hw2_regression_classification_webscraping_v2

February 28, 2018

1 Data-X Spring 2018: Homework 02

1.0.1 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.

1.1 Part 1

1.1.1 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 [1]: # Import Package
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.linear_model import Perceptron
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb


%matplotlib inline
```

```
/home/dasapunar/anaconda3/envs/data-x/lib/python3.6/site-packages/sklearn/cross_validation.py:41
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
# Distribution of each variable.
# Reading FileD
df = pd.read_csv('Energy.csv')
```

In [3]:
```
# Describing data (General)
print("Describing Data in a general view...")
df.describe()
```

```
Describing Data in a general view...
```

Out[3]:

	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	

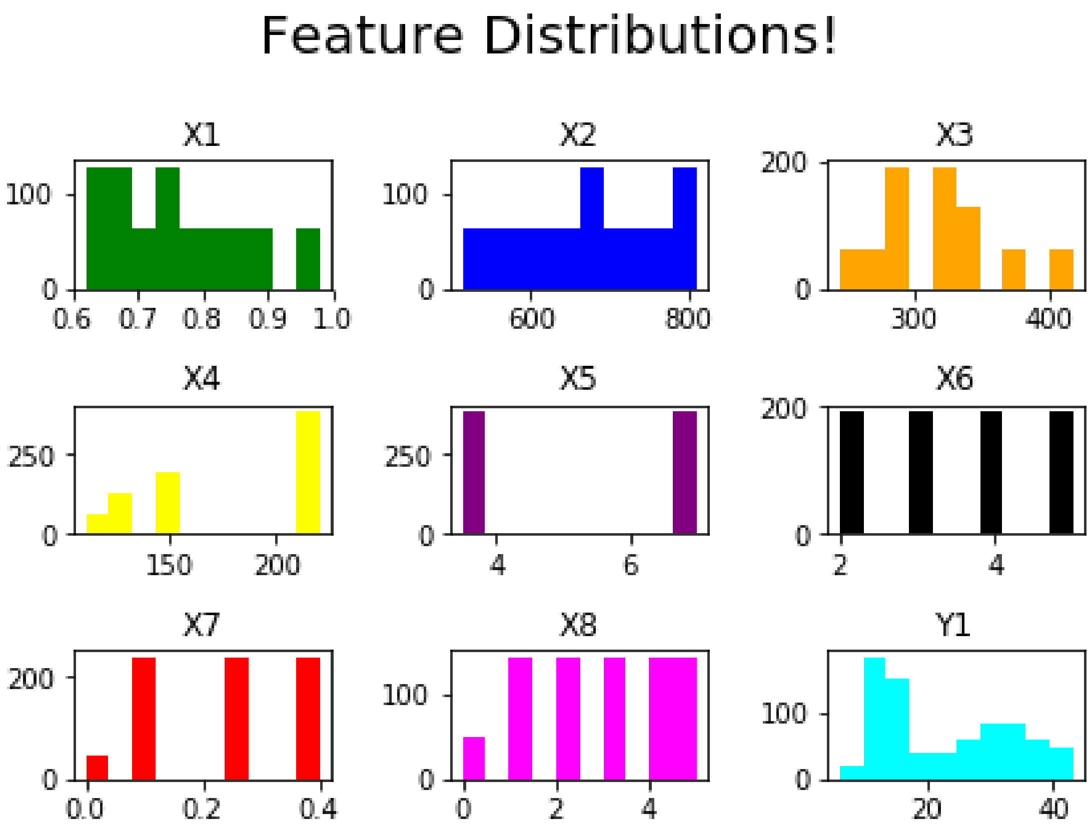
	X7	X8	Y1
count	768.000000	768.00000	768.000000
mean	0.234375	2.81250	22.307201
std	0.133221	1.55096	10.090196
min	0.000000	0.00000	6.010000
25%	0.100000	1.75000	12.992500
50%	0.250000	3.00000	18.950000
75%	0.400000	4.00000	31.667500
max	0.400000	5.00000	43.100000

```
In [4]: # Describe data features in terms of type, distribution range and mean values.

        def nice_display_basic_statistics(maxi, mini, mean):
            """
            Print in a nice way the data features distribution range and mean values

            Arguments:
                maxi -- python float containing the Max of the feature
                mini -- python float containing the Min of the feature
                mean -- python float containing the Mean of the feature

            """

            print("Max: ", maxi)
            print("Min: ", mini)
            print("Mean", mean)


        def nice_display(column_name, dtype, maxi, mini, mean):
            """
            Print in a nice way the data features in terms of type, distribution range and mean

            Arguments:
                column_name -- python string containing the name of the feature
                dtype -- python string containing the dtype of the feature
                maxi -- python float containing the Max of the feature
                mini -- python float containing the Min of the feature
                mean -- python float containing the Mean of the feature

            """

            if dtype == "float64":
                print("The feature " + column_name + ": ")
                print("Type: Float so is Continuous!")
                nice_display_basic_statistics(maxi, mini, mean)

            else:
                print("The feature " + column_name + ": ")
                print("Type: Integer so is Continuous!")
                nice_display_basic_statistics(maxi, mini, mean)

            print("-" * 30)

In [5]: # Describe data features in terms of type, distribution range and mean values.
        for i in df.columns:
            nice_display(i, df[i].dtype, df[i].max(), df[i].min(), df[i].mean())

The feature X1:
```

```
Type: Float so is Continuous!
Max:  0.98
Min:  0.62
Mean 0.7641666666666677
------------------------------
The feature X2:
Type: Float so is Continuous!
Max:  808.5
Min:  514.5
Mean 671.7083333333334
------------------------------
The feature X3:
Type: Float so is Continuous!
Max:  416.5
Min:  245.0
Mean 318.5
------------------------------
The feature X4:
Type: Float so is Continuous!
Max:  220.5
Min:  110.25
Mean 176.60416666666666
------------------------------
The feature X5:
Type: Float so is Continuous!
Max:  7.0
Min:  3.5
Mean 5.25
------------------------------
The feature X6:
Type: Integer so is Continuous!
Max:  5
Min:  2
Mean 3.5
------------------------------
The feature X7:
Type: Float so is Continuous!
Max:  0.4
Min:  0.0
Mean 0.23437500000000186
------------------------------
The feature X8:
Type: Integer so is Continuous!
Max:  5
Min:  0
Mean 2.8125
------------------------------
The feature Y1:
```

```
Type: Float so is Continuous!
Max:  43.1
Min:  6.01
Mean 22.307200520833305
------------------------------
```

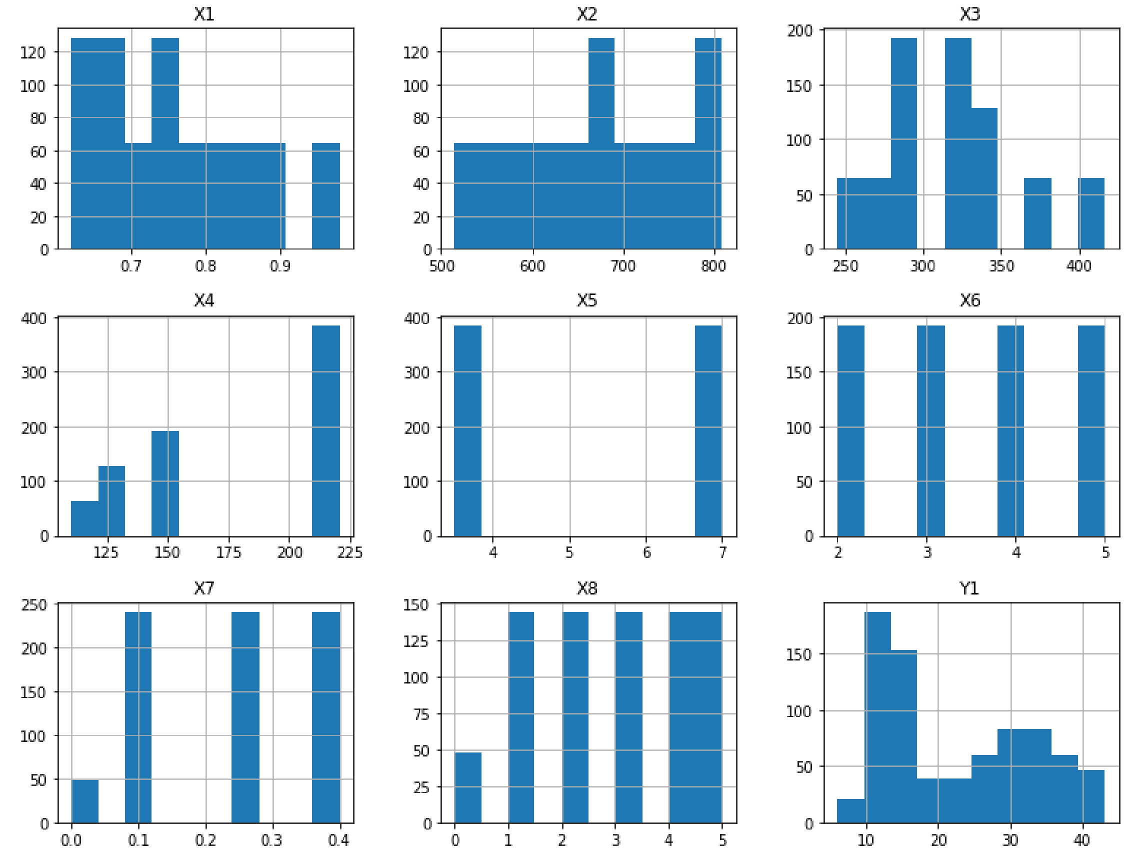
In [6]:
```python
# Distribution of each variable.

f = plt.figure()

# Specify the grid
ax1 = plt.subplot2grid((3,3), (0,0))
ax2 = plt.subplot2grid((3,3), (0,1))
ax3 = plt.subplot2grid((3,3), (0,2))
ax4 = plt.subplot2grid((3,3), (1,0))
ax5 = plt.subplot2grid((3,3), (1,1))
ax6 = plt.subplot2grid((3,3), (1,2))
ax7 = plt.subplot2grid((3,3), (2,0))
ax8 = plt.subplot2grid((3,3), (2,1))
ax9 = plt.subplot2grid((3,3), (2,2))

ax1.hist(df["X1"], color="Green")
ax2.hist(df["X2"], color="Blue")
ax3.hist(df["X3"], color="Orange")
ax4.hist(df["X4"], color="Yellow")
ax5.hist(df["X5"], color="Purple")
ax6.hist(df["X6"], color="Black")
ax7.hist(df["X7"], color="Red")
ax8.hist(df["X8"], color="Magenta")
ax9.hist(df["Y1"], color="Cyan")

# Add titles
ax1.set_title('X1')
ax2.set_title('X2')
ax3.set_title('X3')
ax4.set_title('X4')
ax5.set_title('X5')
ax6.set_title('X6')
ax7.set_title('X7')
ax8.set_title('X8')
ax9.set_title('Y1')


f.suptitle('Feature Distributions!',fontsize=20, y=1.1) # y location

f.tight_layout()
```




```
In [7]: # Adittional INFO
        df.info()
```

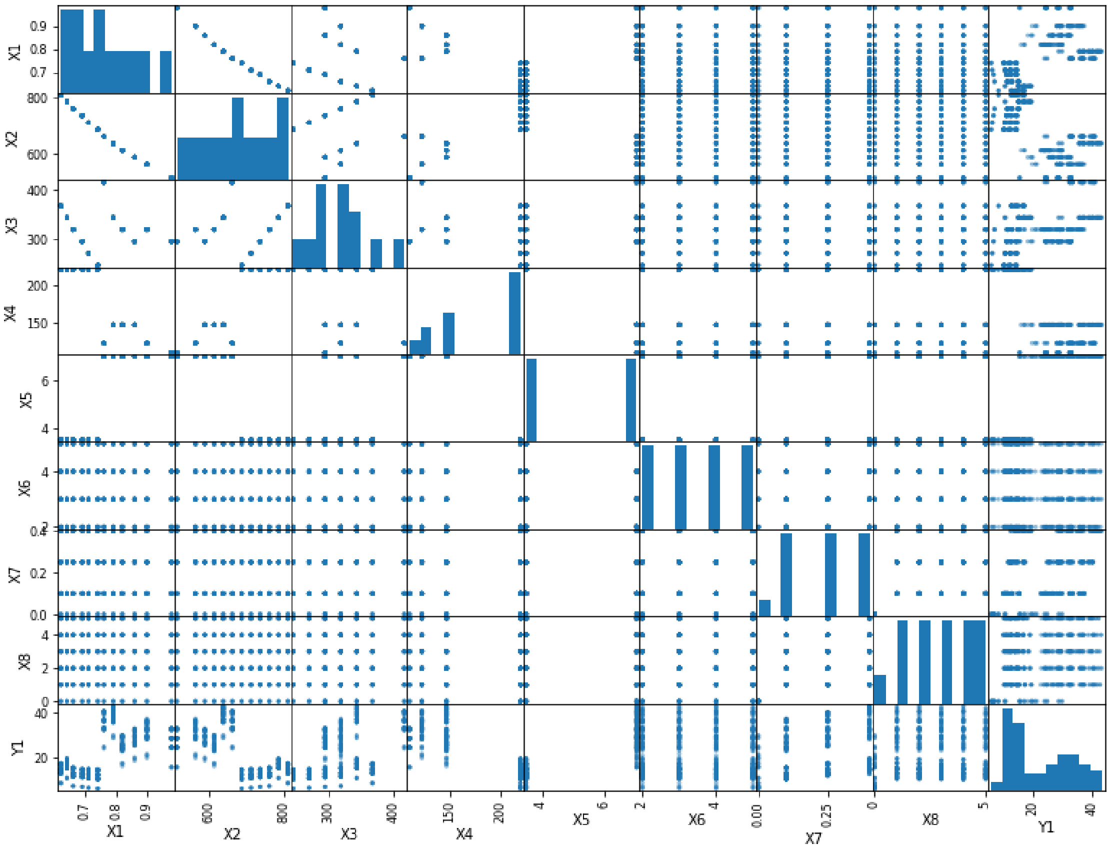
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
X1    768 non-null float64
X2    768 non-null float64
X3    768 non-null float64
X4    768 non-null float64
X5    768 non-null float64
X6    768 non-null int64
X7    768 non-null float64
X8    768 non-null int64
Y1    768 non-null float64
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
```

```
In [8]: # More graphs (with another method)
        df.hist(figsize=(13,10))
        plt.show()
```




```
In [9]: # Relations between features
        pd.tools.plotting.scatter_matrix(df,figsize=(13,10));
```

```
/home/dasapunar/anaconda3/envs/data-x/lib/python3.6/site-packages/ipykernel_launcher.py:2: Futur
```



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 [10]: # SHUFFLE data.
         data = shuffle(df).reset_index(drop=True)
```

```
In [11]: # Get NaNs
         print('Number of NaNs in the dataframe:\n',data.isnull().sum())
         data.head()
```

```
Number of NaNs in the dataframe:
 X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
```

```
X8    0
Y1    0
dtype: int64
```

```
Out[11]:       X1     X2     X3     X4   X5  X6    X7  X8     Y1
         0  0.64  784.0  343.0  220.5  3.5   3  0.25   4  16.69
         1  0.86  588.0  294.0  147.0  7.0   5  0.40   2  31.64
         2  0.86  588.0  294.0  147.0  7.0   4  0.25   5  28.31
         3  0.79  637.0  343.0  147.0  7.0   3  0.10   3  35.48
         4  0.86  588.0  294.0  147.0  7.0   3  0.40   5  31.81
```

```
In [12]: # Separate X from the Data Set.
         X=data.iloc[:,:-1]
         X.head()
```

```
Out[12]:       X1     X2     X3     X4   X5  X6    X7  X8
         0  0.64  784.0  343.0  220.5  3.5   3  0.25   4
         1  0.86  588.0  294.0  147.0  7.0   5  0.40   2
         2  0.86  588.0  294.0  147.0  7.0   4  0.25   5
         3  0.79  637.0  343.0  147.0  7.0   3  0.10   3
         4  0.86  588.0  294.0  147.0  7.0   3  0.40   5
```

```
In [13]: # Get Labels from the Data Set.
         Y=data['Y1']
         Y.head()
```

```
Out[13]: 0    16.69
         1    31.64
         2    28.31
         3    35.48
         4    31.81
         Name: Y1, dtype: float64
```

```
In [14]: # Which are my shapes?
         print("Feature vector shape=", X.shape)
         print("Class shape=", Y.shape)
```

```
Feature vector shape= (768, 8)
Class shape= (768,)
```

Split Data

```
In [15]: # Split data into Training and Validation set  using sklearn function.
         x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_state=
         print ('Number of samples in training data:',len(x_train))
         print ('Number of samples in validation data:',len(x_test))
```

```
Number of samples in training data: 652
Number of samples in validation data: 116
```

Train Model

```
In [16]: # Name our Logistic Regression object.
         LinearRegressionModel= LinearRegression()

         LinearRegressionModel.fit(x_train, y_train)
         Z_train=LinearRegressionModel.predict(x_train)
         Z_test = LinearRegressionModel.predict(x_test)

         # The Coefficients.
         print('Coefficients:', LinearRegressionModel.coef_)

         # The Interception.
         print('Intercept:', LinearRegressionModel.intercept_)
```

```
Coefficients: [ -6.27201639e+01   1.09565418e+12  -1.09565418e+12  -2.19130836e+12
   4.09228840e+00   3.15251385e-02   1.94709175e+01   1.98792642e-01]
Intercept: 81.3071319018
```

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)

Mean Squared Error and Accuracy for Training and Test.

```
In [17]: # The Mean Squared Error.
         print("Mean squared error of training:",np.mean((Z_train - y_train) ** 2))
         print("Mean squared error of test:",np.mean((Z_test - y_test) ** 2))

         # The Accuracy for Training and Test.
         print("Accuracy for Training: ", LinearRegressionModel.score(x_train, y_train)* 100, "%
         print("Accuracy for Test", LinearRegressionModel.score(x_test, y_test) * 100, "%")
```

```
Mean squared error of training: 8.28029588498402
Mean squared error of test: 9.558285805743669
Accuracy for Training:  91.7059531879 %
Accuracy for Test 91.4697157536 %
```

__ 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

Different Experiments ASSUMING THAT THE QUESTION AIM TO THE AMOUNT OF DATA CORRESPONDS TO THE WHOLE DATA SET! :)

```
In [18]: def different_experiments(X, Y, amount_training_data, costs_train, costs_test, variance
             """
             Print in a nice way the experiment.

             Arguments:
                 X -- Pandas Dataframe. X whole Data Set.
                 Y -- Pandas Dataframe. Labels of the Data Set.
                 amount_training_data -- Integer. Number of rows of the Training Set (n_x).
                 costs_train -- List Object. Array with all the costs (square mean error) in the
                                Training Set of the different experiments.
                 costs_test -- List Object. Array with all the costs (square mean error) in the
                               Test Set of the different experiments
                 variances -- List Object. Array with the variances between Train and Test of
                              the different experiments.
                 biases -- List Object. Array with the biases (Testing Accuracy) between Train
                           and Test of the different experiments.

             Return:
                 costs_train -- List Object. Array with all the costs (square mean error) in the
                                Training Set of the different experiments.
                 costs_test -- List Object. Array with all the costs (square mean error) in the
                               Test Set of the different experiments
                 variances -- List Object. Array with the variances between Train and Test of
                              the different experiments.
                 biases -- List Object. Array with the biases (Testing Accuracy) between Train
                           and Test of the different experiments.

             """

             error_test = 0

             # For beatiful display.
             print("-"*20, "AMOUNT OF TRAINING DATA: ", amount_training_data, " -"*20)

             # Spliting with the Amount Of Training Data.
             percentage = amount_training_data/X.shape[0]
             x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=1-percentage, r
             print ('Number of samples in training data:',len(x_train))
             print ('Number of samples in validation data:',len(x_test))

             # Name our logistic regression object.
             LinearRegressionModel= LinearRegression()

             # Fit Model.
             LinearRegressionModel.fit(x_train, y_train)
```

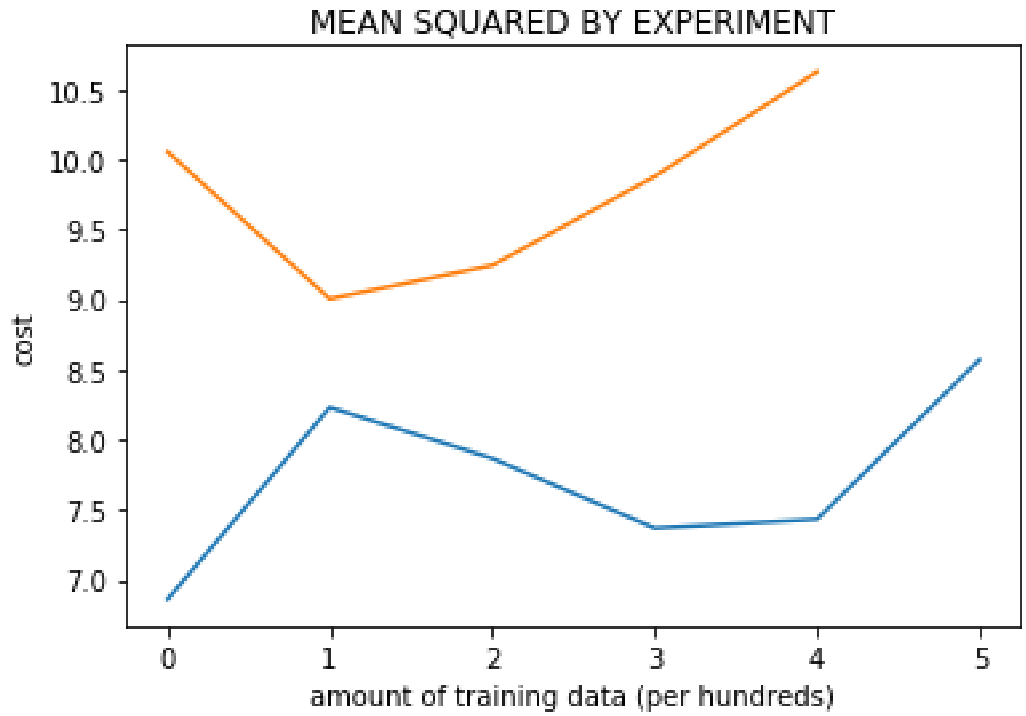
```python
    # Calculate Predict Vector for Train.
    Z_train=LinearRegressionModel.predict(x_train)

    # The Coefficients.
    print('Coefficients:', LinearRegressionModel.coef_)

    # The Interception.
    print('Intercept:', LinearRegressionModel.intercept_)

    # The mean squared error for Train.
    error_train = np.mean((Z_train - y_train) ** 2)
    print("Mean squared error of training:",error_train)

    # Costs, Bias or Accuracy for Training Set.
    costs_train.append(error_train)
    bias = LinearRegressionModel.score(x_train, y_train)* 100
    biases.append(bias)

    # For the border case that we don't have Test Set, the Training Set have the whole
    if amount_training_data != X.shape[0]:
        # Calculate Predict Vector for Test.
        Z_test = LinearRegressionModel.predict(x_test)

        # The mean squared error for Test.
        error_test = np.mean((Z_test - y_test) ** 2)
        costs_test.append(error_test)
        print("Mean squared error of test:",error_test)

        # Costs, Bias or Accuracy for Test Set.
        test_accuracy = LinearRegressionModel.score(x_test, y_test) * 100
        variance =  bias - test_accuracy
        variances.append(variance)
        print("Accuracy for Test", test_accuracy, "%")
        print("Variance: ", variance, "\n")

    # Printing Accuracy for Training.
    print("Accuracy for Training: ", bias, "%")

    return costs_train, costs_test, variances, biases
```

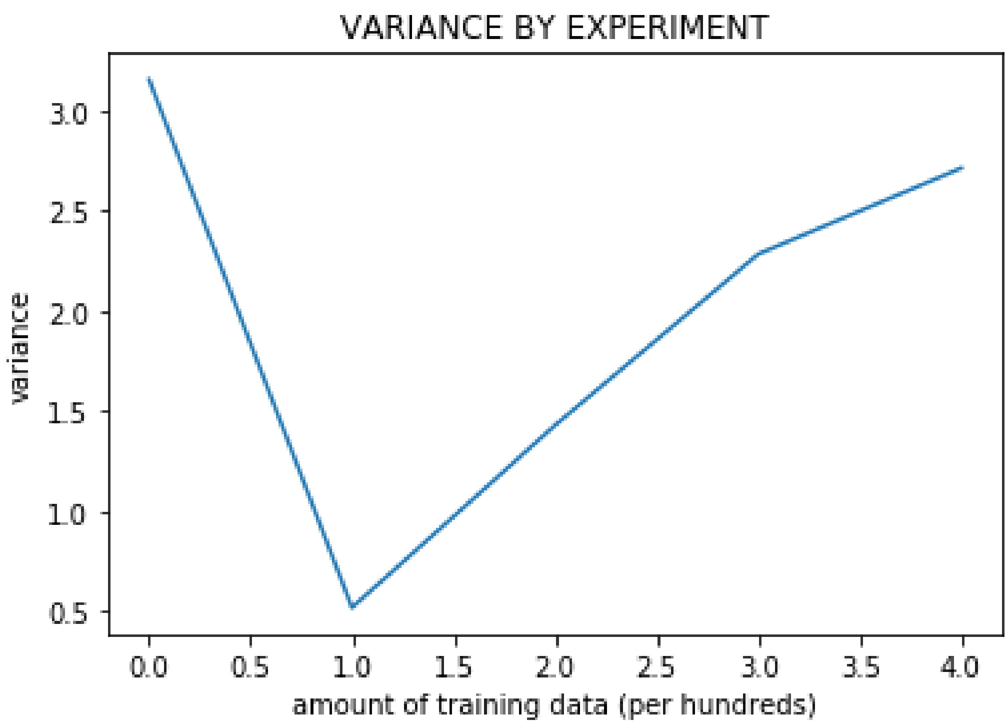
In [19]:
```python
# I AM ASSUMING THAT THE QUESTION AIM TO THE AMOUNT OF DATA CORRESPONDS TO THE WHOLE DA

costs_train = []
costs_tests = []
variances = []
biases = []
```

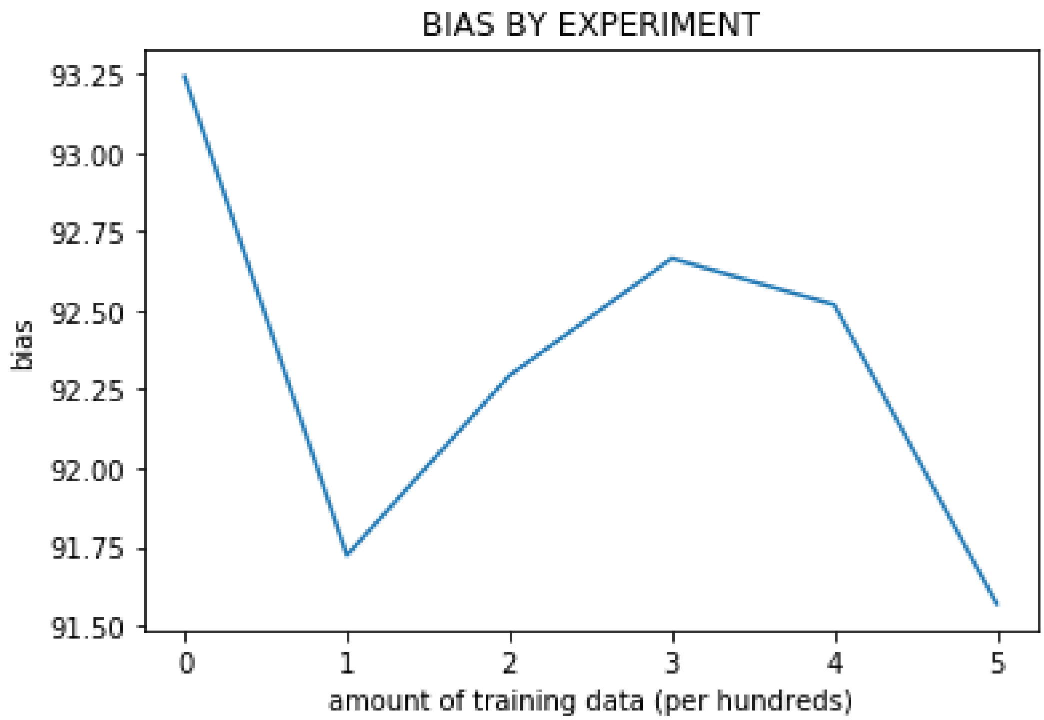
```python
costs_train, costs_tests, variances, biases = different_experiments(X,Y,100, costs_trai
costs_train, costs_tests, variances, biases = different_experiments(X,Y,200, costs_trai
costs_train, costs_tests, variances, biases = different_experiments(X,Y,300, costs_trai
costs_train, costs_tests, variances, biases = different_experiments(X,Y,400, costs_trai
costs_train, costs_tests, variances, biases = different_experiments(X,Y,500, costs_trai
costs_train, costs_tests, variances, biases = different_experiments(X,Y,X.shape[0], cos

plt.plot(costs_train)
plt.plot(costs_tests)
plt.ylabel('cost')
plt.xlabel('amount of training data (per hundreds)')
plt.title("MEAN SQUARED BY EXPERIMENT")
plt.show()

plt.plot(variances)
plt.ylabel('variance')
plt.xlabel('amount of training data (per hundreds)')
plt.title("VARIANCE BY EXPERIMENT")
plt.show()

plt.plot(biases)
plt.ylabel('bias')
plt.xlabel('amount of training data (per hundreds)')
plt.title("BIAS BY EXPERIMENT")
plt.show()
```

```
-------------------- AMOUNT OF TRAINING DATA:  100  - - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 100
Number of samples in validation data: 668
Coefficients: [ -6.58149892e+01  -7.50706783e-02   4.23500339e-02  -5.87103561e-02
   3.22688591e+00  -6.21573454e-01   1.77688367e+01   3.99317420e-01]
Intercept: 99.3068567102
Mean squared error of training: 6.861696633206532
Mean squared error of test: 10.0561126219286
Accuracy for Test 90.0883258428 %
Variance:  3.15336845013

Accuracy for Training:  93.2416942929 %
-------------------- AMOUNT OF TRAINING DATA:  200  - - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 200
Number of samples in validation data: 568
Coefficients: [ -5.96928554e+01  -5.95169120e-02   3.38038401e-02  -4.66603761e-02
   4.10805180e+00  -3.80409889e-01   1.83549451e+01   2.07711712e-01]
Intercept: 80.1083144657
Mean squared error of training: 8.230024342575234
Mean squared error of test: 9.00504450696288
Accuracy for Test 91.2024252296 %
Variance:  0.520393741623
```

```
Accuracy for Training:  91.7228189712 %
-------------------- AMOUNT OF TRAINING DATA:  300  - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 300
Number of samples in validation data: 468
Coefficients: [ -6.27469196e+01  -1.72613304e+12   1.72613304e+12   3.45226607e+12
   4.38104826e+00  -2.33913640e-01   1.95733226e+01   1.51494914e-01]
Intercept: 79.8503666667
Mean squared error of training: 7.8679619439222295
Mean squared error of test: 9.24468978316069
Accuracy for Test 90.8641381645 %
Variance:  1.42963043122

Accuracy for Training:  92.2937685958 %
-------------------- AMOUNT OF TRAINING DATA:  400  - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 400
Number of samples in validation data: 368
Coefficients: [ -6.01471603e+01  -5.37195576e-02   2.92611652e-02  -4.14903613e-02
   4.54078439e+00  -7.94759696e-02   1.96165414e+01   1.75521079e-01]
Intercept: 73.6930034228
Mean squared error of training: 7.370591504625857
Mean squared error of test: 9.879626173233907
Accuracy for Test 90.3820624432 %
Variance:  2.2828014008

Accuracy for Training:  92.664863844 %
-------------------- AMOUNT OF TRAINING DATA:  500  - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 500
Number of samples in validation data: 268
Coefficients: [ -5.88992226e+01  -5.51729787e-02   3.32768034e-02  -4.42248910e-02
   4.32115867e+00  -2.12417027e-02   1.92891418e+01   2.33375566e-01]
Intercept: 73.7587011471
Mean squared error of training: 7.434169642561867
Mean squared error of test: 10.624545252766051
Accuracy for Test 89.8060354152 %
Variance:  2.71229539669

Accuracy for Training:  92.5183308118 %
-------------------- AMOUNT OF TRAINING DATA:  768  - - - - - - - - - - - - - - - - - - - -
Number of samples in training data: 768
Number of samples in validation data: 0
Coefficients: [ -6.59942150e+01   1.31365691e+12  -1.31365691e+12  -2.62731382e+12
   4.15567333e+00  -2.28028174e-02   1.99438759e+01   2.03900281e-01]
Intercept: 85.3072005208
Mean squared error of training: 8.572116563756724
Accuracy for Training:  91.5694726373 %
```


MEAN SQUARED BY EXPERIMENT
cost
amount of training data (per hundreds)
10.5
10.0
9.5
9.0
8.5
8.0
7.5
7.0
0
1
2
3
4
5


VARIANCE BY EXPERIMENT
variance
amount of training data (per hundreds)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



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

Prepare Data

```
In [20]: # Cut the data Frame with the corresponding new Labels.
         df["Y1"] = pd.cut(df["Y1"], bins=3, labels=["Low", "Medium", "High"])
         df
```

```
Out[20]:       X1     X2     X3      X4   X5  X6   X7  X8      Y1
         0   0.98  514.5  294.0  110.25  7.0   2  0.0   0     Low
         1   0.98  514.5  294.0  110.25  7.0   3  0.0   0     Low
```

2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	Low
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	Low
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	Medium
5	0.90	563.5	318.5	122.50	7.0	3	0.0	0	Medium
6	0.90	563.5	318.5	122.50	7.0	4	0.0	0	Medium
7	0.90	563.5	318.5	122.50	7.0	5	0.0	0	Medium
8	0.86	588.0	294.0	147.00	7.0	2	0.0	0	Medium
9	0.86	588.0	294.0	147.00	7.0	3	0.0	0	Medium
10	0.86	588.0	294.0	147.00	7.0	4	0.0	0	Medium
11	0.86	588.0	294.0	147.00	7.0	5	0.0	0	Low
12	0.82	612.5	318.5	147.00	7.0	2	0.0	0	Low
13	0.82	612.5	318.5	147.00	7.0	3	0.0	0	Low
14	0.82	612.5	318.5	147.00	7.0	4	0.0	0	Low
15	0.82	612.5	318.5	147.00	7.0	5	0.0	0	Low
16	0.79	637.0	343.0	147.00	7.0	2	0.0	0	Medium
17	0.79	637.0	343.0	147.00	7.0	3	0.0	0	Medium
18	0.79	637.0	343.0	147.00	7.0	4	0.0	0	Medium
19	0.79	637.0	343.0	147.00	7.0	5	0.0	0	Medium
20	0.76	661.5	416.5	122.50	7.0	2	0.0	0	Medium
21	0.76	661.5	416.5	122.50	7.0	3	0.0	0	Medium
22	0.76	661.5	416.5	122.50	7.0	4	0.0	0	Medium
23	0.76	661.5	416.5	122.50	7.0	5	0.0	0	Medium
24	0.74	686.0	245.0	220.50	3.5	2	0.0	0	Low
25	0.74	686.0	245.0	220.50	3.5	3	0.0	0	Low
26	0.74	686.0	245.0	220.50	3.5	4	0.0	0	Low
27	0.74	686.0	245.0	220.50	3.5	5	0.0	0	Low
28	0.71	710.5	269.5	220.50	3.5	2	0.0	0	Low
29	0.71	710.5	269.5	220.50	3.5	3	0.0	0	Low
..	...	...	...	...	...	..	...	..	...
738	0.79	637.0	343.0	147.00	7.0	4	0.4	5	High
739	0.79	637.0	343.0	147.00	7.0	5	0.4	5	High
740	0.76	661.5	416.5	122.50	7.0	2	0.4	5	High
741	0.76	661.5	416.5	122.50	7.0	3	0.4	5	High
742	0.76	661.5	416.5	122.50	7.0	4	0.4	5	High
743	0.76	661.5	416.5	122.50	7.0	5	0.4	5	High
744	0.74	686.0	245.0	220.50	3.5	2	0.4	5	Low
745	0.74	686.0	245.0	220.50	3.5	3	0.4	5	Low
746	0.74	686.0	245.0	220.50	3.5	4	0.4	5	Low
747	0.74	686.0	245.0	220.50	3.5	5	0.4	5	Low
748	0.71	710.5	269.5	220.50	3.5	2	0.4	5	Low
749	0.71	710.5	269.5	220.50	3.5	3	0.4	5	Low
750	0.71	710.5	269.5	220.50	3.5	4	0.4	5	Low
751	0.71	710.5	269.5	220.50	3.5	5	0.4	5	Low
752	0.69	735.0	294.0	220.50	3.5	2	0.4	5	Low
753	0.69	735.0	294.0	220.50	3.5	3	0.4	5	Low
754	0.69	735.0	294.0	220.50	3.5	4	0.4	5	Low
755	0.69	735.0	294.0	220.50	3.5	5	0.4	5	Low
756	0.66	759.5	318.5	220.50	3.5	2	0.4	5	Low

```
757  0.66  759.5  318.5  220.50  3.5   3  0.4   5      Low
758  0.66  759.5  318.5  220.50  3.5   4  0.4   5      Low
759  0.66  759.5  318.5  220.50  3.5   5  0.4   5      Low
760  0.64  784.0  343.0  220.50  3.5   2  0.4   5      Low
761  0.64  784.0  343.0  220.50  3.5   3  0.4   5      Low
762  0.64  784.0  343.0  220.50  3.5   4  0.4   5      Low
763  0.64  784.0  343.0  220.50  3.5   5  0.4   5      Low
764  0.62  808.5  367.5  220.50  3.5   2  0.4   5      Low
765  0.62  808.5  367.5  220.50  3.5   3  0.4   5      Low
766  0.62  808.5  367.5  220.50  3.5   4  0.4   5      Low
767  0.62  808.5  367.5  220.50  3.5   5  0.4   5      Low

[768 rows x 9 columns]
```

In [21]:
```
# Shuffle data and storage in new variable "data2", since we are in the same Part I.
data2= shuffle(df).reset_index(drop=True)
data2.head()
```

Out[21]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1
0	0.64	784.0	343.0	220.5	3.5	5	0.25	4	Low
1	0.86	588.0	294.0	147.0	7.0	2	0.25	1	Medium
2	0.90	563.5	318.5	122.5	7.0	5	0.10	3	Medium
3	0.64	784.0	343.0	220.5	3.5	4	0.10	3	Low
4	0.76	661.5	416.5	122.5	7.0	4	0.10	3	High

Data Analisis

In [22]:
```
# Get NaNs
print('Number of NaNs in the dataframe2:\n',data2.isnull().sum())
```

```
Number of NaNs in the dataframe2:
 X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
X8    0
Y1    0
dtype: int64
```

In [23]:
```
# Information about data.
data2.describe()
```

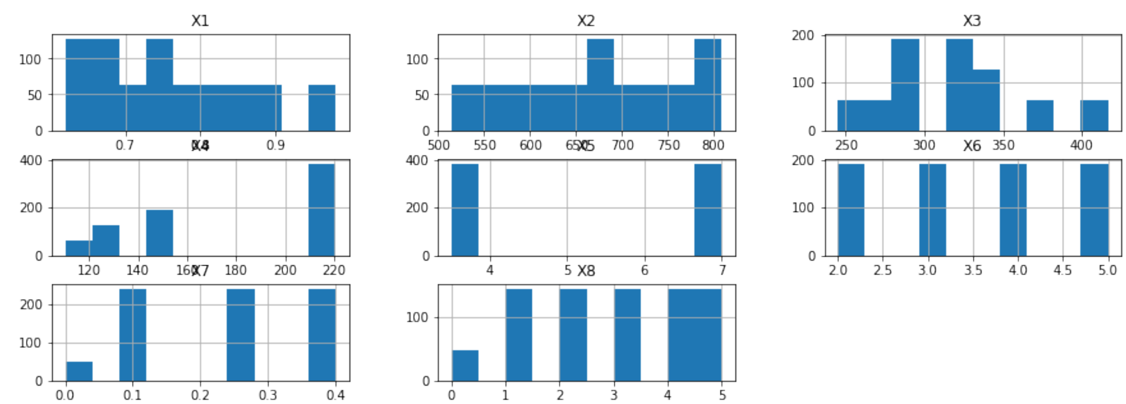
Out[23]:

	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
count	768.000000	768.00000
mean	0.234375	2.81250
std	0.133221	1.55096
min	0.000000	0.00000
25%	0.100000	1.75000
50%	0.250000	3.00000
75%	0.400000	4.00000
max	0.400000	5.00000

In [24]:
```python
# Get feature distribution of each continuous valued feature
data2.hist(figsize=(15,5))
plt.show()
```



In [25]:
```python
# Separate X from the Data Set.
X2=data2.iloc[:,:-1]
X2.head()
```

Out[25]:

	X1	X2	X3	X4	X5	X6	X7	X8
0	0.64	784.0	343.0	220.5	3.5	5	0.25	4
1	0.86	588.0	294.0	147.0	7.0	2	0.25	1
2	0.90	563.5	318.5	122.5	7.0	5	0.10	3
3	0.64	784.0	343.0	220.5	3.5	4	0.10	3
4	0.76	661.5	416.5	122.5	7.0	4	0.10	3

In [26]:
```python
# Get Labels from the Data Set.
Y2=data2['Y1']
Y2.head()
```

Out[26]:
```
0       Low
1    Medium
2    Medium
3       Low
4      High
Name: Y1, dtype: category
Categories (3, object): [Low < Medium < High]
```

In [27]:
```
# How many values I have in each Label?
Y2.value_counts()
```

Out[27]:
```
Low       377
High      200
Medium    191
Name: Y1, dtype: int64
```

In [28]:
```
# Maps Label to integers
    # - Low: 0
    # - High: 1
    # - Medium : 2
Y2=Y2.map({'Low': 0, 'High': 1,'Medium' :2})
print (Y2.value_counts())

# Show how is my Label array.
Y2.head()
```

```
0    377
1    200
2    191
Name: Y1, dtype: int64
```

Out[28]:
```
0    0
1    2
2    2
3    0
4    1
Name: Y1, dtype: int64
```

In [29]:
```
# Which are my shapes?
print("Feature vector shape=", X2.shape)
print("Class shape=", Y2.shape)
```

```
Feature vector shape= (768, 8)
Class shape= (768,)
```

Data Split

```
In [30]: # Split data into Training Set and Validation Set  using sklearn function.
         # Storage the different Data with the label "2", since we are in the same Part I.
         x_train2, x_test2, y_train2, y_test2 = train_test_split(X2, Y2, test_size=0.15, random_
         print ('Number of samples in training data:',len(x_train2))
         print ('Number of samples in validation data:',len(x_test2))
```

```
Number of samples in training data: 652
Number of samples in validation data: 116
```

Train Model

```
In [31]: # Name our Logistic Regression object
         LogisticRegressionModel = LogisticRegression()

         # Train the Model
         LogisticRegressionModel.fit(x_train2, y_train2)
```

```
Out[31]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Training Accuracy

```
In [32]: # Training Accuracy: Float in the variable: training_accuracy2
         training_accuracy2 = LogisticRegressionModel.score(x_train2,y_train2)
         print ('Training Accuracy:',training_accuracy2)
```

```
Training Accuracy: 0.88036809816
```

Prediction of Train

```
In [33]: # Prediction2 of Train.
         prediction_train2 = LogisticRegressionModel.predict(x_train2)
```

Find Error

```
In [34]: def find_error(real_label,predicted_label):
             """
             Find the error between Prediction and "Real Label".

             Arguments:
                 real_label -- label in data
                 predicted_label -- label predicted by the model

             """
```

```python
# Empty array of Zeros, with the length of "real_label"
Loss_Array = np.zeros(len(real_label))

for i,value in enumerate(real_label):

    if value == predicted_label[i]:
        Loss_Array[i] = 0
    else:
        Loss_Array[i] = 1

print ("Y-realLabel   Z-predictedLabel   Error \n")
for i,value in enumerate(real_label):
    print (value,"\t\t" ,predicted_label[i],"\t\t",Loss_Array[i])

error_rate = np.average(Loss_Array)
print ("\nThe error rate is ", error_rate)
print ('\nThe accuracy of the model is ',1-error_rate )
```

Error of Experiment

```
In [35]: find_error(y_train2, prediction_train2)
```

```
Y-realLabel   Z-predictedLabel   Error

2                  2                  0.0
0                  0                  0.0
2                  1                  1.0
1                  1                  0.0
2                  2                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
2                  2                  0.0
1                  1                  0.0
2                  2                  0.0
0                  0                  0.0
2                  2                  0.0
2                  2                  0.0
0                  0                  0.0
2                  2                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
```

2	1	1.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
2	1	1.0
2	2	0.0
2	2	0.0
1	2	1.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	0	1.0
0	0	0.0
0	0	0.0
2	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	2	1.0
1	1	0.0
2	1	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	2	1.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	2	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	2	1.0
0	0	0.0
2	1	1.0
0	0	0.0
1	1	0.0
1	2	1.0
1	2	1.0
0	0	0.0
1	1	0.0

1	1	0.0
2	0	1.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
2	2	0.0

2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	0	1.0
2	1	1.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	2	1.0
0	0	0.0
2	2	0.0
1	2	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	2	1.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	2	1.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	2	1.0
2	1	1.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	2	1.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
2	2	0.0
0	0	0.0

1	2	1.0
2	1	1.0
1	1	0.0
2	2	0.0
0	2	1.0
1	1	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0
2	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	2	1.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	2	1.0
0	0	0.0
0	0	0.0

2	2	0.0
1	2	1.0
1	2	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	2	1.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	2	1.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	1	1.0
2	2	0.0
1	1	0.0
2	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
2	1	1.0
0	0	0.0
0	0	0.0

0	0	0.0
2	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	2	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	2	1.0
0	0	0.0
2	2	0.0
0	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	0	1.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0

0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	1	1.0
2	2	0.0
2	2	0.0
0	0	0.0
1	2	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
1	1	0.0

1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	2	1.0
1	2	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	2	1.0
2	2	0.0
2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
2	1	1.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	0	1.0
2	2	0.0
1	2	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	2	1.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	2	1.0
0	0	0.0
2	2	0.0
2	1	1.0
1	1	0.0
1	1	0.0
1	2	1.0
1	1	0.0
0	0	0.0

2	2	0.0
2	0	1.0
2	2	0.0
1	1	0.0
2	2	0.0
2	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
2	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	0	1.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
1	2	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	2	1.0
2	0	1.0
2	1	1.0
0	0	0.0
2	2	0.0
2	1	1.0
0	0	0.0
1	1	0.0

1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	0	1.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
1	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0

```
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  2                  1.0

The error rate is  0.11963190184

The accuracy of the model is  0.88036809816
```

Validation Accuracy and Variance:

```
In [36]: # Validation Accuracy: Float in the variable: validation_accuracy2
         validation_accuracy2 = LogisticRegressionModel.score(x_test2,y_test2)
         print('Accuracy of the model on unseen validation data: ',validation_accuracy2)

         # Variance: Float. Difference between Training and Test.
         variance2 = training_accuracy2 - validation_accuracy2
         print("Variance: ", variance2)
```

```
Accuracy of the model on unseen validation data:  0.827586206897
Variance:  0.052781891263
```

Prediction of Test.

```
In [37]: # Prediction of Test.
         y_pred2 = LogisticRegressionModel.predict(x_test2)
```

Confusion Matrix

```
In [38]: from sklearn.metrics import confusion_matrix

         # Confusion Matrix. We're looking for the Diagonal.
         ConfusionMatrix = pd.DataFrame(confusion_matrix(y_test2, y_pred2),columns=['Predicted 0
         print ('Confusion matrix of test data is: \n',ConfusionMatrix)
```

```
Confusion matrix of test data is:
           Predicted 0  Predicted 1  Predicted 2
Actual 0           61            0            2
Actual 1            0           17            9
Actual 2            3            6           18
```

__ 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
more at: https://en.wikipedia.org/wiki/Feature_scaling

Pre-Processing with the Same Scale

In [39]:
```
from sklearn import preprocessing
print(x_train2.head())

df.rename(index=str, columns={"A": "a", "B": "c"})
x_scaled = pd.DataFrame(preprocessing.scale(X2)).rename(index=str, columns={ 0: "X1",
                                                                             1: "X2",
                                                                             2: "X3",
                                                                             3: "X4",
                                                                             4: "X5",
                                                                             5: "X6",
                                                                             6: "X7",
                                                                             7: "X8"})
print(x_scaled)
```

```
       X1     X2     X3     X4   X5  X6    X7  X8
458  0.86  588.0  294.0  147.0  7.0   5  0.10   2
635  0.62  808.5  367.5  220.5  3.5   3  0.40   3
457  0.82  612.5  318.5  147.0  7.0   4  0.40   3
674  0.79  637.0  343.0  147.0  7.0   4  0.25   1
277  0.86  588.0  294.0  147.0  7.0   3  0.25   3
            X1        X2        X3        X4   X5        X6        X7        X8
0    -1.174613  1.275625  0.561951  0.972512 -1.0  1.341641  0.117363  0.766154
1     0.906580 -0.950920 -0.561951 -0.655880  1.0 -1.341641  0.117363 -1.169393
2     1.284979 -1.229239  0.000000 -1.198678  1.0  1.341641 -1.009323  0.120972
3    -1.174613  1.275625  0.561951  0.972512 -1.0  0.447214 -1.009323  0.120972
4    -0.039417 -0.115966  2.247806 -1.198678  1.0  0.447214 -1.009323  0.120972
5    -0.985413  0.997307  0.000000  0.972512 -1.0  1.341641  0.117363 -1.169393
6    -1.363812  1.553943  1.123903  0.972512 -1.0 -0.447214  0.117363  0.766154
7    -0.039417 -0.115966  2.247806 -1.198678  1.0  1.341641 -1.009323  0.120972
8    -0.039417 -0.115966  2.247806 -1.198678  1.0 -1.341641  1.244049  0.120972
9    -1.174613  1.275625  0.561951  0.972512 -1.0 -1.341641 -1.009323  0.120972
10    0.528182 -0.672602  0.000000 -0.655880  1.0 -1.341641  0.117363  0.120972
11   -0.701614  0.718989 -0.561951  0.972512 -1.0 -1.341641 -1.760447 -1.814575
12   -1.363812  1.553943  1.123903  0.972512 -1.0  1.341641  1.244049  0.120972
13   -0.039417 -0.115966  2.247806 -1.198678  1.0 -0.447214  0.117363  0.120972
14   -0.228616  0.162352 -1.685854  0.972512 -1.0  0.447214 -1.760447 -1.814575
15    0.906580 -0.950920 -0.561951 -0.655880  1.0 -1.341641 -1.009323  1.411336
16   -0.701614  0.718989 -0.561951  0.972512 -1.0  0.447214  0.117363  0.120972
17   -1.363812  1.553943  1.123903  0.972512 -1.0  1.341641  1.244049 -0.524211
18   -0.039417 -0.115966  2.247806 -1.198678  1.0  0.447214  0.117363 -0.524211
19   -1.174613  1.275625  0.561951  0.972512 -1.0  0.447214  0.117363 -1.169393
```

```
20  -0.039417 -0.115966  2.247806 -1.198678  1.0 -1.341641  1.244049  1.411336
21  -1.363812  1.553943  1.123903  0.972512 -1.0 -1.341641  0.117363  0.120972
22  -1.363812  1.553943  1.123903  0.972512 -1.0 -1.341641 -1.009323  0.120972
23  -1.363812  1.553943  1.123903  0.972512 -1.0  0.447214  1.244049  0.120972
24  -1.174613  1.275625  0.561951  0.972512 -1.0  0.447214 -1.009323 -1.169393
25   0.528182 -0.672602  0.000000 -0.655880  1.0 -0.447214  0.117363  0.766154
26  -0.985413  0.997307  0.000000  0.972512 -1.0  1.341641  0.117363  0.766154
27   2.041777 -1.785875 -0.561951 -1.470077  1.0 -1.341641 -1.009323  0.766154
28  -0.228616  0.162352 -1.685854  0.972512 -1.0  1.341641  0.117363 -1.169393
29  -1.174613  1.275625  0.561951  0.972512 -1.0  0.447214 -1.009323 -0.524211
..        ...       ...       ...       ...  ...       ...       ...       ...
738  2.041777 -1.785875 -0.561951 -1.470077  1.0  1.341641  1.244049 -1.169393
739  0.528182 -0.672602  0.000000 -0.655880  1.0 -0.447214  1.244049  0.766154
740 -0.228616  0.162352 -1.685854  0.972512 -1.0  0.447214 -1.009323  1.411336
741  0.906580 -0.950920 -0.561951 -0.655880  1.0  1.341641 -1.760447 -1.814575
742  2.041777 -1.785875 -0.561951 -1.470077  1.0 -1.341641  0.117363  0.766154
743 -0.512415  0.440670 -1.123903  0.972512 -1.0  0.447214  1.244049  1.411336
744 -0.039417 -0.115966  2.247806 -1.198678  1.0  0.447214  1.244049  0.120972
745  1.284979 -1.229239  0.000000 -1.198678  1.0  1.341641  1.244049  0.120972
746 -1.363812  1.553943  1.123903  0.972512 -1.0 -0.447214 -1.009323  0.766154
747 -0.228616  0.162352 -1.685854  0.972512 -1.0 -1.341641  1.244049  0.766154
748  1.284979 -1.229239  0.000000 -1.198678  1.0  0.447214 -1.009323  0.766154
749  0.244383 -0.394284  0.561951 -0.655880  1.0  0.447214  1.244049  0.120972
750 -0.985413  0.997307  0.000000  0.972512 -1.0  1.341641  1.244049  1.411336
751  1.284979 -1.229239  0.000000 -1.198678  1.0 -0.447214 -1.009323  0.766154
752  0.244383 -0.394284  0.561951 -0.655880  1.0 -1.341641  1.244049 -1.169393
753 -0.039417 -0.115966  2.247806 -1.198678  1.0  1.341641  1.244049  0.120972
754 -0.985413  0.997307  0.000000  0.972512 -1.0 -0.447214  1.244049  0.120972
755  0.244383 -0.394284  0.561951 -0.655880  1.0  1.341641  0.117363 -1.169393
756 -0.512415  0.440670 -1.123903  0.972512 -1.0 -0.447214  1.244049  1.411336
757 -1.363812  1.553943  1.123903  0.972512 -1.0 -0.447214  1.244049 -1.169393
758  1.284979 -1.229239  0.000000 -1.198678  1.0 -0.447214 -1.009323  1.411336
759 -0.512415  0.440670 -1.123903  0.972512 -1.0 -0.447214  0.117363  0.766154
760  0.244383 -0.394284  0.561951 -0.655880  1.0 -0.447214  1.244049 -1.169393
761  0.244383 -0.394284  0.561951 -0.655880  1.0  0.447214 -1.760447 -1.814575
762  2.041777 -1.785875 -0.561951 -1.470077  1.0  1.341641 -1.009323  1.411336
763  2.041777 -1.785875 -0.561951 -1.470077  1.0 -1.341641 -1.009323 -0.524211
764 -0.039417 -0.115966  2.247806 -1.198678  1.0  1.341641 -1.760447 -1.814575
765  1.284979 -1.229239  0.000000 -1.198678  1.0  1.341641 -1.009323 -0.524211
766 -0.512415  0.440670 -1.123903  0.972512 -1.0 -0.447214  0.117363 -0.524211
767 -1.363812  1.553943  1.123903  0.972512 -1.0  0.447214 -1.009323 -1.169393

[768 rows x 8 columns]
```

In [40]:
```
# Which are my shapes?
print("Feature vector shape=", x_scaled.shape)
print("Class shape=", Y2.shape)
```

```
Feature vector shape= (768, 8)
Class shape= (768,)
```

Split Data

```
In [41]: # Split data into Training Set and Validation Set  using sklearn function.
         # Storage the different Data with the label "3", since we are in the same Part I.
         x_train3, x_test3, y_train3, y_test3 = train_test_split(x_scaled, Y2, test_size=0.15, r
         print ('Number of samples in training data:',len(x_train3))
         print ('Number of samples in validation data:',len(x_test3))
```

```
Number of samples in training data: 652
Number of samples in validation data: 116
```

Train Scaled Model

```
In [42]: # Name our Logistic Regression object. With the
         LogisticRegressionModel2 = LogisticRegression()

         # Train the Model
         LogisticRegressionModel2.fit(x_train3, y_train3)
```

```
Out[42]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Training Accuracy

```
In [43]: # Training Accuracy: Float in the variable: training_accuracy3
         training_accuracy3 = LogisticRegressionModel2.score(x_train3,y_train3)
         print ('Training Accuracy:',training_accuracy3)
```

```
Training Accuracy: 0.91717791411
```

Prediction in Train.

```
In [44]: # Prediction3 in Train.
         prediction_train3 = LogisticRegressionModel2.predict(x_train3)
```

Error of Experiment

```
In [45]: # Find error:
         find_error(y_train3, prediction_train3)
```

Y-realLabel	Z-predictedLabel	Error
2	2	0.0
0	0	0.0
2	1	1.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0

2	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	0	1.0
0	0	0.0
0	0	0.0
2	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	2	1.0
1	1	0.0
2	1	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	1	1.0
0	0	0.0
1	1	0.0
1	2	1.0
1	2	1.0
0	0	0.0
1	1	0.0
1	1	0.0
2	0	1.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	1	1.0

0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	0	1.0
2	1	1.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0

0	0	0.0
0	2	1.0
0	0	0.0
2	1	1.0
1	2	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	2	1.0
2	2	0.0
1	1	0.0

2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	2	1.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
2	1	1.0
1	1	0.0
2	2	0.0
0	2	1.0
1	1	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0

0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0
2	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	2	1.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	2	1.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	1	1.0
2	2	0.0
1	1	0.0
2	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	2	0.0
1	1	0.0

2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	2	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	2	1.0
0	0	0.0
2	2	0.0
0	2	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
2	0	1.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0

1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	2	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	2	1.0
2	2	0.0

2	2	0.0
0	0	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	2	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	1	1.0
0	0	0.0
2	2	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

2	0	1.0
2	2	0.0
1	2	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	2	1.0
0	0	0.0
2	2	0.0
2	1	1.0
1	1	0.0
1	1	0.0
1	2	1.0
1	1	0.0
0	0	0.0
2	2	0.0
2	0	1.0
2	2	0.0
1	1	0.0
2	2	0.0
2	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
2	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	0	1.0
2	2	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0

0	0	0.0
1	1	0.0
1	1	0.0
2	2	0.0
1	2	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
2	2	0.0
2	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	0	1.0
2	1	1.0
0	0	0.0
2	2	0.0
2	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
2	2	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0
2	2	0.0
1	1	0.0
2	0	1.0
0	0	0.0
0	0	0.0
2	2	0.0
2	2	0.0
2	2	0.0
2	2	0.0
1	1	0.0

```
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
1               1               0.0
0               0               0.0
2               2               0.0
1               1               0.0
0               0               0.0
0               0               0.0
0               0               0.0
2               2               0.0
1               1               0.0
1               1               0.0
0               0               0.0
0               0               0.0
1               1               0.0
0               0               0.0
1               1               0.0
0               0               0.0
1               1               0.0
0               0               0.0
1               1               0.0
2               2               0.0
1               1               0.0
0               0               0.0
0               0               0.0
1               1               0.0

The error rate is  0.0828220858896

The accuracy of the model is  0.91717791411
```

Validation Accuracy and Variance:

In [46]:
```python
# Validation Accuracy: Float in the variable: validation_accuracy3
validation_accuracy3 = LogisticRegressionModel2.score(x_test3,y_test3)
print('Accuracy of the model on unseen validation data: ',validation_accuracy3)

# Variance: Float. Difference between Training and Test.
variance3 = training_accuracy3 - validation_accuracy3
print("Variance: ", variance3)
```

```
Accuracy of the model on unseen validation data:  0.870689655172
Variance:  0.046488258938
```

Prediction in Test.

```
In [47]: # Prediction3 in Test.
         y_pred3 = LogisticRegressionModel2.predict(x_test3)
```

Confusion Matrix

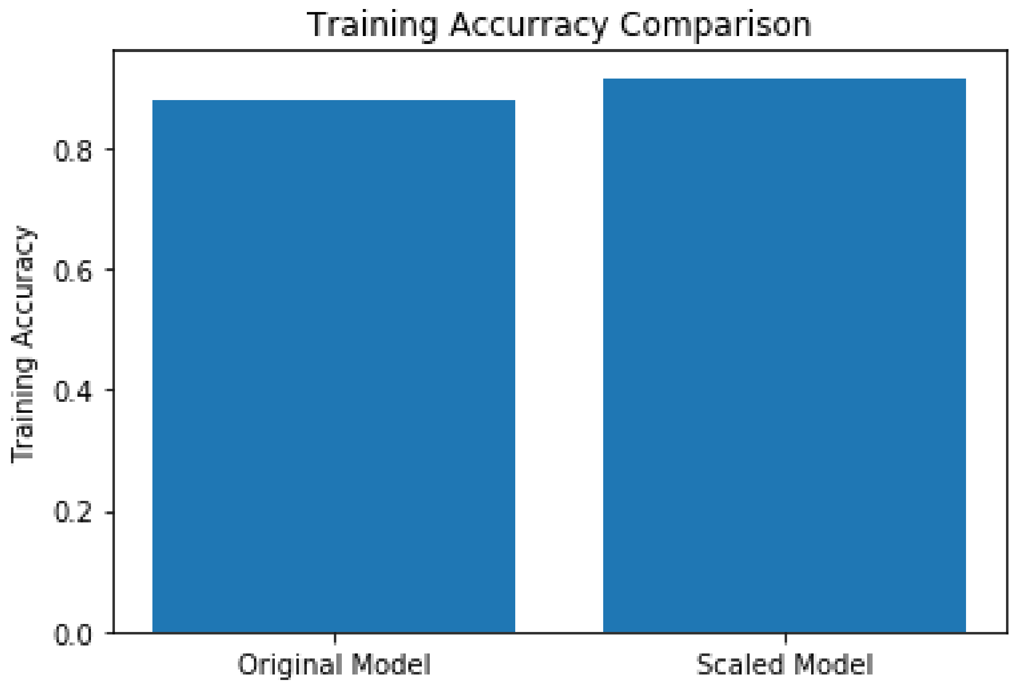
```
In [48]: # Confusion Matrix. We're looking for the Diagonal.
         ConfusionMatrix2 = pd.DataFrame(confusion_matrix(y_test3, y_pred3),columns=['Predicted
         print ('Confusion matrix of test data is: \n',ConfusionMatrix2)
```

```
Confusion matrix of test data is:
           Predicted 0  Predicted 1  Predicted 2
Actual 0           61            0            2
Actual 1            0           20            6
Actual 2            3            4           20
```

Comparison Two Models

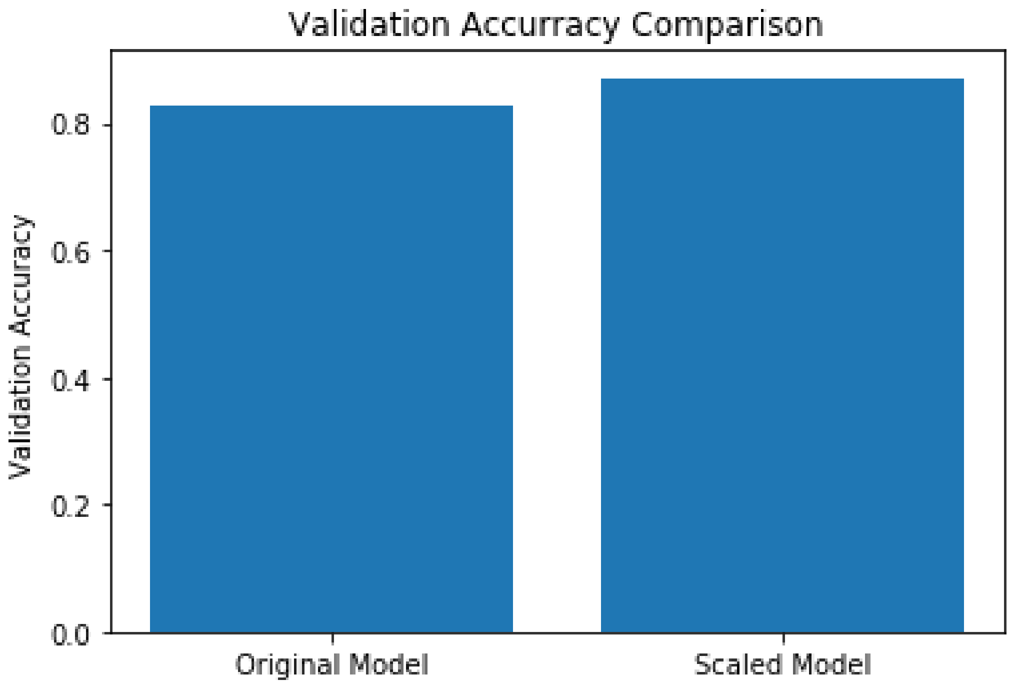
Training Accurracy Comparison

```
In [49]: # Training Accurracy Comparison
         experiments_t = ("Original Model", "Scaled Model")
         accuracies_t = [training_accuracy2, training_accuracy3]

         plt.bar(np.arange(len(experiments_t)), accuracies_t, align='center', alpha=1)
         plt.xticks(np.arange(len(experiments_t)), experiments_t)
         plt.ylabel('Training Accuracy')
         plt.title('Training Accurracy Comparison')

         plt.show()
```



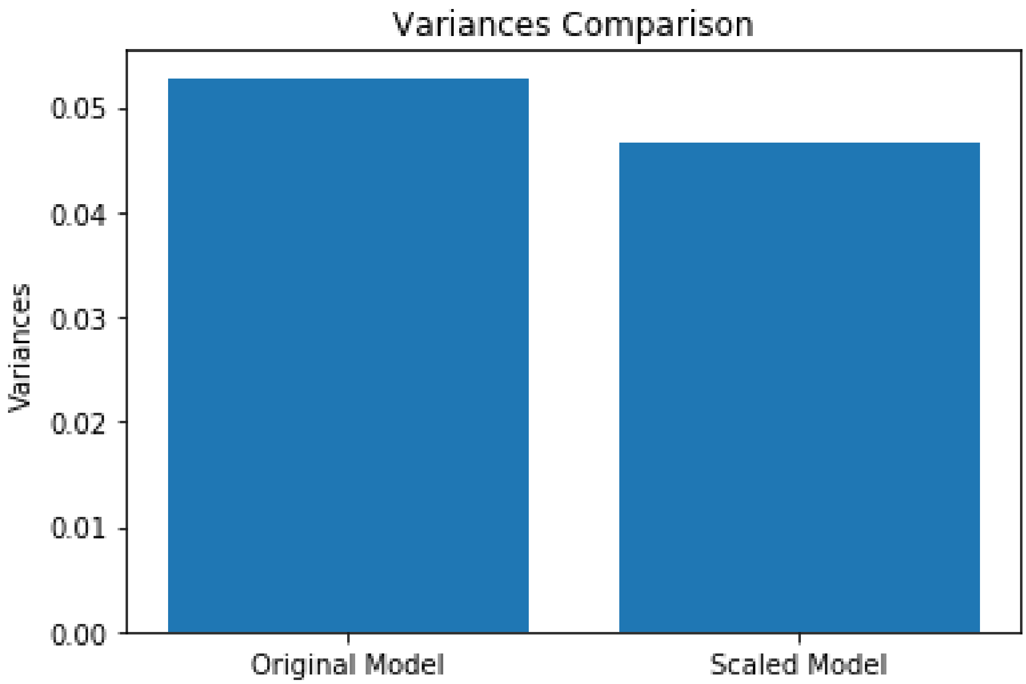
Validation Accurracy Comparison

```
In [50]: # Validation Accurracy Comparison
         experiments_v = ("Original Model", "Scaled Model")
         accuracies_v = [validation_accuracy2, validation_accuracy3]

         plt.bar(np.arange(len(experiments_v)), accuracies_v, align='center', alpha=1)
         plt.xticks(np.arange(len(experiments_v)), experiments_v)
         plt.ylabel('Validation Accuracy')
         plt.title('Validation Accurracy Comparison')

         plt.show()
```



Variances Comparison

```
In [51]: # Variances Comparison
         experiments = ("Original Model", "Scaled Model")
         variances = [variance2, variance3]

         plt.bar(np.arange(len(experiments)), variances, align='center', alpha=1)
         plt.xticks(np.arange(len(experiments)), experiments)
         plt.ylabel('Variances')
         plt.title('Variances Comparison')

         plt.show()
```



1.2 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)

Read Data

```
In [52]: # Reading FileD
         df = pd.read_csv('diabetesdata.csv')
```

Analisis of the Data Features

```
In [53]: # Describing data (General)
         print("Describing Data in a general view...")
         df.describe()
```

```
Describing Data in a general view...
```

Out[53]:

```
       TimesPregnant  glucoseLevel          BP     insulin         BMI  \
count     768.000000    734.000000  768.000000  768.000000  768.000000
mean        3.845052    121.016349   69.105469   79.799479   31.992578
std         3.369578     31.660240   19.355807  115.244002    7.884160
min         0.000000      0.000000    0.000000    0.000000    0.000000
25%         1.000000     99.000000   62.000000    0.000000   27.300000
50%         3.000000    117.000000   72.000000   30.500000   32.000000
75%         6.000000    141.000000   80.000000  127.250000   36.600000
max        17.000000    199.000000  122.000000  846.000000   67.100000

         Pedigree         Age  IsDiabetic
count  768.000000  735.000000  768.000000
mean     0.471876   33.353741    0.348958
std      0.331329   11.772944    0.476951
min      0.078000   21.000000    0.000000
25%      0.243750   24.000000    0.000000
50%      0.372500   29.000000    0.000000
75%      0.626250   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

In [54]:

```
# Describe data features in terms of type, distribution range and mean values.
for i in df.columns:
    nice_display(i, df[i].dtype, df[i].max(), df[i].min(), df[i].mean())
```

```
The feature TimesPregnant:
Type: Integer so is Continuous!
Max:  17
Min:  0
Mean 3.8450520833333335
------------------------------
The feature glucoseLevel:
Type: Float so is Continuous!
Max:  199.0
Min:  0.0
Mean 121.01634877384195
------------------------------
The feature BP:
Type: Integer so is Continuous!
Max:  122
Min:  0
Mean 69.10546875
------------------------------
The feature insulin:
Type: Integer so is Continuous!
Max:  846
```

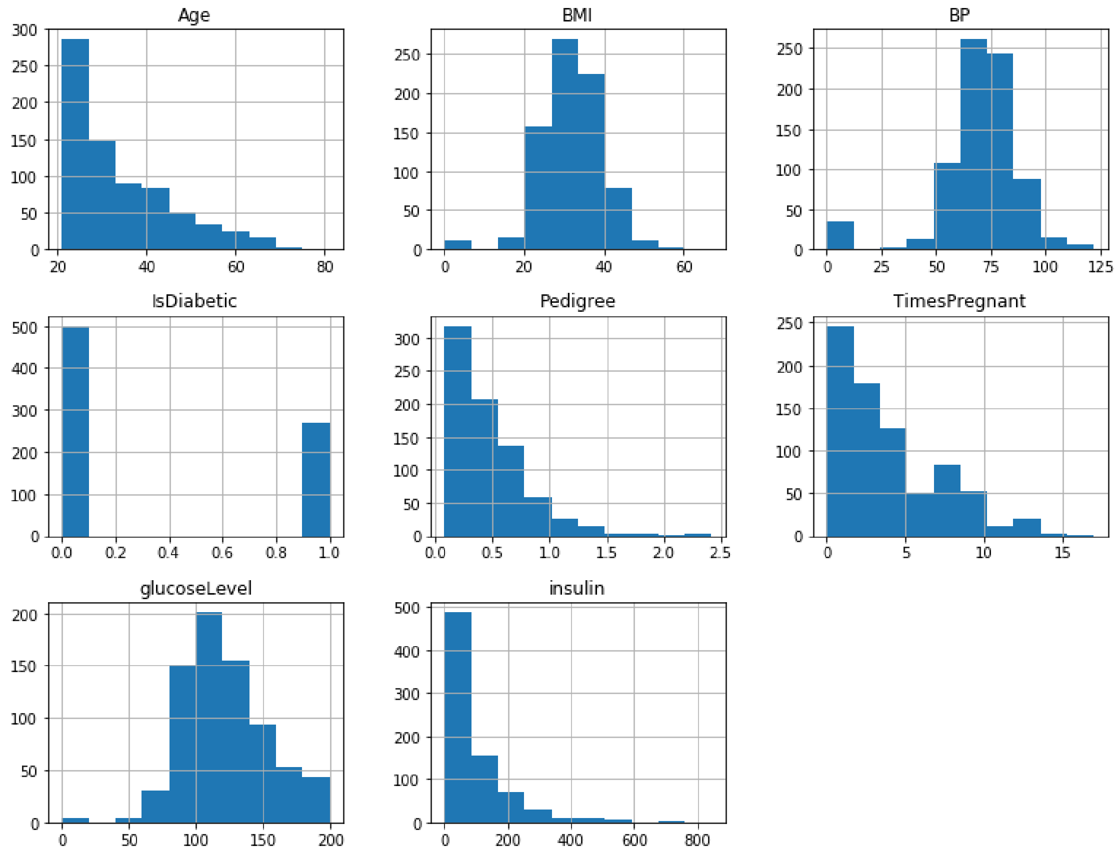
```
Min:  0
Mean 79.79947916666667
------------------------------
The feature BMI:
Type: Float so is Continuous!
Max:  67.1
Min:  0.0
Mean 31.992578124999977
------------------------------
The feature Pedigree:
Type: Float so is Continuous!
Max:  2.42
Min:  0.078
Mean 0.4718763020833327
------------------------------
The feature Age:
Type: Float so is Continuous!
Max:  81.0
Min:  21.0
Mean 33.35374149659864
------------------------------
The feature IsDiabetic:
Type: Integer so is Continuous!
Max:  1
Min:  0
Mean 0.3489583333333333
------------------------------
```

```
In [55]: # Adittional INFO
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 8 columns):
TimesPregnant    768 non-null int64
glucoseLevel     734 non-null float64
BP               768 non-null int64
insulin          768 non-null int64
BMI              768 non-null float64
Pedigree         768 non-null float64
Age              735 non-null float64
IsDiabetic       768 non-null int64
dtypes: float64(4), int64(4)
memory usage: 48.1 KB
```

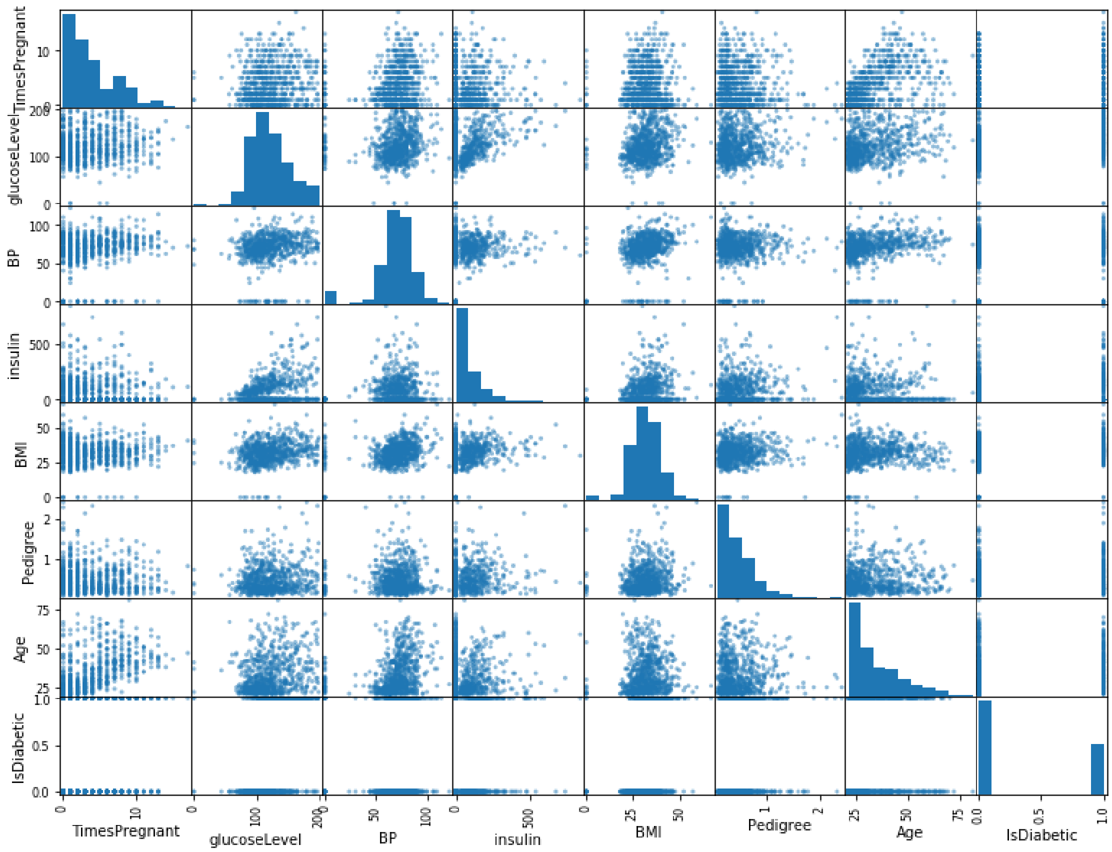
```
In [56]: # Distributions for each feature
```

```
df.hist(figsize=(13,10))
plt.show()
```




```
In [57]: # Relations between features
         pd.tools.plotting.scatter_matrix(df,figsize=(13,10));
```

```
/home/dasapunar/anaconda3/envs/data-x/lib/python3.6/site-packages/ipykernel_launcher.py:2: Futur
```




```
In [58]: # SHUFFLE data.
         data = shuffle(df).reset_index(drop=True)
```

```
In [59]: # Balanced data set?
         data['IsDiabetic'].value_counts()
```

```
Out[59]: 0    500
         1    268
         Name: IsDiabetic, dtype: int64
```

```
In [60]:  # Bayes Error for prediction accuracy
         _[0]/(sum(_))
```

```
Out[60]: 0.6510416666666663
```

Check NaN Values

```
In [61]: # Get NaNs
         print('Number of NaNs in the dataframe:\n',data.isnull().sum())
         data.head()
```

```
Number of NaNs in the dataframe:
 TimesPregnant     0
glucoseLevel     34
BP                0
insulin           0
BMI               0
Pedigree          0
Age              33
IsDiabetic        0
dtype: int64
```

Out[61]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	9	156.0	86	155	34.3	1.189	42.0	1
1	0	95.0	85	36	37.4	0.247	24.0	1
2	9	106.0	52	0	31.2	0.380	42.0	0
3	3	111.0	62	0	22.6	0.142	21.0	0
4	6	115.0	60	0	33.7	0.245	40.0	1

__ 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.__

Preprocess data to replace NaN values in a feature(if any) using mean of the feature.

In [62]:
```
data["glucoseLevel"]= data["glucoseLevel"].fillna(data["glucoseLevel"].mean())
data["Age"]= data["Age"].fillna(data["Age"].mean())
```

In [63]:
```
# Get NaNs again.
print('Number of NaNs in the dataframe:\n',data.isnull().sum())
```

```
Number of NaNs in the dataframe:
 TimesPregnant    0
glucoseLevel     0
BP               0
insulin          0
BMI              0
Pedigree         0
Age              0
IsDiabetic       0
dtype: int64
```

Split Data

In [64]:
```
# Separate X from the Data Set.
X=data.iloc[:,:-1]
X.head()
```

```
Out[64]:    TimesPregnant  glucoseLevel  BP  insulin   BMI  Pedigree   Age
         0              9         156.0  86      155  34.3     1.189  42.0
         1              0          95.0  85       36  37.4     0.247  24.0
         2              9         106.0  52        0  31.2     0.380  42.0
         3              3         111.0  62        0  22.6     0.142  21.0
         4              6         115.0  60        0  33.7     0.245  40.0
```

```
In [65]: # Get Labels from the Data Set.
         Y=data['IsDiabetic']
         Y.head()
```

```
Out[65]: 0    1
         1    1
         2    0
         3    0
         4    1
         Name: IsDiabetic, dtype: int64
```

```
In [66]: # Which are my shapes?
         print("Feature vector shape=", X.shape)
         print("Class shape=", Y.shape)
```

```
Feature vector shape= (768, 7)
Class shape= (768,)
```

```
In [67]: # Split Data
         x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1
         print ('Number of samples in training data:',len(x_train))
         print ('Number of samples in validation data:',len(x_test))
```

```
Number of samples in training data: 614
Number of samples in validation data: 154
```

Train Different Models

```
In [68]: def model_flow(Model, x_train, x_test, y_train, y_test):
             """
             Print all the flow of a model: Fit, Training Accuracy, Error by Example, Validation

             Arguments:
                 x_train -- Pandas Dataframe. Features Training Set.
                 x_test -- Pandas Dataframe. Features Test Set.
                 y_train -- Pandas Dataframe. Label Train Set.
                 y_test -- Pandas Dataframe. Label Test Set.

             """
```

```python
    # Train the Model
    Model.fit(x_train, y_train)

    # Training Accuracy
    training_accuracy = Model.score(x_train, y_train)
    print ('Training Accuracy:',training_accuracy)

    # Prediction of Train.
    prediction_train = Model.predict(x_train)

    # Find Error
    find_error(y_train, prediction_train)

    # Validation Accuracy
    validation_accuracy = Model.score(x_test,y_test)
    print('Accuracy of the model on unseen validation data: ',validation_accuracy)

    # Prediction of Test.
    y_pred = Model.predict(x_test)

    # Variance: Float. Difference between Training and Test.
    variance = training_accuracy - validation_accuracy
    print("Variance: ", variance)


def different_models(model_name, x_train, x_test, y_train, y_test):
    """
    Differentiate models and call model_flow function

    Arguments:
        model_name -- Python String. Corresponding to the model name.
        x_train -- Pandas Dataframe. Features Training Set.
        x_test -- Pandas Dataframe. Features Test Set.
        y_train -- Pandas Dataframe. Label Train Set.
        y_test -- Pandas Dataframe. Label Test Set.

    """
    if model_name == "LogisticRegression":
        # Model Object
        model = LogisticRegression()
        model_flow(model, x_train, x_test, y_train, y_test)

    elif model_name == "SVM":
        # Model Object
        model = SVC()
        model_flow(model, x_train, x_test, y_train, y_test)

    elif model_name == "Perceptron":
```

```
            # Model Object
            model = Perceptron()
            model_flow(model, x_train, x_test, y_train, y_test)

        elif model_name == "kNN":
            # Model Object
            model = KNeighborsClassifier(n_neighbors = 2)
            model_flow(model, x_train, x_test, y_train, y_test)

        elif model_name == "xgboost":
            # Model Object
            model = xgb.XGBClassifier(n_estimators=1000)
            model_flow(model, x_train, x_test, y_train, y_test)

        elif model_name == "random forest":
            # Model Object
            model = RandomForestClassifier(n_estimators=1000)
            model_flow(model, x_train, x_test, y_train, y_test)
```

```
In [69]: different_models("LogisticRegression", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 0.775244299674
Y-realLabel   Z-predictedLabel   Error

1                  1                 0.0
0                  0                 0.0
0                  0                 0.0
0                  1                 1.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
1                  0                 1.0
0                  0                 0.0
0                  1                 1.0
1                  0                 1.0
0                  0                 0.0
0                  0                 0.0
0                  1                 1.0
0                  0                 0.0
0                  0                 0.0
```

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0

1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0

0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	1	1.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0

0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
1	1	0.0
0	1	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0

0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	1	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	1	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

```
0               0               0.0
1               1               0.0
0               0               0.0
1               1               0.0
0               1               1.0
1               0               1.0
0               0               0.0
0               0               0.0
0               0               0.0
1               0               1.0
0               0               0.0
1               1               0.0
1               1               0.0
1               0               1.0

The error rate is  0.224755700326

The accuracy of the model is  0.775244299674
Accuracy of the model on unseen validation data:  0.753246753247
Variance:  0.0219975464275
```

```
In [70]: different_models("SVM", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 1.0
Y-realLabel   Z-predictedLabel   Error

1               1               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
1               1               0.0
0               0               0.0
0               0               0.0
1               1               0.0
0               0               0.0
0               0               0.0
```

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0

0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0

```
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
1                  1                  0.0

The error rate is  0.0

The accuracy of the model is  1.0
Accuracy of the model on unseen validation data:  0.62987012987
Variance:  0.37012987013
```

```
In [71]: different_models("Perceptron", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 0.667752442997
Y-realLabel   Z-predictedLabel   Error

1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  1                  1.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  0                  1.0
0                  0                  0.0
0                  0                  0.0
```

```
1                   0                   1.0
0                   0                   0.0
0                   0                   0.0
0                   0                   0.0
0                   0                   0.0
0                   0                   0.0
0                   0                   0.0
0                   1                   1.0
1                   0                   1.0
0                   0                   0.0
0                   0                   0.0
1                   0                   1.0
1                   0                   1.0
0                   0                   0.0
1                   0                   1.0
0                   0                   0.0
0                   0                   0.0
```

```
/home/dasapunar/anaconda3/envs/data-x/lib/python3.6/site-packages/sklearn/linear_model/stochasti
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
0                   0                   0.0
1                   0                   1.0
0                   0                   0.0
1                   0                   1.0
0                   0                   0.0
1                   0                   1.0
1                   1                   0.0
1                   0                   1.0
1                   1                   0.0
0                   0                   0.0
0                   0                   0.0
0                   0                   0.0
1                   1                   0.0
0                   0                   0.0
1                   0                   1.0
1                   0                   1.0
1                   0                   1.0
1                   0                   1.0
1                   0                   1.0
0                   0                   0.0
1                   0                   1.0
0                   0                   0.0
1                   0                   1.0
0                   0                   0.0
0                   0                   0.0
```

0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	1	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0

1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0

0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	1	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0

1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0

0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0

0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

```
0                 1                 1.0
1                 0                 1.0
1                 0                 1.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
1                 0                 1.0
0                 1                 1.0
1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
1                 0                 1.0
1                 1                 0.0
1                 0                 1.0

The error rate is  0.332247557003

The accuracy of the model is  0.667752442997
Accuracy of the model on unseen validation data:  0.616883116883
Variance:  0.0508693261136
```

```
In [72]: different_models("kNN", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 0.832247557003
Y-realLabel   Z-predictedLabel   Error

1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
```

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0

1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0

```
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
1                 0                 1.0
1                 1                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
1                 0                 1.0
0                 0                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
1                 1                 0.0
0                 0                 0.0
1                 1                 0.0
1                 1                 0.0
1                 0                 1.0

The error rate is  0.167752442997

The accuracy of the model is  0.832247557003
Accuracy of the model on unseen validation data:  0.720779220779
Variance:  0.111468336224
```

In [73]: different_models("xgboost", x_train, x_test, y_train, y_test)

```
Training Accuracy: 1.0
Y-realLabel   Z-predictedLabel   Error

1                 1                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
0                 0                 0.0
```

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0

1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0

1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

```
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
1                  1                  0.0

The error rate is  0.0

The accuracy of the model is  1.0
Accuracy of the model on unseen validation data:  0.714285714286
Variance:  0.285714285714
```

```
In [74]: different_models("random forest", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 1.0
Y-realLabel   Z-predictedLabel   Error

1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
```

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0

1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0

1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0

1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

```
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
1                  1                  0.0

The error rate is  0.0

The accuracy of the model is  1.0
Accuracy of the model on unseen validation data:  0.798701298701
Variance:  0.201298701299
```

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

Cut features: BP, insulin, BMI & Pedigree

In [75]:
```
data["BP"] = pd.cut(data["BP"], bins=3, labels=[0, 1, 2])
data["insulin"] = pd.cut(data["insulin"], bins=3, labels=[0, 1, 2])
data["BMI"] = pd.cut(data["BMI"], bins=3, labels=[0, 1, 2])
data["Pedigree"] = pd.cut(data["Pedigree"], bins=3, labels=[0, 1, 2])
data.head()
```

Out[75]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	9	156.0	2	0	1	1	42.0	1
1	0	95.0	2	0	1	0	24.0	1
2	9	106.0	1	0	1	0	42.0	0
3	3	111.0	1	0	1	0	21.0	0
4	6	115.0	1	0	1	0	40.0	1

In [76]:
```
# BP Ratio
data[["BP", "IsDiabetic"]].groupby(["BP"]).mean()
```

Out[76]:

BP	IsDiabetic
0	0.450000
1	0.307282
2	0.466667

In [77]:
```
# Insulin Ratio
data[["insulin", "IsDiabetic"]].groupby(["insulin"]).mean()
```

Out[77]:

insulin	IsDiabetic
0	0.337017
1	0.538462
2	0.600000

In [78]:
```
# BMI Ratio
data[["BMI", "IsDiabetic"]].groupby(["BMI"]).mean()
```

Out[78]:

BMI	IsDiabetic
0	0.039216
1	0.358297
2	0.611111

In [79]:
```
# Pedigree Ratio
data[["Pedigree", "IsDiabetic"]].groupby(["Pedigree"]).mean()
```

Out[79]:

Pedigree	IsDiabetic
0	0.327007
1	0.540541
2	0.444444

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.

Create Guess Matrix for Age and glucoseLevel

```
In [96]: # Get NaNs.
         print('Number of NaNs in the dataframe:\n',df.isnull().sum())
```

```
Number of NaNs in the dataframe:
 TimesPregnant     0
glucoseLevel     34
BP                0
insulin           0
BMI               0
Pedigree          0
Age              33
IsDiabetic        0
dtype: int64
```

```
In [97]: df["BP"] = pd.cut(df["BP"], bins=3, labels=[0, 1, 2])
         df["insulin"] = pd.cut(df["insulin"], bins=3, labels=[0, 1, 2])
         df["BMI"] = pd.cut(df["BMI"], bins=3, labels=[0, 1, 2])
         df["Pedigree"] = pd.cut(df["Pedigree"], bins=3, labels=[0, 1, 2])
         df.head()
```

Out[97]:

	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	0	1	0	21.0	0
4	0	137.0	0	0	1	2	33.0	1

```
In [98]: guess_age = np.zeros((3,3),dtype=int) # Initialize Matrix
         guess_age
```

```
Out[98]: array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]])
```

```
In [99]: guess_glucose_level = np.zeros((3,3),dtype=int) # Initialize Matrix
         guess_glucose_level
```

```
Out[99]: array([[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]])
```

```
In [100]: for i in range(0, 3):
              for j in range(0, 3):
                  aux_age = df[(df['BMI'] == i) & (df['BP'] == j)]['Age'].dropna().median()
                  guess_age[i,j] = int(aux_age)

                  aux_glucose_level = df[(df['BP'] == i) & (df['Pedigree'] == j)]['glucoseLevel'
                  guess_glucose_level[i,j] = int(aux_glucose_level)
```

```
In [101]: # Guess Age Matrix
          guess_age
```

```
Out[101]: array([[24, 25, 55],
                 [29, 29, 37],
                 [33, 32, 31]])
```

```
In [102]: # Guess Glucose Level Matrix
          guess_glucose_level
```

```
Out[102]: array([[115, 127, 137],
                 [112, 115, 149],
                 [133, 129, 159]])
```

```
In [104]: # Replace NaN Values of Age with the Guess Age Matrix
          for i in range(0, 3):
                  for j in range(0, 3):
                      df.loc[ (df["Age"].isnull()) & (df['BMI'] == i)& (df['BP'] == j),'Age'] =


          df['Age'] = df['Age'].astype(int)
```

```
In [106]: # Replace NaN Values of Age with the Guess Age Matrix
          for i in range(0, 3):
                  for j in range(0, 3):
                      df.loc[ (df["glucoseLevel"].isnull()) & (df['BP'] == i) & \
                             (df['Pedigree'] == j),'glucoseLevel'] = guess_age[i,j]


          df['glucoseLevel'] = df['glucoseLevel'].astype(int)
```

```
In [107]: # Get NaNs, to probe my proccedure above
          print('Number of NaNs in the dataframe:\n',df.isnull().sum())
```

```
Number of NaNs in the dataframe:
 TimesPregnant    0
glucoseLevel     0
BP               0
insulin          0
BMI              0
Pedigree         0
Age              0
IsDiabetic       0
dtype: int64
```

```
In [108]: df.head()
```

Out[108]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	148	1	0	1	0	50	1
1	1	29	1	0	1	0	31	0
2	8	183	1	0	1	0	29	1
3	1	29	1	0	1	0	21	0
4	0	137	0	0	1	2	33	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.

Pre-Processing: Make glucoseLevel and Age categorical Variables

```
In [111]: df["Age"] = pd.cut(df["Age"], bins=5, labels=[0, 1, 2, 3, 4])
          df["glucoseLevel"] = pd.cut(df["glucoseLevel"], bins=5, labels=[0, 1, 2, 3, 4])
          df.head()
```

Out[111]:

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

Split Data

```
In [114]: # Separate X from the Data Set.
          X=df.iloc[:,:-1]
          X.head()
```

```
Out[114]:    TimesPregnant glucoseLevel BP insulin BMI Pedigree Age
          0              6            3  1       0   1        0   2
          1              1            0  1       0   1        0   0
          2              8            4  1       0   1        0   0
          3              1            0  1       0   1        0   0
          4              0            3  0       0   1        2   0
```

```
In [115]: # Get Labels from the Data Set.
          Y=df['IsDiabetic']
          Y.head()
```

```
Out[115]: 0    1
          1    0
          2    1
          3    0
          4    1
          Name: IsDiabetic, dtype: int64
```

```
In [116]: x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=
          print ('Number of samples in training data:',len(x_train))
          print ('Number of samples in validation data:',len(x_test))
```

```
Number of samples in training data: 614
Number of samples in validation data: 154
```

Perceptron

```
In [117]: different_models("Perceptron", x_train, x_test, y_train, y_test)
```

```
/home/dasapunar/anaconda3/envs/data-x/lib/python3.6/site-packages/sklearn/linear_model/stochasti
  "and default tol will be 1e-3." % type(self), FutureWarning)


Training Accuracy: 0.662866449511
Y-realLabel   Z-predictedLabel   Error

0                  1                 1.0
0                  0                 0.0
0                  0                 0.0
1                  0                 1.0
1                  0                 1.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
1                  0                 1.0
0                  0                 0.0
0                  0                 0.0
0                  0                 0.0
```

1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0

1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0

1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0

0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	1	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0

0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	1	1.0
1	0	1.0
1	0	1.0
1	0	1.0
0	1	1.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0

```
0                  0                  0.0
1                  0                  1.0
1                  0                  1.0
1                  0                  1.0
0                  0                  0.0
0                  0                  0.0
1                  0                  1.0
1                  0                  1.0
1                  0                  1.0
0                  0                  0.0
1                  0                  1.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  1                  1.0
1                  0                  1.0
1                  0                  1.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0

The error rate is  0.337133550489

The accuracy of the model is  0.662866449511
Accuracy of the model on unseen validation data:  0.688311688312
Variance:  -0.0254452388003
```

Logistic Regression

```
In [118]: different_models("LogisticRegression", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 0.758957654723
Y-realLabel   Z-predictedLabel   Error

0                  1                  1.0
0                  0                  0.0
0                  0                  0.0
1                  0                  1.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
```

0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
0	1	1.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	1	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	0	1.0
0	0	0.0
0	1	1.0
1	0	1.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	1	1.0
1	0	1.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	1	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0

1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0
1	0	1.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
1	0	1.0
0	1	1.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0

1	0	1.0
1	0	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0
1	1	0.0
1	0	1.0
0	1	1.0
0	0	0.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0

```
1               1               0.0
1               1               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
1               1               0.0
1               1               0.0
1               1               0.0
0               0               0.0
0               0               0.0
1               1               0.0
1               0               1.0
1               0               1.0
0               0               0.0
1               0               1.0
0               0               0.0
0               0               0.0
0               0               0.0
0               0               0.0
0               1               1.0
1               0               1.0
1               0               1.0
0               0               0.0
0               0               0.0
0               0               0.0
0               1               1.0
1               1               0.0
0               0               0.0
0               0               0.0
0               0               0.0

The error rate is  0.241042345277

The accuracy of the model is  0.758957654723
Accuracy of the model on unseen validation data:  0.694805194805
Variance:  0.0641524599179
```

Random Forest

```
In [119]: different_models("random forest", x_train, x_test, y_train, y_test)
```

```
Training Accuracy: 0.907166123779
Y-realLabel   Z-predictedLabel   Error

0               0               0.0
0               0               0.0
```

0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	0	1.0

0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0

1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	1	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	0	1.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	1	1.0
1	0	1.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0

0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
1	0	1.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	0	1.0

1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	0	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0

0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	1	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0

1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	1	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0

0	0	0.0
0	1	1.0
0	0	0.0
1	0	1.0
0	0	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	0	1.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
0	0	0.0
0	0	0.0
0	0	0.0
1	1	0.0
1	1	0.0
1	1	0.0
1	1	0.0
0	0	0.0
0	0	0.0

```
0                  0                  0.0
0                  1                  1.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0
1                  1                  0.0
0                  0                  0.0
0                  0                  0.0
0                  0                  0.0

The error rate is  0.0928338762215

The accuracy of the model is  0.907166123779
Accuracy of the model on unseen validation data:  0.642857142857
Variance:  0.264308980921
```

1.2.1 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 Re-**