Data-X Spring 2018: Homework 02

Regression, Classification, Webscraping

Authors: Sana Iqbal (Part 1, 2, 3), Alexander Fred-Ojala (Extra Credit)

In this homework, you will do some exercises with prediction-classification, regression and web-scraping.

Part 1

Data:

Data Source: Data file is uploaded to bCourses and is named: **Energy.csv**

The dataset was created by Angeliki Xifara (Civil/Structural Engineer) and was processed by Athanasios Tsanas, Oxford Centre for Industrial and Applied Mathematics, University of Oxford, UK).

Data Description:

The dataset contains eight attributes of a building (or features, denoted by X1...X8) and response being the heating load on the building, y1.

- X1 Relative Compactness
- X2 Surface Area
- X3 Wall Area
- X4 Roof Area
- X5 Overall Height
- X6 Orientation
- X7 Glazing Area
- X8 Glazing Area Distribution
- y1 Heating Load

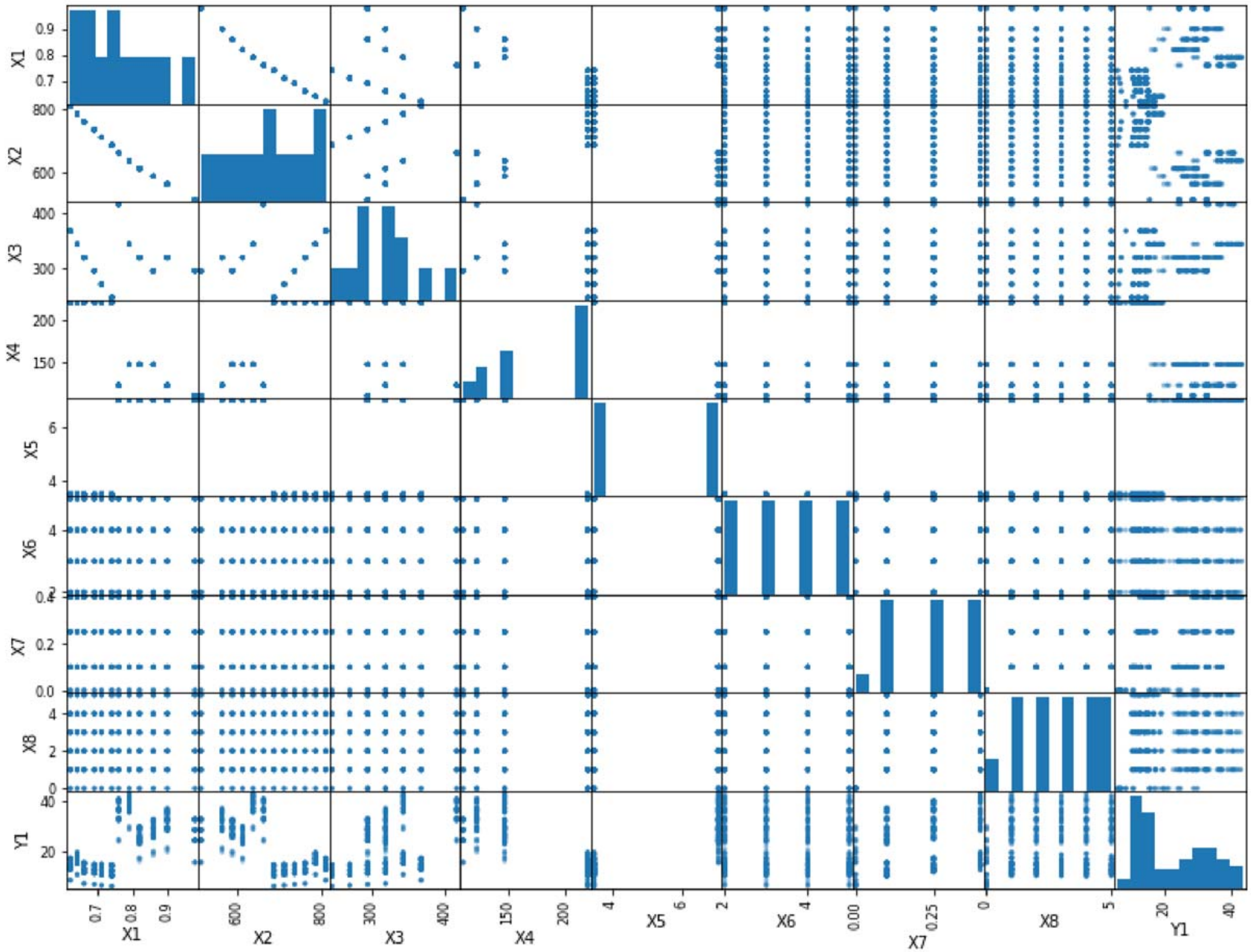
Q1:Read the data file in python. Describe data features in terms of type, distribution range and mean values. Plot feature

distributions.This step should give you clues about data sufficiency.

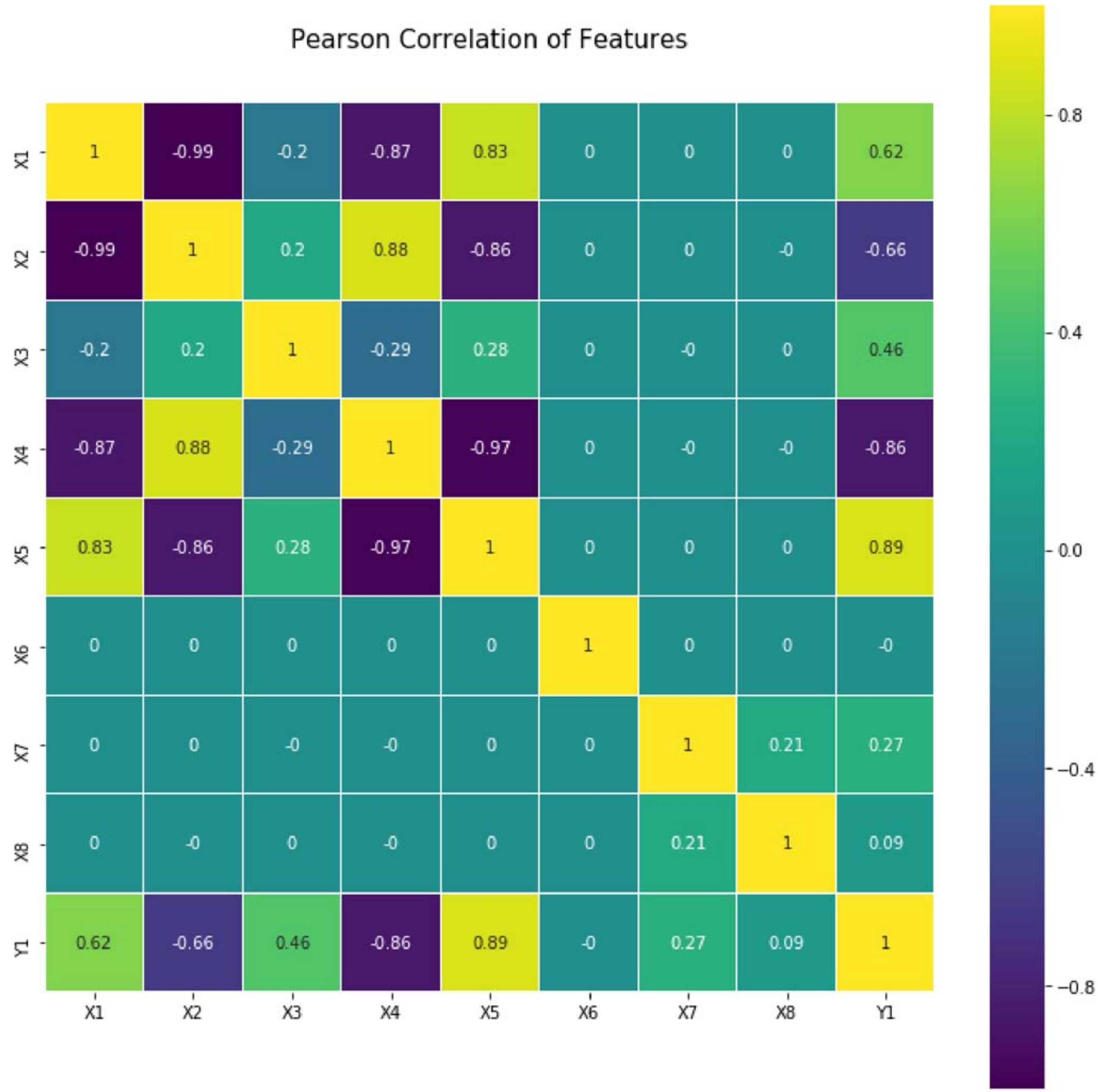
In [9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns




df = pd.read_csv('Energy.csv')
print(df.describe())
print(df.info())

# df.hist()
# plt.show()

pd.plotting.scatter_matrix(df, figsize=(13,10))
plt.show()

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(df.corr().round(2)\
            ,linewidths=0.1,vmax=1.0, square=True, cmap=colormap, \
            linecolor='white', annot=True);
plt.show()

print("There appears to be sufficient data as there are no missing values. However, in certain cases we see a hig
```

```
               X1          X2          X3          X4         X5          X6  \
count  768.000000  768.000000  768.000000  768.000000  768.00000  768.000000
mean     0.764167  671.708333  318.500000  176.604167    5.25000    3.500000
std      0.105777   88.086116   43.626481   45.165950    1.75114    1.118763
min      0.620000  514.500000  245.000000  110.250000    3.50000    2.000000
25%      0.682500  606.375000  294.000000  140.875000    3.50000    2.750000
50%      0.750000  673.750000  318.500000  183.750000    5.25000    3.500000
75%      0.830000  741.125000  343.000000  220.500000    7.00000    4.250000
max      0.980000  808.500000  416.500000  220.500000    7.00000    5.000000

               X7         X8          Y1
count  768.000000  768.00000  768.000000
mean     0.234375    2.81250   22.307201
```

```
std      0.133221    1.55096   10.090196
min      0.000000    0.00000    6.010000
25%      0.100000    1.75000   12.992500
50%      0.250000    3.00000   18.950000
75%      0.400000    4.00000   31.667500
max      0.400000    5.00000   43.100000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
X1    768 non-null float64
X2    768 non-null float64
X3    768 non-null float64
X4    768 non-null float64
X5    768 non-null float64
X6    768 non-null int64
X7    768 non-null float64
X8    768 non-null int64
Y1    768 non-null float64
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
None
```



X1
X2
X3
X4
X5
X6
X7
X8
Y1




```
There appears to be sufficient data as there are no missing values. However, in certain cases we see a high cor
relation between variables such as X5 and Y1 despite X5 having two very dominant discrete outputs as seen by it
s histogram.
```

REGRESSION: LABELS ARE CONTINUOUS VALUES. Here the model is trained to predict a continuous value for each instance. On inputting a feature vector into the model, the trained model is able to predict a continuous value for that instance.

Q2.1: Train a linear regression model on 85 percent of the given dataset, what is the intercept value and coefficient values.

In [6]:
```
import sklearn as sk

X = df.drop("Y1", axis=1) # Training & Validation data
Y = df["Y1"] # Response / Target Variable

print(X.shape, Y.shape)

np.random.seed(1337) # set random seed for reproducibility

from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.15)

print(X_train.shape, Y_train.shape)
print(X_val.shape, Y_val.shape)

from sklearn.linear_model import LinearRegression
linreg = LinearRegression() # instantiate
linreg.fit(X_train, Y_train) # fit
Y_pred = linreg.predict(X_train) # predict
print("Intercept: ", linreg.intercept_, " Coeff: ", linreg.coef_)

acc_lin = sum(Y_pred == Y_train)/len(Y_train)*100
print('Linear Regression accuracy:', str(acc_lin),'%')
```

```
(768, 8) (768,)
(652, 8) (652,)
(116, 8) (116,)
Intercept:  75.0551533742  Coeff:  [ -5.93110193e+01   4.03306349e+11  -4.03306349e+11  -8.06612698e+11
   4.26403299e+00  -5.36061317e-02   1.96930632e+01   1.56880439e-01]
Linear Regression accuracy: 0.0 %
```

Q.2.2: Report model performance using 'ROOT MEAN SQUARE' error metric on:

1. Data that was used for training(Training error)

2. On the 15 percent of unseen data (test error)

In [7]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt

rms = sqrt(mean_squared_error(Y_train, Y_pred))
print("RMSE of training data: ", rms)

Y_pred = linreg.predict(X_val) # predict

rms = sqrt(mean_squared_error(Y_val, Y_pred))
print("RMSE of test data: ", rms)
```

```
RMSE of training data:  2.8438518411413054
RMSE of test data:  3.338851955651158
```

Q2.3: Lets us see the effect of amount of data on the performance of prediction model.Use varying amounts of Training data (100,200,300,400,500,all) to train regression models and report training error and validation error in each case. Validation data/Test data is the same as above for all these cases.

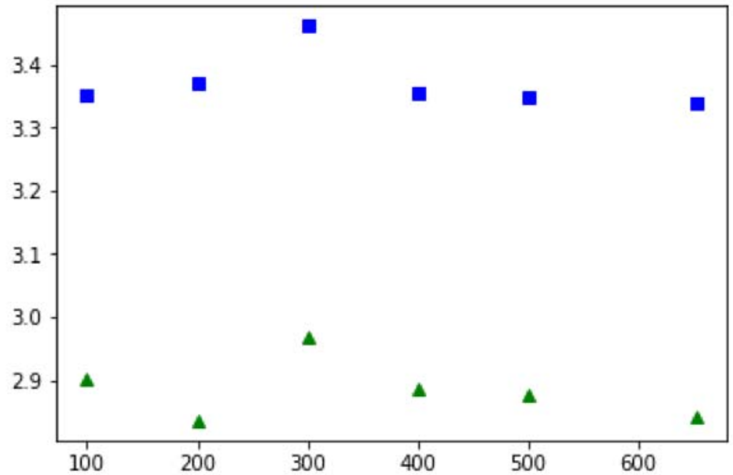
Plot error rates vs number of training examples.Comment on the relationshipyou observe in the plot, between the amount of data used to train the model and the validation accuracy of the model.

Hint: Use array indexing to choose varying data amounts

In [12]:
```
linreg = LinearRegression() # instantiate
data_amounts = [100,200,300,400,500, len(X_train)]
train_error = []
val_error = []
for data in data_amounts:
#     X = df.iloc[0:data].drop("Y1", axis=1) # Training & Validation data
#     Y = df.iloc[0:data]["Y1"] # Response / Target Variable
#     X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.15)


    linreg.fit(X_train[0:data], Y_train[0:data]) # fit
    Y_pred = linreg.predict(X_train[0:data]) # predict
    rms = sqrt(mean_squared_error(Y_train[0:data], Y_pred))
    train_error.append(rms)

    Y_pred = linreg.predict(X_val) # predict
    rms = sqrt(mean_squared_error(Y_val, Y_pred))
    val_error.append(rms)


results = pd.DataFrame(data={'data_amount':data_amounts, 'train_error':train_error, 'val_error':val_error})
# print(data_amounts)
# print(train_error)
# print(val_error)
print(results)
plt.plot(data_amounts, train_error, 'g^', data_amounts, val_error, 'bs')
# plt.legend(handles=['Train Error', 'Validation Error'])
plt.show()

# x, y = pd.Series(data_amounts, name="data_amounts"), pd.Series(train_error, name="train_error")
# ax = sns.regplot(x=x, y=y, marker="+")

print("The graph is inconclusive if the error will decrease with a larger training set. This un-informative graph
```

```
   data_amount  train_error  val_error
0          100     2.903690   3.351222
1          200     2.836287   3.369271
2          300     2.970171   3.461067
3          400     2.885219   3.352798
```

```
4          500     2.876597   3.347439
5          652     2.843852   3.338852
```




```
The graph is inconclusive if the error will decrease with a larger training set. This un-informative graph is p
artially due to the random seeding of 1337, if another number such as 42 was used, the graph is quite clearer t
o suggest there is an ideal amount before the test error score worsens.
```

CLASSIFICATION: LABELS ARE DISCRETE VALUES. Here the model is trained to classify each instance into a set of predefined discrete classes. On inputting a feature vector into the model, the trained model is able to predict a class of that instance. You can also output the probabilities of an instance belnging to a class.

Q 3.1: Bucket values of 'y1' i.e 'Heating Load' from the original dataset into 3 classes:

0: 'Low' (< 15),
1: 'Medium' (15-30),
2: 'High' (>30)

This converts the given dataset into a classification problem, classes being, Heating load is: *low, medium or high*. Use this datset with transformed 'heating load' for creating a logistic regression classifiction model that predicts heating load type of a building. Use test-train split ratio of 0.15.

Report training and test accuracies and confusion matrices.

HINT: Use pandas.cut

In [14]:
```
df.Y1 = pd.cut(df.Y1,[0,15,30,float("inf")],labels=["low", "medium", "high"])
df
X = df.iloc[0:data].drop("Y1", axis=1) # Training & Validation data
Y = df.iloc[0:data]["Y1"] # Response / Target Variable
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.15)

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

logreg = LogisticRegression() # instantiate
logreg.fit(X_train, Y_train) # fit
Y_pred = logreg.predict(X_train) # predict
acc_log = sum(Y_pred == Y_train)/len(Y_train)*100
print('Logistic Regression accuracy of train data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_train, Y_pred))

Y_pred = logreg.predict(X_val) # predict
acc_log = sum(Y_pred == Y_val)/len(Y_val)*100
print('Logistic Regression accuracy of test data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_val, Y_pred))
```

```
Logistic Regression accuracy of train data: 81.23 %
[[ 97   0  38]
 [  0 216   0]
 [  8  58 137]]
Logistic Regression accuracy of test data: 82.65 %
[[20  0  6]
 [ 0 32  0]
 [ 1 10 29]]
```

Q3.2: One of the preprocessing steps in Data science is Feature Scaling i.e getting all our data on the same scale by setting same Min-Max of feature values. This makes training less sensitive to the scale of features . Scaling is important in algorithms that use distance based classification, SVM or K means or involve gradient descent optimization.If we Scale features in the range [0,1] it is called unity based normalization.

Perform unity based normalization on the above dataset and train the model again, compare model performance in training and validation with your previous model.

refer:http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler (http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler)
more at: https://en.wikipedia.org/wiki/Feature_scaling (https://en.wikipedia.org/wiki/Feature_scaling)

In [15]:
```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_val_minmax = min_max_scaler.fit_transform(X_val)

logreg = LogisticRegression() # instantiate
logreg.fit(X_train_minmax, Y_train) # fit
Y_pred = logreg.predict(X_train_minmax) # predict

acc_log = sum(Y_pred == Y_train)/len(Y_train)*100
print('Logistic Regression accuracy of train data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_train, Y_pred))

Y_pred = logreg.predict(X_val_minmax) # predict
acc_log = sum(Y_pred == Y_val)/len(Y_val)*100
print('Logistic Regression accuracy of test data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_val, Y_pred))

print("There is an improvement in the accuracy on both the test and train sets.\nAlso, the confusion matrix shows
```

```
Logistic Regression accuracy of train data: 80.32 %
[[103   0  32]
 [  0 216   0]
 [ 19  58 126]]
Logistic Regression accuracy of test data: 87.76 %
[[25  0  1]
 [ 0 32  0]
 [ 1 10 29]]
There is an improvement in the accuracy on both the test and train sets.
Also, the confusion matrix shows fewer errors
```

Part 2

1. Read `diabetesdata.csv` **file into a pandas dataframe. Analyze the data features, check for NaN values. About the data:**

1. **TimesPregnant**: Number of times pregnant
2. **glucoseLevel**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. **BP**: Diastolic blood pressure (mm Hg)
4. **insulin**: 2-Hour serum insulin (mu U/ml)

5. **BMI**: Body mass index (weight in kg/(height in m)^2)
6. **pedigree**: Diabetes pedigree function
7. **Age**: Age (years)
8. **IsDiabetic**: 0 if not diabetic or 1 if diabetic)

2. Preprocess data to replace NaN values in a feature(if any) using mean of the feature.
Train logistic regression, SVM, perceptron, kNN, xgboost and random forest models using this preprocessed data with 20% test split.Report training and test accuracies.

In [16]:
```
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bays
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier #stochastic gradient descent
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb


import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('diabetesdata.csv')

# Replace NaN values with mean
df.isnull().values.any()
df = df.fillna(df.mean())
df.isnull().values.any()

X = df.drop("IsDiabetic", axis=1) # Training & Validation data
Y = df["IsDiabetic"] # Response / Target Variable

# LOGISTIC REGRESSION
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.20)
logreg = LogisticRegression() # instantiate
logreg.fit(X_train, Y_train) # fit
Y_pred = logreg.predict(X_train) # predict
acc_log = sum(Y_pred == Y_train)/len(Y_train)*100
print('Logistic Regression accuracy of train data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_train, Y_pred))

Y_pred = logreg.predict(X_val) # predict
acc_log = sum(Y_pred == Y_val)/len(Y_val)*100
print('Logistic Regression accuracy of test data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_val, Y_pred))
print("\n")

# SVM
svc = SVC()
svc.fit(X_train, Y_train)
acc_svc = svc.score(X_train, Y_train)
```

```
print("SVM accuracy of train data:", acc_svc)

acc_svc = svc.score(X_val, Y_val)
print("SVM accuracy of test data:", acc_svc)
print("\n")


# Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
acc_perceptron = perceptron.score(X_train, Y_train)
print("Perception accuracy of train data:", acc_perceptron)

acc_perceptron = perceptron.score(X_val, Y_val)
print("Perception accuracy of test data:", acc_perceptron)
print("\n")

# KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
acc_knn = knn.score(X_train, Y_train)
print("KNN accuracy of train data: ", acc_knn)

acc_knn = knn.score(X_val, Y_val)
print("KNN accuracy of test data: ", acc_knn)
print("\n")

# XGBoost, same API as scikit-learn
gradboost = xgb.XGBClassifier(n_estimators=1000)
gradboost.fit(X_train, Y_train)
acc_gradboost = gradboost.score(X_train, Y_train)
print("XGBoost accuracy of train data: ", acc_gradboost)

acc_gradboost = gradboost.score(X_val, Y_val)
print("XGBoost accuracy of test data: ", acc_gradboost)
print("\n")

# Random Forest
random_forest = RandomForestClassifier(n_estimators=1000)
random_forest.fit(X_train, Y_train)
acc_random_forest = random_forest.score(X_train, Y_train)
print("Random Forest accuracy of train data: ", acc_random_forest)
```

```
acc_random_forest = random_forest.score(X_val, Y_val)
print("Random Forest accuracy of test data: ", acc_random_forest)
print("\n")
```

```
C:\Users\Ashis Ghosh\AppData\Local\conda\conda\envs\data-x\lib\site-packages\sklearn\cross_validation.py:41: De
precationWarning: 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 di
fferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Logistic Regression accuracy of train data: 76.55 %
[[355  43]
 [101 115]]
Logistic Regression accuracy of test data: 77.27 %
[[87 15]
 [20 32]]


SVM accuracy of train data: 1.0
SVM accuracy of test data: 0.662337662338


Perception accuracy of train data: 0.431596091205
Perception accuracy of test data: 0.441558441558


KNN accuracy of train data:  0.842019543974
KNN accuracy of test data:  0.714285714286


XGBoost accuracy of train data:  1.0
XGBoost accuracy of test data:  0.74025974026


Random Forest accuracy of train data:  1.0
Random Forest accuracy of test data:  0.811688311688
```

3. What is the ratio of diabetic persons in 3 equirange bands of 'BMI' and 'Pedigree' in the provided dataset.

Convert these features - 'BP','insulin','BMI' and 'Pedigree' into categorical values by mapping different bands of values of these features to integers 0,1,2.

HINT: USE pd.cut with bin=3 to create 3 bins

In [17]:
```
df.BP = pd.cut(df.BP, 3, labels=[0,1,2])
df.insulin = pd.cut(df.insulin, 3, labels=[0,1,2])
df.BMI = pd.cut(df.BMI, 3, labels=[0,1,2])
df.Pedigree = pd.cut(df.Pedigree, 3, labels=[0,1,2])
```

In [18]:
```
print("Ratio of diabetic persons by BMI range")
print("BMI 0: ", df[df.BMI==0].IsDiabetic.sum()/df.IsDiabetic.sum())
print("BMI 1: ", df[df.BMI==1].IsDiabetic.sum()/df.IsDiabetic.sum())
print("BMI 2: ", df[df.BMI==2].IsDiabetic.sum()/df.IsDiabetic.sum())

print("\n")

print("Ratio of diabetic persons by Pedigree range")
print("Pedigree 0: ", df[df.Pedigree==0].IsDiabetic.sum()/df.IsDiabetic.sum())
print("Pedigree 1: ", df[df.Pedigree==1].IsDiabetic.sum()/df.IsDiabetic.sum())
print("Pedigree 2: ", df[df.Pedigree==2].IsDiabetic.sum()/df.IsDiabetic.sum())

print("\n")

print("Ratio of diabetic persons in each BMI range")
print("BMI 0: ", df[df.BMI==0].IsDiabetic.sum()/len(df[df.BMI==0]))
print("BMI 1: ", df[df.BMI==1].IsDiabetic.sum()/len(df[df.BMI==1]))
print("BMI 2: ", df[df.BMI==2].IsDiabetic.sum()/len(df[df.BMI==2]))

print("\n")

print("Ratio of diabetic persons in each Pedigree range")
print("Pedigree 0: ", df[df.Pedigree==0].IsDiabetic.sum()/len(df[df.Pedigree==0]))
print("Pedigree 1: ", df[df.Pedigree==1].IsDiabetic.sum()/len(df[df.Pedigree==1]))
print("Pedigree 2: ", df[df.Pedigree==2].IsDiabetic.sum()/len(df[df.Pedigree==2]))
```

```
Ratio of diabetic persons by BMI range
BMI 0:  0.007462686567164179
BMI 1:  0.9104477611940298
BMI 2:  0.08208955223880597


Ratio of diabetic persons by Pedigree range
Pedigree 0:  0.835820895522388
Pedigree 1:  0.14925373134328357
Pedigree 2:  0.014925373134328358


Ratio of diabetic persons in each BMI range
BMI 0:  0.0392156862745098
BMI 1:  0.35829662261380324
BMI 2:  0.6111111111111112
```

```
Ratio of diabetic persons in each Pedigree range
Pedigree 0:  0.327007299270073
Pedigree 1:  0.5405405405405406
Pedigree 2:  0.4444444444444444
```

4. Now consider the original dataset again, instead of generalizing the NAN values with the mean of the feature we will try assigning values to NANs based on some hypothesis. For example for age we assume that the relation between BMI and BP of people is a reflection of the age group.We can have 9 types of BMI and BP relations and our aim is to find the median age of each of that group:

Your Age guess matrix will look like this:

BMI	0	1	2
BP			
0	a00	a01	a02
1	a10	a11	a12
2	a20	a21	a22

Create a guess_matrix for NaN values of *'Age'* (using 'BMI' and 'BP') and *'glucoseLevel'* (using 'BP' and 'Pedigree') for the given dataset and assign values accordingly to the NaNs in 'Age' or *'glucoseLevel'* .

Refer to how we guessed age in the titanic notebook in the class.

In [19]:
```
df = pd.read_csv('diabetesdata.csv')
df.BP = pd.cut(df.BP, 3, labels=[0,1,2])
# df.insulin = pd.cut(df.insulin, 3, labels=[0,1,2])
df.BMI = pd.cut(df.BMI, 3, labels=[0,1,2])
df.Pedigree = pd.cut(df.Pedigree, 3, labels=[0,1,2])
print(df.head())

guess_ages = np.zeros((3,3),dtype=int) #initialize
guess_glucose = np.zeros((3,3),dtype=int) #initialize

# Fill the NA's for the Age columns
# with "qualified guesses"

for i in range(0, 3):
    for j in range(0,3):
        guess_df_age = df[(df['BMI'] == i)&(df['BP'] == j)]['Age'].dropna()
        guess_df_glucose = df[(df['BP'] == i)&(df['Pedigree'] == j)]['glucoseLevel'].dropna()

        # Extract the median age for this group
        # (less sensitive) to outliers
        age_guess = guess_df_age.median()
        glucose_guess = guess_df_glucose.median()

        # Convert random age float to int
        guess_ages[i,j] = int(age_guess)
        guess_glucose[i,j] = int(glucose_guess)

print('Guess_Age table:\n',guess_ages)
print('Guess_Glucose table:\n',guess_glucose)
print ('\nAssigning age values to NAN age values in the dataset...')

for i in range(0, 3):
    for j in range(0, 3):
        df.loc[ (df.Age.isnull()) & (df.BMI == i) \
                & (df.BP == j),'Age'] = guess_ages[i,j]
        df.loc[ (df.glucoseLevel.isnull()) & (df.BP == i) \
                & (df.Pedigree == j),'glucoseLevel'] = guess_glucose[i,j]


print()

print('Done!')
```

```
print(df.isnull().values.any())
print(df.isnull().sum())
df.head()
```

```
   TimesPregnant  glucoseLevel BP  insulin BMI Pedigree   Age  IsDiabetic
0              6         148.0  1        0   1        0  50.0           1
1              1           NaN  1        0   1        0  31.0           0
2              8         183.0  1        0   1        0   NaN           1
3              1           NaN  1       94   1        0  21.0           0
4              0         137.0  0      168   1        2  33.0           1
Guess_Age table:
 [[24 25 55]
 [29 29 37]
 [33 32 31]]
Guess_Glucose table:
 [[115 127 137]
 [112 115 149]
 [133 129 159]]

Assigning age values to NAN age values in the dataset...

Done!
False
TimesPregnant    0
glucoseLevel     0
BP               0
insulin          0
BMI              0
Pedigree         0
Age              0
IsDiabetic       0
dtype: int64
```

Out[19]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	148.0	1	0	1	0	50.0	1
1	1	112.0	1	0	1	0	31.0	0
2	8	183.0	1	0	1	0	29.0	1
3	1	112.0	1	94	1	0	21.0	0
4	0	137.0	0	168	1	2	33.0	1

5. Now, convert 'glucoseLevel' and 'Age' features also to categorical variables of 5 categories each.

Use this dataset (with all features in categorical form) to train perceptron, logistic regression and random forest models using 20% test split. Report training and test accuracies.

In [20]:
```
df.glucoseLevel = pd.cut(df.glucoseLevel, 5, labels=[0,1,2,3,4])
df.Age = pd.cut(df.Age, 5, labels=[0,1,2,3,4])
df.head()
```

Out[20]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	3	1	0	1	0	2	1
1	1	2	1	0	1	0	0	0
2	8	4	1	0	1	0	0	1
3	1	2	1	94	1	0	0	0
4	0	3	0	168	1	2	0	1

In [21]:

```
X = df.drop("IsDiabetic", axis=1) # Training & Validation data
Y = df["IsDiabetic"] # Response / Target Variable

print(X.shape, Y.shape)

np.random.seed(1337) # set random seed for reproducibility

from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.20)

# Perceptron
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
acc_perceptron = perceptron.score(X_train, Y_train)
print("Perception accuracy of train data:", acc_perceptron)

acc_perceptron = perceptron.score(X_val, Y_val)
print("Perception accuracy of test data:", acc_perceptron)
print("\n")

# Logistic Regression
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.20)
logreg = LogisticRegression() # instantiate
logreg.fit(X_train, Y_train) # fit
Y_pred = logreg.predict(X_train) # predict
acc_log = sum(Y_pred == Y_train)/len(Y_train)*100
print('Logistic Regression accuracy of train data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_train, Y_pred))

Y_pred = logreg.predict(X_val) # predict
acc_log = sum(Y_pred == Y_val)/len(Y_val)*100
print('Logistic Regression accuracy of test data:', str(round(acc_log,2)),'%')
print(confusion_matrix(Y_val, Y_pred))
print("\n")


# Random Forest
random_forest = RandomForestClassifier(n_estimators=1000)
random_forest.fit(X_train, Y_train)
```

```
acc_random_forest = random_forest.score(X_train, Y_train)
print("Random Forest accuracy of train data: ", acc_random_forest)

acc_random_forest = random_forest.score(X_val, Y_val)
print("Random Forest accuracy of test data: ", acc_random_forest)
print("\n")
```

```
(768, 7) (768,)
Perception accuracy of train data: 0.491856677524
Perception accuracy of test data: 0.480519480519


Logistic Regression accuracy of train data: 76.22 %
[[357  43]
 [103 111]]
Logistic Regression accuracy of test data: 73.38 %
[[89 11]
 [30 24]]


Random Forest accuracy of train data:  0.960912052117
Random Forest accuracy of test data:  0.753246753247
```

Type *Markdown* and LaTeX: α^2

Part 3

1. **Derive the expression for the optimal parameters in the linear regression equation, i.e. solve the normal equation for Ordinary Least Squares for the case of Simple Linear Regression, when we only have one input and one output**

Given a set of *n* points (X_i, Y_i) where Yi is dependent on Xi by a linear relation, find the best-fit line,

$$Z_i = aX_i + b$$

that minimizes the **sum of squared errors in Y**,i.e:

$$minimize \sum_i (Y_i - Z_i)^2$$

i. Show that

$$intercept \quad b = \overline{Y} - a.\overline{X} \quad and \quad slope \quad a = \frac{\sum_i (X_i - \overline{X})\square(Y_i - \overline{Y})}{\sum_i (X_i - \overline{X})^2}$$

where $\overline{X}$ and $\overline{Y}$ are the averages of the X values and the Y values, respectively.

ii. Show that slope *a* can be written as $a = r.(S_y/S_x)$ where S_y = the standard deviation of the Y values and S_x= the standard deviation of the X values and *r* is the correlation coefficient.

Please try to write a nice LateXed version of your answer, and do the derivations of the expressions as nicely as possible

---

$$Z_i = aX_i + b, b = \bar{Y} - a\bar{X}$$
$$min \sum_i (Y_i - Z_i)^2$$
$$\sum_i (Y_i - aX_i + b)^2$$
$$\sum_i (Y_i - aX_i + \bar{Y} - a\bar{X})^2$$
$$\sum_i (a(\bar{X} - X_i -) + (Y_i - \bar{Y}))^2$$
$$\sum_i [a^2(X_i - \bar{X})^2 - 2a(X_i - \bar{X})(Y_i - \bar{Y}) + (Y_i - \bar{Y})^2]$$

$$min(f) \quad at \quad f' = 0$$

$$=> 2a \sum_i (X_i - \bar{X})^2 - 2 \sum_i (X_i - \bar{X})(Y_i - \bar{Y}) = 0$$
$$a = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_i (X_i - \bar{X})^2}$$

$$a = r\frac{S_y}{S_x}$$

$$= r\frac{\sqrt{\frac{1}{N}\sum_i^N (Y_i - \bar{Y})^2}}{\sqrt{\frac{1}{N}\sum_i^N (X_i - \bar{X})^2}}$$
$$= r\frac{\sqrt{\frac{1}{N}\sum_i^N (Y_i - \bar{Y})^2}\sqrt{\frac{1}{N}\sum_i^N (X_i - \bar{X})^2}}{\sum_i (X_i - \bar{X})^2}N$$

$$where\ r\ is\ the\ Pearson\ Correlation\ Coefficient,\ defined\ as:$$
$$r = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\frac{1}{N}\sum_i^N (Y_i - \bar{Y})^2}\sqrt{\frac{1}{N}\sum_i^N (X_i - \bar{X})^2}}\frac{1}{N}$$

$$=> a = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\frac{1}{N}\sum_i^N (Y_i - \bar{Y})^2}\sqrt{\frac{1}{N}\sum_i^N (X_i - \bar{X})^2}}\frac{\sqrt{\frac{1}{N}\sum_i^N (Y_i - \bar{Y})^2}\sqrt{\frac{1}{N}\sum_i^N (X_i - \bar{X})^2}}{\sum_i (X_i - \bar{X})^2}$$
$$=> a = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_i (X_i - \bar{X})^2}$$

In [ ]:

```

```

Two Extra Credit Points: Fun with Webscraping & Text manipulation

(Mandatory for Grad students!)

`NOTE`: **If you are a Graduate Section student (enrolled in 290), the Extra Credit Questions are mandatory.**

1. Statistics in Presidential Debates

Your first task is to scrape Presidential Debates from the Commission of Presidential Debates website: http://www.debates.org/index.php?page=debate-transcripts (http://www.debates.org/index.php?page=debate-transcripts).

To do this, you are not allowed to manually look up the URLs that you need, instead you have to scrape them. The root url to be scraped is the one listed above, namely: http://www.debates.org/index.php?page=debate-transcripts (http://www.debates.org/index.php?page=debate-transcripts)

1. By using `requests` and `BeautifulSoup` find all the links / URLs on the website that links to transcriptions of **First Presidential Debates** from the years [2012, 2008, 2004, 2000, 1996, 1988, 1984, 1976, 1960]. In total you should find 9 links / URLs tat fulfill this criteria.
2. When you have a list of the URLs your task is to create a Data Frame with some statistics (see example of output below):
   A. Scrape the title of each link and use that as the column name in your Data Frame.
   B. Count how long the transcript of the debate is (as in the number of characters in transcription string). Feel free to include `\` characters in your count, but remove any breakline characters, i.e. `\n`. You will get credit if your count is +/- 10% from our result.
   C. Count how many times the word **war** was used in the different debates. Note that you have to convert the text in a smart way (to not count the word **warranty** for example, but counting **war.**, **war!**, **war,** or **War** etc.
   D. Also scrape the most common used word in the debate, and write how many times it was used. Note that you have to use the same strategy as in 3 in order to do this.

Tips:

In order to solve question 3 and 4 above it can be useful to work with Regular Expressions and explore methods on strings like `.strip()`, `.replace()`, `.find()`, `.count()`, `.lower()` etc. Both are very powerful tools to do string processing in Python. To count common words for example I used a `Counter` object and a Regular expression pattern for only words, see example:

```
from collections import Counter
import re

counts = Counter(re.findall(r"[\w']+", text.lower()))
```

Read more about Regular Expressions here: https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.html)

Example output of all of the answers to EC Question 1:

	October 3, 2012: The First Obama-Romney Presidential Debate	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Debate char length	94627	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
war_count	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
most_common_w	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
most_common_w_count	757	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

.

In [23]:
```
import requests
from bs4 import BeautifulSoup
from collections import Counter
import re

result = requests.get("http://www.debates.org/index.php?page=debate-transcripts")
c = result.content

soup = BeautifulSoup(c)
years = [2012, 2008, 2004, 2000, 1996, 1988, 1984, 1976, 1960]
samples = soup.find_all("a")

links = []
headings = []

for link in samples:
    if "The First" in link.text:
        links.append(link['href'])
        headings.append(link.text)

entries = None

for link in links:
    debate = requests.get(link)
    debate = debate.content
    debate = BeautifulSoup(debate)
    debate_content = debate.find("div", {"id": "content-sm"})
    debate_content = debate_content.find_all("p")

    text = None
    for content in debate_content:
        if (text!=None):
            text = text + content.text
        else:
            text = content.text


    text = text.replace("\n", "").replace("\'", "'")
    # print(len(text))
    counts = Counter(re.findall(r"[\w']+", text.lower()))
    # print(counts.most_common()[0])
    # print(counts["war"])
```

```
    # entry = [headings[0]]
    entry = [len(text)]
    entry.append(counts["war"] + counts["wars"])
    entry.append(counts.most_common()[0][0])
    entry.append(counts.most_common()[0][1])

    if (entries!=None):
        entries.append(entry)
    else:
        entries = [entry]

entries
```

```
Out[23]: [[94594, 5, 'the', 757],
          [182386, 48, 'the', 1470],
          [82685, 64, 'the', 857],
          [91040, 11, 'the', 919],
          [93057, 15, 'the', 876],
          [87458, 14, 'the', 803],
          [86654, 3, 'the', 865],
          [80701, 7, 'the', 856],
          [60901, 3, 'the', 778]]
```

In [24]:
```
row_names = ["Debate char length", "war_count", "most_common_w", "most_common_w_count"]

df = pd.DataFrame(entries, index=headings, columns=row_names )
df = df.T
df
```

Out[24]:

	October 3, 2012: The First Obama-Romney Presidential Debate	September 26, 2008: The First McCain-Obama Presidential Debate	September 30, 2004: The First Bush-Kerry Presidential Debate	October 3, 2000: The First Gore-Bush Presidential Debate	October 6, 1996: The First Clinton-Dole Presidential Debate	September 25, 1988: The First Bush-Dukakis Presidential Debate	October 7, 1984: The First Reagan-Mondale Presidential Debate	September 23, 1976: The First Carter-Ford Presidential Debate	Septemb 26, 196 The Fi Kenned Nix President Deba
Debate char length	94594	182386	82685	91040	93057	87458	86654	80701	609
war_count	5	48	64	11	15	14	3	7	
most_common_w	the	the	the	the	the	the	the	the	t
most_common_w_count	757	1470	857	919	876	803	865	856	7

2. Download and read in specific line from many data sets

Scrape the first 27 data sets from this URL http://people.sc.fsu.edu/~jburkardt/datasets/regression/ (http://people.sc.fsu.edu/~jburkardt/datasets/regression/) (i.e.`x01.txt` - `x27.txt`). Then, save the 5th line in each data set, this should be the name of the data set author (get rid of the # symbol, the white spaces and the comma at the end).

Count how many times (with a Python function) each author is the reference for one of the 27 data sets. Showcase your results, sorted, with the most common author name first and how many times he appeared in data sets. Use a Pandas DataFrame to show your results, see example.

Example output of the answer EC Question 2:

	Counts
Authors	
Helmut Spaeth	[illegible]
[illegible]	3
[illegible]	2
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]

In [26]:
```
datasets = range(1,28)
authors = []
for count in datasets:
    result = requests.get("http://people.sc.fsu.edu/~jburkardt/datasets/regression/x"+'{:0>2}'.format(count)+".t
    soup = BeautifulSoup(result.content)
    text = soup.find("p").text.splitlines()
    author = re.split(' and |, ',text[4])
    for author in author:
#         print(author)
        authors.append(author.replace("#    ", "").replace(",", ""))
print(authors)
set(authors)
```

```
['Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spaeth', 'R J Freund', 'P D Minto
n', 'D G Kleinbaum', 'L L Kupper', 'Helmut Spaeth', 'D G Kleinbaum', 'L L Kupper', 'K A Brownlee', 'Helmut Spae
th', 'Helmut Spaeth', 'S Chatterjee', 'B Price', 'Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spa
eth', 'Helmut Spaeth', 'R J Freund', 'P D Minton', 'Helmut Spaeth', 'Helmut Spaeth', 'Helmut Spaeth', 'S Chatte
rjee', 'B Price', 'S Chatterjee', 'B Price', 'S Chatterjee', 'B Price', 'S C Narula', 'J F Wellington', 'S C Na
rula', 'J F Wellington']
```

Out[26]:
```
{'B Price',
 'D G Kleinbaum',
 'Helmut Spaeth',
 'J F Wellington',
 'K A Brownlee',
 'L L Kupper',
 'P D Minton',
 'R J Freund',
 'S C Narula',
 'S Chatterjee'}
```

In [27]:
```
counter=Counter(authors)
df = pd.DataFrame.from_dict(counter, orient='index').reset_index()
df = df.rename(columns={'index':'Authors', 0:'Counts'})
df.sort_values("Counts", ascending=False).reset_index()
```

Out[27]:

	index	Authors	Counts
0	0	Helmut Spaeth	16
1	6	S Chatterjee	4
2	7	B Price	4
3	1	R J Freund	2
4	2	P D Minton	2
5	3	D G Kleinbaum	2
6	4	L L Kupper	2
7	8	S C Narula	2
8	9	J F Wellington	2
9	5	K A Brownlee	1

In [ ]: