Description

The dataset contains several medical predictor (Independent) variables and one target variable, (Outcome). Predictor variables include:

1. Pregnancies
2. Glucose
3. BloodPressure
4. SkinThickness
5. Insulin
6. BMI
7. DiabetesPedigreeFunction
8. Age

Dataset url: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

1) Pregnancies: Number of times pregnant

2) Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

3) BloodPressure: Diastolic blood pressure (mm Hg)

4) SkinThickness: Triceps skin fold thickness (mm)

5) Insulin: 2-Hour serum insulin (mu U/ml)

6) BMI: Body mass index (weight in kg/(height in m)^2)

7) DiabetesPedigreeFunction: Diabetes pedigree function

8) Age: Age (years)

9) Outcome: Class variable (0 or 1)

Step 1: Importing Libraries

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

Step 2: Load the Dataset

```
diabetes_dataset = pd.read_csv('diabetes.csv')
```

Step 3: Exploratory Data Analysis

Exploratory Data Analysis (EDA), also known as Data Exploration, is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

3.1) Understanding Your Variables

3.1.1) Head of the dataset
3.1.2) The shape of the dataset
3.1.3) List types of columns
3.1.4) Info of the dataset
3.1.5) Summary of the dataset

3.1.1) Head of the Dataset

```
# Display first five records
diabetes_dataset.head()

# Display last five records
diabetes_dataset.tail()

# Disply random records
diabetes_dataset.sample(5)
```

3.1.2) The Shape of Dataset

```
 # Numbers of rows and columns
diabetes_dataset.shape
```

3.1.3) List types of columns

```
# List types of all columns
diabetes_dataset.dtypes
```

3.1.4) Info of the Dataset

```
# Checking for null values
diabetes_dataset.info()

# Statistical Summary
diabetes_dataset.describe()
```

Step 4: Split the data frame in X & Y

```
target_name = 'Outcome'

# Separate object for target feature
y = diabetes_dataset[target_name]

# Separate obhect for input feature
X = diabetes_dataset.drop(target_name,axis=1)
```

```
X.head()
```

```
y.head()
```

Step 5: Apply Feature Scaling

Various Data Scaling Techniques:

1. Normalizer
2. MinMax Scaler
3. Binarizer
4. Standard Scaler

```
# Apply Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
SSX = scaler.fit_transform(X)
```

Step 6: Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(SSX,y,test_size=0.2,random_state=2)
```

```
X_train.shape,y_train.shape
```

Step 7: Build CLASSIFICATION Algorithms

8.1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(X_train,y_train)
```

8.2) K-Nearest Neighbors Classifier(KNN)

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
```

8.3) Naive-Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
```

8.4) Support Vector Machine (SVM)

```
from sklearn.svm import SVC
sv = SVC(kernel='linear')
sv.fit(X_train,y_train)
```

8.5) Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

8.6) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(criterion='entropy')
rf.fit(X_train,y_train)
```

Step 8: Making Prediction

9.1) Making Prediction using Logistic Regression

```
lr_pred = lr.predict(X_test)
```

9.2) Making Prediction using KNN

```
knn_pred = knn.predict(X_test)
```

9.3) Making Prediction using Naive Bayes

```
nb_pred = nb.predict(X_test)
```

9.4) Making Prediction using SVM

```
sv_pred = sv.predict(X_test)
```

9.5) Making Prediction using Decision Tree

```
dt_pred = dt.predict(X_test)
```

9.6) Making Prediciton using Random Forest

```
rf_pred = rf.predict(X_test)
```

Step 9: Model Evaluation

10.1) Train & Test Scores

```
from sklearn.metrics import accuracy_score

# Train & Test Scores of Logistic Regression
print("Accuracy (Train) score of Logistic Regression
",lr.score(X_train,y_train)*100)
print("Accuracy (Test) score of Logistic Regression ",
lr.score(X_test,y_test)*100)
print("Accuracy score of Logistic Regression ",
accuracy_score(y_test,lr_pred)*100)

# Train & Test Scores of KNN
print("Accuracy (Train) score of KNN ",knn.score(X_train,y_train)*100)
```

```
print("Accuracy (Test) score of KNN ", knn.score(X_test,y_test)*100)
print("Accuracy score of KNN ", accuracy_score(y_test,knn_pred)*100)

# Train & Test Scores of Naive-Bayes
print("Accuracy (Train) score of Naive Bayes ",nb.score(X_train,y_train)*100)
print("Accuracy (Test) score of Naive Bayes ", nb.score(X_test,y_test)*100)
print("Accuracy score of Naive Bayes ", accuracy_score(y_test,nb_pred)*100)

# Train & Test Scores of SVM
print("Accuracy (Train) score of SVM ",sv.score(X_train,y_train)*100)
print("Accuracy (Test) score of SVM ", sv.score(X_test,y_test)*100)
print("Accuracy score of SVM ", accuracy_score(y_test,sv_pred)*100)

# Train & Test Scores of Decision Tree
print("Accuracy (Train) score of Decision Tree
",dt.score(X_train,y_train)*100)
print("Accuracy (Test) score of Decision Tree ", dt.score(X_test,y_test)*100)
print("Accuracy score of Decision Tree ", accuracy_score(y_test,dt_pred)*100)

# Train & Test Scores of Random Forest
print("Accuracy (Train) score of Random Forest
",rf.score(X_train,y_train)*100)
print("Accuracy (Test) score of Random Forest ", rf.score(X_test,y_test)*100)
print("Accuracy score of Random Forest ", accuracy_score(y_test,rf_pred)*100)
```

Step 10: Making a prediction System

```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = lr.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

Step 11: Saving the trained Model

```
import pickle
filename = 'diabetes_model.sav'
pickle.dump(lr,open(filename,'wb'))
```