<u>ML Project</u>

Question:- diabetes- predict whether patient will be diabetic or not.

ML project for predicting patient will be diabetic or not.

Excel file containing data for patients having columns Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age Outcome.

Outcome variable is (0 for non-diabetic, 1 for diabetic).

```python
import pandas as pd
df = pd.read_excel('C:/Users/Windows/OneDrive/Documents/data_scienc/ML/Proj/diabetes_updated.xlsx')
df
```

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

This imports the Pandas library, which is a powerful tool for data manipulation and analysis in Python. It provides functions for working with data in tabular formats like Excel, CSV, and others.

Read_excel function is used to read file present on specified location and save it into df dataframe.

Perform basic exploratory data analysis (EDA):

df.info() Check data types and missing value

df.describe() Summary statistics for numerical columns.

```python
print(df.info())
print(df.describe())
```

```
<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
None
       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  
```

Check weather values are null or not for each column.

```python
df.isnull().sum()
```

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

Use StandardScalar or to MinMaxScalar scale numeric features so that all values are on a similar scale. This improves the performance of certain ML models.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df.iloc[:, :-1])  # Exclude the 'Outcome' column
df_scaled
```

```
array([[ 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]])
```

Import StandardScaer class from sklearn.preprocessing module. StadardScaer is a tool for feature scaling that standardizes column..

Outcome column is not scaled it has 0 and 1 value as it used to determine diabetic and non diabetic

For that iloc function is used.it will exclude Outcome column.

Fit_trnsform() is combined metod for :-

Fits the scaler to data. i.e. calculates mean and standard deviation for each Column.

Each column is now standardized with a mean of 0 and variance of 1.

Split the dataset into training and testing sets to ensure the model is evaluated on unseen data.

```
from sklearn.model_selection import train_test_split
X = df.iloc[:, :-1]  # (all columns except 'Outcome')
y = df['Outcome']     # (Outcome column)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Imports train_test_split function.

which is used to split a dataset into training and testing subsets.

X having all columns except outcome column.

Y represent target column.

Test_size=0.2 specifies that 20% of data will be used for testing while remaining 80% will be for trainng.

Random_state=42 Ensures reproducibility by fixing the random seed. This guarantees that the data is split same way every time code is run.

```
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(y_train), y=y_train)
class_weight_dict = {i : w for i, w in enumerate(class_weights)}
```

Computes class weights automatically based on the distribution of labels in y_train.

"balanced" ensures that the model gives equal importance to both classes (diabetic & non-diabetic), even if they are imbalanced.

Creates a dictionary mapping class labels (0 for non-diabetic, 1 for diabetic) to their respective weights.

In datasets like diabetes prediction, cases of diabetes (Outcome = 1) are often less frequent than non-diabetes cases (Outcome = 0).

If the dataset is imbalanced, the model may predict "non-diabetic" more often, leading to a biased classifier.

Using class weights forces the model to pay more attention to minority classes (diabetic cases).

- Using class weights forces the model to pay more attention to minority classes (diabetic cases).

A. Using Logistic Regression:-

2. Initializing and Training Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=500, class_weight=class_weight_dict)
model.fit(X_train, y_train)
```

```
LogisticRegression

LogisticRegression(class_weight={0: 0.7675, 1: 1.4345794392523366},
                   max_iter=500)
```

What this does?

- LogisticRegression(max_iter=500, class_weight=class_weight_dict)

- max_iter=500: Allows the model to run more iterations for convergence.

- class_weight=class_weight_dict: Applies the computed class weights to balance the training.

- model.fit(X_train, y_train)

- Trains the logistic regression model on the preprocessed X_train and y_train dataset.

Logistic Regression is a supervised learning algorithm used for binary classification (e.g., predicting 0 or 1, diabetic or non-diabetic).

After training, the model can now be used to:

Predict Outcomes: Predict whether a sample is diabetic or non-diabetic using unseen data.

Check the model's accuracy and performance with metrics like accuracy score, precision, recall, etc.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 0.7467532467532467

Confusion Matrix:
 [[78 21]
 [18 37]]

Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

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

accuracy_score:- Measures the proportion of correctly predicted labels (y_pred) compared to actual labels (y_test).

confusion_matrix:- Provides a summary of prediction results in a matrix format, showing true positives, false positives, true negatives, and false negatives.

classification_report:- Displays key classification metrics like precision, recall, F1-score, and support.

model.predict(X_test):-

Uses the trained model to predict the outcomes for the test dataset (X_test).

accuracy_score(y_test, y_pred)

Compares the predicted labels () with the actual labels () and computes the percentage of correct predictions. accuracy_score(y_test, y_pred) confusion_matrix(y_test, y_pred):-

Generates a matrix summarizing the model's predictions

- . Rows represent actual classes.
- . Columns represent predicted classes.

True Positives (TP): Correctly predicted positive cases.

True Negatives (TN): Correctly predicted negative cases.

False Positives (FP): Incorrectly predicted positive cases.

False Negatives (FN): Incorrectly predicted negative cases.

classification_report(y_test, y_pred):-

Generates a detailed summary of evaluation metrics:

Precision: Ratio of true positive predictions to total positive predictions

Recall (Sensitivity): Ratio of true positive predictions to actual positive cases.

F1-score: Harmonic mean of precision and recall.

Support: Number of instances for each class in y_test

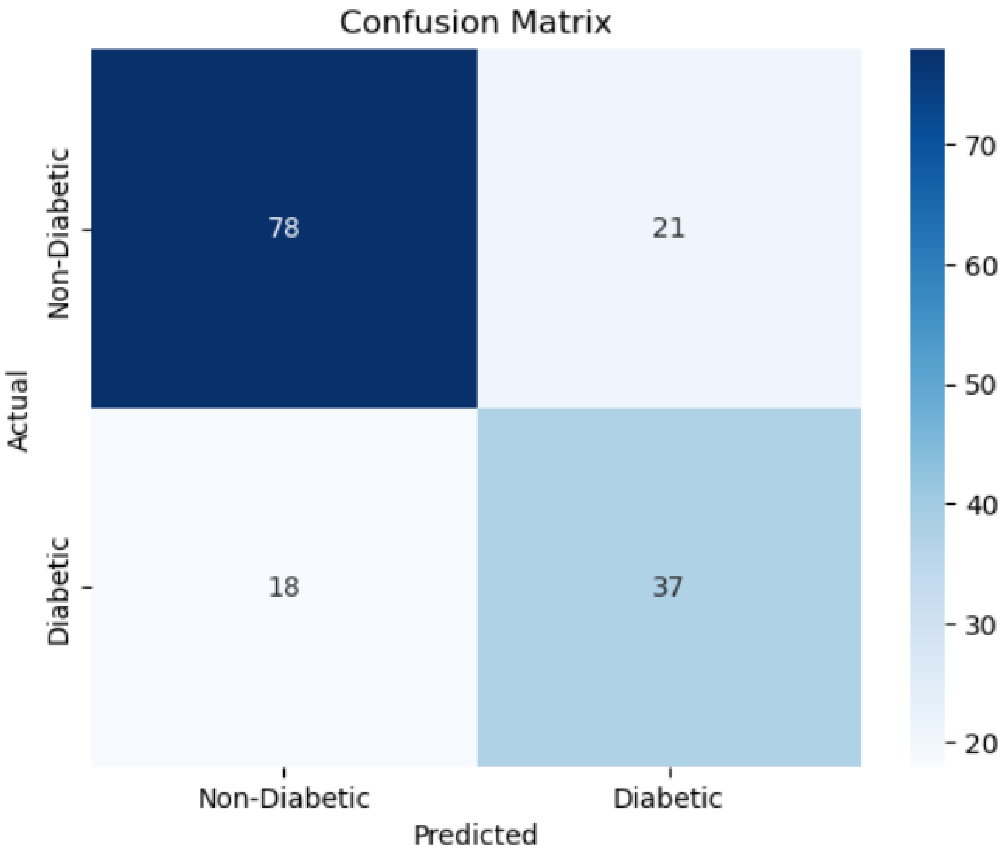
```
import matplotlib.pyplot as plt
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Non-Diabetic", "Diabetic"],
            yticklabels=["Non-Diabetic", "Diabetic"])
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.title("Confusion Matrix")
plt.show()
```

cm = confusion_matrix(y_test, y_pred):-

Generates a matrix that summarizes the performance of a classification model save it in cm.

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Non-Diabetic", "Diabetic"], yticklabels=["Non-Diabetic", "Diabetic"])

Creates a heatmap from the confusion matrix () to visualize the model's prediction results



Conclusion:-

Accuracy:-

The model correctly predicted the outcome (either "Non-Diabetic" or "Diabetic") for 75% of the test data.

Confusion Matrix:-

True Negative:-

The model correctly predicted 78 cases as "Non-Diabetic".

False Positive:-

The model incorrectly predicted 21 cases as "Diabetic" when they were actually "Non-Diabetic”.

False Negative:-

The model incorrectly predicted 18 cases as "Non-Diabetic" when they were actually "Diabetic"

True Positive:-

The model correctly predicted 37 cases as "Diabetic".

Precision::-

Class 0 (Non-Diabetic):-

Precision = 0.81:-

Out of all the samples predicted as "Non-Diabetic," 81% were correct.

This indicates good performance in predicting Non-Diabetic cases, though some false positives.

Recall = 0.79:-

Out of all the actual "Non-Diabetic" cases, 79% were correctly identified.

F1-Score = 0.80 :-

The balance between precision and recall is high, indicating overall good performance for this class.

Support = 99:-

There are 99 actual "Non-Diabetic" cases in the dataset. The performance metrics for this class are based on these samples.

Class 1 (Diabetic):

Precision = 0.64:

Out of all the samples predicted as "Diabetic," 64% were correct.

Recall = 0.67:-

Out of all the actual "Diabetic" cases, 67% were correctly identified.

F1-Score = 0.65:-

The balance between precision and recall is moderate for this class

Support = 55:-

There are 55 actual "Diabetic" cases in the dataset.

B. Using Random Forest Classifier:-

```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight="balanced")
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)
y_pred_rf
```

```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

RandomForestClassifier(n_estimators=100, random_state=42, class_weight=class_weight_dict)

n_estimators=100: Uses 100 decision trees in the forest.

random_state=42: Ensures reproducibility.

class_weight=class_weight_dict: Applies the computed class weights to handle imbalance.

rf_model.fit(X_train, y_train)

Trains the Random Forest model using the dataset.

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Calculate metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred_rf))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Accuracy Score: 0.7792207792207793

Confusion Matrix:
 [[87 13]
 [21 33]]

Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.87      0.84       100
           1       0.72      0.61      0.66        54

    accuracy                           0.78       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.77      0.78      0.77       154
```

Model achieved 77.9% accuracy, meaning it correctly predicted diabetes in about 78% of cases.

Class 0 (Non-Diabetic):

Precision (81%): When predicting non-diabetic, it's mostly correct.

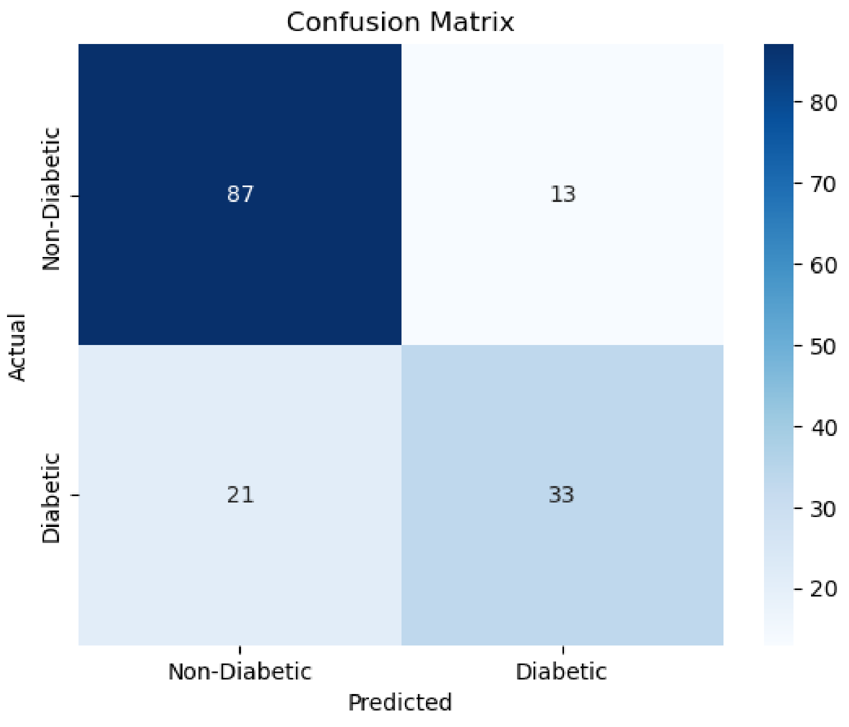
Recall (87%): It correctly identifies most non-diabetic cases.

Class 1 (Diabetic):

Precision (72%): Some false positives (non-diabetic predicted as diabetic).

Recall (61%): Lower recall means some true diabetic cases were missed.

```python
import matplotlib.pyplot as plt
import seaborn as sns

cm1 = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm1, annot=True, fmt="d", cmap="Blues", xticklabels=["Non-Diabetic", "Diabetic"],
            yticklabels=["Non-Diabetic", "Diabetic"])
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.title("Confusion Matrix")
plt.show()
```



<u>Conclusion</u>:-

<u>Accuracy:-</u>

The model correctly predicted the outcome (either "Non-Diabetic" or "Diabetic") for 78% of the test data.

Confusion Matrix:-

True Negative:-

The model correctly predicted 87 cases as "Non-Diabetic".

False Positive:-

The model incorrectly predicted 13 cases as "Diabetic" when they were actually "Non-Diabetic”.

False Negative:-

The model incorrectly predicted 21 cases as "Non-Diabetic" when they were actually "Diabetic"

True Positive:-

The model correctly predicted 33 cases as "Diabetic".