ARTIFICIAL INTELLIGENCE

AI-BASED DIABETES PREDICTION SYSTEMS

Phase 3 : Development Part 1

Topic : Building your project by loading and preprocessing the dataset.

Team Leader : Pandeeswaran C.K (pandeeswaranckit21@jkkmct.edu.in)

Team Members : Krishnan S (krishnansit21@jkkmct.edu.in)

Parthiban M (parthibanmit21@jkkmct.edu.in)

Dinesh M (dineshmit21@jkkmct.edu.in)

Naveen S (naveensit21@jkkmct.edu.in)

Diabetes Prediction

Introduction:

Diabetes prediction refers to the use of data analysis, statistical modeling, and machine learning algorithms to assess an individual's likelihood of developing diabetes. By analyzing various risk factors and health-related information, predictive models can estimate the probability of an individual developing diabetes in the future. These models are typically developed using historical data from large populations, and they can be used to identify high-risk individuals, allowing for early intervention and lifestyle modifications .

Given Dataset:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1

Necessary Steps

1.Import Libraries

2.Load the dataset

3.Exploratory Data Analytics(EDA)

4.Split the Data

5.Feature Scaling

1.Import Libraries:

Start by importing the necessary libraries

Program:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

fro sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
```

2.Load the dataset:

Once you've chosen a dataset, you'll need to load it into your programming environment. If you're using Python, you can use libraries like Pandas to read data from common file formats (e.g., CSV, Excel).

Program:

```
# Load the dataset
data = pd.read_csv('diabetes.csv')
```

3.Exploratory Data Analytics:

Perform EDA to understand your data better. This includes checking for missing values , exploring the data's statistics , visualizing it to identify patterns.

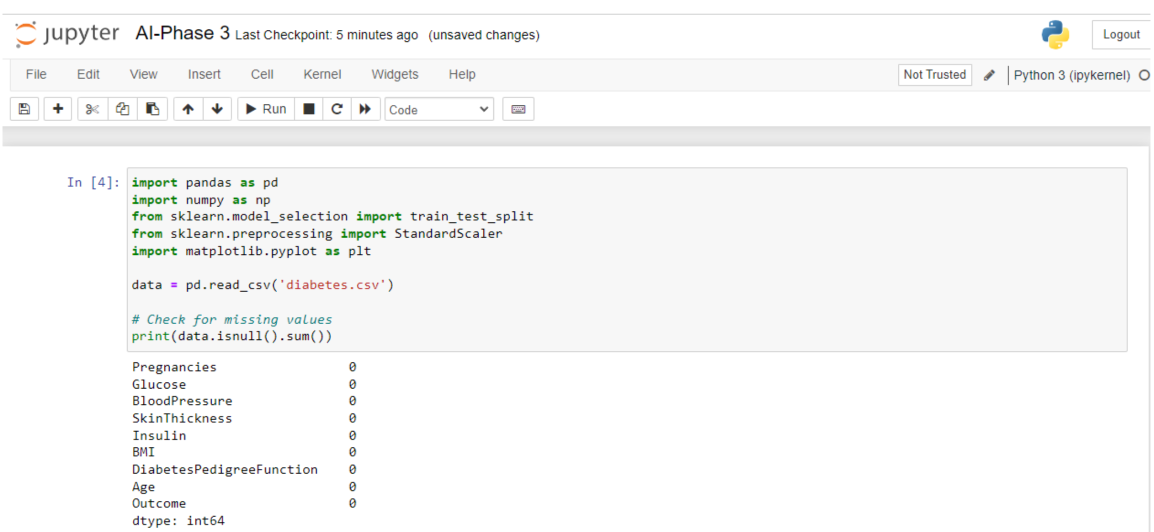
Program:

```
# Check for missing values
print(data.isnull().sum())
# Summary statistics
print(data.describe())
# Display the first few rows of the dataset
print(data.head())
#Display the Boxplot for Insulin
Import seaborn as sns
sns.boxplot=(x,data["insulin"]);
```

```
# Histogram and density graphs of all variables were accessed.
fig, ax = plt.subplots(4,2, figsize=(16,16))
sns.distplot(data.Age, bins = 20, ax=ax[0,0])
sns.distplot(data.Pregnancies, bins = 20, ax=ax[0,1])
sns.distplot(data.Glucose, bins = 20, ax=ax[1,0])
sns.distplot(data.BloodPressure, bins = 20, ax=ax[1,1])
sns.distplot(data.SkinThickness, bins = 20, ax=ax[2,0])
sns.distplot(data.Insulin, bins = 20, ax=ax[2,1])
sns.distplot(data.DiabetesPedigreeFunction, bins = 20,
ax=ax[3,0])
sns.distplot(data.BMI, bins = 20, ax=ax[3,1])


# The histagram of the Age variable was reached.
data["Age"].hist(edgecolor = "yellow");


import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve,
roc_auc_score
import seaborn as sns

# Assuming y_test and y_pred are the true labels and model
predictions
cm = confusion_matrix(y_test, y_pred)

# Create a confusion matrix plot
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
```

```
plt.show()
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test,
model.predict_proba(X_test)[:,1])
roc_auc = roc_auc_score(y_test, model.predict(X_test))
# Create an ROC curve plot
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area
= %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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")
plt.show()
```

Output :



jupyter AI-Phase 3 Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3 (ipykernel)

```
dtype: int64
```

In [5]:

```python
# Summary statistics
print(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
```

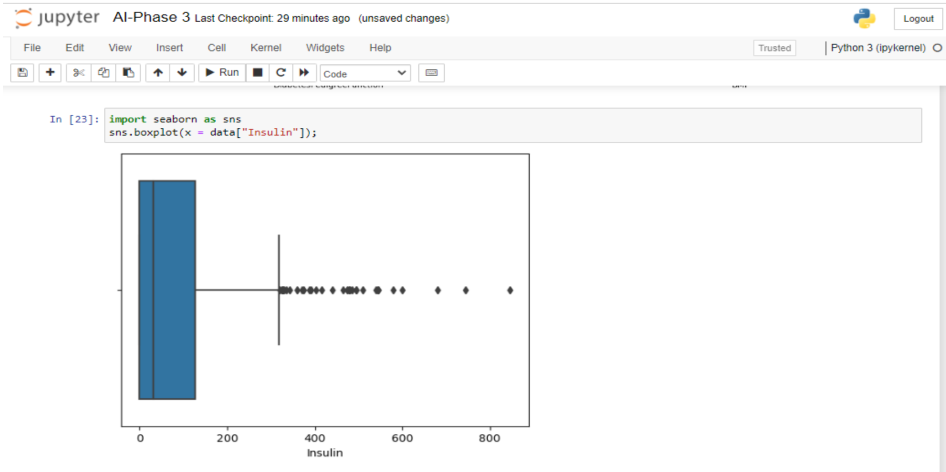
jupyter AI-Phase 3 Last Checkpoint: 6 minutes ago (autosaved)

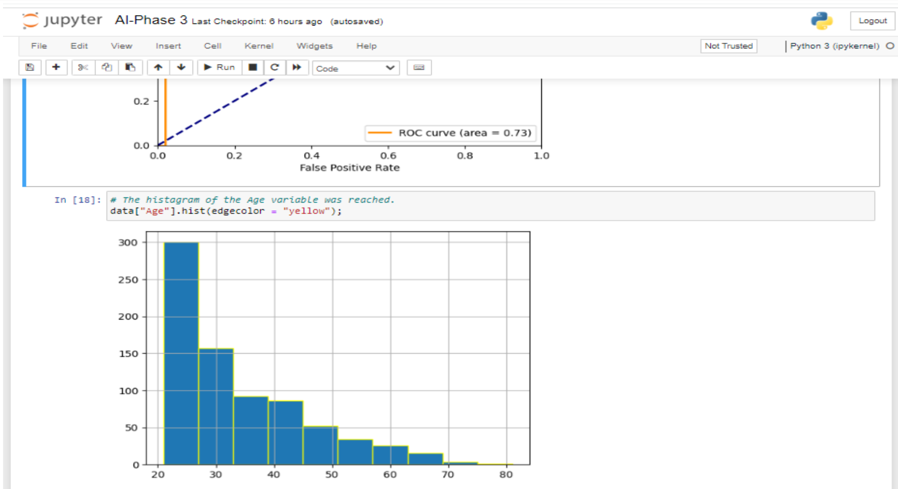
File Edit View Insert Cell Kernel Widgets Help — Trusted — Python 3 (ipykernel)

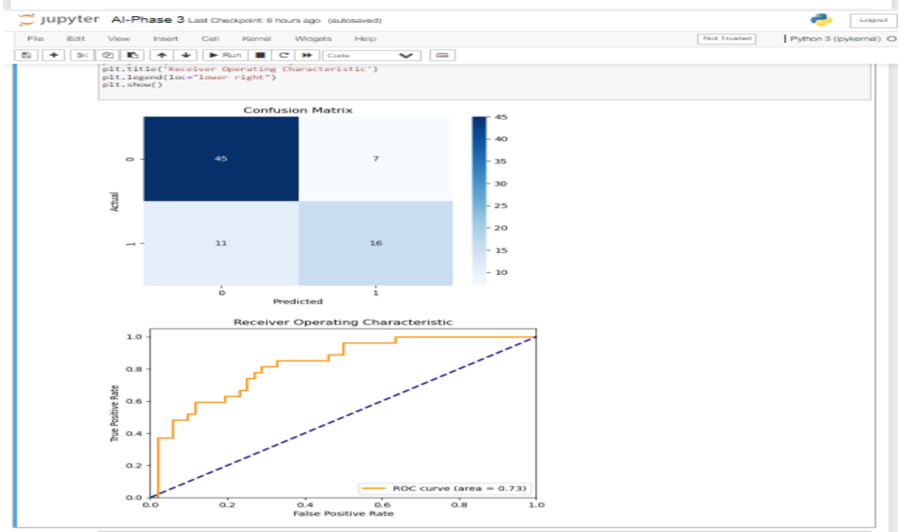
In [6]:

```python
# Display the first few rows of the dataset
print(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
```


jupyter AI-Phase 3 Last Checkpoint: 29 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [23]:
import seaborn as sns
sns.boxplot(x = data["Insulin"]);
0
200
400
600
800
Insulin


jupyter AI-Phase 3 Last Checkpoint: 6 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
0.2
0.0
ROC curve (area = 0.73)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
In [18]:
# The histogram of the Age variable was reached.
data["Age"].hist(edgecolor = "yellow");
300
250
200
150
100
50
0
20
30
40
50
60
70
80


jupyter AI-Phase 3 Last Checkpoint: 6 hours ago (autosaved)
Python 3 (ipykernel)
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
Confusion Matrix
45
7
11
16
Actual
Predicted
0
1
Receiver Operating Characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.73)

4.Split the Data:

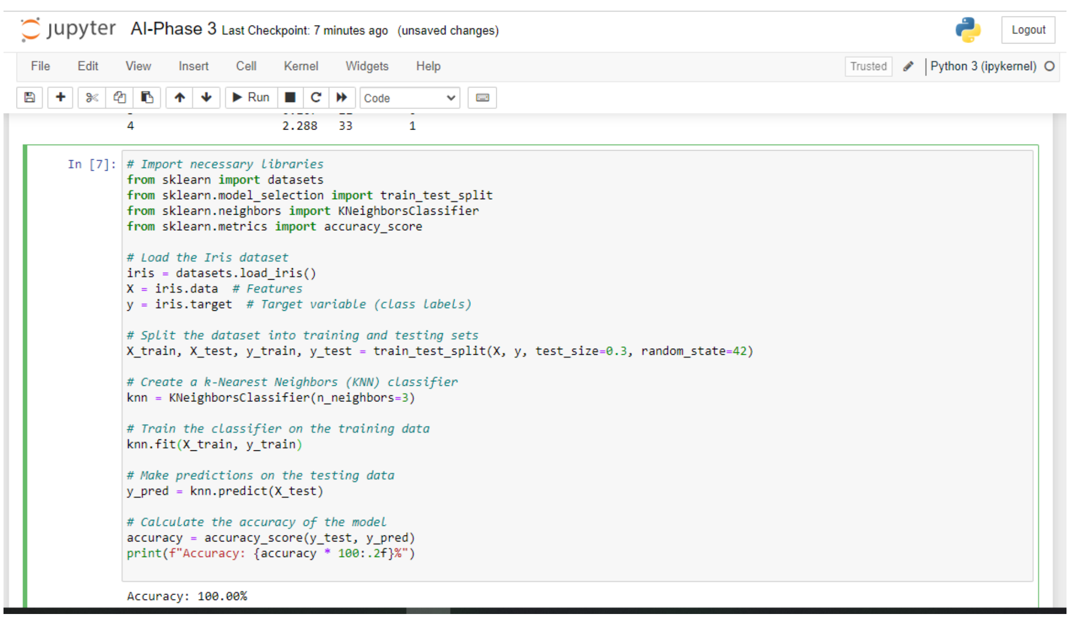
Divide your dataset into training and testing sets. The training set is used to train your AI model, while the testing set is used to evaluate its performance.

Program:

```
# Import necessary libraries

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

# Load the Iris dataset

iris = datasets.load_iris()

X = iris.data  # Features

y = iris.target  # Target variable (class labels)


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)


# Create a k-Nearest Neighbors (KNN) classifier
```

```
knn = KNeighborsClassifier(n_neighbors=3)

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

# Make predictions on the testing data
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Output :



5.Feature Scaling:

Scaling is an essential preprocessing step in many machine learning and data analysis tasks, including diabetes prediction. Scaling is particularly important when your dataset contains features with different scales or units.

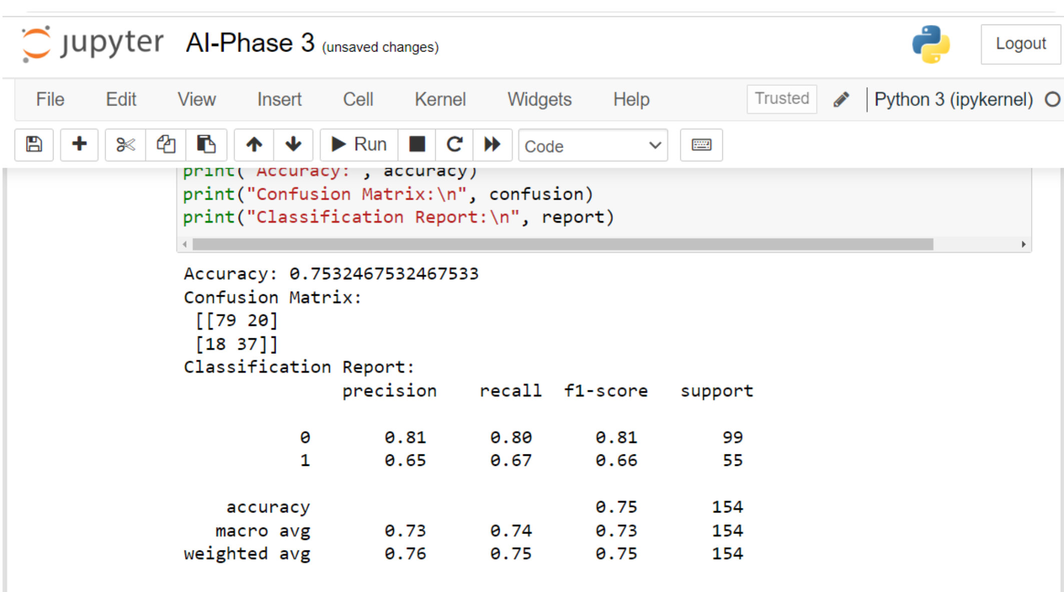
Program:

```
# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix


# Step 1: Load the dataset

data = pd.read_csv('diabetes.csv')


# Step 2: Data preprocessing

# Handle missing values (impute with the mean)

data.fillna(data.mean(), inplace=True)
```

```
# Split the data into features and target variable

X = data.drop('Outcome', axis=1)  # Features

y = data['Outcome']  # Target variable


# Split the dataset into a training set and a testing set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Standardize features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Step 3: Train a machine learning model (Logistic Regression)

model = LogisticRegression()

model.fit(X_train, y_train)


# Step 4: Make predictions

y_pred = model.predict(X_test)

# Step 5: Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)

confusion = confusion_matrix(y_test, y_pred)

report = classification_report(y_test, y_pred)


# Display evaluation metrics

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", confusion)

print("Classification Report:\n", report)
```

Output :



Preprocessing the Dataset

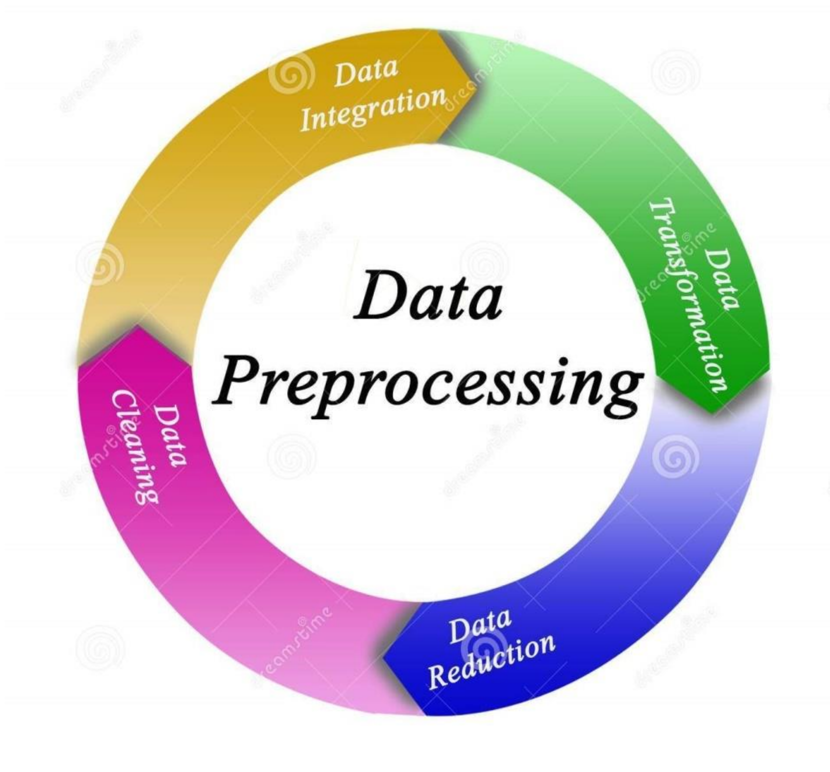
Data preprocessing is the process of

1.Data Cleaning

2. Data Transforming

3. Data Integration

4. Data Reduction



1.Data Cleaning :

This involves identifying and correcting errors and inconsistencies in the data. For example, this may involve removing duplicate records, correcting typos, and filling in missing values.

2.Data Transforming

This involves converting the data into a format that is suitable for the analysis task. For example, this may involve converting categorical data to numerical data, or scaling the data to a suitable range.

3.Data integration

This involves combining data from multiple sources into a single dataset. This may invove resolving inconsistencies in the dataset , such as different data formats or different variable names.

Program For Data Preprocessing:

```
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.metrics import accuracy_score


# Load the dataset

data = pd.read_csv('diabetes.csv')

# Data Cleaning

# Handling missing values (replace 0s with NaN in some columns)

data[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] =
data[["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]].replace(0, float('nan'))


# Handling outliers (for example, removing rows with missing values)

data.dropna(inplace=True)


# Data Integration: No integration needed in this example.


# Data Transformation

X = data.drop("Outcome", axis=1)  # Features

y = data["Outcome"]  # Target variable

# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create and train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
# Make predictions on the testing data
y_pred = model.predict(X_test)


# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Output :

```
# Load the dataset

data = pd.read_csv('diabetes.csv')

# Data Cleaning
# Handling missing values (replace 0s with NaN in some columns)
data[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] = data[["Glucose", "BloodPressure", "SkinThickness", "Insu

# Handling outliers (for example, removing rows with missing values)
data.dropna(inplace=True)

# Data Integration: No integration needed in this example.

# Data Transformation
X = data.drop("Outcome", axis=1)  # Features
y = data["Outcome"]  # Target variable

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

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 77.22%
```

Importance of loading and processing dataset:

Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for diabetes prediction models, as diabetes prediction datasets are often complex and noisy.

By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Challenges involved in loading and preprocessing a Diabetes Prediction dataset:

1.Handling missing values

2.Encoding categorical variables

3.Scaling the features

4.Splitting the dataset into training and testing sets

How to overcome the challenges of loading and preprocessing a Diabetes prediction dataset:

1.Use a data preprocessing library.

2.Carefully consider the specific needs of your model.

3. Validate the preprocessed data.

Conclusion:

Data preprocessing emerged as a pivotal aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.