In [1]:

```
#import all the necessary packages.

from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import math
import time
import re
import os
import seaborn as sns
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances
from matplotlib import gridspec
from scipy.sparse import hstack
import plotly
import plotly.figure_factory as ff
from plotly.graph_objs import Scatter, Layout

plotly.offline.init_notebook_mode(connected=True)
warnings.filterwarnings("ignore")
```

In [2]:

```
import os
os.chdir('C:/Users/kingsubham27091995/Desktop/AppliedAiCouse/CASE STUDIES/On the Plague trail')
```

In [3]:

```
train_data=pd.read_csv("train.csv")
test_data=pd.read_csv("test.csv")
sample_data=pd.read_csv("sample.csv")
```

In [4]:

```
print("Number of data points in train data:{0} and Number of features in train data:{1}".format(tr
ain_data.shape[0],train_data.shape[1]))
print("Number of data points in test data:{0} and Number of features in test data:{1}".format(test
_data.shape[0],test_data.shape[1]))
```

```
Number of data points in train data:40000 and Number of features in train data:37
Number of data points in test data:22446 and Number of features in test data:30
```

In [5]:

```
train_data.head(5)
```

Out[5]:

	ID	DateTime	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	WindTx	ISSRecp
0	PR00001	07-12-2040 0:15	53.5	53.6	53.5	85	49.1	2	SSE	0.5	...	1	100.0
1	PR00002	07-12-2040 0:30	53.5	53.5	53.4	85	49.1	2	SSE	0.5	...	1	100.0

	ID	DateTime	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	WindTx	ISSRecpt
2	PR00003	07-12-2040 0:45	53.3	53.5	53.2	85	48.9	2	SSE	0.5	...	1	100.0
3	PR00004	07-12-2040 1:00	53.1	53.3	53.0	86	49.0	2	S	0.5	...	1	100.0
4	PR00005	07-12-2040 1:15	52.9	53.1	52.9	86	48.8	2	S	0.5	...	1	100.0

5 rows × 37 columns

In [6]:

```
test_data.head(5)
```

Out[6]:

	ID	DateTime	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	InTemp	InHum
0	PR40001	08-04-2041 11:30	82.6	83.6	80.8	38	54.4	4	SSE	1.0	...	68.3	29
1	PR40002	08-04-2041 11:45	82.6	83.2	82.1	36	52.9	4	S	1.0	...	69.3	58
2	PR40003	08-04-2041 12:00	83.6	84.5	82.4	38	55.3	4	S	1.0	...	68.4	30
3	PR40004	08-04-2041 12:15	85.1	85.5	83.4	37	55.9	4	S	1.0	...	69.9	56
4	PR40005	08-04-2041 12:30	86.5	87.3	85.1	37	57.1	4	SSE	1.0	...	68.5	67

5 rows × 30 columns

Converting to python DateTime in Train_Data

In [7]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='DateTime' else x for x in list(train_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039

train_data['Date'] = pd.to_datetime(train_data['DateTime'])
train_data.drop('DateTime', axis=1, inplace=True)
#train_data.sort_values(by=['DateTime'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
train_data = train_data[cols]


train_data.head(2)
```

Out[7]:

	ID	Date	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	WindTx	ISSRecpt
0	PR00001	2040-07-12 00:15:00	53.5	53.6	53.5	85	49.1	2	SSE	0.5	...	1	100.0

	ID	Date	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	WindTx	ISSRecpt
1	PR00002	2040-07-12 00:30:00	53.5	53.5	53.4	85	49.1	2	SSE	0.5	...	1	100.0

2 rows × 37 columns

Converting to python DateTime in Test_Data

In [8]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='DateTime' else x for x in list(test_data.columns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039

test_data['Date'] = pd.to_datetime(test_data['DateTime'])
test_data.drop('DateTime', axis=1, inplace=True)
#train_data.sort_values(by=['DateTime'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
test_data = test_data[cols]


test_data.head(2)
```

Out[8]:

	ID	Date	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	InTemp	InHum	In
0	PR40001	2041-08-04 11:30:00	82.6	83.6	80.8	38	54.4	4	SSE	1.0	...	68.3	29	3
1	PR40002	2041-08-04 11:45:00	82.6	83.2	82.1	36	52.9	4	S	1.0	...	69.3	58	5

2 rows × 30 columns

Extracting Features from Train Data

In [9]:

```
train_data['Year'] = train_data['Date'].dt.year
train_data['Month'] = train_data['Date'].dt.month
train_data['Day'] = train_data['Date'].dt.day
```

Extracting Features from Test Data

In [10]:

```
test_data['Year'] = test_data['Date'].dt.year
test_data['Month'] = test_data['Date'].dt.month
test_data['Day'] = test_data['Date'].dt.day
```

In [11]:

```
train_data.columns
```

Out[11]:

```
Index(['ID', 'Date', 'TempOut', 'HiTemp', 'LowTemp', 'OutHum', 'DewPt',
       'WindSpeed', 'WindDir', 'WindRun', 'HiSpeed', 'HiDir', 'WindChill',
```

```
       'WindSpeed', 'WindDir', 'WindRun', 'HiSpeed', 'HiDir', 'WindChill',
       'HeatIndex', 'THWIndex', 'Bar', 'Rain', 'RainRate', 'HeatDD', 'CoolDD',
       'InTemp', 'InHum', 'InDew', 'InHeat', 'InEMC', 'InAirDensity',
       'WindSamp', 'WindTx', 'ISSRecpt', 'ArcInt', 'PA', 'PB', 'PC', 'PD',
       'PE', 'PF', 'PG', 'Year', 'Month', 'Day'],
      dtype='object')
```

In [12]:

```
test_data.columns
```

Out[12]:

```
Index(['ID', 'Date', 'TempOut', 'HiTemp', 'LowTemp', 'OutHum', 'DewPt',
       'WindSpeed', 'WindDir', 'WindRun', 'HiSpeed', 'HiDir', 'WindChill',
       'HeatIndex', 'THWIndex', 'Bar', 'Rain', 'RainRate', 'HeatDD', 'CoolDD',
       'InTemp', 'InHum', 'InDew', 'InHeat', 'InEMC', 'InAirDensity',
       'WindSamp', 'WindTx', 'ISSRecpt', 'ArcInt', 'Year', 'Month', 'Day'],
      dtype='object')
```

Vectorising Categorical Features

In [13]:

```
print("Number of categories in HiDir in train data:",train_data['HiDir'].value_counts().count())
print("Number of categories in HiDir in test data:",test_data['HiDir'].value_counts().count())
```

```
Number of categories in HiDir in train data: 17
Number of categories in HiDir in test data: 17
```

In [14]:

```
print("Number of categories in WindDir in test data:",train_data['WindDir'].value_counts().count()
)
print("Number of categories in WindDir in test data:",test_data['WindDir'].value_counts().count())
```

```
Number of categories in WindDir in test data: 17
Number of categories in WindDir in test data: 17
```

REFERENCE:: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.categories.html

In [15]:

```
categorical_cols = ['HiDir','WindDir']
for i in categorical_cols:
    if i == 'HiDir':
        train_data[i] = train_data[i].astype('category')
        train_data[i].cat.categories = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
        train_data[i] = train_data[i].astype('int')

        test_data[i] = test_data[i].astype('category')
        test_data[i].cat.categories = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
        test_data[i] = test_data[i].astype('int')
    else:
        train_data[i] = train_data[i].astype('category')
        train_data[i].cat.categories = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
        train_data[i] = train_data[i].astype('int')

        test_data[i] = test_data[i].astype('category')
        test_data[i].cat.categories = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
        test_data[i] = test_data[i].astype('int')
```

Vectorising Numerical Features

In [16]:

```
from sklearn.preprocessing import StandardScaler
import warnings
```

```
import warnings
warnings.filterwarnings("ignore")

numerical_cols = [ 'TempOut', 'HiTemp', 'LowTemp', 'OutHum', 'DewPt',
       'WindSpeed', 'WindDir', 'WindRun', 'HiSpeed', 'HiDir', 'WindChill',
       'HeatIndex', 'THWIndex', 'Bar', 'Rain', 'RainRate', 'HeatDD', 'CoolDD',
       'InTemp', 'InHum', 'InDew', 'InHeat', 'InEMC', 'InAirDensity',
       'WindSamp', 'WindTx', 'ISSRecpt', 'ArcInt', 'Year' , 'Month', 'Day']

vectorizer = StandardScaler()
vectorizer.fit(train_data[numerical_cols])
train_data[numerical_cols] = vectorizer.transform(train_data[numerical_cols])
test_data[numerical_cols] = vectorizer.transform(test_data[numerical_cols])
```

In [17]:

```
print(f"Mean : {vectorizer.mean_}, \n\n Standard deviation : {np.sqrt(vectorizer.var_)}")
```

```
Mean :  [5.85086250e+01 5.89752300e+01 5.80567850e+01 7.29157500e+01
 4.81568725e+01 2.34865000e+00 9.90457500e+00 5.87162500e-01
 6.02867500e+00 9.87182500e+00 5.83733350e+01 5.81392025e+01
 5.80039500e+01 3.00719467e+01 5.05500000e-04 3.94950000e-03
 9.44551000e-02 2.68369500e-02 6.91713450e+01 4.72592500e+01
 4.71814950e+01 6.74065500e+01 9.04387225e+00 7.45691275e-02
 3.51205575e+02 1.00000000e+00 9.99979375e+01 1.50000000e+01
 2.04059863e+03 6.45807500e+00 1.55859250e+01],

 Standard deviation : [1.21194884e+01 1.23232730e+01 1.19161858e+01 2.08732209e+01
 7.89567247e+00 2.34633612e+00 5.00519920e+00 5.86584029e-01
 4.80819121e+00 5.08932178e+00 1.21668479e+01 1.18584751e+01
 1.19121542e+01 1.45419917e-01 4.23373000e-03 5.80011332e-02
 8.44495860e-02 6.11230907e-02 2.03694143e+00 1.38890547e+01
 8.36358727e+00 2.68500765e+00 2.41533607e+00 6.44136739e-04
 6.97792175e-01 0.00000000e+00 1.06522280e-01 0.00000000e+00
 5.66280063e-01 3.28615159e+00 8.96099977e+00]
```

In [18]:

```
train_data.head(5)
```

Out[18]:

	ID	Date	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	PA	PB	PC	P
0	PR00001	2040-07-12 00:15:00	-0.413270	-0.436185	-0.382403	0.578936	0.119449	-0.148593	0.418650	-0.148593	...	1	1	1	1
1	PR00002	2040-07-12 00:30:00	-0.413270	-0.444300	-0.390795	0.578936	0.119449	-0.148593	0.418650	-0.148593	...	1	1	1	1
2	PR00003	2040-07-12 00:45:00	-0.429773	-0.444300	-0.407579	0.578936	0.094118	-0.148593	0.418650	-0.148593	...	1	1	1	1
3	PR00004	2040-07-12 01:00:00	-0.446275	-0.460529	-0.424363	0.626844	0.106783	-0.148593	0.019065	-0.148593	...	1	1	1	1
4	PR00005	2040-07-12 01:15:00	-0.462777	-0.476759	-0.432755	0.626844	0.081453	-0.148593	0.019065	-0.148593	...	1	1	1	1

5 rows × 40 columns

In [19]:

```
test_data.head(5)
```

Out[19]:

	ID	Date	TempOut	HiTemp	LowTemp	OutHum	DewPt	WindSpeed	WindDir	WindRun	...	InHeat	In
0	PR40001	2041-08-04 11:30:00	1.987821	1.998233	1.908599	-1.672753	0.790702	0.703799	0.418650	0.703799	...	-1.045267	-1.22
1	PR40002	2041-08-04 11:45:00	1.987821	1.965774	2.017694	-1.768570	0.600725	0.703799	0.019065	0.703799	...	0.407243	0.70
2	PR40003	2041-08-04 12:00:00	2.070333	2.071265	2.042870	-1.672753	0.904689	0.703799	0.019065	0.703799	...	-0.970779	-1.15
3	PR40004	2041-08-04 12:15:00	2.194100	2.152413	2.126789	-1.720662	0.980680	0.703799	0.019065	0.703799	...	0.481730	0.54
4	PR40005	2041-08-04 12:30:00	2.309617	2.298478	2.269452	-1.720662	1.132662	0.703799	0.418650	0.703799	...	0.481730	1.38

5 rows × 33 columns

In [20]:

```
output_columns = ['PA','PB','PC','PD','PE','PF','PG']
```

In [21]:

```
X_tr = train_data.drop(['ID','Date','PA','PB','PC','PD','PE','PF','PG'], axis=1)
X_te = test_data.drop(['ID','Date'],axis=1)
```

In [22]:

```
y_train=train_data[['PA', 'PB', 'PC', 'PD','PE', 'PF', 'PG']]
```

Random Forest Regressor

In [23]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV



# initialize Our first RandomForestRegressor model...
regr2 = RandomForestRegressor(max_features='sqrt')

# declare parameters for hyperparameter tuning
parameters = {'n_estimators':[100,300,500,700],'max_depth':[1,3,5]}

# Perform cross validation
clf = GridSearchCV(regr2,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = 5,
                    n_jobs = -1,
                    verbose = 1)
result = clf.fit(X_tr, y_train)

# Summarize results
print("Best: %f using %s" % (result.best_score_, result.best_params_))
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f 1(%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:  5.9min finished
```

```
Best: -107549.158046 using {'max_depth': 5, 'n_estimators': 300}
-122353.501228 1(169713.175156) with: {'max_depth': 1, 'n_estimators': 100}
-120605.232828 1(171362.513125) with: {'max_depth': 1, 'n_estimators': 300}
-120924.514418 1(171885.192974) with: {'max_depth': 1, 'n_estimators': 500}
-120321.778137 1(171237.822516) with: {'max_depth': 1, 'n_estimators': 700}
-120567.124478 1(164975.705974) with: {'max_depth': 3, 'n_estimators': 100}
-114155.273419 1(162288.306190) with: {'max_depth': 3, 'n_estimators': 300}
-118477.744125 1(162421.344207) with: {'max_depth': 3, 'n_estimators': 500}
-114541.376423 1(160526.765732) with: {'max_depth': 3, 'n_estimators': 700}
-109552.673827 1(152012.168329) with: {'max_depth': 5, 'n_estimators': 100}
-107549.158046 1(151557.735205) with: {'max_depth': 5, 'n_estimators': 300}
-107601.697910 1(153161.886580) with: {'max_depth': 5, 'n_estimators': 500}
-109110.328331 1(152789.400688) with: {'max_depth': 5, 'n_estimators': 700}
```

In [24]:

```
rfr = RandomForestRegressor(max_depth=5,n_estimators=300)
```

In [25]:

```
submission = pd.DataFrame()
submission['ID'] = test_data['ID']
```

In [26]:

```
for i in output_columns:
    y_train = train_data[i]
    rfr.fit(X_tr,y_train)
    predict_target = rfr.predict(X_te)
    print(rfr.score(X_te, predict_target),rfr.score(X_tr, y_train))
    submission[i] = [ round(p,0) for p in predict_target]
submission.to_csv('sample1.csv', header=True, index=False)
```

```
1.0 0.9639204954422342
1.0 0.9647900840078213
1.0 0.9656580768834093
1.0 0.9664233118813518
1.0 0.9669051772589868
1.0 0.9675461486681091
1.0 0.9681162584097579
```

This gave a leaderboard score of 86.7

XGBoostRegressor

In [27]:

```
import xgboost as xgb

# initialize Our first XGBoost model...
regr = xgb.XGBRegressor(silent=False, random_state=15)
#regr = MultiOutputRegressor(regr1)

# declare parameters for hyperparameter tuning
parameters = {'learning_rate':[0.001,0.01,0.1],'n_estimators':[100,300,500,700],'max_depth':[1,2,3]
}

# Perform cross validation
clf = GridSearchCV(regr,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
```

```
                    cv=5,
                    n_jobs = -1,
                    verbose = 1)
result = clf.fit(X_tr, y_train)

# Summarize results
print("Best: %f using %s" % (result.best_score_, result.best_params_))
means = result.cv_results_['mean_test_score']
stds = result.cv_results_['std_test_score']
params = result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f 1(%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 33.0min finished
```

```
Best: -737.781344 using {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}
-2008.549506 1(3521.827224) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}
-1790.156092 1(3274.549665) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}
-1673.605009 1(3057.394891) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}
-1628.042324 1(2878.543656) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}
-1988.482668 1(3416.923017) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}
-1710.614365 1(3015.248275) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}
-1491.255979 1(2682.983204) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}
-1331.677702 1(2400.961159) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}
-1951.924804 1(3350.079832) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}
-1625.386836 1(2882.926103) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}
-1400.456057 1(2527.122677) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}
-1241.225812 1(2247.589448) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}
-1622.416599 1(2659.121678) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}
-1540.481768 1(2085.169947) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}
-1405.827482 1(1889.120791) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}
-1295.626693 1(1778.502181) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}
-1173.034860 1(2067.127758) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}
-898.102552 1(1340.032176) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}
-846.381472 1(1272.638400) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}
-979.986170 1(1230.711407) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}
-1081.067518 1(1929.338207) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
-782.960688 1(1179.142744) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
-737.781344 1(1096.686832) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}
-741.788135 1(1068.832902) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}
-1182.204072 1(1683.283173) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}
-961.057362 1(1350.359424) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}
-912.277675 1(1235.155530) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}
-957.475139 1(1200.102368) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}
-1023.692678 1(1221.200679) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
-1238.825512 1(1383.991782) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}
-1298.032384 1(1442.507382) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}
-1338.145037 1(1490.287880) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}
-741.368162 1(1052.650374) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
-781.264575 1(1021.917543) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}
-790.669323 1(1019.730809) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
-796.272487 1(1014.840983) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}
```

In [28]:

```
xgb1 = xgb.XGBRegressor(learning_rate= 0.01, max_depth= 3, n_estimators= 500, nthread=-1)
xgb1
```

Out[28]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, importance_type='gain',
       learning_rate=0.01, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=500, n_jobs=1,
       nthread=-1, objective='reg:linear', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

In [29]:

```
for i in output_columns:
    y_train = train_data[i]
    xgb1.fit(X_tr,y_train)
    predict_target = xgb1.predict(X_te)
    print(xgb1.score(X_te, predict_target),xgb1.score(X_tr, y_train))
    submission[i] = [ round(p,0) for p in predict_target]
submission.to_csv('sample2.csv', header=True, index=False)
```

```
1.0 0.9657776449378069
1.0 0.9665721300210486
1.0 0.9672145334728744
1.0 0.9673735586552568
1.0 0.9680925566132271
1.0 0.9685889127102202
1.0 0.9689896775851773
```

This gave a leaderboard score of 88.19

In [30]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names=["Model Name","LeaderBoard Score(max(0,(100-rmse)))"]
ptable.add_row(["RandomForestRegressor","86.7"])
ptable.add_row(["XGBoostRegressor","88.19"])

print(ptable)
print()
```

```
+-----------------------+--------------------------------------+
|       Model Name      | LeaderBoard Score(max(0,(100-rmse))) |
+-----------------------+--------------------------------------+
| RandomForestRegressor |                 86.7                 |
|    XGBoostRegressor   |                88.19                 |
+-----------------------+--------------------------------------+
```