Diabetes Prediction using machine learning

Introduction

In this project, we will use the Pima Indians Dataset from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict diagnostically whether a patient has diabetes or not based on several medical predictor variables

Dataset Description:

The prediction of diabetes is based on the following variables:

- **Pregnancy:** This column represents the number of times pregnant.
- **Glucose:** This column represents the blood sugar level.
- **Blood Pressure:** This column represents diastolic blood pressure (mm Hg).
- **Skin Thickness:** This column represents triceps skin fold thickness (mm).
- **Insulin:** This column represents the insulin level
- **BMI:** This column represents the Body Mass Index.
- **Diabetes Pedigree:** This column represents the diabetes pedigree function.
- **Age:** This column represents the age of the person in years.
- **Outcome:** This column indicates the presence or absence of diabetes.

import libraries:

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

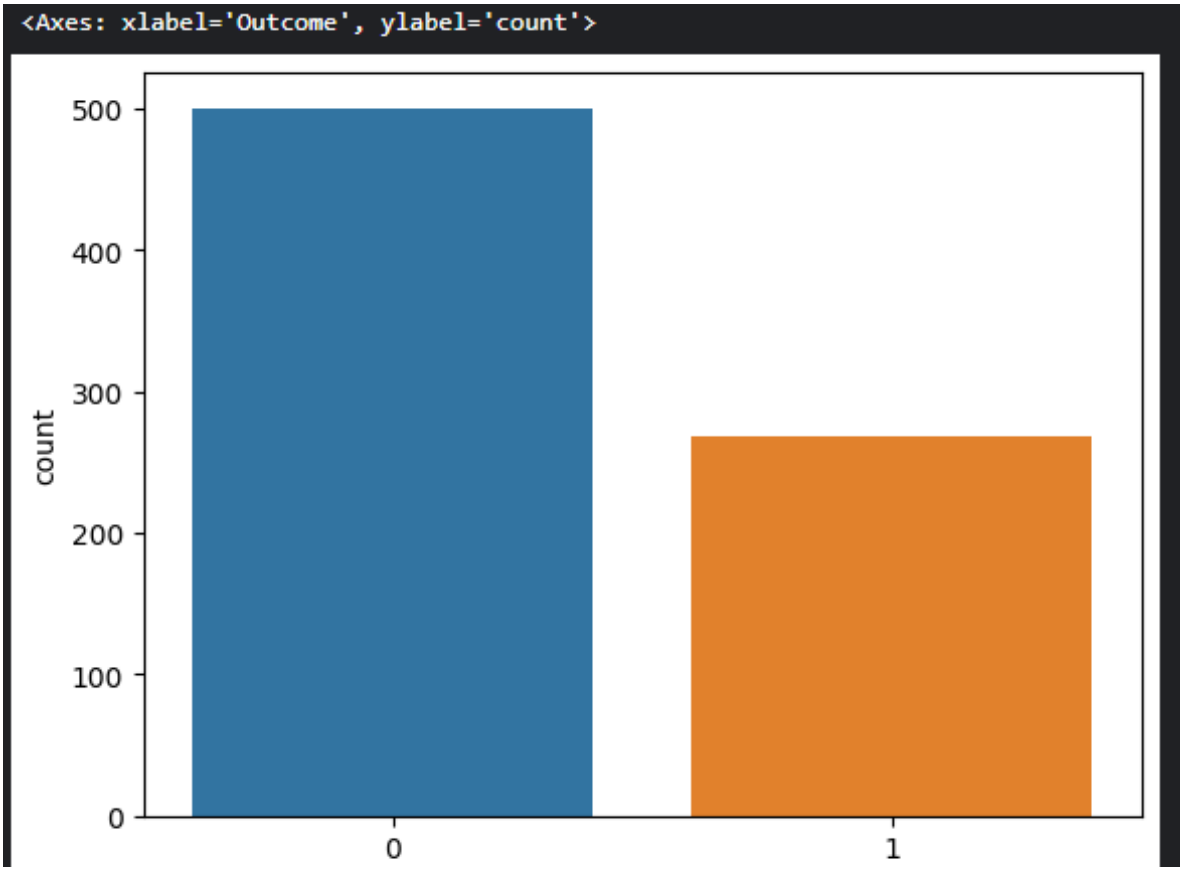
```
df=pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
```

Data Visualization:

The bar graph below shows that there are 268 cases of diabetes and 500 cases of non-diabetic.

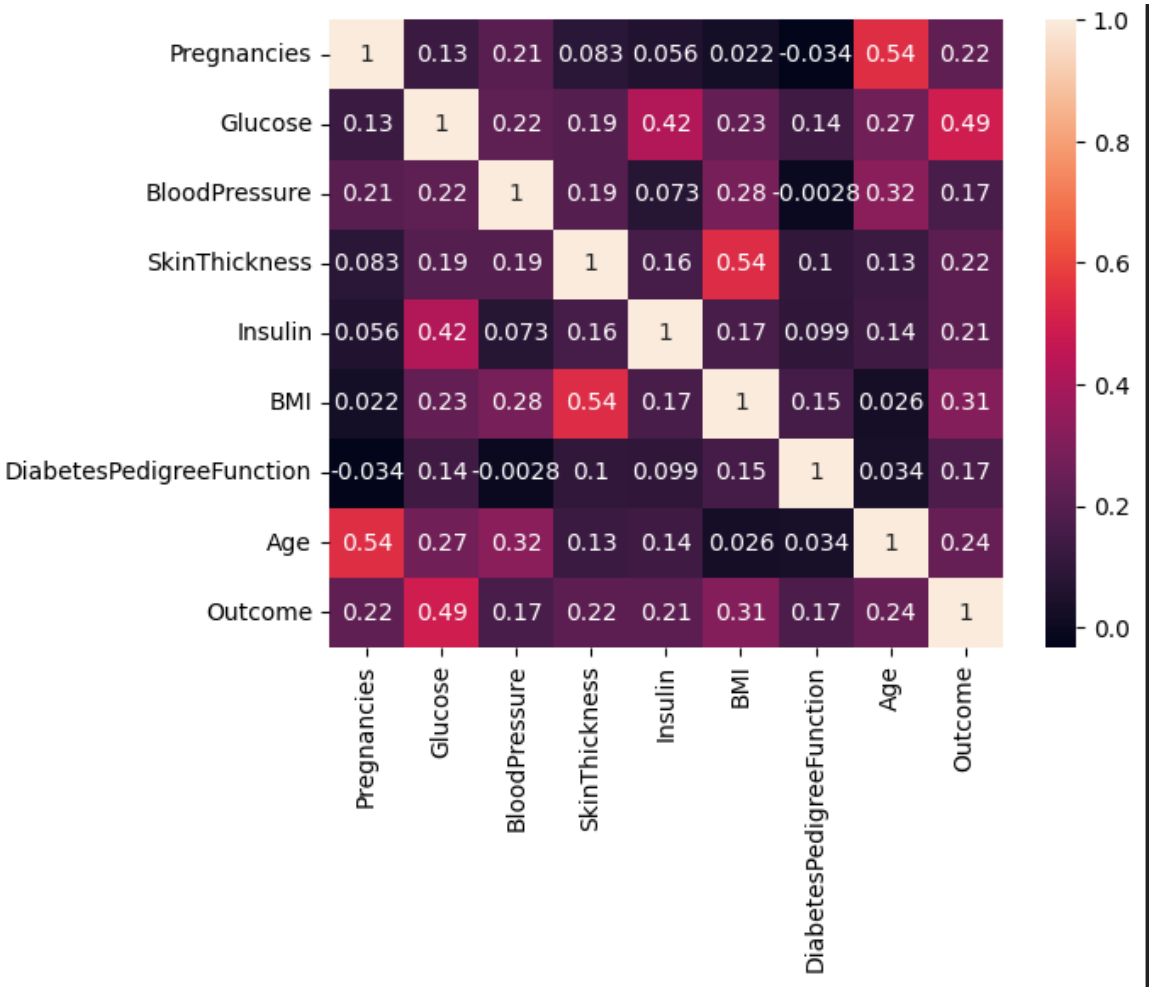
```
# Outcome countplot
sns.countplot(x = 'Outcome',data = df)
```

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



From the correlation heatmap, it's evident that there is a strong correlation between diabetes and the features: Glucose, BMI, Age, and Insulin. Considering these strong correlations, we can select these features to accept input from the user and predict the outcome.

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



Data preprocessing:

Checking for null values:

We can observe that certain features such as Glucose, Blood pressure, Insulin, and BMI contain zero values, which typically represent missing data.

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

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
```

We replaced the zero values with NaN (Not a Number) to denote missing data.

```
# Replacing zero values with NaN

df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)
```

```
# Count of NaN
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

We have replaced the NaN values with the mean.

```
# Replacing NaN with mean values
df["Glucose"] = df["Glucose"].fillna(df["Glucose"].mean())
df["BloodPressure"] = df["BloodPressure"].fillna(df["BloodPressure"].mean())
df["SkinThickness"] = df["SkinThickness"].fillna(df["SkinThickness"].mean())
df["Insulin"] = df["Insulin"].fillna(df["Insulin"].mean())
df["BMI"] = df["BMI"].fillna(df["BMI"].mean())
```

We apply the MinMaxScaler to scale the features of a dataset, ensuring that all features are within the specified range of 0 to 1

```
# Feature scaling using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
dataset_scaled = sc.fit_transform(df)
dataset_scaled = pd.DataFrame(dataset_scaled)
```

```
dataset_scaled = pd.DataFrame(dataset_scaled)
```

Glucose, Insulin, Age, and BMI exhibit strong correlations with the outcome, as indicated by the feature correlation heatmap. So, we select these features as X and the outcome as Y

```
# Selecting features - [Glucose, Insulin, BMI, Age]
X = dataset_scaled.iloc[:, [1, 4, 5, 7]].values
Y = dataset_scaled.iloc[:, 8].values
```

The dataset is subsequently divided using train_test_split with an 80:20 ratio.

```
# Splitting X and Y
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 42, stratify = df['Outcome'] )
```

Data Modeling:

For modeling, we employ three models: logistic regression, K-nearest neighbors, and random forest

Random Forest:

```
# Logistic Regression Algorithm
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state = 42)
logreg.fit(X_train, Y_train)
Y_pred_logreg = logreg.predict(X_test)
```

KNN:

```
# K nearest neighbors Algorithm
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 24, metric = 'minkowski', p = 2)
knn.fit(X_train, Y_train)
Y_pred_knn = knn.predict(X_test)
```

Random Forest:

```
# Random forest Algorithm
from sklearn.ensemble import RandomForestClassifier
randFor = RandomForestClassifier(n_estimators = 11, criterion = 'entropy', random_state = 42)
randFor.fit(X_train, Y_train)
Y_pred_randFor = ranfor.predict(X_test)
```

Model Evaluation

We use the accuracy metric to evaluate the performance of our models.

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

K Nearest Neighbors achieved the highest accuracy score, followed by Random Forest, and then Logistic Regression

```
# Accuracy on test set
print("Logistic Regression: " + str(accuracy_logreg * 100))
print("K Nearest neighbors: " + str(accuracy_knn * 100))
print("Random Forest: " + str(accuracy_randFor * 100))
```

```
Logistic Regression: 72.07792207792207
K Nearest neighbors: 78.57142857142857
Random Forest: 75.97402597402598
```

the confusion matrix represents :

- True Negative (TN): 87 indicates the number of instances correctly classified as non-diabetic
- True postive (TP): 34 indicates the number of instances correctly classified as diabetic
- False Positive (FP): 13 indicates the number of instances incorrectly classified as diabetic when they are actually non-diabetic

- False Negative(FN) : 20 indicates the number of instances incorrectly classified as non-diabetic when they are actually diabetic