Introduction to Data Mining

"Final Project"

"Topic: Diabetes Classification"

Instructor: Sir Sajjad Haider

Group Members:

- *M. Jaffar Abbas – 19666*
- *Shaheryar Raza – 19681*
- *Subhan Bawany – 19686*
- *Zohair Abbas – 19688*

Problem Description:

Diabetes is a serious chronic disease which affects a person's ability to effectively regulate levels of glucose in the blood, which can result in reduced quality of life and life expectancy. It is generally characterized by the body not making enough insulin or the body being unable to use the produced insulin effectively as per the body needs. The food that we eat is broken down into sugars during digestion and those sugars are released into the bloodstream. After this, pancreas gets the signal to release insulin which has the task of enabling the cells in human body to use those sugars in bloodstream for energy.

Despite so many advancements in the medical world and technology, there is still no cure for diabetes. We might have seen the elders in our family injecting the insulin in their body at least once in a day, and that is a way of keeping sugars of diabetic patients in control. The complications related to heart, vision, and kidney diseases are often associated with chronically high levels of sugar in the bloodstream for the people with diabetes. Since there is no cure for it, there exists strategies like healthy eating habits, losing weight, being active, and receiving medical treatments like insulin injections to alleviate the harms of diabetes among patients. The best thing that we can possibly do is early diagnosis of diabetes among people through predictive models as it can lead to the lifestyle changes and more effective treatment for people.

Diabetes causes significant financial burden on the economy of United States as diagnosed diabetes costs nearly US$ 327 billion and total costs with undiagnosed diabetes and prediabetes are approximately US$ 400 billion annually. According to the data collected in 2018 by Centers for Disease Control and Prevention (CDC), 34.2 million Americans have diabetes, and 88 million people have prediabetes. Moreover, the estimates by the CDC indicate that 1 in 5 diabetic patients, and 8 out of 10 prediabetics are not aware about their risk. There are different types of diabetes but the most common is type II diabetes and its prevalence varies by age, education, income, location, race, and other social determinants of health. Therefore, we need some sort of data collection method which collects the data of above-mentioned attributes of individual and build predictive models which can somehow predict if a person has diabetes or not.

Thankfully, we have one such collection method or collection of responses in the form of Behavioral Risk Factor Surveillance System (BRFSS) which is a health-related telephone survey and is annually collected by CDC. Approximately, responses from 400,000 Americans are collected each year through this survey. The survey collects information about health-related risk behaviors, chronic health conditions, and the use of preventative services by the people. It has been conducted every year since 1984.

So, we want to utilize the data collected from BRFSS and answer our main problem which is stated below:

"Can survey questions from the BRFSS provide accurate predictions of whether an individual has diabetes?"

Along with this, we want to explore some research questions like:

"What risk factors are most predictive of diabetes risk?"

"Can we use a subset of the risk factors to accurately predict whether an individual has diabetes or not?"

"Can we create a short form of BRFSS using feature selection to accurately predict if someone might have diabetes or is at high risk of diabetes?"

Considering our main business problem, we aim to apply the classification techniques like Random Forest, Decision Trees and others that we have learned in our Introduction to Data Mining course. Moreover, for the research questions that we aim to explore during our journey of classification, we plan to use the feature selection techniques and then apply classification algorithms to see the what insights we can get from the dataset.

Data Description:

The dataset that we have used in our project reflects to a survey which classifies that an individual has diabetes or not. The dataset named diabetes _ binary _ 5050split _ health _ indicators _ BRFSS2015 contains 70,692 responses of the survey that is the actual size of the dataset.

This dataset is a *balanced* dataset containing 50-50 responses of people being positive of diabetes or not. The target class is 'Diabetes_binary' that contains binary values, 0 for no diabetes and 1 for prediabetes or diabetes. Each row comprises of 21 unique feature variables that holds significance in some sense. Below are all the features:

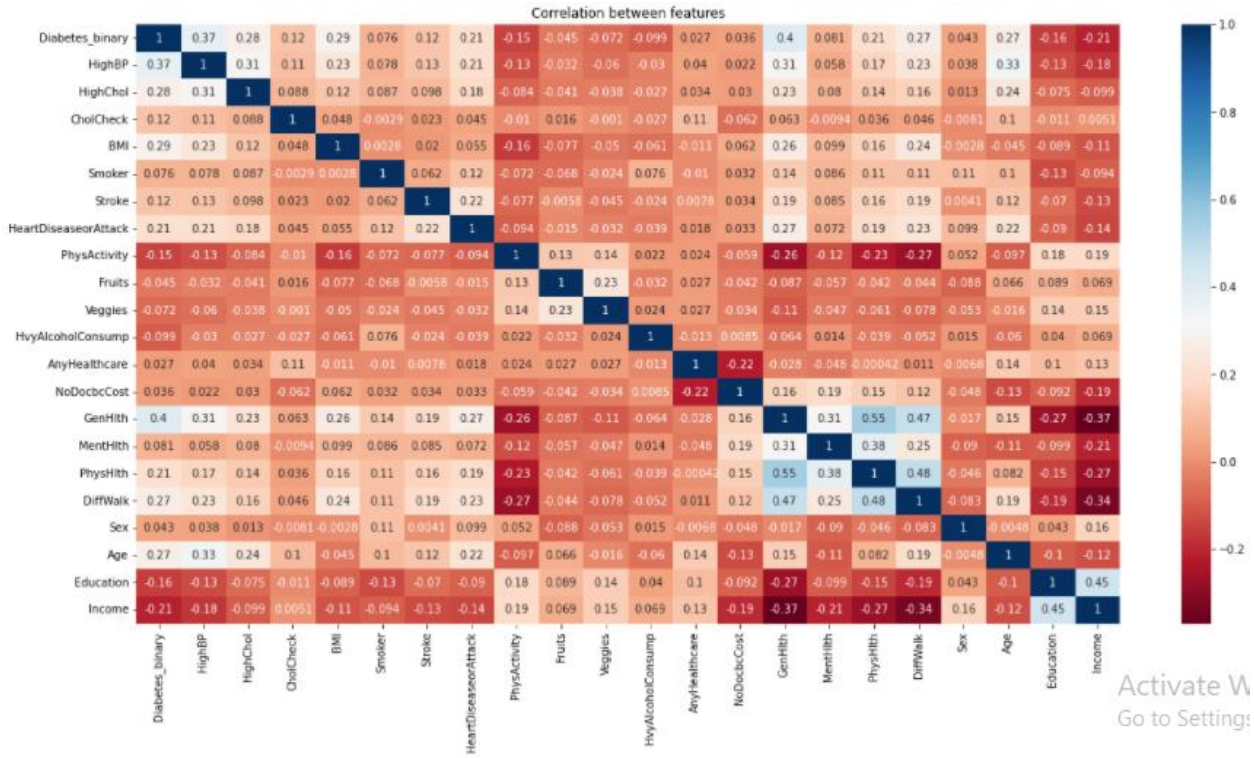
- **Diabetes_binary:** This is the target column which declares that an individual has no diabetes, prediabetes, or diabetes.
- **HighBP** and **HighChol:** These two separate columns shows that if a person has blood pressure or cholesterol so it is of high category which is represented by 1 or normal denoted by 0.
- **CholCheck:** It checks that an individual has checked its cholesterol in 5 years by representing it by 1 and 0 for the opposite criteria.
- **BMI:** It is Body Mass Index of the individual. The maximum value is 98 and minimum value is 12.
- **Smoker:** It denotes the count of 100 cigarettes smoked by a person by 1 and 0 for person who smoked less than hundred cigarettes.
- **Stroke** or **HeartDiseaseorAttack**: These two columns show that people having stroked or heart disease such as coronial heart disease or myocardial infarction are shown by 1 and 0 for vice versa.
- **PhysActivity:** The value is 1 for those individuals involved in Physical Activity in last 30 days and 0 for no physical activity.
- **Fruits** and **Veggies:** These two columns give value as 1 for those who have consumed fruit or vegetable more than once per day.
- **HvyAlcoholConsump:** This column identifies for adult men having 14 or greater drinks per week and women having 7 or greater drinks per week as 1 for 'yes' and 0 for 'no'.

- **AnyHealthcare:** It gives the value 1 for those having health care coverage and 0 for vice versa.
- **NoDocbcCost:** This column identifies those individuals who didn't see a doctor for 12 months due to cost, 1 indicates yes while 0 shows no.
- **GenHlth:** It is the rating of one's general health from scale 1-5, 1 being the highest and 5 the lowest. This column is in ordinal manner.
- **MentHlth** and **PhysHlth:** These two columns are similar as they keep a check of having poor mental health in past 30 days for MentHlth and for PhysHlth it keeps check of having any physical injury in past 30 days.
- **DiffWalk:** This column tells that if an individual is having difficulty in walking or climbing stairs, so the value is 1 otherwise 0.
- **Sex:** The value 0 is for female and 1 for male.
- **Age:** This column gives value 1 for age between 18 and 24, 9 for age between 60 and 64 and 13 for 80 years and older.
- **Education:** This column indicates the level of education from a scale 1 – 6.
- **Income:** This column shows the level of income on a scale 1 – 8.

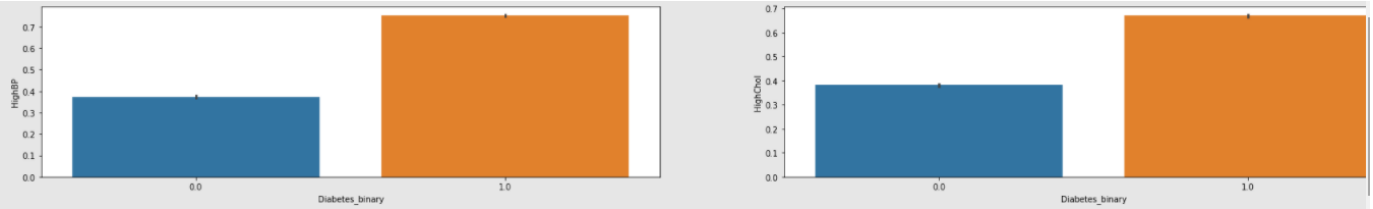
Quality of data plays a major role when modeling and extracting insights from it. Talking about the quality of this dataset, we can say that this dataset has 21 complete columns with no missing values. All the columns are accurate and has relevancy for the intended use when referring to the business problem. There has been no error value entered in the columns, so we don't need to clean the data. There is no such column in this dataset that has constant values throughout the column. All these characteristics indicates that dataset is refined enough to conduct the modeling and experiment different algorithms.

Understanding Data using Visualizations / EDA:

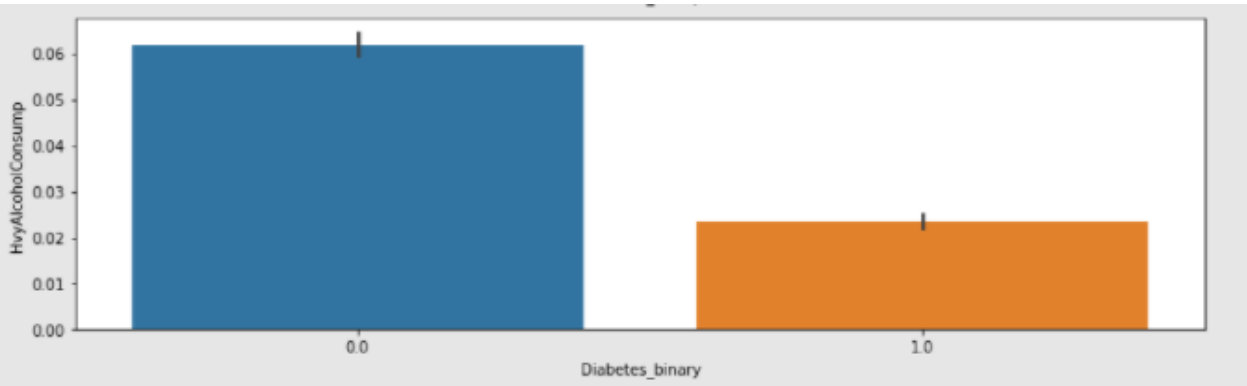
EDA and Data Visualization are some of the important steps of Data Analysis which can help in understanding the data and the underlying interactions between the different features. Here are some of the things which we tried in order to find out the relation between the features.



Insight 1: Based on this visualization, we can conclude that HighBP, BMI, GenHlth, Age are some of the features positively correlated to Diabetes_binary while Income, Education, PhysActivity are some of the negatively correlated features.



Insight 2: A person with Diabetes is also likely to have a problem of High BP and High Cholesterol.



Insight 3: A person with Diabetes is not likely to have a heavy Alcohol consumption

Data Cleaning:

Data cleaning is the process of correcting or removing incorrect, damaged data within a dataset. Fortunately, data set in our case was clean. Following code samples would assure the process of checking whether data cleaning is required or not.

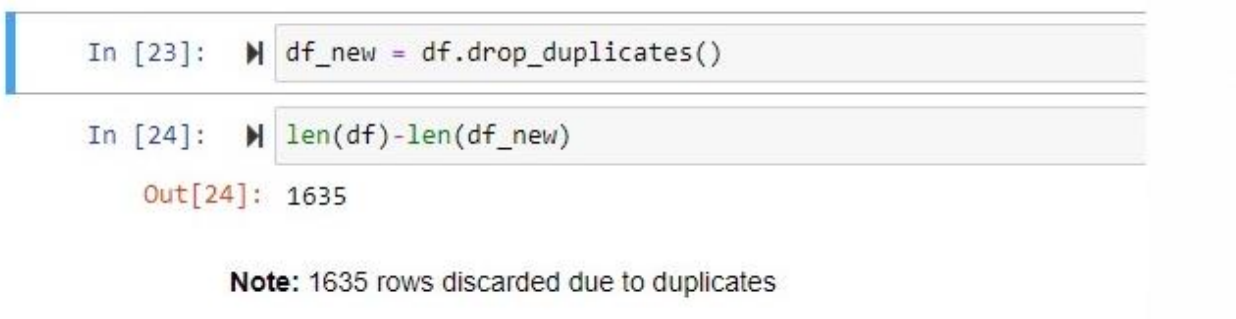
```
In [18]: df.isna().sum()

Out[18]: Diabetes_binary         0
         HighBP                  0
         HighChol                0
         CholCheck               0
         BMI                     0
         Smoker                  0
         Stroke                  0
         HeartDiseaseorAttack    0
         PhysActivity            0
         Fruits                  0
         Veggies                 0
         HvyAlcoholConsump       0
         AnyHealthcare           0
         NoDocbcCost             0
         GenHlth                 0
         MentHlth                0
         PhysHlth                0
         DiffWalk                0
         Sex                     0
         Age                     0
         Education               0
         Income                  0
         dtype: int64
```

```
In [19]: df.isnull().sum()

Out[19]: Diabetes_binary         0
         HighBP                  0
         HighChol                0
         CholCheck               0
         BMI                     0
         Smoker                  0
         Stroke                  0
         HeartDiseaseorAttack    0
         PhysActivity            0
         Fruits                  0
         Veggies                 0
         HvyAlcoholConsump       0
         AnyHealthcare           0
         NoDocbcCost             0
         GenHlth                 0
         MentHlth                0
         PhysHlth                0
         DiffWalk                0
         Sex                     0
         Age                     0
         Education               0
         Income                  0
         dtype: int64
```

Note: Hence no null or NA values in the dataset

However, if it was found to be dirty for example inconsistency in data or null values/Not applicable values or strange naming conventions then following approaches would be taken to remove dirtiness.

Steps to deal with Duplicate Data if found: In order to cater duplicate data in data set we remove it to clean our data, code snippet is attached to show cast how recurrence of data is handled.



Note: 1635 rows discarded due to duplicates

Approach for handling typos and strange naming conventions:

If there are rows which were found to have any naming or typo difference then we would bring those rows to a standard value, since our data labels are binary and ordinal in all of the cases so we didn't handle this problem.

Approaches to take if any Null or Missing Data was there:

We planned several techniques to play with our missing or null values for our data, in which main one are, to fill it with our own assumptions or placing mean value or mode values but doing so would lose integrity of our data, next approach we could take was to remove that record which had any missing our null value in any field but going with this option would also bring a drawback to us in terms of reducing data size etc. As specified above with the help of code snippet to check that whether there were any missing or null values or not, so we couldn't find any in our case to deal with it.

Modeling:

Under the process of modeling, we tried several known models such as Decision Trees, Random Forest, Gradient Boosted and Naïve Bayes, while working on all of these models we got to experience different output either in terms of raising our accuracy for target variable or in decreasing as well. In all model we split our training and test data in the ratio of 70:30.

```
In [17]: X = df_new.drop('Diabetes_binary', axis = 1)
         Y = df_new['Diabetes_binary']
         print(X.shape)
         print(Y.shape)

         (69057, 21)
         (69057,)

In [18]: x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
         print(x_train.shape)
         print(x_test.shape)
         print(y_train.shape)
         print(y_test.shape)

         (48339, 21)
         (20718, 21)
         (48339,)
         (20718,)
```

Decision Tree Models:

We tried out three different settings under this model to figure out the trend in our accuracy.

Decision Tree

```
In [33]: desicion_tree = DecisionTreeClassifier(random_state = 0)
         desicion_tree.fit(x_train, y_train)
         desicion_tree.score(x_test, y_test)

Out[33]: 0.6508832898928468
```

Note: Default Settings

```
In [34]: desicion_tree = DecisionTreeClassifier(random_state = 0, criterion="entropy")
         desicion_tree.fit(x_train, y_train)
         desicion_tree.score(x_test, y_test)

Out[34]: 0.6528139781832223
```

Note: Changed criteria to Entropy

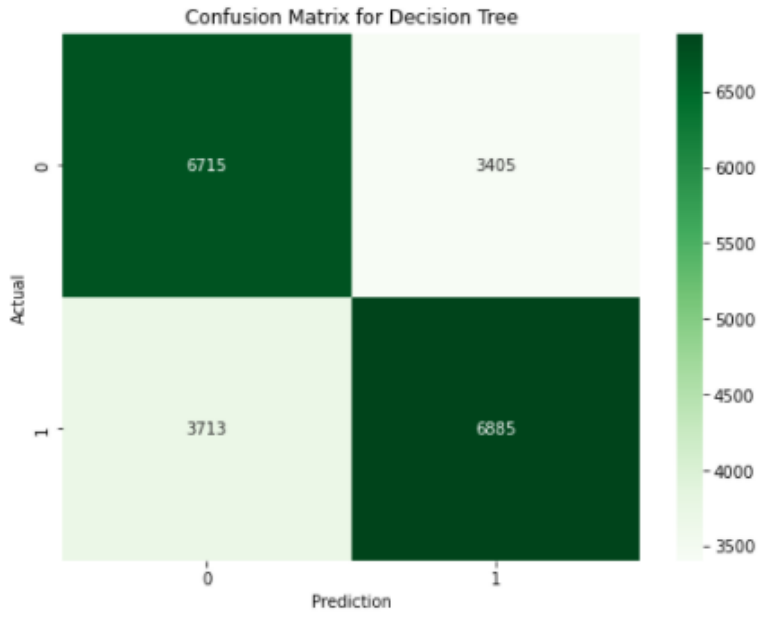
```
In [35]: desicion_tree = DecisionTreeClassifier(random_state = 0, criterion="entropy", splitter="random")
         desicion_tree.fit(x_train, y_train)
         desicion_tree.score(x_test, y_test)

Out[35]: 0.6564340187276764
```

Note: Changed splitter to random

- In the first model of Decision Tree, we used default settings which gave us accuracy around 65.1%. By getting idea through this model, we played over other parameters to get change in our result.
- In the second model we changed our criterion to “Entropy” which was “Gini” in default setting, with this change in our model we got to see that our accuracy was increased slightly to 65.3%.
- In our third try with Decision Tree Model we simply replace our splitter option to “Random” which was basically “best” in our first case and kept the criterion as “Entropy” same as in case 2. By changing our model in terms of splitter we noticed much greater increase in accuracy as compared to our initial model which was 65.6%.

*The best model for Decision Tress is the **third one** as it brought the highest accuracy of 0.656 when we changed splitter and criterion in same model.*

```
In [38]: plt.figure(figsize = (8,6))
         sns.heatmap(confusion_matrix_dt, annot = True, fmt = ".0f", cmap = 'Greens')
         plt.title("Confusion Matrix for Decision Tree")
         plt.xlabel("Prediction")
         plt.ylabel("Actual")
         plt.show()
```



Confusion Matrix of our best Decision tree Model

The confusion matrix tells that for Diabetic patients the model correctly predicted 6885 individuals. We can deduce the following:

- True Positives: 6885
- True Negatives: 6715
- False Positives: 3405
- False Negatives: 3713

```
In [39]: report_dt = metrics.classification_report(y_test, predictions_dt)
         print(report_dt)
```

	precision	recall	f1-score	support
0	0.64	0.66	0.65	10120
1	0.67	0.65	0.66	10598
accuracy			0.66	20718
macro avg	0.66	0.66	0.66	20718
weighted avg	0.66	0.66	0.66	20718

This is the final report of the Decision Tree Model. Through this report we can see that for non-diabetic patients the precision is 0.64, recall is 0.66 and F1-score is 0.65. On the other hand, for diabetic patients the precision is 0.67, recall is 0.65 and F1-score is 0.66. Overall accuracy is 0.66 for this model.

Random Forest Models:

While working with Random Forest we tried 6 different configurations in search of getting better and better accuracy.

Random Forest

```
In [19]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1)
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.5s finished

Out[19]: 0.7337098175499566
```

Note: Default Settings

```
In [21]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=500)
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:   25.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    3.4s finished

Out[21]: 0.7370402548508543
```


Note: Increased Models

```
In [22]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=500, criterion="entropy")
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:   27.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    3.0s finished

Out[22]: 0.7380055989960421
```

Note: Changed criteria to Entropy

```
In [23]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=500, criterion="entropy", max_features="lo
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:   27.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    2.9s finished

Out[23]: 0.7380055989960421
```

Note: Changed Max Features to Log2 (no changes)

```
In [24]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=500, criterion="entropy", max_features="lc
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:   45.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    3.4s finished

Out[24]: 0.7289313640312771
```

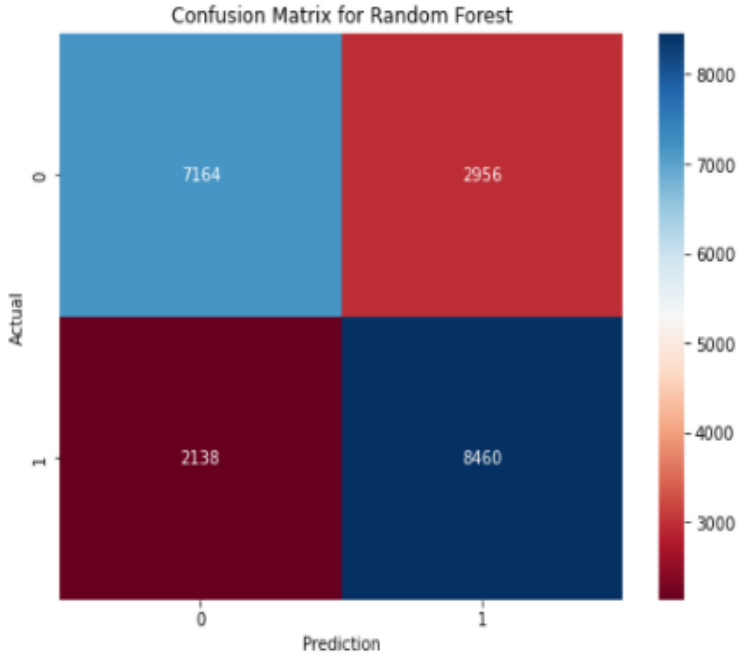
Note: Changed Bootstrap to False, Accuracy decreased

```
In [25]: random_forest = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=1000, max_depth=10)
         random_forest.fit(x_train, y_train)
         random_forest.score(x_test, y_test)

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1000 out of 1000 | elapsed:   26.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1000 out of 1000 | elapsed:    2.7s finished

Out[25]: 0.7541268462206777
```

Note: Increased models with max depth 10, Accuracy increased

- In the first Model default configuration was used which had 100 number of trees and “Gini” as its criterion, this model was tested to get a generic picture of our result through Random Forest which gave higher accuracy of 73.4% much higher than the best model of Decision Tree in fact.
- In the Second model we figured out little bit higher accuracy just by increasing number of trees to 500 while keeping rest of the configuration same as the first one.
- Under third model we kept number of trees as 500 and changed our criterion from Gini to Entropy. However, we could not see a significant rise in our accuracy.
- In our fourth approach we added a parameter of max_features which basically consider the number of feature when looking for best split. When keeping its value as Log 2 we saw that there were not any changes in our accuracy.
- Going with our fifth approach where we changed Bootstrap parameter which uses samples to build trees, so when we kept the value of this parameter as false, so it decreased our accuracy by 0.009, which proved that this technique was not helpful for us.
- In our final model where we increased our number of trees to 1000 and used another parameter of max_depth which denotes the maximum depth of a tree in model. By keeping the value of max_depth as 10 we found out **highest accuracy** (75.41%) among all the other models of Random Forest.

```
In [29]: plt.figure(figsize = (8,6))
sns.heatmap(confusion_matrix_rf, annot = True, fmt = ".0f", cmap = 'RdBu')
plt.title("Confusion Matrix for Random Forest")
plt.xlabel("Prediction")
plt.ylabel("Actual")
plt.show()
```



Confusion Matrix of our best Random Forest Model

The confusion matrix tells that for Diabetic patients the model correctly predicted 8460 individuals. We can deduce the following:

- True Positives: 8460
- True Negatives: 7164
- False Positives: 2956
- False Negatives: 2138

In [32]:
```
report_rf = metrics.classification_report(y_test, predictions_rf)
print(report_rf)
```

```
              precision    recall  f1-score   support

           0       0.77      0.71      0.74     10120
           1       0.74      0.80      0.77     10598

    accuracy                           0.75     20718
   macro avg       0.76      0.75      0.75     20718
weighted avg       0.76      0.75      0.75     20718
```

This is the final report of the Random Forest Model. Through this report we can see that for non-diabetic patients the precision is 0.77, recall is 0.71 and F1-score is 0.74. On the other hand, for diabetic patients the precision is 0.74, recall is 0.80 and F1-score is 0.77. Overall accuracy is 0.75 for this model.

Gradient Boosting Models:

We used gradient boosting algorithm to classify our target class. We were able to test 6 different gradient boosting settings to visualize the changes in accuracy.

Gradient Boosting

```
In [40]: gradient_boosting = GradientBoostingClassifier(random_state = 0)
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[40]: 0.7545612510860121
```

Note: Default settings

```
In [46]: gradient_boosting = GradientBoostingClassifier(random_state = 0, n_estimators = 300)
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[46]: 0.7561058017183125
```

Note: Increased models

```
In [42]: gradient_boosting = GradientBoostingClassifier(random_state = 0, n_estimators = 300, learning_rate = 0.15)
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[42]: 0.7547060527077903
```

Note: Changed learning rate to 0.15

- The first case is simply running on default setting and has given an accuracy of 0.754, this can be taken as the basic criteria to judge gradient boosting with different parameters.
- In the second case we increased the number of estimators to 300. Number of estimators is basically the number of boosting stages that needs to perform. Increasing the models resulted in a slight increase of accuracy which is 0.756.
- Now for the third case we used the second case and increased the value of one of the parameters which is learning rate. Keeping the learning rate to 0.15 we noticed the accuracy decreased by 0.002 which is 0.754 but still it is better than the default setting.

```
In [43]: gradient_boosting = GradientBoostingClassifier(random_state = 0, n_estimators = 300, criterion='mse')
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[43]: 0.7561058017183125
```

Note: Changed criteria to Mean Squared Error

```
In [44]: gradient_boosting = GradientBoostingClassifier(random_state = 0, n_estimators = 300, max_depth = 10)
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[44]: 0.7252630562795637
```

Note: Changed max depth to 10

```
In [45]: gradient_boosting = GradientBoostingClassifier(random_state = 0, n_estimators = 300, subsample = 0.8)
         gradient_boosting.fit(x_train, y_train)
         gradient_boosting.score(x_test, y_test)

Out[45]: 0.7553335264021623
```

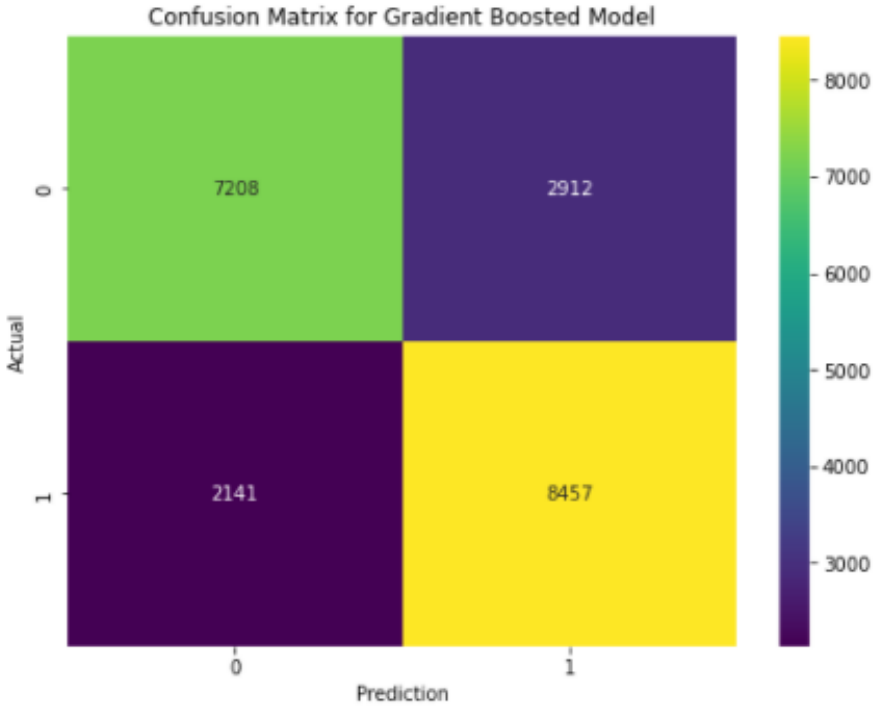
Note: Changed subsample to 80%

- In the fourth case we kept the number of models to 300 and changed the criterion to *mean square error*. The criterion is the function to measure the quality of the split. The accuracy came as 0.756 same as the second case. So we can conclude that the changing the criterion had no effect on accuracy.
- In the fifth case we used another parameter maximum depth by keeping it value to 10 and keeping number of models to 300. Maximum depth is used to limit the number of nodes in a tree. As a result of change in this parameter we can see that the accuracy decreased by 0.03 when compared to default setting.
- In the last case we used subsample which is the fraction of sample used for fitting individual base learning, keeping its value less than 1.0 can reduce variance and increase bias. This parameter influenced accuracy as the output was 0.755.

*The best model for Gradient Boosting is the **second case** as it had the accuracy of 0.756 when we increased the number of estimators to 300.*

In [50]:

```
plt.figure(figsize = (8,6))
sns.heatmap(confusion_matrix_gb, annot = True, fmt = ".0f", cmap = 'viridis')
plt.title("Confusion Matrix for Gradient Boosted Model")
plt.xlabel("Prediction")
plt.ylabel("Actual")
plt.show()
```



Confusion Matrix of our best Gradient Boosting Model

This confusion matrix is of the best model of gradient boosting. The confusion matrix tells that for Diabetic patients the model correctly predicted 8457 individuals. We can deduce the following:

- True Positives: 8457
- True Negatives: 7208
- False Positives: 2912
- False Negatives: 2141

```
In [51]: report_gb = metrics.classification_report(y_test, predictions_gb)
         print(report_gb)
```

```
              precision    recall  f1-score   support

           0       0.77      0.71      0.74     10120
           1       0.74      0.80      0.77     10598

    accuracy                           0.76     20718
   macro avg       0.76      0.76      0.76     20718
weighted avg       0.76      0.76      0.76     20718
```

This is the final report of the Gradient Boosted Algorithm. Through this report we can see that for non-diabetic patients the precision is 0.77, recall is 0.71 and F1-score is 0.74. On the other hand, for diabetic patients the precision is 0.74, recall is 0.80 and F1-score is 0.77. Overall accuracy is 0.76 for this model.

Naive Bayes Models:

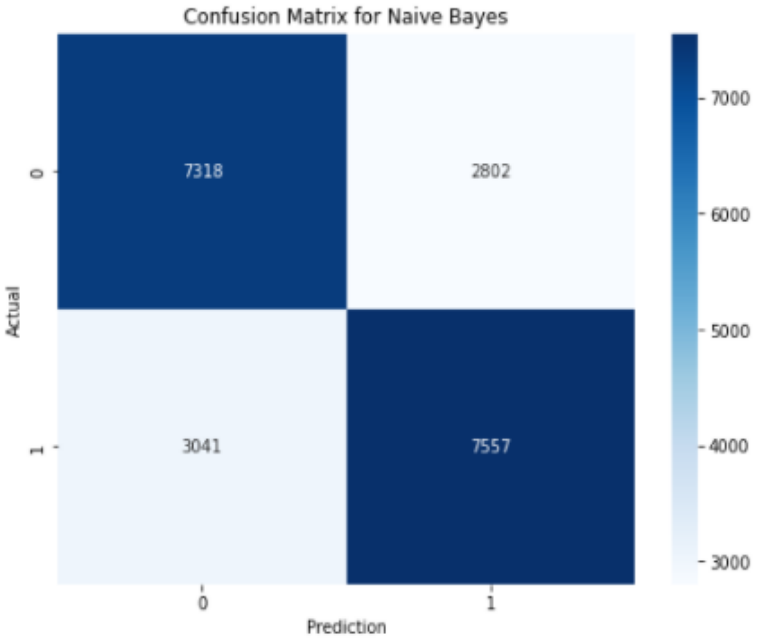
This algorithm was used keeping in mind the application of bayes theorem and how is this helpful in classification. As this algorithm doesn't have such parameter to differentiate or increase the accuracy so we only used Gaussian Naïve Bayes.

Naive Bayes

```
In [52]: naive_bayes = GaussianNB()
         naive_bayes.fit(x_train, y_train)
         naive_bayes.score(x_test, y_test)

Out[52]: 0.7179747079833961
```

As we can see that Gaussian Naive Bayes is used and the accuracy is 0.717 so we consider this as our best model.

```
In [55]: plt.figure(figsize = (8,6))
         sns.heatmap(confusion_matrix_nb, annot = True, fmt = ".0f", cmap = 'Blues')
         plt.title("Confusion Matrix for Naive Bayes")
         plt.xlabel("Prediction")
         plt.ylabel("Actual")
         plt.show()
```



Confusion Matrix of our best Naive Bayes Model

This is the confusion matrix of the Naïve Bayes Algorithm. It has correctly predicted 7557 individuals who are diabetic. Hence, we can deduce the following:

- True Positives: 7557
- True Negatives: 7318
- False Positives: 2802
- False Negatives: 3041

```
In [56]: report_nb = metrics.classification_report(y_test, predictions_nb)
         print(report_nb)
```

```
              precision    recall  f1-score   support

           0       0.71      0.72      0.71     10120
           1       0.73      0.71      0.72     10598

    accuracy                           0.72     20718
   macro avg       0.72      0.72      0.72     20718
weighted avg       0.72      0.72      0.72     20718
```

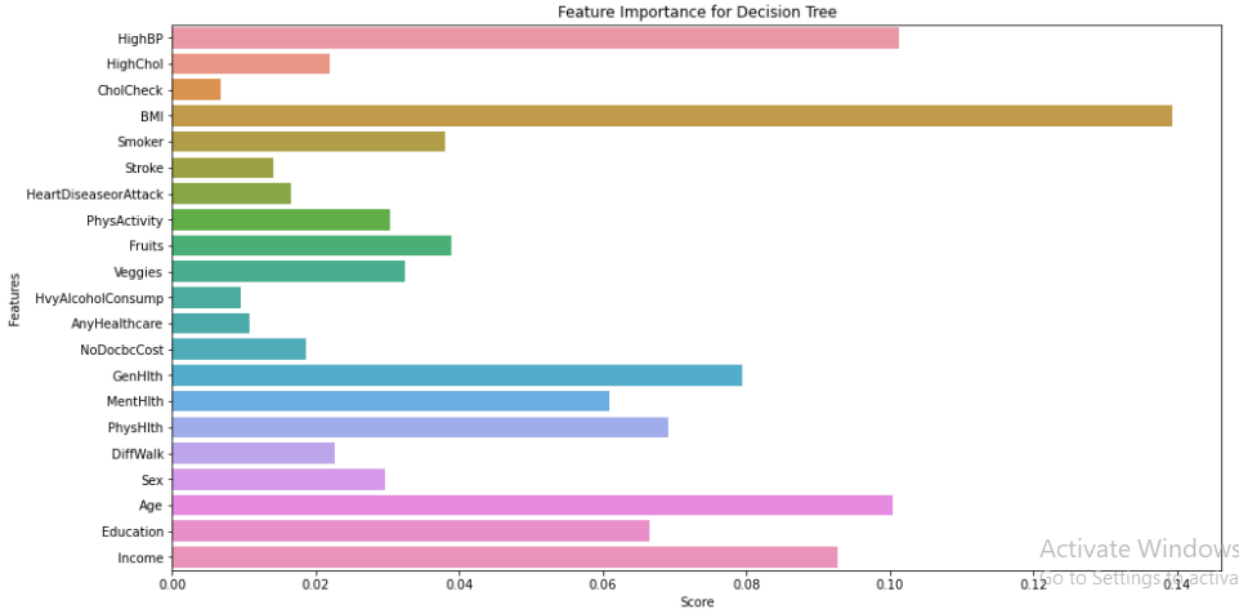
This is the final classification report of the Naive Bayes Algorithm. Through this report we can see that for non-diabetic patients the precision is 0.71, recall is 0.72 and F1-score is 0.71. On the other hand, for diabetic patients the precision is 0.73, recall is 0.71 and F1-score is 0.72. Overall accuracy is 0.72 for this model.

Feature Selection:

After training our models we came to an idea to reduce the number of columns that have minimal effect on the accuracy. This method was used in order to get more insight of our dataset and to increase predictive power of classification algorithms that were used. As a result, we selected important features from our dataset. This was done based on observation after several experiments.

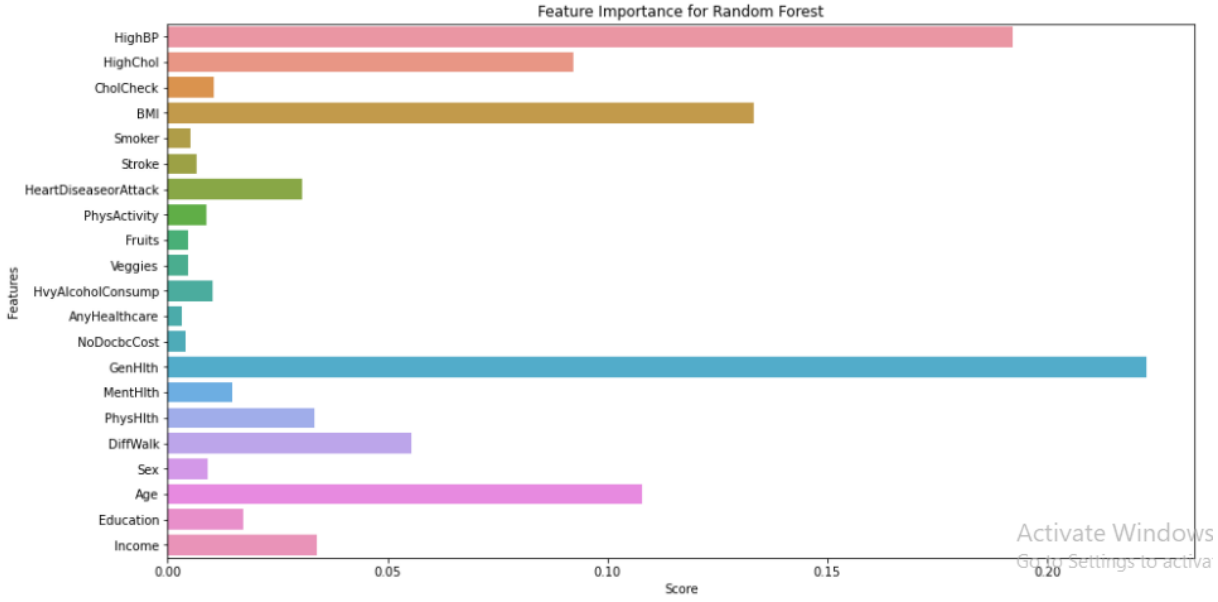
We were able to identify 10 features from our dataset for each algorithm that were highly correlated and have importance with the accuracy. Below are attached the visuals showing feature selection.

Decision Tree:



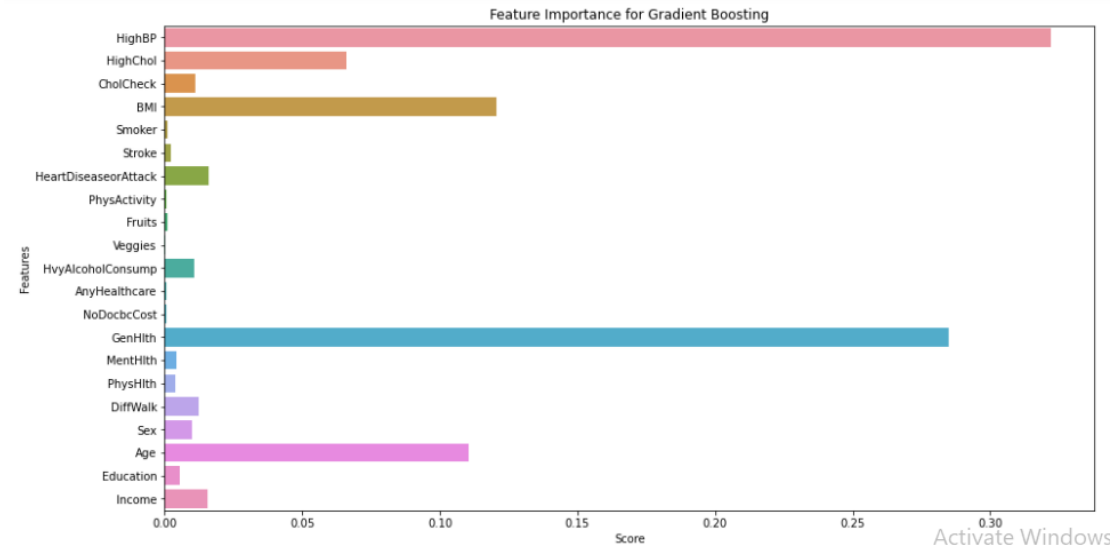
The above graph shows the most important features that were selected. We can see the values of each feature and these values indicate their importance. The most impactful feature for Decision Tree is BMI and the least important feature is CholCheck.

Random Forest:



This graph is used to select feature for Random Forest. In this graph we can deduce that the feature with the highest value that is GenHlth is the most important feature for this algorithm. The feature that has the minimal effect is AnyHealthCare.

Gradient Boosting:



The above graph is used to select features for Gradient Boosting. We can see that HighBP will play a major role when predicting the target class.

Naive Bayes:

As Naïve Bayes are based on the methodology to determine conditional and unconditional probability so there are no coefficients related to features that determine the importance. As a result, no method was followed to extract important features for this algorithm.

Modeling with Feature Selection:

Based on observations, we have concluded that the following features seems to be important than others, therefore we are selected following features and tested our models again with these features only. The features are as follows:

- High BP
- Stroke
- High Chol
- PhysHlth
- Chol check
- Heart Disease or attack
- BMI
- GenHlth
- Smoker
- Age

Random Forest

Previously, the accuracy achieved by this model was 75.41%

```
In [92]: random_forest_new = RandomForestClassifier(random_state = 0, verbose = 1, n_estimators=1000, max_depth=10)
         random_forest_new.fit(x_train_new, y_train_new)
         random_forest_new.score(x_test_new, y_test_new)

         [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
         [Parallel(n_jobs=1)]: Done 1000 out of 1000 | elapsed:   37.0s finished
         [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
         [Parallel(n_jobs=1)]: Done 1000 out of 1000 | elapsed:    3.7s finished
Out[92]: 0.7499758663963703
```

Note: Seems like accuracy decreased a bit in this case

Decision Tree

Previously, the accuracy achieved by this model was 65.64%

```
In [94]: desicion_tree_new = DecisionTreeClassifier(random_state = 0, criterion="entropy", splitter="random")
         desicion_tree_new.fit(x_train_new, y_train_new)
         desicion_tree_new.score(x_test_new, y_test_new)
Out[94]: 0.6799401486629983
```

Note: Seems like accuracy increased in this case

Random Forest Model with feature selection: When top 10 most important features were extracted and the previously best model among Random Forest was tested so we saw that our accuracy decreased and got nearer to other models of Random Forest, accuracy went down to 74.9%.

Decision Tree Model with Feature Selection: Our best model of Decision Tree before feature selection was giving accuracy around 65.6% but after implementing feature selection in this model it went up to 67.9% which was a sharp increase to our previous best accuracy.

Gradient Boosting

```
Previously, the accuracy achieved by this model was 75.61%
```

```
In [95]: gradient_boosting_new = GradientBoostingClassifier(random_state = 0, n_estimators = 300)
         gradient_boosting_new.fit(x_train_new, y_train_new)
         gradient_boosting_new.score(x_test_new, y_test_new)

Out[95]: 0.7512790809923737
```

Note: Seems like accuracy decreased a bit in this case

Naive Bayes

Previously, the accuracy achieved by this model was 71.8%

```
In [97]: naive_bayes_new = GaussianNB()
         naive_bayes_new.fit(x_train_new, y_train_new)
         naive_bayes_new.score(x_test_new, y_test_new)

Out[97]: 0.7306689834926151
```

Note: Seems like accuracy increased in this case

Gradient Boosting Model with feature selection: After selecting the most 10 important features so we compared the accuracy with the previous best model of the same algorithm. Surprisingly the accuracy decreased, the previous best accuracy was 75.6% and after feature selection the accuracy went down to 75.1%.

Naive Bayes Model with Feature Selection: After the process of feature selection, we used the most significant columns that could impact the accuracy of the model. We can see that this process increased the accuracy to 73.1% as previously this model was predicting accurately with 71.8%.

Findings:

After experimenting new models with different parameters, we observed few things that will help to cater anyone who has this dataset and needs to understand how the target class is predicted. There were **no such limitations** in our dataset. As mentioned in the problem statement these following questions will be answered.

1. ***Can survey questions from the BRFSS provide accurate predictions of whether an individual has diabetes?***

   Based on our model prediction we can conclude that using this dataset we can only predict with 75% accuracy that an individual will be diabetic. Due to this we can suggest another feature in this dataset which could be number of meals per day. This feature will help in predicting diabetes in an individual as having greater number of meals can increase calorie intake and can reduce the time between each meal which are connected directly or indirectly of a person having diabetes.

2. ***What risk factors are most predictive of diabetes risk?***

   Although every feature included had a greater or minimal impact on the predictive power but according to us following features were the most highlighted risk:

   - HighBP: High Blood Pressure is an obvious link to diabetes as both are side by side.
   - GenHlth: General Health refers to an individual's daily health or activities. If a person's general health is compromised so according to the models used, they are more prone to diabetes.
   - BMI: Body Mass Index plays a major role as if a person's BMI is out of range so he or she has a higher risk to diabetes.
   - Age: Age is one of the important factors as a person who is older has a higher risk to diabetes as compared to a person who is young.

3. ***Can we use a subset of the risk factors to accurately predict whether an individual has diabetes?***

   Yes, we can use the subset of the risk factors to accurately predict diabetic individuals. The subset is:

   {'HighBP', 'HighChol', 'BMI', 'PhysHlth', 'HeartDiseaseorAttack', 'GenHlth', 'Age'}

   This subset is mentioned due to the feature selection done above in the report.

4. ***Can we create a short form of questions from the BRFSS using feature selection to accurately predict if someone might have diabetes or is at high risk of diabetes? / Advice to the organization:***

   After performing feature selection on our dataset several models' accuracy increased. Hence, this supports the idea of making a shorter from of questions that will classify that an individual has diabetes. Moreover, we can also claim that not all the features had an impact on the prediction few of the features had minimal effect which were catered during feature selection.

When will our model expire?

As our dataset is completely clean and comprises mostly of binary features, so there are very less chances or scenarios in which our model will not be able to perform up to the mark; hence, after performing all the techniques we didn't notice any case in which our model expires or needs to be re-trained

Conclusion:

This report contains all the stages of the CRISP-DM cycle and is designed according to our findings and observations of the data. We made sure that we explain the insights of the data through different visualizations as well so that a person without the technical background can also understand and will be able to extract important information.