Project Air Quality

Domain Name: Environment Air quality

Abstract:

Contains the responses of a gas multisensor device deployed on the field in an Italian city. Hourly responses averages are recorded along with gas concentrations references from a certified analyzer.

Dataset: Air quality of an Italian city

(https://archive.ics.uci.edu/ml/datasets/Air+quality)

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer.

Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129,2,2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.

Attributes of the dataset are:

Sl No		Attribute		Description
0		Date		Date (DD/MM/YYYY)
1		Time		Time (HH.MM.SS)
2		CO(GT)		True hourly averaged concentration CO in mg/m^3 (reference analyzer)
3		PT08.S1(CO)		PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
4		NMHC(GT)		True hourly averaged overall Non Metanic HydroCarbons

				concentration in microg/m^3 (reference analyzer)
5		C6H6(GT)		True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)
6		PT08.S2(NMHC)		PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
7		NOx(GT)		True hourly averaged NOx concentration in ppb (reference analyzer)
8		PT08.S3(NOx)		PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
9		NO2(GT)		True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)
10		PT08.S4(NO2)		PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
11		PT08.S5(O3)		PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
12		T		Temperature in Â°C
13		RH		Relative Humidity (%)
14		AH		AH Absolute Humidity

Problem:

Humans are very sensitive to humidity, as the skin relies on the air to get rid of moisture. The process of sweating is your body's attempt to keep cool and maintain its current temperature. If the air is at 100-

percent relative humidity, sweat will not evaporate into the air. As a result, we feel much hotter than the actual temperature when the relative humidity is high. If the relative humidity is low, we can feel much cooler than the actual temperature because our sweat evaporates easily, cooling us off. For example, if the air temperature e is 75 degrees Fahrenheit (24 degrees Celsius) and the relative humidity is zero percent, the air temperature feels like 69 degrees Fahrenheit (21 C) to our bodies. If the air temperature is 75 degrees Fahrenheit (24 C) and the relative humidity is 100 percent, we feel like it's 80 degrees (27 C) out.

Objective:

So we will **predict the Relative Humidity** of a given point of time based on the all other attributes affecting the change in RH.

Content:

1) Load data

2) Basic statistics

3) Data Cleaning

4) Co-relation between variables

5) Influence of features on output-RH

6) Baseline Linear Regression

6a) Conclusion of Baseline Linear Regression

7) Feature Engineering and testing model

7a) Conclusion of Feature Engineering and testing

8) Decision Tree Regression

9) Random Forest Regression

10) Support Vector Machine

11) Conclusion

In [1]:

```
#Import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.pylab import rcParams
import seaborn as sns
```

```
rcParams['figure.figsize']=10,8
```

1) Load data

In [3]:

linkcode

```
#define header
col=['DATE','TIME','CO_GT','PT08_S1_CO','NMHC_GT','C6H6_GT','PT08_S2_NMHC'
,
'NOX_GT','PT08_S3_NOX','NO2_GT','PT08_S4_NO2','PT08_S5_O3','T','RH','AH']


#define number of columns from csv
use=list(np.arange(len(col)))

#read the data from csv
df_air=pd.read_csv(local_path+'AirQualityUCI.csv',header=None,skiprows=1,n
ames=col,na_filter=True,
                    na_values=-200,usecols=use)
df_air.head()
```

Out[3]:

	DATE	TIME	CO_GT	PT08_S1_CO	NMHC_GT	C6H6_GT	PT08_S2_NMHC	NOX_GT	PT08_S3_NOX	NO2_GT	PT08_S4_NO2	PT08_S5_O3	T	RH	AH
0	3/10/2004	18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	113.0	1692.0	1268.0	13.6	48.9	0.7578
1	3/10/2004	19:00:00	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	92.0	1559.0	972.0	13.3	47.7	0.7255
2	3/10/2004	20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	114.0	1555.0	1074.0	11.9	54.0	0.7502
3	3/10/2004	21:00:00	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	122.0	1584.0	1203.0	11.0	60.0	0.7867
4	3/10/2004	22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	116.0	1490.0	1110.0	11	59	0.7888

													.2	.6	

In [4]:

```
#See the end records of dataframe
df_air.tail()
```

Out[4]:

	DATE	TIME	CO_GT	PT08_S1_CO	NMHC_GT	C6H6_GT	PT08_S2_NMHC	NOX_GT	PT08_S3_NOX	NO2_GT	PT08_S4_NO2	PT08_S5_O3	T	RH	AH
9466	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9467	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9468	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9469	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9470	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN						

```
df_air.dtypes
```

Out[5]:

```
DATE             object
TIME             object
CO_GT           float64
PT08_S1_CO      float64
NMHC_GT         float64
C6H6_GT         float64
PT08_S2_NMHC    float64
NOX_GT          float64
PT08_S3_NOX     float64
NO2_GT          float64
PT08_S4_NO2     float64
```

```
PT08_S5_O3         float64
T                  float64
RH                 float64
AH                 float64
dtype: object
```

In [6]:

```
#drop end rows with NaN values
df_air.dropna(how='all',inplace=True)
#drop RH NAN rows
df_air.dropna(thresh=10,axis=0,inplace=True)
```

In [7]:

```
df_air.shape
```

Out[7]:

```
(8991, 15)
```

2) Basic statistics

In [8]:

```
df_air.describe()
```

Out[8]:

	CO_GT	PT08_S1_CO	NMHC_GT	C6H6_GT	PT08_S2_NMHC	NOX_GT	PT08_S3_NOX	NO2_GT	PT08_S4_NO2	PT08_S5_O3	T	RH	AH
count	7344.000000	8991.000000	887.000000	8991.000000	8991.000000	7396.000000	8991.000000	7393.000000	8991.000000	8991.000000	8991.000000	8991.000000	8991.000000
mean	2.129711	1099.833166	218.607666	10.083105	939.153376	242.189292	835.493605	112.145137	1456.264598	1022.906128	18.317829	49.234201	1.025530
std	1.436472	217.080037	206.615130	7.449820	266.831429	206.312007	256.817320	47.629141	346.206794	398.484288	8.832116	17.316892	0.403813
min	0.100000	647.000000	7.000000	0.100000	383.000000	2.000000	322.000000	2.000000	551.000000	221.000000	-1.900000	9.200000	0.184700
25%	1.100000	937.000000	66.000000	4.400000	734.500000	97.000000	658.000000	77.000000	1227.000000	731.500000	11.800000	35.800000	0.736800

50%	1.800000	1063.000000	145.00000	8.200000	909.000000	178.00000	806.000000	109.00000	1463.000000	963.00000	17.800000	49.600000	0.995400
75%	2.800000	1231.000000	297.00000	14.000000	1116.000000	321.00000	969.500000	140.00000	1674.000000	1273.500000	24.400000	62.500000	1.313700
max	11.900000	2040.000000	1189.000000	63.700000	2214.000000	1479.000000	2683.000000	33					

3) Data Cleaning

In [9]:

```
#Split hour from time into new column
df_air['HOUR']=df_air['TIME'].apply(lambda x: int(x.split(':')[0]))
df_air.HOUR.head()
```

Out[9]:

```
0      18
1      19
2      20
3      21
4      22
Name: HOUR, dtype: int64
```

How many missing values now?

In [10]:

```
print('Count of missing values:\n',df_air.shape[0]-df_air.count())
```

```
Count of missing values:
 DATE                0
TIME                0
CO_GT            1647
PT08_S1_CO          0
NMHC_GT          8104
C6H6_GT             0
PT08_S2_NMHC        0
NOX_GT           1595
PT08_S3_NOX         0
NO2_GT           1598
PT08_S4_NO2         0
PT08_S5_O3          0
T                   0
RH                  0
AH                  0
```

```
HOUR              0
dtype: int64
```

Fill missing value strategy

-CO_GT, NOX_GT, NO2_GT will be filled by monthly average of that particular hour

-NHHC_GT will be dropped as it has 90% missing data

In [11]:

```
df_air['DATE']=pd.to_datetime(df_air.DATE, format='%m/%d/%Y')    #Format date column
```

In [12]:

```
# set the index as date
df_air.set_index('DATE',inplace=True)
```

In [13]:

```
df_air['MONTH']=df_air.index.month      #Create month column (Run once)
df_air.reset_index(inplace=True)
#df_air.head()
```

Drop column NMHC_GT; it has 90% missing data

In [14]:

```
df_air.drop('NMHC_GT',axis=1,inplace=True)     #drop col
```

Fill NaN values with monthly average of particular hour

In [15]:

```
df_air['CO_GT']=df_air['CO_GT'].fillna(df_air.groupby(['MONTH','HOUR'])['CO_GT'].transform('mean'))
df_air['NOX_GT']=df_air['NOX_GT'].fillna(df_air.groupby(['MONTH','HOUR'])['NOX_GT'].transform('mean'))
df_air['NO2_GT']=df_air['NO2_GT'].fillna(df_air.groupby(['MONTH','HOUR'])['NO2_GT'].transform('mean'))
```

In [16]:

```
print('Left out missing value:',df_air.shape[0]-df_air.count() )
```

```
Left out missing value: DATE              0
TIME              0
```

```
CO_GT              30
PT08_S1_CO          0
C6H6_GT             0
PT08_S2_NMHC        0
NOX_GT            261
PT08_S3_NOX         0
NO2_GT            261
PT08_S4_NO2         0
PT08_S5_O3          0
T                   0
RH                  0
AH                  0
HOUR                0
MONTH               0
dtype: int64
```

Fill left out NaaN values with hourly average value

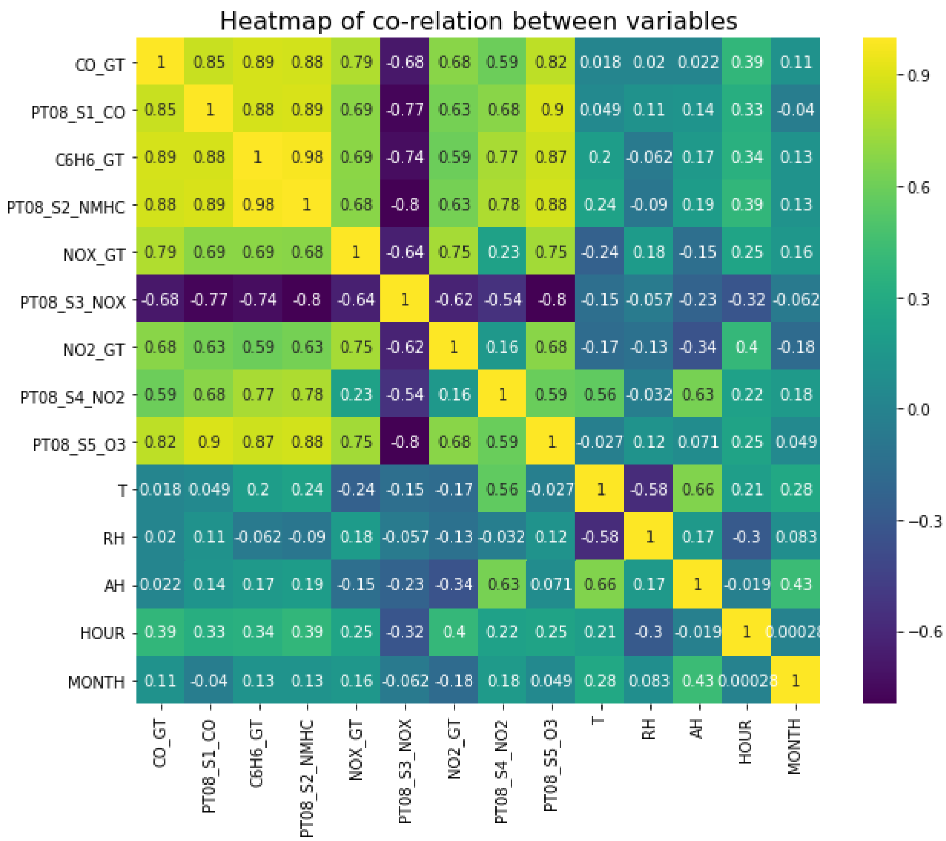
In [17]:

```
df_air['CO_GT']=df_air['CO_GT'].fillna(df_air.groupby(['HOUR'])['CO_GT'].t
ransform('mean'))
df_air['NOX_GT']=df_air['NOX_GT'].fillna(df_air.groupby(['HOUR'])['NOX_GT'
].transform('mean'))
df_air['NO2_GT']=df_air['NO2_GT'].fillna(df_air.groupby(['HOUR'])['NO2_GT'
].transform('mean'))
```

4) Understand co-relation between variables

In [18]:

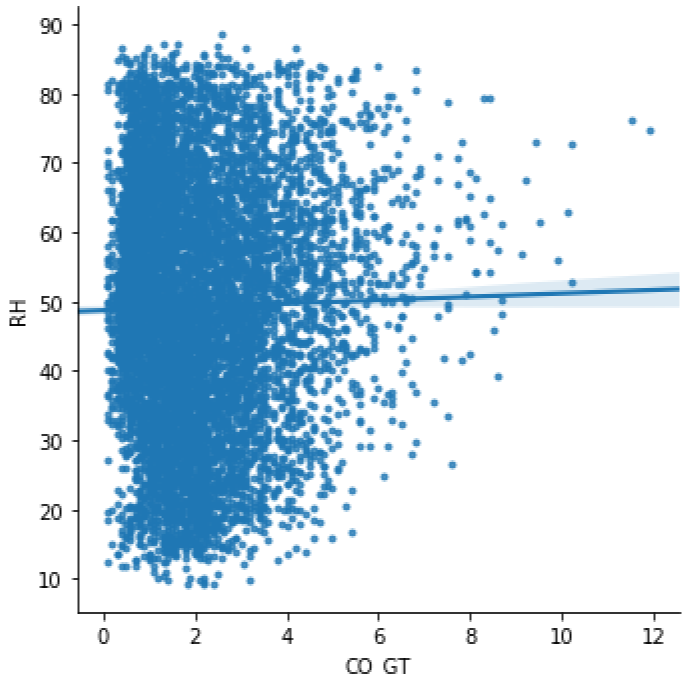
```
#Use heatmap to see corelation between variables
sns.heatmap(df_air.corr(),annot=True,cmap='viridis')
plt.title('Heatmap of co-relation between variables',fontsize=16)
plt.show()
```



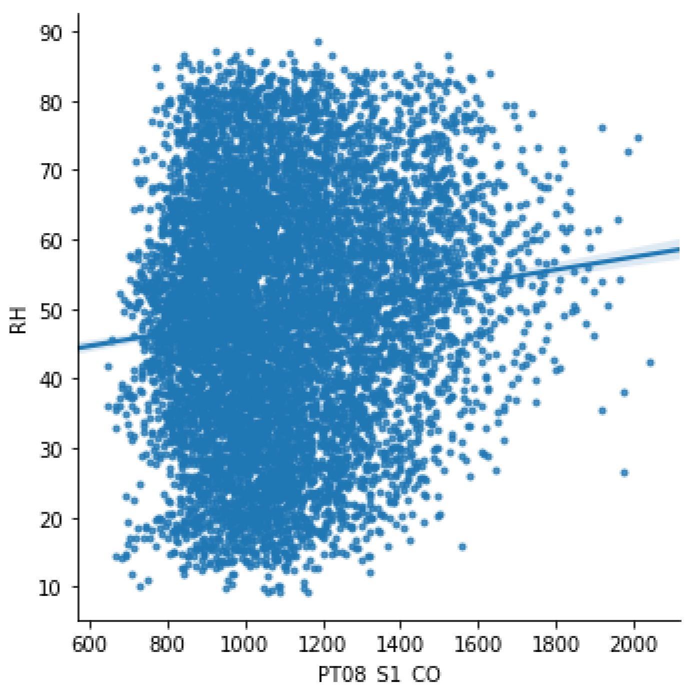
5) Try to understand degree of linearity between RH output and other input features

In [19]:

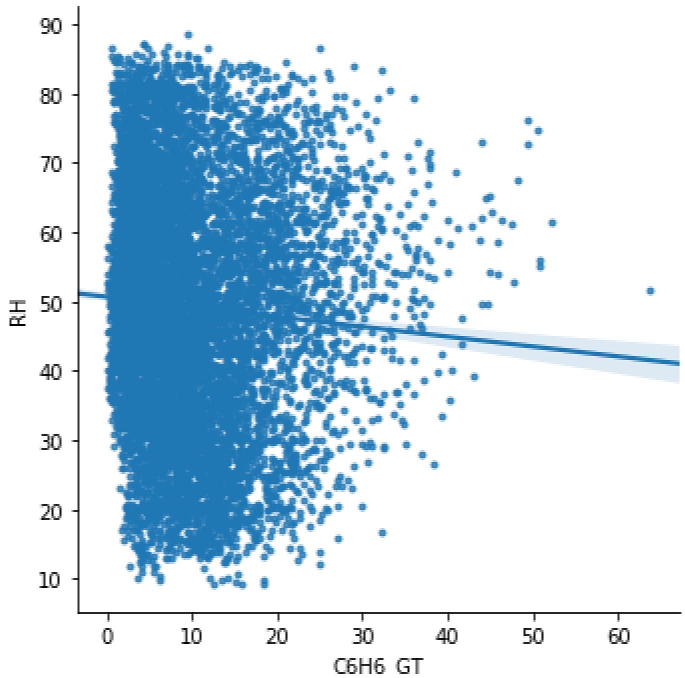
```
#plot all X-features against output variable RH
col_=df_air.columns.tolist()[2:]
for i in df_air.columns.tolist()[2:]:
    sns.lmplot(x=i,y='RH',data=df_air,markers='.')
```



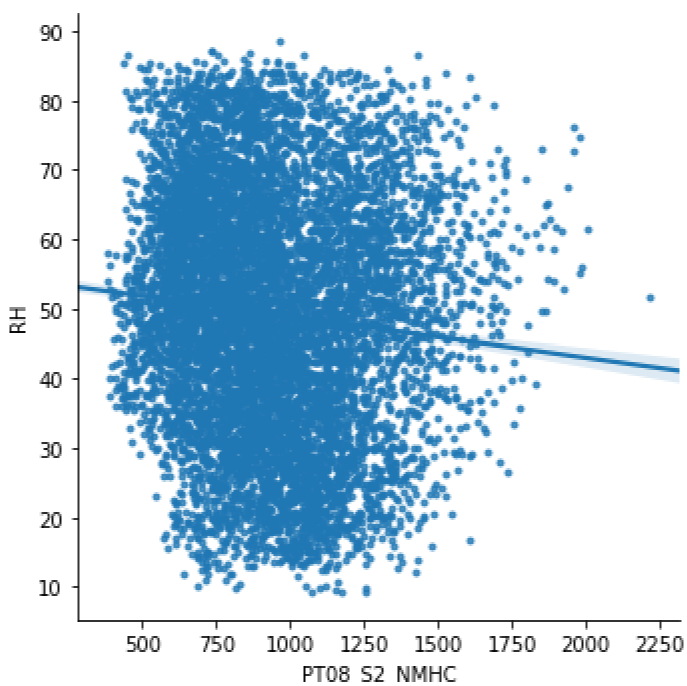
RH
CO_GT



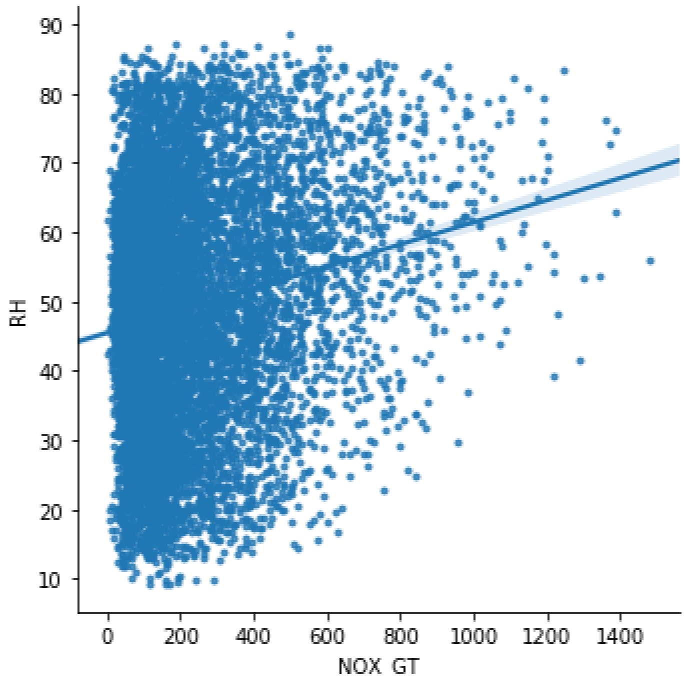
RH
PT08_S1_CO



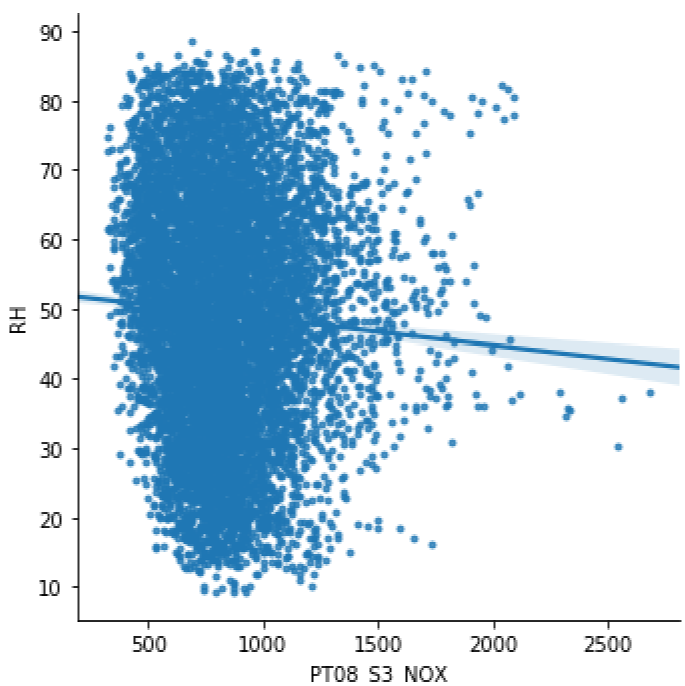
RH
90
80
70
60
50
40
30
20
10
0
10
20
30
40
50
60
C6H6_GT



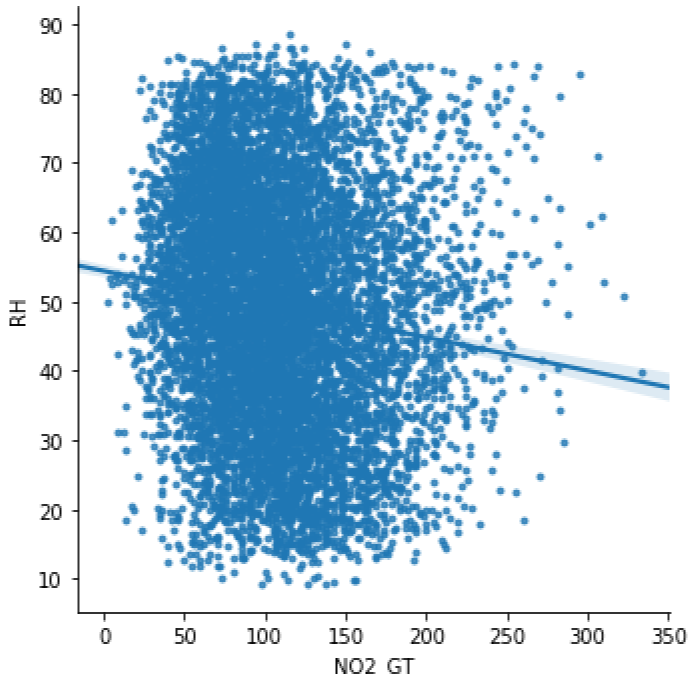
RH
90
80
70
60
50
40
30
20
10
500
750
1000
1250
1500
1750
2000
2250
PT08_S2_NMHC

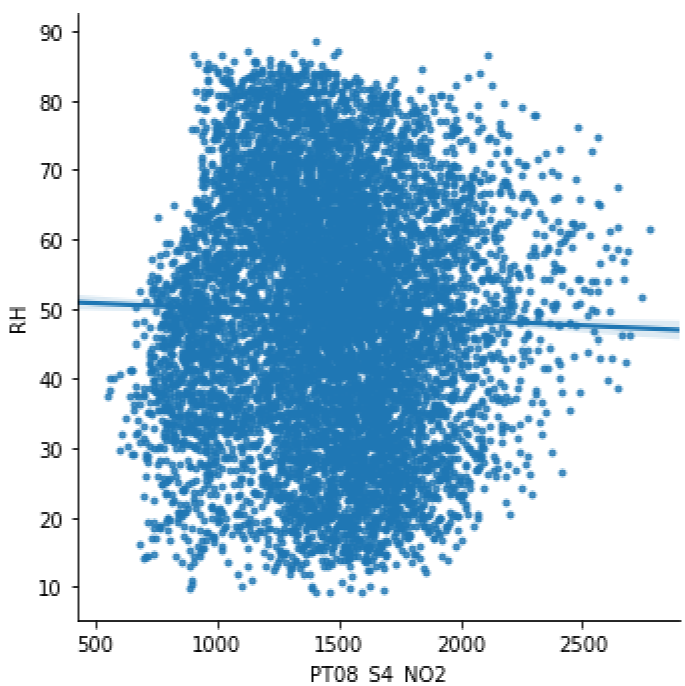


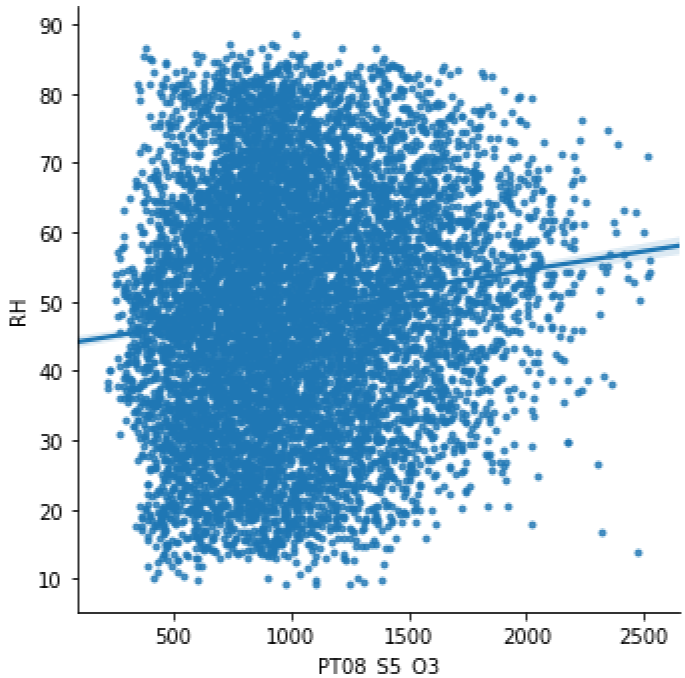
RH
NOX_GT

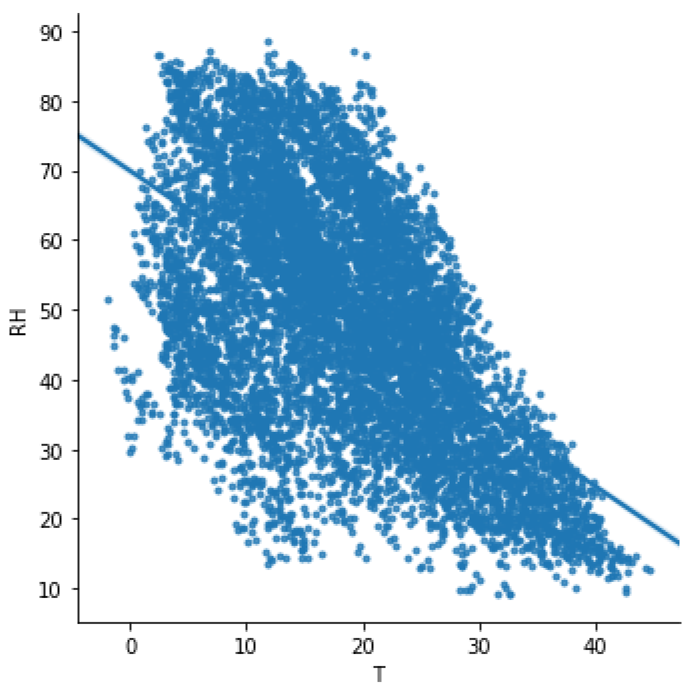


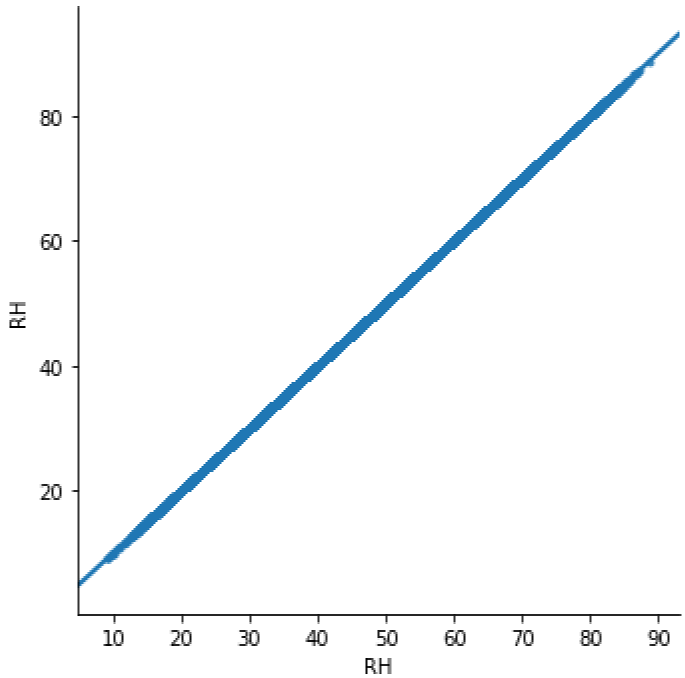
RH
PT08_S3_NOX


RH
NO2_GT
90
80
70
60
50
40
30
20
10
0
50
100
150
200
250
300
350

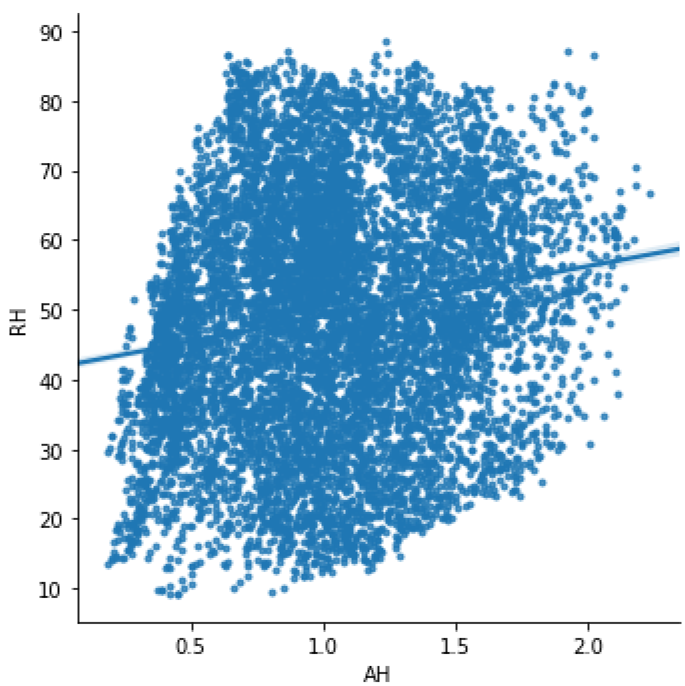

RH
PT08_S4_NO2
90
80
70
60
50
40
30
20
10
500
1000
1500
2000
2500


90
80
70
60
50
40
30
20
10
RH
500
1000
1500
2000
2500
PT08_S5_O3

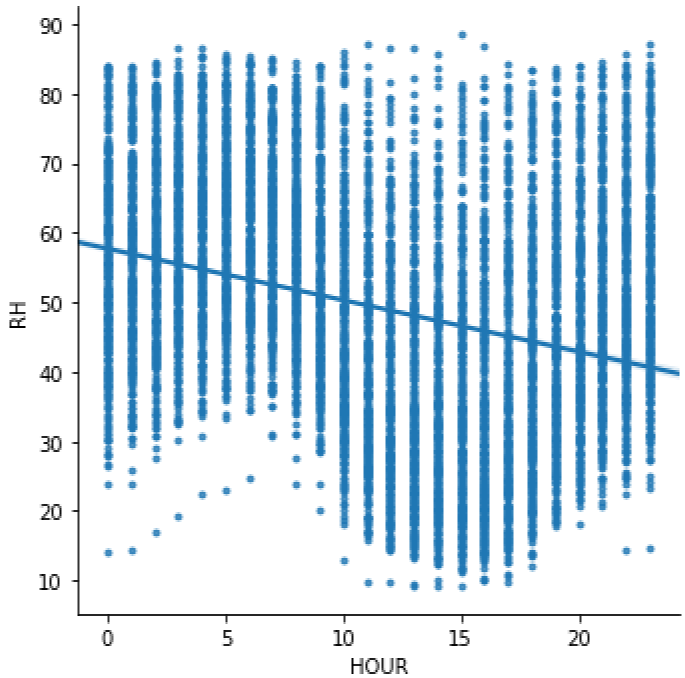

90
80
70
60
50
40
30
20
10
RH
0
10
20
30
40
T

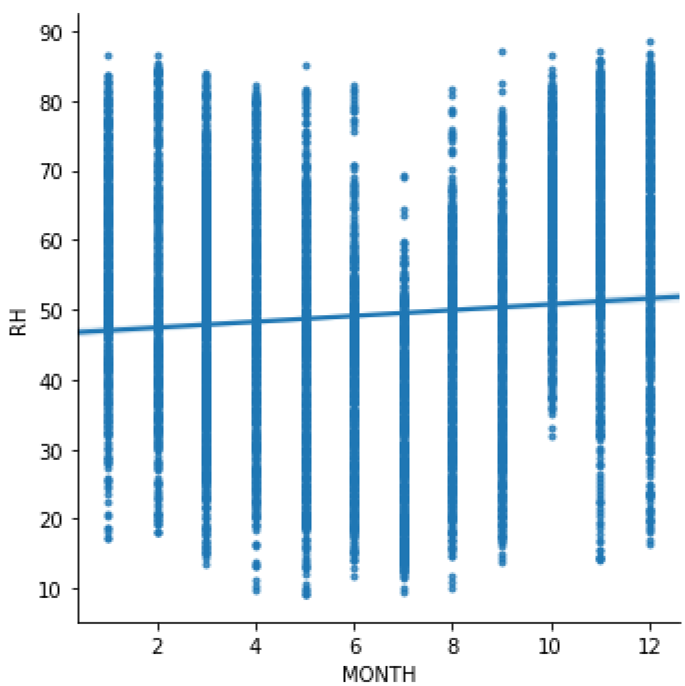


80
60
40
20
RH
10
20
30
40
50
60
70
80
90
RH



90
80
70
60
50
40
30
20
10
RH
0.5
1.0
1.5
2.0
AH





6) Linear Regression

In [20]:

```
from sklearn.preprocessing import StandardScaler          #import normalisation package
from sklearn.model_selection import train_test_split       #import train test split
from sklearn.linear_model import LinearRegression          #import linear
```

```
regression package
from sklearn.metrics import mean_squared_error,mean_absolute_error
#import mean squared error and mean absolute error
```

Define Feature (X) and Target (y)

In [21]:

```
X=df_air[col_].drop('RH',1)      #X-input features
y=df_air['RH']                   #y-input features
```

Normalize Feature variable

In [22]:

```
ss=StandardScaler()      #initiatilise
```

In [23]:

```
X_std=ss.fit_transform(X)      #apply stardardisation
```

Train test split

In [24]:

```
#split the data into train and test with test size and 30% and train size
as 70%
X_train, X_test, y_train, y_test=train_test_split(X_std,y,test_size=0.3,
random_state=42)
```

In [25]:

```
print('Training data size:',X_train.shape)
print('Test data size:',X_test.shape)
```

```
Training data size: (6293, 13)
Test data size: (2698, 13)
```

Train the model

In [26]:

```
lr=LinearRegression()
lr_model=lr.fit(X_train,y_train)          #fit the linear model on train
data
```

In [27]:

```
print('Intercept:',lr_model.intercept_)
print('--------------------------------')
print('Slope:')
list(zip(X.columns.tolist(),lr_model.coef_))
```

```
Intercept: 49.217630461
--------------------------------
Slope:
```

Out[27]:

```
[('CO_GT', -1.7367447259994688),
 ('PT08_S1_CO', 3.4037741865264262),
 ('C6H6_GT', -5.697492496373167),
 ('PT08_S2_NMHC', -1.1962342483257395),
 ('NOX_GT', 3.5036899671340738),
('PT08_S3_NOX', -0.70018468936766876),
 ('NO2_GT', -1.1080890551814175),
 ('PT08_S4_NO2', 6.8771350831151477),
 ('PT08_S5_O3', -1.2881546341603678),
 ('T', -20.184910618985896),
 ('AH', 12.063387650671071),
 ('HOUR', -0.61784140965068213),
 ('MONTH', 1.3399283374747475)]
```

Prediction

In [28]:

```
y_pred=lr_model.predict(X_test)                      #predict using the model
rmse=np.sqrt(mean_squared_error(y_test,y_pred))      #calculate rmse
print('Baseline RMSE of model:',rmse)
```

```
Baseline RMSE of model: 6.01289437122
```

6a) Conclusion of baseline linear regression model:

This means that we can predict RH using all the features together with **RMSE as 6.01**. Let us call it as baseline model.

7) Feature engineering and testing model:

Try with multiple feature combination and see if RMSE is improving

Build RMSE function

In [29]:

```
# write function to measure RMSE
def train_test_RMSE(feature):
    X=df_air[feature]
    y=df_air['RH']
    X_std_one=ss.fit_transform(X)

X_trainR,X_testR,y_trainR,y_testR=train_test_split(X_std_one,y,test_size=0
.3,random_state=42)
    lr_model_one=lr.fit(X_trainR,y_trainR)
    y_predR=lr_model_one.predict(X_testR)
    return np.sqrt(mean_squared_error(y_testR,y_predR))
```

In [30]:

```
col_.remove('RH')          #remove output
```

In [31]:

```
print('List of features:',col_)     #print list of features
```

```
List of features: ['CO_GT', 'PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC',
'NOX_GT', 'PT08_S3_NOX', 'NO2_GT', 'PT08_S4_NO2', 'PT08_S5_O3', 'T', 'AH',
'HOUR', 'MONTH']
```

In [32]:

```
print('RMSE with Features as',col_[0:2],train_test_RMSE(col_[0:2]))
print('--------------------------')
print('RMSE with Features as',col_[0:6],train_test_RMSE(col_[0:6]))
print('--------------------------')
print('RMSE with Features as',col_[0:9],train_test_RMSE(col_[0:9]))
print('--------------------------')
print('RMSE with Features as',col_[1:5],train_test_RMSE(col_[2:9]))
print('--------------------------')
print('RMSE with Features as',col_[0:11],train_test_RMSE(col_[0:11]))
print('--------------------------')
print('RMSE with Features as',col_[1:12],train_test_RMSE(col_[1:12]))
print('--------------------------')
print('RMSE with Features as',col_[0:13],train_test_RMSE(col_[0:13]))
```

```
RMSE with Features as ['CO_GT', 'PT08_S1_CO'] 17.1072232499
--------------------------
RMSE with Features as ['CO_GT', 'PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC',
'NOX_GT', 'PT08_S3_NOX'] 14.7879244799
```

```
-------------------------
RMSE with Features as ['CO_GT', 'PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC',
'NOX_GT', 'PT08_S3_NOX', 'NO2_GT', 'PT08_S4_NO2', 'PT08_S5_O3']
12.875243451
-------------------------
RMSE with Features as ['PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC', 'NOX_GT']
13.3641023495
-------------------------
RMSE with Features as ['CO_GT', 'PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC',
'NOX_GT', 'PT08_S3_NOX', 'NO2_GT', 'PT08_S4_NO2', 'PT08_S5_O3', 'T', 'AH']
6.09653798867
-------------------------
RMSE with Features as ['PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC', 'NOX_GT',
'PT08_S3_NOX', 'NO2_GT', 'PT08_S4_NO2', 'PT08_S5_O3', 'T', 'AH', 'HOUR']
6.07110993628
-------------------------
RMSE with Features as ['CO_GT', 'PT08_S1_CO', 'C6H6_GT', 'PT08_S2_NMHC',
'NOX_GT', 'PT08_S3_NOX', 'NO2_GT', 'PT08_S4_NO2', 'PT08_S5_O3', 'T', 'AH',
'HOUR', 'MONTH'] 6.01289437122
```

7a) Conclusion of Feature Engineering and testing:

After this experiment it looks that baseline model is performing best

8) Decision Tree Regression

Let us try to apply Decision tree regression technique and see if any improvement happens

In [33]:

```
from sklearn.tree import DecisionTreeRegressor          #Decision tree regression model
from sklearn.cross_validation import cross_val_score    #import cross validation score package
from sklearn.model_selection import GridSearchCV        #import grid search cv
dt_one_reg=DecisionTreeRegressor()
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
dt_model=dt_one_reg.fit(X_train,y_train)          #fit the model
y_pred_dtone=dt_model.predict(X_test)             #predict
```

RMSE of RH prediction

In [35]:

```
#calculate RMSE
print('RMSE of Decision Tree
Regression:',np.sqrt(mean_squared_error(y_pred_dtone,y_test)))
```

```
RMSE of Decision Tree Regression: 1.35369384553
```

Conclusion:(Decision Tree Regression)

When decision tree regression has been applied we observe significant improvement of **RMSE value to 1.36**

9) Random Forest Regression

Let us apply Random Forest regression and measure RMSE

In [36]:

```
from sklearn.ensemble import RandomForestRegressor            #import
random forest regressor
rf_reg=RandomForestRegressor()
```

Fit the RF model and predict

In [37]:

```
rf_model=rf_reg.fit(X_train,y_train)          #fit model
y_pred_rf=rf_model.predict(X_test)            #predict
```

RMSE of RH prediction

In [38]:

```
#Calculate RMSE
print('RMSE of predicted RH in RF
model:',np.sqrt(mean_squared_error(y_test,y_pred_rf)))
```

```
RMSE of predicted RH in RF model: 0.871016145245
```

linkcode

Lets try to improve on baseline RF model

```
#define rf parameters
rf_params={'n_estimators':[10,20],'max_depth':[8,10],'max_leaf_nodes':[70,
90]}
#define rf grid search
rf_grid=GridSearchCV(rf_reg,rf_params,cv=10)
```

In [40]:

```
rf_model_two=rf_grid.fit(X_train,y_train)      #fit the model wtih all grid
parameters
```

```
/opt/conda/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:718: DeprecationWarning: The
default of the `iid` parameter will change from True to False in version
0.22 and will be removed in 0.24. This will change numeric results when
test-set sizes are unequal.
  DeprecationWarning)
```

In [41]:

linkcode

```
y_pred_rf_two=rf_model_two.predict(X_test)         #predict
```

In [42]:

```
#Calculate RMSE
print('RMSE using RF grid search
method',np.sqrt(mean_squared_error(y_test,y_pred_rf_two)))
```

```
RMSE using RF grid search method 1.99486892421
```

Conclusion: Random Forest

Applying Random Forest regression the predicted **RMSE has improved to 0.86**, the default RF algorithm is giving better RMSE value than grid search applied different parameters.

10) Support Vector Machine

In [43]:

```
from sklearn.svm import SVR              #import support vector regressor
sv_reg=SVR()
```

In [44]:

```
sv_model=sv_reg.fit(X_train,y_train)     #train the model
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/svm/base.py:194: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

In [45]:

```
y_pred_sv=sv_model.predict(X_test)          #predict
```

In [46]:

```
#Calculate RMSE of SVR
print('RMSE of SVR model:',np.sqrt(mean_squared_error(y_test,y_pred_sv)))
```

```
RMSE of SVR model: 3.89916669053
```

linkcode

Conclusion:

For designing the model for predicting RH, I have applied Linear Regression, Decision Tree, Random Forest, Support Vector Machine. When tested on test data below are RMSE obtained from different algorithms: