Importing the necessary libraries.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings("ignore")
```

Data Collection and Analysis

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

Displaying the first 5 rows of dataframe

```
diabetes_data.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Displaying the last 5 rows of dataframe

```
diabetes_data.tail()
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
763           10      101             76             48      180  32.9
```

```
764            2      122              70              27        0  36.8

765            5      121              72              23      112  26.2

766            1      126              60               0        0  30.1

767            1       93              70              31        0  30.4


     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

No. of rows and columns

```
shape = diabetes_data.shape
print("Rows",shape[0])
print("Columns",shape[1])

Rows 768
Columns 9

diabetes_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Check for missing values

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

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Check for duplicate values

```
diabetes_data.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
763    False
764    False
765    False
766    False
767    False
Length: 768, dtype: bool
```

```
diabetes_data.duplicated().sum()
```

```
0
```

Statistical measures of data

```
diabetes_data.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  120.894531      69.105469      20.536458
79.799479
std       3.369578   31.972618      19.355807      15.952218
115.244002
min       0.000000    0.000000       0.000000       0.000000
0.000000
25%       1.000000   99.000000      62.000000       0.000000
0.000000
50%       3.000000  117.000000      72.000000      23.000000
30.500000
```

```
75%       6.000000  140.250000      80.000000      32.000000
127.250000
max      17.000000  199.000000     122.000000      99.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

```
diabetes_data['Outcome'].value_counts()
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

0 -> Non-Diabetic 1 -> Diabetic

```
diabetes_data.groupby('Outcome').mean()
```

```
         Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
Outcome

0           3.298000  109.980000      68.184000      19.664000
68.792000
1           4.865672  141.257463      70.824627      22.164179
100.335821

                BMI  DiabetesPedigreeFunction        Age
Outcome
0         30.304200                  0.429734  31.190000
1         35.142537                  0.550500  37.067164
```

Separate the dataset into independent and dependent variable

```
X = diabetes_data.drop(labels='Outcome',axis=1)
y = diabetes_data['Outcome']

print(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
0              6      148             72             35        0  33.6
```

```
1                1       85              66              29         0  26.6
2                8      183              64               0         0  23.3
3                1       89              66              23        94  28.1
4                0      137              40              35       168  43.1
..             ...      ...             ...             ...       ...   ...
763             10      101              76              48       180  32.9
764              2      122              70              27         0  36.8
765              5      121              72              23       112  26.2
766              1      126              60               0         0  30.1
767              1       93              70              31         0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

```
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

```
scaler = StandardScaler()

standardized_data = scaler.fit_transform(X)

print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

```
X = standardized_data
```

Train Test Split

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,stratify=y,random_state=2)

print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the model

1. Logistic Regression

```
reg = LogisticRegression()

reg.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
X_test_pred = reg.predict(X_test)

accuracy = accuracy_score(y_test, X_test_pred)
print("Accuracy Score is:",accuracy)

print()
print("Report")
report = classification_report(y_test,X_test_pred)
print(report)
```

```
Accuracy Score is: 0.7597402597402597

Report
              precision    recall  f1-score   support

           0       0.77      0.89      0.83       100
           1       0.72      0.52      0.60        54

    accuracy                           0.76       154
   macro avg       0.75      0.70      0.72       154
weighted avg       0.75      0.76      0.75       154
```

1. Decision Tree Classifier

```
DTC = DecisionTreeClassifier()

DTC.fit(X_train, y_train)

DecisionTreeClassifier()

X_test_pred = DTC.predict(X_test)

accuracy = accuracy_score(y_test, X_test_pred)
print("Accuracy Score is:",accuracy)

print()
print("Report")
report = classification_report(y_test,X_test_pred)
print(report)

Accuracy Score is: 0.7077922077922078

Report
              precision    recall  f1-score   support

           0       0.75      0.83      0.79       100
           1       0.60      0.48      0.54        54

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.66       154
weighted avg       0.70      0.71      0.70       154
```

1. Support Vector Classifier

```
classifier = SVC()

classifier.fit(X_train, y_train)

SVC()
```

```
X_test_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, X_test_pred)
print("Accuracy Score is:",accuracy)

print()
print("Report")
report = classification_report(y_test,X_test_pred)
print(report)
```

```
Accuracy Score is: 0.7272727272727273

Report
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       100
           1       0.67      0.44      0.53        54

    accuracy                           0.73       154
   macro avg       0.71      0.66      0.67       154
weighted avg       0.72      0.73      0.71       154
```

1. Random Forest Classifier

```
RFC = RandomForestClassifier(n_estimators=100)

RFC.fit(X_train,y_train)

RandomForestClassifier()

X_test_pred = RFC.predict(X_test)

accuracy = accuracy_score(y_test, X_test_pred)
print("Accuracy Score is:",accuracy)

print()
print("Report")
report = classification_report(y_test,X_test_pred)
print(report)
```

```
Accuracy Score is: 0.7467532467532467

Report
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       100
           1       0.68      0.52      0.59        54

    accuracy                           0.75       154
   macro avg       0.73      0.69      0.70       154
```

```
weighted avg       0.74      0.75      0.74       154
```

As the Logistic Regression is predicting more accurately so we use it for creating the predictive system

Creating a predictive system

```
input_data = [4,110,92,0,0,37.6,0.191,30] # y = 0

# Convert the list to a numpy array for easy manipulation
input_data = np.array(input_data)

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

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

prediction = reg.predict(std_data)

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

```
The person is not diabetic
```