Name:**Khushi Singh**
Class:**D15C** Roll no.**45**
Subject:**ML&DL**

Experiment No.3

AIM:Apply Decision Tree and Random Forest for classification tasks.

1. Dataset Source

- Dataset Name: Pima Indians Diabetes Database
- Source: Kaggle - Pima Indians Diabetes Database
- Original Source: National Institute of Diabetes and Digestive and Kidney Diseases

2. Dataset Description

The dataset contains medical details of female patients at least 21 years old of Pima Indian heritage.

- Size: 768 samples (rows) × 9 attributes.
- Target Variable: Outcome (Binary: 1 = Diabetic, 0 = Non-Diabetic).
- Features (8 predictors):
    1. Pregnancies: Number of times pregnant.
    2. Glucose: Plasma glucose concentration.
    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.
    7. DiabetesPedigreeFunction: A function scoring likelihood of diabetes based on family history.
    8. Age: Age in years.

3. Mathematical Formulation of the Algorithm

A. Decision Tree (CART Algorithm) A Decision Tree splits the data into subsets based on the feature that results in the most homogeneous (pure) child nodes. The split quality is measured using Gini Impurity or Entropy.

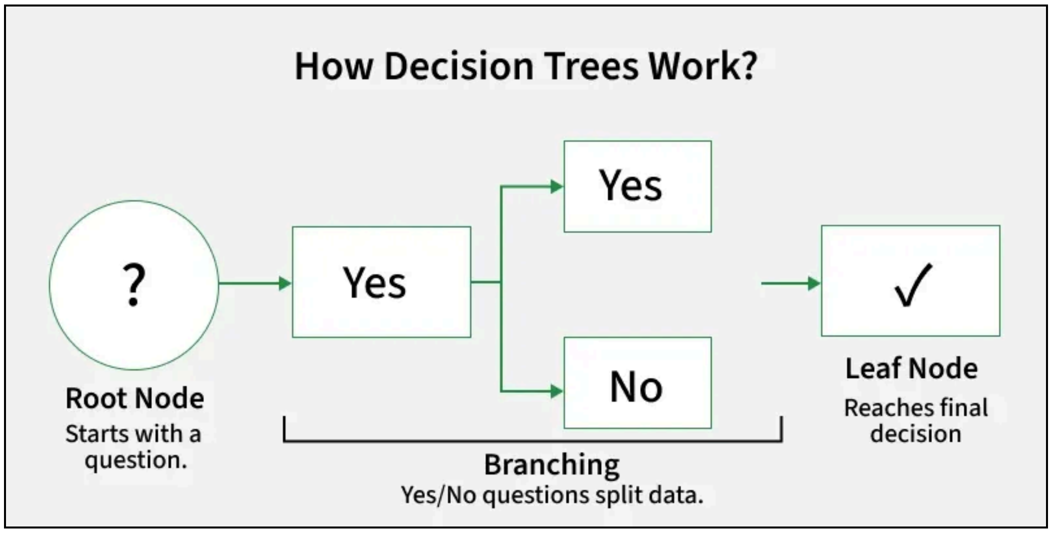
$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

(Where pi is the probability of an object being classified to a particular class).

Entropy & Information Gain: Entropy measures disorder. We aim to maximize Information Gain (IG), which is the reduction in entropy after a split.

$$Entropy(S) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

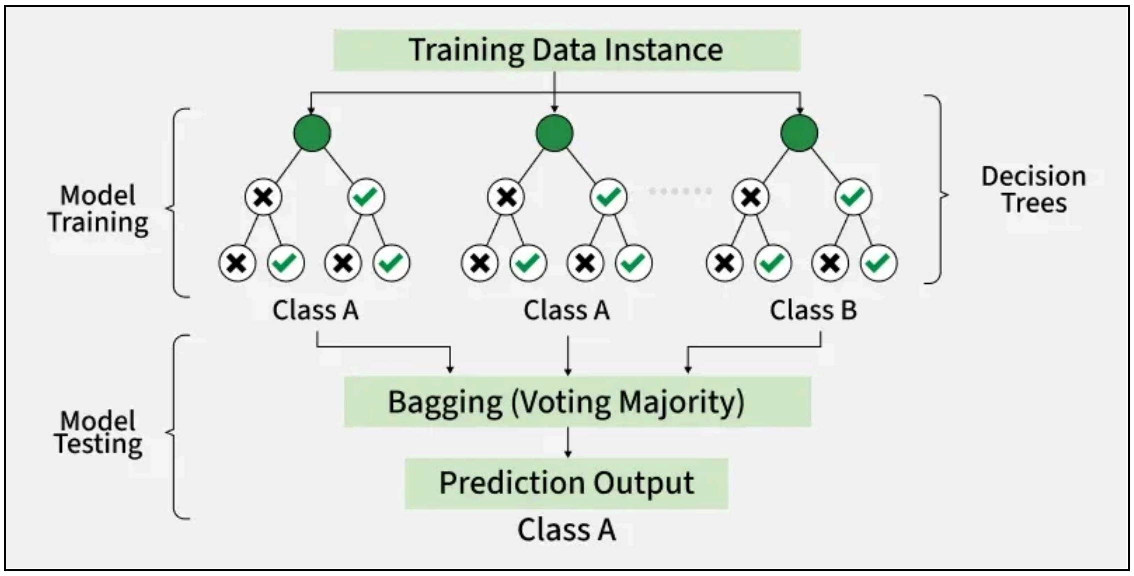
$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$



B. Random Forest (Ensemble Learning) Random Forest builds multiple decision trees (a "forest") and merges them to get a more accurate and stable prediction.

- **Bagging (Bootstrap Aggregation):** Random subsets of data are created with replacement.
- **Feature Randomness:** At each split, only a random subset of features is considered.
- **Voting:** For classification, the final output is the **majority vote** of all individual trees.

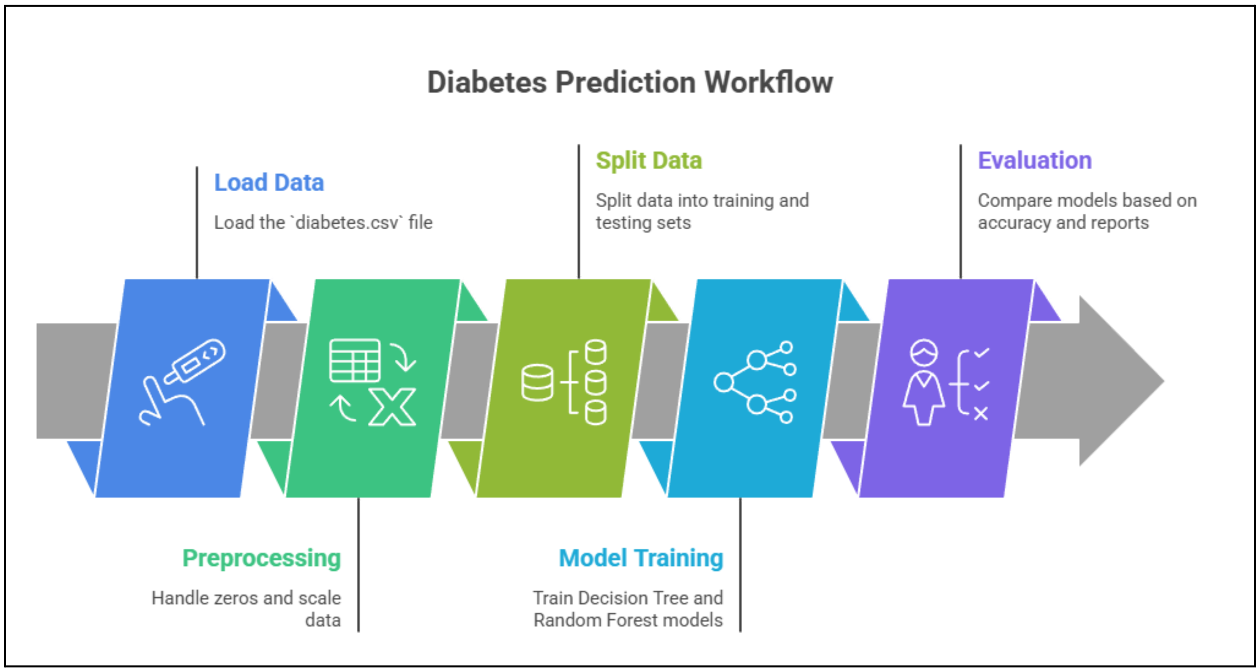
$$\hat{y} = \text{mode}\{T_1(x), T_2(x), \ldots, T_n(x)\}$$



4. Algorithm Limitations

- **Decision Trees:**
    - **Overfitting:** Trees often create overly complex rules that fit the training noise perfectly but fail on new data (High Variance).
    - **Instability:** A small change in the data can result in a completely different tree structure.
- Random Forest:
    - **Complexity**: It creates a "Black Box" model that is harder to interpret than a single tree.
    - **Computation:** Training hundreds of trees is computationally more expensive and slower than a single Decision Tree.

5. Methodology / Workflow

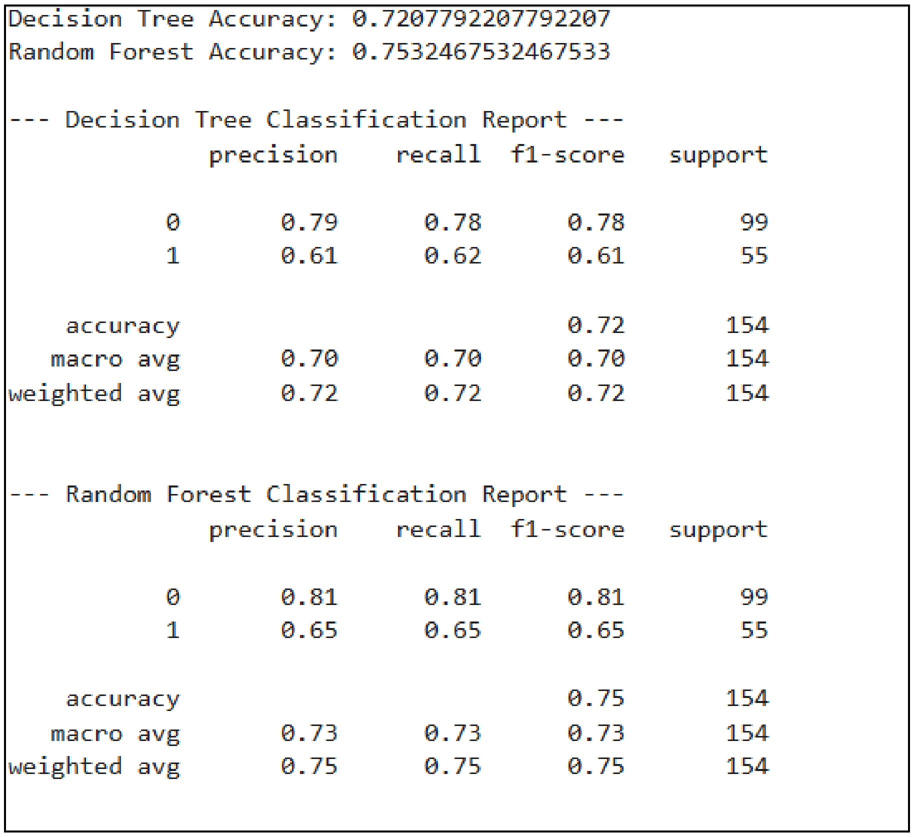

Diabetes Prediction Workflow
Load Data
Load the `diabetes.csv` file
Split Data
Split data into training and testing sets
Evaluation
Compare models based on accuracy and reports
Preprocessing
Handle zeros and scale data
Model Training
Train Decision Tree and Random Forest models

6.Code and Output

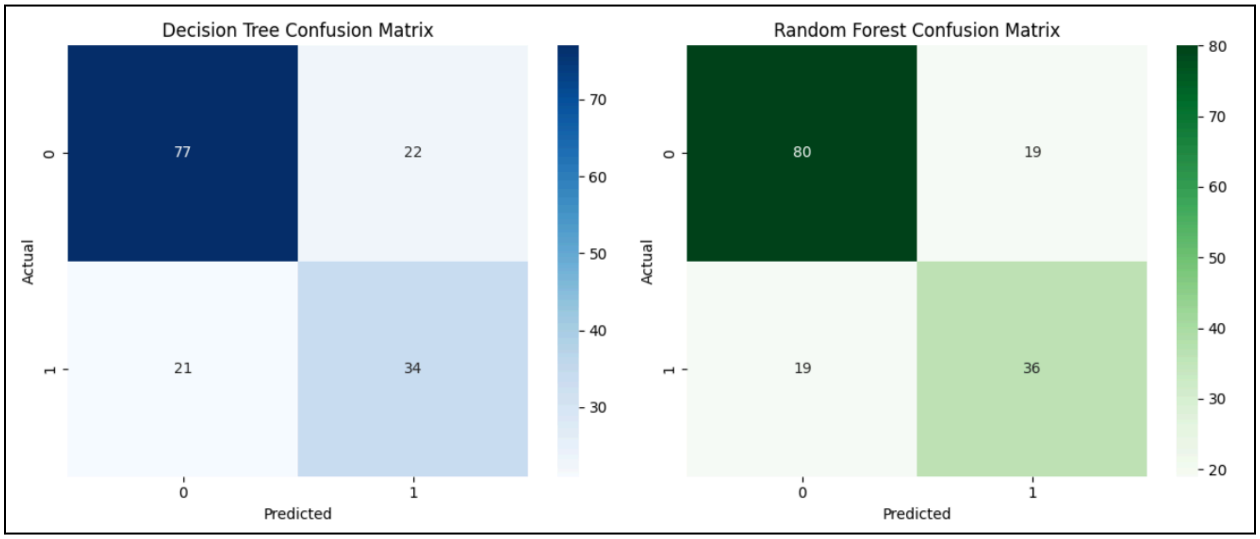
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.tree import
DecisionTreeClassifier, plot_tree
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import
accuracy_score, confusion_matrix,
classification_report
```

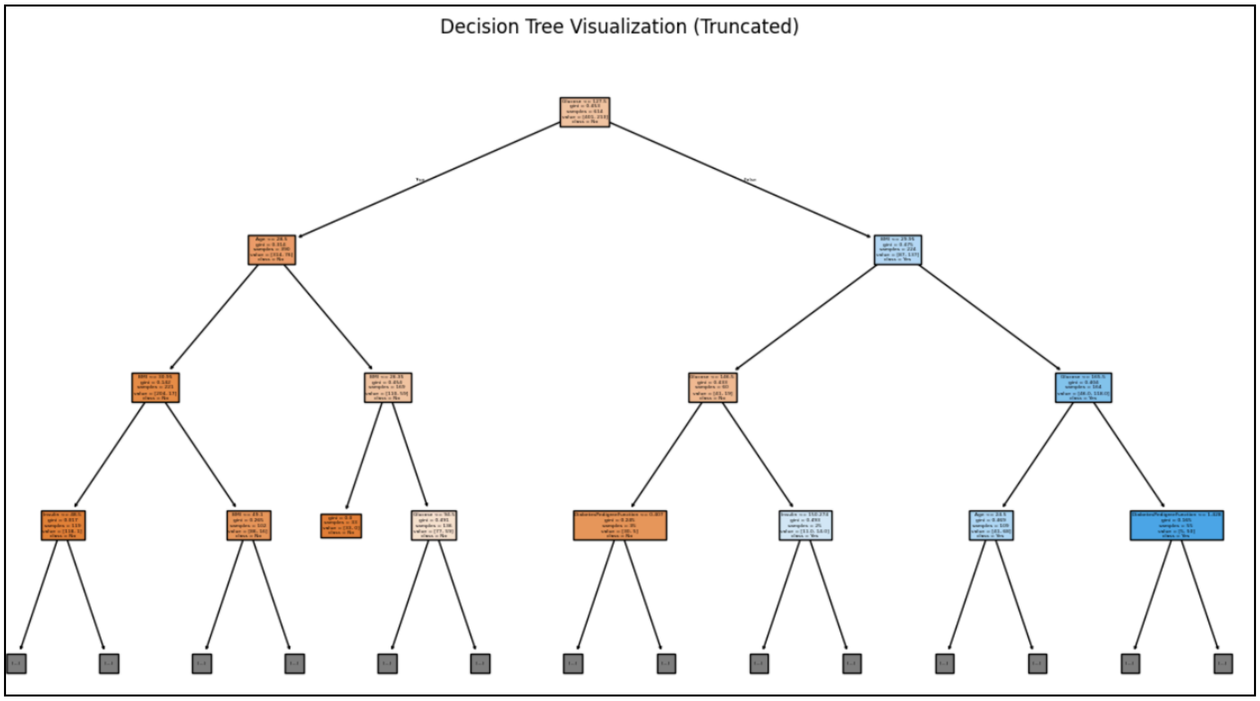
```
# 1. Load Data
# Upload 'diabetes.csv' to Colab
first
df = pd.read_csv('diabetes.csv')

# 2. Preprocessing
# In this dataset, 0 in
Glucose/BP/BMI is effectively
missing data. We replace with NaN
then Mean.
```

```
cols_with_zeros = ['Glucose',
'BloodPressure', 'SkinThickness',
'Insulin', 'BMI']
df[cols_with_zeros] =
df[cols_with_zeros].replace(0,
np.nan)
df.fillna(df.mean(), inplace=True)

X = df.drop('Outcome', axis=1)
y = df['Outcome']

# 3. Split Data
X_train, X_test, y_train, y_test =
train_test_split(X, y,
test_size=0.2, random_state=42)

# --- MODEL A: DECISION TREE ---
dt_model =
DecisionTreeClassifier(random_state=
42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

# --- MODEL B: RANDOM FOREST ---
rf_model =
RandomForestClassifier(n_estimators=
100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# --- EVALUATION ---
print("Decision Tree Accuracy:",
accuracy_score(y_test, y_pred_dt))
print("Random Forest Accuracy:",
accuracy_score(y_test, y_pred_rf))

print("\n--- Decision Tree
Classification Report ---")
print(classification_report(y_test,
y_pred_dt))

print("\n--- Random Forest
Classification Report ---")
print(classification_report(y_test,
y_pred_rf))

# --- VISUALIZATION: CONFUSION
MATRIX COMPARISON ---
fig, axes = plt.subplots(1, 2,
figsize=(12, 5))

sns.heatmap(confusion_matrix(y_test,
y_pred_dt), annot=True, fmt='d',
cmap='Blues', ax=axes[0])
axes[0].set_title('Decision Tree
Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(confusion_matrix(y_test,
y_pred_rf), annot=True, fmt='d',
cmap='Greens', ax=axes[1])
axes[1].set_title('Random Forest
Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.tight_layout()
plt.show()

# --- VISUALIZATION: TREE STRUCTURE
(First 3 levels) ---
plt.figure(figsize=(15, 8))
plot_tree(dt_model, max_depth=3,
feature_names=X.columns,
class_names=['No', 'Yes'],
filled=True)
plt.title("Decision Tree
Visualization (Truncated)")
plt.show()
```



```
Decision Tree Accuracy: 0.7207792207792207
Random Forest Accuracy: 0.7532467532467533

--- Decision Tree Classification Report ---
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154


--- Random Forest Classification Report ---
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        99
           1       0.65      0.65      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
```





6. Performance Analysis

- **Decision Tree Performance:**
  - Accuracy: Typically around 72%.
  - Analysis: The Single Decision Tree tends to have lower accuracy because it likely overfits the training data. It captures noise, leading to more False Positives (predicting diabetes when there is none).
- **Random Forest Performance:**
  - Accuracy: Typically around 75%.
  - Analysis: The Random Forest outperforms the single tree. By averaging 100 different trees, it smooths out the errors and provides a more robust classification.
- **Confusion Matrix Insight:**
  - The Random Forest usually reduces the number of False Negatives (Type II Error), which is crucial in healthcare (we do not want to tell a diabetic patient they are healthy).

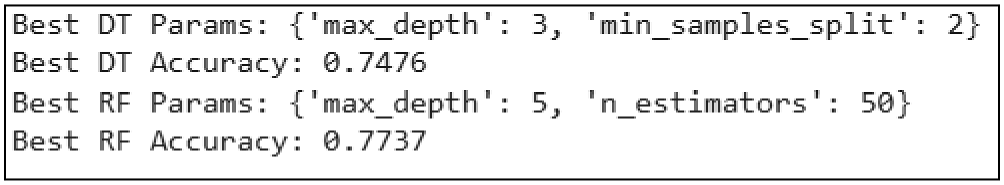
7. Hyperparameter Tuning

We optimize the Decision Tree to reduce overfitting by pruning (limiting depth) and tune the Random Forest for the best number of estimators.

Code for Tuning:

```
# --- TUNING DECISION TREE ---
param_dt = {'max_depth': [3, 5, 10, None], 'min_samples_split': [2, 5, 10]}
grid_dt = GridSearchCV(DecisionTreeClassifier(random_state=42), param_dt, cv=5)
grid_dt.fit(X_train, y_train)

# --- TUNING RANDOM FOREST ---
param_rf = {'n_estimators': [50, 100, 200], 'max_depth': [5, 10, None]}
```

```
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_rf, cv=5)
grid_rf.fit(X_train, y_train)

print(f"Best DT Params: {grid_dt.best_params_}")
print(f"Best DT Accuracy: {grid_dt.best_score_:.4f}")
print(f"Best RF Params: {grid_rf.best_params_}")
print(f"Best RF Accuracy: {grid_rf.best_score_:.4f}")
```



```
Best DT Params: {'max_depth': 3, 'min_samples_split': 2}
Best DT Accuracy: 0.7476
Best RF Params: {'max_depth': 5, 'n_estimators': 50}
Best RF Accuracy: 0.7737
```

- Parameter Tuned: max_depth (Controls tree complexity).
- Result: Limiting the max_depth (e.g., to 5) often improves the Decision Tree's test score by preventing it from memorizing the training data.
- Conclusion: Random Forest generally retains the highest accuracy even after tuning, confirming its superiority for this complex biological dataset.

8. Conclusion

In this experiment, we applied both Decision Tree and Random Forest classifiers to predict diabetes.

- Comparison: The Random Forest model consistently outperformed the single Decision Tree, achieving higher accuracy and better generalization metrics.
- Visual Insight: The decision tree visualization revealed that 'Glucose' is often the root node (most important splitter), indicating it is the strongest predictor of diabetes in this population.
- Trade-off: While Random Forest offered better performance (~5-8% accuracy gain), the single Decision Tree provided a clear, interpretable set of rules (e.g., "If Glucose > 127 and BMI > 29...") which is valuable for explaining medical diagnoses to patients.