\*\*\*\*\*\*\*\*\*\*\* Data preprocessing \*\*\*\*\*\*\*\*\*\*\*\*

1. Sorted the entire data in excel firstly

* Sorted first level by APMC
* Added level and sorted by commodity
* Added level and sorted by date (that included both month and year)

2. This sorting gave me the data of each commodity in a particular APMC with sorted dates.

3. Some of the data was in the capital and some in small letter. To make the data uniform and

decrease the duplicacy due to letter case, I converted the entire APMC and Commodity columns in lower cases using =LOWER() function in excel.

4. Uploaded both the files using pandas

df1 = pd.read\_csv(r'C:\Users\Anirudh Garg\Desktop\socialcops\Monthly\_data\_cmo.csv')

df2 = pd.read\_csv(r'C:\Users\Anirudh Garg\Desktop\socialcops\CMO\_MSP\_Mandi.csv')

5. Some of the data in the modal\_price column were filled with ‘0’ but had either max\_price or min\_price. I filled up the entire ‘0’ data in the modal\_price by the average of max and min price using python code.

for index, row in df1.iterrows():

if row['modal\_price'] == 0:

row['modal\_price'] = (row['max\_price'] + row['min\_price'])/2

6. What I understood from the Objective of your problem statement was to find the fluctuations in the price of a commodity at a particular APMC. After sorting the data as mentioned in the first point, I found a total of 4834 clusters of commodities. (See the shared excel file)

7. After sorting I combined the APMC and commodity column and created a new column apmc\_commodity.

df1["apmc\_commodity"] = df1["apmc"] + "\_" + df1["commodity"]

8. I label encoded the apmc\_commodity column and found that there were total 4834 clusters over which I needed to work.

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

df1['label'] = le.fit\_transform(df1['apmc\_commodity'])

9. After this, I dropped some columns which were not useful.

df1.drop('APMC', axis=1, inplace=True)

df1.drop('Commodity', axis=1, inplace=True)

df1.drop('Year', axis=1, inplace=True)

df1.drop('Month', axis=1, inplace=True)

df1.drop('district\_name', axis=1, inplace=True)

df1.drop('state\_name', axis=1, inplace=True)

10. Filled the missing values in “ msprice” by the mean after label encoding and then grouping by the commodities in python.

df2\_grouped = df2.groupby('label')

import numpy

for group2\_name, df2\_group in df2\_grouped:

df2\_group['msprice'].fillna(df2\_group['msprice'].mean(), inplace=True)

\*\*\*\*\*\*\*\*\*\* Data preprocessing over \*\*\*\*\*\*\*\*\*\*

Now coming to the **first objective**: Test and filter outliers.

Firstly I used the group-by function in python and grouped all the label encoded data. Then I used for loop to iterate over each group and found the mean and standard deviation of “arrival\_in\_qtl” and “modal\_price” of each every group. The rows with the values exclusive of [(mean - 2\*std), (mean + 2\*std)] were removed from the data.

#arrivals\_in\_qtl

import numpy

for group\_name, df\_group in df1\_grouped:

arr = df\_group['arrivals\_in\_qtl']

elements = numpy.array(arr)

mean = numpy.mean(elements, axis=0)

sd = numpy.std(elements, axis=0)

drop0=np.array(arr.index)

drop1=drop0[elements<(mean-2\*sd)]

drop2=drop0[elements>(mean+2\*sd)]

drop1=np.concatenate([drop1,drop2])

df1.drop(drop1,inplace=True,axis=0)

#modal\_price

df1\_grouped2 = df1.groupby('label')

for group\_name2, df\_group2 in df1\_grouped2:

arr = df\_group2['modal\_price']

elements = np.array(arr)

mean = np.mean(elements, axis=0)

sd = np.std(elements, axis=0)

drop0=np.array(arr.index)

drop1=drop0[elements<(mean-2\*sd)]

drop2=drop0[elements>(mean+2\*sd)]

drop1=np.concatenate([drop1,drop2])

df1.drop(drop1,inplace=True,axis=0)

After removing the outliers, 56855 number of rows were remaining in the dataset.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* first objective over \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Now coming to the **Second objective :**

Understand price fluctuations accounting the seasonal effect

* Detect seasonality type (multiplicative or additive) for each cluster of APMC and commodities
* De-seasonalise prices for each commodity and APMC according to the detected seasonality type

1. For this firstly I converted the datatype and format of the date from ‘object’ to ‘datetime64[ns]’.

df1['date'] = pd.to\_datetime(df1['date'])

2. Then I set date as the index of the dataset to plot the time series graphs for the modal\_prices.

df1.set\_index('date', inplace=True)

3. Then I used for loop to plot all the time series plots

for group\_name3, df\_group3 in df1\_grouped3:

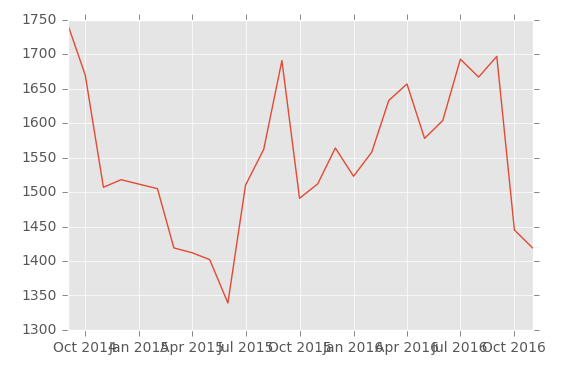
ts = df\_group3['modal\_price']

plt.plot(ts)

plt.show()

This gave 4834 graphs of time series for all commodities at their apmc.

Example graph for Paddy-unhusked commodity at aamgoan



4. I used ‘statsmodels.tsa.stattools’ for finding different statistical values of the time series data like rolling mean and standard deviation.

#Plot rolling statistics:

orig = plt.plot(timeseries, color='blue',label='Original')

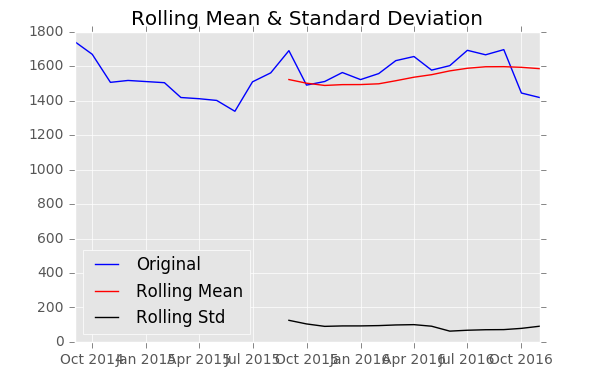
mean = plt.plot(rolmean, color='red', label='Rolling Mean')

std = plt.plot(rolstd, color='black', label = 'Rolling Std')

plt.legend(loc='best')

plt.title('Rolling Mean & Standard Deviation')

plt.show(block=False)



5. Also performed dickey fuller test which gave the parameters like p-value, no. of observations etc. (only some values were under my understanding)

#Perform Dickey-Fuller test:

print 'Results of Dickey-Fuller Test:'

dftest = adfuller(timeseries, autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

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

dfoutput['Critical Value (%s)'%key] = value

print dfoutput

If p-value > 0.05 then the plot was non-stationary

If p-value < 0.05 then the plot was stationary

6. I went through different methods like Moving average and exponentially weighted moving average to remove the trend and smooth the plotted curves but were of no use as we need to remove the seasonality of the data. Also, the data was less in each cluster due to which moving average algorithm did not work properly.

7. I used decomposition function to decompose the plot into three parts i.e. Trend, seasonality and residual. Applied the function using for loop and plotted graphs for all three parts to understand the trend and seasonality.

from statsmodels.tsa.seasonal import seasonal\_decompose

for group\_name3, df\_group3 in df1.groupby('label'):

ts = df\_group3['modal\_price']

decomposition = seasonal\_decompose(ts , filt=None, freq=10)

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

plt.subplot(411)

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

plt.show()

plt.legend(loc='best')

plt.subplot(412)

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

plt.show()

plt.legend(loc='best')

plt.subplot(413)

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

plt.show()

plt.legend(loc='best')

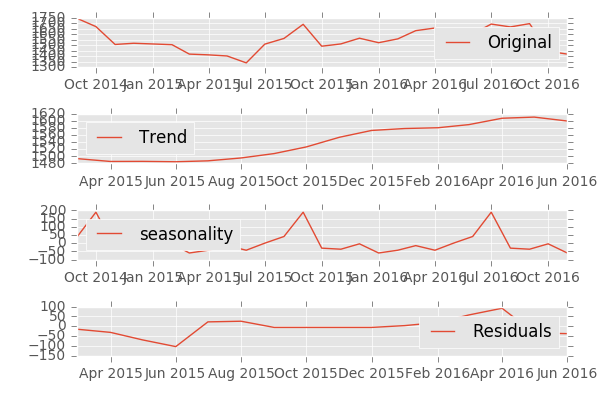
plt.subplot(414)

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

plt.show()

plt.legend(loc='best')

plt.tight\_layout()



8. After this, the major problem faced was to automatically find the type of seasonality using python code and not manually.

And then to find the prices of each commodity after removing the seasonality effect.

If the seasonality is additive we do

New curve = Trend + residual

Or

New curve = Old curve - seasonality.

If the seasonality is multiplicative we do

New curve = Trend \* residual

Or

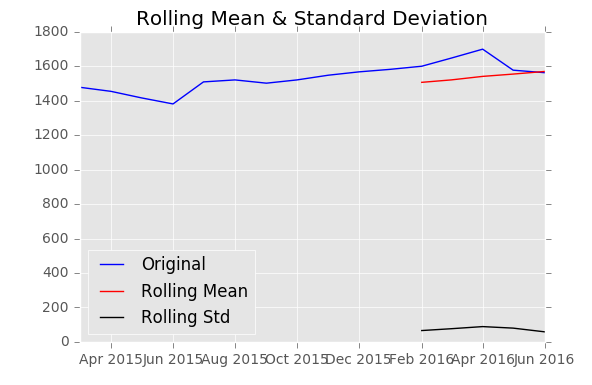
New curve = Old curve / seasonality.

I de-sesonalised the data in the graphs but was unable to find the corresponding prices for each commodity.

ts\_decompose = residual + trend

ts\_decompose.dropna(inplace=True)

test\_stationarity(ts\_decompose)



This was the curve obtained after de seasonalising.

But it was n’t possible to find the seasonality manually by analyzing every curve for more than 4000 clusters due to which I wasn’t able to find the seasonality and final curves for all the clusters.

.