In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Import train and test data

In [3]:

```
train = pd.read_csv('train.csv', index_col = 'ID')
test = pd.read_csv('test.csv', index_col = 'ID')
```

In [4]:

```
train.head()
```

Out[4]:

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	130.81	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	0	0	0	0
6	88.53	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	0	0	0	0
7	76.26	az	w	n	c	d	x	j	x	0	...	0	0	0	0	0	0	1	0	0	0
9	80.62	az	t	n	f	d	x	l	e	0	...	0	0	0	0	0	0	0	0	0	0
13	78.02	az	v	n	f	d	h	d	n	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

Replacing strings with numbers in train and test dataframes. Note that a combined list of all unique strings is prepared for each feature (containing string) for both train and test data before replacing it with numbers. This is done to ensure that each strings gets mapped to same number for both train and test data.

In [5]:

```
for col in train.columns:
    if(train[col].dtype != np.float64 and train[col].dtype != np.int64):

        # making a list of unique strings in train and test feature
        unique_train = train[col].unique().tolist()
        unique_test = test[col].unique().tolist()

        # making a combined list
        for member in unique_test:
            if member not in unique_train:
                unique_train.append(member)

        # mapping with numbers
        map_dict = dict(zip(unique_train, range(len(unique_train))))
        train[col] = train[col].replace(to_replace = map_dict)
        test[col] = test[col].replace(to_replace = map_dict)
```

In [6]:

```
train.head()
```

Out[6]:

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	130.81	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
6	88.53	0	1	1	1	0	1	1	0	0	...	1	0	0	0	0	0	0	0	0	0
7	76.26	1	2	2	2	0	2	0	1	0	...	0	0	0	0	0	0	1	0	0	0
9	80.62	1	1	2	3	0	2	1	2	0	...	0	0	0	0	0	0	0	0	0	0
13	78.02	1	0	2	3	0	3	2	3	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

Checking if train or test data has any NaN value. Also, getting summary of data

In [7]:

```
print(train.isnull().values.any())
print(test.isnull().values.any())
train.describe()
```

```
False
False
```

Out[7]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	4
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	
mean	100.669318	12.110715	6.467569	7.851509	2.415301	0.002138	16.839629	3.031124	11.457591	0.013305	...	0.318841	0.057258	0.314802	0.020670	0.009503	0.008078	0.007603	
std	12.679381	8.315637	4.789927	5.644031	1.361654	0.073900	6.357474	2.554581	7.040194	0.114590	...	0.466082	0.232363	0.464492	0.142294	0.097033	0.089524	0.086872	
min	72.110000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	90.820000	6.000000	3.000000	4.000000	2.000000	0.000000	11.000000	1.000000	5.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	99.150000	11.000000	6.000000	7.000000	2.000000	0.000000	17.000000	2.000000	12.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	109.010000	15.000000	7.000000	10.000000	3.000000	0.000000	22.000000	6.000000	17.000000	0.000000	...	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	
max	265.320000	46.000000	26.000000	43.000000	6.000000	3.000000	28.000000	11.000000	24.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 377 columns

Splitting features and labels. Also, performing min_max scalling on features

In [8]:

```
X_train = train.iloc[:,1:]
y_train = train.iloc[:,0]

from sklearn.preprocessing import MinMaxScaler
scalling = MinMaxScaler().fit(X_train)
X_train_scalled = scalling.transform(X_train)
test_scalled = scalling.transform(test)
```

Regression with linear model. Cross-validation score shows that the linear model performs very poorly.

In [9]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

reg = LinearRegression()
print(cross_val_score(reg, X_train_scalled, y_train, cv=10))
```

```
[ -1.07634404e+20  -8.80253819e+18  -3.87843071e+20  -3.25262026e+18
  -1.83210847e+17  -4.14602806e+18  -6.87249190e+19  -2.38388418e+19
  -1.27734972e+20  -1.44969182e+21]
```

Regression with Lasso model (L1 regularization). As the number of features are very large, Lasso regularization would assign lesser weights to non-important features and in-turn reduce their contributation in the final regression model.

In [10]:

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

grid_values = {'alpha': [0.0235, 0.024, 0.0245]}
grid_lasso = GridSearchCV(Lasso(), param_grid = grid_values, cv=10, scoring = 'r2')
grid_lasso.fit(X_train_scalled, y_train)
predict_lasso = grid_lasso.predict(test_scalled)

print('Mean score matrix: ', grid_lasso.cv_results_['mean_test_score'])
print('Grid best parameter (max. accuracy): ', grid_lasso.best_params_)
print('Grid best score (accuracy): ', grid_lasso.best_score_)
```

```
Mean score matrix:  [ 0.56297736  0.56298228  0.56297693]
Grid best parameter (max. accuracy):  {'alpha': 0.024}
Grid best score (accuracy):  0.562982281071
```

Let's also try Ridge regression (L2 regularization)

In [11]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

grid_values = {'alpha': [40, 40.5, 41]}
grid_ridge = GridSearchCV(Ridge(), param_grid = grid_values, cv=10, scoring = 'r2')
grid_ridge.fit(X_train_scalled, y_train)
predict_ridge = grid_ridge.predict(test_scalled)

print('Mean score matrix: ', grid_ridge.cv_results_['mean_test_score'])
print('Grid best parameter (max. accuracy): ', grid_ridge.best_params_)
print('Grid best score (accuracy): ', grid_ridge.best_score_)
```

```
Mean score matrix:  [ 0.55376436  0.55376492  0.55376475]
Grid best parameter (max. accuracy):  {'alpha': 40.5}
Grid best score (accuracy):  0.553764919528
```

Regression with Xgboost. It shows the best R2 score. We will use this model to do final predictions.

In [22]:

```
import xgboost as xgb

grid_values = {'n_estimators': [74,75,76], 'learning_rate': [0.13,0.135,0.14], 'max_depth': [1,2,3]}
grid_xgb = GridSearchCV(xgb.XGBRegressor(), param_grid = grid_values, cv=10, scoring = 'r2')
grid_xgb.fit(X_train_scalled, y_train)
predict_xgb = grid_xgb.predict(test_scalled)

print('Mean score matrix: ', grid_xgb.cv_results_['mean_test_score'])
print('Grid best parameter (max. accuracy): ', grid_xgb.best_params_)
print('Grid best score (accuracy): ', grid_xgb.best_score_)
```

```
Mean score matrix:  [ 0.55597401  0.55620147  0.55675942  0.58165304  0.5818088   0.58176478
  0.57938243  0.57941404  0.57938819  0.55721329  0.55745941  0.55774666
  0.58204994  0.58214103  0.58202328  0.57727201  0.57720783  0.57713946
  0.55807455  0.55816391  0.55836724  0.58153668  0.58149164  0.58166668
  0.57805699  0.57808365  0.57817218]
Grid best parameter (max. accuracy):  {'learning_rate': 0.135, 'max_depth': 2, 'n_estimators': 75}
Grid best score (accuracy):  0.582141029072
```

In [ ]:

```
final_predictions = pd.DataFrame()
final_predictions['id'] = test.index
final_predictions['y'] = pd.Series(predict_xgb)
final_predictions.to_csv('predictions.csv', index=False)
```

In [ ]: