```
import pandas as pd
import math as m
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import statsmodels.api as sm


from sklearn.cross_validation import KFold
from sklearn import cross_validation
from sklearn.preprocessing import Imputer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge



df_train_rdkit = pd.read_csv("train_rdkit.csv")
df_train_SM = pd.read_csv("df_train_withSMFeatures.csv")
# df_train = pd.read_csv("train.csv")
# dfs = [df_train_rdkit,df_train_SM,df_train]
# df_train_all = reduce(lambda left,right: pd.merge(left,right,on='smiles'),
df_train_all = pd.merge(df_train_SM,df_train_rdkit,on="smiles")


############ Data preprocessing ############

## Clean the data a little ##
df_train_X = df_train_all.drop(['smiles'],axis=1) ##Use all the columns exce
df_train_X=df_train_X.dropna(axis=1,how='all') ## Drop all the columns that
df_train_X =df_train_X.loc[:, (df_train_X != 0).any(axis=0)] ## Drop the all

for column in df_train_X:
    df_train_X[column].replace([np.inf, -np.inf], 0)  ## Replace all the inf

## Remove NaNs and center them at the mean
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(df_train_X)
df_train_X = imp.transform(df_train_X)

## Create the Ys ##
df_train_Y =  pd.read_csv("train.csv",usecols =['gap']).values

```

```

print "Shape of X train is ",df_train_X.shape
print "Shape of Y train is ", df_train_Y.shape



degree = 1 ## Set the degree you want your basis function
### Sinsusoidal basis transformation
def makeMatrixsinusoid(yourMatrix,deg):
    f= np.vectorize((lambda x: m.sin(x/deg)))
    holder = yourMatrix
    for i in xrange(1,deg+1):
        holder= np.append(holder,f(yourMatrix),1)
    return holder


# def makeMatrixsinusoid(yourMatrix,deg,numtimes):
#   holder = yourMatrix
#   for i in xrange(1,numtimes+1):
#       for j in xrange(yourMatrix.shape[1]):
#           print holder.shape
#           holder = np.append(holder,makeArraysinusoid(yourMatrix[:,j],i).r
#       return holder


def runLR(X_tr,y_tr,X_ts,y_ts):
    LR = LinearRegression()
    LR.fit(X_train, y_train)

    RMSE_LR = m.sqrt(np.mean((LR.predict(X_test)-y_test)**2))
    print "RMSE for Baseline Linear Regression is %s"%RMSE_LR


## Use PCA for dimensionality reduction
def PCAreduction(matrix,dims):

    pca = PCA(n_components=dims, whiten=True).fit(matrix)
    print "Explaing variance for each PC is %s"%pca.explained_variance_ratio
    print "Total variance explained is %s"%pca.explained_variance_ratio_.sum
    return pca.transform(matrix)

def main():
    df_train_X_sine = makeArraysinusoid(df_train_X,2)
    # Create simple train/validate set from the training set
    X_train, X_test, y_train, y_test = cross_validation.train_test_split(df_

```

```python




if __name__ == "__main__":
    # execute only if run as a script
    main()


```