Salad and Go Sales Forecast- By Gaurav Sharma(gsharm16@asu.edu)

1. Introduction

The provided data represents a time series where salad sales on each day is represented against the date for last 3 years. The task is to understand the data in order to come up with underlying assumptions required for building the model and then implementing the model to come up with the forecast of upcoming two weeks.

The analysis is done primarily using Python 3.7.2 on Anaconda-Spyder IDE and Jupyter Notebook) with one initial figure from MS-Excel.

2. Descriptive Analytics (Understanding, Summarization and Cleaning)

The data provided ranges from 2nd January 2016 to 27th January 2019.

2.1 General observations/Missing Values

The sample of the initial data:

Index	date	salescount
0	1/2/16	20
1	1/3/16	13
2	1/4/16	32
3	1/5/16	29
4	1/6/16	30
5	1/7/16	35
6	1/8/16	37
7	1/9/16	21
8	1/10/16	16
9	1/11/16	44
10	1/12/16	31
11	1/13/16	32
12	1/14/16	43

```
In [107]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1116 entries, 0 to 1115
Data columns (total 2 columns):
date          1116 non-null object
salescount    1116 non-null int64
dtypes: int64(1), object(1)
memory usage: 17.5+ KB
```

There are some observations which can be noted by looking at the data which can be seen:

a) There are few values with zero or less sales count value. Either they are bad data or on the particular day, the sales were really bad.

54	2/25/16	0	63	3/5/16	0
357	12/24/16	6			
358	12/25/16	0			

b) There are few missing dates like January 1st, November 22nd, December 24th, and December 25th. It is easy to understand that the mentioned dates are the public holidays and the store was closed on that day.

364	12/31/16	3
365	1/2/17	16

2.1.1 Assumption:

It is not clear that the rows with entries zero and less values are bad inputs by the store or in reality, the data was like that. It is assumed that the mentioned data is correct and the sales on the mentioned day is very low. Hence, there is no change performed to such data.

2.1.2 Handling Missing Values:

There are various possible options to take care of such missing values based on the decision of imputing or not imputing the data.

i) If it is decided to impute the data. The below measures can be taken:
   a) Taking average of the values for a week/month before the date
   b) Taking median of the values for a week/month before the date
   c) Taking the average/median value of the corresponding week or month
   d) Taking the average of K period values before the date where K can take any values from 1,2,3,4….

ii) If it is decided to not impute the data according to some method, because there may be some pattern which we will definitely mis-represent if we impute as per the above-mentioned measures. In such case, we can keep the sales against those dates as zero and in actual, the sales were zero since the store was closed.

Note: I have taken the later case(ii) where I have kept the sales of holidays as zero.

2.2 Adding basic required columns

The additional columns for the day of the week, month of the year and the year are added. It is done to enhance the descriptive abilities of the data because now, we can understand the weekly and monthly patterns of the data.

The sample of the updated data is given below:

```
In [5]: data.head(10)
```

Out[5]:

	date	salescount	day	month	year
0	1/2/16	20	Saturday	January	2016
1	1/3/16	13	Sunday	January	2016
2	1/4/16	32	Monday	January	2016
3	1/5/16	29	Tuesday	January	2016
4	1/6/16	30	Wednesday	January	2016
5	1/7/16	35	Thursday	January	2016
6	1/8/16	37	Friday	January	2016
7	1/9/16	21	Saturday	January	2016
8	1/10/16	16	Sunday	January	2016
9	1/11/16	44	Monday	January	2016

```
In [8]: missing_values = data.isna()
        missing_values
```

Out[8]:

	date	salescount	day	month	year
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
1111	False	False	False	False	False
1112	False	False	False	False	False
1113	False	False	False	False	False
1114	False	False	False	False	False
1115	False	False	False	False	False

1116 rows × 5 columns

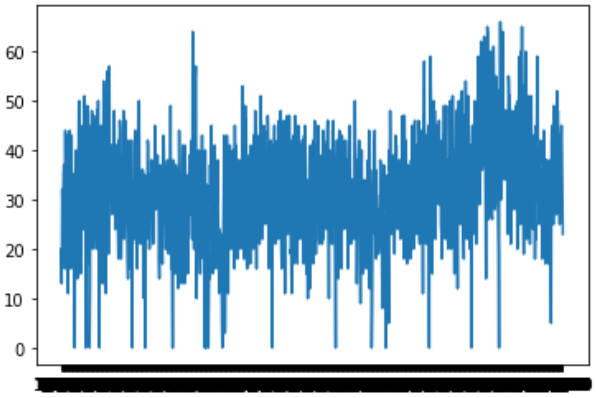
```
In [6]: data.info()
```

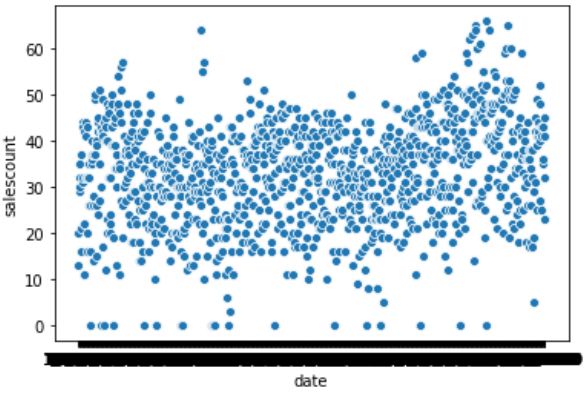
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1116 entries, 0 to 1115
Data columns (total 5 columns):
date          1116 non-null object
salescount    1116 non-null int64
day           1116 non-null object
month         1116 non-null object
year          1116 non-null int64
dtypes: int64(2), object(3)
memory usage: 43.7+ KB
```

2.3 Initial visualizations of the data:

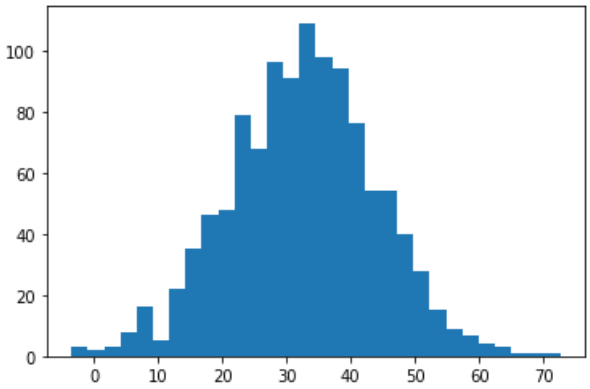
2.3.1 Overall data visualizations

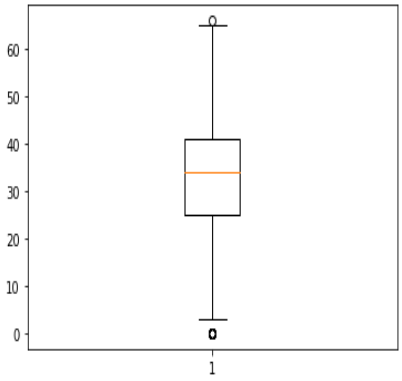
a) Date-wise data-plotting (Line and Scatter Plots)





b) Understanding Frequency and range of sales count (Histogram & Box-and-Whisker Plot):



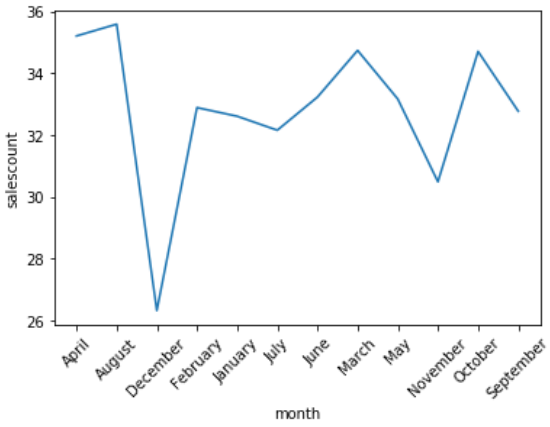


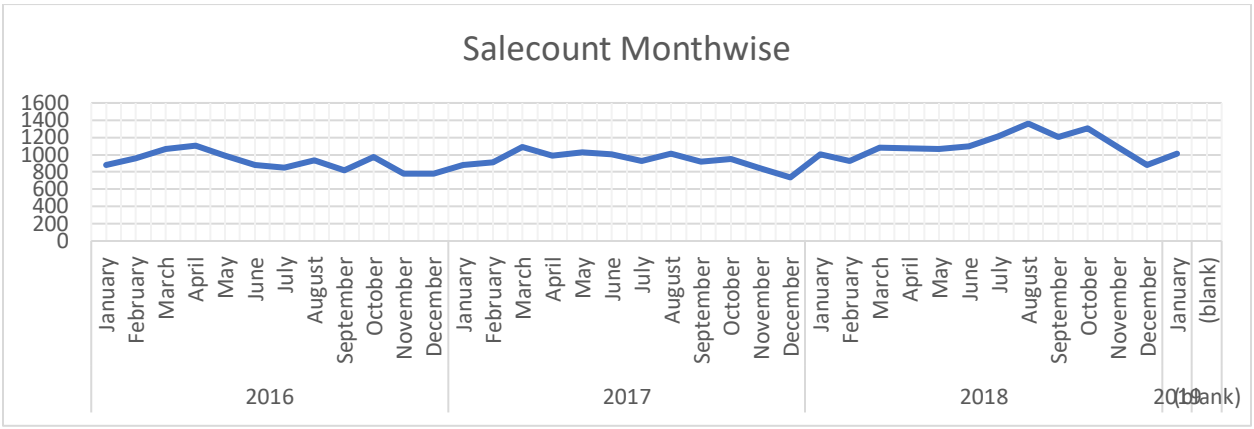
c) Visualizing monthly data among whole timing horizon:

```
In [106]: group_monthwise_mean = salad_sales.groupby(['month'], as_index=False).mean()
          group_monthwise_sum = salad_sales.groupby(['month'], as_index=False).sum()
          group_monthwise_median = salad_sales.groupby(['month'], as_index=False).median()

In [107]: group_monthwise_mean[group_monthwise_mean.columns[0:2]]
```

Out[107]:

	month	salescount
0	April	35.200000
1	August	35.580645
2	December	26.318681
3	February	32.882353
4	January	32.603448
5	July	32.150538
6	June	33.222222
7	March	34.731183
8	May	33.161290
9	November	30.483146
10	October	34.698925
11	September	32.766667





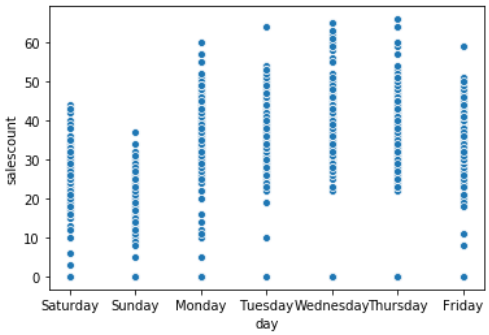
Note: Although from the mean data graph it looks like there is some seasonality in the data monthly but if we visualize monthly data for different years, it is observed that there is no fix ups and downs in specific months. For 2016, highest sales happened in April while in 2017, 2018, it is in March and August respectively. Hence, there is no seasonality as much month wise.

d) Visualizing weekly data among whole timing horizon:

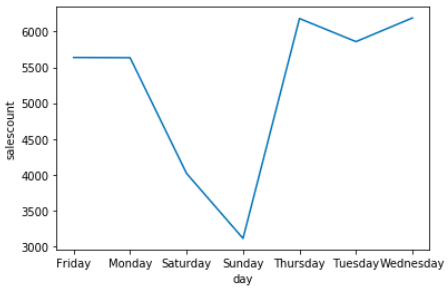
```
In [62]: group_daywise_mean = data.groupby(['day'], as_index=False).mean()
         group_daywise_sum = data.groupby(['day'], as_index=False).sum()
         group_daywise_median = data.groupby(['day'], as_index=False).median()
         group_daywise_mean[group_daywise_mean.columns[0:2]]
```

Out[62]:

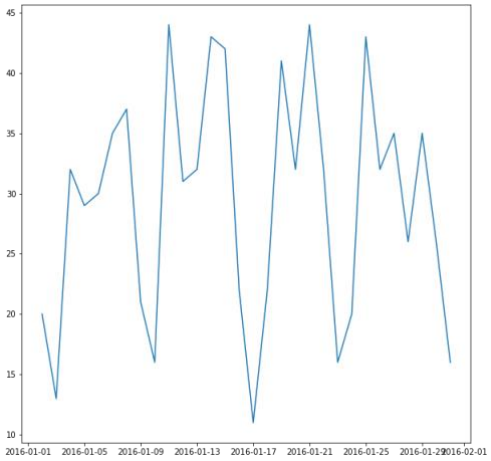
	day	salescount
0	Friday	35.237500
1	Monday	35.664557
2	Saturday	24.950311
3	Sunday	19.425000
4	Thursday	38.886792
5	Tuesday	37.088608
6	Wednesday	38.693750



```
In [66]: plt.figure()
         sns.lineplot(x= 'day', y= 'salescount', data = group_daywise_sum)
         plt.show()
         plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```



Note: Here it is visible

a) from figure b, that the sales on Saturday and Sunday are low. There is no trend visible.
b) From figure b, it is confirmed that over whole timing horizon as well, sales on Saturday and Sunday are low in comparison to other weekdays.
c) The last figure shows the variation within a month for all 4 weeks. **It seems there is a seasonality in the data, and it is of 7 days.**

Caution: **All the above conclusions made about seasonality and trend is made on the basis of visual inspection. It is highly advisable to check all of them statistically.**

3. Assumptions Check

3.1 Augmented Dickey Fuller test

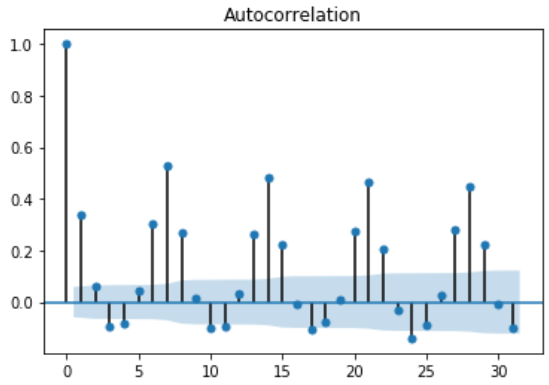
Now, to check for **seasonality** and ensuring the above made conclusions in above section, we will conduct the ADF test and calculate the ADF Statistics, P value and critical values to test the hypothesis.
The test also tells about the **stationarity** of the data. But there are some other checks as well which we will do apart from this for stationarity. The results of the Augmented Dickey Fuller test are mentioned below:

```
In [98]: import statsmodels.api as sm
         from statsmodels.tsa.stattools import adfuller

In [99]: result = adfuller(x)

In [100]: print("ADF Statistics",result[0])
          print("P value", result[1])
          print("critical values")
          #for key, value in result[4].item():
          #print("\t%s, %.3f" % (key,value))
          result[4]
```

```
ADF Statistics -3.2183217533051844
P value 0.018948339208315956
critical values
Out[100]: {'1%': -3.436336023678866,
           '5%': -2.8641831050780513,
           '10%': -2.568177274243656}
```

As the p-value is less than 0.05, we can consider the data as ‘stationary’

We will also plot the autocorrelation graph to understand the seasonality of the data:



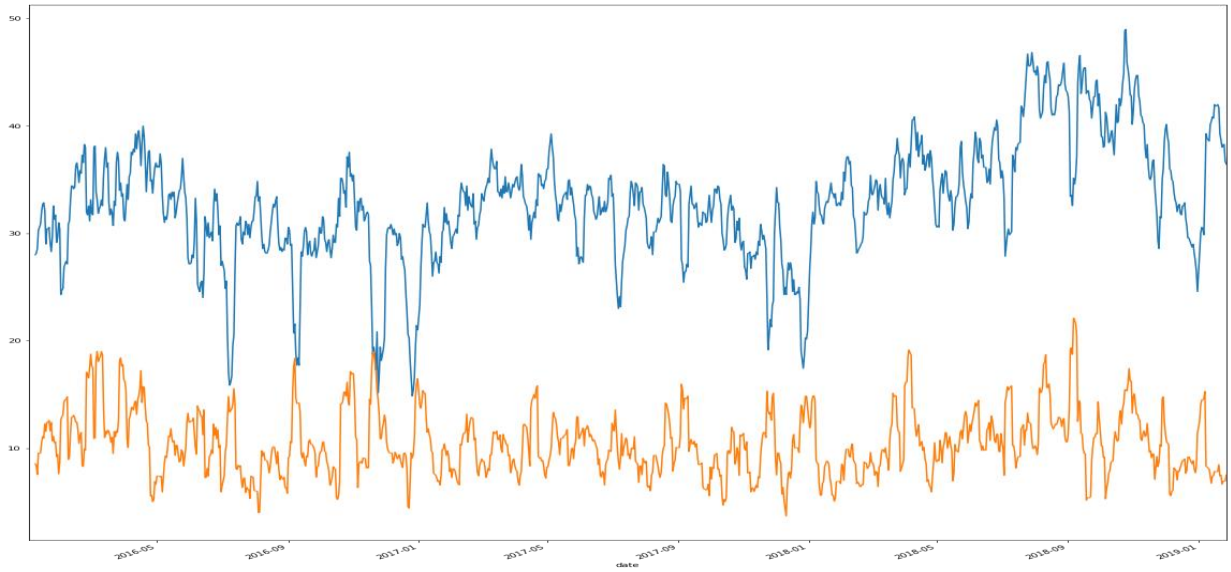
The graph confirms the earlier made conclusion of 7-days seasonality.

3.2 K-Period Mean and Deviation Pattern (Confirming Stationarity)

As a further check for stationarity, we can visualize the pattern of the mean and deviation (checking variance) taken over a certain period. Green color shows the actual data, while blue and orange color shows the mean and standard deviation pattern.

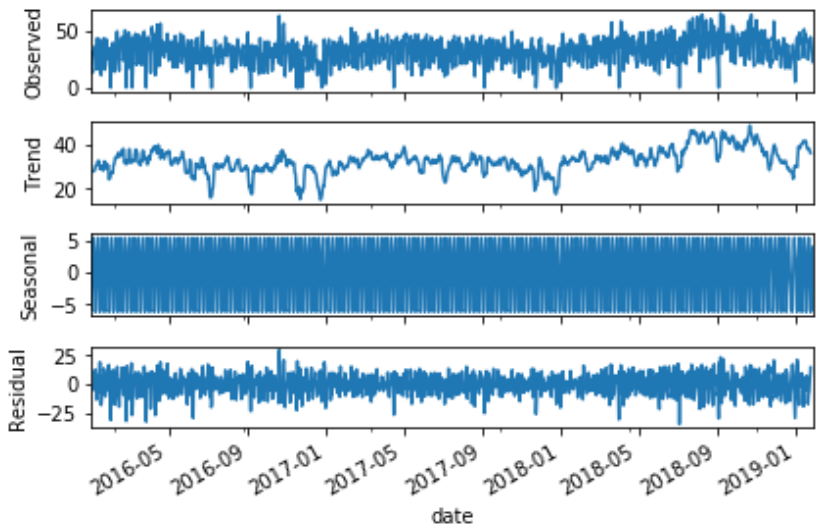
```
plt.figure(figsize=(20,20))
timeseries.rolling(7).mean().plot(label= "7 days mean")
timeseries.rolling(7).std().plot(label= "7 days std dev")
timeseries.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24c80320>
```



As per the above analysis, the data shows no trend as the pattern can be seen as fluctuating.

3.3. Seasonal Decomposition



4. Model Selection

Now, we know that our data is stationary with no trend but has the seasonality of 7 days.

I have given thought over the below model possibilities:

a) Simple Exponential Smoothing (ES)

b) K Period Moving Average (MA)
c) Simple Auto-Regressive Model (AR)
d) Auto-Regressive Integrated Moving Average (ARIMA)

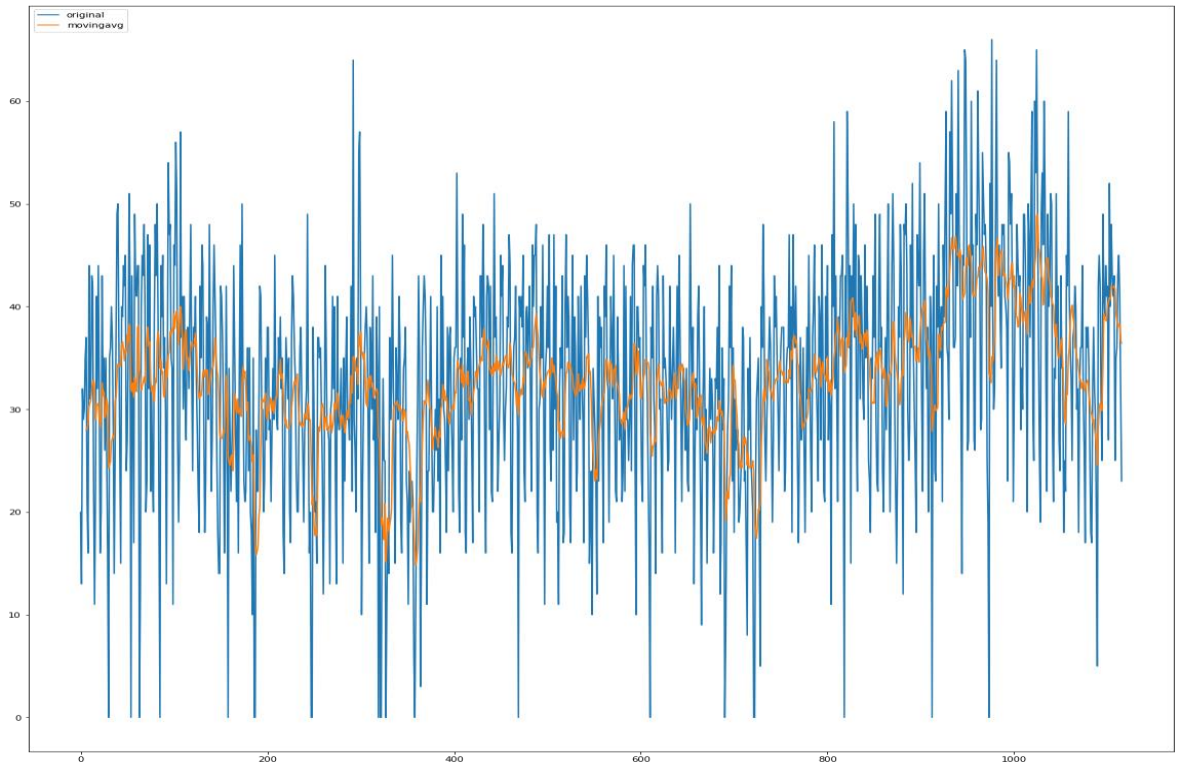
4.1 Why not Simple Exponential Smoothing?

Exponential smoothing methods are appropriate for non-stationary data (i.e. data with a trend and seasonal data). SES only has one component called *level* (with a smoothing parameter denoted as "alpha"). It is a weighted average of the previous level and the current observation. The method is better for Few data points, Irregular data, No seasonality or trend.

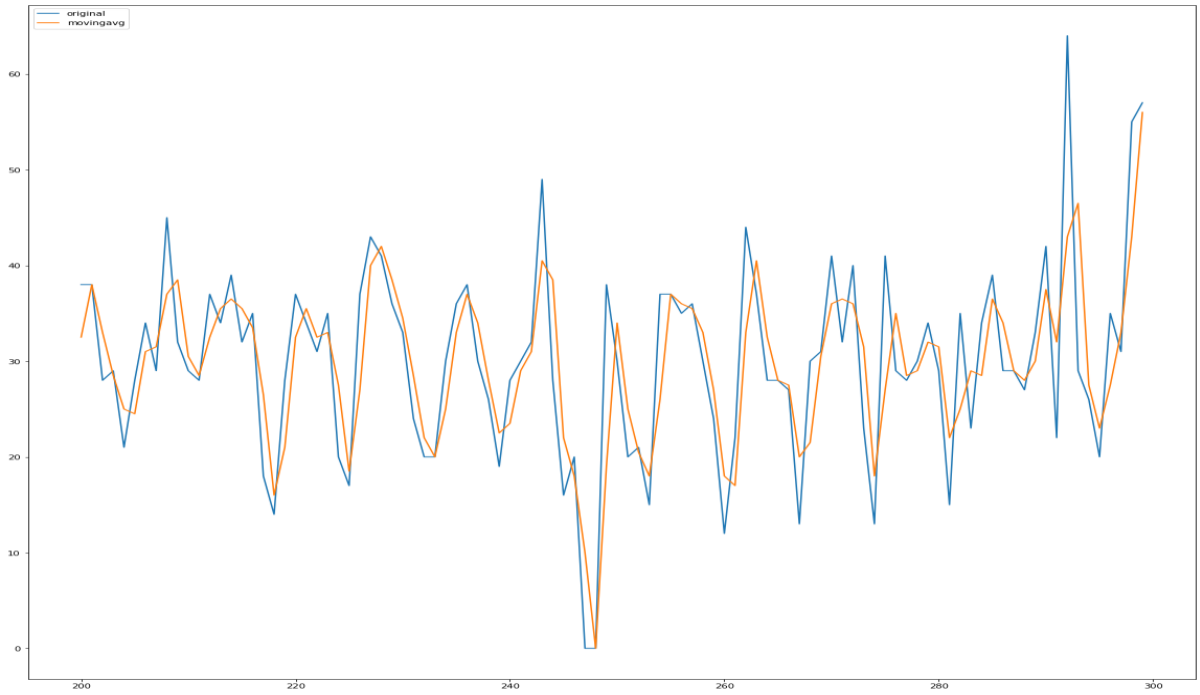
Since we have established that the data is stationary, I have not selected the simple exponential smoothing for the model building.

4.2 Why not Simple Moving Average?

It can be understood by the below fitted model of MA on the given sales data(2-period). The below figure suggests that we are getting almost a similar value as expected.



But looking closely at the data, we will find that almost all the values differs very less from expected.



The issue can be thought as overfitting. While training the model, it gives promising results, but such data behaves very badly in the test case scenario and provide a very large realization error. Hence this method is not good for predicting the results in our case.

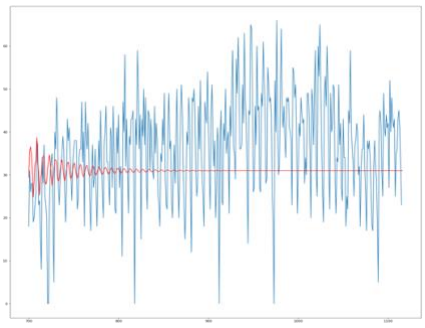
Error Calculations:

```
In [317]: residuals = mov["salescount"]-mov["avg1"]

In [321]: residuals.std()
Out[321]: 6.607571956870912

In [318]: error = mean_squared_error(mov["salescount"].iloc[6:],mov["avg1"].iloc[6:])
          #plt.boxplot(residuals)

In [319]: error
Out[319]: 43.71711711711712
```

4.3 Why not just Auto-regressive model?

The results from the fitted Auto-regressive model are shown below:

```
In [237]: dataset = data["salescount"]
          train = dataset[:700]
          test = dataset[700:]
          prediction = []

In [238]: from statsmodels.tsa.ar_model import AR
          from sklearn.metrics import mean_squared_error

In [239]: model_ar = AR(train)
          model_ar_fit = model_ar.fit()
          prediction = model_ar_fit.predict(start=700, end=1116)

In [243]: plt.figure(figsize=(20,20))
          plt.plot(test)
          plt.plot(prediction, color="red")
```



The accuracy of the model is quite low, and it gives constant value for a large number of test data values.

4.4 Why ARIMA?

ARIMA models are a general class of models to forecast **stationary** time series.

The model contains three parts:

- A weighted sum of lagged ***values*** of the series (***Auto-regressive*** (***AR***) part)
- A weighted sum of lagged ***forecasted errors*** of the series (***Moving-average*** (***MA***) part)
- A ***difference*** of the time series (***Integrated*** (***I***) part)

In our case proved above, the test statistic is below the critical values, we fail to reject the null hypothesis, our series is trend stationary.
As our data has established stationarity, the ARIMA model is a good choice to test.

5. Model Building

5.1 Train-Test Split

First thing before modeling, we split the data in two parts as train and test data. The idea is to build the model on the train data and the prediction will be done on the test data to check the accuracy of the model.

66% data is taken as train and 34% data is taken as test data.

```
In [204]: Y = y.values
          size = int(len(Y) * 0.66)
          train, test = Y[0:size], Y[size:len(Y)]

In [205]: train.shape
Out[205]: (736,)

In [206]: test.shape
Out[206]: (380,)
```

5.2. Fitting the ARIMA Model:

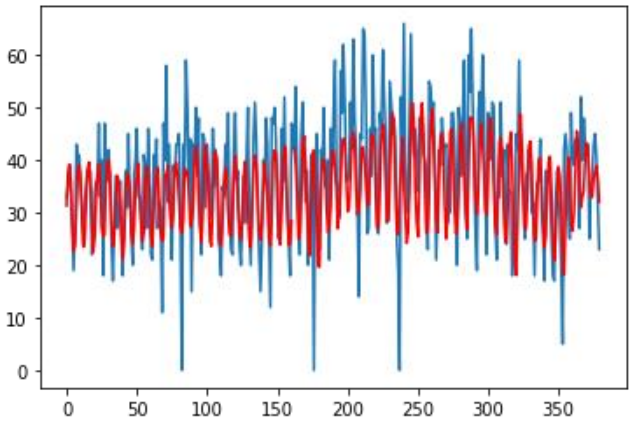
5.2.1 Code Snippet

```
In [207]: history = [m for m in train]
          predictions = list()
          for t in range(len(test)):
              model = ARIMA(history, order=(7,0,3))
              model_fit = model.fit(disp=0)
              output = model_fit.forecast()
              yhat = output[0]
              predictions.append(yhat)
              obs = test[t]
              history.append(obs)
```

The model is fitted on the train data and is predicted on the test data.

5.2.2 Visualization for the predicted vs expected data

The data for the 380 test points is plotted as below:



5.2.3 Code-Run Issue

While running the ARIMA model, there will a problem in calculating hessian matrix. Refer the attached screenshot.

```
history.append(obs)
/Users/gauravsharma/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:492: HessianInversionWarning: Inv
erting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
/Users/gauravsharma/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:492: HessianInversionWarning: Inv
erting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
/Users/gauravsharma/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:492: HessianInversionWarning: Inv
erting hessian failed, no bse or cov_params available
  'available', HessianInversionWarning)
/Users/gauravsharma/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:492: HessianInversionWarning: Inv
erting hessian failed, no bse or cov_params available
```

The issue can be solved by scaling the time series data which will not affect the results as well.

5.3 Parameter Tuning (Grid Search Loop)

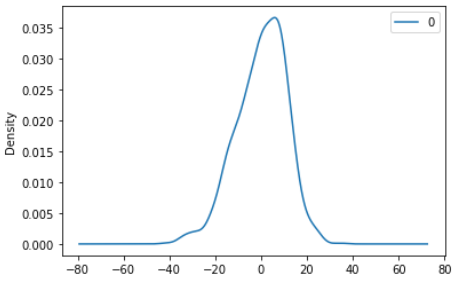
Two methods were adopted for the same. One is the manual search by changing the parameters and second is to create a grid search loop over a range to identify the best possible parameter.

Various combinations of the parameters were tried like (7,0,3), (5,1,0), (0,0,1), (0,1,0), (1,1,1), (1,0,2) with seasonality of 7 as mentioned. Few of them will clearly not give the best results but I just tried them so that I can check the change in values.

The output for few of the above combinations along with the residual plot:

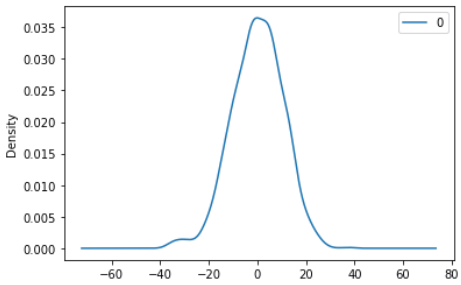
```
: model = ARIMA(y, order=(0,1,1))
  model_fit = model.fit(disp=0)
  print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:           D.salescount   No. Observations:                 1115
Model:                 ARIMA(0, 1, 1)   Log Likelihood               -4252.043
Method:                       css-mle   S.D. of innovations             10.951
Date:                Sat, 01 Feb 2020   AIC                           8510.086
Time:                        10:12:41   BIC                           8525.136
Sample:                             1   HQIC                          8515.776

==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                      0.0059      0.015      0.403      0.687      -0.023       0.035
ma.L1.D.salescount        -0.9562      0.009   -101.454      0.000      -0.975      -0.938
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1            1.0458           +0.0000j            1.0458            0.0000
-----------------------------------------------------------------------------
```




```
: model = ARIMA(y, order=(1,0,2))
  model_fit = model.fit(disp=0)
  print(model_fit.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:             salescount   No. Observations:                 1116
Model:                     ARMA(1, 2)   Log Likelihood               -4214.099
Method:                       css-mle   S.D. of innovations             10.557
Date:                Sat, 01 Feb 2020   AIC                           8438.198
Time:                        10:12:06   BIC                           8463.286
Sample:                             0   HQIC                          8447.682

==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                     32.8982      1.831     17.969      0.000      29.310      36.487
ar.L1.salescount           0.9932      0.004    224.402      0.000       0.985       1.002
ma.L1.salescount          -0.6986      0.028    -24.995      0.000      -0.753      -0.644
ma.L2.salescount          -0.2581      0.027     -9.442      0.000      -0.312      -0.205
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0069           +0.0000j            1.0069            0.0000
MA.1            1.0354           +0.0000j            1.0354            0.0000
MA.2           -3.7424           +0.0000j            3.7424            0.5000
-----------------------------------------------------------------------------
```



The best error values were for the order (1,1,1). The parameters with the **least AIC value** and the MSE (72.7) was chosen.

6. Conclusion

The Forecast for the 14 days (up to one significant digit) comes out as:

Day 1	39.6
Day 2	41.2
Day 3	43.1
Day 4	43.2
Day 5	39.6
Day 6	29.3
Day 7	23.7
Day 8	39.8
Day 9	41.3
Day 10	43.1
Day 11	43.3
Day 12	39.6
Day 13	29.4
Day 14	23.8