IT307MiniProject(Midsem)

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.shape
```

Sample of what's present in our data set.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2768 entries, 0 to 2767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               2768 non-null   int64  
 1   Glucose                   2768 non-null   int64  
 2   BloodPressure             2768 non-null   int64  
 3   SkinThickness             2768 non-null   int64  
 4   Insulin                   2768 non-null   int64  
 5   BMI                       2768 non-null   float64
 6   DiabetesPedigreeFunction  2768 non-null   float64
 7   Age                       2768 non-null   int64  
 8   Outcome                   2768 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 194.8 KB
```

Feature information, like which data type each feature has got and number of non null counts.

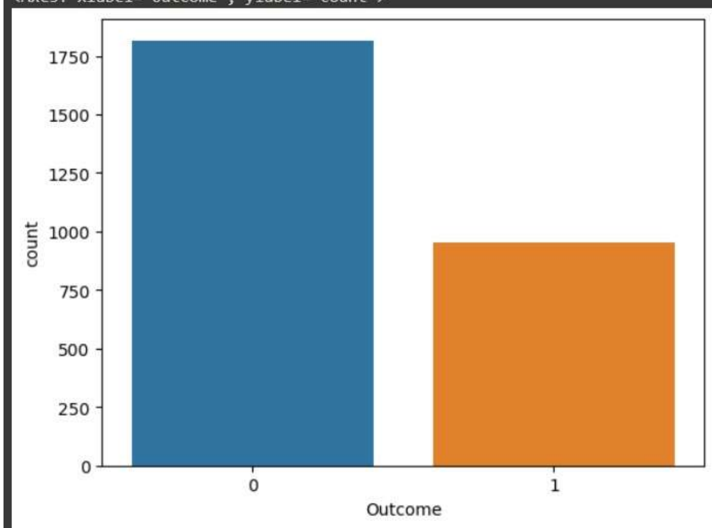
```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	2768.0	3.742775	3.323801	0.000	1.000	3.000	6.000	17.00
Glucose	2768.0	121.102601	32.036508	0.000	99.000	117.000	141.000	199.00
BloodPressure	2768.0	69.134393	19.231438	0.000	62.000	72.000	80.000	122.00
SkinThickness	2768.0	20.824422	16.059596	0.000	0.000	23.000	32.000	110.00
Insulin	2768.0	80.127890	112.301933	0.000	0.000	37.000	130.000	846.00
BMI	2768.0	32.137392	8.076127	0.000	27.300	32.200	36.625	80.60
DiabetesPedigreeFunction	2768.0	0.471193	0.325669	0.078	0.244	0.375	0.624	2.42
Age	2768.0	33.132225	11.777230	21.000	24.000	29.000	40.000	81.00
Outcome	2768.0	0.343931	0.475104	0.000	0.000	0.000	1.000	1.00

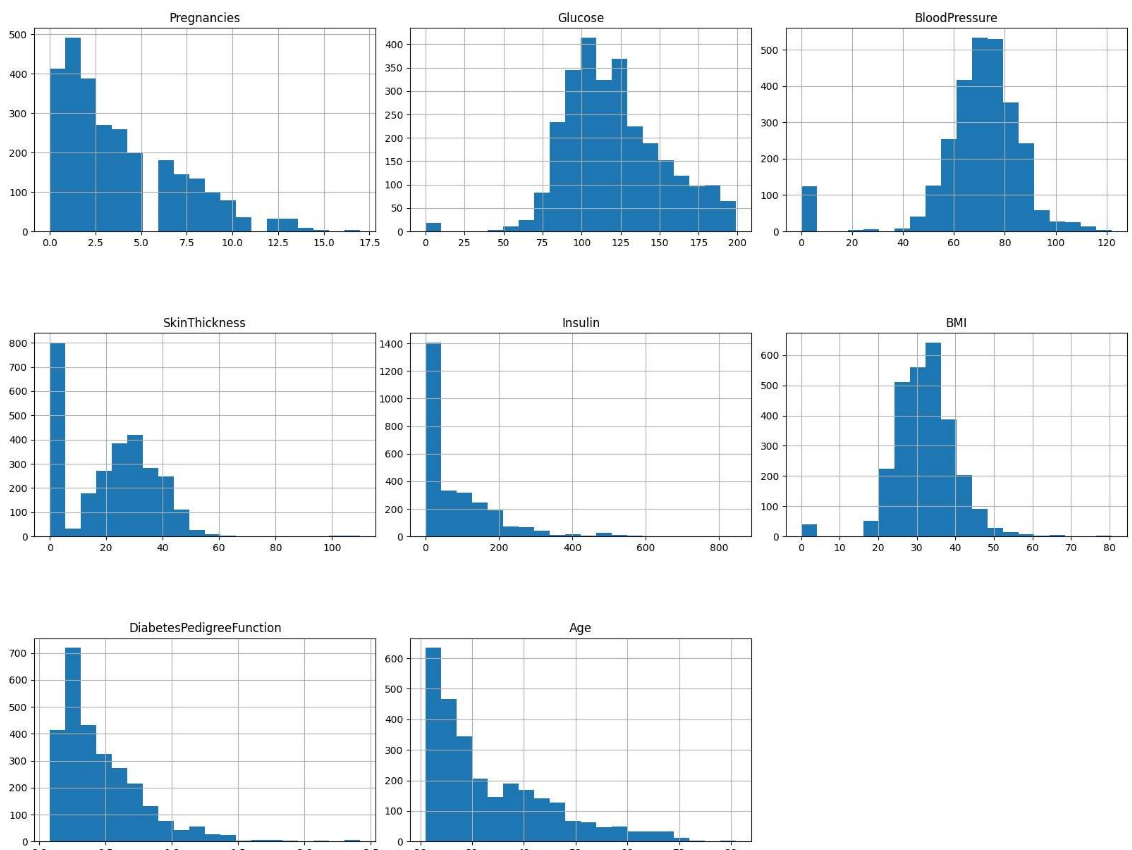
Describing the statistics to our features in a tabular format. This is mostly like the way the whisker plot actually works. A 5-number describer of our dataset.

```
# Outcome countplot
sns.countplot(x = 'Outcome',data = df)
```

```
<Axes: xlabel='Outcome', ylabel='count'>
```

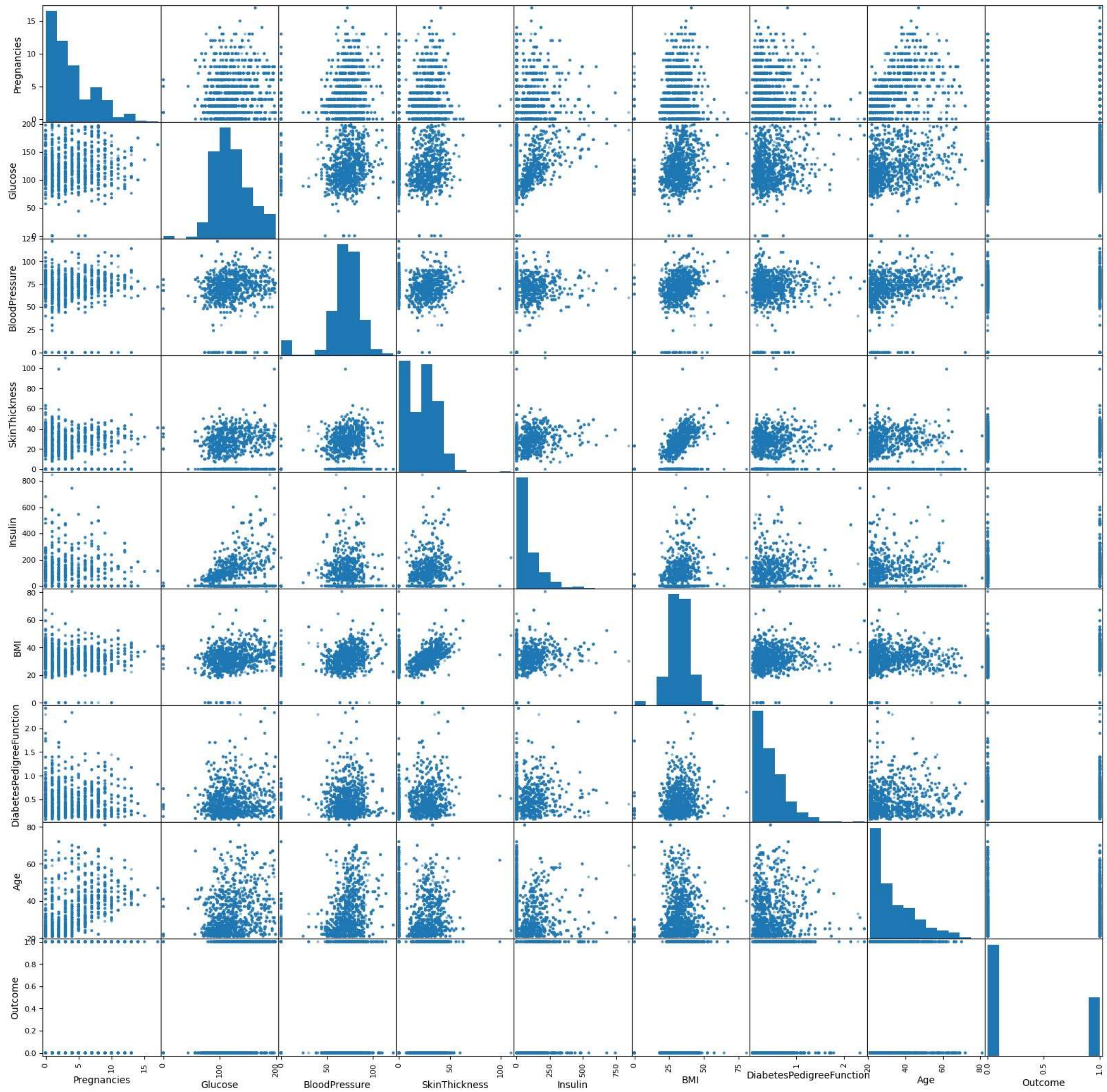


number of diabetic patients and non diabetic patients as shown in our data setas shown in our data set.

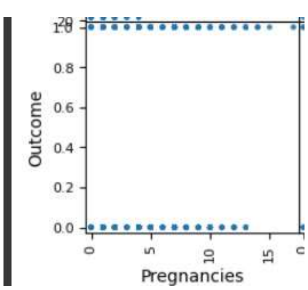


For example if we take, Five pregnancies we have 200 cases present in our dataset. a similar thing applies for all the attributes we have in our data set.

IT307MiniProject(Midsem)



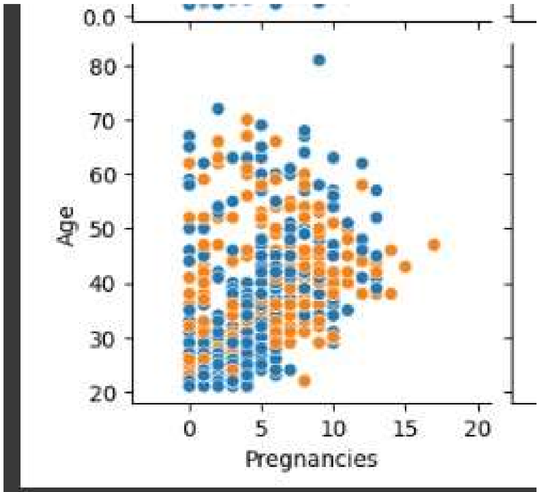
here basically what we are doing ismeans its like a comparison between feature and feature. for example while comparing the outcome versus pregnancy the Plot says we have so many outcomes with one and so many outcomes with 0.



Similar plotting happens in other features also.

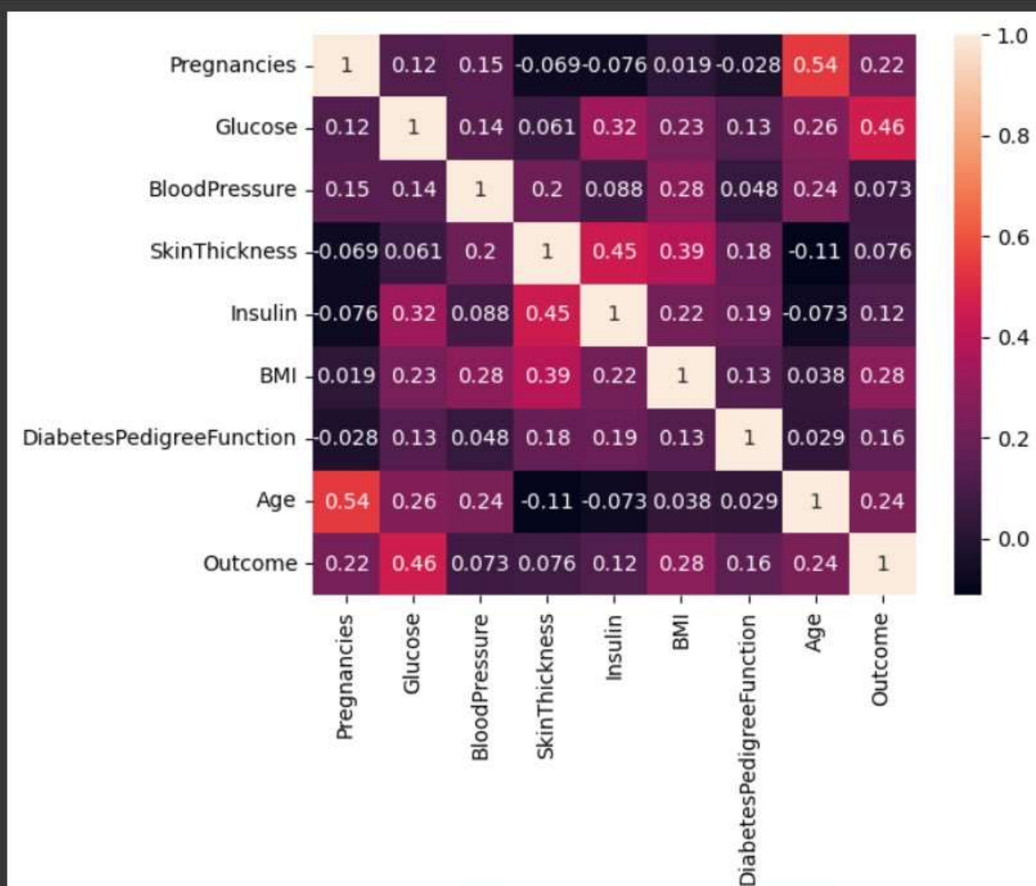
IT307MiniProject(Midsem)

Here in this plot means particularly we are doing pair plot,
We have the following legends where zero(non-diabetic) stands for a blue colour and one(diabetic) stands for Orange Color.



In this particular age vs pregnancy plot here we are interpreting how the diabetic and nondiabetic data is distributed when these two features are considered.

```
# Heatmap
sns.heatmap(df.corr(), annot = True)
plt.show()
```



This is the heatmap we got which says or interprets the level correlation between any two features.

For example If we take the case of outcome and glucose we see that we have a score of

IT307MiniProject(Midsem)

0.46 which means outcome is quite related to glucose level.

Moving on to data preprocessing part.

we are then replacing zeros with nan(not a number).
These are the number of nan counts in our data set.

```
[17] # Replacing NaN with mean values
     df_new["Glucose"].fillna(df_new["Glucose"].mean(), inplace = True)
     df_new["BloodPressure"].fillna(df_new["BloodPressure"].mean(), inplace = True
     df_new["SkinThickness"].fillna(df_new["SkinThickness"].mean(), inplace = True
     df_new["Insulin"].fillna(df_new["Insulin"].mean(), inplace = True)
     df_new["BMI"].fillna(df_new["BMI"].mean(), inplace = True)
```

shifts central tendency to mean. can introduce bias.

Here we're replacing the nan values with the mean values of that attribute which would possibly shift the central tendency of that attribute towards the mean and it also can introduce a bias .

```
df_new.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	2768.0	3.742775	3.323801	0.000	1.000	3.000000	6.00000	17.00
Glucose	2768.0	121.895273	30.500960	44.000	99.000	118.000000	141.00000	199.00
BloodPressure	2768.0	72.404086	11.988255	24.000	64.000	72.000000	80.00000	122.00
SkinThickness	2768.0	29.289634	9.031265	7.000	25.000	29.289634	32.00000	110.00
Insulin	2768.0	154.237830	81.678056	14.000	120.000	154.237830	154.23783	846.00
BMI	2768.0	32.596665	7.103424	18.200	27.575	32.400000	36.62500	80.60
DiabetesPedigreeFunction	2768.0	0.471193	0.325669	0.078	0.244	0.375000	0.62400	2.42
Age	2768.0	33.132225	11.777230	21.000	24.000	29.000000	40.00000	81.00
Outcome	2768.0	0.343931	0.475104	0.000	0.000	0.000000	1.00000	1.00

New values after our preprocessing.

We are then scaling our feature values to come in between zero and one using Min Max scaler.

```
df_new.isnull().sum()
```

```
Pregnancies          0
Glucose             18
BloodPressure      125
SkinThickness      800
```

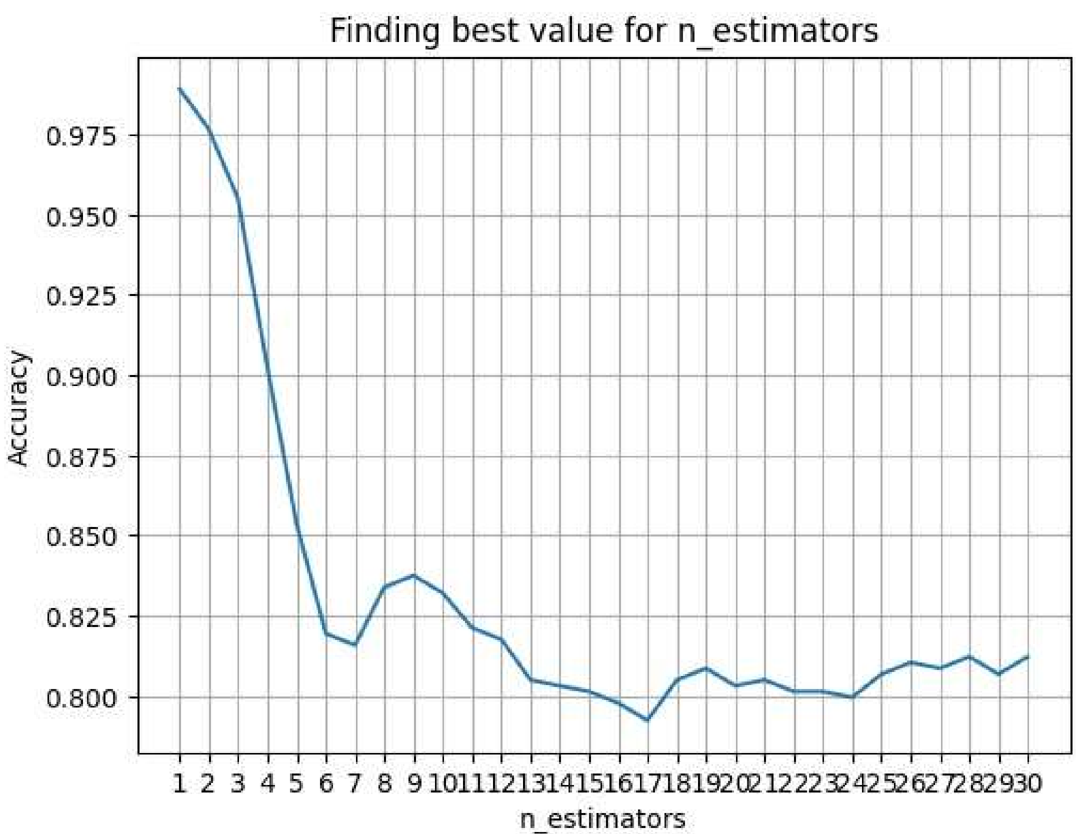
ittal

```
[19] # Feature scaling using MinMaxScaler
     from sklearn.preprocessing import MinMaxScaler
     sc = MinMaxScaler(feature_range = (0, 1))
     df_scaled = sc.fit_transform(df_new)
```

X_scaled = (X - X_min) / (X_max - X_min)

We then Split the data into 80 percent for training and 20 percent for testing.

Will then pass this data to multiple modelsso as to get them trained.
Description of each model is present in the Ipynb file.



It rain KNN we needed to search the best value for this we plotted the above visual plot and we get to know that when we are using only one neighbour we have very high accuracy but this could be very sensitive so we plan to chooseN estimators as around three or 4 which would be a balance between sensitivity and accuracy.

And now we have trained on 6 models and Now we are at a stage where we have evaluated one model out of 6.

And the code for the same looks like this.

```
# Evaluating using accuracy_score metric
from sklearn.metrics import accuracy_score
accuracy_logreg = accuracy_score(Y_test, Y_pred_logreg)
```

```
# Accuracy on test set
print("Logistic Regression: " + str(accuracy_logreg * 100))
```

```
Logistic Regression: 76.3537906137184
```

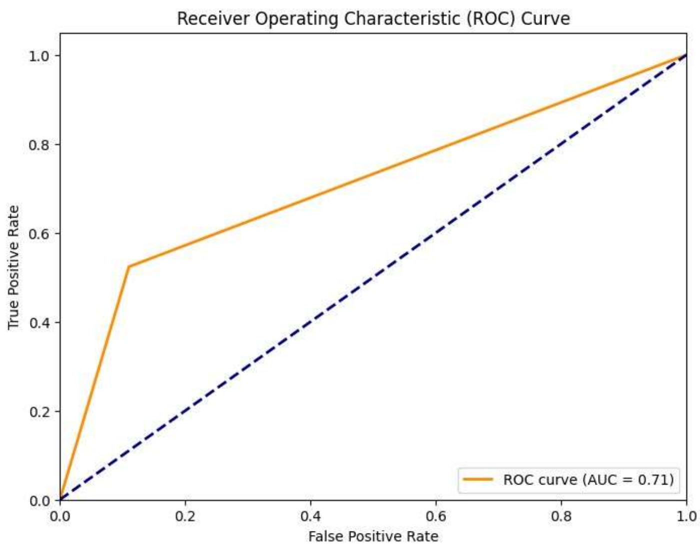
```
from sklearn.metrics import f1_score

#  F1-score
f1score_logreg = f1_score(Y_test, Y_pred_logreg)


print("F1-score:", f1score_logreg)
```

```
F1-score: 0.6042296072507553
```

We got a okay type result for accuracy and an average score for F1 score since this is a classification model we feel that F1 score could be a better evaluator in comparison to accuracy scores in future we are planning to do this for all and compare and evaluate all the models.



Here we have plotted a roc curve and it has a TPR on Y axis and a FPR on X axis.

TPR = TP / (TP + FN)

FPR = FP / (FP + TN)

to interpret this plot we get an ocean that it's above average performing model but not the best yet, we would better have knowledge of other models and then know where this model is standing.

Evaluating all models:

```
# Accuracy on test set
print("Logistic Regression: " + str(accuracy_logreg * 100))
print("K Nearest neighbors: " + str(accuracy_knn * 100))
print("Support Vector Classifier: " + str(accuracy_svc * 100))
print("Naive Bayes: " + str(accuracy_nb * 100))
print("Decision tree: " + str(accuracy_dectree * 100))
print("Random Forest: " + str(accuracy_ranfor * 100))
```

✓ 0.0s

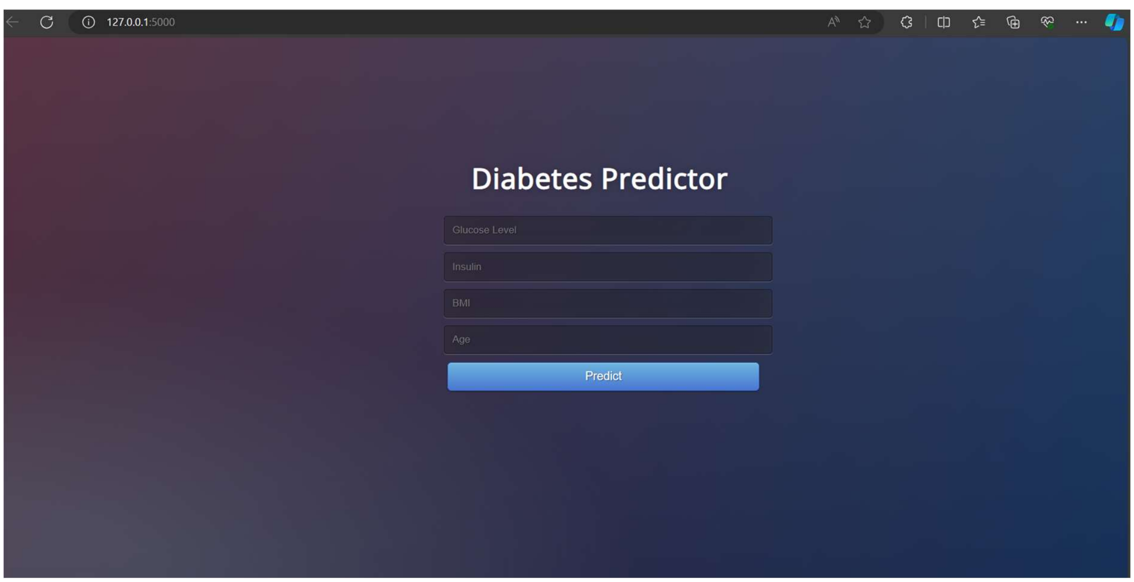
```
Logistic Regression: 76.3537906137184
K Nearest neighbors: 95.48736462093864
Support Vector Classifier: 76.17328519855594
Naive Bayes: 75.81227436823104
Decision tree: 99.45848375451264
Random Forest: 99.45848375451264
```

Here we got the accuracy of each and every model. By seeing those, we can say that decision tree and random forest models have high accuracy.

As we said earlier, we will build knn model and save it for diabetes prediction.

```
PS D:\5th sem\IT307\Diabetes\flask> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 965-031-154
127.0.0.1 - - [05/Nov/2023 17:01:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:01:31] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:01:35] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:01:35] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:03:55] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:03:55] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:04:24] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:04:24] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:04:58] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:04:58] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:05:21] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:05:21] "GET /static/css/style.css HTTP/1.1" 404 -
 * Detected change in 'D:\\5th sem\\IT307\\Diabetes\\flask\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 965-031-154
 * Detected change in 'D:\\5th sem\\IT307\\Diabetes\\flask\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 965-031-154
127.0.0.1 - - [05/Nov/2023 17:06:49] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:06:49] "GET /static/css/style.css HTTP/1.1" 404 -
127.0.0.1 - - [05/Nov/2023 17:07:22] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [05/Nov/2023 17:07:22] "GET /static/css/style.css HTTP/1.1" 404 -
 * Detected change in 'D:\\5th sem\\IT307\\Diabetes\\flask\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 965-031-154
```

This was before entering the values:



And this is the predicted one: