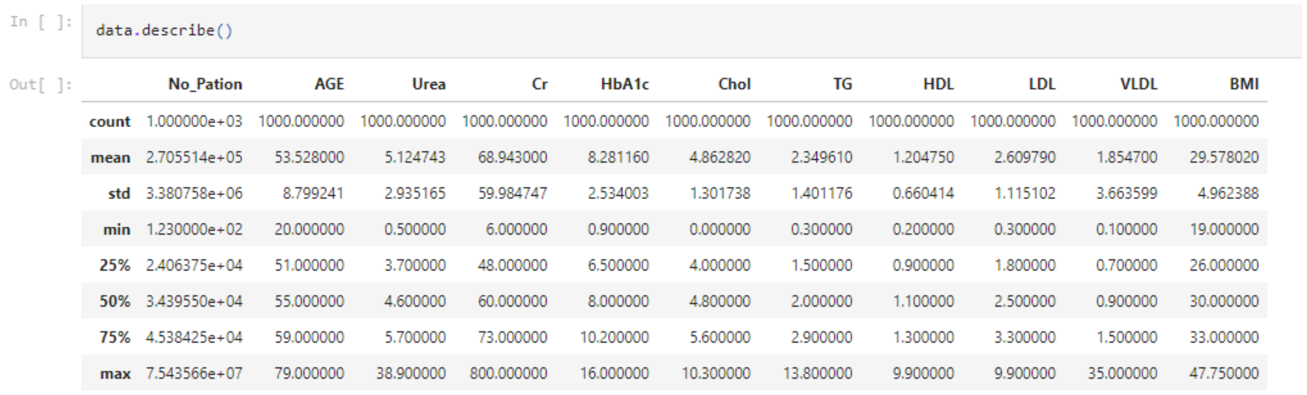
Diabetes Prediction

Team members:

- Mohammed Mohammed Alaa
- Ahmed Abdelhamed Mohamed
- Eslam Ayman Labib
- Osama Usry Mohamed
- Raphael Serwanis Adib

Understanding the data:

The data were imported in pandas Dataframe. We used Dataframe.describe method to get summary statistical analysis so we can understand the data [fig 1].



```
In [ ]: data.describe()
```

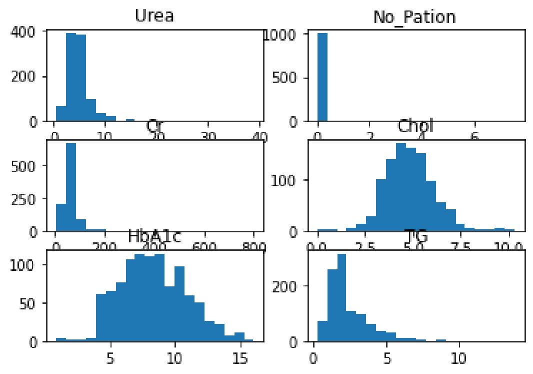
Out[ ]:

	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI
count	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	2.705514e+05	53.528000	5.124743	68.943000	8.281160	4.862820	2.349610	1.204750	2.609790	1.854700	29.578020
std	3.380758e+06	8.799241	2.935165	59.984747	2.534003	1.301738	1.401176	0.660414	1.115102	3.663599	4.962388
min	1.230000e+02	20.000000	0.500000	6.000000	0.900000	0.000000	0.300000	0.200000	0.300000	0.100000	19.000000
25%	2.406375e+04	51.000000	3.700000	48.000000	6.500000	4.000000	1.500000	0.900000	1.800000	0.700000	26.000000
50%	3.439550e+04	55.000000	4.600000	60.000000	8.000000	4.800000	2.000000	1.100000	2.500000	0.900000	30.000000
75%	4.538425e+04	59.000000	5.700000	73.000000	10.200000	5.600000	2.900000	1.300000	3.300000	1.500000	33.000000
max	7.543566e+07	79.000000	38.900000	800.000000	16.000000	10.300000	13.800000	9.900000	9.900000	35.000000	47.750000

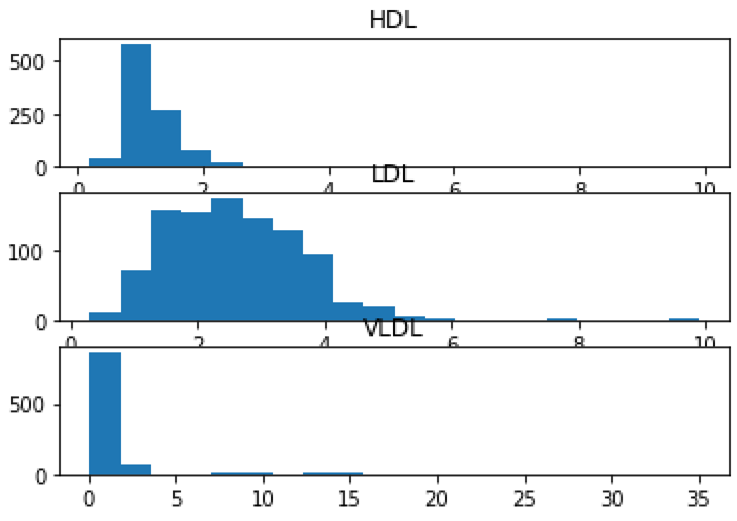
We concluded that the data have potential outliers as several features have a maximum value that is so far from the third quantile. We decided to make three different types of data for the models; one with these outliers, one where we deleted their rows by using the empirical rule and last one, we replaced their value with the mean.

The empirical rules states that about 99.7% fall within 3 standard deviations of the mean. In order to use this rule, features have to follow the normal distribution. So, we checked for the skew of each feature. Most features have visible skew.

```
In [ ]: figure,axis = pt.subplots(3,2)
        axis[0,0].hist(data["Urea"], bins = 20)
        axis[0,0].set_title("Urea")
        axis[0,1].hist(data["No_Pation"], bins = 20)
        axis[0,1].set_title("No_Pation")
        axis[1,0].hist(data["Cr"], bins = 20)
        axis[1,0].set_title("Cr")
        axis[2,0].hist(data["HbA1c"], bins = 20)
        axis[2,0].set_title("HbA1c")
        axis[1,1].hist(data["Chol"], bins = 20)
        axis[1,1].set_title("Chol")
        axis[2,1].hist(data["TG"], bins = 20)
        axis[2,1].set_title("TG")
        pt.show()
```



```
In [ ]: figure,axis = pt.subplots(3)
        axis[0].hist(data["HDL"], bins = 20)
        axis[0].set_title("HDL")
        axis[1].hist(data["LDL"], bins = 20)
        axis[1].set_title("LDL")
        axis[2].hist(data["VLDL"], bins = 20)
        axis[2].set_title("VLDL")
        pt.show()
```



In [ ]:

```python
print("Skew of Urea = " + str(data["Urea"].skew()))
print("Skew of No_Pation = " + str((data["No_Pation"]).skew()))
print("Skew of Cr = " + str(data["Cr"].skew()))
print("Skew of HbA1c = " + str(data["HbA1c"].skew()))
print("Skew of Chol = " + str(data["Chol"].skew()))
print("Skew of TG = " + str(data["TG"].skew()))
print("Skew of HDL = " + str(data["HDL"].skew()))
print("Skew of LDL = " + str(data["LDL"].skew()))
print("Skew of VLDL = " + str(data["VLDL"].skew()))
```

```
Skew of Urea = 4.298927889976489
Skew of No_Pation = 19.56102868778898
Skew of Cr = 8.47415115621381
Skew of HbA1c = 0.22168940098610398
Skew of Chol = 0.6171226608860232
Skew of TG = 2.298456097830948
Skew of HDL = 6.283201244977265
Skew of LDL = 1.1459095882506958
Skew of VLDL = 5.350444676831296
```

So, we calculated the log of these features. Now they follow normal distribution, empirical rule was used.

In [ ]:

```python
data_normal = data.copy(deep = True)
data_deletedRows = data.copy(deep = True)
data_replaced = data.copy(deep = True)
```

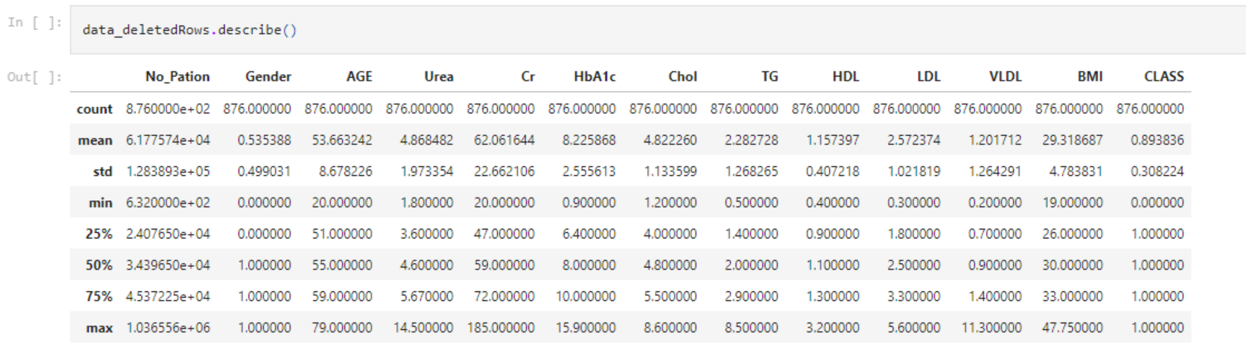
In [ ]:

```python
def empirical_rule(df,colName, log=1, replace = False):
    if log == 1:
        name = str("log"+colName)
        df[name] = np.log(df[colName])
    elif log == 0:
        name = colName
    col_std = df[name].std()
    col_mean = df[name].mean()
    min = col_mean - 3 * col_std
    max = col_mean + 3 * col_std

    if replace == False:
        return df.loc[(df[name]>= min) & (df[name]<=max)]
    else:
        df.loc[ (df[name] <= min), colName] = df[colName].mean()
        df.loc[ (df[name] >= max), colName] = df[colName].mean()
        return df
```

In [ ]:

```python
loglist = ["Urea", "No_Pation", "Cr", "TG", "HDL", "VLDL"]
list = ["Chol","LDL", "HbA1c"]

for i in loglist + list:
    if i in loglist:
        data_deletedRows = empirical_rule(data_deletedRows,i)
        data_deletedRows = data_deletedRows.drop(data_deletedRows.iloc[:,-1].name, axis=1)

        data_replaced = empirical_rule(data_replaced,i, replace=True)
        data_replaced = data_replaced.drop(data_replaced.iloc[:,-1].name, axis=1)
    elif i in list:
        data_deletedRows = empirical_rule(data_deletedRows,i,0)
        data_replaced = empirical_rule(data_replaced,i,0, True)
```

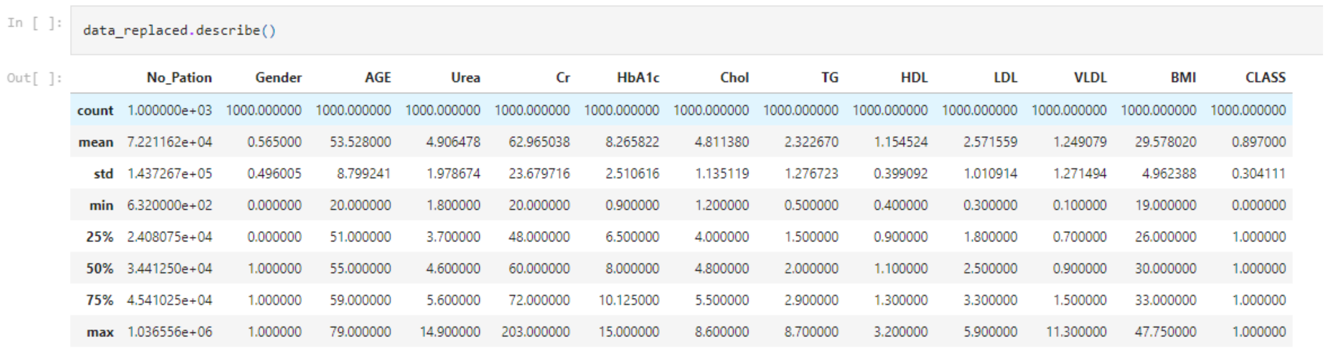
Data after the empirical rule:



```
In [ ]: data_deletedRows.describe()
```

Out[ ]:

	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
count	8.760000e+02	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000	876.000000
mean	6.177574e+04	0.535388	53.663242	4.868482	62.061644	8.225868	4.822260	2.282728	1.157397	2.572374	1.201712	29.318687	0.893836
std	1.283893e+05	0.499031	8.678226	1.973354	22.662106	2.555613	1.133599	1.268265	0.407218	1.021819	1.264291	4.783831	0.308224
min	6.320000e+02	0.000000	20.000000	1.800000	20.000000	0.900000	1.200000	0.500000	0.400000	0.300000	0.200000	19.000000	0.000000
25%	2.407650e+04	0.000000	51.000000	3.600000	47.000000	6.400000	4.000000	1.400000	0.900000	1.800000	0.700000	26.000000	1.000000
50%	3.439650e+04	1.000000	55.000000	4.600000	59.000000	8.000000	4.800000	2.000000	1.100000	2.500000	0.900000	30.000000	1.000000
75%	4.537225e+04	1.000000	59.000000	5.670000	72.000000	10.000000	5.500000	2.900000	1.300000	3.300000	1.400000	33.000000	1.000000
max	1.036556e+06	1.000000	79.000000	14.500000	185.000000	15.900000	8.600000	8.500000	3.200000	5.600000	11.300000	47.750000	1.000000



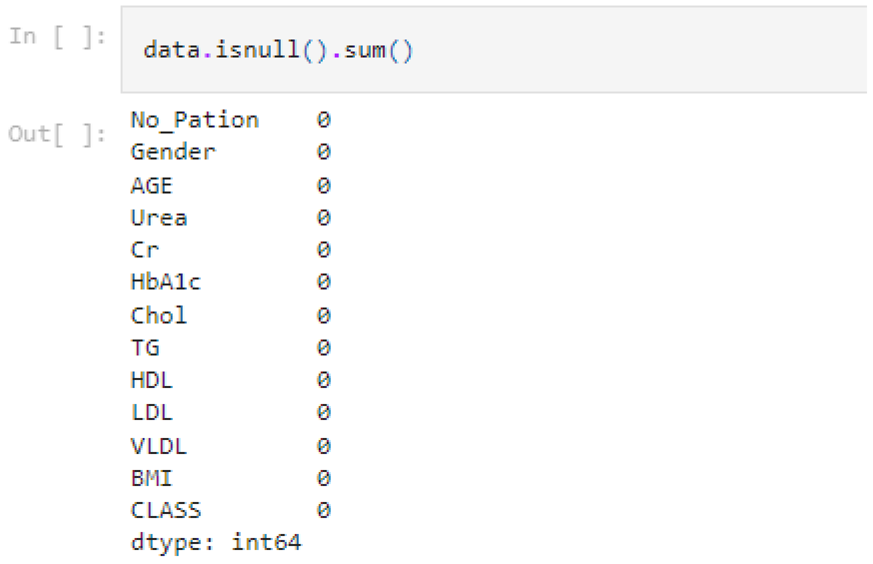
```
In [ ]: data_replaced.describe()
```

Out[ ]:

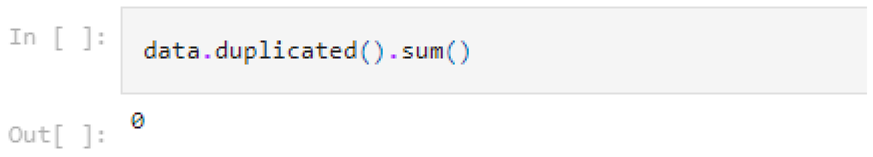
	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	BMI	CLASS
count	1.000000e+03	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	7.221162e+04	0.565000	53.528000	4.906478	62.965038	8.265822	4.811380	2.322670	1.154524	2.571559	1.249079	29.578020	0.897000
std	1.437267e+05	0.496005	8.799241	1.978674	23.679716	2.510616	1.135119	1.276723	0.399092	1.010914	1.271494	4.962388	0.304111
min	6.320000e+02	0.000000	20.000000	1.800000	20.000000	0.900000	1.200000	0.500000	0.400000	0.300000	0.100000	19.000000	0.000000
25%	2.408075e+04	0.000000	51.000000	3.700000	48.000000	6.500000	4.000000	1.500000	0.900000	1.800000	0.700000	26.000000	1.000000
50%	3.441250e+04	1.000000	55.000000	4.600000	60.000000	8.000000	4.800000	2.000000	1.100000	2.500000	0.900000	30.000000	1.000000
75%	4.541025e+04	1.000000	59.000000	5.600000	72.000000	10.125000	5.500000	2.900000	1.300000	3.300000	1.500000	33.000000	1.000000
max	1.036556e+06	1.000000	79.000000	14.900000	203.000000	15.000000	8.600000	8.700000	3.200000	5.900000	11.300000	47.750000	1.000000

Preprocessing:

We checked for nulls and duplicate using methods form pandas dataframe as follow and found nothing:

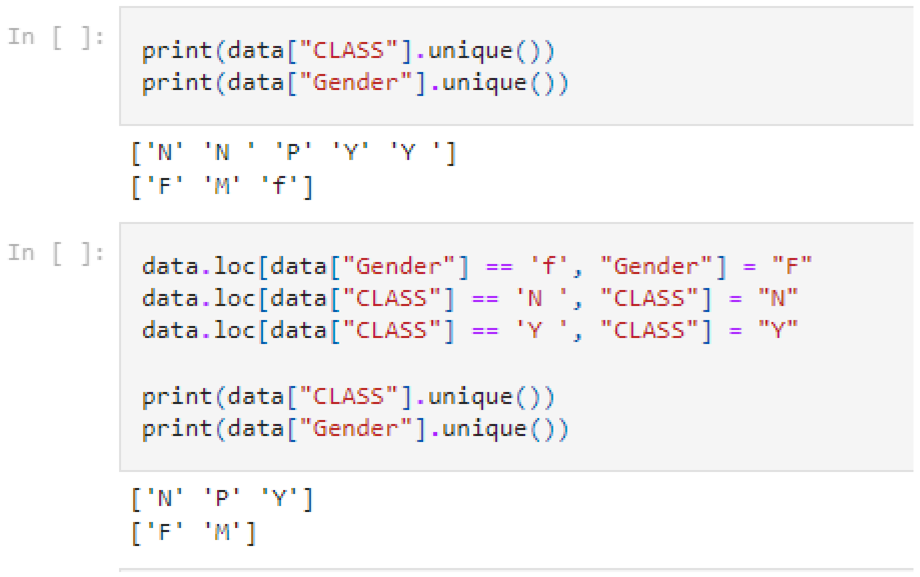


```
In [ ]: data.isnull().sum()

Out[ ]: No_Pation    0
        Gender       0
        AGE          0
        Urea         0
        Cr           0
        HbA1c        0
        Chol         0
        TG           0
        HDL          0
        LDL          0
        VLDL         0
        BMI          0
        CLASS        0
        dtype: int64
```



```
In [ ]: data.duplicated().sum()

Out[ ]: 0
```

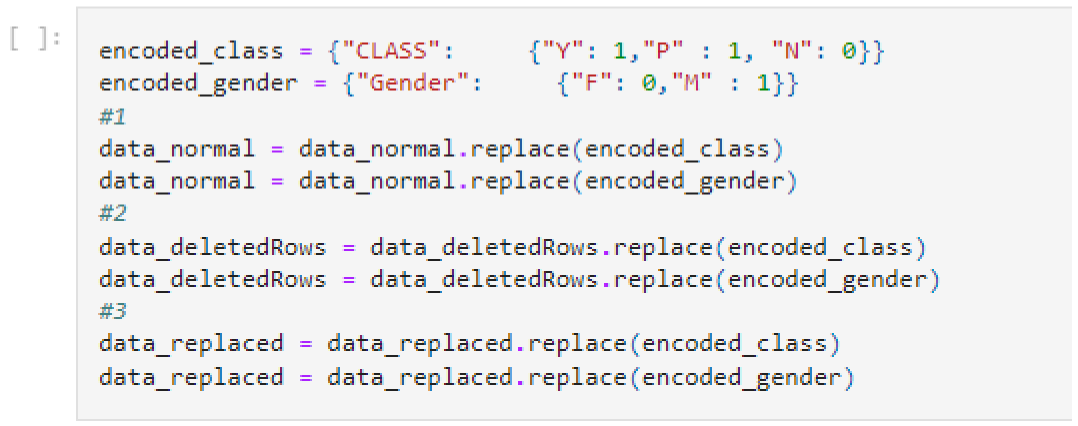
Then we checked for misspelling for categorical features using .unique method. we corrected how categorical features is written:



```
In [ ]:
print(data["CLASS"].unique())
print(data["Gender"].unique())

['N' 'N ' 'P' 'Y' 'Y ']
['F' 'M' 'f']
```

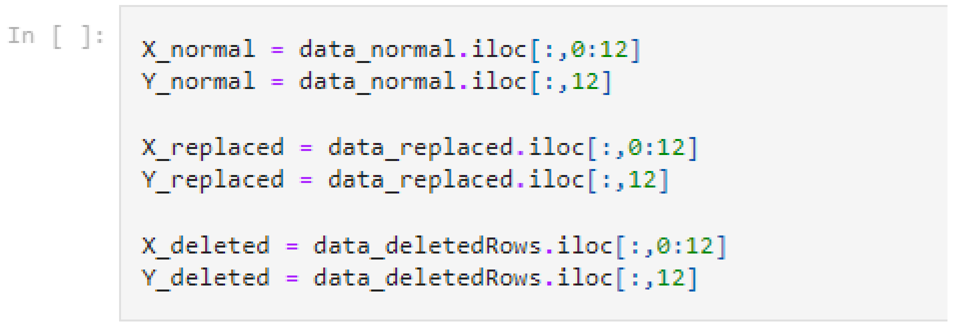
```
In [ ]:
data.loc[data["Gender"] == 'f', "Gender"] = "F"
data.loc[data["CLASS"] == 'N ', "CLASS"] = "N"
data.loc[data["CLASS"] == 'Y ', "CLASS"] = "Y"

print(data["CLASS"].unique())
print(data["Gender"].unique())

['N' 'P' 'Y']
['F' 'M']
```

We have applied the principle of label encoding as shown:



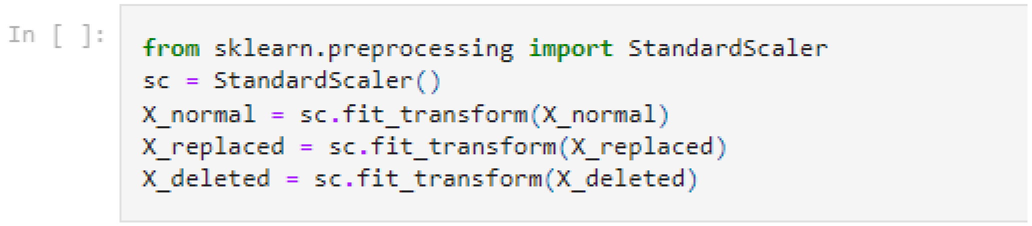
```
[ ]:
encoded_class = {"CLASS":     {"Y": 1,"P" : 1, "N": 0}}
encoded_gender = {"Gender":     {"F": 0,"M" : 1}}
#1
data_normal = data_normal.replace(encoded_class)
data_normal = data_normal.replace(encoded_gender)
#2
data_deletedRows = data_deletedRows.replace(encoded_class)
data_deletedRows = data_deletedRows.replace(encoded_gender)
#3
data_replaced = data_replaced.replace(encoded_class)
data_replaced = data_replaced.replace(encoded_gender)
```

we divided the data into X and Y using iloc method where Y have the label and X have the rest of the features



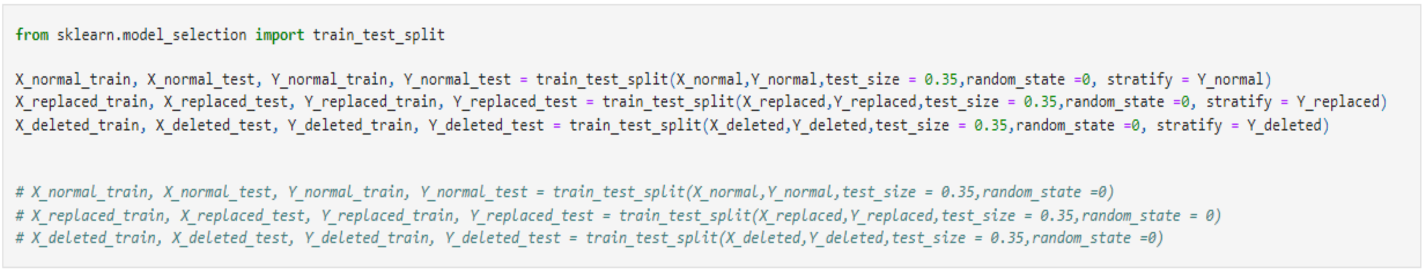
```
In [ ]:
X_normal = data_normal.iloc[:,0:12]
Y_normal = data_normal.iloc[:,12]

X_replaced = data_replaced.iloc[:,0:12]
Y_replaced = data_replaced.iloc[:,12]

X_deleted = data_deletedRows.iloc[:,0:12]
Y_deleted = data_deletedRows.iloc[:,12]
```

Finally, we standardized the data using StandardScaler function from sklearn.preprocessing library, which uses the standard score.



```
In [ ]:  from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_normal = sc.fit_transform(X_normal)
         X_replaced = sc.fit_transform(X_replaced)
         X_deleted = sc.fit_transform(X_deleted)
```

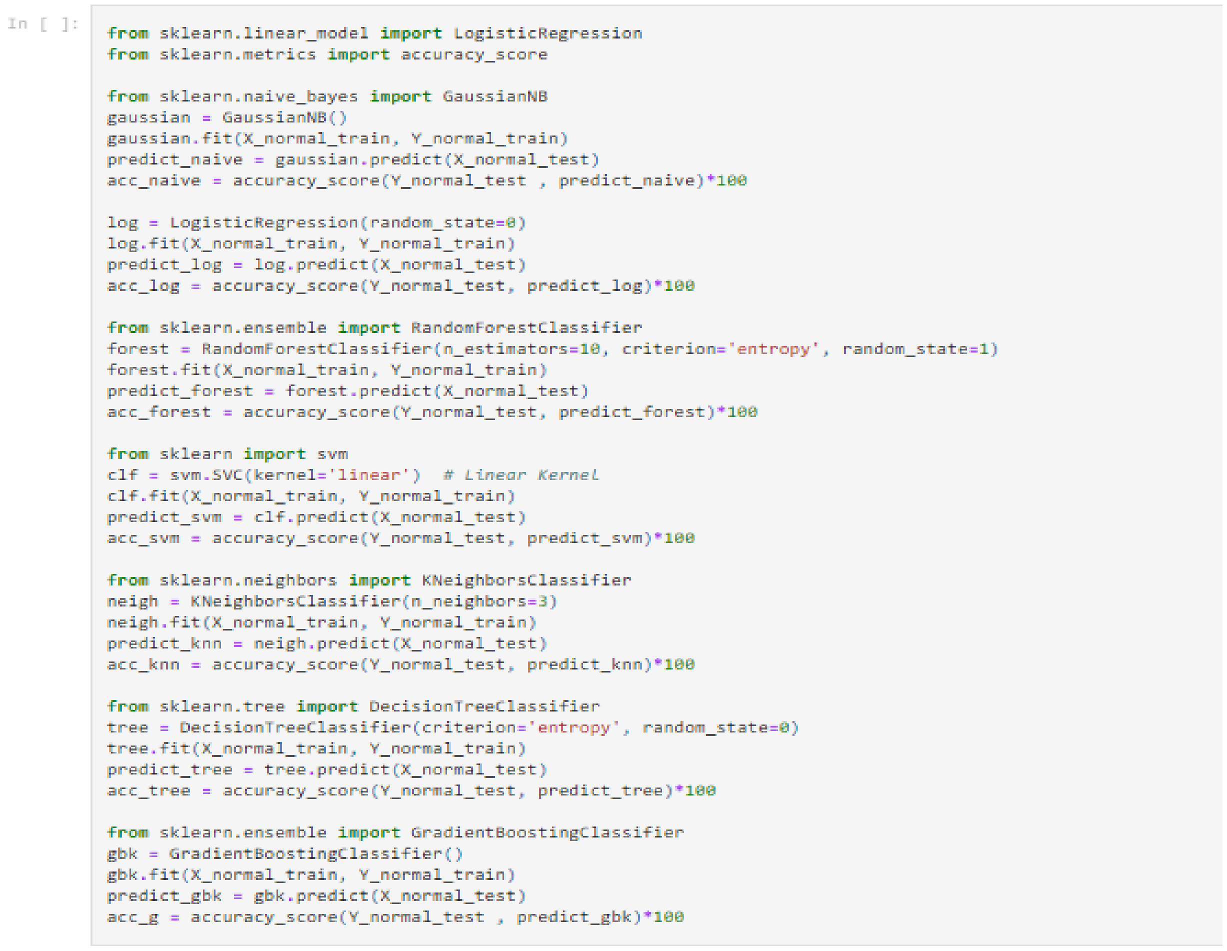
We split the data into test and train using train_test_split method from sklearn library. We used random sampling and stratified sampling, random sampling showed better accuracy further when testing and training the models than stratified sampling depending on the size of the sample.



```
from sklearn.model_selection import train_test_split

X_normal_train, X_normal_test, Y_normal_train, Y_normal_test = train_test_split(X_normal,Y_normal,test_size = 0.35,random_state =0, stratify = Y_normal)
X_replaced_train, X_replaced_test, Y_replaced_train, Y_replaced_test = train_test_split(X_replaced,Y_replaced,test_size = 0.35,random_state =0, stratify = Y_replaced)
X_deleted_train, X_deleted_test, Y_deleted_train, Y_deleted_test = train_test_split(X_deleted,Y_deleted,test_size = 0.35,random_state =0, stratify = Y_deleted)


# X_normal_train, X_normal_test, Y_normal_train, Y_normal_test = train_test_split(X_normal,Y_normal,test_size = 0.35,random_state =0)
# X_replaced_train, X_replaced_test, Y_replaced_train, Y_replaced_test = train_test_split(X_replaced,Y_replaced,test_size = 0.35,random_state = 0)
# X_deleted_train, X_deleted_test, Y_deleted_train, Y_deleted_test = train_test_split(X_deleted,Y_deleted,test_size = 0.35,random_state =0)
```

Classification Models:

We have built 7 different classification models using sklearn functions. And calculated the accuracy using accuracy_score method from the same library.



```
In [ ]:  from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score

         from sklearn.naive_bayes import GaussianNB
         gaussian = GaussianNB()
         gaussian.fit(X_normal_train, Y_normal_train)
         predict_naive = gaussian.predict(X_normal_test)
         acc_naive = accuracy_score(Y_normal_test , predict_naive)*100

         log = LogisticRegression(random_state=0)
         log.fit(X_normal_train, Y_normal_train)
         predict_log = log.predict(X_normal_test)
         acc_log = accuracy_score(Y_normal_test, predict_log)*100

         from sklearn.ensemble import RandomForestClassifier
         forest = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=1)
         forest.fit(X_normal_train, Y_normal_train)
         predict_forest = forest.predict(X_normal_test)
         acc_forest = accuracy_score(Y_normal_test, predict_forest)*100

         from sklearn import svm
         clf = svm.SVC(kernel='linear')  # Linear Kernel
         clf.fit(X_normal_train, Y_normal_train)
         predict_svm = clf.predict(X_normal_test)
         acc_svm = accuracy_score(Y_normal_test, predict_svm)*100

         from sklearn.neighbors import KNeighborsClassifier
         neigh = KNeighborsClassifier(n_neighbors=3)
         neigh.fit(X_normal_train, Y_normal_train)
         predict_knn = neigh.predict(X_normal_test)
         acc_knn = accuracy_score(Y_normal_test, predict_knn)*100

         from sklearn.tree import DecisionTreeClassifier
         tree = DecisionTreeClassifier(criterion='entropy', random_state=0)
         tree.fit(X_normal_train, Y_normal_train)
         predict_tree = tree.predict(X_normal_test)
         acc_tree = accuracy_score(Y_normal_test, predict_tree)*100

         from sklearn.ensemble import GradientBoostingClassifier
         gbk = GradientBoostingClassifier()
         gbk.fit(X_normal_train, Y_normal_train)
         predict_gbk = gbk.predict(X_normal_test)
         acc_g = accuracy_score(Y_normal_test , predict_gbk)*100
```

A comparison between the accuracy when using stratified and random for normal data

sample /model	**Random Sampling Normal data**	**Stratified Sampling Normal data**
Logistic	96.85714285714285	96.85714285714285
Decision Tree	99.14285714285714	99.42857142857143
KNN	95.71428571428572	95.71428571428572
SVM	96.85714285714285	96.57142857142857
Naïve bayes	93.14285714285714	86.85714285714286
Random Forest	98.28571428571429	99.71428571428571
GB	99.14285714285714	99.14285714285714

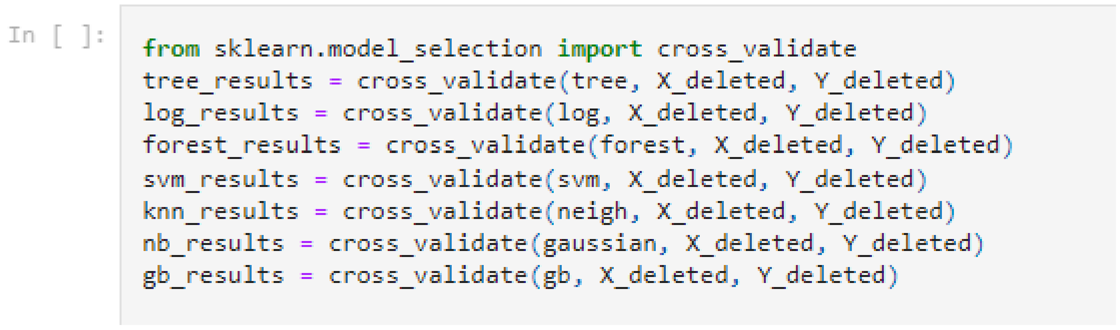
A comparison between the accuracy when using stratified and random for deleted data

sample /model	**Random Sampling**	**Stratified Sampling**
Logistic	96.09120521172639	96.41693811074919
Decision Tree	99.3485342019544	98.37133550488599
KNN	94.78827361563518	92.83387622149837
SVM	95.43973941368078	95.76547231270358
Naïve bayes	93.48534201954396	93.81107491856677
Random Forest	99.0228013029316	97.06840390879479
GB	99.0228013029316	97.39413680781759

A comparison between the accuracy when using stratified and random for replaced data

sample /model	Random Sampling	Stratified Sampling
Logistic	96.0	96.0
Decision Tree	99.14285714285714	99.14285714285714
KNN	95.42857142857143	94.28571428571428
SVM	97.14285714285714	96.57142857142857
Naïve bayes	95.71428571428572	90.0
Random Forest	98.85714285714286	99.71428571428571
GB	99.14285714285714	99.14285714285714

We have used Cross validation using cross_validate method from sklearn.model_selection:



```
In [ ]:
from sklearn.model_selection import cross_validate
tree_results = cross_validate(tree, X_deleted, Y_deleted)
log_results = cross_validate(log, X_deleted, Y_deleted)
forest_results = cross_validate(forest, X_deleted, Y_deleted)
svm_results = cross_validate(svm, X_deleted, Y_deleted)
knn_results = cross_validate(neigh, X_deleted, Y_deleted)
nb_results = cross_validate(gaussian, X_deleted, Y_deleted)
gb_results = cross_validate(gb, X_deleted, Y_deleted)
```

Cross Validation results for normal data:

```
Decision tree 5-fold test score: 84.0 , 99.0 , 100.0 , 98.5 , 99.5
Logistic Regression 5-fold test score: 73.5 , 97.0 , 98.0 , 96.0 , 97.0
Random Forest 5-fold test score: 66.0 , 98.5 , 100.0 , 97.5 , 97.5
SVM 5-fold test score: 72.0 , 96.5 , 97.0 , 97.0 , 97.0
KNN 5-fold test score: 59.0 , 97.5 , 99.5 , 97.0 , 94.5
Naive Bayes 5-fold test score: 69.0 , 99.5 , 99.0 , 99.0 , 94.5
```

Cross Validation results for replaced data:

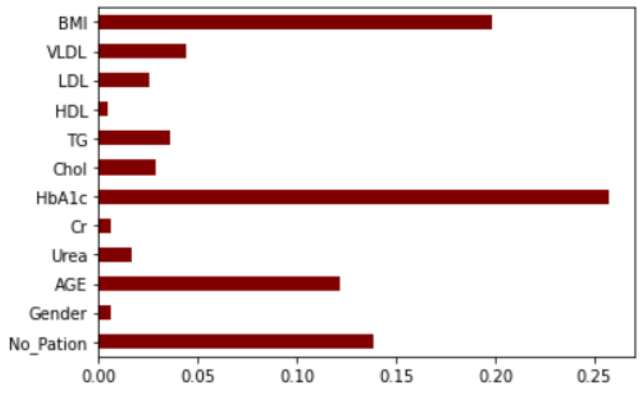
```
Decision tree 5-fold test score: 84.0 , 99.0 , 100.0 , 98.5 , 99.5
Logistic Regression 5-fold test score: 75.5 , 97.0 , 97.5 , 96.0 , 97.0
Random Forest 5-fold test score: 63.0 , 98.5 , 100.0 , 98.0 , 97.0
SVM 5-fold test score: 78.0 , 97.5 , 98.5 , 96.0 , 96.5
KNN 5-fold test score: 60.5 , 98.5 , 97.5 , 95.5 , 90.0
Naive Bayes 5-fold test score: 66.0 , 99.0 , 100.0 , 97.5 , 96.0
```

Cross Validation results for deleted data:

```
Decision tree 5-fold test score: 82.38636363636364 , 98.85714285714286 , 100.0 , 98.85714285714286 , 99.42857142857143
Logistic Regression 5-fold test score: 74.43181818181817 , 96.57142857142857 , 97.71428571428571 , 94.28571428571428 , 96.571428
857
Random Forest 5-fold test score: 61.93181818181818 , 98.85714285714286 , 100.0 , 96.57142857142857 , 98.28571428571429
SVM 5-fold test score: 75.0 , 96.0 , 98.28571428571429 , 95.42857142857143 , 96.0
KNN 5-fold test score: 60.79545454545454 , 98.85714285714286 , 97.14285714285714 , 95.42857142857143 , 86.28571428571429
Naive Bayes 5-fold test score: 60.79545454545454 , 99.42857142857143 , 100.0 , 97.14285714285714 , 95.42857142857143
```

Feature Importance: We used mutual_info_classif method from sklearn.feature_selection to find out how each feature has a relation with the Class label:

```
In [ ]:  from sklearn.feature_selection import mutual_info_classif
         import matplotlib.pyplot as plt

         importances = mutual_info_classif(X_deleted,Y_deleted)
         feat_importances = pd.Series(importances, data_deletedRows.columns[0:len(data_deletedRows.columns)-1])
         feat_importances.plot(kind = 'barh', color = 'maroon')
         plt.show()
```



We concluded that the factors (No_pation, age, HbA1c and BMI) have the highest influence on determining which case belongs.

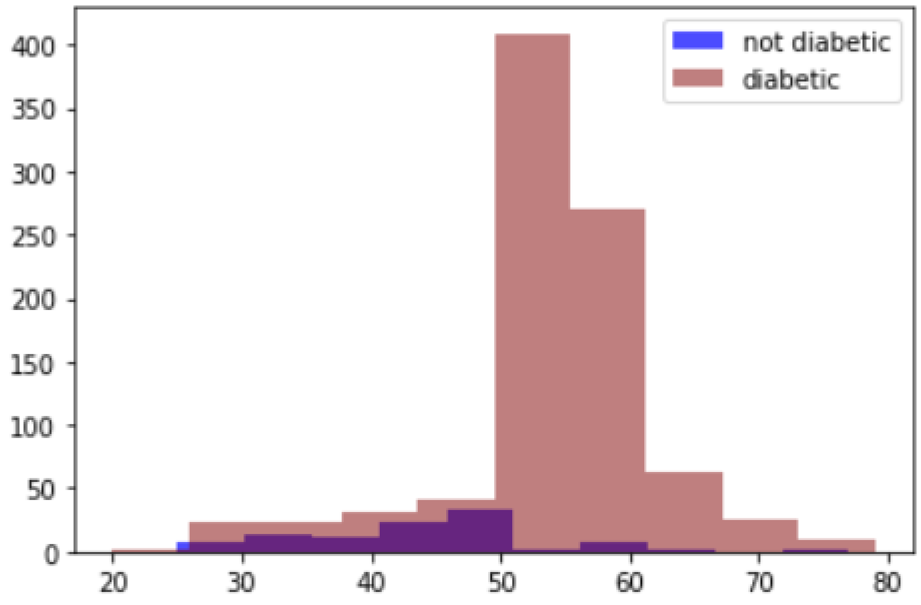
Data Analysis:

We found that 14.71% of women don't have diabetes, however 85.29% of them have diabetes. For men, 6.9% don't have diabetes and 93.1% have the disease. We concluded that men are more prone to be diagnosed as diabetic patient.

```
In [ ]: print(data_normal.groupby(["Gender","CLASS"])["CLASS"].count())
```

```
Gender  CLASS
0       0         64
        1        371
1       0         39
        1        526
Name: CLASS, dtype: int64
```

We discovered that certain age group suffer from the disease more than others.


not diabetic
diabetic