TEAM-16

Predicting Diabetes with Logistic Regression

Animesh Kumar-8
Payas S Nimje-32
Ruchita Singhal-42

Problem Statement - Predicting whether a person has diabetes or not by machine learning

Overview

We'll be using Machine Learning to predict whether a person has diabetes or not, based on information about the patient such as blood pressure, body mass index (BMI), age, In particular, all patients here belong to the Pima Indian heritage (subgroup of Native Americans), and are females of ages 21 and above.

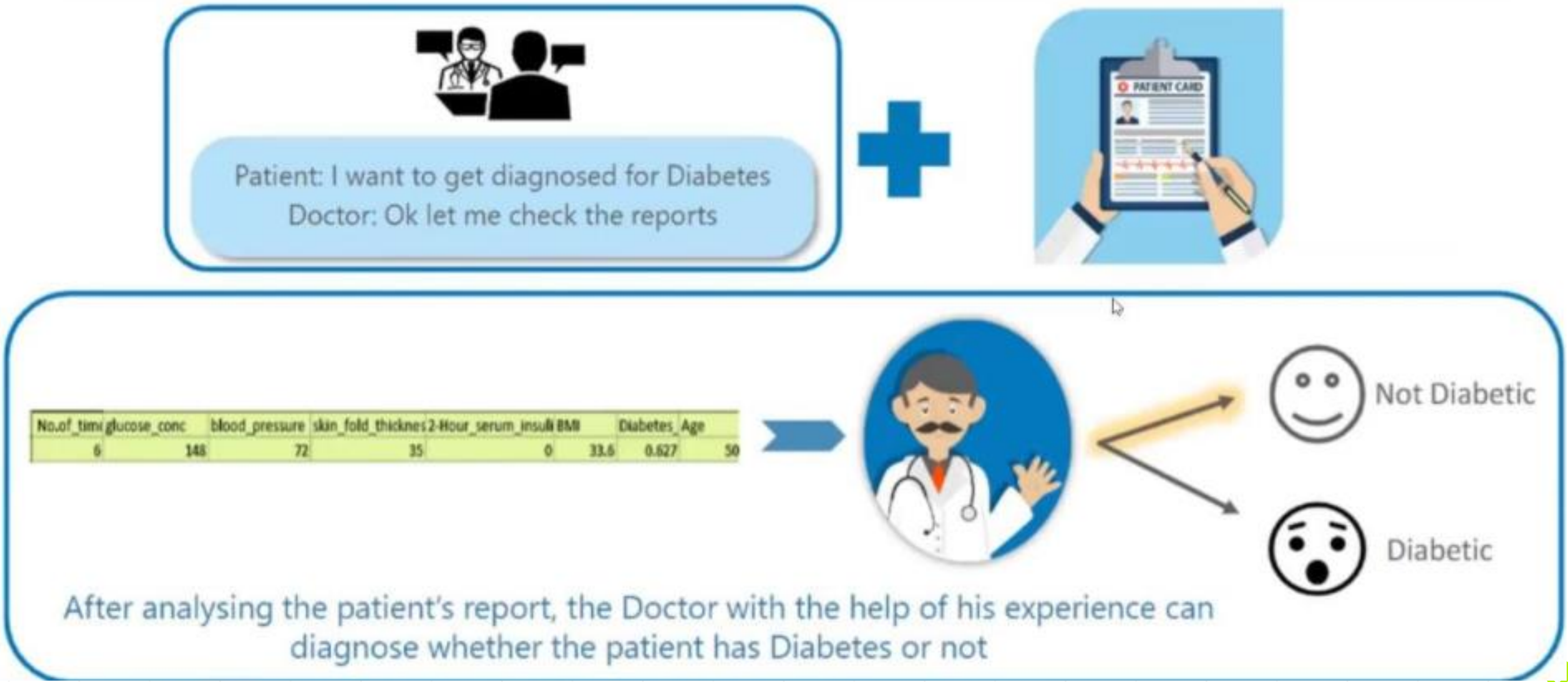
Method

We'll be using Python and some of its popular data science related packages. First of all, we will import pandas to read our data from a CSV file and manipulate it for further use. We will also use numpy to convert out data into a format suitable to feed our classification model. We'll use seaborn and matplotlib for visualizations. We will then import Logistic Regression algorithm from sklearn

We now want to train a Machine to do the Doctor's task.
For this purpose we have to train the machine with the same **experience/knowledge** using the historical data.

No.of_tim	glucose_conc	blood_pressure	skin_fold_thicknes	2-Hour_serum_insuli	BMI	Diabetes_	Age	Is_Diabetic
6	148	72	35	0	33.6	0.627	50	YES
1	85	66	29	0	26.6	0.351	31	NO
8	183	64	0	0	23.3	0.672	32	YES
1	89	66	23	94	28.1	0.167	21	NO
0	137	40	35	168	43.1	2.288	33	YES
5	116	74	0	0	25.6	0.201	30	NO
3	78	50	32	88	31	0.248	26	YES
10	115	0	0	0	35.3	0.134	29	NO
2	197	70	45	543	30.5	0.158	53	YES
8	125	96	0	0	0	0.232	54	YES
4	110	92	0	0	37.6	0.191	30	NO
10	168	74	0	0	38	0.537	34	YES
10	139	80	0	0	27.1	1.441	57	NO
1	189	60	23	846	30.1	0.398	59	YES
5	166	72	19	175	25.8	0.587	51	YES

Business Implication



No.of_timi	glucose_conc	blood_pressure	skin_fold_thicknes	2-Hour_serum_insuli	BMI	Diabetes_	Age
6	148	72	35	0	33.6	0.627	50

Data explanation

The following features have been provided to help us predict whether a person is diabetic or not

Pregnancies: Number of times pregnant
Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test
Blood Pressure: Diastolic blood pressure (mm Hg)
Skin Thickness: Triceps skin fold thickness (mm)
Insulin: 2-Hour serum insulin (mu U/ml)
BMI: Body mass index (weight in kg/(height in m)2)
DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
Age: Age (years)
Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

Data Description

```
In [3]: data=pd.read_csv('E:\MBA\TRIM-3\python-project\diabetes.csv')

In [7]: data
Out[7]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0
...	...	...	...	...	...	...	...	...	...
1995	2	75	64	24	55	29.7	0.370	33	0
1996	8	179	72	42	130	32.7	0.719	36	1
1997	6	85	78	0	0	31.2	0.382	42	0
1998	0	129	110	46	130	67.1	0.319	26	1
1999	2	81	72	15	76	30.1	0.547	25	0

2000 rows × 9 columns

We have our data saved in a CSV file called diabetes.csv. We first read our dataset into a Pandas data frame called diabetes, and then use the head() function to show the first five records from our dataset

Let’s check the missing values in our dataset and Dataypes

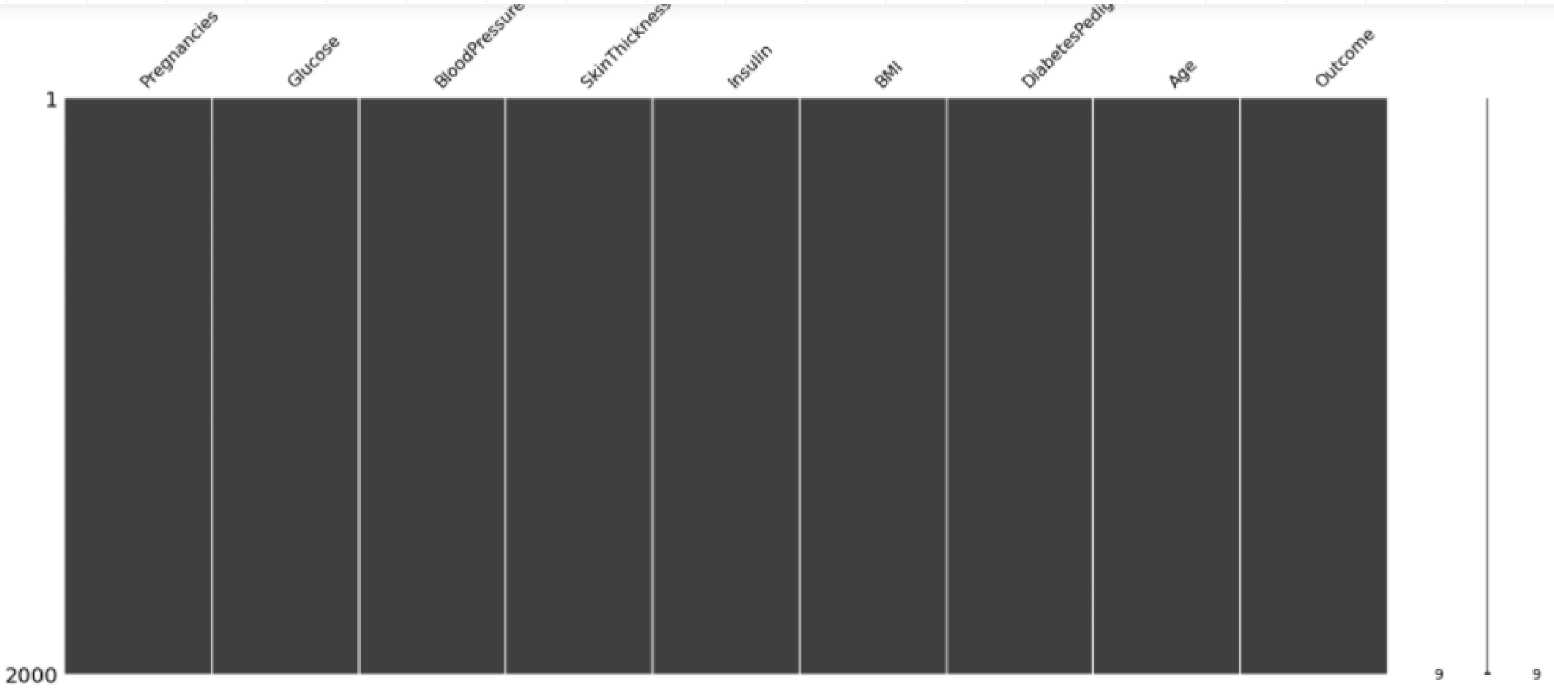
```
In [8]: df.isnull().sum()

Out[8]:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We don’t have any missing values

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2000 non-null   int64
 1   Glucose                   2000 non-null   int64
 2   BloodPressure             2000 non-null   int64
 3   SkinThickness             2000 non-null   int64
 4   Insulin                   2000 non-null   int64
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64
 8   Outcome                   2000 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

We have 7 features as integer and 2 in decimal form

Check the missing values Through Missingno Plot



Data Description

In [12]:
```
df.describe().T
```

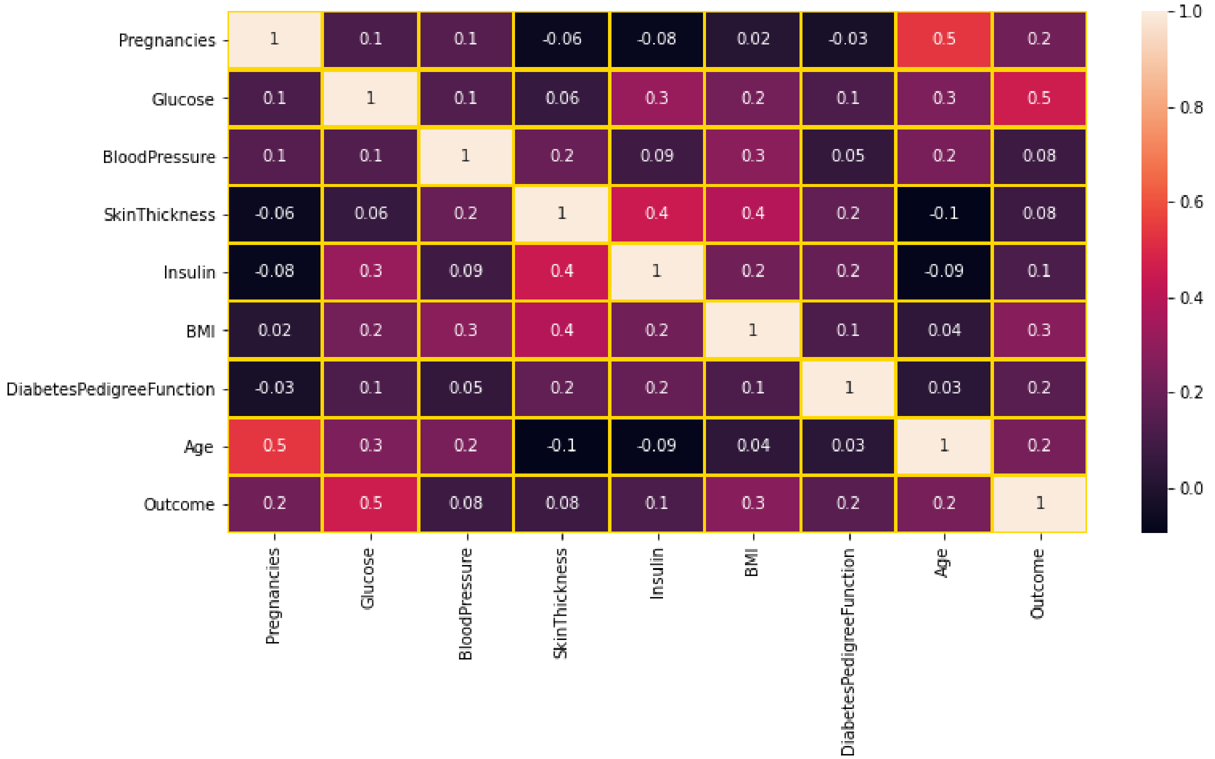
Out[12]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	2000.0	3.70350	3.306063	0.000	1.000	3.000	6.000	17.00
Glucose	2000.0	121.18250	32.068636	0.000	99.000	117.000	141.000	199.00
BloodPressure	2000.0	69.14550	19.188315	0.000	63.500	72.000	80.000	122.00
SkinThickness	2000.0	20.93500	16.103243	0.000	0.000	23.000	32.000	110.00
Insulin	2000.0	80.25400	111.180534	0.000	0.000	40.000	130.000	744.00
BMI	2000.0	32.19300	8.149901	0.000	27.375	32.300	36.800	80.60
DiabetesPedigreeFunction	2000.0	0.47093	0.323553	0.078	0.244	0.376	0.624	2.42
Age	2000.0	33.09050	11.786423	21.000	24.000	29.000	40.000	81.00
Outcome	2000.0	0.34200	0.474498	0.000	0.000	0.000	1.000	1.00

Exploratory Data Analysis

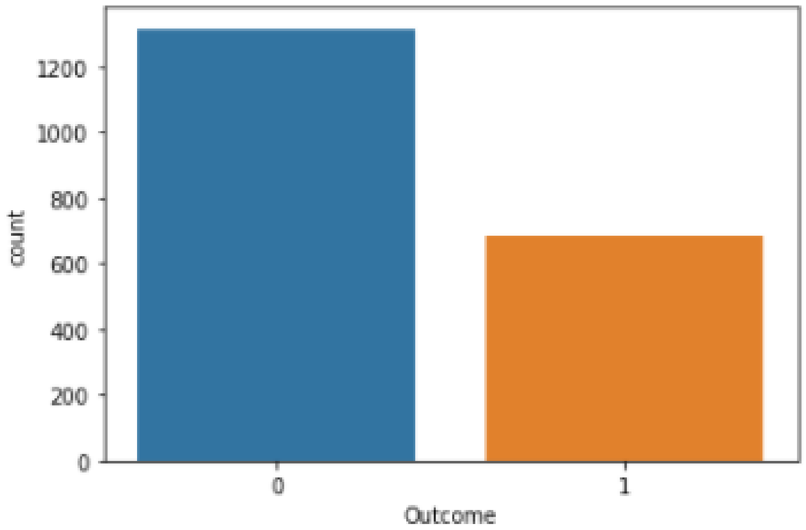
Correlation Matrix

As we can see from the table and the heat map, glucose levels, age, BMI have significant correlation with the outcome variable

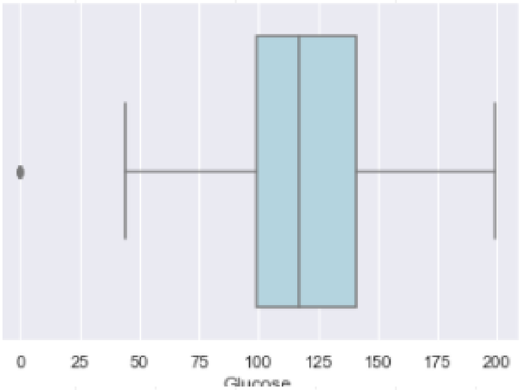


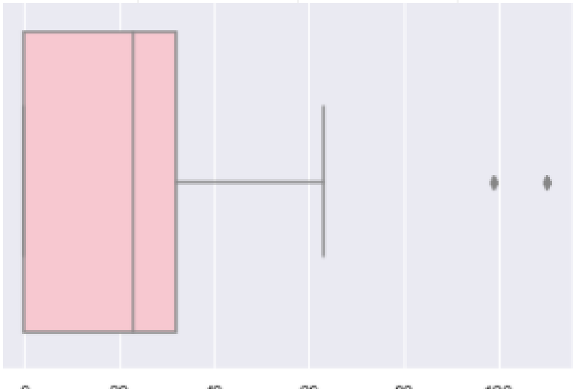
```
sns.countplot(x='Outcome',data=data)
```

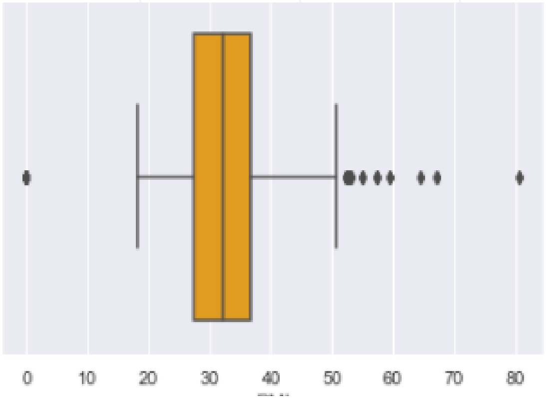
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```

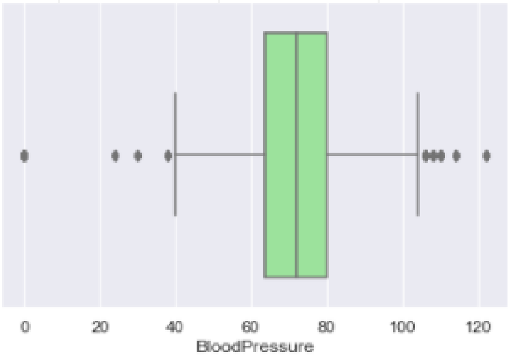


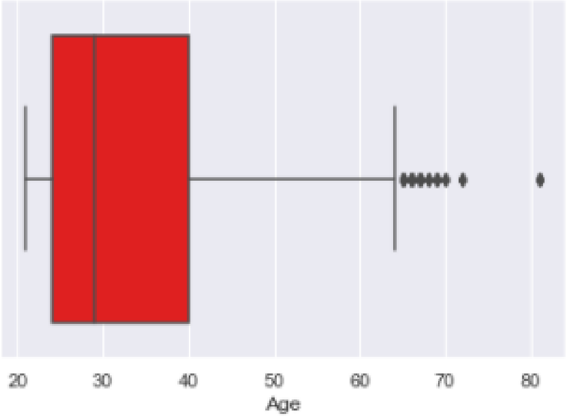
Check for outliers- the dots outside the boxplot shows the outliers

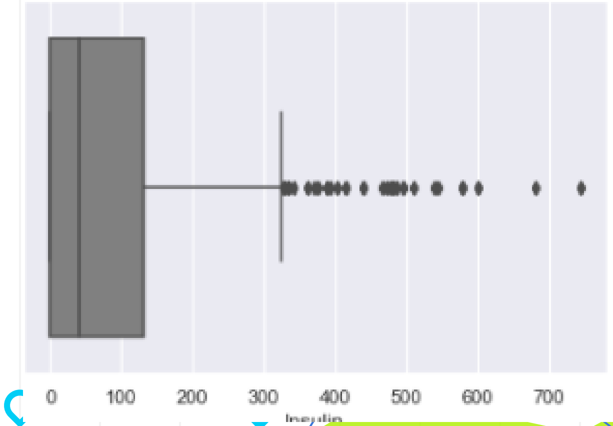










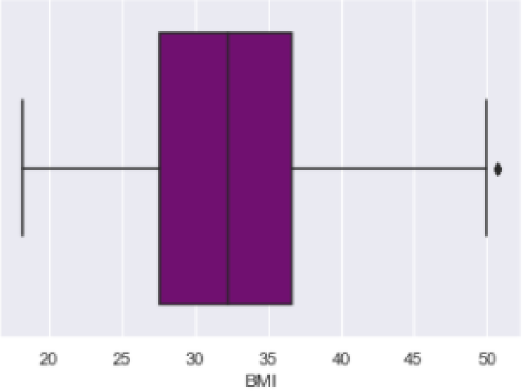


Treating Outliers- by use of Inter Quartile range

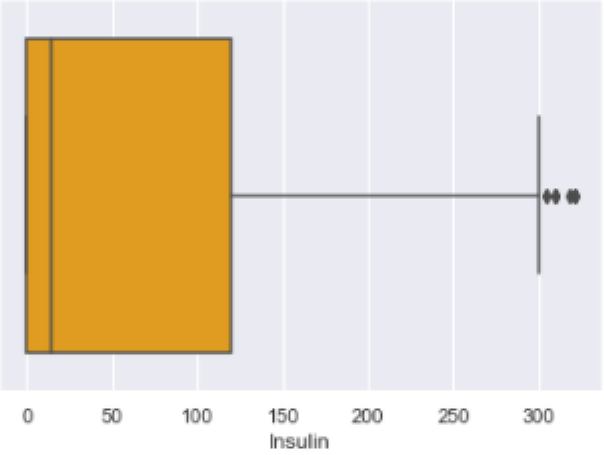
In [33]:

```
q3 = data['BMI'].quantile(.75)
q1 = data['BMI'].quantile(.25)
iqr = q3-q1
iqr
upperrange = q3+1.5*iqr
bottomrange = q1-1.5*iqr
data2 = data[(data['BMI']>bottomrange) & (data['BMI']<upperrange)]
sns.boxplot(data=data2,x='BMI',color = 'purple')
```

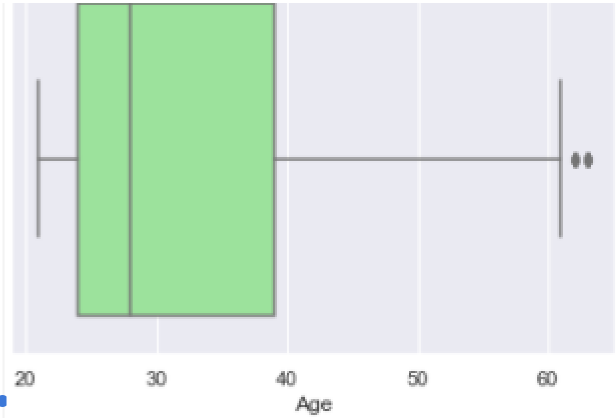
Out[33]: <AxesSubplot:xlabel='BMI'>



BMI



Insulin



Age

Analysis Through Mean Values grouped by Outcome(Likelihood)

In [15]:
```
df.groupby('Outcome').mean()
```

Out[15]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.168693	110.586626	68.094985	20.052432	70.563830	30.567477	0.434676	31.081307
1	4.732456	141.568713	71.166667	22.633041	98.897661	35.320468	0.540681	36.956140

Dataset Preparation (Normalization)

In [18]:
```
#Normalizing the data
means = np.mean(x, axis=0)
stds = np.std(x, axis=0)

X = (x - means)/stds
X
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	-0.515394	0.524553	-0.372481	0.873645	-0.722016	0.172683	-1.063246	1.180424
1	-1.120495	-1.159756	0.670080	0.625186	0.402563	0.737249	-0.735551	-0.856326
2	-1.120495	0.742890	-3.604422	-1.300374	-0.722016	1.473638	0.491759	-0.177409
3	-1.120495	0.430980	-0.059713	1.308449	1.527142	1.240448	-0.327478	-0.771462
4	-0.817945	0.555744	-0.372481	1.246334	3.596367	1.044077	0.201161	-1.026055
...	...	...	...	...	...	...	...	...
1995	-0.515394	-1.440474	-0.268225	0.190382	-0.227201	-0.305970	-0.312021	-0.007680
1996	1.299907	1.803381	0.148800	1.308449	0.447546	0.062225	0.766899	0.246914
1997	0.694807	-1.128565	0.461568	-1.300374	-0.722016	-0.121872	-0.274924	0.756101
1998	-1.120495	0.243835	2.129667	1.556908	0.447546	4.284191	-0.469686	-0.601732
1999	-0.515394	-1.253329	0.148800	-0.368651	-0.038272	-0.256877	0.235167	-0.686597

2000 rows × 8 columns

Segregating the data into Dependent and Independent variables

In [16]:
```
# Model Building

x = df.drop(['Outcome'], axis=1) #independent value
x
```

Out[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	2	138	62	35	0	33.6	0.127	47
1	0	84	82	31	125	38.2	0.233	23
2	0	145	0	0	0	44.2	0.630	31
3	0	135	68	42	250	42.3	0.365	24
4	1	139	62	41	480	40.7	0.536	21
...	...	...	...	...	...	...	...	...
1995	2	75	64	24	55	29.7	0.370	33
1996	8	179	72	42	130	32.7	0.719	36
1997	6	85	78	0	0	31.2	0.382	42
1998	0	129	110	46	130	67.1	0.319	26
1999	2	81	72	15	76	30.1	0.547	25

2000 rows × 8 columns

In [17]:
```
y = df.Outcome #dependent value
y
```

Out[17]:
```
0       1
1       0
2       1
3       1
4       0
       ..
1995    0
1996    1
1997    0
1998    1
1999    0
Name: Outcome, Length: 2000, dtype: int64
```

Splitting the data into training and testing set

In [19]:
```
#Test Train Split
from sklearn.model_selection import train_test_split, cross_val_score, ShuffleSplit, GridSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 365)
```

MODEL Comparision

In [20]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression().fit(X_train, y_train)
log_reg
```

Out[20]:
```
LogisticRegression()
```

Model Prediction

In [25]:
```
y_pred = log_reg.predict(X_test)
y_pred
```

Out[25]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0], dtype=int64)
```

Prediction, Model Accuracy, Confusion Matrix

In [27]:
```
print('Probability:', y_probs[:1])
print('Prediction:', y_pred[:1])
```

```
Probability: [[0.62466566 0.37533434]]
Prediction: [0]
```

In [28]:
```
from sklearn.metrics import accuracy_score
log_score = accuracy_score(y_test, y_pred)
print("Model accouracy = ", log_score * 100)
```

```
Model accouracy =  78.5
```

In [29]:
```
print('accuracy score of the model is',log_reg.score(X_test, y_test) * 100)
```

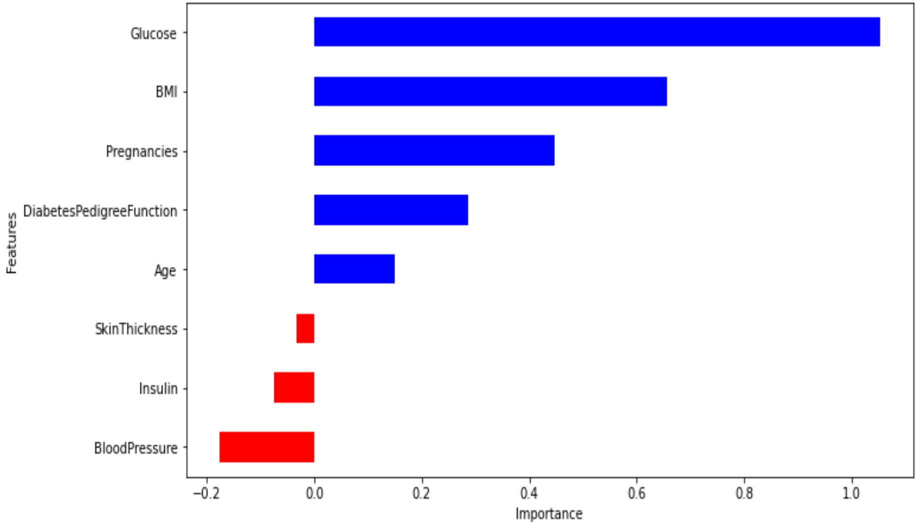
```
accuracy score of the model is 78.5
```

In [30]:
```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[30]:
```
array([[231,  21],
       [ 65,  83]], dtype=int64)
```

Variable Significance Level

Out[23]: Text(0.5, 0, 'Importance')

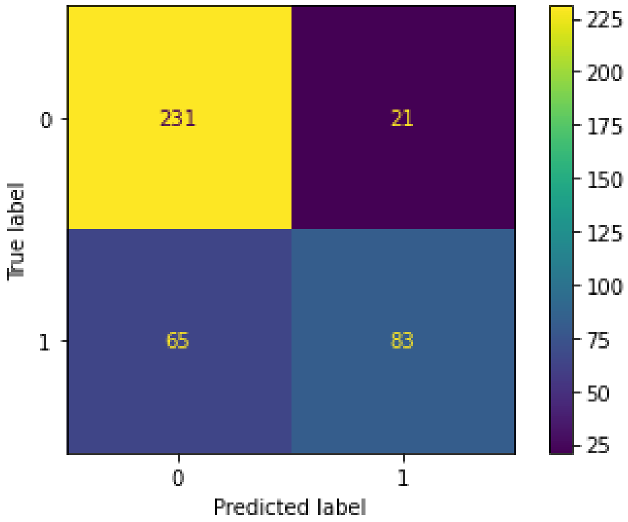



```
In [21]: coef = list(log_reg.coef_[0])
         coef

Out[21]: [0.4460220020404043,
          1.0516271413393867,
          -0.17702875388896225,
          -0.032704992570814835,
          -0.07551321077762954,
          0.6563619357224418,
          0.2858376434041782,
          0.14967169950086961]
```

Confusion Matrix

```
In [30]: plot_confusion_matrix(log_reg, X_test, y_test, values_format='.5g')
```

Out[30]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27d3b839c10>



Comparing Logistic Model With other Classification Models

Random Forest

Random forest

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
r_for = RandomForestClassifier().fit(X_train, y_train)
r_for
```

Out[31]:
```
RandomForestClassifier()
```

In [32]:
```
y_pred = r_for.predict(X_test)
y_pred[:20]
```

Out[32]:
```
array([1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1],
      dtype=int64)
```

In [33]:
```
rf_score = accuracy_score(y_test, y_pred)
rf_score
```

Out[33]:
```
0.9775
```

KNN

KNN

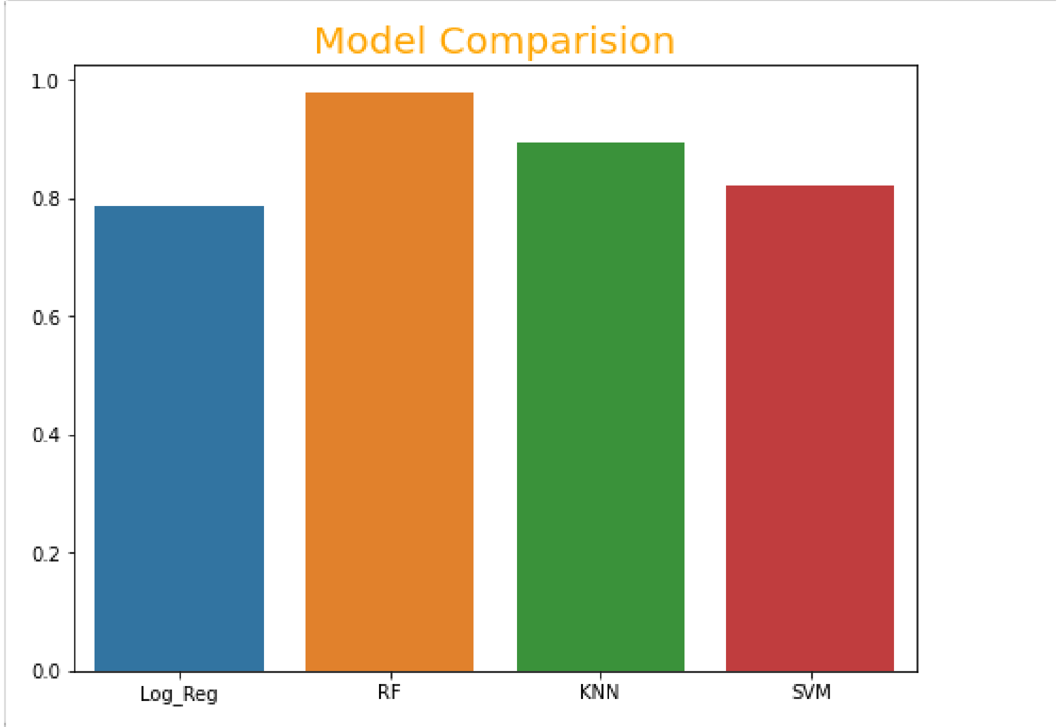
```
In [35]: from sklearn.neighbors import KNeighborsClassifier

In [36]: knn = KNeighborsClassifier(n_neighbors=3).fit(X_train,y_train)
         knn

Out[36]: KNeighborsClassifier(n_neighbors=3)

In [37]: y_pred = knn.predict(X_test)
         y_pred[:20]

Out[37]: array([0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1],
               dtype=int64)

In [38]: knn_score = accuracy_score(y_test,y_pred)
         knn_score

Out[38]: 0.8925

In [39]: confusion_matrix(y_test,y_pred)

Out[39]: array([[238,  14],
                [ 29, 119]], dtype=int64)
```

Support Vector Machine (SVM)

Support Vector Machine (SVM)

```
In [40]: from sklearn.svm import SVC

In [41]: svm_model = SVC(C=5,degree=9,kernel = 'poly').fit(X_train,y_train)
         svm_model

Out[41]: SVC(C=5, degree=9, kernel='poly')

In [42]: y_pred = svm_model.predict(X_test)
         y_pred[:20]

Out[42]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
               dtype=int64)

In [43]: svm_score = accuracy_score(y_test,y_pred)
         svm_score

Out[43]: 0.8225

In [44]: confusion_matrix(y_test,y_pred)

Out[44]: array([[250,   2],
                [ 69,  79]], dtype=int64)
```

Model Comparison



THANKS!

GitHub Link for the project.

https://github.com/Payas10/Python_Project

https://github.com/Animesh100-Kumar/Python-Project_Diabetes-Prediction

https://github.com/RuchitaSinghal/Python-Project