Fatima Jinnah Women University

Department Of Software Engineering

PROJECT

Course Title

Machine Learning

Submitted To

Engr. Sidra Ejaz

Submitted By

Mariam Fatima

Section: A (6TH Semester)

Registration No: 2021-BSE-020

Date of Submission

June 04, 2024

DIABETES PREDICTION SYSTEM

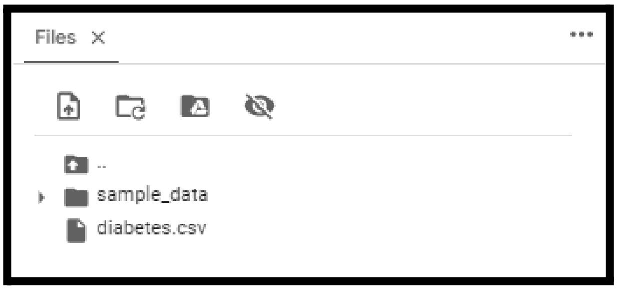
A diabetes dataset typically refers to a collection of structured information or data points related to diabetes, a chronic medical condition characterized by elevated levels of blood sugar (glucose). These datasets are commonly used in research, healthcare analytics, and machine learning applications to better understand disease, develop predictive models, and improve patient care.

By using this dataset (labeled), we have implemented all the data analysis steps on Google Colab using Python Language and, in the end, we have generated a front end of our prediction system using HTML, CSS and Django.

The system takes some values as an input and then predicts the result whether the user is diabetic or not. All the steps are explained below:

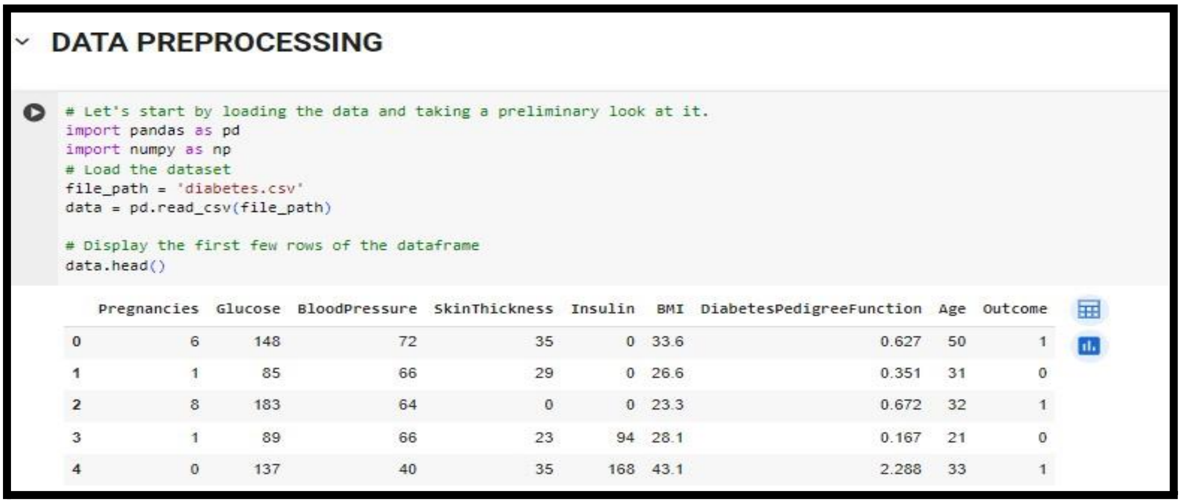
STEP 01

Upload the diabetes.csv file on your Colab Notebook to perform data analysis.



STEP 02

The most initial step is Data Preprocessing. In this step we have imported the NumPy and Pandas Libraries to access all the required elements of the data analysis procedure further on. Then the csv file is being read. And the first five rows of dataset are displayed.



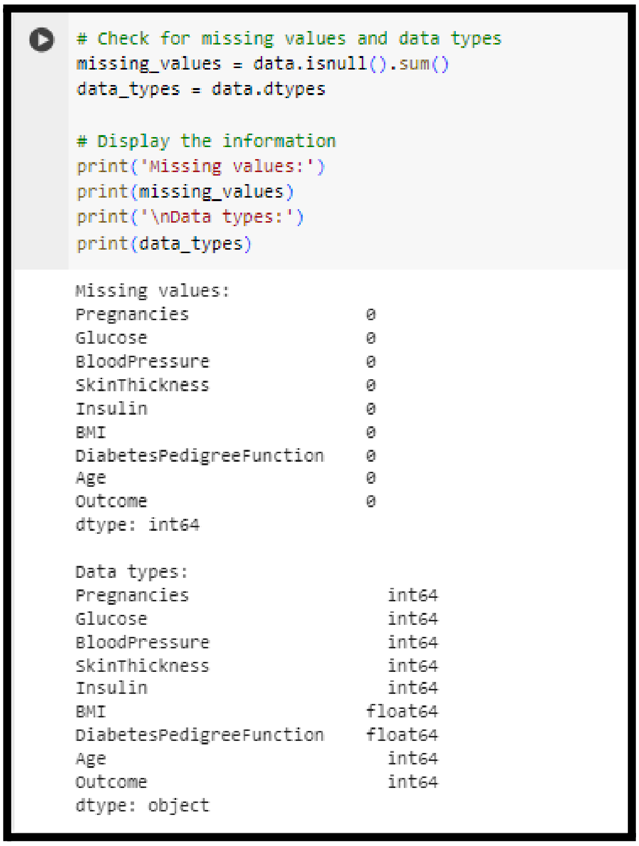
DATA PREPROCESSING

```
# Let's start by loading the data and taking a preliminary look at it.
import pandas as pd
import numpy as np
# Load the dataset
file_path = 'diabetes.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
data.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

STEP 03

This code checks for missing values and displays the data types in the 'data' DataFrame, printing the counts of missing values and data types for each column.

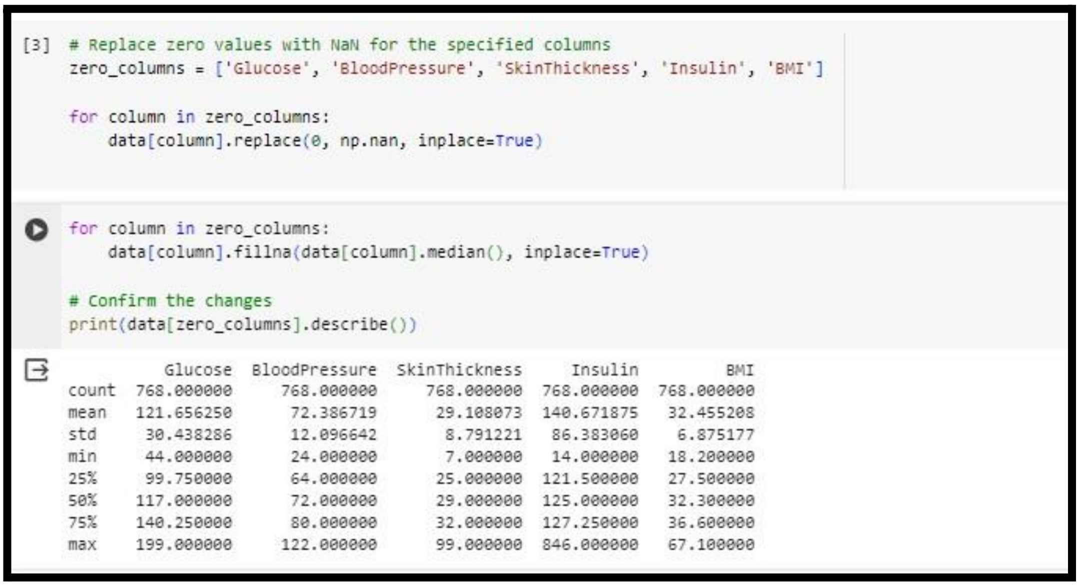


```
# Check for missing values and data types
missing_values = data.isnull().sum()
data_types = data.dtypes

# Display the information
print('Missing values:')
print(missing_values)
print('\nData types:')
print(data_types)
```

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

Data types:
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

STEP 04

This code iterates through specified columns ('Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI') in the 'data' DataFrame and replaces zero values with NaN. Second part of this code fills missing (NaN) values in the specified columns ('Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI') of the 'data' DataFrame with their respective column medians and then prints descriptive statistics for those columns to confirm the changes.



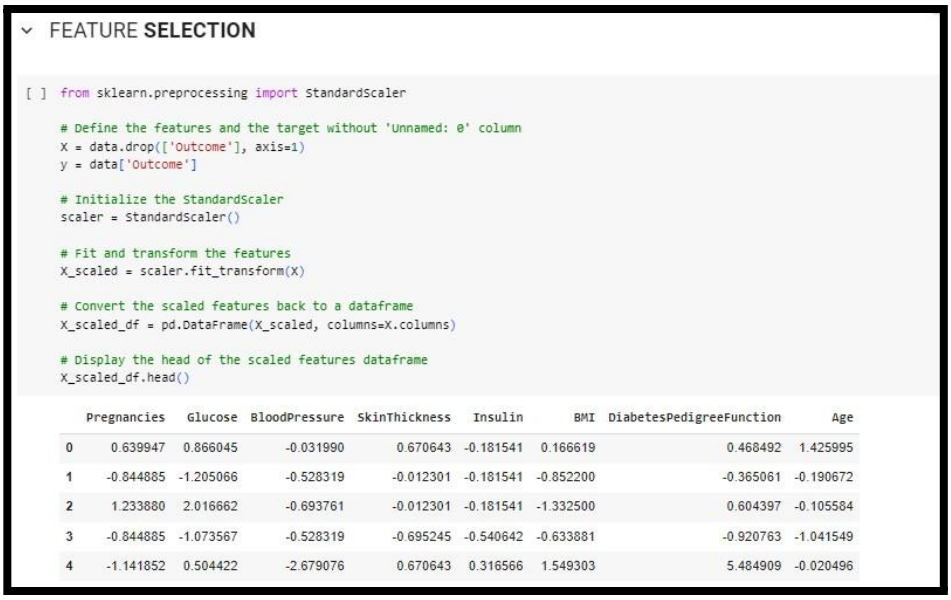
```
[3] # Replace zero values with NaN for the specified columns
    zero_columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

    for column in zero_columns:
        data[column].replace(0, np.nan, inplace=True)
```

```
for column in zero_columns:
    data[column].fillna(data[column].median(), inplace=True)

# Confirm the changes
print(data[zero_columns].describe())
```

```
          Glucose  BloodPressure  SkinThickness     Insulin         BMI
count  768.000000     768.000000     768.000000  768.000000  768.000000
mean   121.656250      72.386719      29.108073  140.671875   32.455208
std     30.438286      12.096642       8.791221   86.383060    6.875177
min     44.000000      24.000000       7.000000   14.000000   18.200000
25%     99.750000      64.000000      25.000000  121.500000   27.500000
50%    117.000000      72.000000      29.000000  125.000000   32.300000
75%    140.250000      80.000000      32.000000  127.250000   36.600000
max    199.000000     122.000000      99.000000  846.000000   67.100000
```

STEP 05

This code uses the StandardScaler from scikit-learn to standardize the features in the 'data' DataFrame. It defines the features (X) and the target variable ('Outcome' stored in y), removes the 'Outcome' column from features, initializes the StandardScaler, fits it to the features, transforms the features, converts the scaled features back to a DataFrame, and finally, displays the head of the scaled features DataFrame.



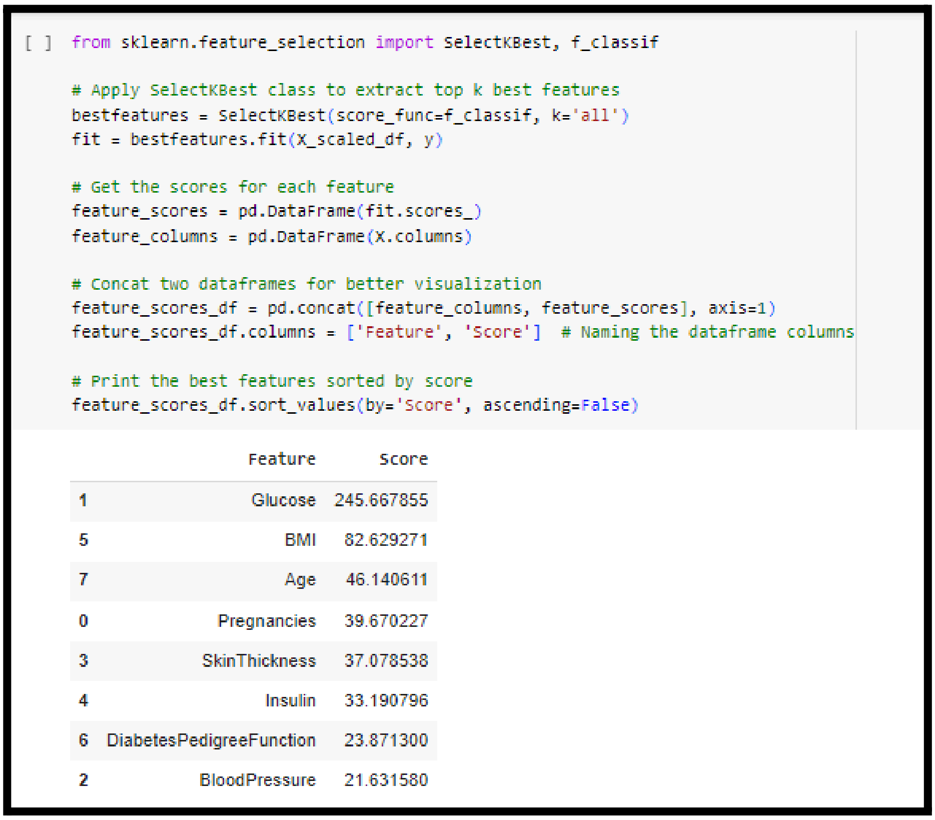
FEATURE **SELECTION**

```
[ ] from sklearn.preprocessing import StandardScaler

    # Define the features and the target without 'Unnamed: 0' column
    X = data.drop(['Outcome'], axis=1)
    y = data['Outcome']

    # Initialize the StandardScaler
    scaler = StandardScaler()

    # Fit and transform the features
    X_scaled = scaler.fit_transform(X)

    # Convert the scaled features back to a dataframe
    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

    # Display the head of the scaled features dataframe
    X_scaled_df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.866045	-0.031990	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.205066	-0.528319	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.016662	-0.693761	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.073567	-0.528319	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.504422	-2.679076	0.670643	0.316566	1.549303	5.484909	-0.020496

STEP 06

This code uses the SelectKBest method with the f_classif scoring function to evaluate and rank features in the standardized 'X_scaled_df' DataFrame based on their significance in predicting the target variable 'y' (Outcome). It then creates a DataFrame ('feature_scores_df') displaying the features and their corresponding scores, sorted in descending order by score

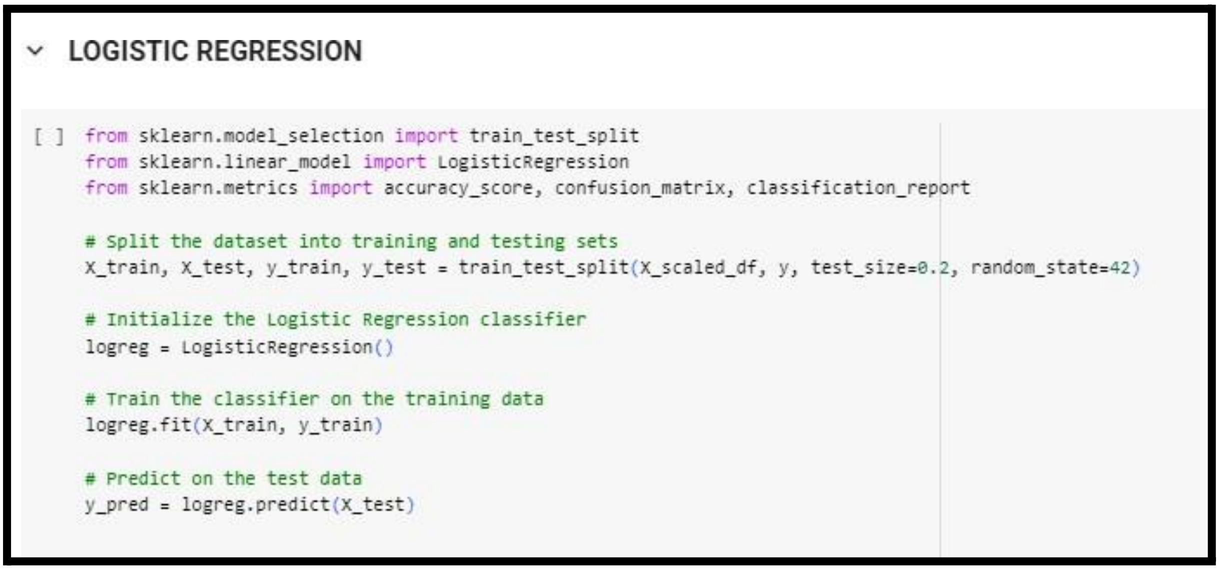


```
[ ] from sklearn.feature_selection import SelectKBest, f_classif

    # Apply SelectKBest class to extract top k best features
    bestfeatures = SelectKBest(score_func=f_classif, k='all')
    fit = bestfeatures.fit(X_scaled_df, y)

    # Get the scores for each feature
    feature_scores = pd.DataFrame(fit.scores_)
    feature_columns = pd.DataFrame(X.columns)

    # Concat two dataframes for better visualization
    feature_scores_df = pd.concat([feature_columns, feature_scores], axis=1)
    feature_scores_df.columns = ['Feature', 'Score']  # Naming the dataframe columns

    # Print the best features sorted by score
    feature_scores_df.sort_values(by='Score', ascending=False)
```

```
                    Feature       Score
1                   Glucose  245.667855
5                       BMI   82.629271
7                       Age   46.140611
0               Pregnancies   39.670227
3             SkinThickness   37.078538
4                   Insulin   33.190796
6  DiabetesPedigreeFunction   23.871300
2             BloodPressure   21.631580
```

STEP 07

This code splits the standardized features ('X_scaled_df') and the target variable ('y') into training and testing sets using the train_test_split function. It then initializes a Logistic Regression classifier, trains the classifier on the training data, and makes predictions on the test data, storing the results in 'y_pred'.



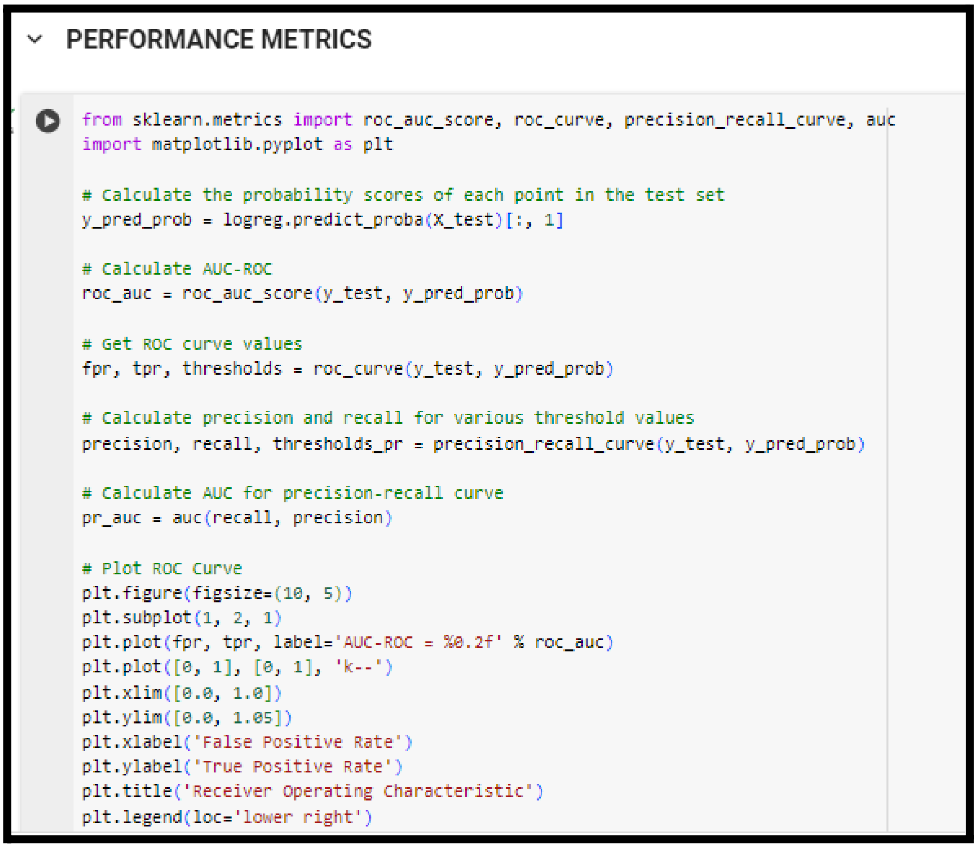
LOGISTIC REGRESSION

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size=0.2, random_state=42)

    # Initialize the Logistic Regression classifier
    logreg = LogisticRegression()

    # Train the classifier on the training data
    logreg.fit(X_train, y_train)

    # Predict on the test data
    y_pred = logreg.predict(X_test)
```

STEP 08

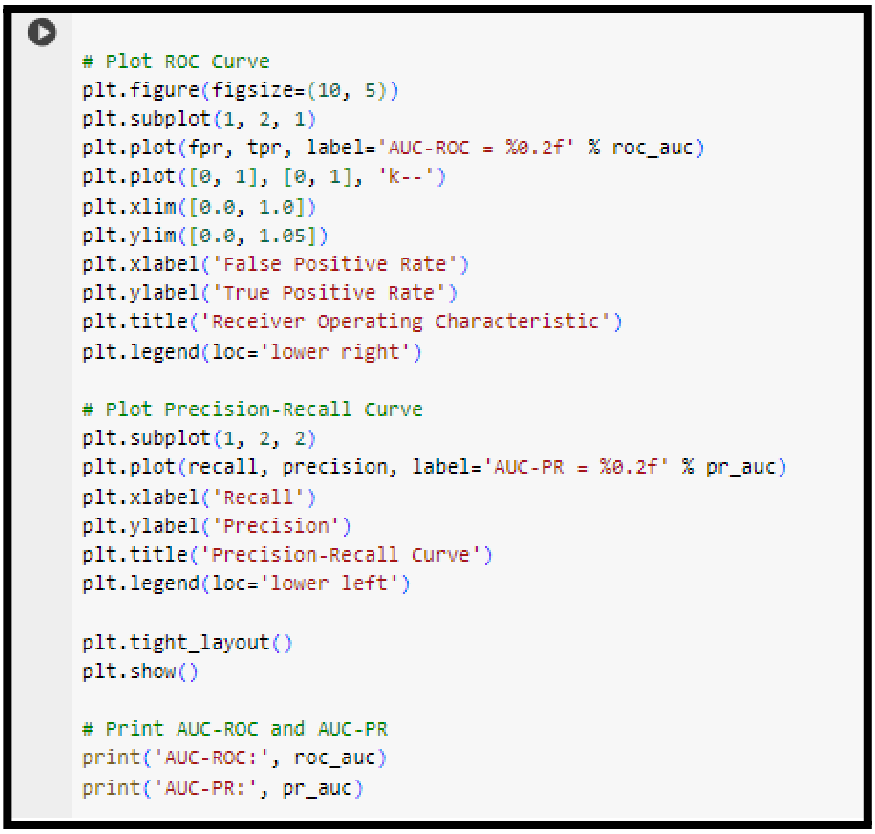
This code calculates the probability scores ('y_pred_prob') using the trained Logistic Regression classifier ('logreg') on the test data and then computes the Area Under the Curve

(AUC) for both the Receiver Operating Characteristic (ROC) curve and the Precision-Recall (PR) curve. It plots both curves on a single figure, with AUC values labeled, and prints the AUC-ROC and AUC-PR values.

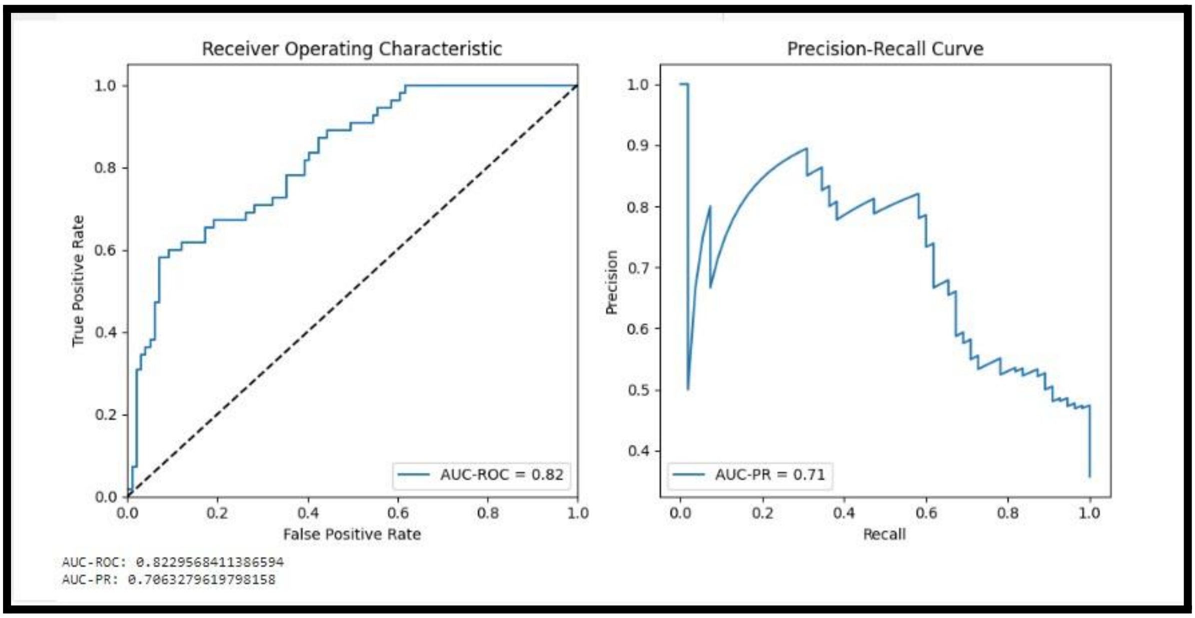


PERFORMANCE METRICS

```
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve, auc
import matplotlib.pyplot as plt

# Calculate the probability scores of each point in the test set
y_pred_prob = logreg.predict_proba(X_test)[:, 1]

# Calculate AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Get ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Calculate precision and recall for various threshold values
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_pred_prob)

# Calculate AUC for precision-recall curve
pr_auc = auc(recall, precision)

# Plot ROC Curve
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, label='AUC-ROC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
```



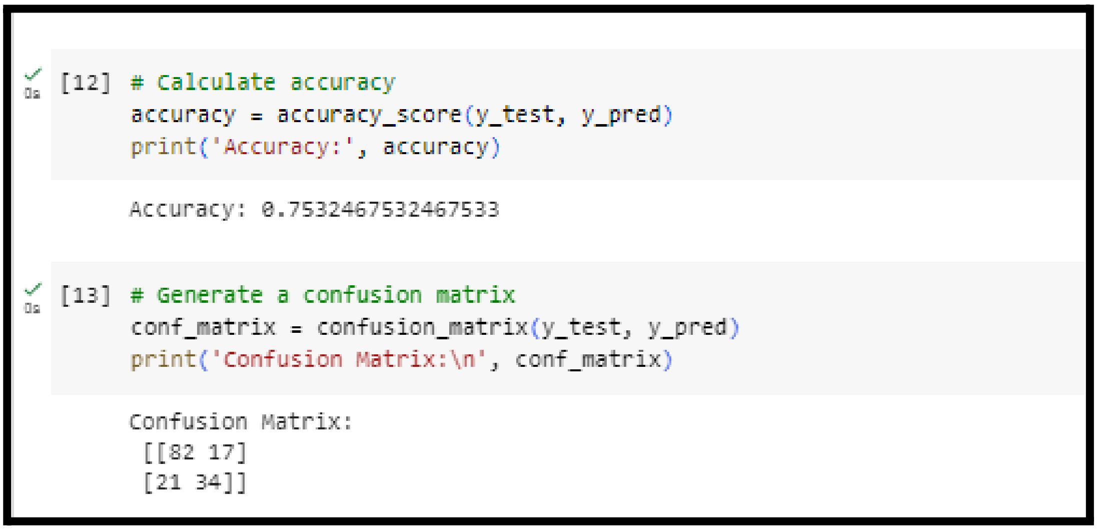
```
# Plot ROC Curve
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, label='AUC-ROC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')

# Plot Precision-Recall Curve
plt.subplot(1, 2, 2)
plt.plot(recall, precision, label='AUC-PR = %0.2f' % pr_auc)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')

plt.tight_layout()
plt.show()

# Print AUC-ROC and AUC-PR
print('AUC-ROC:', roc_auc)
print('AUC-PR:', pr_auc)
```



```
AUC-ROC: 0.8229568411386594
AUC-PR: 0.7063279619798158
```

STEP 09

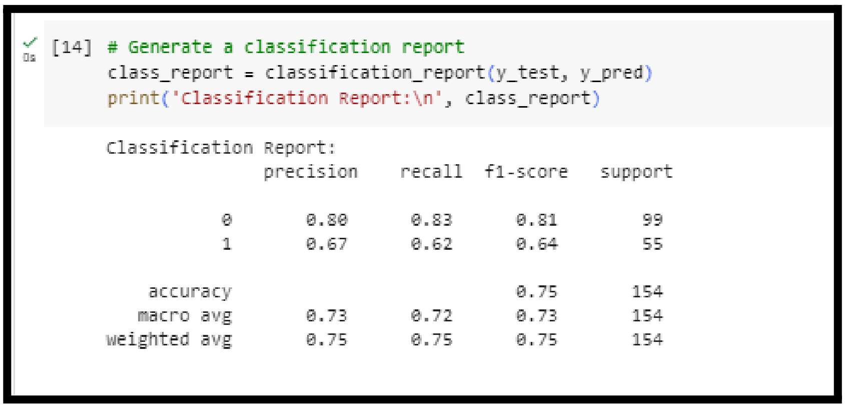
This code calculates the accuracy of the Logistic Regression classifier's predictions on the test data and prints the accuracy value. Additionally, it generates a confusion matrix ('conf_matrix') to evaluate the performance of the classifier, displaying the count of true positive, true negative, false positive, and false negative predictions.



```
[12] # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print('Accuracy:', accuracy)

     Accuracy: 0.7532467532467533

[13] # Generate a confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred)
     print('Confusion Matrix:\n', conf_matrix)

     Confusion Matrix:
      [[82 17]
      [21 34]]
```

STEP 10

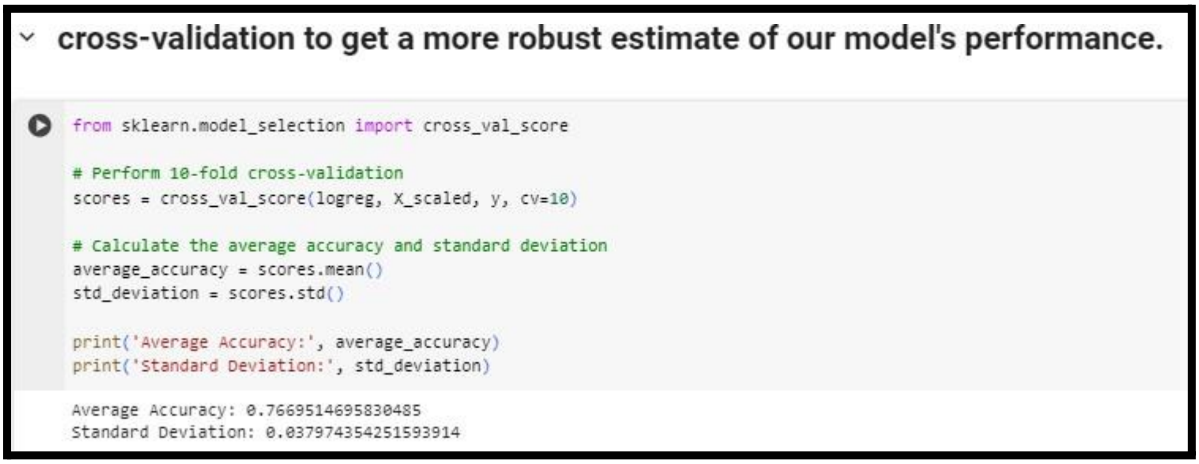
This code generates a classification report for the Logistic Regression classifier's predictions on the test data, providing metrics such as precision, recall, F1-score, and support for each class. The report is then printed for evaluation.



```
[14] # Generate a classification report
     class_report = classification_report(y_test, y_pred)
     print('Classification Report:\n', class_report)

     Classification Report:
                   precision    recall  f1-score   support

                0       0.80      0.83      0.81        99
                1       0.67      0.62      0.64        55

         accuracy                           0.75       154
        macro avg       0.73      0.72      0.73       154
     weighted avg       0.75      0.75      0.75       154
```

STEP 11

This code performs 10-fold cross-validation using the Logistic Regression classifier ('logreg') on the standardized features ('X_scaled') and target variable ('y'). It calculates accuracy scores for each fold and then computes the average accuracy and standard deviation of the scores. The results are printed to provide an estimate of the model's generalization performance.

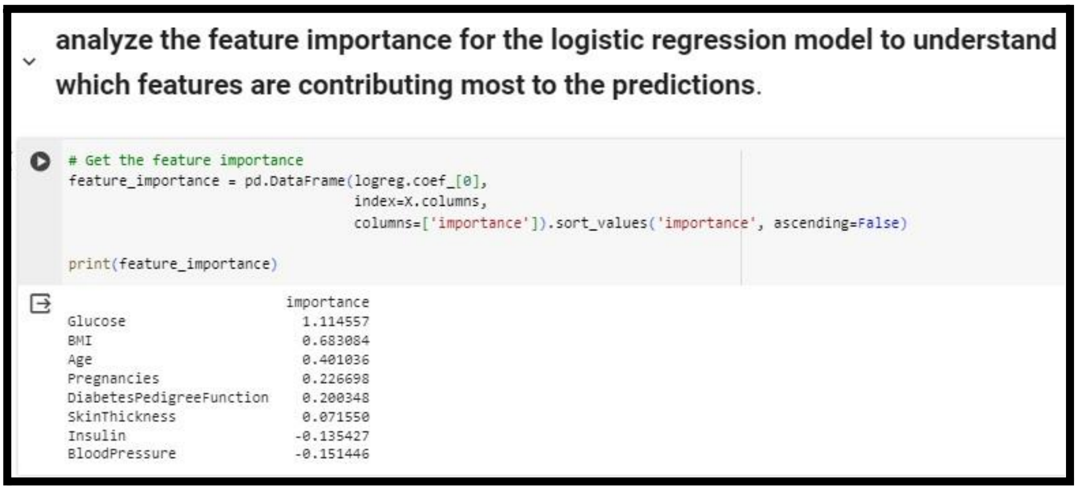


cross-validation to get a more robust estimate of our model's performance.

```
from sklearn.model_selection import cross_val_score

# Perform 10-fold cross-validation
scores = cross_val_score(logreg, X_scaled, y, cv=10)

# Calculate the average accuracy and standard deviation
average_accuracy = scores.mean()
std_deviation = scores.std()

print('Average Accuracy:', average_accuracy)
print('Standard Deviation:', std_deviation)

Average Accuracy: 0.7669514695830485
Standard Deviation: 0.037974354251593914
```

STEP 12

This code retrieves and prints the feature importance of the Logistic Regression classifier. It creates a DataFrame ('feature_importance') with the coefficients of each feature, indexed by feature names, and then sorts them in descending order based on their importance values. This information can help identify which features have a stronger impact on the model's predictions.



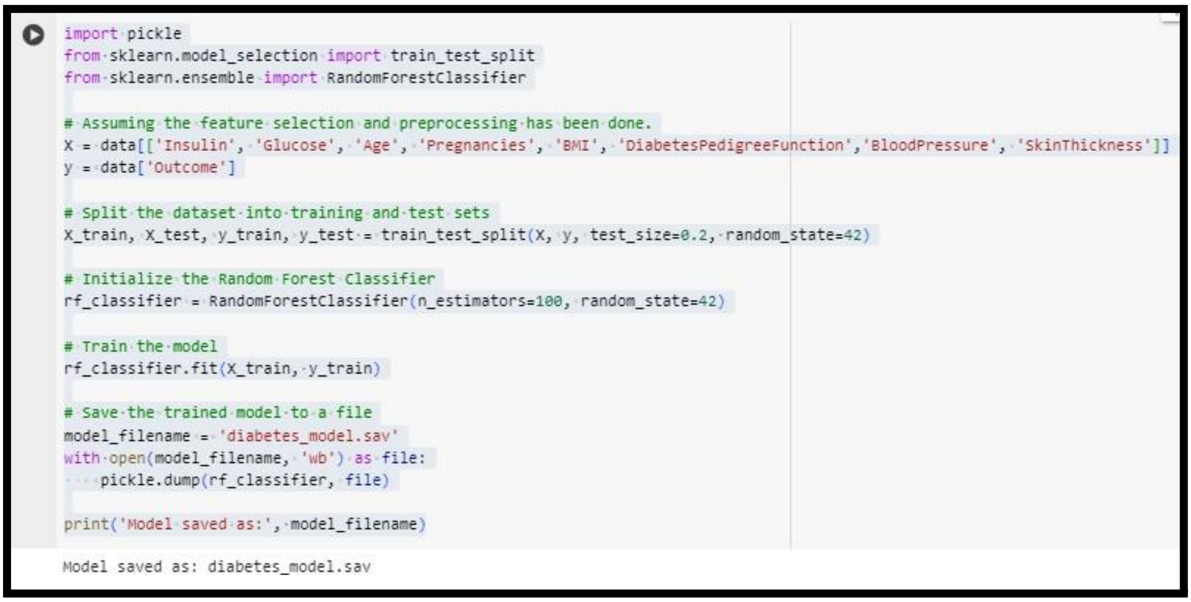
analyze the feature importance for the logistic regression model to understand which features are contributing most to the predictions.

```
# Get the feature importance
feature_importance = pd.DataFrame(logreg.coef_[0],
                                  index=X.columns,
                                  columns=['importance']).sort_values('importance', ascending=False)

print(feature_importance)
```

```
                          importance
Glucose                     1.114557
BMI                         0.683084
Age                         0.401036
Pregnancies                 0.226698
DiabetesPedigreeFunction    0.200348
SkinThickness               0.071550
Insulin                    -0.135427
BloodPressure              -0.151446
```

CREATING SYSTEM MODEL

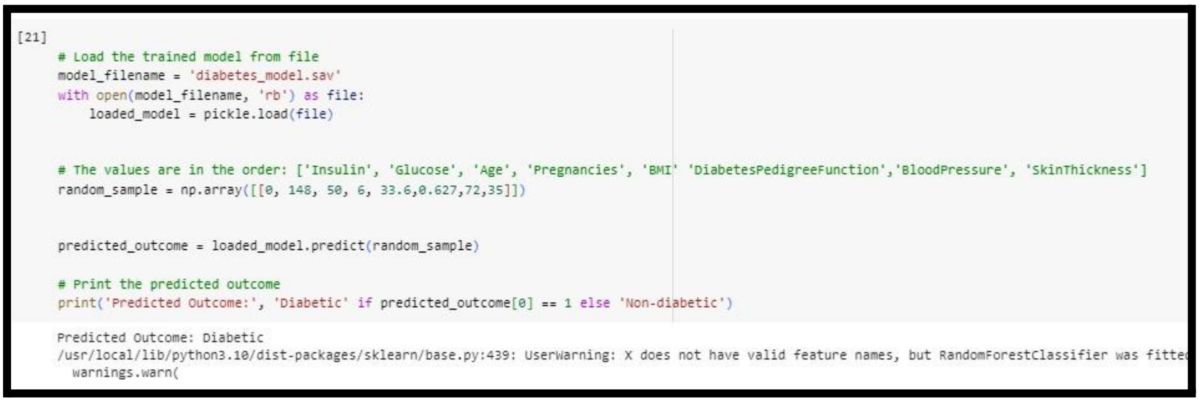
This code trains a Random Forest Classifier on the specified features ('Insulin', 'Glucose', 'Age', 'Pregnancies', 'BMI', 'DiabetesPedigreeFunction', 'BloodPressure', 'SkinThickness') and the target variable ('Outcome'). It then saves the trained model to a file named 'diabetes_model.sav' using the pickle module. The printed message indicates that the model has been successfully saved.



```
import pickle
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Assuming the feature selection and preprocessing has been done.
X = data[['Insulin', 'Glucose', 'Age', 'Pregnancies', 'BMI', 'DiabetesPedigreeFunction','BloodPressure', 'SkinThickness']]
y = data['Outcome']

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_classifier.fit(X_train, y_train)

# Save the trained model to a file
model_filename = 'diabetes_model.sav'
with open(model_filename, 'wb') as file:
    pickle.dump(rf_classifier, file)

print('Model saved as:', model_filename)
```

```
Model saved as: diabetes_model.sav
```

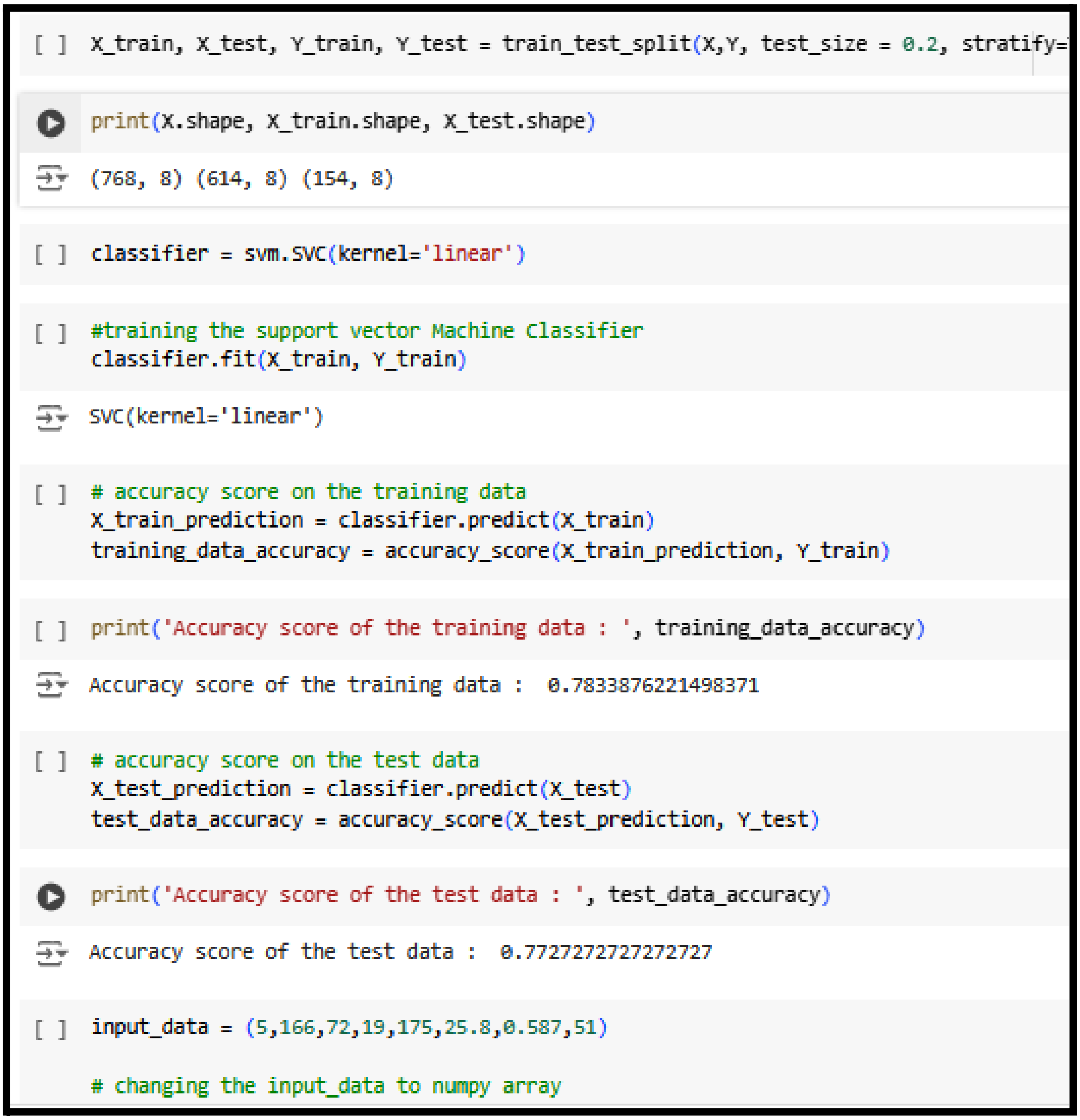
This code loads a previously trained Random Forest Classifier from the file 'diabetes_model.sav' using the pickle module. It then uses the loaded model to predict the outcome (Diabetic or Non-diabetic) for a random sample provided in the 'random_sample' array. The predicted outcome is printed based on the model's prediction for the input data.



```
[21]
    # Load the trained model from file
    model_filename = 'diabetes_model.sav'
    with open(model_filename, 'rb') as file:
        loaded_model = pickle.load(file)


    # The values are in the order: ['Insulin', 'Glucose', 'Age', 'Pregnancies', 'BMI' 'DiabetesPedigreeFunction','BloodPressure', 'SkinThickness']
    random_sample = np.array([[0, 148, 50, 6, 33.6,0.627,72,35]])


    predicted_outcome = loaded_model.predict(random_sample)

    # Print the predicted outcome
    print('Predicted Outcome:', 'Diabetic' if predicted_outcome[0] == 1 else 'Non-diabetic')
```

```
Predicted Outcome: Diabetic
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitte
  warnings.warn(
```

SUPPORT VECTOR MACHINE CLASSIFIER



```
[ ]  X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=
```

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

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

```
[ ]  classifier = svm.SVC(kernel='linear')
```

```
[ ]  #training the support vector Machine Classifier
     classifier.fit(X_train, Y_train)
```

```
SVC(kernel='linear')
```

```
[ ]  # accuracy score on the training data
     X_train_prediction = classifier.predict(X_train)
     training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[ ]  print('Accuracy score of the training data : ', training_data_accuracy)
```

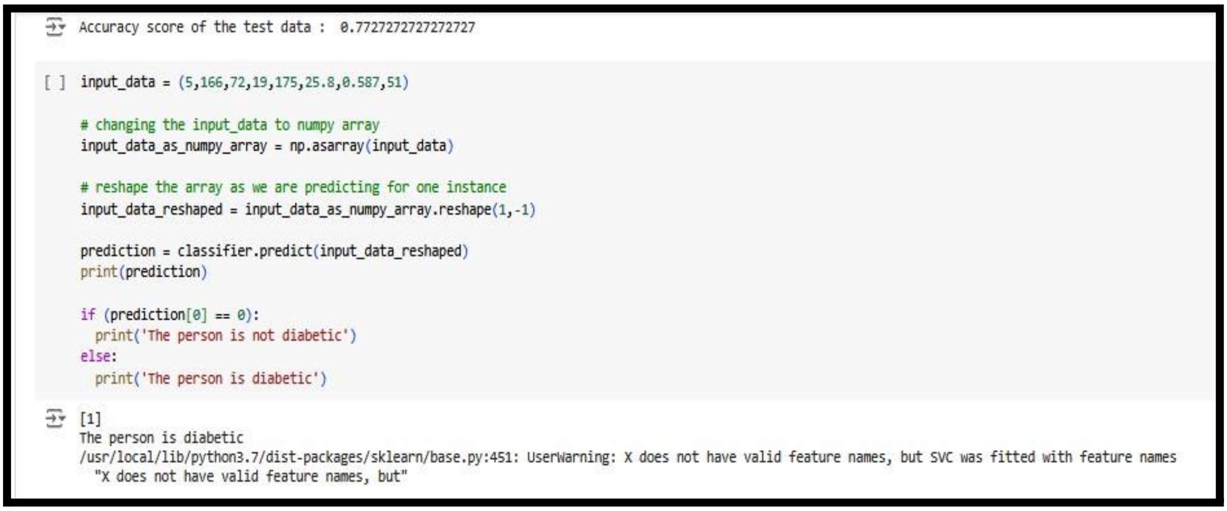
```
Accuracy score of the training data :  0.7833876221498371
```

```
[ ]  # accuracy score on the test data
     X_test_prediction = classifier.predict(X_test)
     test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7727272727272727
```

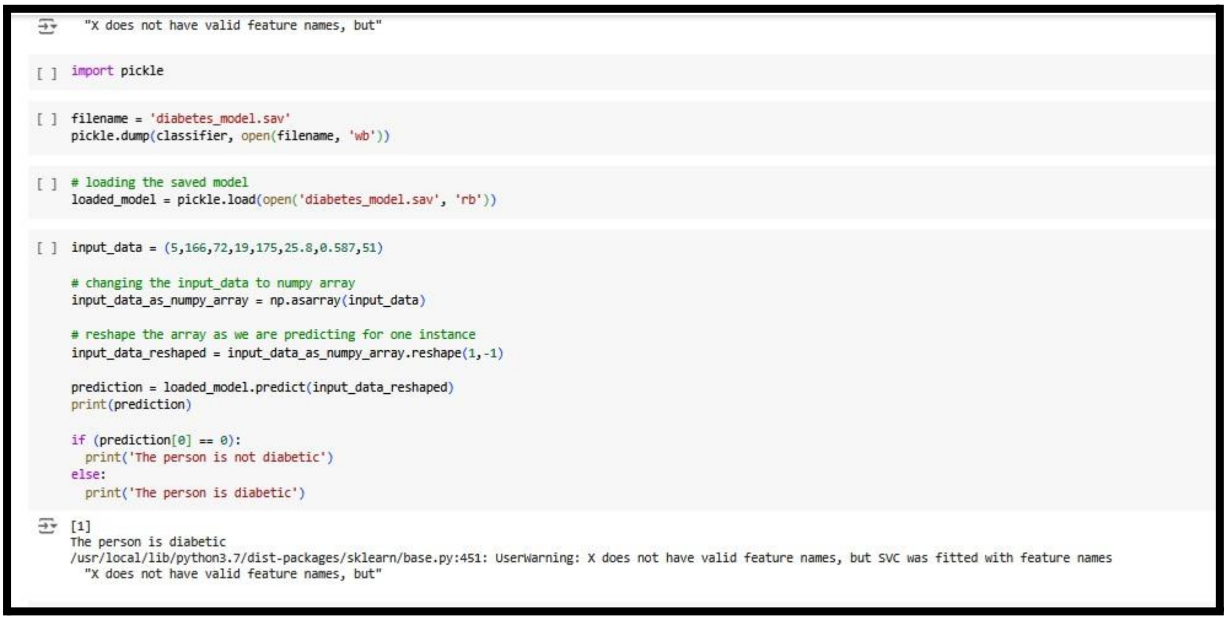
```
[ ]  input_data = (5,166,72,19,175,25.8,0.587,51)

     # changing the input_data to numpy array
```



```
Accuracy score of the test data :  0.7727272727272727
```

```
[ ] 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)

    prediction = classifier.predict(input_data_reshaped)
    print(prediction)

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

```
[1]
The person is diabetic
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  "X does not have valid feature names, but"
```



```
  "X does not have valid feature names, but"
```

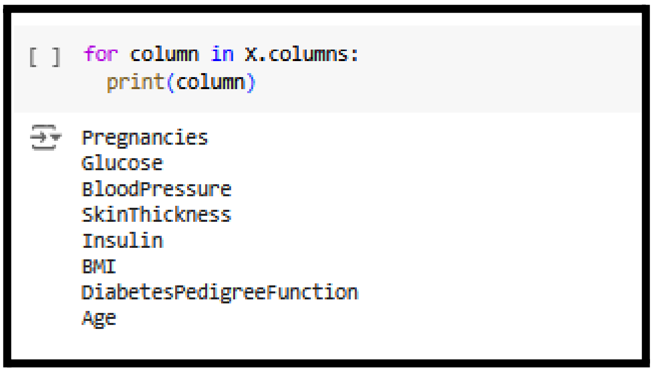
```
[ ] import pickle
```

```
[ ] filename = 'diabetes_model.sav'
    pickle.dump(classifier, open(filename, 'wb'))
```

```
[ ] # loading the saved model
    loaded_model = pickle.load(open('diabetes_model.sav', 'rb'))
```

```
[ ] 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)

    prediction = loaded_model.predict(input_data_reshaped)
    print(prediction)

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

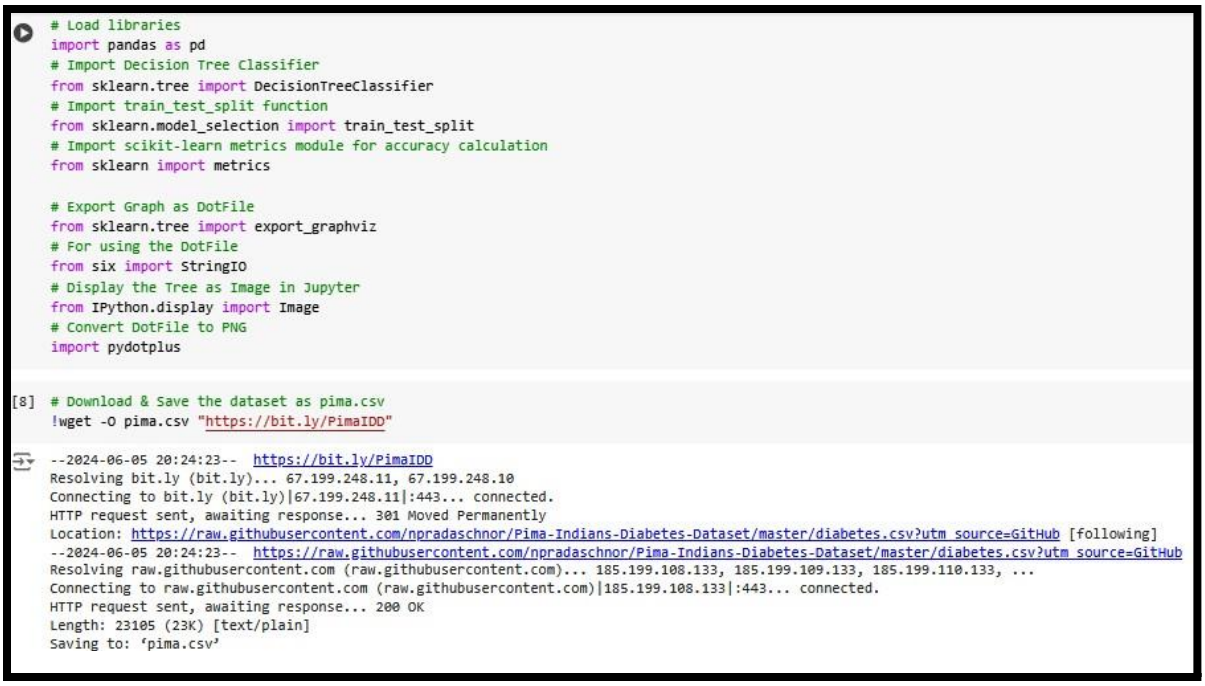
```
[1]
The person is diabetic
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but SVC was fitted with feature names
  "X does not have valid feature names, but"
```



```
[ ] for column in X.columns:
      print(column)
```

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

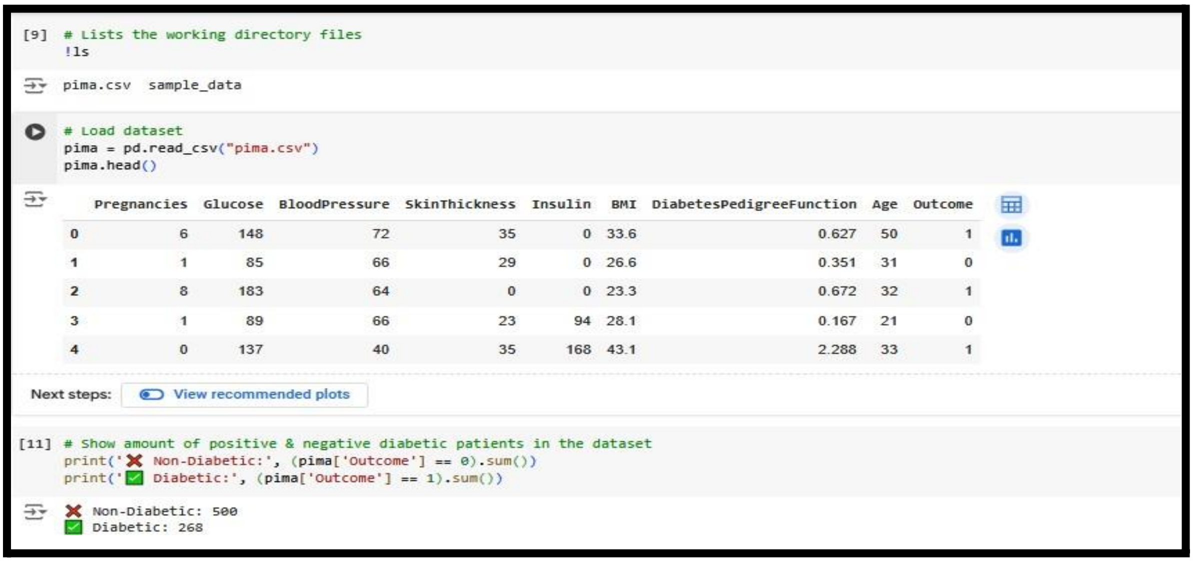
DECISION TREE CLASSIFICATION



```
# Load libraries
import pandas as pd
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Export Graph as DotFile
from sklearn.tree import export_graphviz
# For using the DotFile
from six import StringIO
# Display the Tree as Image in Jupyter
from IPython.display import Image
# Convert DotFile to PNG
import pydotplus
```

```
[8] # Download & Save the dataset as pima.csv
    !wget -O pima.csv "https://bit.ly/PimaIDD"
```

```
--2024-06-05 20:24:23--  https://bit.ly/PimaIDD
Resolving bit.ly (bit.ly)... 67.199.248.11, 67.199.248.10
Connecting to bit.ly (bit.ly)|67.199.248.11|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes.csv?utm_source=GitHub [following]
--2024-06-05 20:24:23--  https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes.csv?utm_source=GitHub
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23105 (23K) [text/plain]
Saving to: ‘pima.csv’
```



```
[9] # Lists the working directory files
    !ls
```

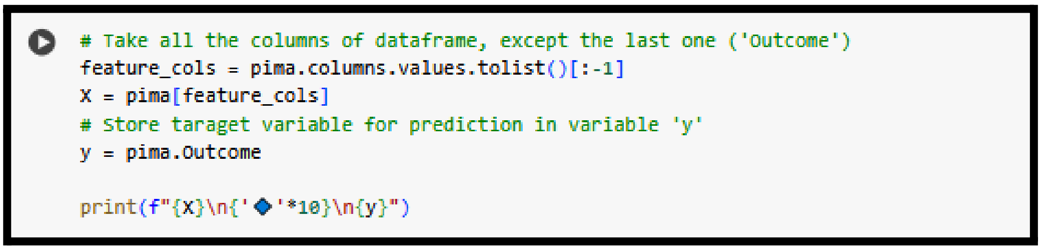
```
pima.csv  sample_data
```

```
# Load dataset
pima = pd.read_csv("pima.csv")
pima.head()
```

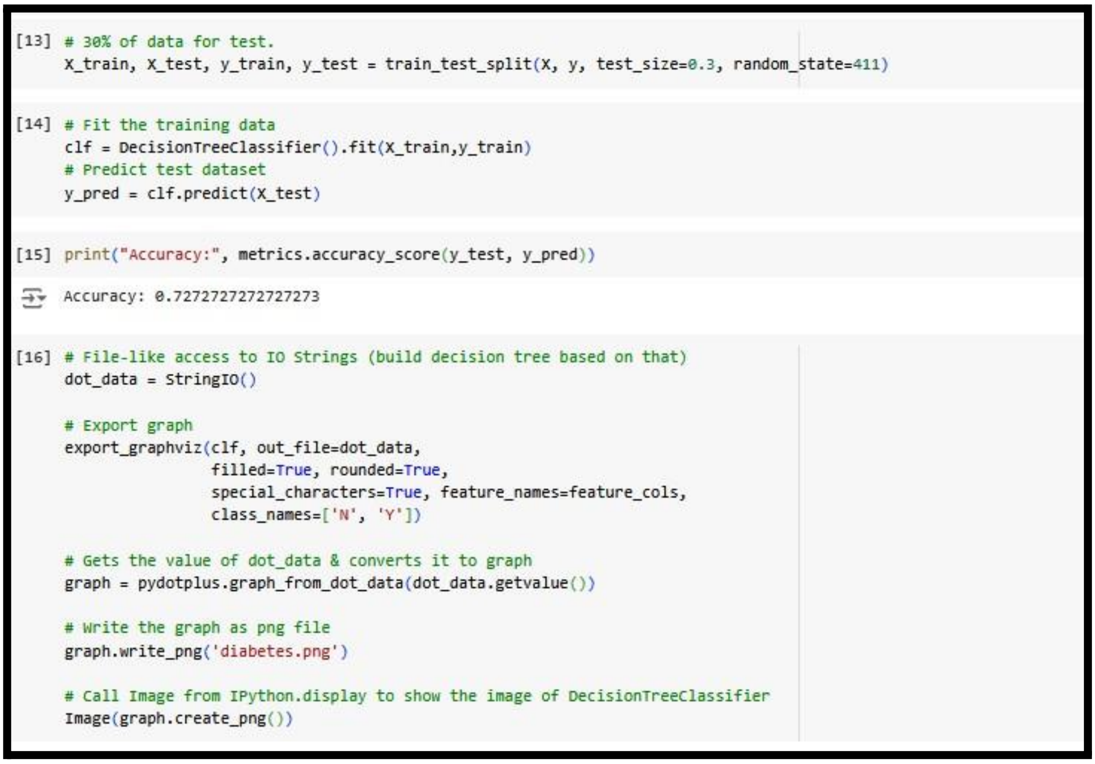
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
[11] # Show amount of positive & negative diabetic patients in the dataset
     print('❌ Non-Diabetic:', (pima['Outcome'] == 0).sum())
     print('✅ Diabetic:', (pima['Outcome'] == 1).sum())
```

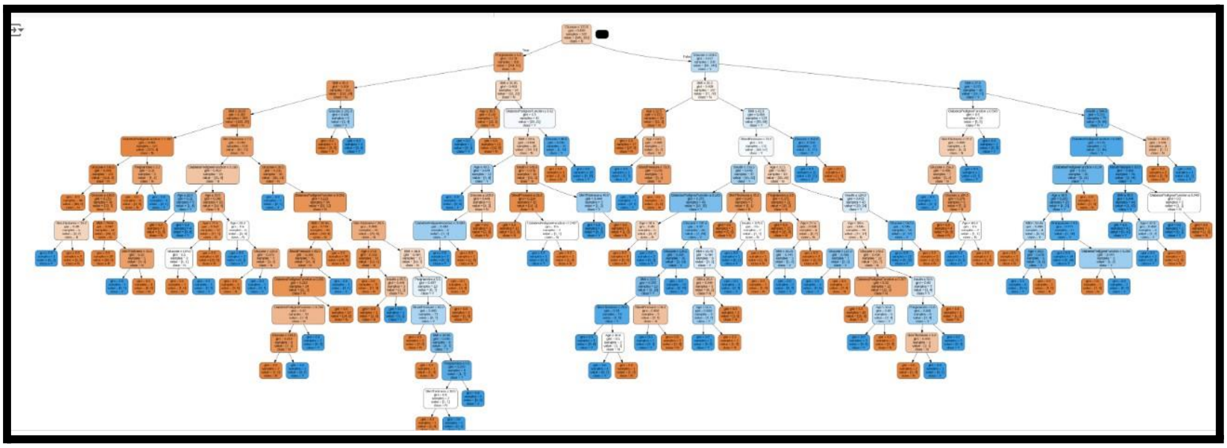
```
❌ Non-Diabetic: 500
✅ Diabetic: 268
```

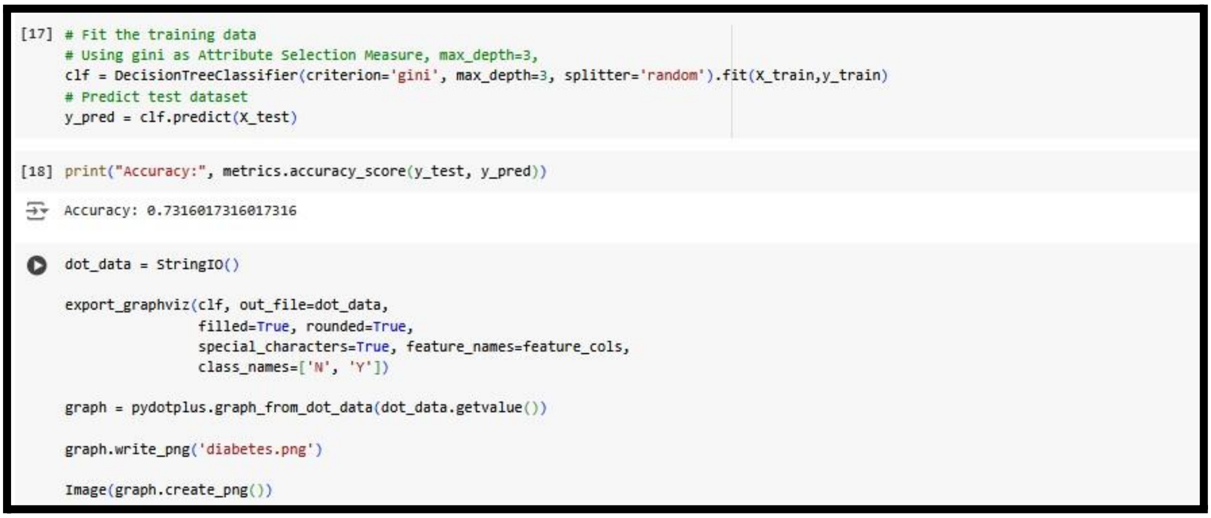


```
# Take all the columns of dataframe, except the last one ('Outcome')
feature_cols = pima.columns.values.tolist()[:-1]
X = pima[feature_cols]
# Store taraget variable for prediction in variable 'y'
y = pima.Outcome

print(f"{X}\n{'🔷'*10}\n{y}")
```

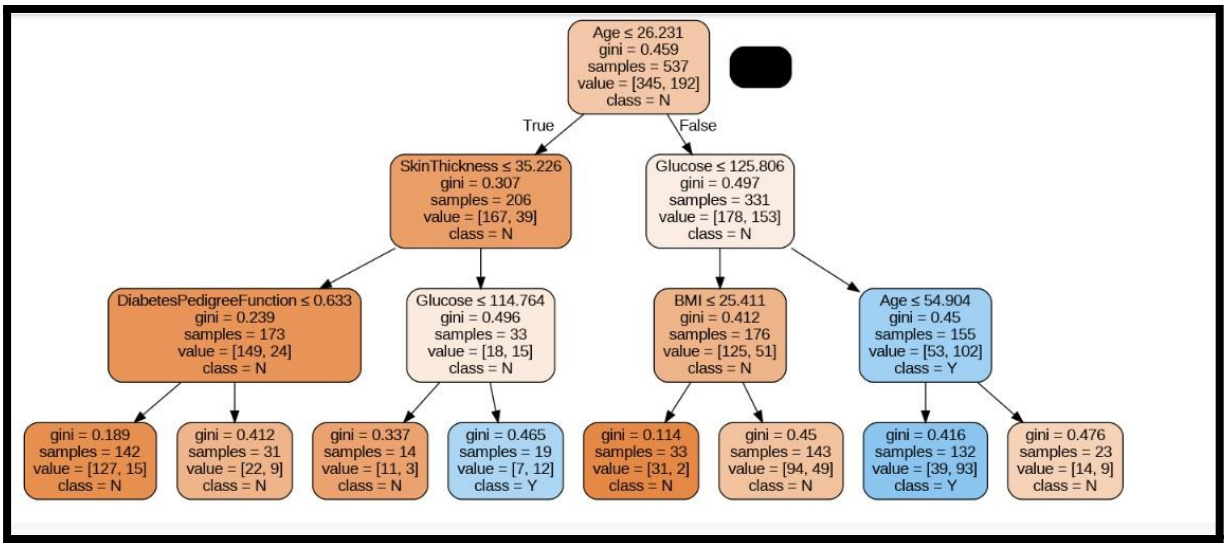


```
[13] # 30% of data for test.
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=411)

[14] # Fit the training data
     clf = DecisionTreeClassifier().fit(X_train,y_train)
     # Predict test dataset
     y_pred = clf.predict(X_test)

[15] print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

     Accuracy: 0.7272727272727273

[16] # File-like access to IO Strings (build decision tree based on that)
     dot_data = StringIO()

     # Export graph
     export_graphviz(clf, out_file=dot_data,
                     filled=True, rounded=True,
                     special_characters=True, feature_names=feature_cols,
                     class_names=['N', 'Y'])

     # Gets the value of dot_data & converts it to graph
     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

     # Write the graph as png file
     graph.write_png('diabetes.png')

     # Call Image from IPython.display to show the image of DecisionTreeClassifier
     Image(graph.create_png())
```





```
[17] # Fit the training data
     # Using gini as Attribute Selection Measure, max_depth=3,
     clf = DecisionTreeClassifier(criterion='gini', max_depth=3, splitter='random').fit(X_train,y_train)
     # Predict test dataset
     y_pred = clf.predict(X_test)

[18] print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

     Accuracy: 0.7316017316017316

     dot_data = StringIO()

     export_graphviz(clf, out_file=dot_data,
                     filled=True, rounded=True,
                     special_characters=True, feature_names=feature_cols,
                     class_names=['N', 'Y'])

     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

     graph.write_png('diabetes.png')

     Image(graph.create_png())
```



CRAFTING SYSTEM FRONTEND

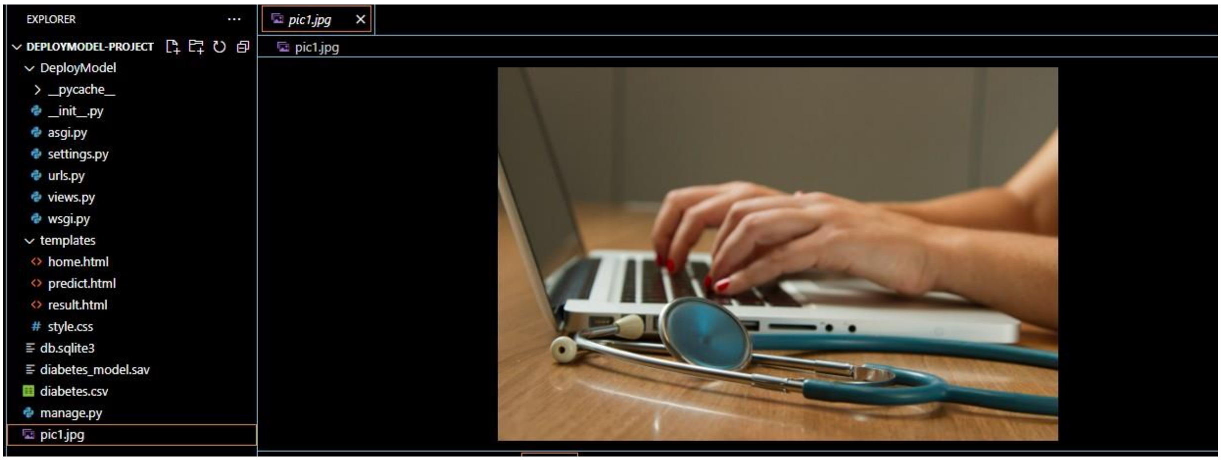
Initially we have created python files according the code in the Colab documentation. Then we installed Django on our command prompt and VS code terminal using the command:

pip install django

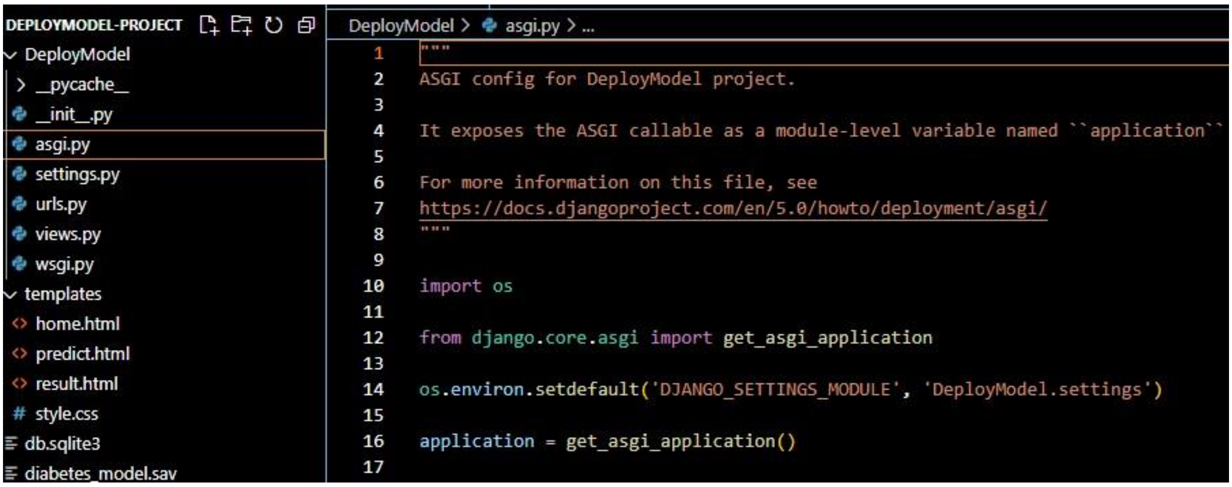
Also installed the other basic libraries like NumPy and Pandas in the system for the slight execution of our project.

After installation you can check its version using command:

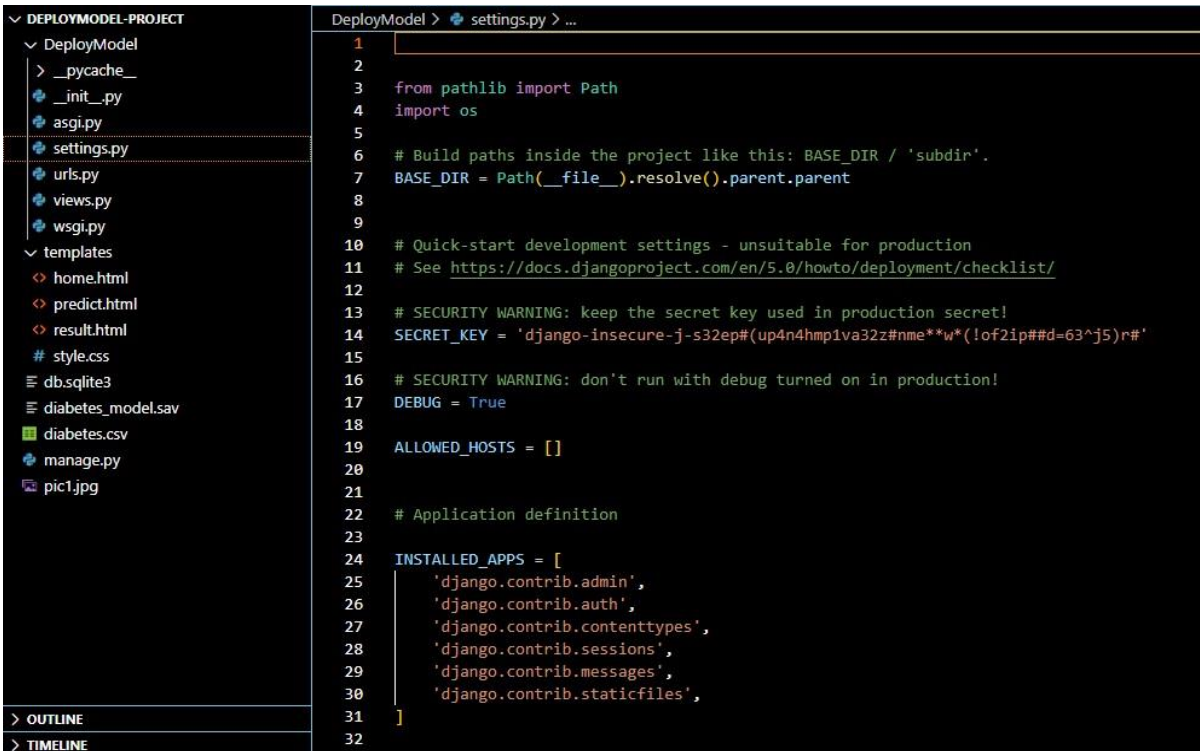
django-admin –version



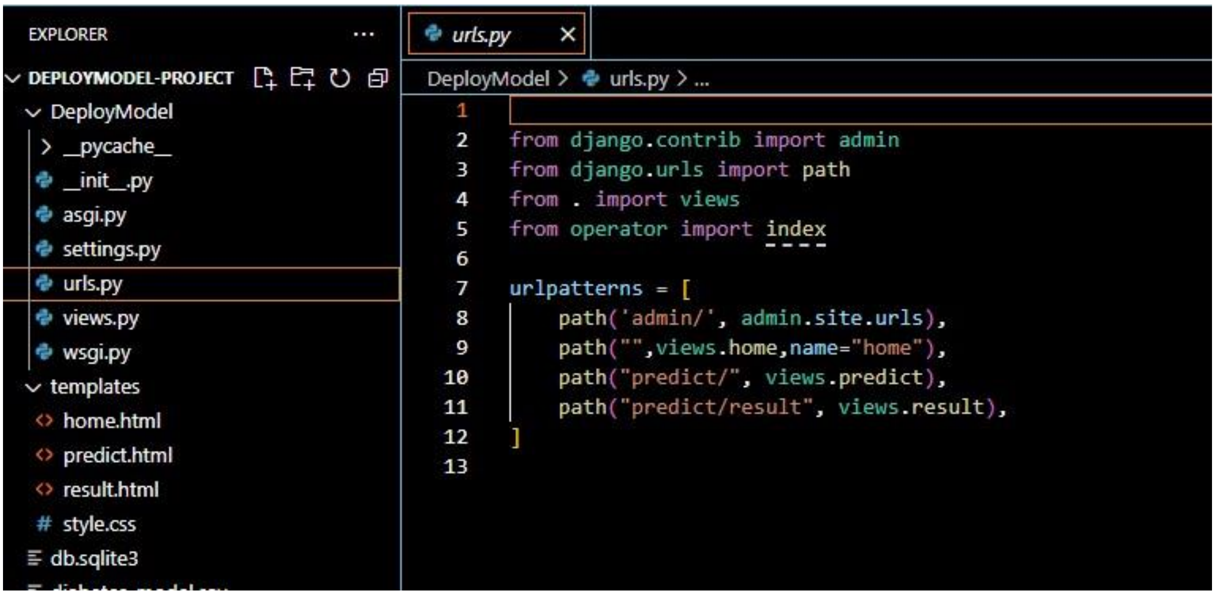
PYTHON FILES



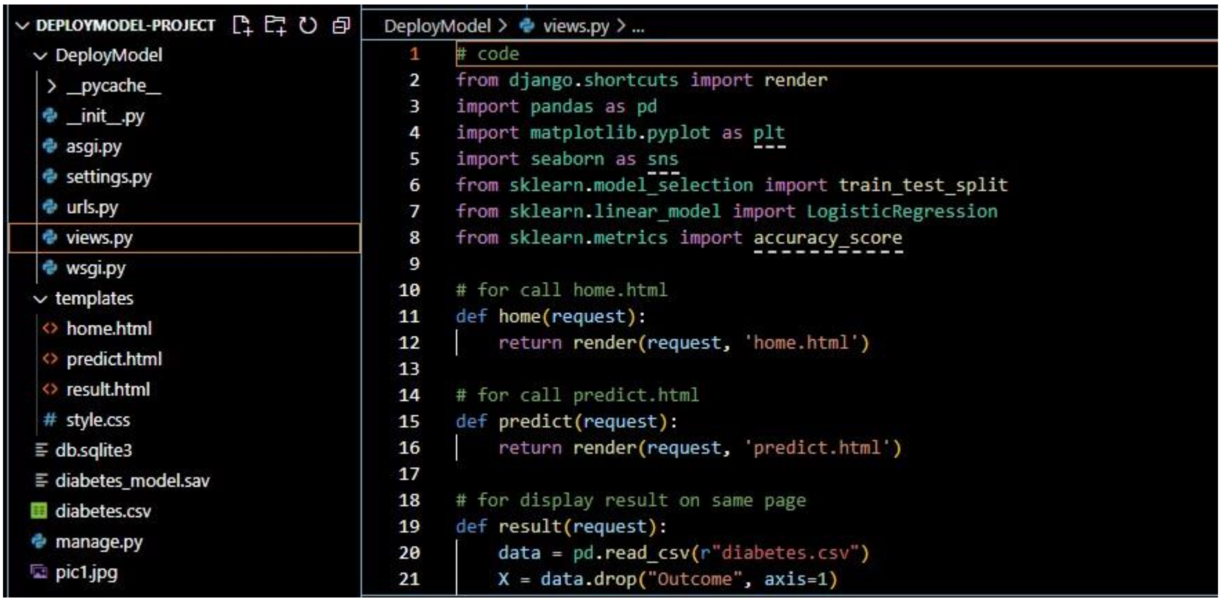
```python
"""
ASGI config for DeployModel project.

It exposes the ASGI callable as a module-level variable named ``application``

For more information on this file, see
https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'DeployModel.settings')

application = get_asgi_application()

```



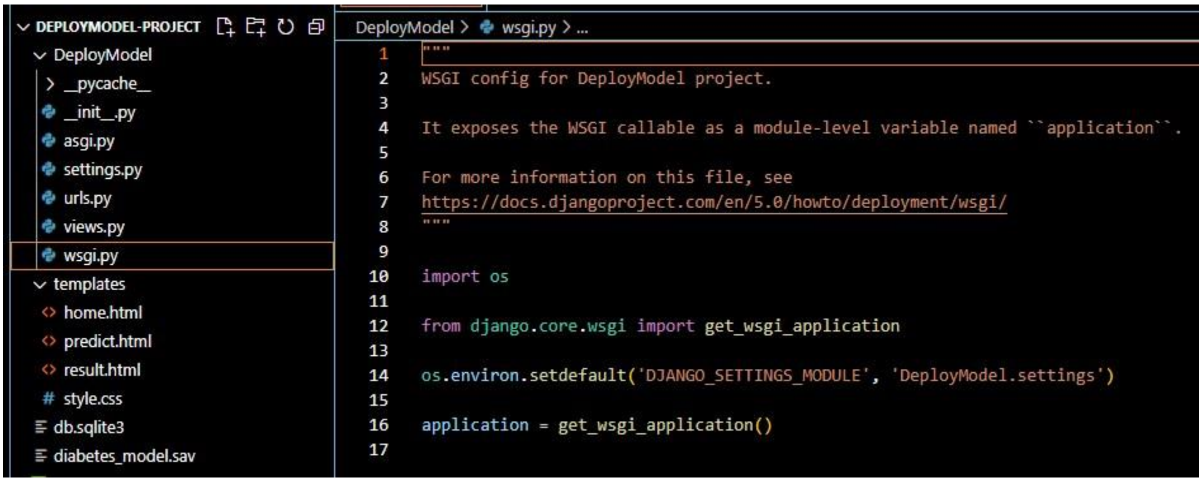
```python


from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-j-s32ep#(up4n4hmp1va32z#nme**w*(!of2ip##d=63^j5)r#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

```



```python

from django.contrib import admin
from django.urls import path
from . import views
from operator import index

urlpatterns = [
    path('admin/', admin.site.urls),
    path("",views.home,name="home"),
    path("predict/", views.predict),
    path("predict/result", views.result),
]

```

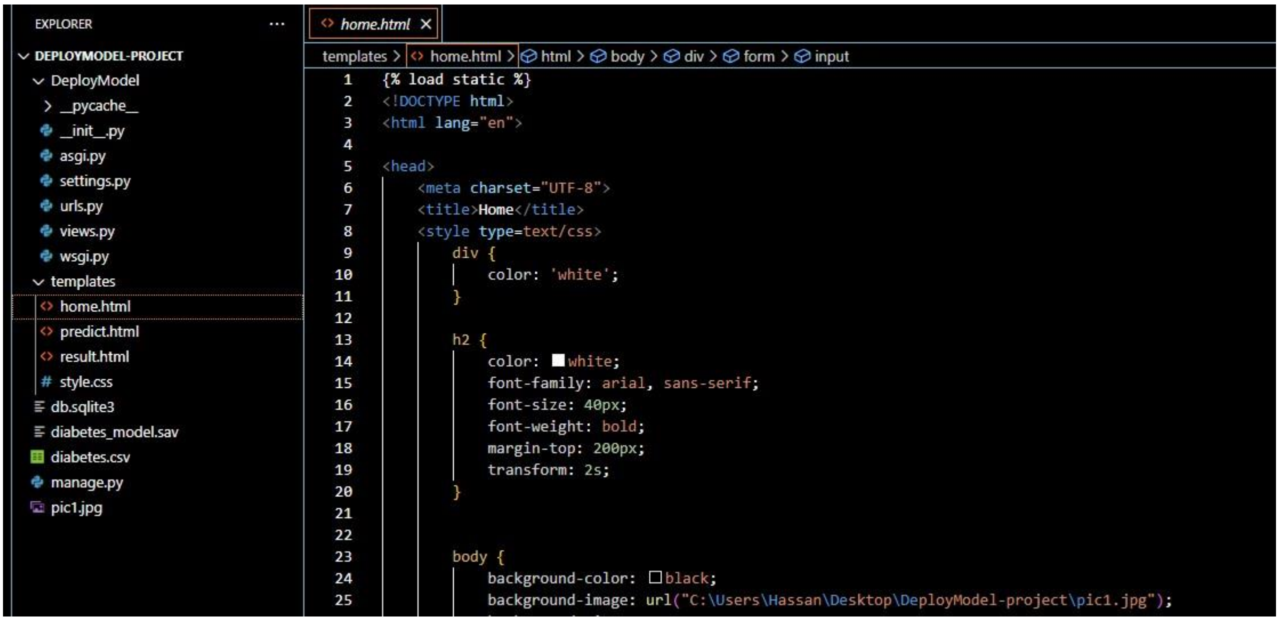


```python
# code
from django.shortcuts import render
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# for call home.html
def home(request):
    return render(request, 'home.html')

# for call predict.html
def predict(request):
    return render(request, 'predict.html')

# for display result on same page
def result(request):
    data = pd.read_csv(r"diabetes.csv")
    X = data.drop("Outcome", axis=1)
```

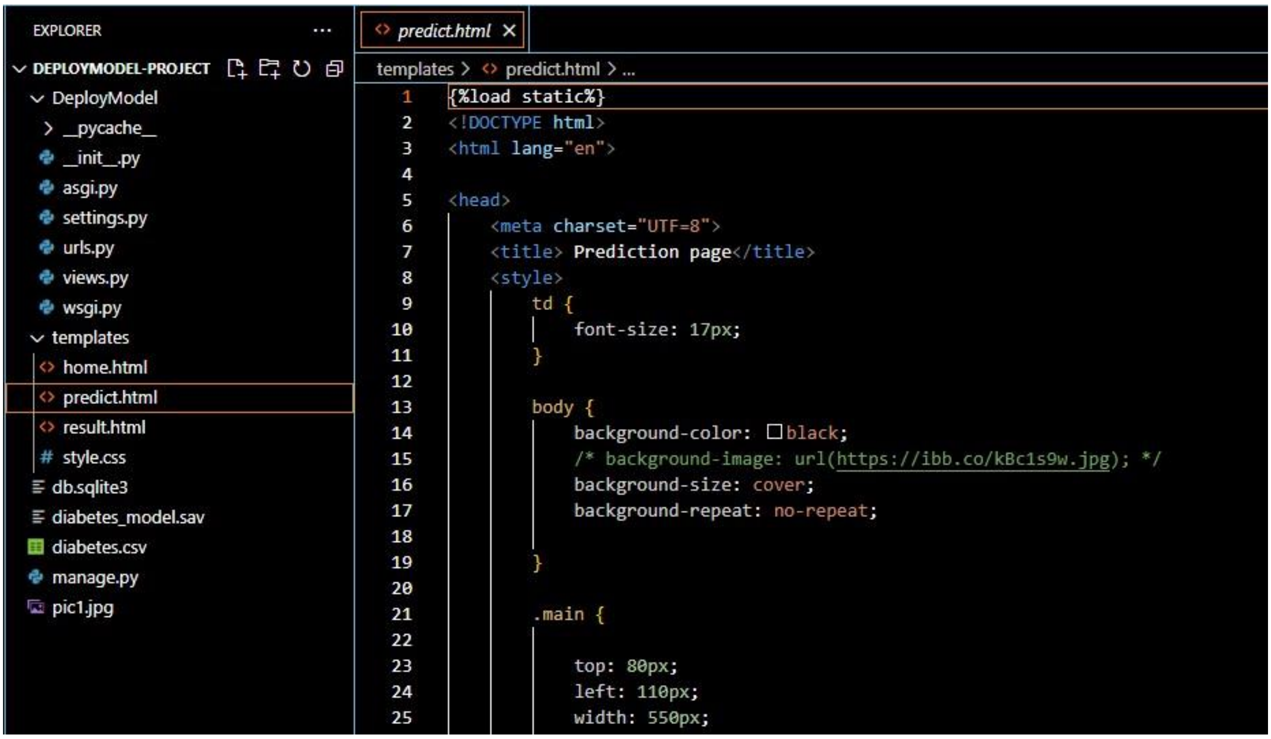


```python
"""
WSGI config for DeployModel project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'DeployModel.settings')

application = get_wsgi_application()

```

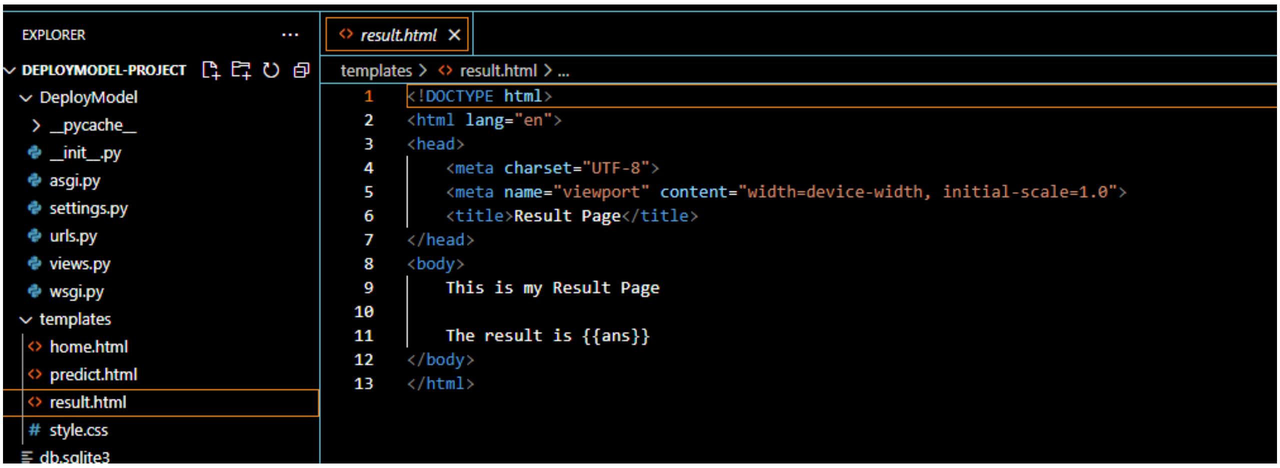
SOURCE FILES



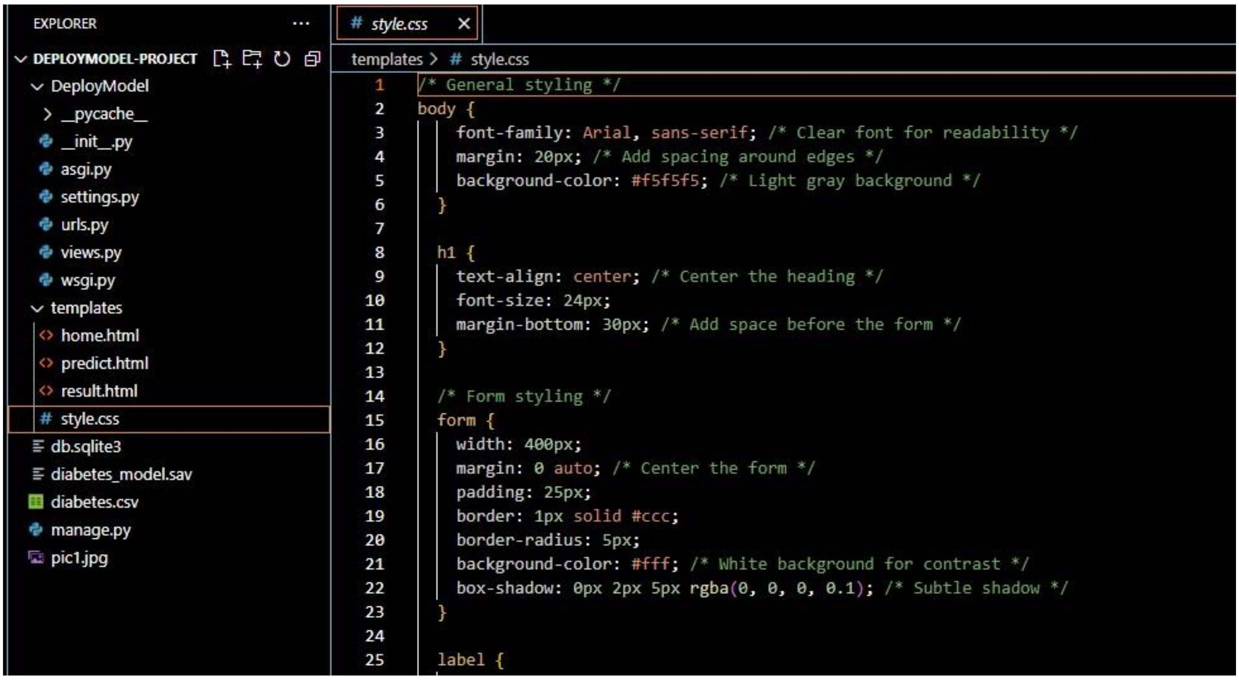
```
{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Home</title>
    <style type=text/css>
        div {
            color: 'white';
        }

        h2 {
            color: ■white;
            font-family: arial, sans-serif;
            font-size: 40px;
            font-weight: bold;
            margin-top: 200px;
            transform: 2s;
        }


        body {
            background-color: □black;
            background-image: url("C:\Users\Hassan\Desktop\DeployModel-project\pic1.jpg");
```



```
{%load static%}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF=8">
    <title> Prediction page</title>
    <style>
        td {
            font-size: 17px;
        }

        body {
            background-color: □black;
            /* background-image: url(https://ibb.co/kBc1s9w.jpg); */
            background-size: cover;
            background-repeat: no-repeat;

        }

        .main {

            top: 80px;
            left: 110px;
            width: 550px;
```

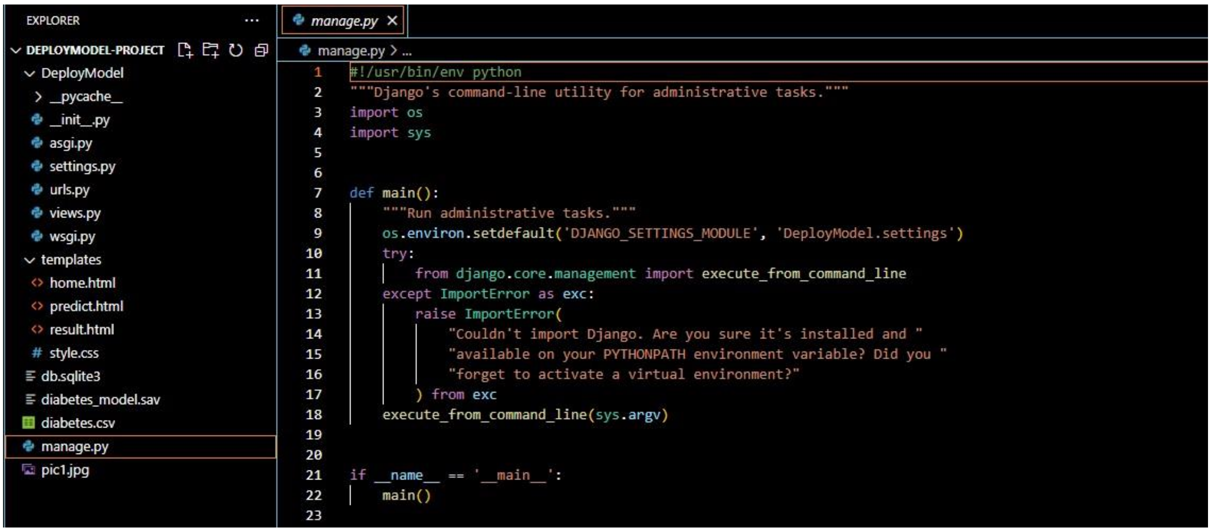


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Result Page</title>
</head>
<body>
    This is my Result Page

    The result is {{ans}}
</body>
</html>
```



```css
/* General styling */
body {
    font-family: Arial, sans-serif; /* Clear font for readability */
    margin: 20px; /* Add spacing around edges */
    background-color: #f5f5f5; /* Light gray background */
  }

  h1 {
    text-align: center; /* Center the heading */
    font-size: 24px;
    margin-bottom: 30px; /* Add space before the form */
  }

  /* Form styling */
  form {
    width: 400px;
    margin: 0 auto; /* Center the form */
    padding: 25px;
    border: 1px solid #ccc;
    border-radius: 5px;
    background-color: #fff; /* White background for contrast */
    box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.1); /* Subtle shadow */
  }

  label {
```

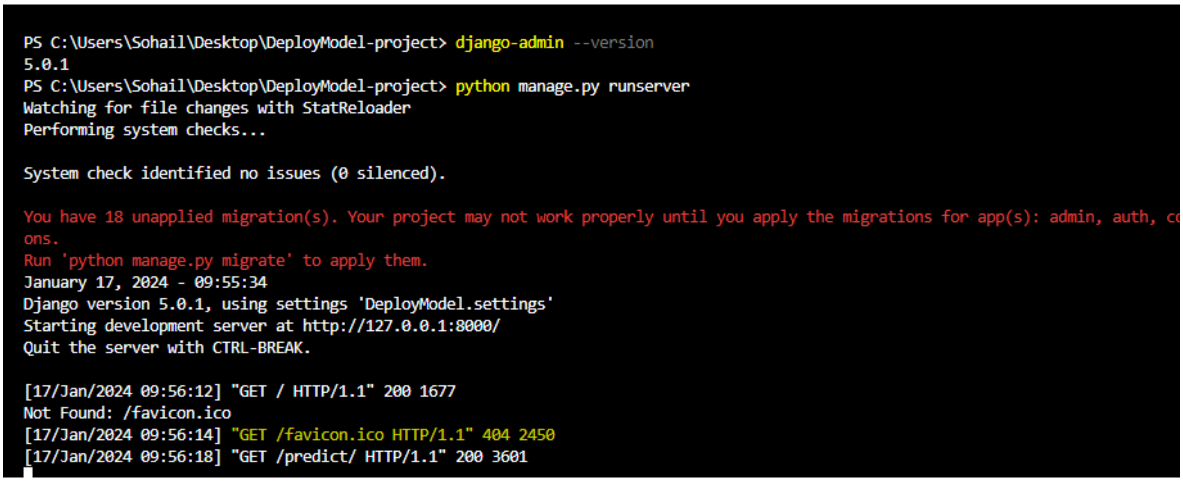
MANAGE FILE (manage.py)



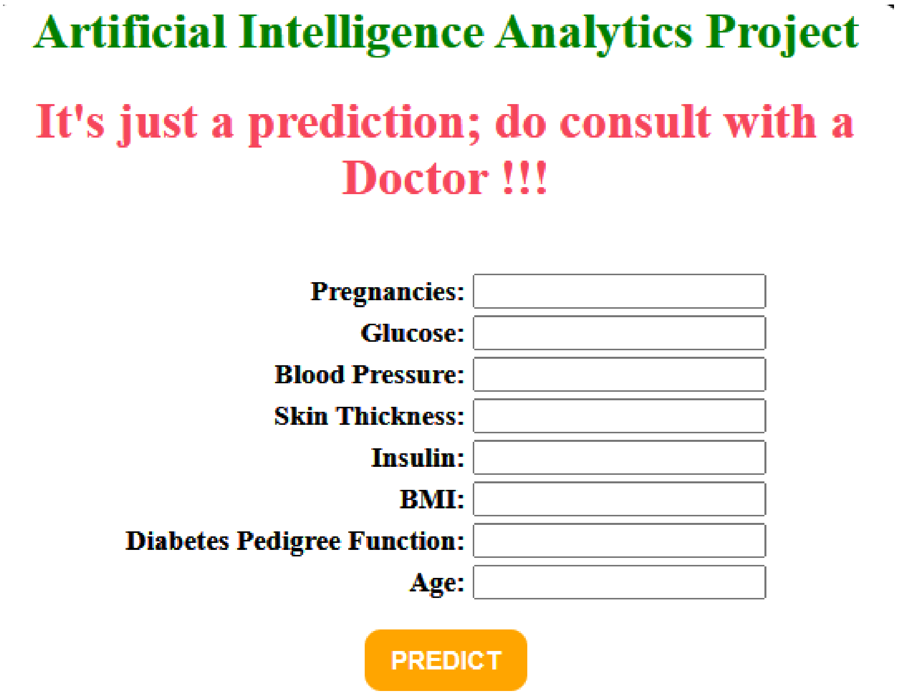
```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'DeployModel.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Now to run the application on web, this command is used:

python manage.py runserver



```
PS C:\Users\Sohail\Desktop\DeployModel-project> django-admin --version
5.0.1
PS C:\Users\Sohail\Desktop\DeployModel-project> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, co
ons.
Run 'python manage.py migrate' to apply them.
January 17, 2024 - 09:55:34
Django version 5.0.1, using settings 'DeployModel.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[17/Jan/2024 09:56:12] "GET / HTTP/1.1" 200 1677
Not Found: /favicon.ico
[17/Jan/2024 09:56:14] "GET /favicon.ico HTTP/1.1" 404 2450
[17/Jan/2024 09:56:18] "GET /predict/ HTTP/1.1" 200 3601
```



Artificial Intelligence Analytics Project

It's just a prediction; do consult with a Doctor !!!

Pregnancies:

Glucose:

Blood Pressure:

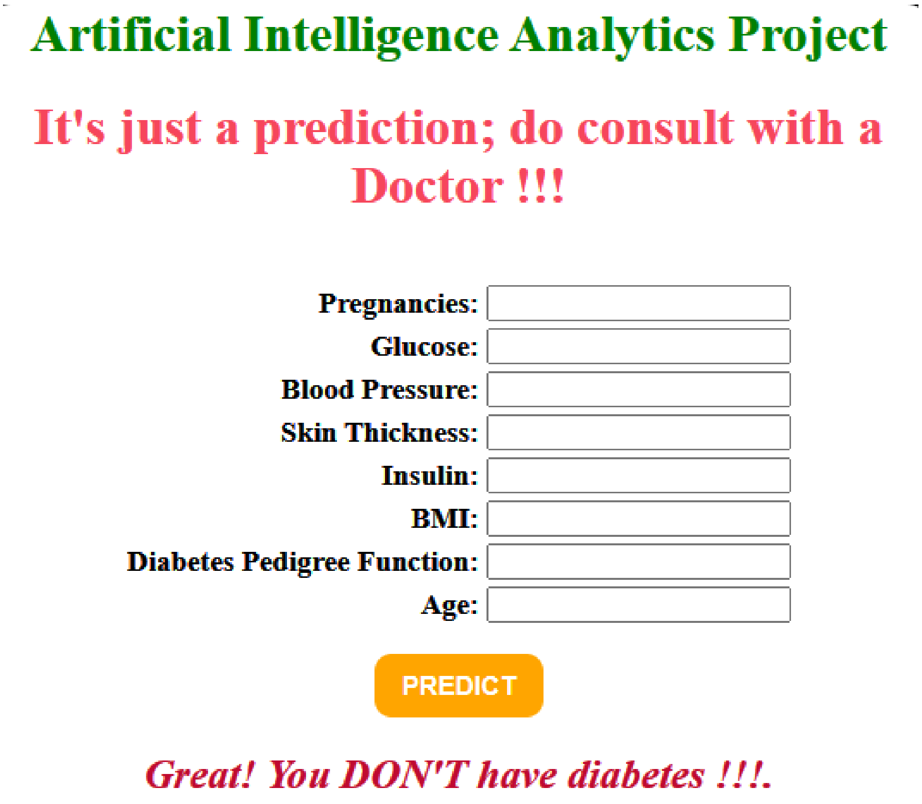
Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:



Artificial Intelligence Analytics Project

It's just a prediction; do consult with a Doctor !!!

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

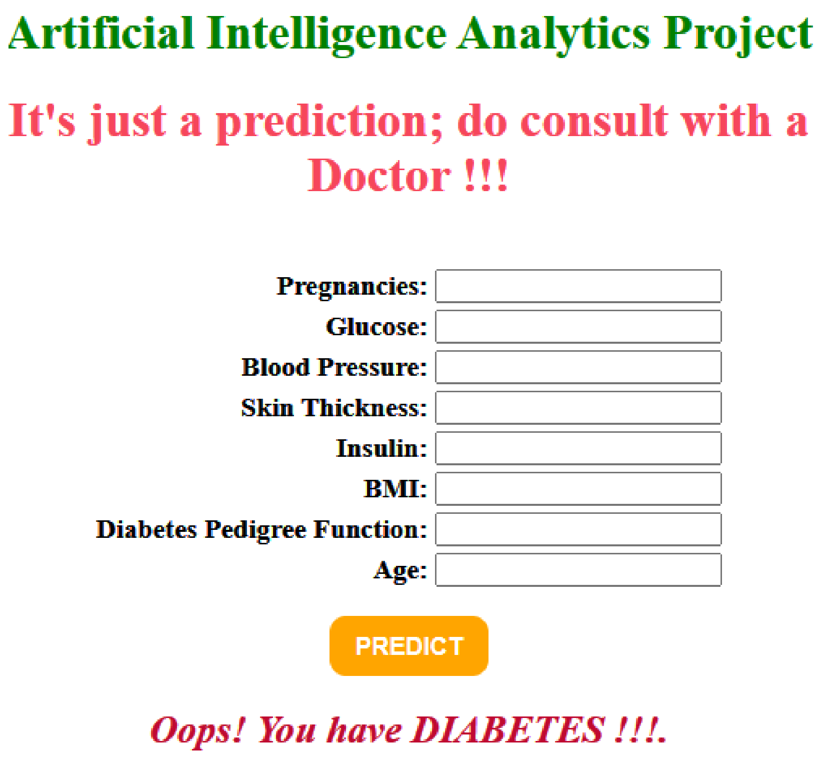
Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Great! You DON'T have diabetes !!!.



Artificial Intelligence Analytics Project

It's just a prediction; do consult with a Doctor !!!

Pregnancies:

Glucose:

Blood Pressure:

Skin Thickness:

Insulin:

BMI:

Diabetes Pedigree Function:

Age:

Oops! You have DIABETES !!!.