset*

furniture.head()

*#Checking the dimension of the time series data*

furniture.shape

*#Checking the str/character type variable*

furniture.describe(include = 'O')

*#Checking the numeric and continuous variable*

furniture.describe(include = 'float64')

*#Checking the numeric and integer variable*

furniture.describe(include = 'int64')

*#Info command to have a glance on the data types and the missing values*

furniture.info()

*#Check the time spam*

furniture['Order Date'].min(), furniture['Order Date'].max()

cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State', 'Postal Code', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product Name', 'Quantity', 'Discount', 'Profit']

furniture.drop(cols, axis = 1, inplace = True)

furniture = furniture.sort\_values('Order Date')

*#checking null values*

furniture.isnull().sum()

furniture.head()

furniture = furniture.groupby("Order Date")['Sales'].sum().reset\_index()

furniture.head()

*#using the pd to\_datetime we convert the order of date format so that python treat date as date not object*

furniture["Order Date"] = pd.to\_datetime(furniture["Order Date"])

furniture.set\_index("Order Date", inplace = True)

furniture.index

furniture.head()

furniture.info()

y = furniture["Sales"].resample('MS').mean() *#MS mean Month Start*

y['2017']

y

# ****Visually checking the time series for trend and other components****

y.plot(figsize=(15,6))

plt.show()

**The plot clearly indicates that the time series has seasonality pattern. The sales are always low at the beginning of the year and high at the end of the year. There is always an upward trend within any single year with a couple of low months in the mid of the year**.

# ****Checking Stationarity****

**our first step in time-series analysis should be to check whether there is any evidence of a trend or seasonal effects and, if there is, remove them. Augmented Dickey-Fuller(ADF) statistic is one of the more widely used statistic test to check whether your time series is stationary or non-stationary. It uses an autoregressive model and optimizes an information criterion across multiple different lag values.**

**The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary(has some time-dependent structure). The alternate hypothesis(rejecting the null hypothesis) is that the time series is stationary.**

# ****Performing the Dicky Fuller Test****

from pandas import Series

from statsmodels.tsa.stattools import adfuller

result = adfuller(y)

print("ADF Statistic: **%f**" % result[0]) *#%f means float*

print('P-value: **%f**' % result[1])

print('Critical Values:')

for key, value **in** result[4].items():

print('**\t%s**: **%.3f**' % (key, value)) *#%.3f means float with 3 decimal point*

**This suggest that we can reject the null hypothesis with a significance level of less than 1% (i.e. a low propability that the result is a statistical fluke). Rejecting the null hypothesis means that the process has no unit root, and in turn that the time series is stationary or does not have time-dependent structure.**

**The p-value is 0.000009, which is way below the threshold (0.05). Hence the null-hypothesis is rejected. It suggest the time series does not have a unit root, meaning it is stationary.**

# ****Decomposing****

**Decomposing the time series into three distinct components: trend, seasonality,and noise**

from statsmodels.tsa.seasonal import seasonal\_decompose

decomposition = seasonal\_decompose(y)

plt.plot(y, label = 'Original')

plt.legend(loc = 'best')

trend = decomposition.trend

plt.show()

plt.plot(trend, label = "Trend")

plt.legend(loc = "best")

seasonal = decomposition.seasonal

plt.show()

plt.plot(seasonal, label = 'Seasonal')

plt.legend(loc = 'best')

residual = decomposition.resid

plt.show()

plt.plot(residual, label = 'Residual')

plt.legend(loc = 'best')

**By visualizing the decomposition components of the original time series we can say that the sales of furniture is unstable, along with its ARIMA(p,d,q)**

# Forecasting using the best AR I MA(p,d,q) and Seasonality's(P,D,Q)

**Time Series Forecasting using ARIMA. We will use ARIMA for forecasting our time series. ARIMA is also denoted as ARIMA(p,d,q)**

p = d = q = range(0, 2)

pdq = list(itertools.product(p,d,q))

seasonal\_pdq = [(X[0], X[1], X[2], 12) for X **in** list(itertools.product(p, d, q))]

print('Examples of paramter combination for Seasonal ARIMA....')

print('SARIMAX: **{}** x **{}**'.format(pdq[1], seasonal\_pdq[1]))

print('SARIMAX: **{}** x **{}**'.format(pdq[1], seasonal\_pdq[2]))

print('SARIMAX: **{}** x **{}**'.format(pdq[2], seasonal\_pdq[3]))

print('SARIMAX: **{}** x **{}**'.format(pdq[2], seasonal\_pdq[4]))

from pylab import rcParams *#Param means Parameter*

for param **in** pdq:

for param\_seasonal **in** seasonal\_pdq:

try:

mod = sm.tsa.statespace.SARIMAX(y, order=param,

seasonal\_order=param\_seasonal,

enforce\_stationarity=False,

enforce\_invertibility= False)

results = mod.fit()

print('ARIMA**{}**x**{}**12- AIC:**{}**'.format(param, param\_seasonal, results.aic))

except:

continue

mod = sm.tsa.statespace.SARIMAX(y,

order=(1, 1, 1),

seasonal\_order=(1, 1, 0, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])

*#set forecasts to start at 2017–01–01 to the end of the data to forecast*

pred = results.get\_prediction(start=pd.to\_datetime('2017-01-01'), dynamic=False)

pred\_ci = pred.conf\_int()

ax = y['2014':].plot(label='observed')

pred.predicted\_mean.plot(ax=ax, label='One-step ahead Forecast',alpha=1, figsize=(14, 7))

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.2)

ax.set\_xlabel('Date')

ax.set\_ylabel('Furniture Sales')

plt.legend()

plt.show()

y\_forecasted = pred.predicted\_mean

y\_truth = y['2017-01-01':]

mse = ((y\_forecasted - y\_truth) \*\* 2).mean()

print('The Root Mean Squared Error of our forecasts is **{}**'.format(round(np.sqrt(mse), 2)))

**Matplotlib** allows you to adjust the **transparency** of a graph plot using the **alpha** attribute. By default, **alpha**=1. If you want to make the graph plot more transparent, then you can make **alpha** less than 1, such as 0.5 or 0.25. If you want to make the graph plot less transparent, then you can make **alpha** greater than 1.

**In statistics, the mean squared error (MSE) of an estimator measures the average of the squares of the errors - that is, the average squared difference between the estimated value and what is estimated. The MSE is a measure of the quality of an estimator - it is always non - negative, and the smaller the MSE, the closer we are to finding the line of best fit.**

**Root Mean Square Error (RMSE) tells us that our model was able to forecast the average daily furniture sales in the test set within 151.64 of the real sales. Our furniture daily sales range from around 400 to over 1200. This is a pretty good model so far. Mean Absolute Percentage Error and Mean absolute Deviation we can also check instead of RMSE.**

pred\_uc = results.get\_forecast(steps=13)

pred\_ci = pred\_uc.conf\_int()

ax = y.plot(label='observed', figsize=(14, 7))

pred\_uc.predicted\_mean.plot(ax=ax, label='Forecast')

ax.fill\_between(pred\_ci.index,

pred\_ci.iloc[:, 0],

pred\_ci.iloc[:, 1], color='k', alpha=.25)

ax.set\_xlabel('Date')

ax.set\_ylabel('Furniture Sales')

print(pred\_ci)

plt.legend()

plt.show()

# Diagnostics

1. Errors follows normality
2. Errors should not have auto correlation (ACF, no spikes beyond the limits)
3. Errors should not have any spikes (if the spikes are present, that particular time period, model didn't predict properly)

results.plot\_diagnostics(figsize=(10,8))

plt.show()

The KDE plot of the residuals on the top right is almost similar with the normal distribution.

The qq-plot on the bottom left shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this is a strong indication that the residuals are normally distributed.

The residuals over time (top left plot) don't display any obvious seasonality and appear to be white noise. This is confirmed by the autocorrelation (i.e. correlogram) plot on the bottom right, which shows that the time series residuals have low correlation with lagged versions of itself.