Sales Forecasting of Retail Clothing Product Categories

November 26, 2017

1 Problem Description :

Build a framework that provides monthly forecasts of the 12 months (ie., from Jan 2016 to Dec 2106) for WomenClothing product category alone with influencing factors of the sales such as holiday events, weather changes, macro economic factors etc. using the regression model.

1.0.1 Importing the required packages

```
In [1]: import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from matplotlib import style
        from sklearn import preprocessing
        from sklearn import feature_selection
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_regression
        from sklearn.linear_model import LinearRegression
        from statsmodels.formula.api import OLS
        from sklearn.svm import SVR
        from sklearn.model_selection import GridSearchCV, StratifiedKFold
        from statsmodels.stats.outliers_influence import variance_inflation_factor
        from sklearn.ensemble import RandomForestRegressor
```

1.0.2 Setting the current working directory and loading the files

```
In [2]: os.chdir("I:\DATA-SCIENCE\SalesForecasting")
        sales_data = pd.read_csv("Train.csv")
        holidays_data = pd.read_excel("Events_HolidaysData.xlsx")
        economic_data = pd.read_excel("MacroEconomicData.xlsx")
        weather_data_xlsx = pd.ExcelFile("WeatherData.xlsx")
```

1.0.3 Defining the Error Metric to be used

```
In [3]: def mean_absolute_percentage_error(y_true, y_pred):
            y_true, y_pred = np.array(y_true), np.array(y_pred)
            return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

2 1. Basic Understanding of the Dataset

2.1 1.1 Sales Data

```
In [4]: print("Datatypes of each columns :")
        print(sales_data.dtypes)
```

```
Datatypes of each columns :
Year                           int64
Month                          int64
ProductCategory               object
Sales(In ThousandDollars)    float64
dtype: object
```

```
In [5]: # Removing the instances of 'MenClothing' and 'OtherClothing' from sales data
        sales_data = sales_data.loc[sales_data.ProductCategory == "WomenClothing" ,:]
```

```
In [6]: print("The first 5 instances from the table:")
        sales_data.head()
```

```
The first 5 instances from the table:
```

Out[6]:

	Year	Month	ProductCategory	Sales(In ThousandDollars)
0	2009	1	WomenClothing	1755.0
3	2009	2	WomenClothing	1729.0
6	2009	3	WomenClothing	2256.0
9	2009	4	WomenClothing	2662.0
12	2009	5	WomenClothing	2732.0

```
In [7]: print("The last 5 instances from the table:")
        sales_data.tail()
```

```
The last 5 instances from the table:
```

Out[7]:

	Year	Month	ProductCategory	Sales(In ThousandDollars)
237	2015	8	WomenClothing	3897.0
240	2015	9	WomenClothing	3881.0
243	2015	10	WomenClothing	4372.0
246	2015	11	WomenClothing	4401.0
249	2015	12	WomenClothing	5874.0

2.2 1.2 Holidays Data

```
In [8]: print("Datatypes of each columns :")
        print(holidays_data.dtypes)
```

```
Datatypes of each columns :
Year                     int64
MonthDate      datetime64[ns]
Event                  object
DayCategory            object
dtype: object
```

```
In [9]: print("The first 2 instances from the table:")
        holidays_data.head()
```

```
The first 2 instances from the table:
```

```
Out[9]:    Year  MonthDate                       Event      DayCategory
        0  2009 2001-01-01              New Year's Day  Federal Holiday
        1  2009 2019-01-01  Martin Luther King Jr. Day  Federal Holiday
        2  2009 2014-02-01             Valentine's Day           Event
        3  2009 2016-02-01             Presidents' Day  Federal Holiday
        4  2009 2012-04-01               Easter Sunday           Event
```

2.2.1 1.2.3 Modifying the "MonthDate" column

```
In [10]: # Extracting the date and month from the 'MonthDate' column.
         holidays_data.MonthDate = holidays_data.MonthDate.astype(str).str[2:7]
```

```
In [11]: print("The first 5 instances from the table after processing:")
         holidays_data.head()
```

```
The first 5 instances from the table after processing:
```

```
Out[11]:    Year MonthDate                       Event      DayCategory
         0  2009     01-01              New Year's Day  Federal Holiday
         1  2009     19-01  Martin Luther King Jr. Day  Federal Holiday
         2  2009     14-02             Valentine's Day           Event
         3  2009     16-02             Presidents' Day  Federal Holiday
         4  2009     12-04               Easter Sunday           Event
```

```
In [12]: print("The categories and their count in 'DayCategory' column:")
         print(holidays_data.DayCategory.value_counts())
```

```
The categories and their count in 'DayCategory' column:
Federal Holiday    88
Event              62
Name: DayCategory, dtype: int64
```

```
In [13]: print(holidays_data.Event.value_counts())
```

```
Martin Luther King Jr. Day     8
Labor Day                      8
Easter Sunday                  8
Thanksgiving Day               8
Columbus Day (Most regions)    8
Memorial Day                   8
New Year's Day                 8
Presidents' Day                8
Halloween                      8
Christmas Eve                  8
New Year's Eve                 8
Veterans Day                   8
Independence Day               8
Mother's Day                   8
Father's Day                   8
Valentine's Day                8
Christmas Day                  8
'Independence Day' observed    3
'Christmas Day' observed       3
Thomas Jefferson's Birthday    3
Election Day                   2
'New Year's Day' observed      2
Day After Christmas Day        1
Name: Event, dtype: int64
```

2.3 1.3 Macroeconimic Data

```
In [14]: print("Datatypes of each columns :")
         print(economic_data.dtypes)
```

```
Datatypes of each columns :
Year-Month                                                          object
Monthly Nominal GDP Index (inMillion$)                             float64
Monthly Real GDP Index (inMillion$)                                float64
CPI                                                                float64
PartyInPower                                                        object
unemployment rate                                                  float64
CommercialBankInterestRateonCreditCardPlans                        float64
Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan  float64
Earnings or wages  in dollars per hour                             float64
AdvertisingExpenses (in Thousand Dollars)                           object
Cotton Monthly Price - US cents per Pound(lbs)                     float64
Change(in%)                                                        float64
Average upland planted(million acres)                              float64
Average upland harvested(million acres)                            float64
yieldperharvested acre                                               int64
Production (in  480-lb netweright in million bales)                float64
```

```
Mill use  (in  480-lb netweright in million bales)                  float64
Exports                                                             float64
dtype: object
```

In [15]:
```
# Printing the statistics of the macro economic table
economic_data.describe(include = 'all')
```

Out[15]:

	Year-Month	Monthly Nominal GDP Index (inMillion$)	\
count	96	96.000000	
unique	96	NaN	
top	2015 - Apr	NaN	
freq	1	NaN	
mean	NaN	16490.078125	
std	NaN	1427.554038	
min	NaN	14317.372922	
25%	NaN	15210.701514	
50%	NaN	16422.454368	
75%	NaN	17772.032416	
max	NaN	19015.393408	

	Monthly Real GDP Index (inMillion$)	CPI	PartyInPower	\
count	96.000000	96.000000	96	
unique	NaN	NaN	1	
top	NaN	NaN	Democrats	
freq	NaN	NaN	96	
mean	15548.932194	252.372552	NaN	
std	773.076316	9.290857	NaN	
min	14345.676097	233.402000	NaN	
25%	14919.668252	242.474500	NaN	
50%	15442.996869	254.680500	NaN	
75%	16262.237629	260.381500	NaN	
max	16918.050624	265.421000	NaN	

	unemployment rate	CommercialBankInterestRateonCreditCardPlans	\
count	96.000000	96.000000	
unique	NaN	NaN	
top	NaN	NaN	
freq	NaN	NaN	
mean	7.442708	12.519479	
std	1.766397	0.722375	
min	4.600000	11.820000	
25%	5.675000	11.950000	
50%	7.750000	12.220000	
75%	9.000000	13.057500	
max	10.000000	14.260000	

	Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan	\

count	96.000000
unique	NaN
top	NaN
freq	NaN
mean	10.447604
std	0.560651
min	9.450000
25%	10.030000
50%	10.470000
75%	10.940000
max	11.440000

	Earnings or wages in dollars per hour \
count	96.000000
unique	NaN
top	NaN
freq	NaN
mean	23.793229
std	1.147167
min	21.960000
25%	22.797500
50%	23.800000
75%	24.772500
max	26.040000

	AdvertisingExpenses (in Thousand Dollars) \
count	96
unique	12
top	?
freq	85
mean	NaN
std	NaN
min	NaN
25%	NaN
50%	NaN
75%	NaN
max	NaN

	Cotton Monthly Price - US cents per Pound(lbs) Change(in%) \	
count	96.000000	96.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	91.034479	0.615521
std	33.613974	6.919387
min	51.500000	-23.590000
25%	70.377500	-2.400000
50%	83.905000	0.350000

75%	93.322500	3.482500
max	229.670000	22.850000

	Average upland planted(million acres)	\
count	96.000000	
unique	NaN	
top	NaN	
freq	NaN	
mean	10.647698	
std	1.728418	
min	8.398000	
25%	9.296000	
50%	10.260500	
75%	11.412250	
max	14.431000	

	Average upland harvested(million acres)	yieldperharvested acre	\
count	96.000000	96.000000	
unique	NaN	NaN	
top	NaN	NaN	
freq	NaN	NaN	
mean	8.851885	791.843750	
std	1.102847	25.753621	
min	7.345000	747.000000	
25%	7.586000	772.750000	
50%	9.156500	790.000000	
75%	9.654500	807.000000	
max	10.577000	869.000000	

	Production (in 480-lb netweright in million bales)	\
count	96.000000	
unique	NaN	
top	NaN	
freq	NaN	
mean	14.616010	
std	2.007579	
min	11.751000	
25%	12.551000	
50%	14.959000	
75%	16.250000	
max	18.375000	

	Mill use (in 480-lb netweright in million bales)	Exports
count	96.000000	96.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	3.556104	11.061615

```
std                                                 0.158263    1.710573
min                                                 3.275000    8.500000
25%                                                 3.410000    9.742500
50%                                                 3.572500   10.637500
75%                                                 3.675000   11.493750
max                                                 4.170000   15.280000
```

In [16]:
```
economic_data.isnull().sum()
```

Out[16]:
```
Year-Month                                                          0
Monthly Nominal GDP Index (inMillion$)                              0
Monthly Real GDP Index (inMillion$)                                 0
CPI                                                                 0
PartyInPower                                                        0
unemployment rate                                                   0
CommercialBankInterestRateonCreditCardPlans                         0
Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan   0
Earnings or wages  in dollars per hour                              0
AdvertisingExpenses (in Thousand Dollars)                           0
Cotton Monthly Price - US cents per Pound(lbs)                      0
Change(in%)                                                         0
Average upland planted(million acres)                               0
Average upland harvested(million acres)                             0
yieldperharvested acre                                              0
Production (in  480-lb netweright in million bales)                 0
Mill use  (in  480-lb netweright in million bales)                  0
Exports                                                             0
dtype: int64
```

2.3.1 1.3.1 Categorical Variables

In [17]:
```
print(economic_data["PartyInPower"].value_counts(), "\n")
print("NOTE: The attribute 'PartyInPower' has only one category for all the instances

# Droping the column 'PartyInPower'
economic_data.drop('PartyInPower', axis=1,inplace=True)
```

```
Democrats    96
Name: PartyInPower, dtype: int64

NOTE: The attribute 'PartyInPower' has only one category for all the instances and hence is no
```

In [18]:
```
print(economic_data["AdvertisingExpenses (in Thousand Dollars)"].value_counts(), "\n"
print("NOTE: The attribute 'AdvertisingExpenses (in Thousand Dollars)' has 88% of miss

# Droping the column 'AdvertisingExpenses (in Thousand Dollars)'
economic_data.drop('AdvertisingExpenses (in Thousand Dollars)', axis=1,inplace=True)
```

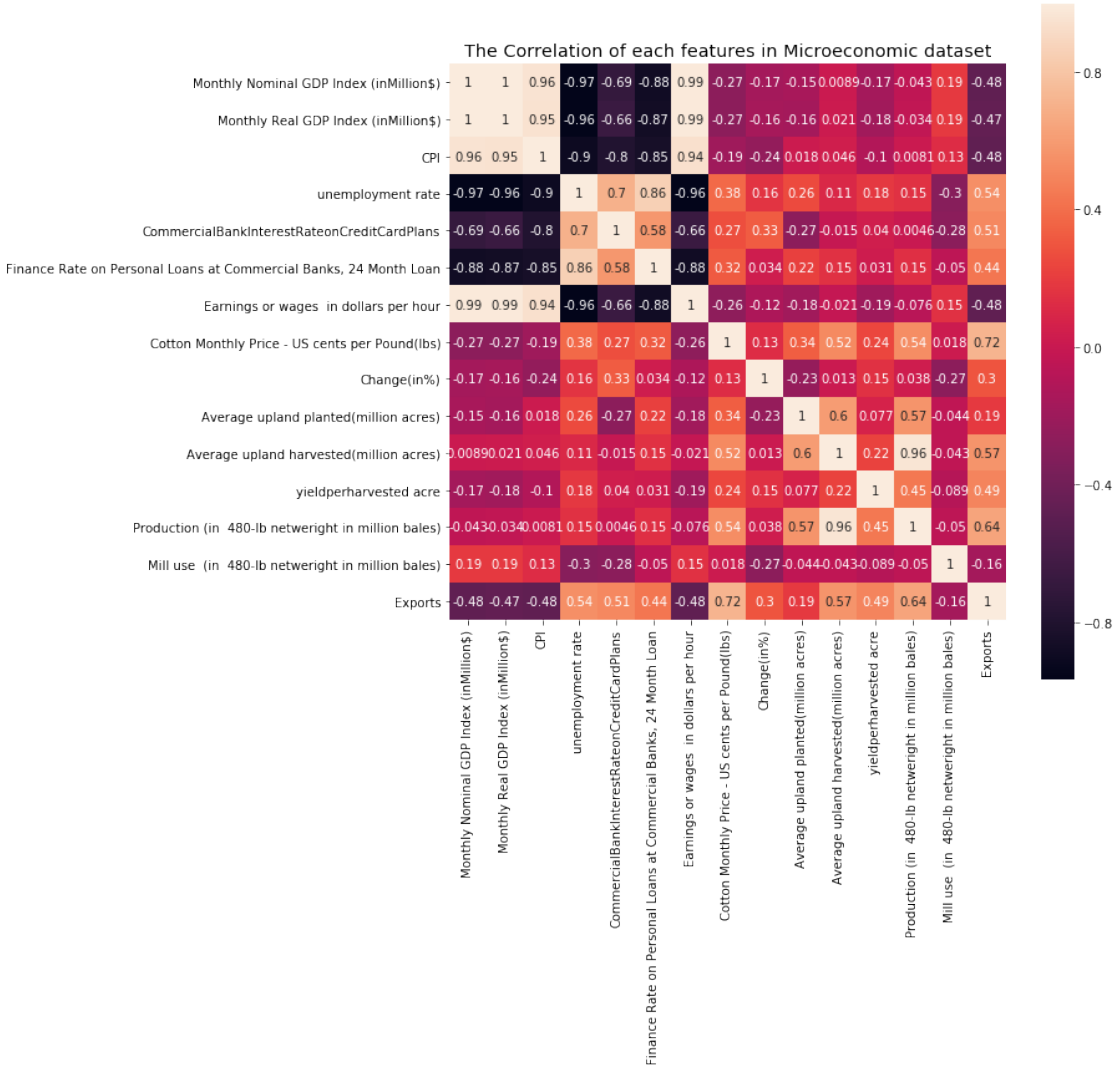
```
?      85
221     1
150     1
201     1
137     1
200     1
248     1
214     1
165     1
116     1
183     1
208     1
Name: AdvertisingExpenses (in Thousand Dollars), dtype: int64

NOTE: The attribute 'AdvertisingExpenses (in Thousand Dollars)' has 88% of missing values whic
```

2.3.2 1.3.2 Numerical Variables

In [19]:
```
# Calculating the correlation of each variables
correlation = economic_data.corr()

### Ploting the correlation
%matplotlib inline
plt.figure(figsize =(10,10))
sns.heatmap(data= correlation, annot =True, square= True)
plt.title('The Correlation of each features in Microeconomic dataset', fontsize= 'x-la
plt.show()
```



2.4 1.4 Weather Data

2.4.1 1.4.1 Creating a single dataframe from the given tables.

```
In [20]: # Adding each sheet from the excel file to a list of dataframes.
         weather_data_list = []
         for i in range(0,len(weather_data_xlsx.sheet_names)):
             weather_data_list.append(weather_data_xlsx.parse(weather_data_xlsx.sheet_names[i])

             # Setting the Year column with proper values
             weather_data_list[i].Year = weather_data_xlsx.sheet_names[i]

         # Shifting each row one step upwards in the dataframe.
         weather_data_list[5].loc[:,"Temp high (řC)":"WeatherEvent"] = weather_data_list[5].loc
```

```
        .shift(-1)
        weather_data_list[5] = weather_data_list[5][:-1].copy()

        # Combining list of weather datas into a single dataframe
        weather_data = pd.DataFrame()
        for df in weather_data_list:
            weather_data = pd.concat([weather_data,df])
```

```
In [21]: print("The first 5 instances from the table:")
         weather_data.head()
```

```
The first 5 instances from the table:
```

Out[21]:

	Year	Month	Day	Temp high (řC)	Temp avg (řC)	Temp low (řC)	\
0	2009	Jan	1.0	-3	-6	-9	
1	2009	Jan	2.0	1	-2	-5	
2	2009	Jan	3.0	3	1	-2	
3	2009	Jan	4.0	6	1	-4	
4	2009	Jan	5.0	6	5	3	

	Dew Point high (řC)	Dew Point avg (řC)	Dew Point low (řC)	Humidityă(%) high	\
0	-16	-17	-19	54	
1	-3	-7	-17	78	
2	-5	-9	-13	72	
3	-10	-12	-13	55	
4	-1	-5	-16	62	

	...	Sea Level Press.ă(hPa) avg	Sea Level Press.ă(hPa) low	\
0	...	1023	1015	
1	...	1012	1007	
2	...	1015	1008	
3	...	1017	1015	
4	...	1014	1013	

	Visibilityă(km) high	Visibilityă(km) avg	Visibilityă(km) low	\
0	16	16	16	
1	16	13	2	
2	16	16	16	
3	16	16	16	
4	16	16	16	

	Windă(km/h) low	Windă(km/h) avg	Windă(km/h) high	Precip.ă(mm) sum	\
0	37	18	60	0	
1	27	10	48	T	
2	27	16	42	T	
3	32	12	40	0	

```
4              23              11               34                T

  WeatherEvent
0          NaN
1         Snow
2          NaN
3          NaN
4          NaN

[5 rows x 23 columns]
```

In [22]:
```
print("The last 5 instances from the table:")
weather_data.tail()
```

```
The last 5 instances from the table:
```

Out[22]:

	Year	Month	Day	Temp high (řC)	Temp avg (řC)	Temp low (řC)	\
361	2016	Dec	27.0	16	10	4	
362	2016	Dec	28.0	4	3	1	
363	2016	Dec	29.0	8	4	1	
364	2016	Dec	30.0	4	3	1	
365	2016	Dec	31.0	7	3	-1	

	Dew Point high (řC)	Dew Point avg (řC)	Dew Point low (řC)	\
361	11	6	-3	
362	-3	-6	-7	
363	7	2	-7	
364	-1	-4	-7	
365	-3	-7	-9	

	Humidityă(%) high	...	Sea Level Press.ă(hPa) avg	\
361	89	...	1012	
362	64	...	1019	
363	96	...	1006	
364	82	...	1006	
365	56	...	1016	

	Sea Level Press.ă(hPa) low	Visibilityă(km) high	Visibilityă(km) avg	\
361	1008	16	16	
362	1014	16	16	
363	1000	16	10	
364	1000	16	14	
365	1012	16	16	

	Visibilityă(km) low	Windă(km/h) low	Windă(km/h) avg	Windă(km/h) high	\
361	14	32	12	53	
362	16	23	8	34	

```
363                   2              24               9               40
364                   1              29              15               47
365                  16              24              11               40

    Precip.ă(mm) sum WeatherEvent
361                0          NaN
362                0          NaN
363             9.91         Rain
364             0.25   Fog , Snow
365                0          NaN

[5 rows x 23 columns]
```

```
In [23]: weather_data.dtypes
```

```
Out[23]: Year                            object
         Month                           object
         Day                            float64
         Temp high (řC)                  object
         Temp avg (řC)                   object
         Temp low (řC)                   object
         Dew Point high (řC)             object
         Dew Point avg (řC)              object
         Dew Point low (řC)              object
         Humidityă(%) high               object
         Humidityă(%) avg                object
         Humidityă(%) low                object
         Sea Level Press.ă(hPa) high     object
         Sea Level Press.ă(hPa) avg      object
         Sea Level Press.ă(hPa) low      object
         Visibilityă(km) high            object
         Visibilityă(km) avg             object
         Visibilityă(km) low             object
         Windă(km/h) low                 object
         Windă(km/h) avg                 object
         Windă(km/h) high                object
         Precip.ă(mm) sum                object
         WeatherEvent                    object
         dtype: object
```

3 2. Indexing all the dataframes with Date as index.

3.1 2.1 Sales Data

```
In [24]: sales_data['Date'] = sales_data.Year.astype(str).str.cat(sales_data.Month.astype(str)
         sales_data['Date'] = pd.to_datetime(sales_data.Date.astype(str) + "-1")
         sales_data.set_index('Date', inplace = True)
         sales_data.drop(["Year","Month"],axis =1, inplace=True)
```

3.2 2.2 Holidays Data

```
In [25]: holidays_data['Date'] = pd.to_datetime(holidays_data.Year.astype(str).str.cat(holidays
         holidays_data.drop(["Year","MonthDate"],axis =1, inplace=True)
         holidays_data.set_index('Date', inplace = True)
```

3.3 2.3 Macroeconomic Data

```
In [26]: economic_data.rename(columns = {'Year-Month':'Date'}, inplace = True)
         economic_data['Date'] = pd.to_datetime(economic_data.Date.astype(str) + "-1")
         economic_data.set_index('Date', inplace = True)
```

3.4 2.4 Weather Data

```
In [27]: weather_data['Date'] = pd.to_datetime(weather_data.Year.astype(str).str.cat(weather_da
                                              .str.cat(weather_data.Day.astype(str), sep='/'))
         weather_data.drop(["Year","Month","Day"],axis =1, inplace=True)
         weather_data.set_index('Date', inplace = True)

In [28]: weather_data.head()
```

Out[28]:

Date	Temp high (řC)	Temp avg (řC)	Temp low (řC)	Dew Point high (řC)
2009-01-01	-3	-6	-9	-16
2009-01-02	1	-2	-5	-3
2009-01-03	3	1	-2	-5
2009-01-04	6	1	-4	-10
2009-01-05	6	5	3	-1

Date	Dew Point avg (řC)	Dew Point low (řC)	Humidityă(%) high
2009-01-01	-17	-19	54
2009-01-02	-7	-17	78
2009-01-03	-9	-13	72
2009-01-04	-12	-13	55
2009-01-05	-5	-16	62

Date	Humidityă(%) avg	Humidityă(%) low	Sea Level Press.ă(hPa) high
2009-01-01	43	32	1025
2009-01-02	57	36	1022
2009-01-03	54	35	1018
2009-01-04	42	29	1020
2009-01-05	48	33	1016

Date	Sea Level Press.ă(hPa) avg	Sea Level Press.ă(hPa) low
2009-01-01	1023	1015
2009-01-02	1012	1007

```
2009-01-03                    1015                    1008
2009-01-04                    1017                    1015
2009-01-05                    1014                    1013

            Visibilityă(km) high Visibilityă(km) avg Visibilityă(km) low  \
Date
2009-01-01                  16                  16                  16
2009-01-02                  16                  13                   2
2009-01-03                  16                  16                  16
2009-01-04                  16                  16                  16
2009-01-05                  16                  16                  16

            Windă(km/h) low Windă(km/h) avg Windă(km/h) high Precip.ă(mm) sum  \
Date
2009-01-01              37              18              60                0
2009-01-02              27              10              48                T
2009-01-03              27              16              42                T
2009-01-04              32              12              40                0
2009-01-05              23              11              34                T

            WeatherEvent
Date
2009-01-01          NaN
2009-01-02         Snow
2009-01-03          NaN
2009-01-04          NaN
2009-01-05          NaN
```

4 3. Exploratory Data Analysis (EDA)

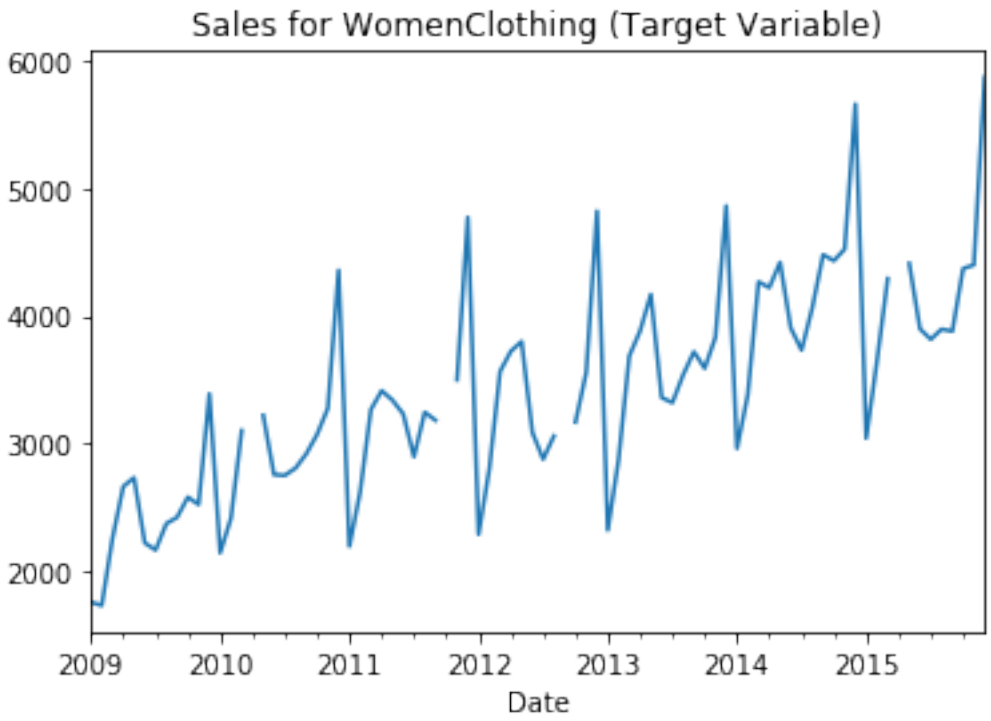
4.1 3.1 Sales Data

4.1.1 3.1.1 Checking for missing values in the data

```
In [29]: sales_data.isnull().sum()
```

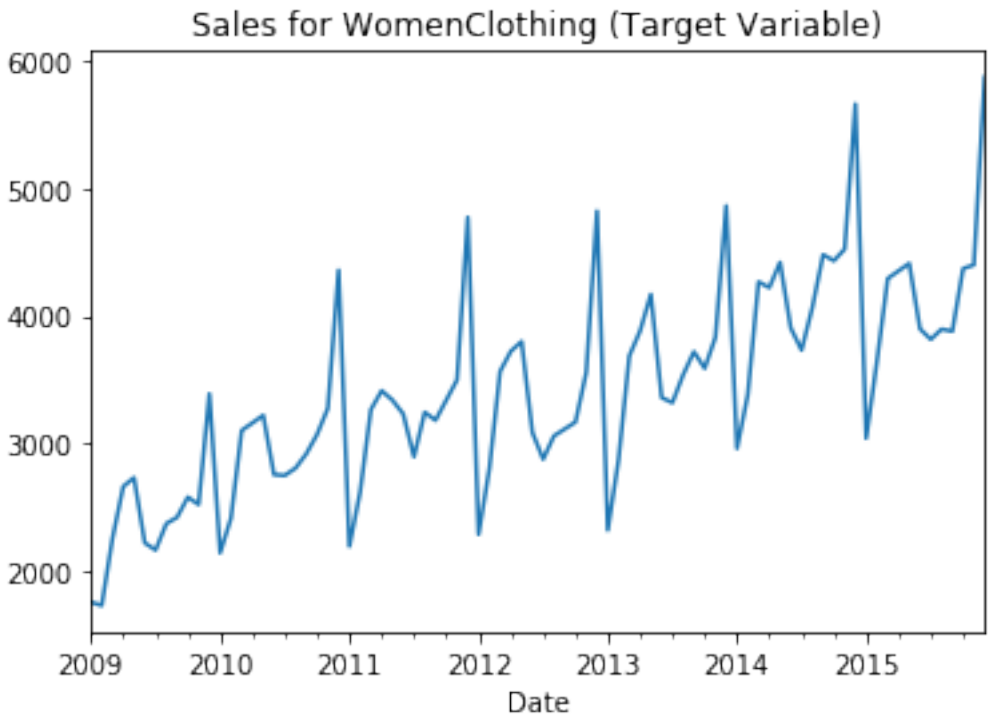
```
Out[29]: ProductCategory             0
         Sales(In ThousandDollars)    4
         dtype: int64
```

```
In [30]: sales_data.plot()
         plt.title("Sales for WomenClothing (Target Variable)")
         plt.legend().remove()
         plt.show()
```




```
In [31]: # Interpolation on the missing values
         sales_data.interpolate(method = 'linear', inplace=True)

In [32]: sales_data.plot()
         plt.title("Sales for WomenClothing (Target Variable)")
         plt.legend().remove()
         plt.show()
```



4.2 3.2 Holidays Data

4.2.1 3.2.1 Checking for missing values in the data

```
In [33]: holidays_data.isnull().sum()
```

```
Out[33]: Event          0
         DayCategory     0
         dtype: int64
```

```
In [34]: # Printing the statistics of the holidays data
         holidays_data.describe(include='all')
```

```
Out[34]:                                Event      DayCategory
         count                            150              150
         unique                            23                2
         top     Martin Luther King Jr. Day  Federal Holiday
         freq                               8               88
```

```
In [35]: holidays_data.tail()
```

```
Out[35]:                                 Event      DayCategory
         Date
         2016-11-24          Thanksgiving Day  Federal Holiday
         2016-12-24             Christmas Eve            Event
```

```
2016-12-25             Christmas Day  Federal Holiday
2016-12-26  'Christmas Day' observed  Federal Holiday
2016-12-31            New Year's Eve            Event
```

4.2.2 3.2.2. Resampling the data

```
In [36]: # Converting 'Events' as 1 and 'Federal Holiday' as 4
         holidays_data['DayCategory'] = holidays_data['DayCategory'].map({'Event':1, 'Federal
         holidays_data = holidays_data.resample('M').sum()
         holidays_data.DayCategory.fillna(value= 0, inplace = True)
```

4.3 3.3 Macroeconomic Data

```
In [37]: print("The first 5 instances from the table:")
         economic_data.head()
```

```
The first 5 instances from the table:
```

```
Out[37]:             Monthly Nominal GDP Index (inMillion$)  \
         Date
         2009-01-01                          14421.752895
         2009-02-01                          14389.200466
         2009-03-01                          14340.701639
         2009-04-01                          14326.815525
         2009-05-01                          14345.904809

                     Monthly Real GDP Index (inMillion$)      CPI  unemployment rate  \
         Date
         2009-01-01                        14407.053343  233.402                7.8
         2009-02-01                        14366.176571  234.663                8.3
         2009-03-01                        14351.786822  235.067                8.7
         2009-04-01                        14351.601731  235.582                9.0
         2009-05-01                        14368.123959  235.975                9.4

                     CommercialBankInterestRateonCreditCardPlans  \
         Date
         2009-01-01                                        12.03
         2009-02-01                                        12.97
         2009-03-01                                        12.97
         2009-04-01                                        12.97
         2009-05-01                                        13.32

                     Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan  \
         Date
         2009-01-01                                              11.44
         2009-02-01                                              11.05
         2009-03-01                                              11.05
         2009-04-01                                              11.05
```

2009-05-01	11.25

Date	Earnings or wages in dollars per hour
2009-01-01	22.05
2009-02-01	22.22
2009-03-01	22.22
2009-04-01	22.13
2009-05-01	22.04

Date	Cotton Monthly Price - US cents per Pound(lbs)	Change(in%)
2009-01-01	57.70	4.02
2009-02-01	55.21	-4.32
2009-03-01	51.50	-6.72
2009-04-01	56.78	10.25
2009-05-01	61.95	9.11

Date	Average upland planted(million acres)
2009-01-01	9.296
2009-02-01	9.296
2009-03-01	9.296
2009-04-01	9.296
2009-05-01	9.297

Date	Average upland harvested(million acres)	yieldperharvested acre
2009-01-01	7.559	799
2009-02-01	7.559	799
2009-03-01	7.559	799
2009-04-01	7.559	787
2009-05-01	7.400	803

Date	Production (in 480-lb netweright in million bales)
2009-01-01	12.589
2009-02-01	12.589
2009-03-01	12.589
2009-04-01	12.400
2009-05-01	12.384

Date	Mill use (in 480-lb netweright in million bales)	Exports
2009-01-01	4.17	11.550
2009-02-01	3.87	11.100
2009-03-01	3.72	11.650
2009-04-01	3.62	12.225

```
2009-05-01                                                    3.52    12.300
```

4.3.1 3.3.1 Checking for missing values in the data

```
In [38]: economic_data.isnull().sum()
```

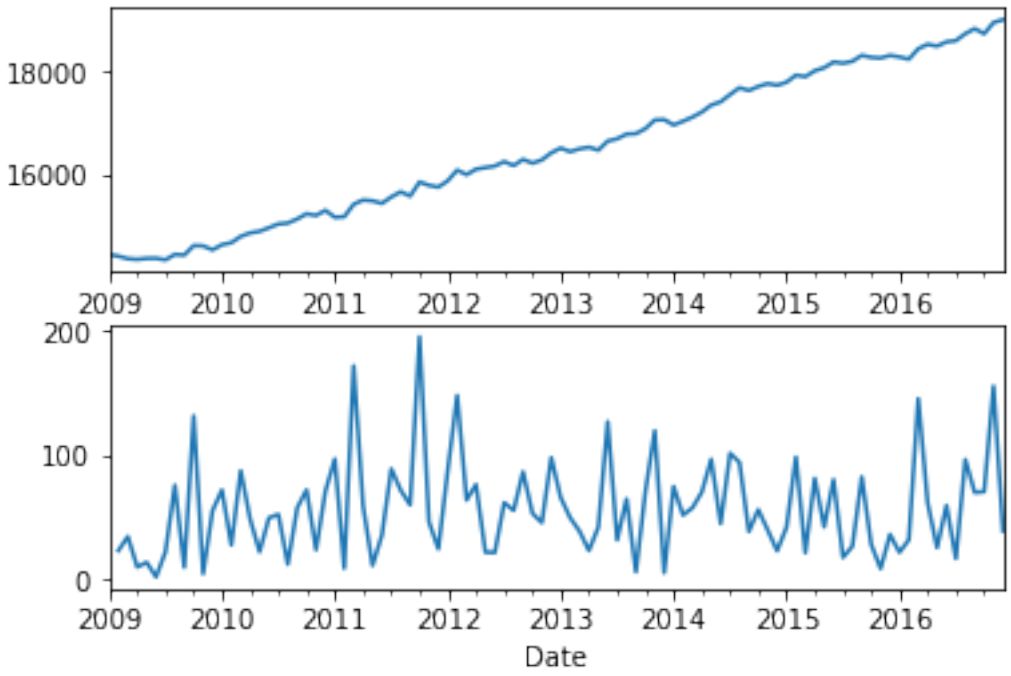
```
Out[38]: Monthly Nominal GDP Index (inMillion$)                                  0
         Monthly Real GDP Index (inMillion$)                                     0
         CPI                                                                     0
         unemployment rate                                                       0
         CommercialBankInterestRateonCreditCardPlans                             0
         Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan       0
         Earnings or wages  in dollars per hour                                  0
         Cotton Monthly Price - US cents per Pound(lbs)                          0
         Change(in%)                                                             0
         Average upland planted(million acres)                                   0
         Average upland harvested(million acres)                                 0
         yieldperharvested acre                                                  0
         Production (in  480-lb netweright in million bales)                     0
         Mill use  (in  480-lb netweright in million bales)                      0
         Exports                                                                 0
         dtype: int64
```

4.3.2 3.3.2 Checking for Outliers in the data

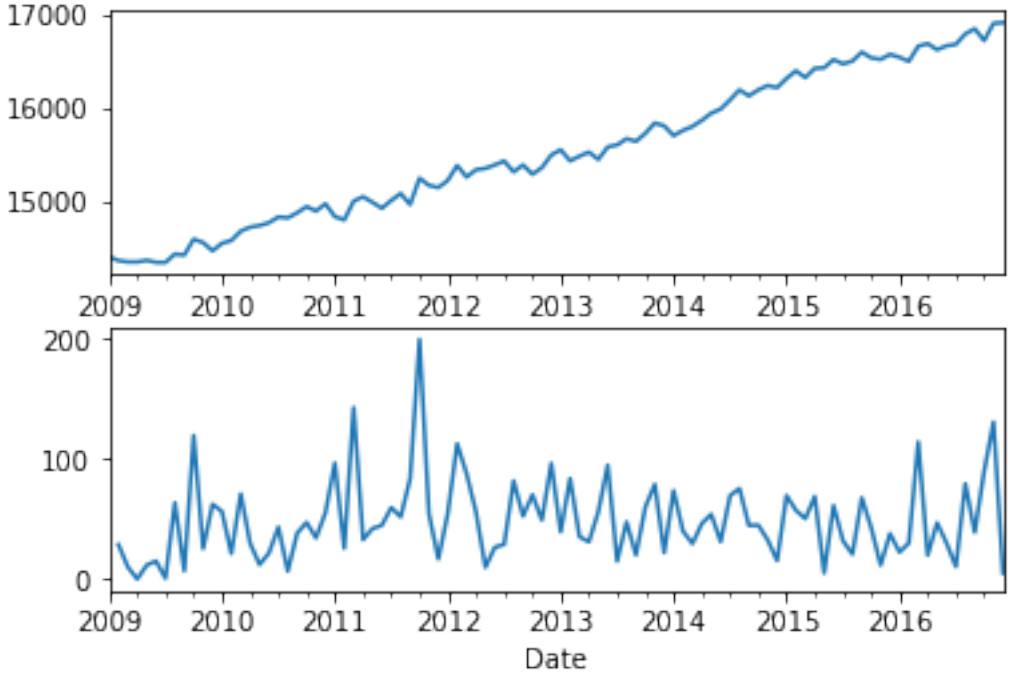
```
In [39]: std_economic_data = pd.DataFrame()
         for columns in economic_data.columns:
             std_economic_data[columns] = economic_data[columns].rolling(window=2,center=False)
             # economic_data[columns] = economic_data[columns].pct_change()

         for columns in economic_data.columns:
             print("\n","Standard Deviation of", columns)
             fig = plt.figure()
             ax1 = plt.subplot2grid((2,1),(0,0))
             ax2 = plt.subplot2grid((2,1),(1,0))
             economic_data[columns].plot(ax=ax1)
             std_economic_data[columns].plot(ax=ax2)
             plt.legend().remove()
             plt.legend().remove()
             plt.show()
```

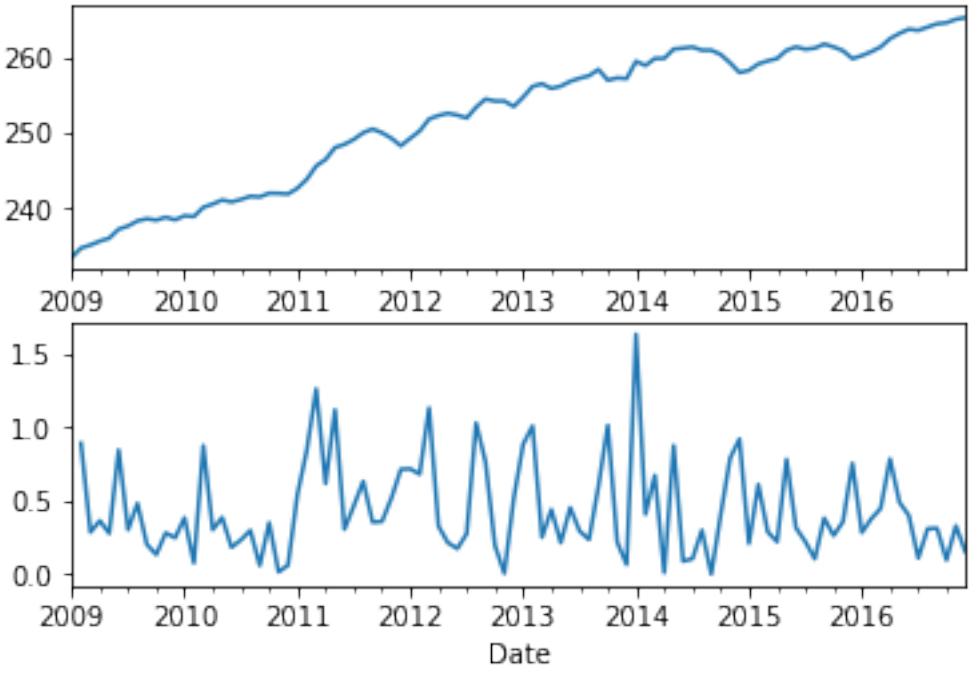
```
 Standard Deviation of Monthly Nominal GDP Index (inMillion$)
```



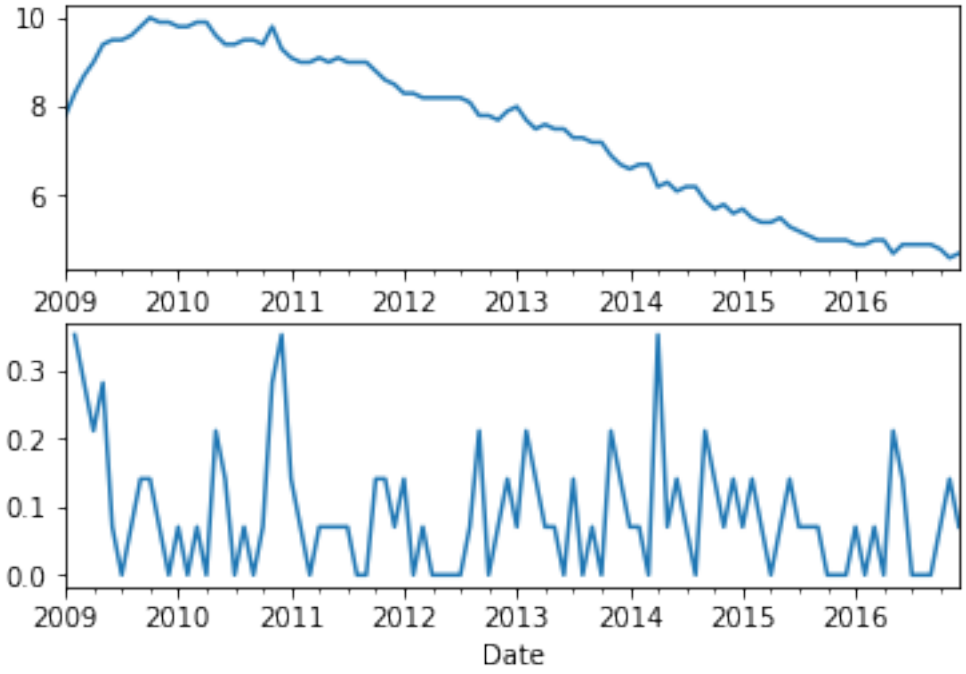
Standard Deviation of Monthly Real GDP Index (inMillion$)



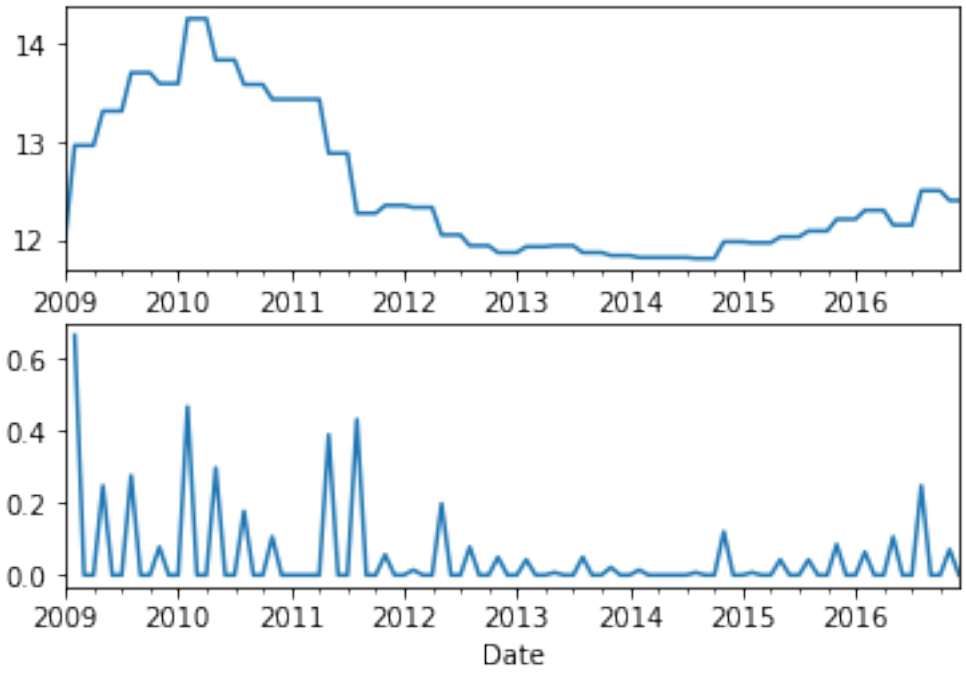
Standard Deviation of CPI



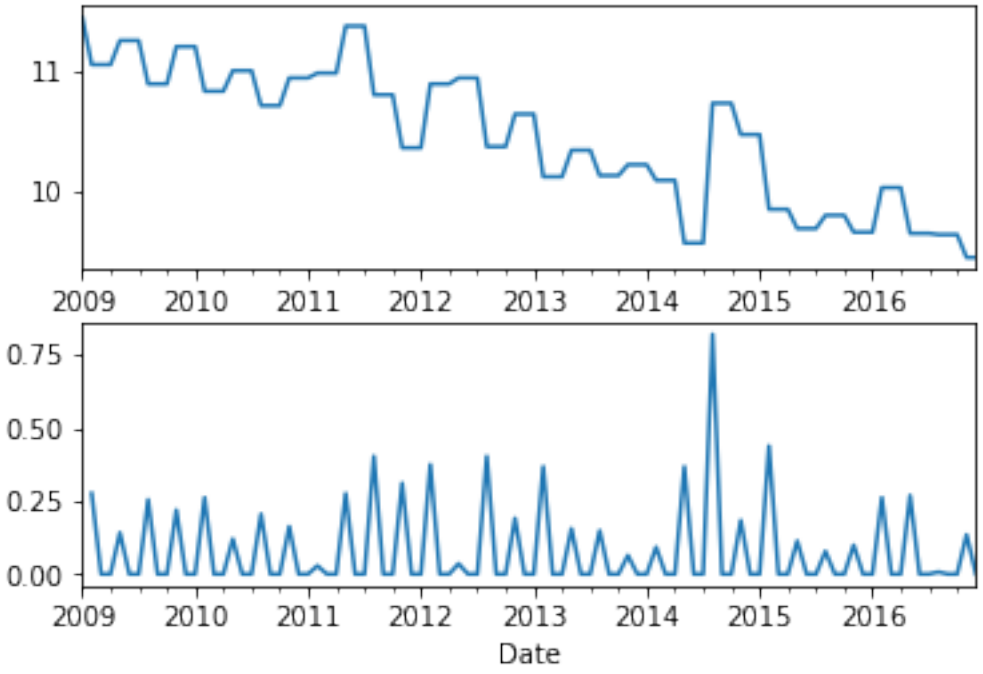
Standard Deviation of unemployment rate



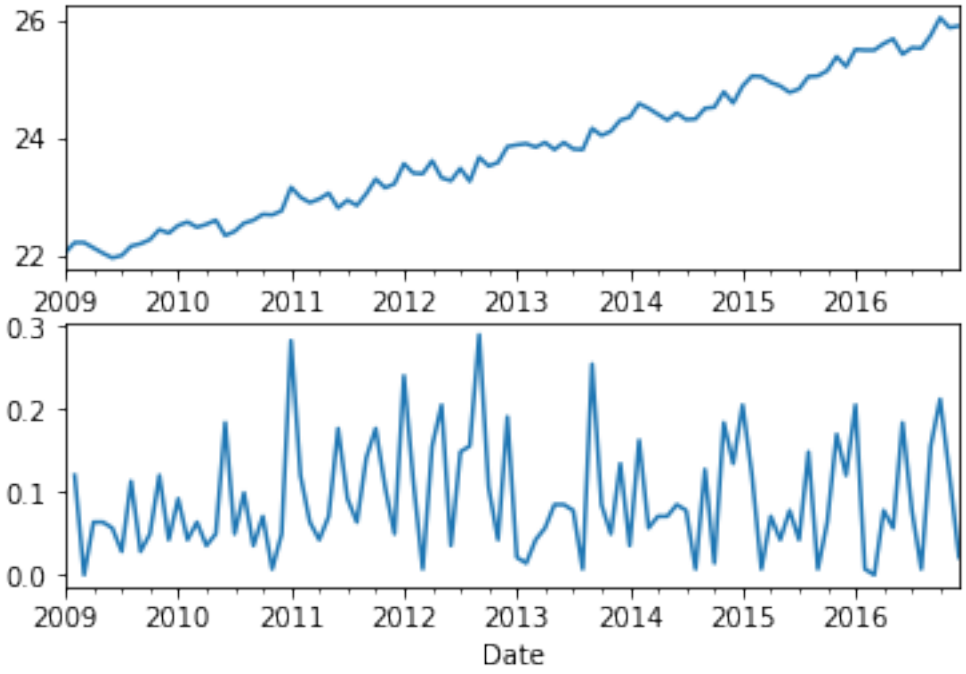
Standard Deviation of CommercialBankInterestRateonCreditCardPlans



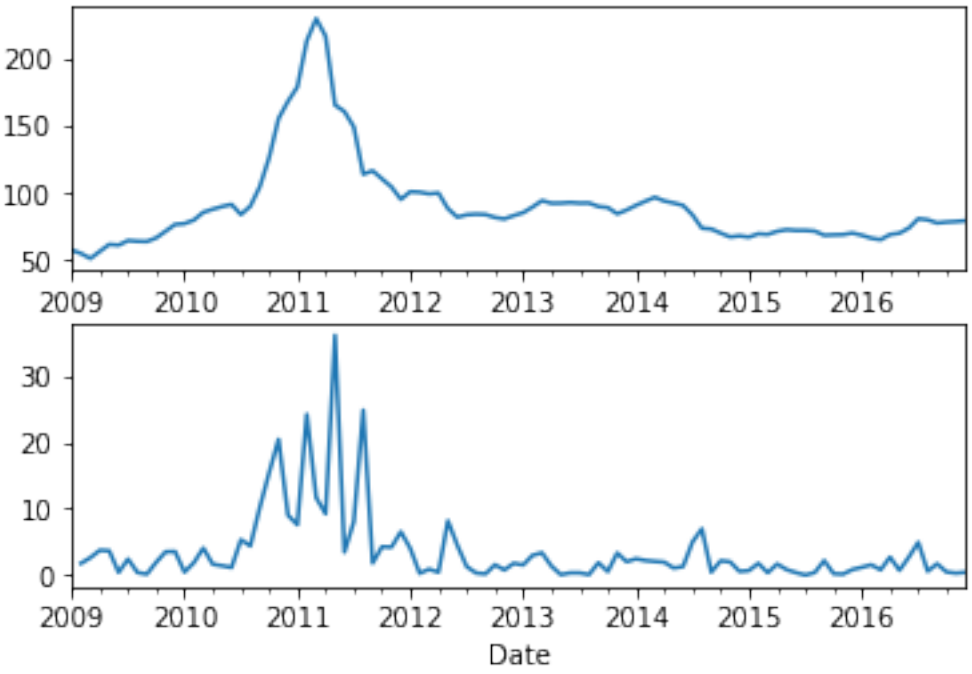
Standard Deviation of Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan



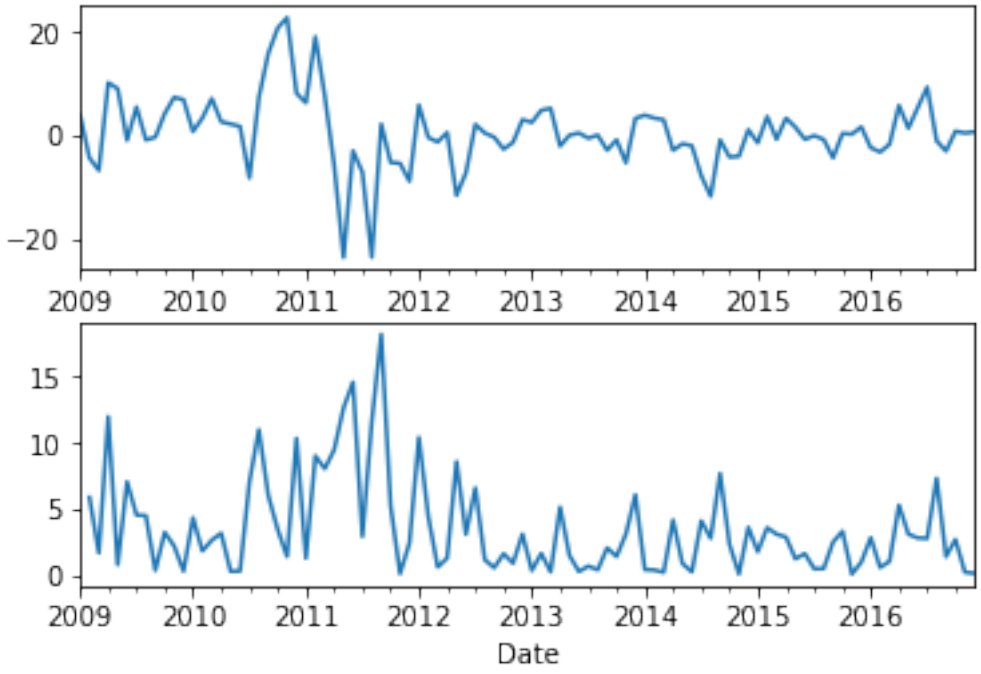
Standard Deviation of Earnings or wages in dollars per hour



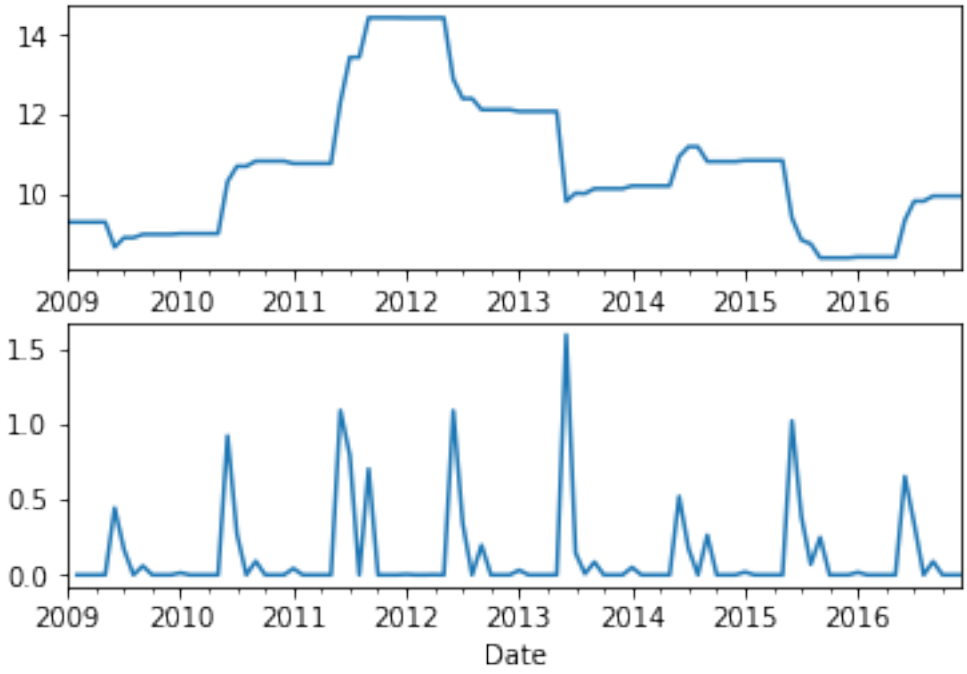
Standard Deviation of Cotton Monthly Price - US cents per Pound(lbs)



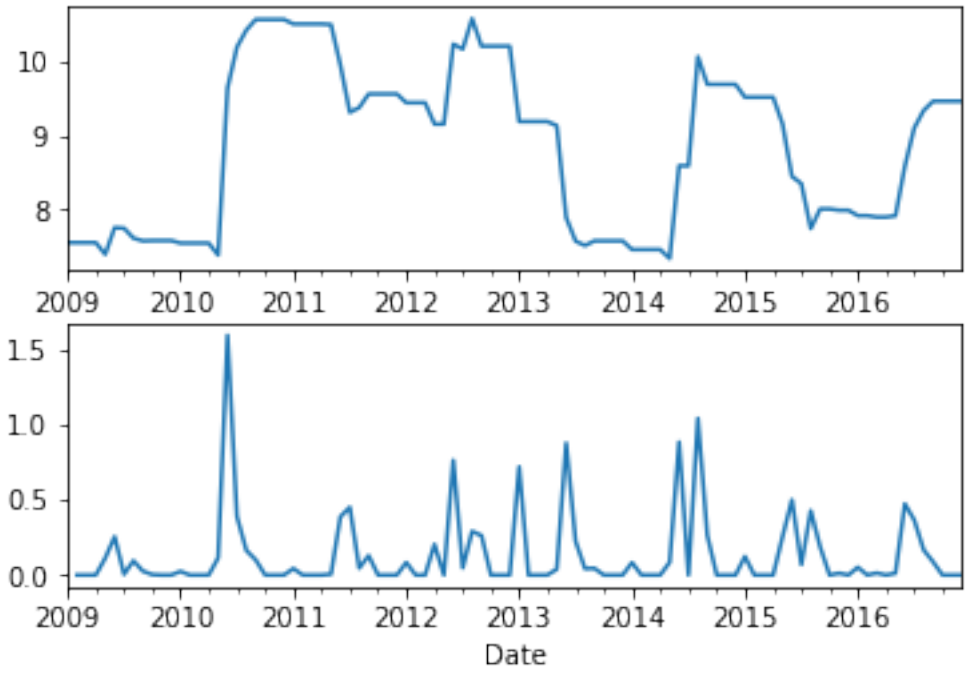
Standard Deviation of Change(in%)



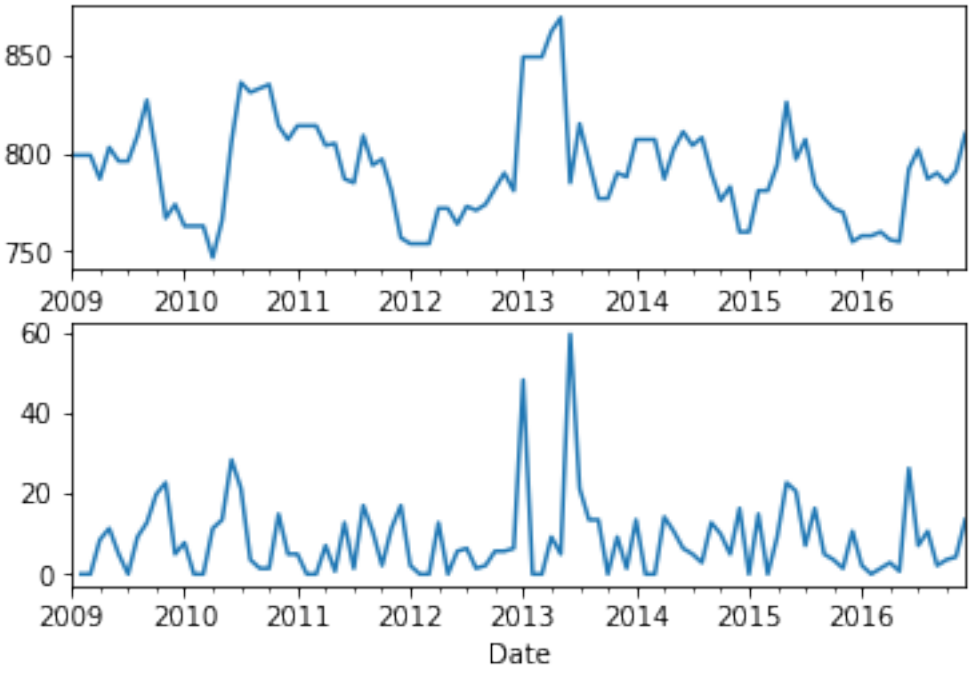
Standard Deviation of Average upland planted(million acres)



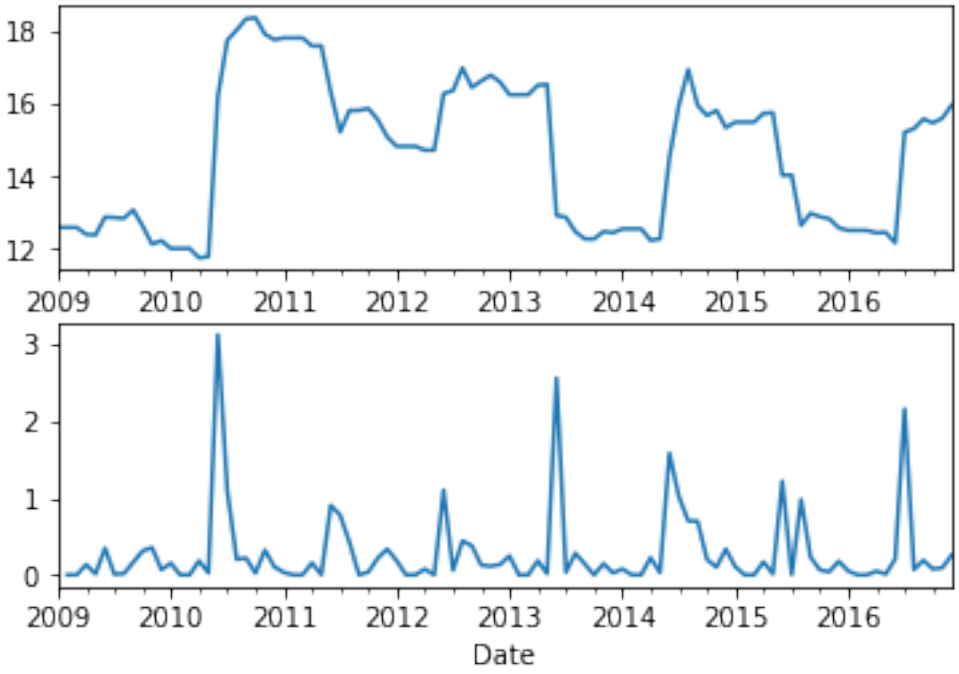
Standard Deviation of Average upland harvested(million acres)



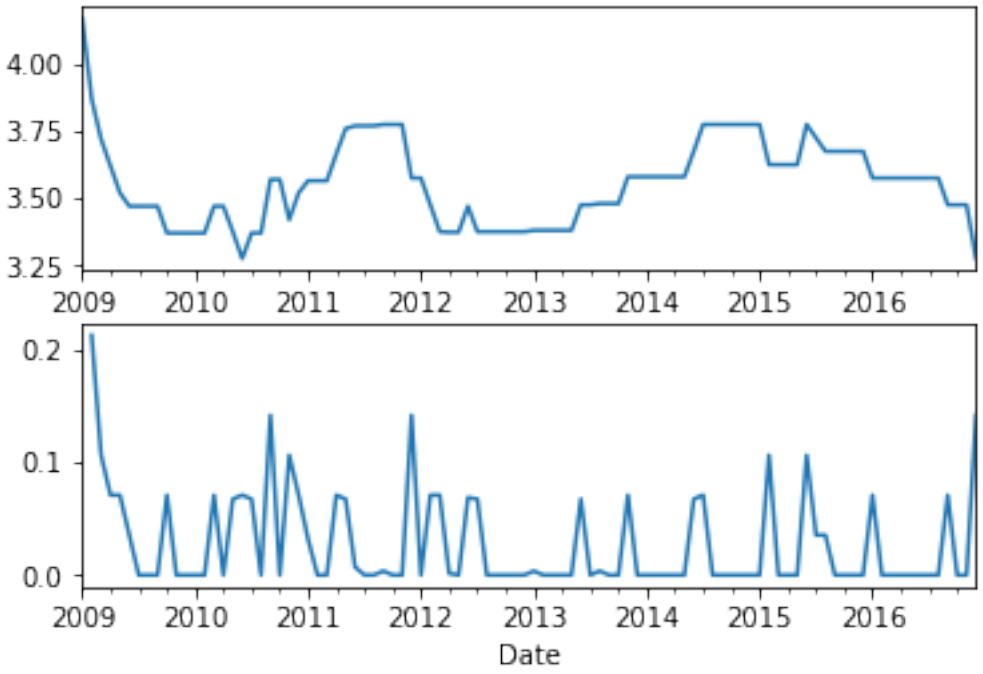
Standard Deviation of yieldperharvested acre



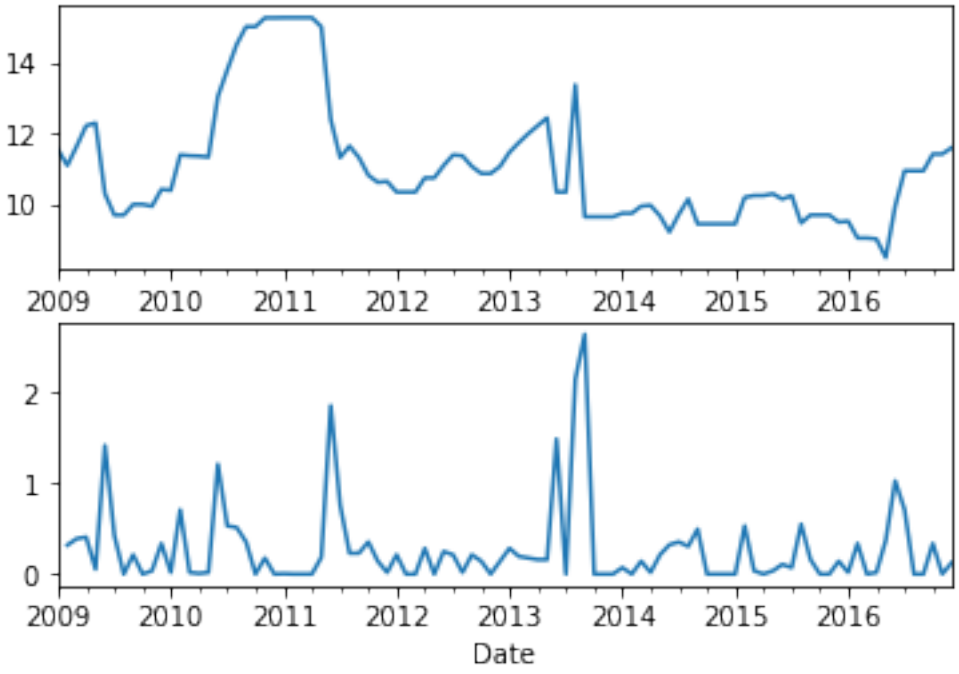
Standard Deviation of Production (in 480-lb netweright in million bales)



Standard Deviation of Mill use (in 480-lb netweright in million bales)



```
Standard Deviation of Exports
```



4.4 3.4 Weather Data

4.4.1 3.4.1 Checking for non-numeric characters in numeric variables

NOTE:- "T" stands for Trace. This is a small amount of precipitation that will wet a raingage but is less than the 0.01 inch measuring limit. Hence converting all the "T" values to 0.01

```
In [40]: # Converting the cells that have the value "T" to 0.01
         weather_data[weather_data.columns[18]] = weather_data[weather_data.columns[18]].apply

In [41]: # Converting all cells that have the value "-" to NaN
         weather_data = weather_data.applymap(lambda x: np.nan if x == '-' else x)

In [42]: # Replacing the missing values in 'WeatherEvent' column to "NA"
         weather_data.WeatherEvent.fillna(value="NotApplicable", inplace=True)

         # Droping the two rows having all columns as NaN
         weather_data = weather_data[~weather_data["Temp high (řC)"].isnull()]
```

In [43]:
```
weather_data.describe(include='all')
```

Out[43]:

	Temp high (řC)	Temp avg (řC)	Temp low (řC)	Dew Point high (řC)
count	2920.000000	2920.000000	2920.000000	2920.000000
unique	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN
mean	17.371918	13.530137	9.429452	8.503767
std	10.230641	9.662644	9.322538	10.045084
min	-9.000000	-14.000000	-18.000000	-24.000000
25%	9.000000	6.000000	2.000000	1.000000
50%	18.000000	14.000000	10.000000	10.000000
75%	26.000000	22.000000	18.000000	17.000000
max	40.000000	34.000000	29.000000	26.000000

	Dew Point avg (řC)	Dew Point low (řC)	Humidityă(%) high
count	2920.000000	2920.000000	2920.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	5.366096	1.854110	77.907192
std	10.509162	10.993565	15.637223
min	-27.000000	-28.000000	28.000000
25%	-3.000000	-7.000000	65.000000
50%	6.000000	2.000000	79.000000
75%	14.000000	11.000000	93.000000
max	24.000000	22.000000	100.000000

	Humidityă(%) avg	Humidityă(%) low	Sea Level Press.ă(hPa) high
count	2920.000000	2920.000000	2903.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	61.460616	44.522945	1019.838099
std	14.433762	15.827029	7.054331
min	20.000000	9.000000	994.000000
25%	51.000000	33.000000	1015.000000
50%	61.000000	42.000000	1020.000000
75%	72.000000	54.000000	1024.000000
max	97.000000	93.000000	1044.000000

	Sea Level Press.ă(hPa) avg	Sea Level Press.ă(hPa) low
count	2903.000000	2903.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	1016.218739	1012.660007
std	7.429849	7.980106

```
min                    980.000000                966.000000
25%                   1012.000000               1008.000000
50%                   1016.000000               1013.000000
75%                   1021.000000               1018.000000
max                   1041.000000               1037.000000

        Visibilityă(km) high  Visibilityă(km) avg  Visibilityă(km) low  \
count            2894.000000          2894.000000          2894.000000
unique                   NaN                  NaN                  NaN
top                      NaN                  NaN                  NaN
freq                     NaN                  NaN                  NaN
mean               15.895646            13.909468            10.449896
std                 0.618441             3.157808             6.142232
min                 8.000000             1.000000             0.000000
25%                16.000000            13.000000             3.000000
50%                16.000000            16.000000            14.000000
75%                16.000000            16.000000            16.000000
max                16.000000            16.000000            16.000000

        Windă(km/h) low  Windă(km/h) avg  Windă(km/h) high  Precip.ă(mm) sum  \
count       2902.000000      2902.000000       2863.000000       2920.000000
unique              NaN              NaN               NaN               NaN
top                 NaN              NaN               NaN               NaN
freq                NaN              NaN               NaN               NaN
mean          23.061682         9.092695         36.004191          3.454425
std            7.262782         4.883618         10.772405          9.851643
min            6.000000         0.000000         11.000000          0.000000
25%           19.000000         6.000000         27.000000          0.000000
50%           23.000000         8.000000         35.000000          0.000000
75%           26.000000        11.000000         42.000000          1.270000
max          159.000000       159.000000        159.000000        147.570000

         WeatherEvent
count            2920
unique              9
top     NotApplicable
freq             1906
mean              NaN
std               NaN
min               NaN
25%               NaN
50%               NaN
75%               NaN
max               NaN
```

4.4.2 3.4.1 Checking for missing vales in the data

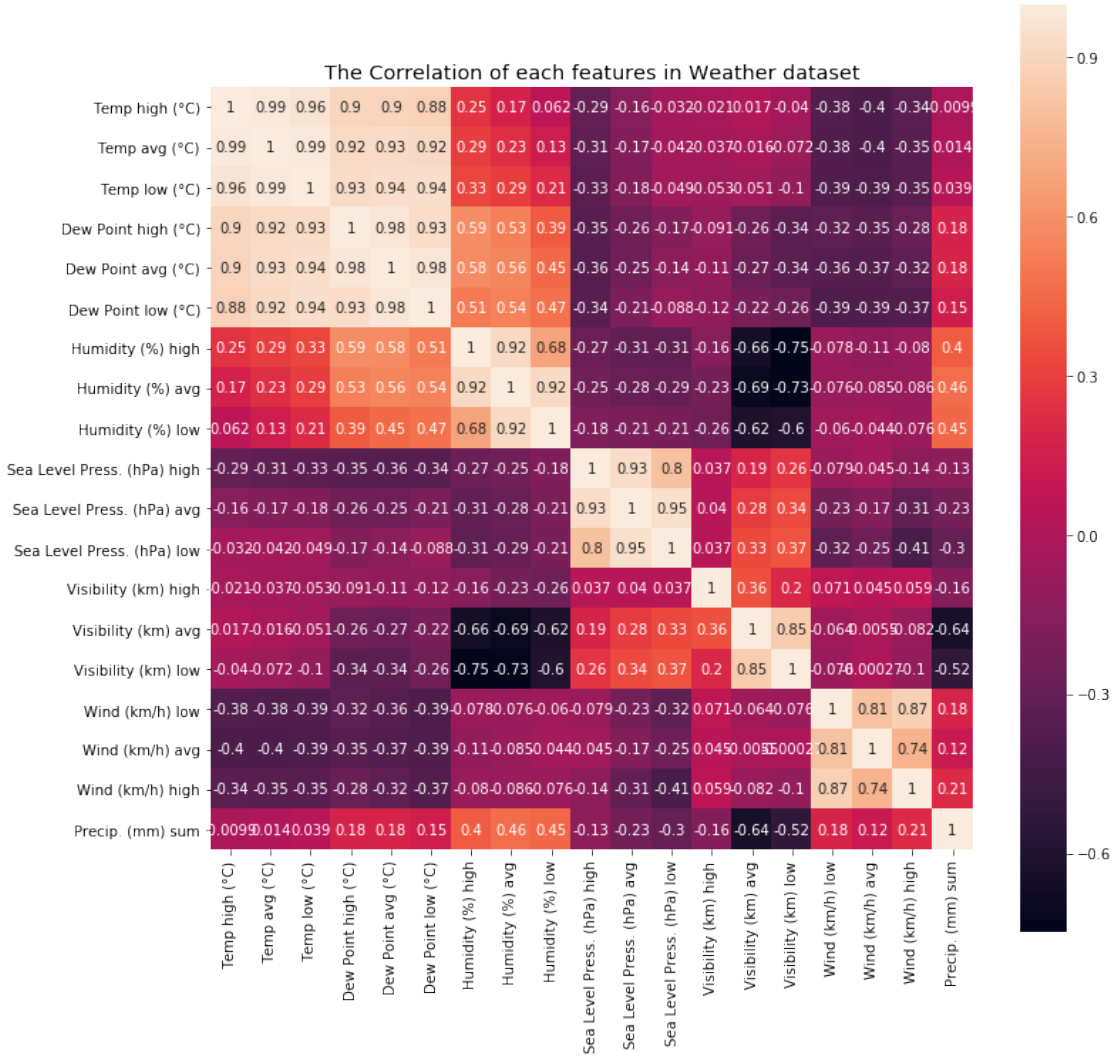
```
In [44]: weather_data.isnull().sum()
```

```
Out[44]: Temp high (řC)                  0
         Temp avg (řC)                   0
         Temp low (řC)                   0
         Dew Point high (řC)             0
         Dew Point avg (řC)              0
         Dew Point low (řC)              0
         Humidityă(%) high               0
         Humidityă(%) avg                0
         Humidityă(%) low                0
         Sea Level Press.ă(hPa) high    17
         Sea Level Press.ă(hPa) avg     17
         Sea Level Press.ă(hPa) low     17
         Visibilityă(km) high           26
         Visibilityă(km) avg            26
         Visibilityă(km) low            26
         Windă(km/h) low                18
         Windă(km/h) avg                18
         Windă(km/h) high               57
         Precip.ă(mm) sum                0
         WeatherEvent                    0
         dtype: int64
```

4.4.3 3.4.3 Calculating the correlation of each variables

```
In [45]: correlation_weather = weather_data.corr()

         ### Ploting the correlation
         %matplotlib inline
         plt.figure(figsize =(12,12))
         sns.heatmap(data= correlation_weather, annot =True, square= True)
         plt.title('The Correlation of each features in Weather dataset', fontsize= 'x-large')
         plt.show()
```



In [46]: `weather_data.dtypes`

Out[46]:

```
Temp high (řC)                 float64
Temp avg (řC)                  float64
Temp low (řC)                  float64
Dew Point high (řC)            float64
Dew Point avg (řC)             float64
Dew Point low (řC)             float64
Humidityă(%) high              float64
Humidityă(%) avg               float64
Humidityă(%) low               float64
Sea Level Press.ă(hPa) high    float64
Sea Level Press.ă(hPa) avg     float64
Sea Level Press.ă(hPa) low     float64
Visibilityă(km) high           float64
```

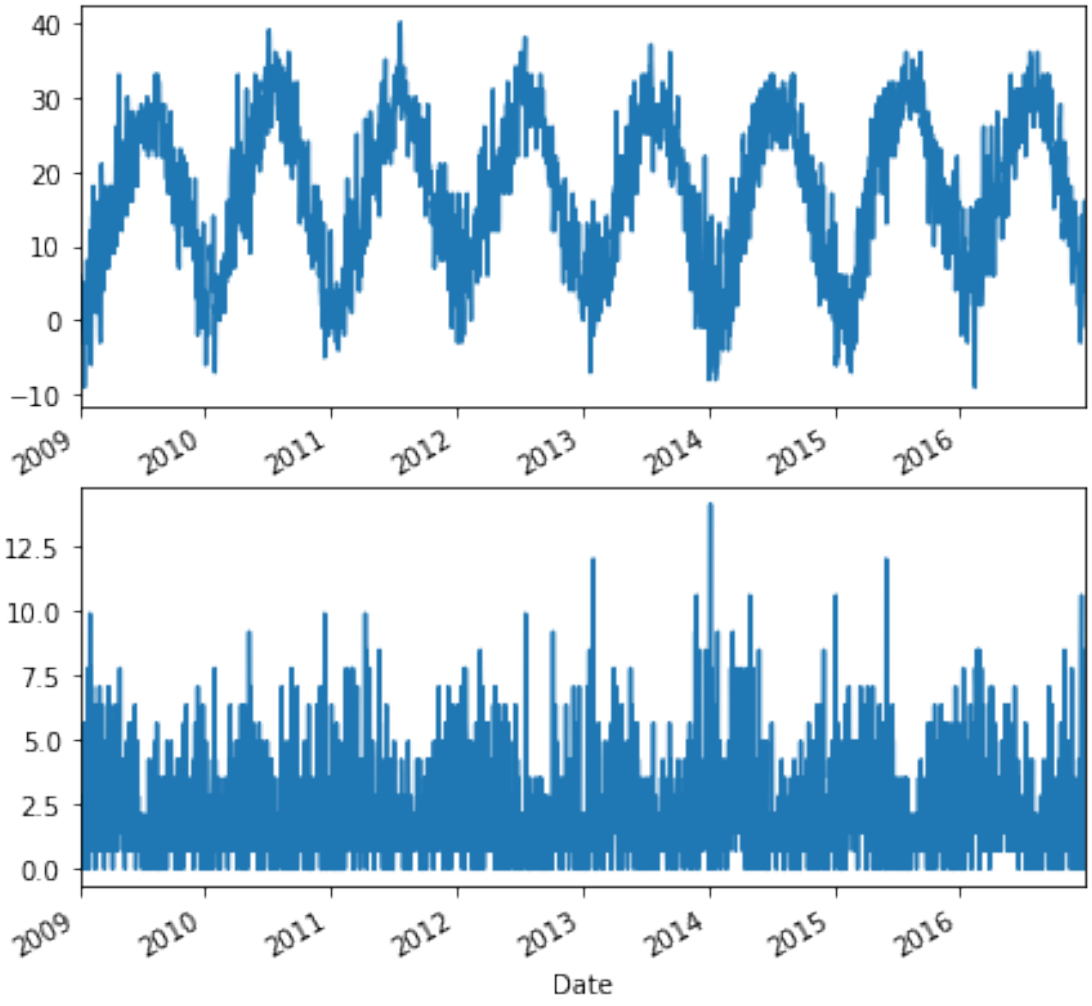
```
Visibilityă(km) avg           float64
Visibilityă(km) low           float64
Windă(km/h) low               float64
Windă(km/h) avg               float64
Windă(km/h) high              float64
Precip.ă(mm) sum              float64
WeatherEvent                   object
dtype: object
```

4.4.4 3.4.3 Checking for Outliers in the data

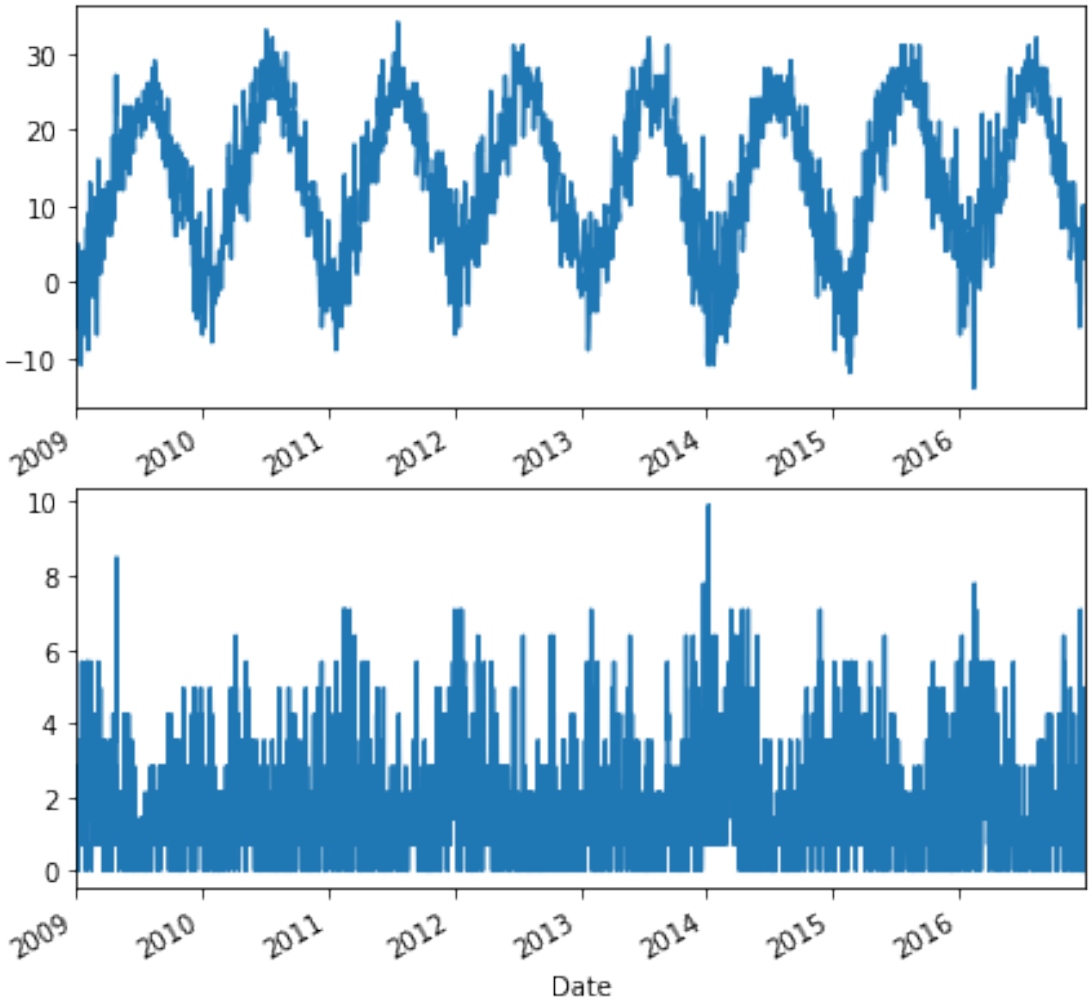
```
In [47]: std_weather_data = pd.DataFrame()
         for columns in weather_data.iloc[:,0:19].columns:
             std_weather_data[columns] = weather_data[columns].rolling(window=2,center=False).s

         for columns in weather_data.iloc[:,0:19].columns:
             print("\n","Standard Deviation of", columns)
             fig = plt.figure(figsize=(7,7))
             ax1 = plt.subplot2grid((2,1),(0,0))
             ax2 = plt.subplot2grid((2,1),(1,0))
             weather_data[columns].plot(ax=ax1)
             std_weather_data[columns].plot(ax=ax2)
             plt.legend().remove()
             plt.legend().remove()
             plt.show()
```

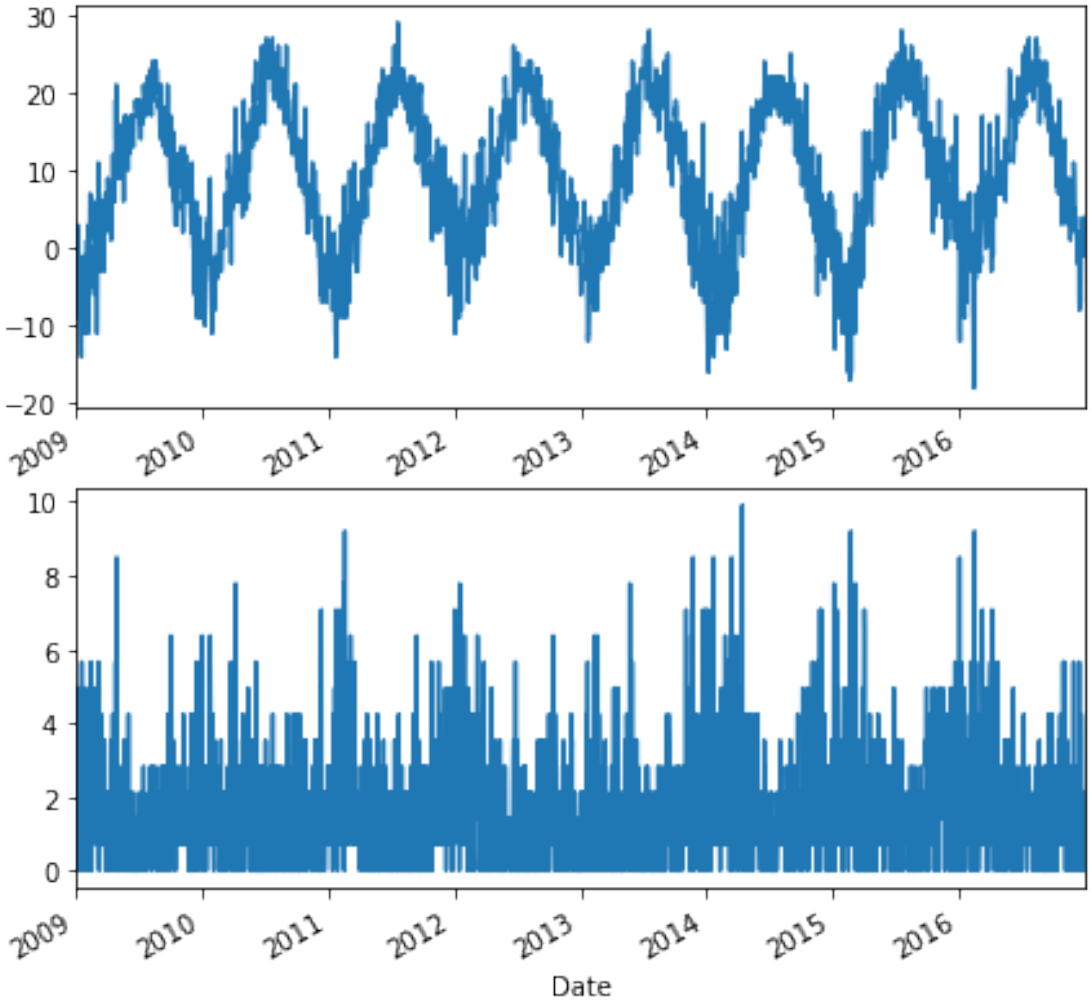
```
 Standard Deviation of Temp high (řC)
```



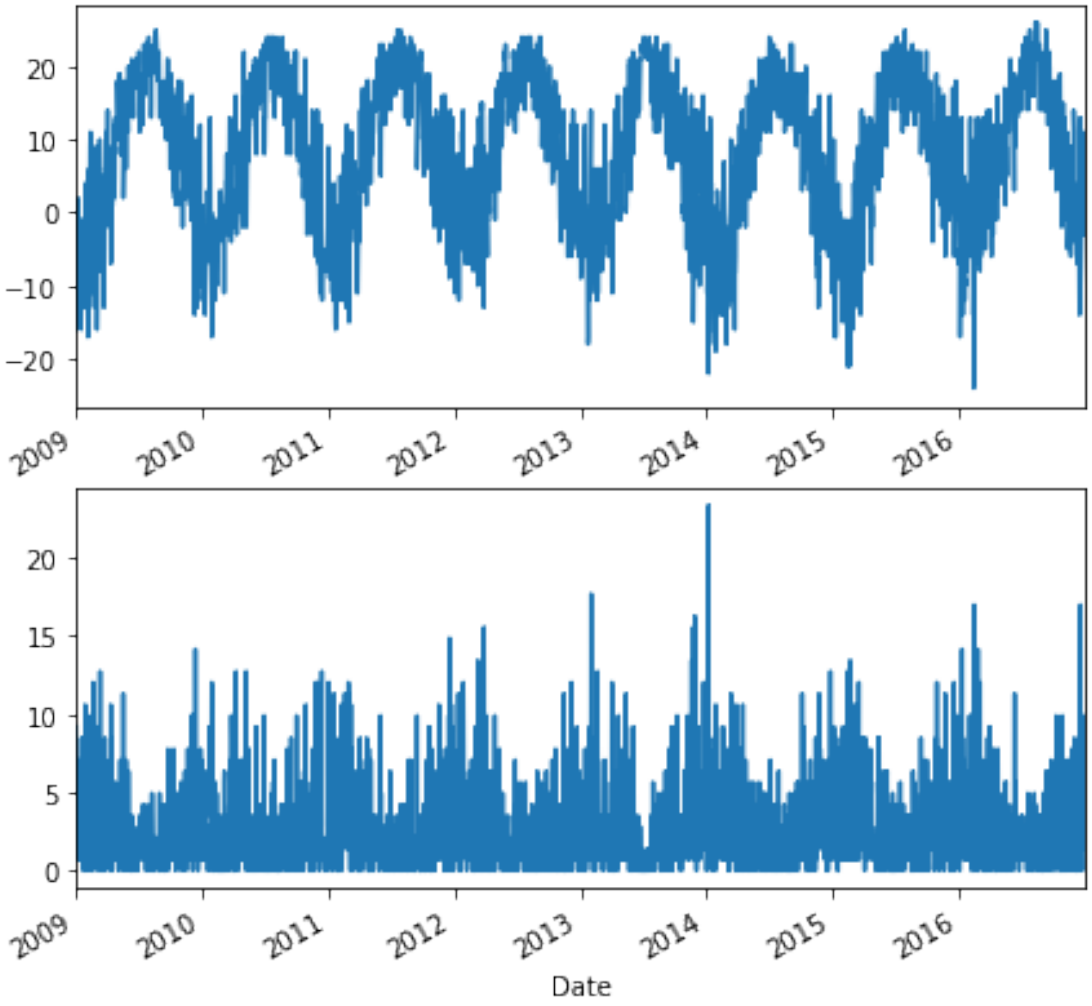
Standard Deviation of Temp avg (řC)



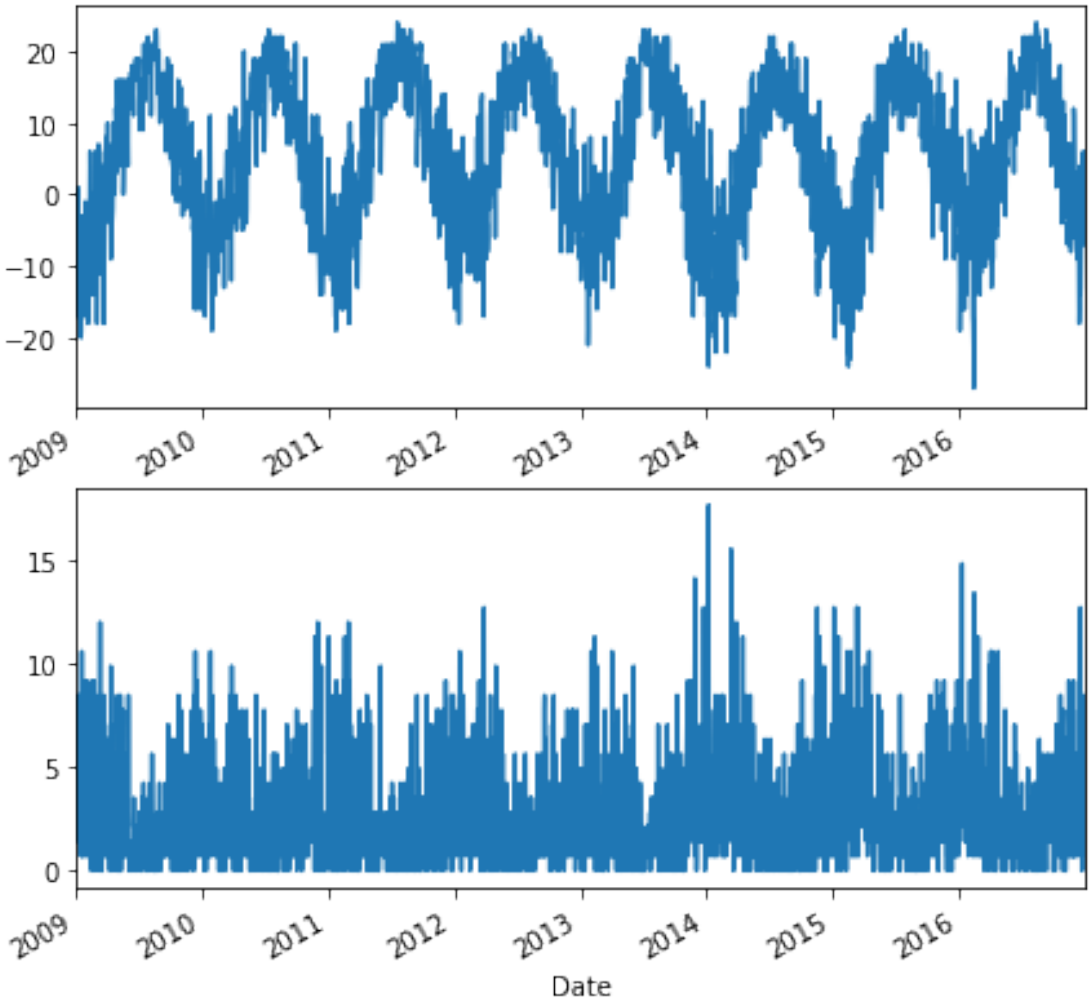
Standard Deviation of Temp low (řC)



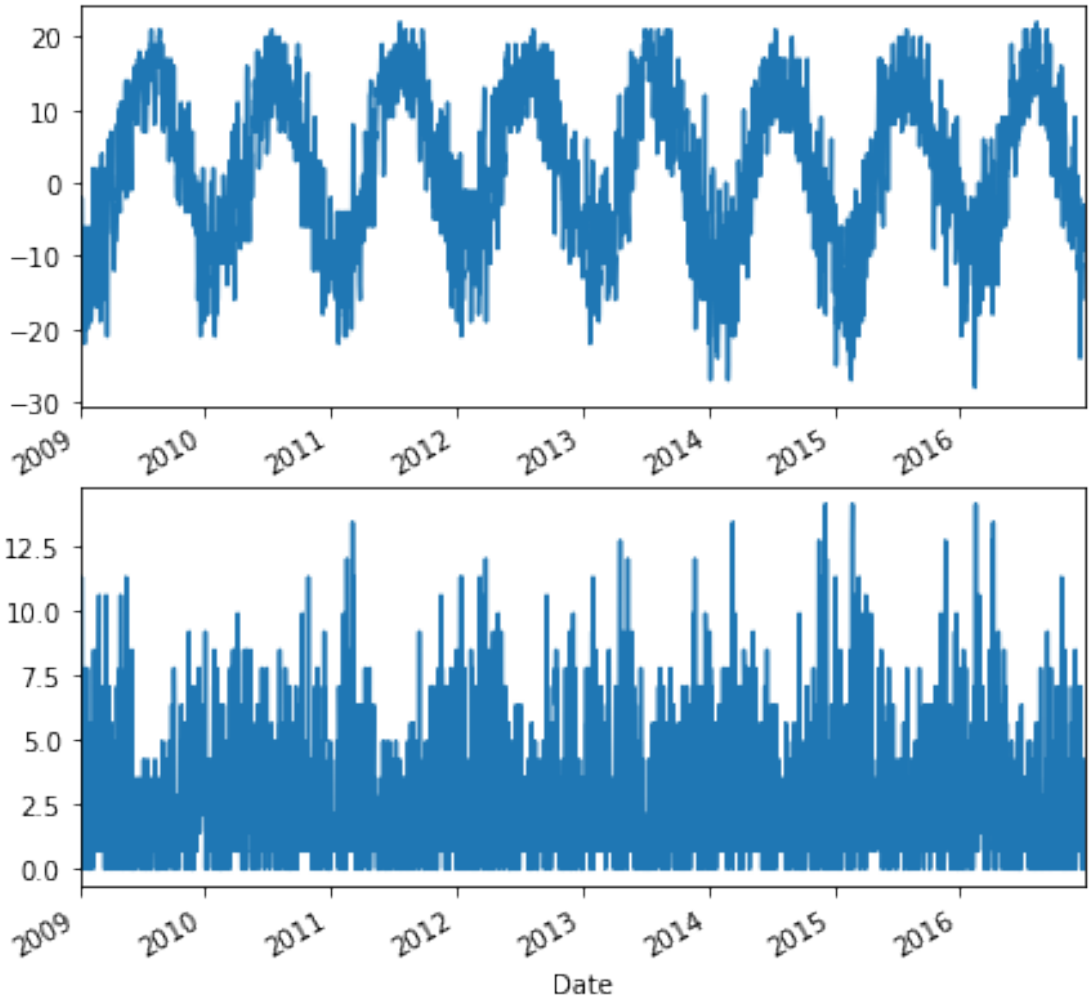
Standard Deviation of Dew Point high (řC)



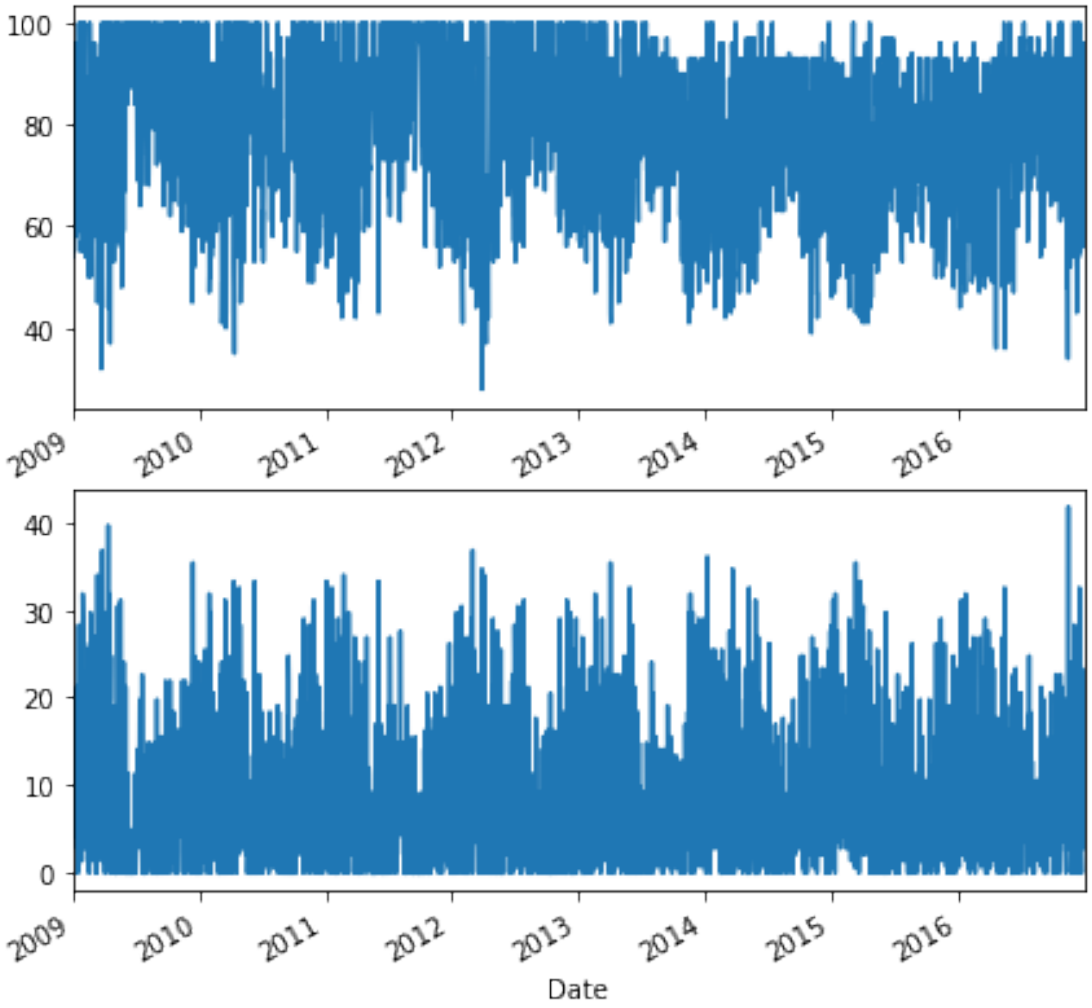
Standard Deviation of Dew Point avg (řC)



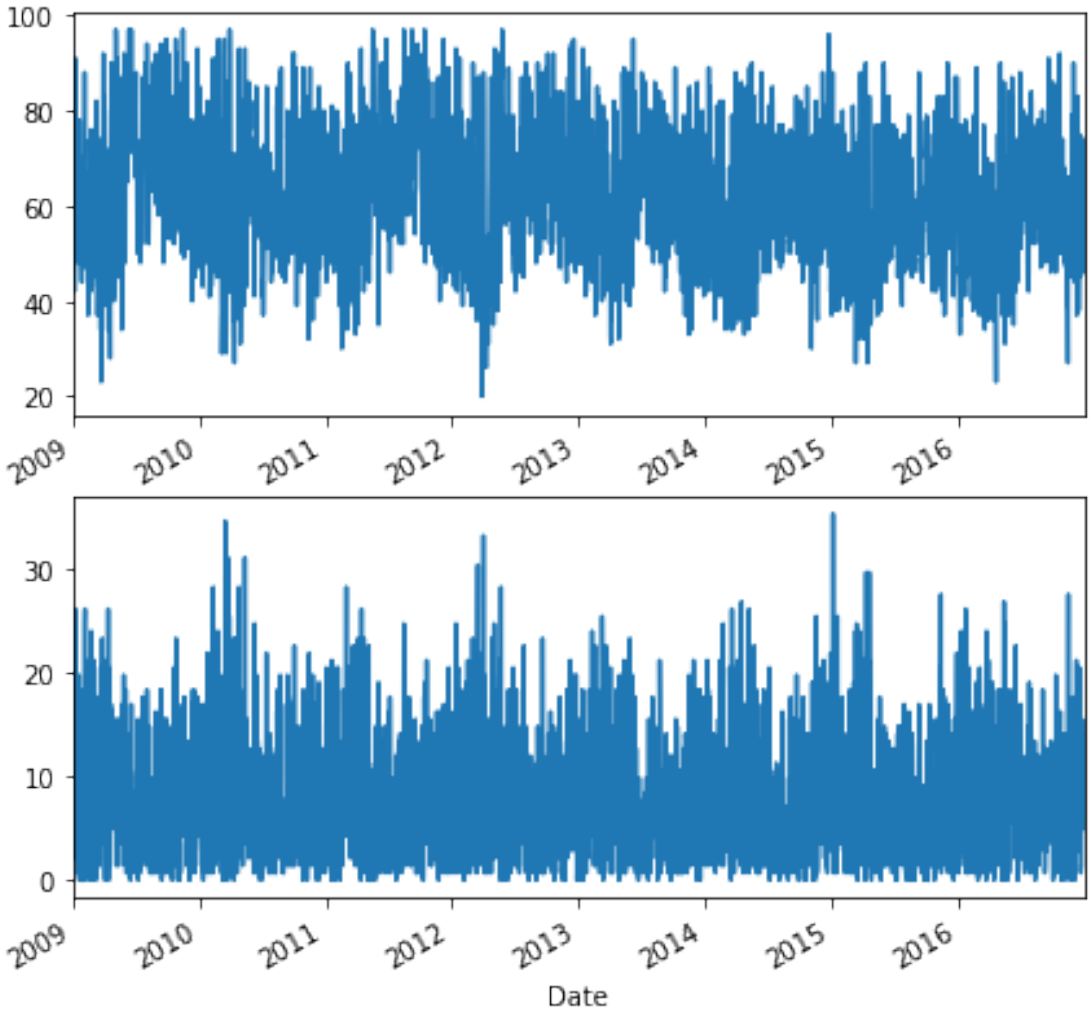
Standard Deviation of Dew Point low (řC)



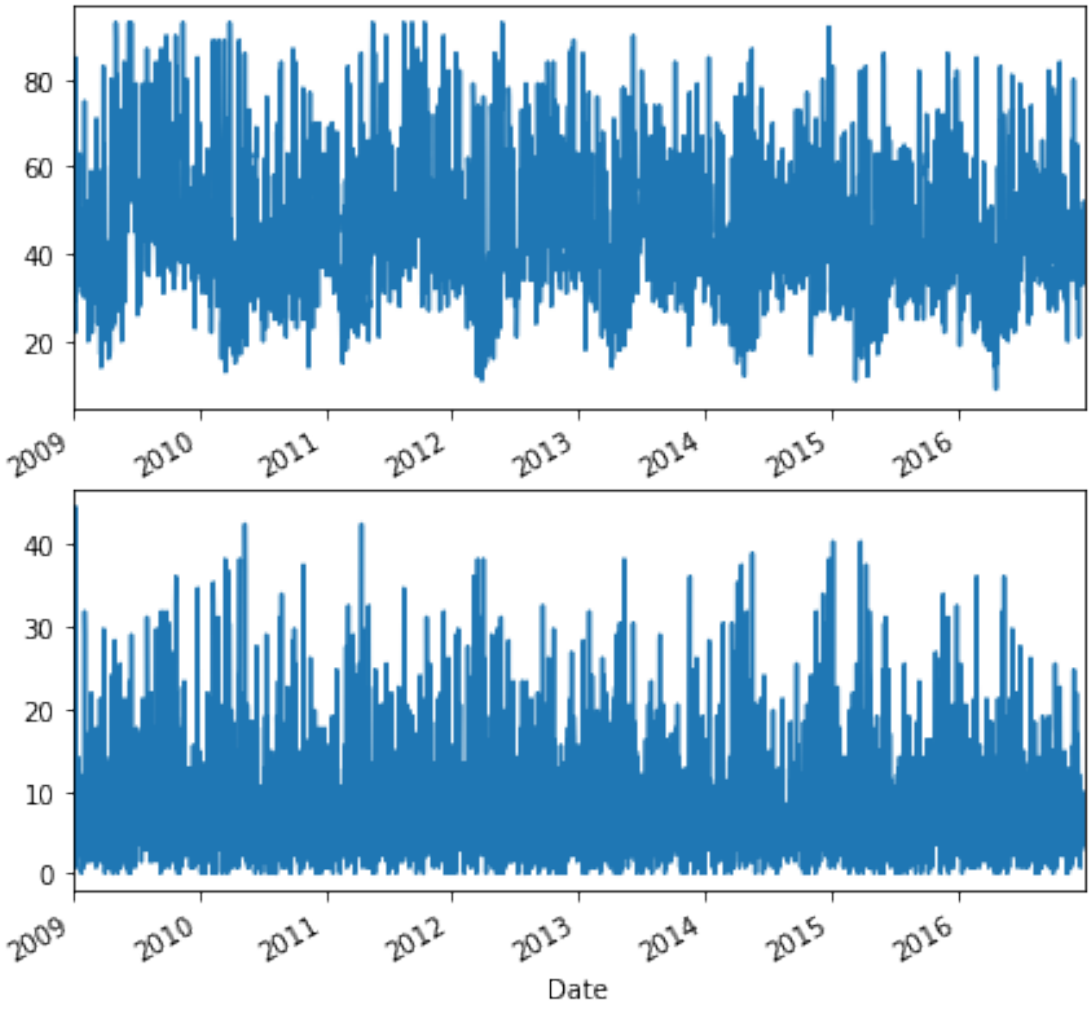
Standard Deviation of Humidityă(%) high



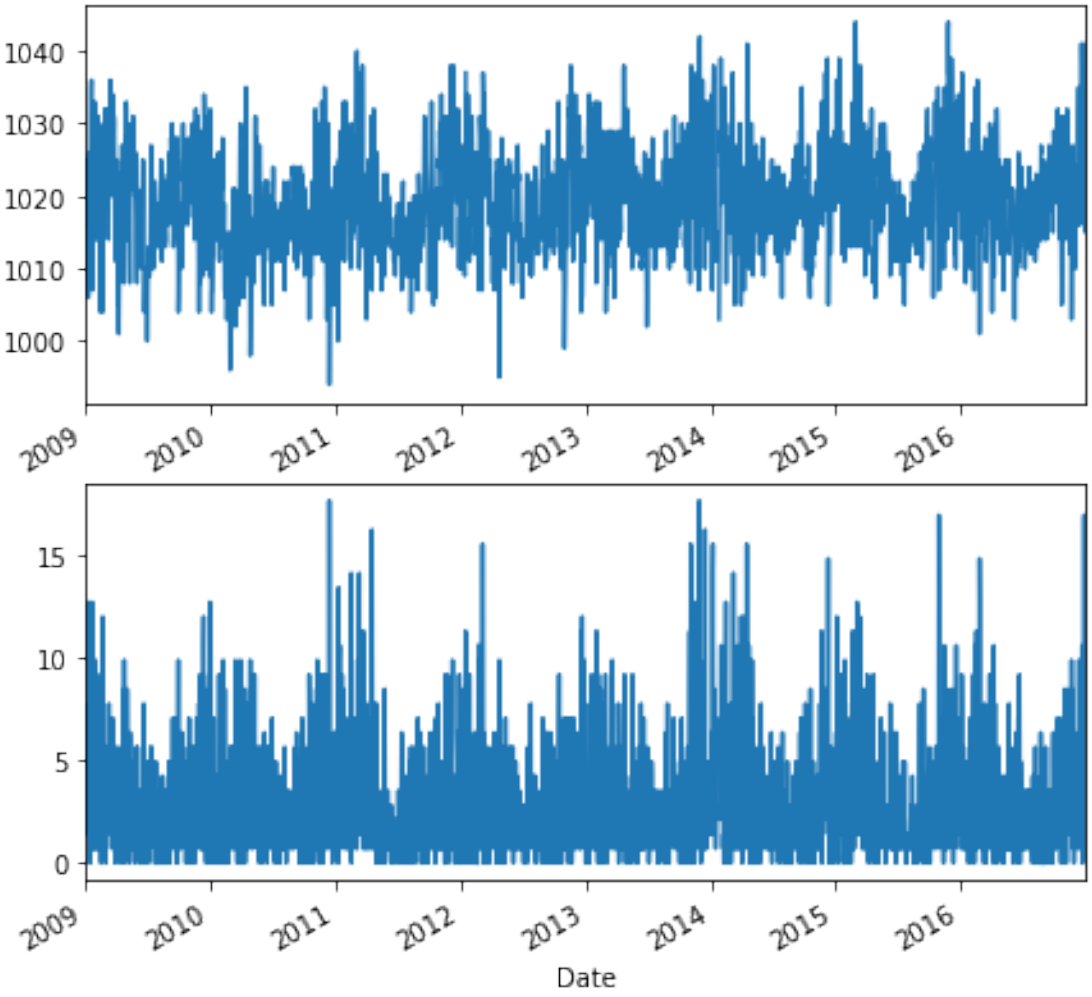
Standard Deviation of Humidityă(%) avg



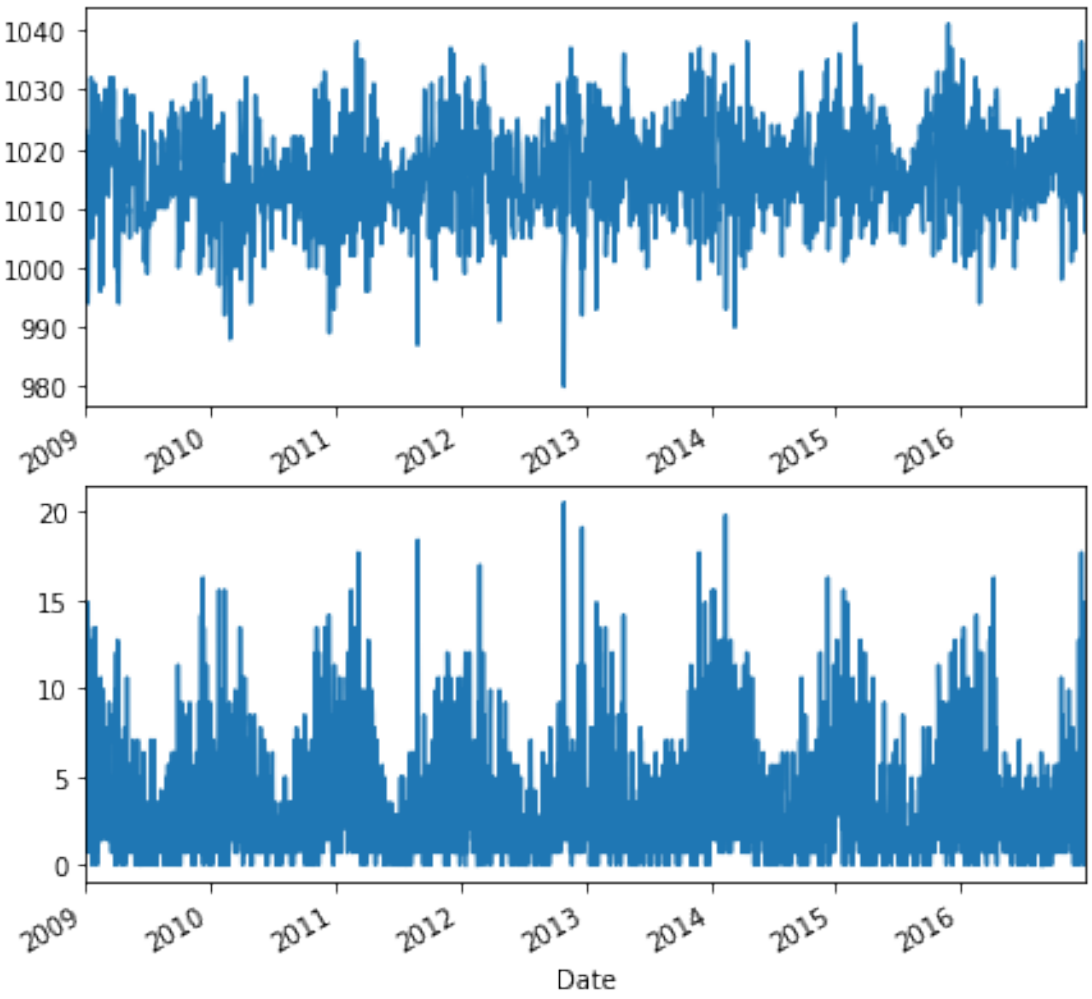
Standard Deviation of Humidityă(%) low



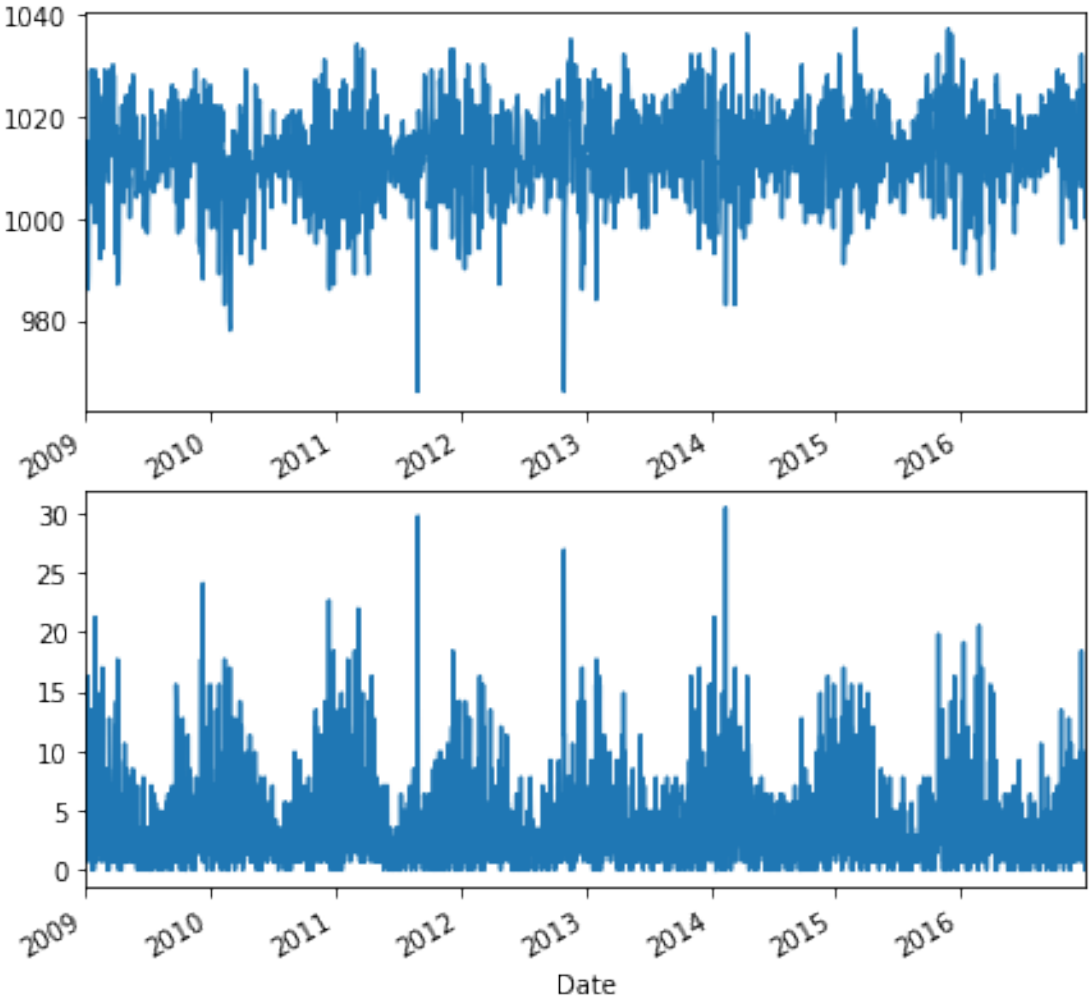
Standard Deviation of Sea Level Press.ã(hPa) high



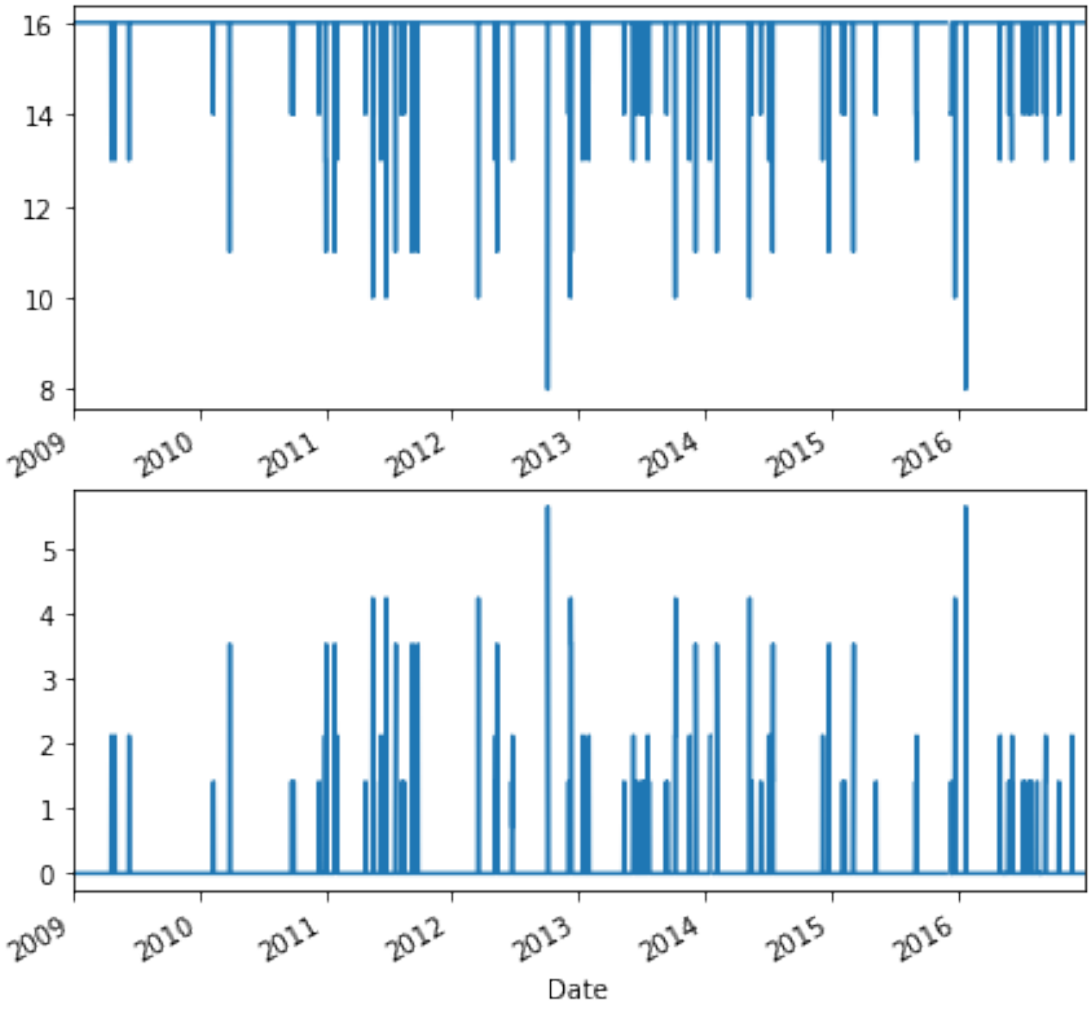
Standard Deviation of Sea Level Press.ă(hPa) avg



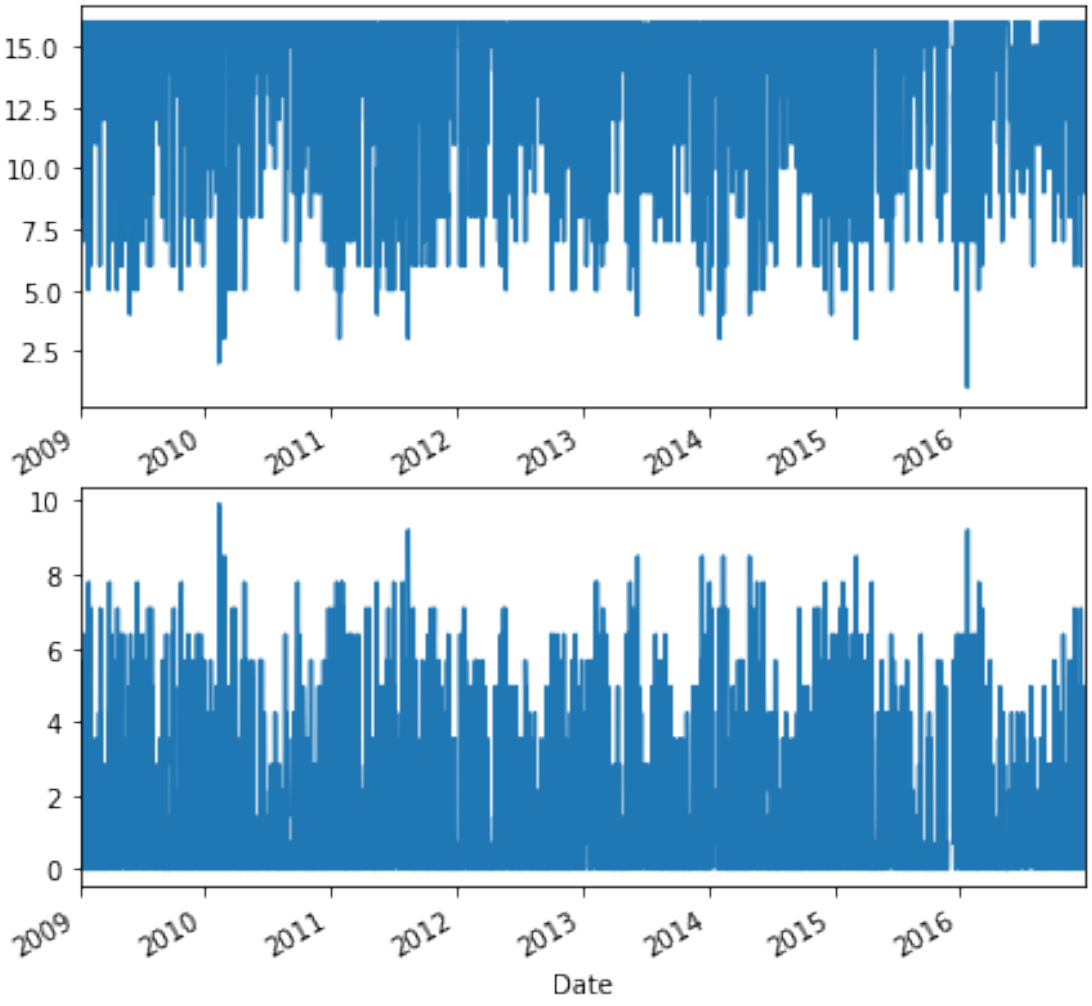
Standard Deviation of Sea Level Press.ă(hPa) low



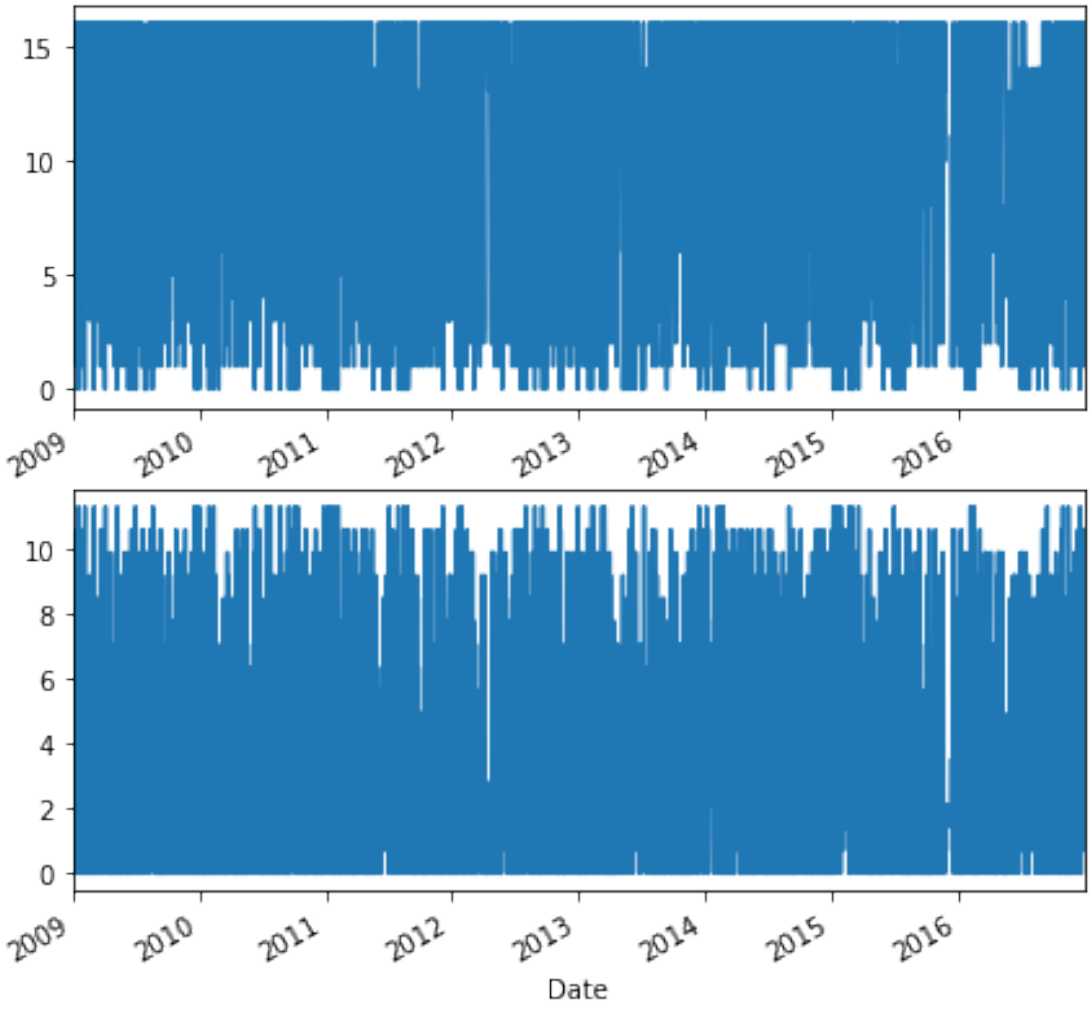
Standard Deviation of Visibilityă(km) high



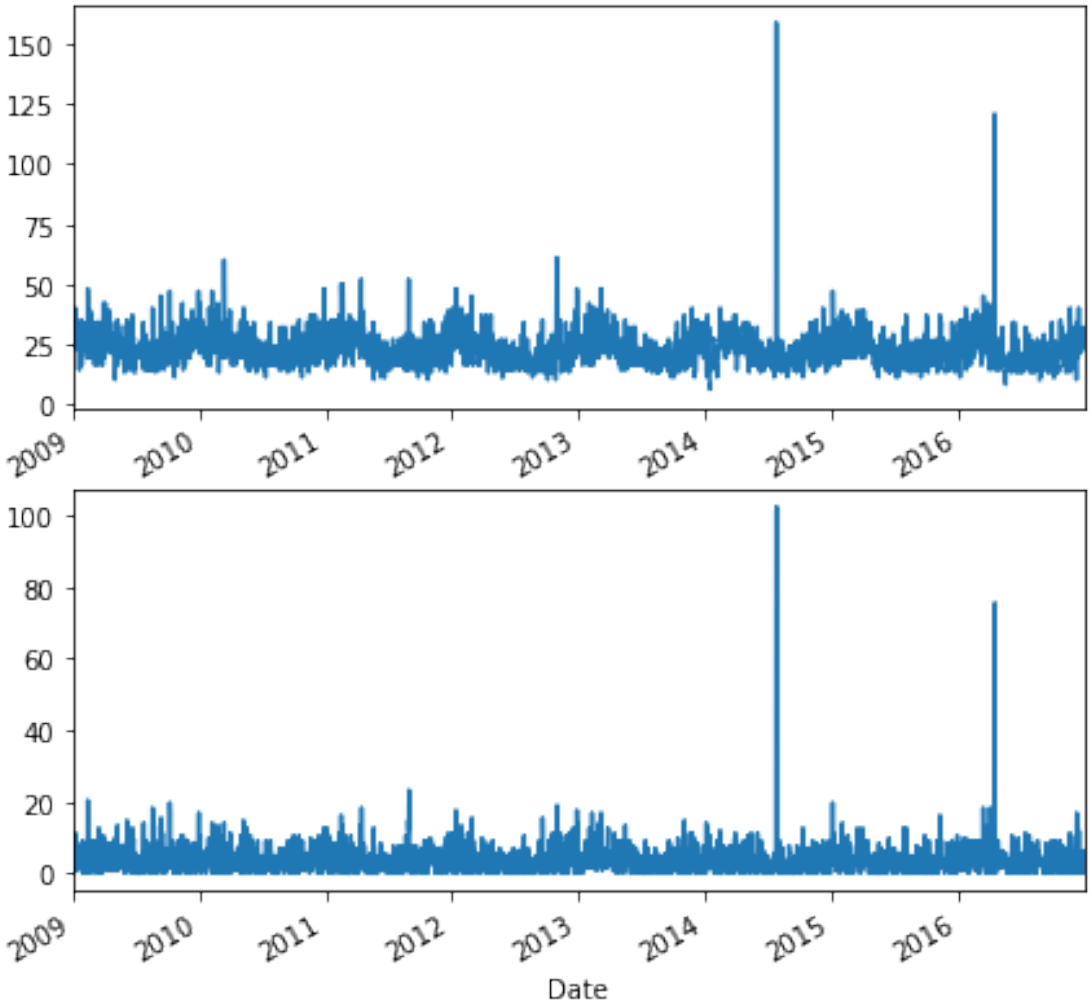
Standard Deviation of Visibilityã(km) avg



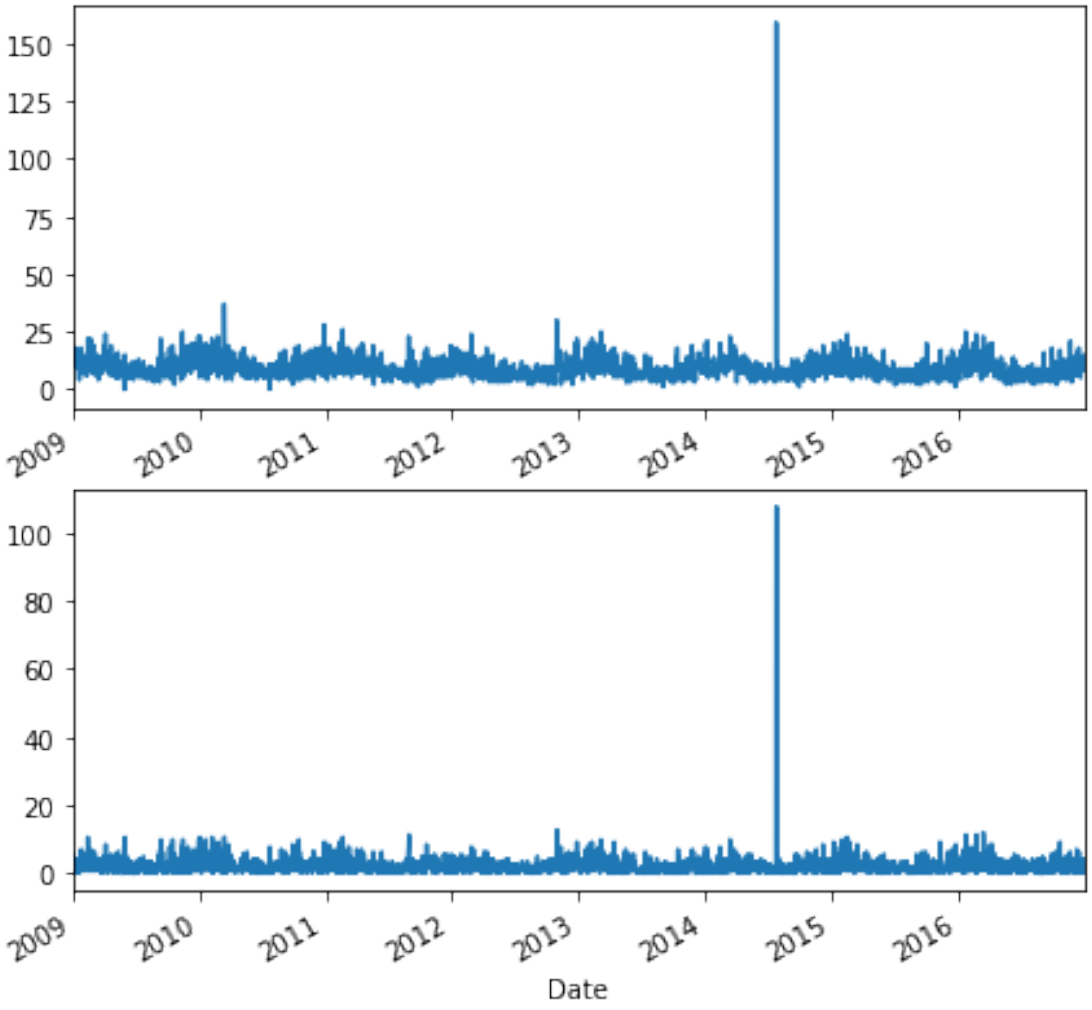
Standard Deviation of Visibilityã(km) low



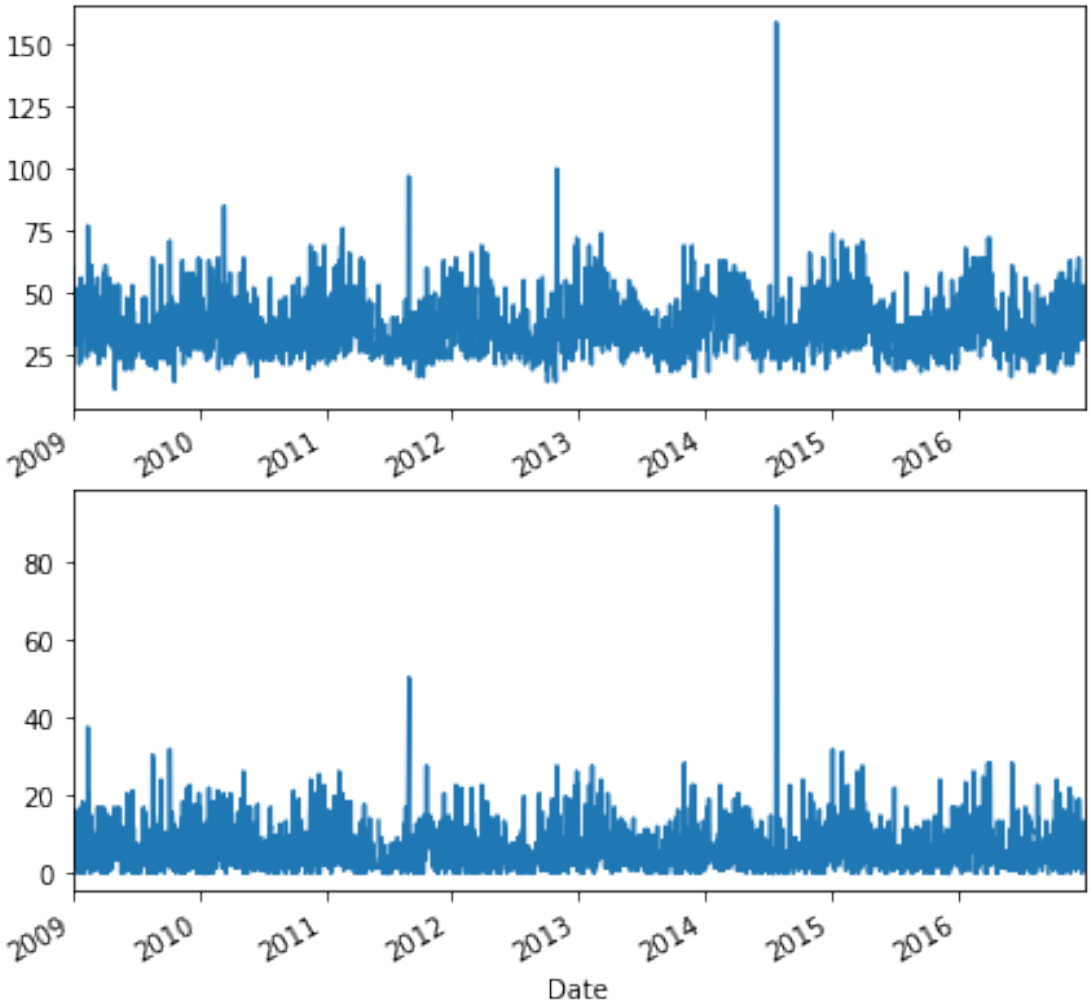
Standard Deviation of Windă(km/h) low



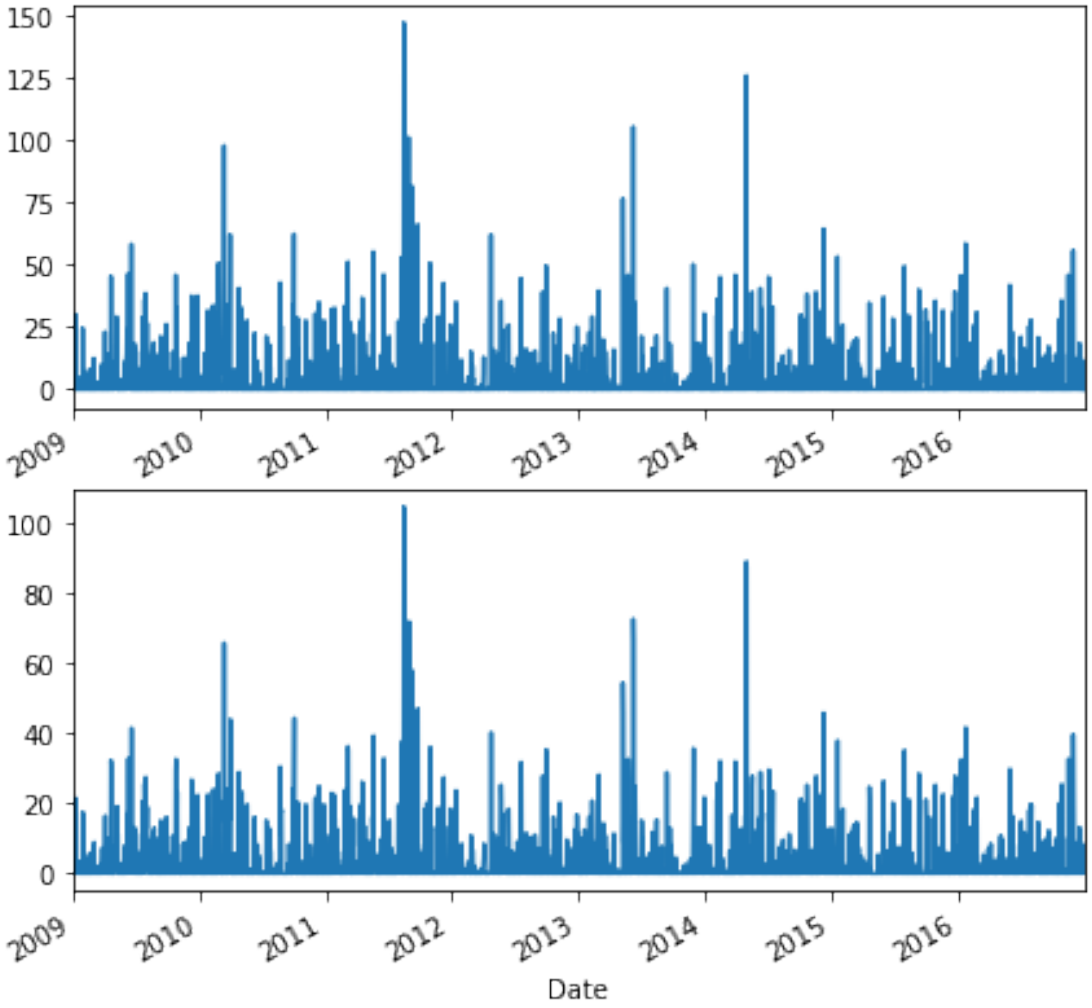
Standard Deviation of Windă(km/h) avg



Standard Deviation of Windă(km/h) high



Standard Deviation of Precip.ă(mm) sum




```
In [48]: ### 3.4.4 Converting 'WeatherEvent' into dummy variables
         weather_dummies = pd.get_dummies(weather_data.WeatherEvent, prefix='WeatherEvent')
         weather_dummies.drop(weather_dummies.columns[8],1, inplace = True)
         weather_data = pd.concat([weather_data, weather_dummies], axis =1)
         weather_data.drop(weather_data.columns[19],1, inplace = True)
```

```
In [49]: # Resampling the weather data with the mean of each month
         weather_data_mean = weather_data.resample('M').mean()
```

5 4. Merging the Dataframes

```
In [50]: # Synchronising the index before merging
         holidays_data.index = economic_data.index
         weather_data_mean.index = economic_data.index
```

```
merged_data = economic_data.join(weather_data_mean)
merged_data = merged_data.join(holidays_data)
```

```
In [51]: merged_data.to_csv("preprocessed_merged_data.csv")
```

6 5. Splitting the data into Train, Test and Validation

6.1 5.1. Splitting the Target Variable into Train and Validation

```
In [52]: split_validation_time = pd.to_datetime('2015-01-01')
         split_test_time = pd.to_datetime('2016-01-01')
         sales_data.drop(sales_data.columns[0],1,inplace = True)
         target_validation = sales_data.loc[sales_data.index >= split_validation_time]
         target_train = sales_data.loc[sales_data.index < split_validation_time]
```

6.2 5.2. Spliting the merged data into Train, Validation and Test

```
In [53]: merged_test = merged_data.loc[merged_data.index >= split_test_time]
         merged_validation = merged_data.loc[(merged_data.index >= split_validation_time) \
                                             & (merged_data.index < split_test_time)]
         merged_train = merged_data.loc[merged_data.index < split_validation_time]
```

7 6. Comparing the features

7.1 6.1. Identifing Feature Importance using ANOVA

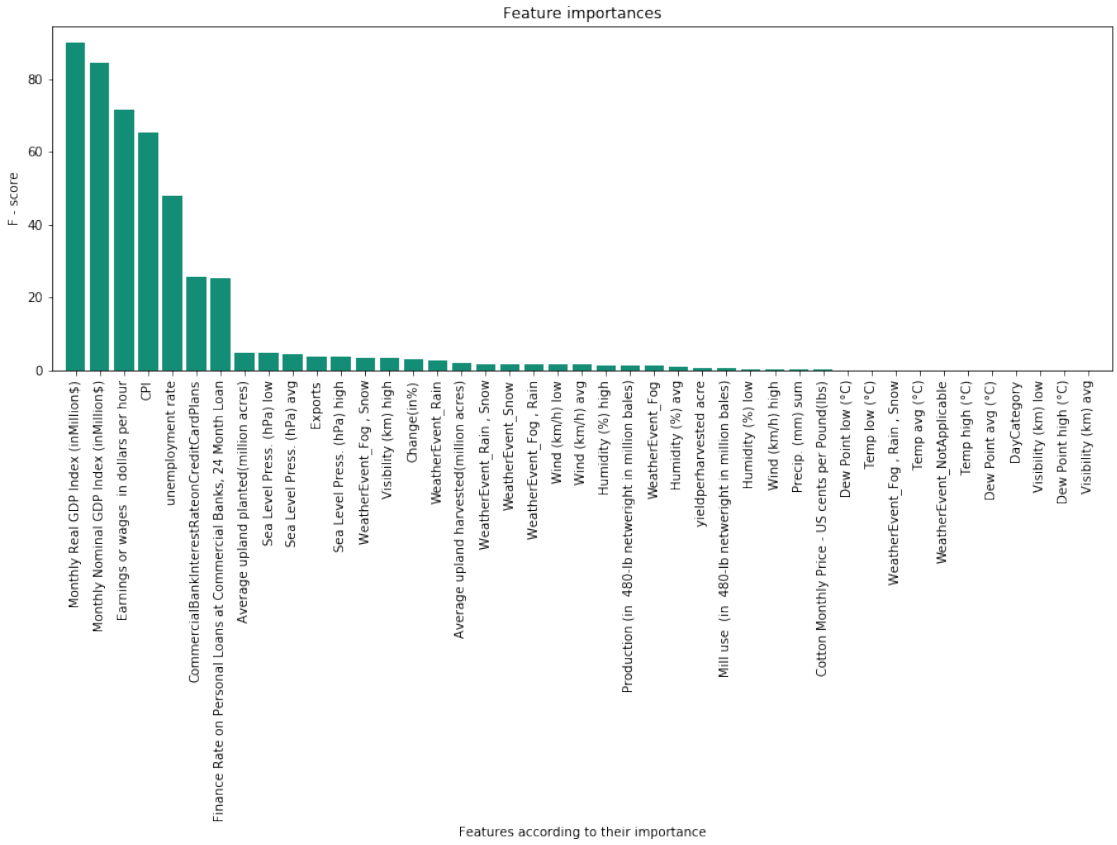
```
In [54]: sel = SelectKBest(k =5, score_func = f_regression)
         sel.fit(merged_train, target_train)
```

```
C:\Users\Thomas\AppData\Local\conda\conda\envs\py36\lib\site-packages\sklearn\utils\validation
  y = column_or_1d(y, warn=True)
```

```
Out[54]: SelectKBest(k=5, score_func=<function f_regression at 0x00000272E8D2EA60>)
```

```
In [55]: important_features = []
         importances = sel.scores_
         indices = np.argsort(importances)[::-1]
         for f in range(merged_train.shape[1]):
             important_features = np.append(important_features, merged_train.columns.values[inc

         plt.figure(figsize =(15,5))
         plt.title("Feature importances")
         plt.bar(range(merged_train.shape[1]), importances[indices],
                color="#138D75", align="center")
         plt.xticks(range(merged_train.shape[1]), important_features, rotation = 90)
         plt.xlim([-1, merged_train.shape[1]])
         plt.ylabel('F - score')
         plt.xlabel('Features according to their importance')
         plt.show()
```



7.2 6.2. Checking for Multicolinearity using Variable Inflation Factor (VIF)

```
In [56]: x_var = merged_train
         y_var = target_train
         x_var_names = merged_train.columns
         for i in range(0, len(x_var_names)):
             y = x_var[x_var_names[i]]
             x = x_var[x_var_names.drop(x_var_names[i])]
             lr1 = LinearRegression()
             rsq = lr1.fit(x,y).score(x,y)
             vif = round(1/(1-rsq),2)
             print (x_var_names[i], " VIF = " , vif)
```

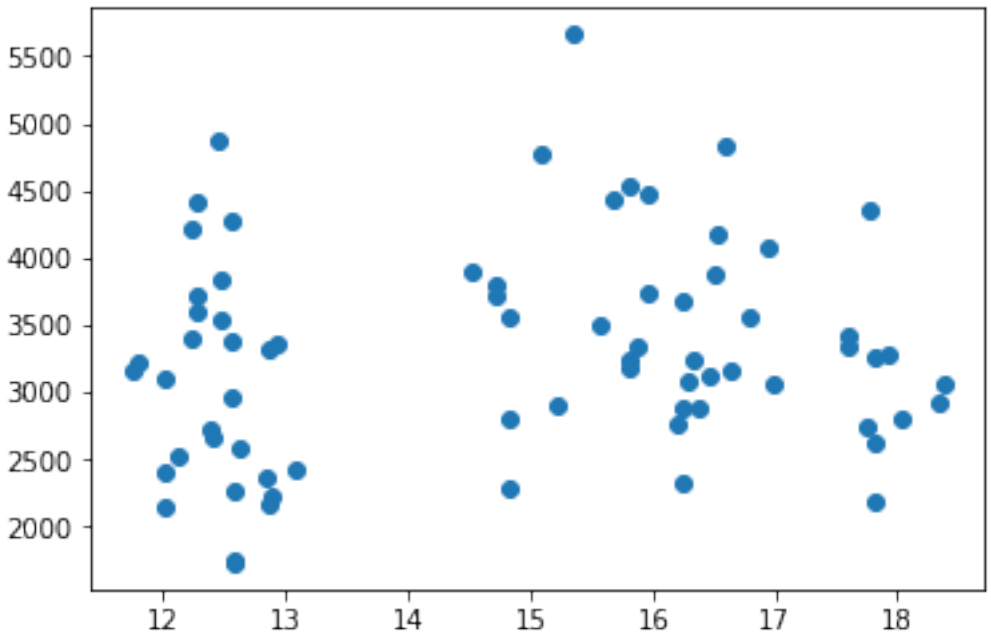
```
Monthly Nominal GDP Index (inMillion$)  VIF =  7354.23
Monthly Real GDP Index (inMillion$)  VIF =  2762.54
CPI  VIF =  722.8
unemployment rate  VIF =  196.72
CommercialBankInterestRateonCreditCardPlans  VIF =  29.37
Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan  VIF =  7.65
Earnings or wages  in dollars per hour  VIF =  157.48
Cotton Monthly Price - US cents per Pound(lbs)  VIF =  14.09
```

```
Change(in%)  VIF =  4.1
Average upland planted(million acres)  VIF =  11.11
Average upland harvested(million acres)  VIF =  425.24
yieldperharvested acre  VIF =  28.39
Production (in  480-lb netweright in million bales)  VIF =  499.07
Mill use  (in  480-lb netweright in million bales)  VIF =  10.51
Exports  VIF =  12.79
Temp high (řC)  VIF =  14787.69
Temp avg (řC)  VIF =  51237.89
Temp low (řC)  VIF =  15049.01
Dew Point high (řC)  VIF =  2956.84
Dew Point avg (řC)  VIF =  4524.26
Dew Point low (řC)  VIF =  2476.58
Humidityă(%) high  VIF =  7516.24
Humidityă(%) avg  VIF =  27894.56
Humidityă(%) low  VIF =  8272.58
Sea Level Press.ă(hPa) high  VIF =  412.21
Sea Level Press.ă(hPa) avg  VIF =  911.76
Sea Level Press.ă(hPa) low  VIF =  211.55
Visibilityă(km) high  VIF =  7.11
Visibilityă(km) avg  VIF =  22.48
Visibilityă(km) low  VIF =  25.29
Windă(km/h) low  VIF =  46.32
Windă(km/h) avg  VIF =  51.48
Windă(km/h) high  VIF =  47.27
Precip.ă(mm) sum  VIF =  6.03
WeatherEvent_Fog  VIF =  35.1
WeatherEvent_Fog , Rain  VIF =  510.08
WeatherEvent_Fog , Rain , Snow  VIF =  89.8
WeatherEvent_Fog , Snow  VIF =  188.8
WeatherEvent_NotApplicable  VIF =  1486.79
WeatherEvent_Rain  VIF =  1807.61
WeatherEvent_Rain , Snow  VIF =  66.07
WeatherEvent_Snow  VIF =  401.16
DayCategory  VIF =  3.94
```

8 7. Building Models

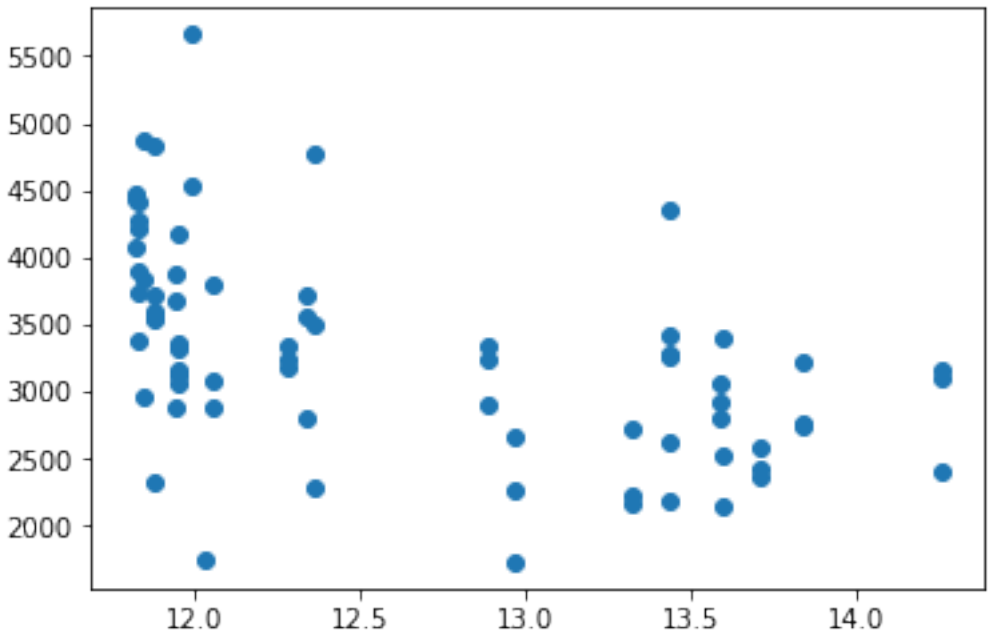
8.1 7.1 Linear Regression Model

```
In [57]: plt.scatter(merged_train[merged_train.columns[12]], target_train["Sales(In ThousandDo
         plt.show()
```

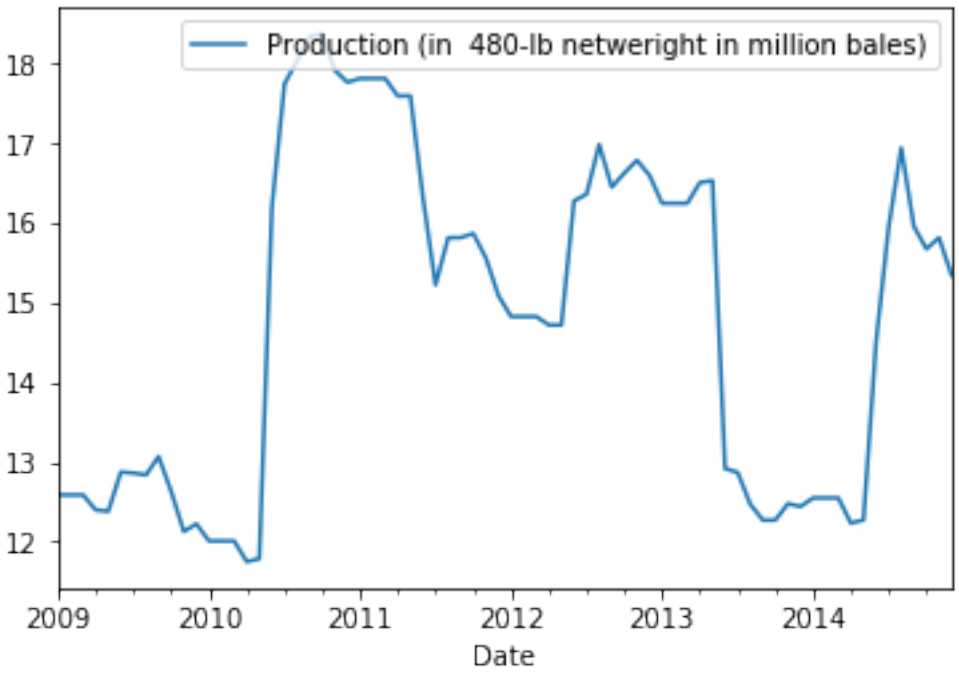



```
In [58]: plt.scatter(merged_train[merged_train.columns[4]], target_train["Sales(In ThousandDoll
         print(merged_train[merged_train.columns[4]].corr(target_train["Sales(In ThousandDollar
```

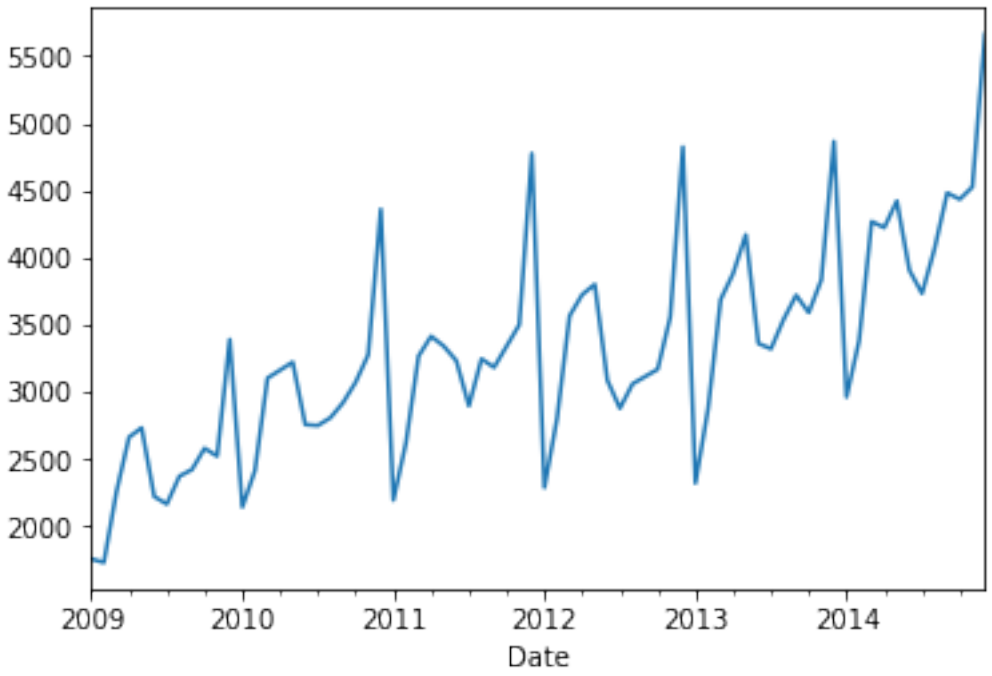
```
-0.517847851408
```



In [59]:
```
merged_train.loc[:,[merged_train.columns[12]]].plot()
plt.show()
```



In [60]:
```
target_train["Sales(In ThousandDollars)"].plot()
plt.show()
```




```
In [61]: model = OLS(target_train,merged_train).fit()
         ols_predicted_train = model.predict(merged_train)
         print(model.summary())
```

```
                                OLS Regression Results
==============================================================================
Dep. Variable:     Sales(In ThousandDollars)   R-squared:                       0.989
Model:                                   OLS   Adj. R-squared:                  0.973
Method:                        Least Squares   F-statistic:                     61.66
Date:                       Sun, 26 Nov 2017   Prob (F-statistic):           7.09e-20
Time:                               16:13:53   Log-Likelihood:                -524.31
No. Observations:                         72   AIC:                             1135.
Df Residuals:                             29   BIC:                             1233.
Df Model:                                 43
Covariance Type:                   nonrobust
================================================================================================
                                                                  coef    std err
------------------------------------------------------------------------------------------------
Monthly Nominal GDP Index (inMillion$)                          8.1343      4.454      1
Monthly Real GDP Index (inMillion$)                            -7.5007      5.277     -1
CPI                                                          -302.1183    167.595     -1
unemployment rate                                             312.1918    685.736      0
CommercialBankInterestRateonCreditCardPlans                   560.0691    428.146      1
Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan -106.7940    410.303     -0
Earnings or wages  in dollars per hour                        -791.6366   1021.048     -0
```

Cotton Monthly Price - US cents per Pound(lbs)	4.5905	6.118	0
Change(in%)	-15.1929	16.181	-0
Average upland planted(million acres)	159.4337	122.893	1
Average upland harvested(million acres)	1445.0112	1128.219	1
yieldperharvested acre	15.7746	12.985	1
Production (in 480-lb netweright in million bales)	-1005.2075	686.948	-1
Mill use (in 480-lb netweright in million bales)	-1058.8116	1199.338	-0
Exports	91.0773	128.228	0
Temp high (řC)	-729.7320	866.379	-0
Temp avg (řC)	1803.9137	1689.680	1
Temp low (řC)	-1093.5668	954.226	-1
Dew Point high (řC)	210.1458	412.324	0
Dew Point avg (řC)	-366.3975	488.564	-0
Dew Point low (řC)	144.6490	340.036	0
Humidityă(%) high	-742.4549	866.947	-0
Humidityă(%) avg	1577.3201	1738.392	0
Humidityă(%) low	-754.5113	890.679	-0
Sea Level Press.ă(hPa) high	375.7656	406.981	0
Sea Level Press.ă(hPa) avg	-896.3900	710.507	-1
Sea Level Press.ă(hPa) low	604.4286	358.400	1
Visibilityă(km) high	-917.2409	1308.837	-0
Visibilityă(km) avg	163.3153	344.987	0
Visibilityă(km) low	32.9727	212.157	0
Windă(km/h) low	-118.1543	135.378	-0
Windă(km/h) avg	68.0206	221.639	0
Windă(km/h) high	128.2112	103.167	1
Precip.ă(mm) sum	83.4529	68.934	1
WeatherEvent_Fog	-2.411e+04	2.92e+04	-0
WeatherEvent_Fog , Rain	-1.974e+04	2.71e+04	-0
WeatherEvent_Fog , Rain , Snow	-1.998e+04	2.86e+04	-0
WeatherEvent_Fog , Snow	-1.988e+04	2.67e+04	-0
WeatherEvent_NotApplicable	-1.843e+04	2.74e+04	-0
WeatherEvent_Rain	-1.937e+04	2.72e+04	-0
WeatherEvent_Rain , Snow	-2.414e+04	2.84e+04	-0
WeatherEvent_Snow	-1.907e+04	2.72e+04	-0
DayCategory	13.8530	40.335	0

Omnibus:	3.813	Durbin-Watson:	2.350
Prob(Omnibus):	0.149	Jarque-Bera (JB):	3.095
Skew:	0.346	Prob(JB):	0.213
Kurtosis:	3.743	Cond. No.	2.61e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.61e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

```
In [62]: print(mean_absolute_percentage_error(target_train, ols_predicted_train))
```

```
26.9818291327
```

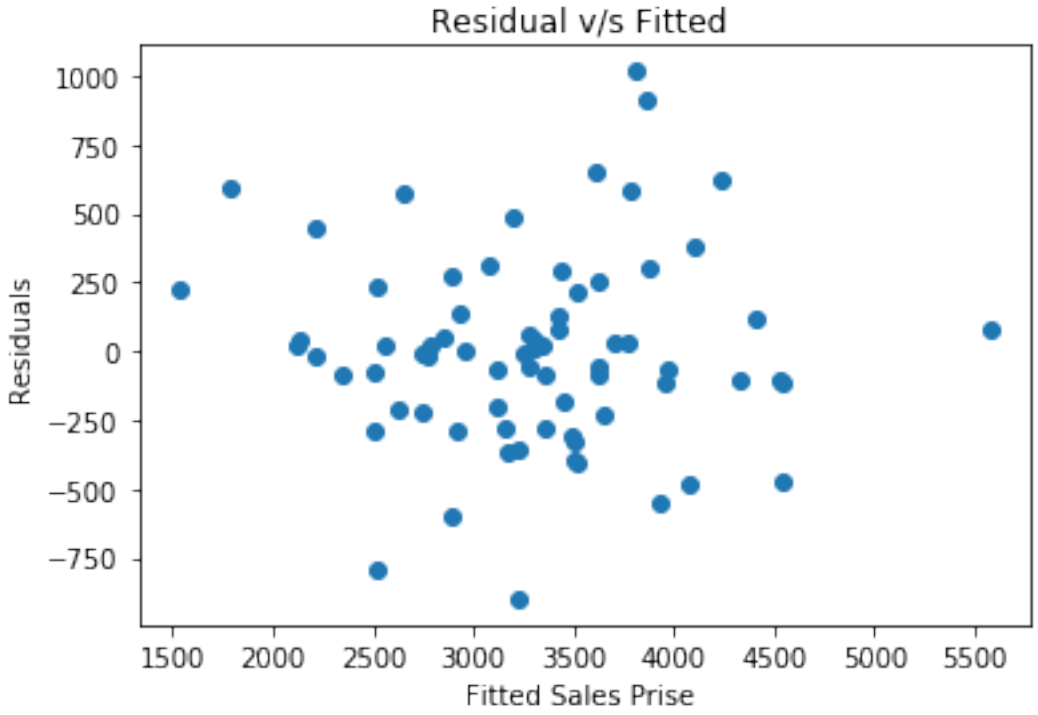
```
In [63]: ols_predicted_validation = model.predict(merged_validation)
```

```
In [64]: print(mean_absolute_percentage_error(target_validation, ols_predicted_validation))
```

```
37.9081812406
```

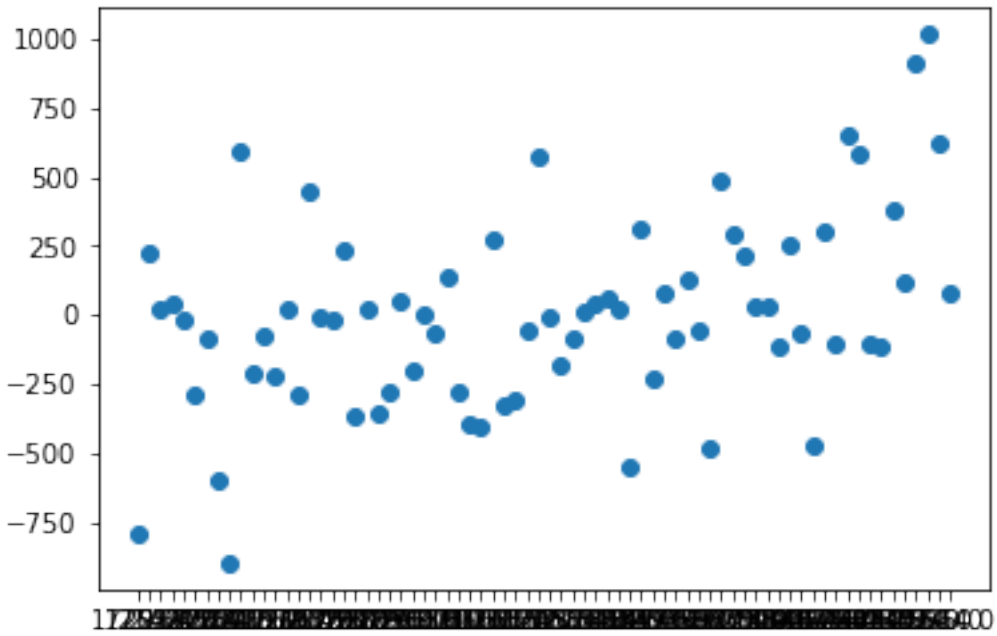
8.1.1 Checking for the Residuals v/s Fitted Y

```
In [65]: plt.scatter(model.fittedvalues, model.resid)
         plt.xlabel('Fitted Sales Prise ')
         plt.ylabel('Residuals')
         plt.title('Residual v/s Fitted ')
         plt.show()
```



8.1.2 Checking for Hetroscedasticity

```
In [66]: # Heteroscedasticity
         plt.scatter(target_train,model.resid)
         plt.show()
```



8.2 7.2 ElasticNetCV

```
In [67]: X_train = np.array(merged_train)
         X_train = preprocessing.scale(X_train)

         X_val = np.array(merged_validation)
         X_val = preprocessing.scale(X_val)

         X_test = np.array(merged_test)
         X_test = preprocessing.scale(merged_test)

         Y_train = np.array(target_train[target_train.columns[0]]).ravel()
         Y_val = np.array(target_validation[target_validation.columns[0]]).ravel()
```

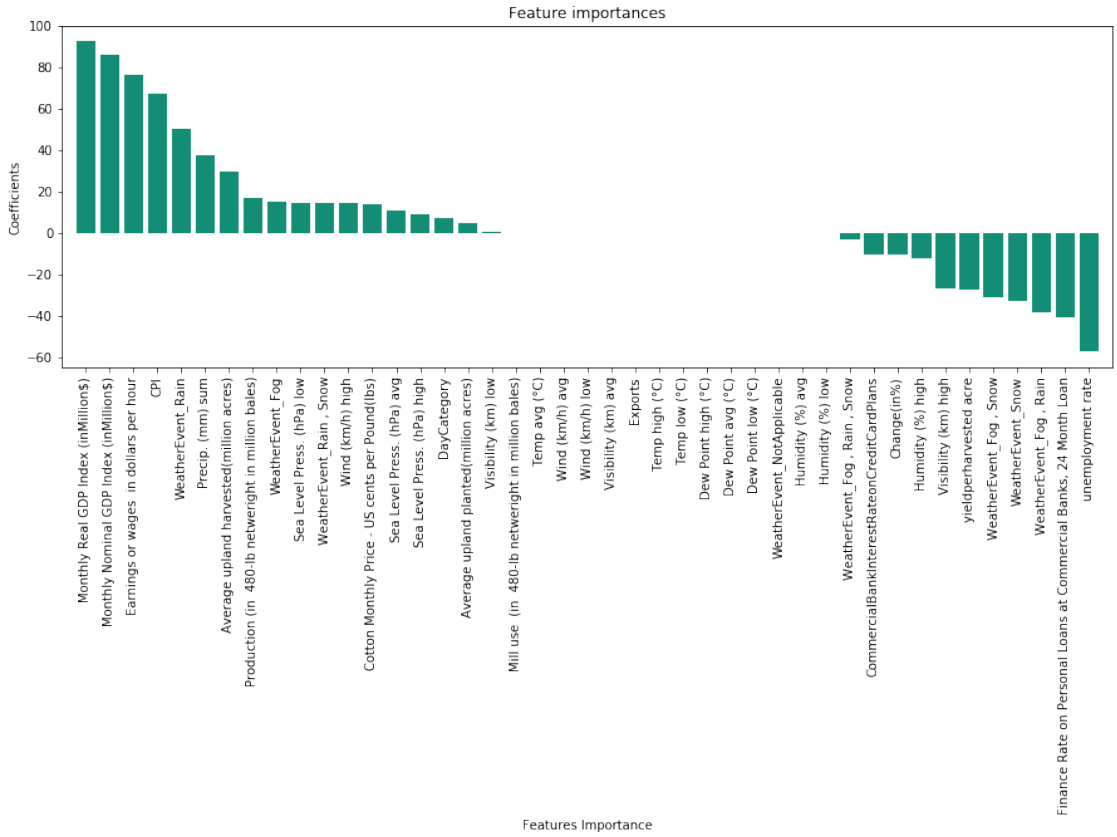
```
In [68]: from sklearn.linear_model import ElasticNetCV
         regr = ElasticNetCV(copy_X=True, cv=20, eps=0.001, fit_intercept=True,
                l1_ratio=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], max_iter=1000, n_alphas
                normalize=False, positive=False, precompute='auto', random_state=0,
                selection='cyclic', tol=0.0001, verbose=0)
         elastic_predicted_train = regr.fit(X_train,Y_train).predict(X_train)
         print(mean_absolute_percentage_error(Y_train,elastic_predicted_train))
         elastic_predicted_val = regr.predict(X_val)
         print(mean_absolute_percentage_error(Y_val,elastic_predicted_val))
         elastic_predicted_test = regr.predict(X_test)
```

```
11.9625829592
```

```
20.0481391264
```

```
In [69]: important_features = []
         importances = regr.coef_
         indices = np.argsort(importances)[::-1]
         for f in range(merged_train.shape[1]):
             important_features = np.append(important_features, merged_train.columns.values[in

         plt.figure(figsize =(15,5))
         plt.title("Feature importances")
         plt.bar(range(merged_train.shape[1]), importances[indices],
                color="#138D75", align="center")
         plt.xticks(range(merged_train.shape[1]), important_features, rotation = 90)
         plt.xlim([-1, merged_train.shape[1]])
         plt.ylabel('Coefficients')
         plt.xlabel('Features Importance')
         plt.show()
```




```
In [70]: elastic_predicted_test = pd.DataFrame(elastic_predicted_test)
```

In [71]:
```
### Reading the template file
final_result = pd.read_csv("template.csv")

final_result = final_result.loc[final_result.ProductCategory == "WomenClothing",].copy

final_result.drop("target",1, inplace= True )
final_result = final_result.join(elastic_predicted_test)
final_result = final_result.rename(columns = {final_result.columns[3]:'target'})

final_result.to_csv("prediction-elasticnet.csv")
```

8.3 7.3 Support Vector Machines

In [72]:
```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_lin = SVR(kernel='linear', C=1e3)

parameter_values = [
  {'C': [35, 40, 50, 60, 70, 80, 90], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
   'kernel': ['rbf'], 'degree':[1, 2, 3]},
  {'C': [35, 40, 50, 60, 70, 80, 90], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
   'kernel': ['poly'], 'degree':[1,2, 3, 4]},
  {'C': [35, 40, 50, 60, 70, 80, 90], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
   'kernel': ['linear'], 'degree':[1,2, 3, 4]}
]
svr_grid = GridSearchCV(estimator= SVR(), param_grid=parameter_values, n_jobs=-1)


Y_predicted_train_rbf = svr_rbf.fit(X_train,Y_train).predict(X_train)
Y_predicted_train_lin = svr_lin.fit(X_train,Y_train).predict(X_train)
Y_predicted_train_grid = svr_grid.fit(X_train,Y_train).predict(X_train)

print(mean_absolute_percentage_error(Y_train,Y_predicted_train_rbf))
print(mean_absolute_percentage_error(Y_train,Y_predicted_train_lin))
print(mean_absolute_percentage_error(Y_train,Y_predicted_train_grid))
```

```
1.64492610964
7.54982746538
10.709251259
```

In [73]:
```
Y_predicted_validation_rbf = svr_rbf.fit(X_train,Y_train).predict(X_val)
Y_predicted_validation_lin = svr_lin.fit(X_train,Y_train).predict(X_val)
Y_predicted_validation_grid = svr_grid.fit(X_train,Y_train).predict(X_val)

print(mean_absolute_percentage_error(Y_val,Y_predicted_validation_rbf.ravel()))
print(mean_absolute_percentage_error(Y_val,Y_predicted_validation_lin.ravel()))
print(mean_absolute_percentage_error(Y_val,Y_predicted_validation_grid.ravel()))
```

```
20.0387855479
29.8213388858
20.4509680696
```

8.4 7.4 RandomForest

When the dataset has many highly correlated features, then from the point of view of the model, any of these correlated features can be used as the predictor, with no concrete preference of one over the others. But once one of them is used, the importance of others is significantly reduced since effectively the impurity they can remove is already removed by the first feature. As a consequence, they will have a lower reported importance. This is not an issue when we want to use feature selection to reduce overfitting, since it makes sense to remove features that are mostly duplicated by other features. But when interpreting the data, it can lead to the incorrect conclusion that one of the variables is a strong predictor while the others in the same group are unimportant, while actually they are very close in terms of their relationship with the response variable.

```
In [74]: # RandomForest with cross validation
         parameter_value = [
           {'n_estimators': [75], 'max_features': [9, 20, 25],'oob_score' : [True], 'criterion
         ]
         rf_grid = GridSearchCV(estimator= RandomForestRegressor(),param_grid=parameter_value,

         rf_grid_predicted_train = rf_grid.fit(merged_train,target_train).predict(merged_train)

         print(mean_absolute_percentage_error(target_train,rf_grid_predicted_train))
         rf_grid_predicted_val = rf_grid.predict(merged_validation)
         print(mean_absolute_percentage_error(target_validation,rf_grid_predicted_val))
```

```
C:\Users\Thomas\AppData\Local\conda\conda\envs\py36\lib\site-packages\sklearn\model_selection\_
  self.best_estimator_.fit(X, y, **fit_params)


26.4791169291
14.5133094642
```

```
In [75]: rf_grid.best_estimator_
```

```
Out[75]: RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=80,
                    max_features=20, max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=75, n_jobs=1, oob_score=True, random_state=None,
                    verbose=0, warm_start=False)
```

```
In [76]: rf_grid_predicted_test = rf_grid.predict(merged_test)
         rf_grid_predicted_test = pd.DataFrame(rf_grid_predicted_test)
```

```
In [77]: ### Reading the template file
         final_result = pd.read_csv("template.csv")

         final_result = final_result.loc[final_result.ProductCategory == "WomenClothing",].copy

         final_result.drop("target",1, inplace= True )
         final_result = final_result.join(rf_grid_predicted_test)
         final_result = final_result.rename(columns = {final_result.columns[3]:'target'})

         final_result.to_csv("prediction-rf-grid.csv")

In [78]: # Random Forest
         rf = RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=75,
                    max_features=20, max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=1000, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)
         rf_predicted_train = rf.fit(merged_train,target_train).predict(merged_train)

         print(mean_absolute_percentage_error(target_train,rf_predicted_train))
         rf_predicted_val = rf.predict(merged_validation)
         print(mean_absolute_percentage_error(target_validation,rf_predicted_val))

C:\Users\Thomas\AppData\Local\conda\conda\envs\py36\lib\site-packages\ipykernel_launcher.py:8:



26.4882283219
14.37205227
```

8.4.1 Ranking the features based on their importance in the forest.

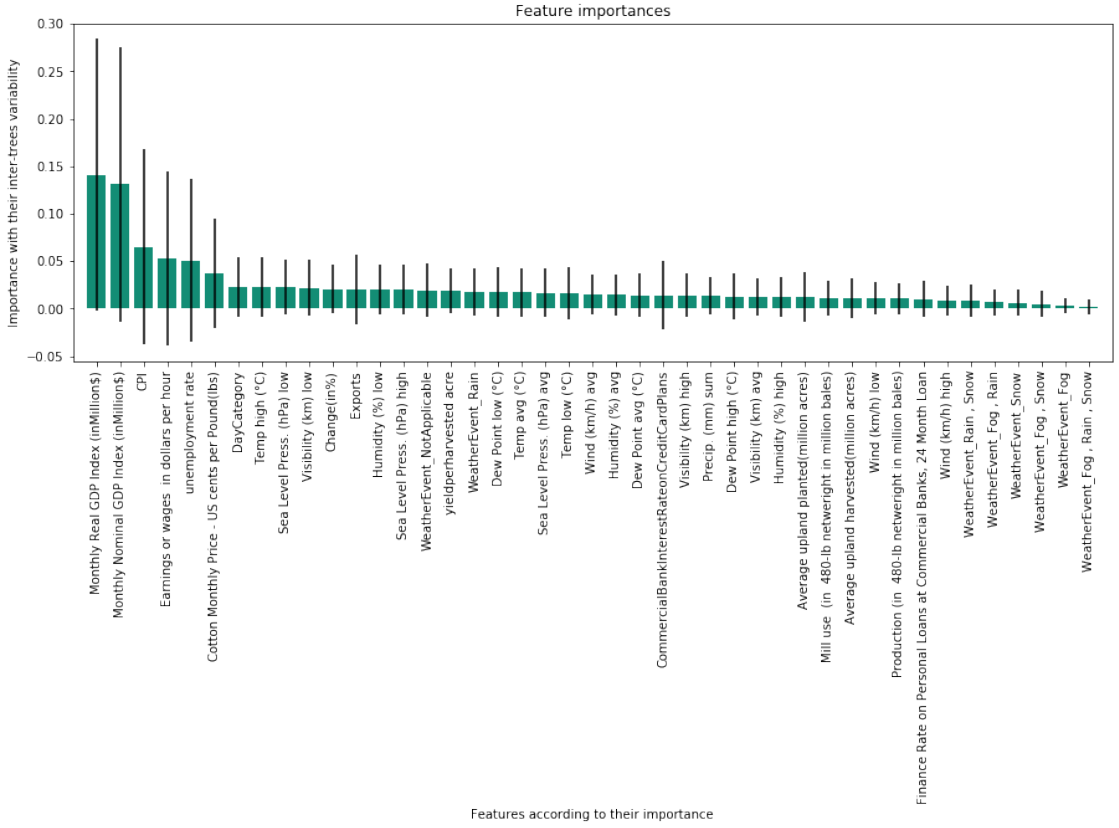
```
In [79]: important_features = []
         importances = rf.feature_importances_
         indices = np.argsort(importances)[::-1]
         print("*************************************")
         print("*                                   *")
         print("*      Ranking of the features      *")
         print("*                                   *")
         print("*************************************")
         for f in range(merged_train.shape[1]):
             print("%d. %s (%f)" % (f+1, merged_train.columns.values[indices[f]], importances[
             important_features = np.append(important_features, merged_train.columns.values[in

*************************************
*                                   *
*      Ranking of the features      *
```

```
*                                    *
************************************
1. Monthly Real GDP Index (inMillion$) (0.140743)
2. Monthly Nominal GDP Index (inMillion$) (0.130897)
3. CPI (0.065103)
4. Earnings or wages  in dollars per hour (0.052644)
5. unemployment rate (0.050833)
6. Cotton Monthly Price - US cents per Pound(lbs) (0.037081)
7. DayCategory (0.022730)
8. Temp high (řC) (0.022630)
9. Sea Level Press.ă(hPa) low (0.022613)
10. Visibilityă(km) low (0.022041)
11. Change(in%) (0.020611)
12. Exports (0.020504)
13. Humidityă(%) low (0.020389)
14. Sea Level Press.ă(hPa) high (0.020207)
15. WeatherEvent_NotApplicable (0.019419)
16. yieldperharvested acre (0.018794)
17. WeatherEvent_Rain (0.017857)
18. Dew Point low (řC) (0.017115)
19. Temp avg (řC) (0.016955)
20. Sea Level Press.ă(hPa) avg (0.016755)
21. Temp low (řC) (0.016150)
22. Windă(km/h) avg (0.015132)
23. Humidityă(%) avg (0.014457)
24. Dew Point avg (řC) (0.013966)
25. CommercialBankInterestRateonCreditCardPlans (0.013904)
26. Visibilityă(km) high (0.013856)
27. Precip.ă(mm) sum (0.013254)
28. Dew Point high (řC) (0.012933)
29. Visibilityă(km) avg (0.012582)
30. Humidityă(%) high (0.012272)
31. Average upland planted(million acres) (0.012130)
32. Mill use  (in  480-lb netweright in million bales) (0.011167)
33. Average upland harvested(million acres) (0.011124)
34. Windă(km/h) low (0.010723)
35. Production (in  480-lb netweright in million bales) (0.010453)
36. Finance Rate on Personal Loans at Commercial Banks, 24 Month Loan (0.010116)
37. Windă(km/h) high (0.008685)
38. WeatherEvent_Rain , Snow (0.008453)
39. WeatherEvent_Fog , Rain (0.006442)
40. WeatherEvent_Snow (0.005891)
41. WeatherEvent_Fog , Snow (0.005066)
42. WeatherEvent_Fog (0.003003)
43. WeatherEvent_Fog , Rain , Snow (0.002319)
```

8.4.2 Plotting Feature importance with Random Forest model

```
In [80]: std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
         plt.figure(figsize =(15,5))
         plt.title("Feature importances")
         plt.bar(range(merged_train.shape[1]), importances[indices],
                color="#138D75", yerr=std[indices], align="center")
         plt.xticks(range(merged_train.shape[1]), important_features, rotation = 90)
         plt.xlim([-1, merged_train.shape[1]])
         plt.ylabel('Importance with their inter-trees variability')
         plt.xlabel('Features according to their importance')
         plt.show()
```




```
In [81]: rf_predicted_test = rf.predict(merged_test)
         rf_predicted_test = pd.DataFrame(rf_predicted_test)
```

```
In [82]: ### Reading the template file
         final_result = pd.read_csv("template.csv")

         final_result = final_result.loc[final_result.ProductCategory == "WomenClothing",].copy

         final_result.drop("target",1, inplace= True )
```

```
final_result = final_result.join(rf_predicted_test)
final_result = final_result.rename(columns = {final_result.columns[3]:'target'})

final_result.to_csv("prediction-rf.csv")
```