Machine Learning – Assignment 2

Ganesh G (25MML0043)

Link: https://drive.google.com/drive/folders/1HIDK8_eav175ae1AAMzFhqMIeCRqa6cM?usp=sharing

1. Decision Tree

AIM:

To understand and apply decision tree algorithms for multi-class classification problems by building, evaluating, and interpreting a decision tree model that predicts prescribed drug types based on patient medical and demographic features.

INTRODUCTION:

A decision tree is a supervised machine learning algorithm used for classification tasks, among others. It works by recursively splitting the data based on feature values into subsets, aiming to create groups as homogeneous as possible with respect to the target variable. This splitting process forms a tree-like structure with a root node, internal decision nodes representing feature-based tests, branches showing the outcomes of these tests, and leaf nodes which provide the classification labels or predictions.

REAL WORLD APPLICATION:

1. Healthcare

- Disease diagnosis: Doctors use decision trees to determine whether a patient has a certain disease based on symptoms, test results, and medical history.
- Example: Predicting if a patient has diabetes using age, BMI, and glucose level.

2. Finance

- Loan approval: Banks apply decision trees to decide whether to approve a loan by analyzing income, employment history, credit score, and repayment history.
- Fraud detection: Identifying fraudulent transactions by analyzing transaction amount, location, and spending patterns.

3. Retail & Marketing

- Customer segmentation: Retailers use decision trees to classify customers into groups (e.g., high spenders, bargain seekers) for targeted marketing.

- Churn prediction: Telecom companies predict whether a customer will leave based on usage data and complaints.

4. Manufacturing

- **Quality control**: Used to predict whether a product will pass or fail inspection based on features like material quality, temperature, or machine settings.

5. Education

- **Student performance prediction**: Schools use decision trees to predict whether a student will pass/fail based on attendance, study hours, and past grades.

ALGORITHM:

1. **Start with the full dataset** as the root node.
2. **Check stopping criteria**
    - If all data points belong to the same class → make this a **leaf node** with that class.
    - If no features remain → make this a **leaf node** with the majority class.
3. **Select the best feature to split**
    - Calculate a **splitting criterion** (such as *Information Gain*, *Gain Ratio*, or *Gini Index*) for each feature.
    - Choose the feature that best separates the data.
4. **Split the dataset** into subsets based on the chosen feature's values.
5. **Create child nodes** for each subset.
6. **Repeat steps 2–5** recursively for each child until:
    - Maximum depth is reached, OR
    - All nodes are pure (only one class), OR
    - No improvement in splitting criterion.
7. **Label leaf nodes** with the majority class (if pure class not possible).

Implementation and results:

Step1: Data Set Overview and preprocessing

```python
#Step1: Data Set Overview and preprocessing
#Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
```

Ganesh G
25MML0043

```python
#load data set
df = pd.read_csv('drug200.csv')

# Display basic info and first few rows
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
None
   Age Sex      BP Cholesterol  Na_to_K   Drug
0   23   F    HIGH        HIGH   25.355  drugY
1   47   M     LOW        HIGH   13.093  drugC
2   47   M     LOW        HIGH   10.114  drugC
3   28   F  NORMAL        HIGH    7.798  drugX
4   61   F     LOW        HIGH   18.043  drugY
```

Ganesh G
25MML0043

```python
# Summary statistics
print(df.describe())
```

```
              Age     Na_to_K
count  200.000000  200.000000
mean    44.315000   16.084485
std     16.544315    7.223956
min     15.000000    6.269000
25%     31.000000   10.445500
50%     45.000000   13.936500
75%     58.000000   19.380000
max     74.000000   38.247000
```

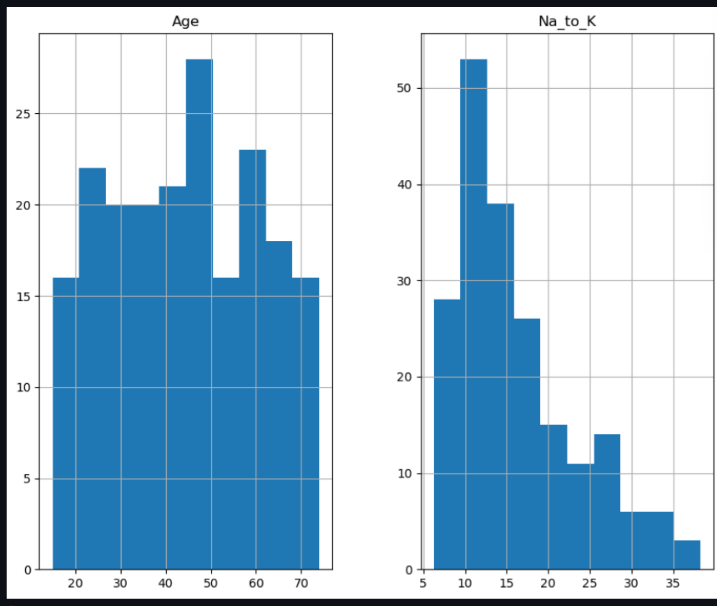
Ganesh G
25MML0043

```python
# Check for missing values
print(df.isnull().sum())
```

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

```python
# Visualize distributions of numerical features
df.hist(figsize=(10,8))
plt.show()
```

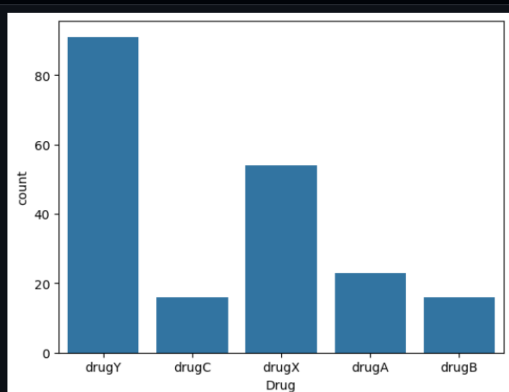
[6]



Ganesh G
25MML0043

```python
# Visualize counts of target classes
sns.countplot(x='Drug', data=df)
plt.show()
```

[7]



Ganesh G
25MML0043


```python
# Encode categorical features: Sex, BP, Cholesterol
categorical_cols = ['Sex', 'BP', 'Cholesterol']
le = LabelEncoder()

for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

print(df.head())
```

[16]

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	drugY
1	47	1	1	0	13.093	drugC
2	47	1	1	0	10.114	drugC
3	28	0	2	0	7.798	drugX
4	61	0	1	0	18.043	drugY

Ganesh G
25MML0043

Step 2: Decision Tree Model Building

```python
#Step 2: Decision Tree Model Building
from sklearn.tree import DecisionTreeClassifier

# Define features and target
X = df.drop('Drug', axis=1)
y = df['Drug']

# Split into train and test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=42)

# Train the model
clf.fit(X_train, y_train)
```

Ganesh G
25MML0043

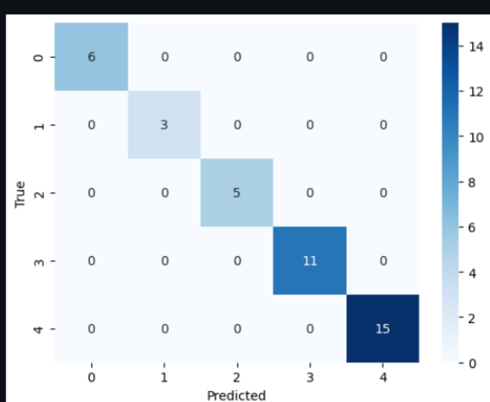
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=42)
```

Step 3: Model Evaluation

```python
#Step 3: Model Evaluation
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns

# Predict on test data
y_pred = clf.predict(X_test)

# Accuracy
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")

# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Detailed classification report
print(classification_report(y_test, y_pred))
```

Ganesh G
25MML0043

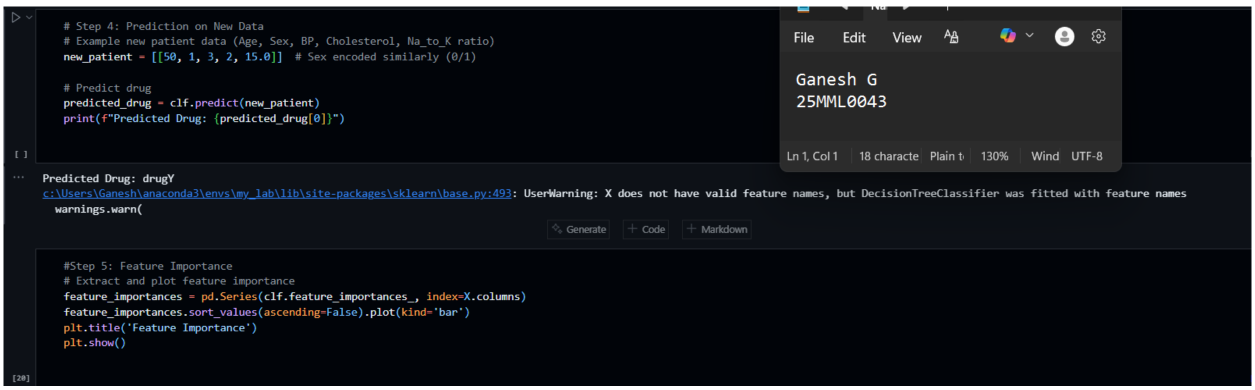
```
Accuracy: 1.0000
```



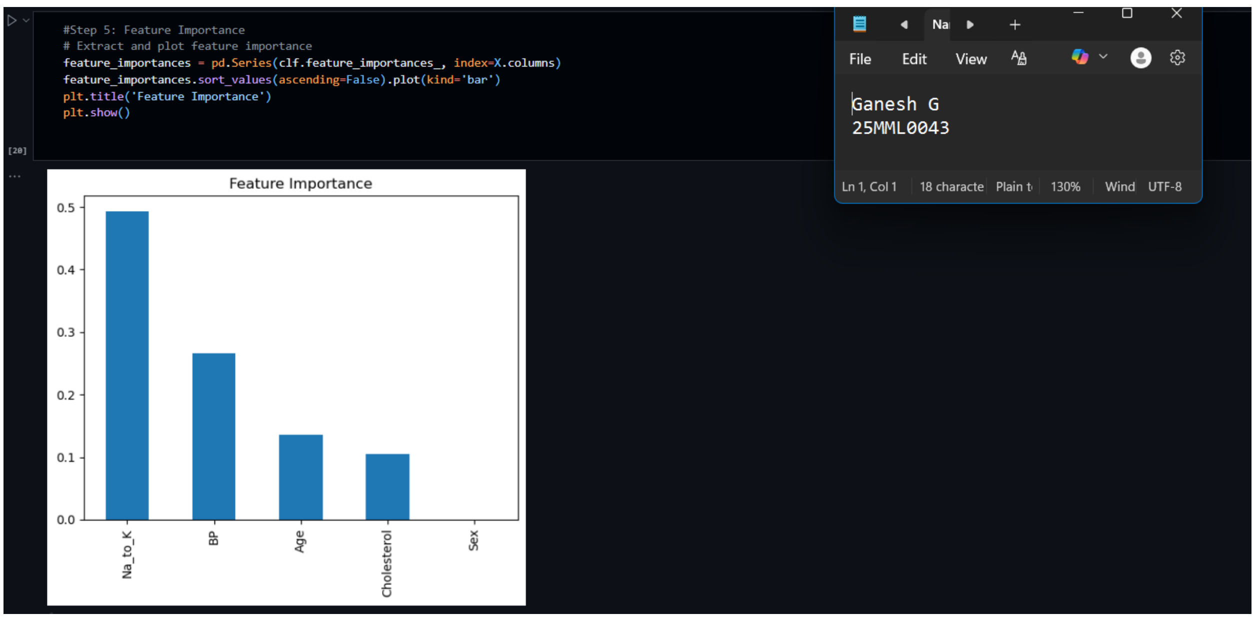
Ganesh G
25MML0043

```
              precision    recall  f1-score   support

       drugA       1.00      1.00      1.00         6
       drugB       1.00      1.00      1.00         3
       drugC       1.00      1.00      1.00         5
       drugX       1.00      1.00      1.00        11
       drugY       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

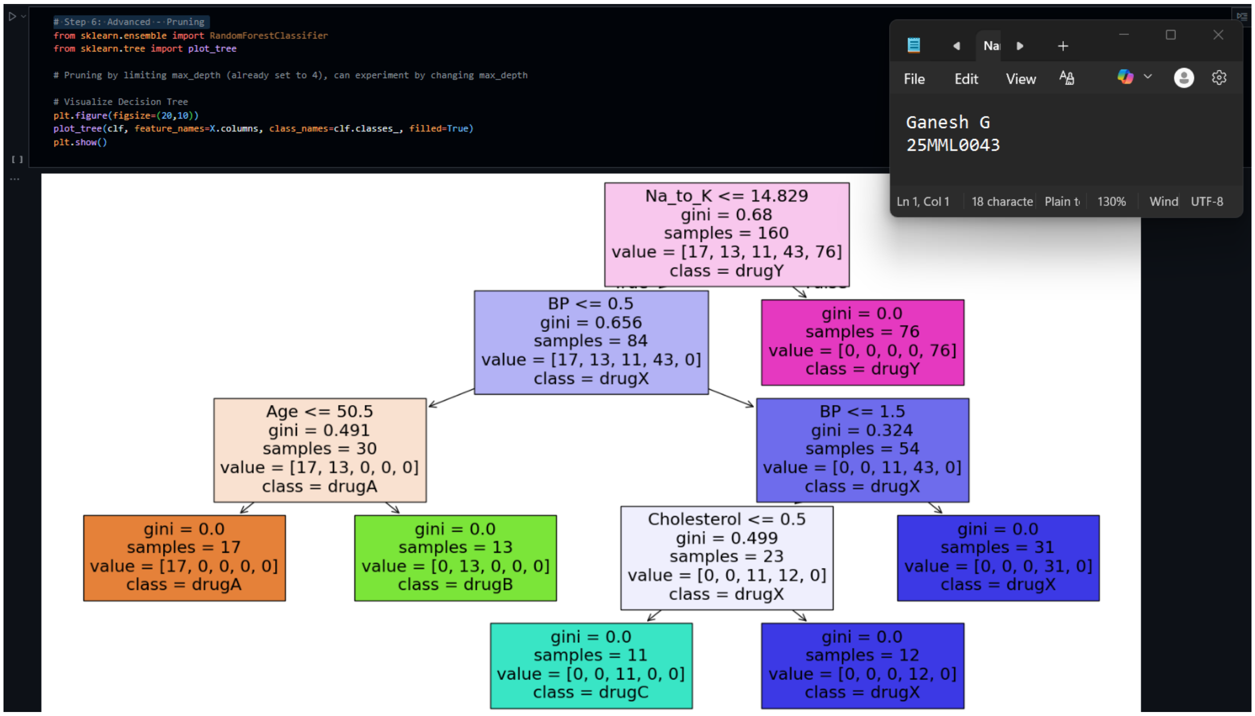
Step 4: Prediction on New Data



Step 5: Feature Importance



Step 6: Advanced - Pruning



2. Random Forest

AIM:

The aim for analyzing a college student placement dataset typically involves predicting whether a student will be placed based on various features like academic performance, skills, internships, work experience, and other factors. Common goals include:

- Predicting the likelihood of a student getting placed using classification algorithms.
- Identifying the important factors influencing placement success.
- Evaluating model performance with accuracy, precision, recall, and F1-score.
- Helping students and institutions improve preparation and strategies for campus recruitment.

Introduction:

The primary objective of this lab is to develop a model that predicts whether a student will be placed in campus recruitment based on various academic and non-academic attributes. Campus placements are a vital milestone for students and institutions as they reflect the outcome of academic efforts and influence the reputation of educational institutes. By analyzing historical placement data using machine learning classification algorithms like

Decision Tree, Logistic Regression, or Naive Bayes, this lab aims to forecast placement likelihood for current students. Predictive modeling helps the placement cell identify potential candidates early and focus on improving their skills to maximize placement chances. This system also provides insights into key factors affecting placement success, enabling data-driven decisions to enhance student career outcomes and institutional placement performance. The lab involves data pre-processing, training machine learning models, and evaluating their accuracy for reliable placement prediction.

Real World Application:

1. Healthcare

- Disease prediction: Used to predict whether a patient has a disease (like diabetes, cancer, or heart disease) based on lab test values, medical history, and lifestyle.
- Example: Classifying breast cancer tumors as *benign* or *malignant* (Random Forest performs well on medical datasets).

2. Finance

- **Credit scoring & loan approval**: Banks use Random Forests to assess whether a person is a good or risky borrower by analyzing income, spending habits, and credit history.
- **Fraud detection**: Identifying unusual transactions that could be fraud.

3. E-commerce & Retail

- **Recommendation systems**: Predict what products a customer might be interested in, based on past purchases and browsing behavior.
- **Customer churn prediction**: Telecom or subscription-based businesses predict which customers are likely to leave, so they can offer retention plans.

4. Cybersecurity

- Intrusion detection systems: Random Forests detect unusual patterns in network traffic that may signal hacking or malware attacks.

5. Text & Sentiment Analysis

- Spam email detection: Filtering spam vs. non-spam emails.
- Sentiment classification: Analyzing whether a product review is positive or negative.

ALGORITHM:

Input: Training dataset D with n samples and m features.

For each tree (repeat K times):

- **Bootstrap sampling**: Randomly select a subset of the data (with replacement) from D.
- **Feature selection**: At each split in the tree, choose a random subset of features (instead of all features).
- **Build a decision tree** using the chosen data and feature subset (often grown fully, without pruning).

Aggregate results:

- **Classification**: Each tree votes for a class. The class with the most votes is the final prediction.
- **Regression**: Take the average of all tree predictions

Implementation and results:

Import libraries and Load dataset

```
# 1. Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import numpy as np
```

File Edit View

Ganesh G
25MML0043

Ln 1, Col 1 18 characte Plain t 140% Wind UTF-8

```
# 2. Load Data
df = pd.read_csv("college_student_placement_dataset.csv")
df.head()
```

	College_ID	IQ	Prev_Sem_Result	CGPA	Academic_Performance	Internship_Experience	Extra_Curricular_Score	Communication_Skills	Projects_Completed	Placement
0	CLG0030	107	6.61	6.28	8	No	8	8	4	No
1	CLG0061	97	5.52	5.37	8	No	7	8	0	No
2	CLG0036	109	5.36	5.83	9	No	3	1	1	No
3	CLG0055	122	5.47	5.75	6	Yes	1	6	1	No
4	CLG0004	96	7.91	7.69	7	No	8	10	2	No

Exploratory Data Analysis (EDA)

```python
# 3. Exploratory Data Analysis (EDA)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   College_ID             10000 non-null  object
 1   IQ                     10000 non-null  int64
 2   Prev_Sem_Result        10000 non-null  float64
 3   CGPA                   10000 non-null  float64
 4   Academic_Performance   10000 non-null  int64
 5   Internship_Experience  10000 non-null  object
 6   Extra_Curricular_Score 10000 non-null  int64
 7   Communication_Skills   10000 non-null  int64
 8   Projects_Completed     10000 non-null  int64
 9   Placement              10000 non-null  object
dtypes: float64(2), int64(5), object(3)
memory usage: 781.4+ KB
None
Placement
No     0.8341
Yes    0.1659
Name: proportion, dtype: float64
```

File Edit View

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8

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

```
                 IQ  Prev_Sem_Result          CGPA  Academic_Performance  \
count  10000.000000     10000.000000  10000.000000          10000.000000
mean      99.471800         7.535673      7.532379              5.546400
std       15.053101         1.447519      1.470141              2.873477
min       41.000000         5.000000      4.540000              1.000000
25%       89.000000         6.290000      6.290000              3.000000
50%       99.000000         7.560000      7.550000              6.000000
75%      110.000000         8.790000      8.770000              8.000000
max      158.000000        10.000000     10.460000             10.000000

       Extra_Curricular_Score  Communication_Skills  Projects_Completed
count            10000.000000          10000.000000        10000.000000
mean                 4.970900              5.561800            2.513400
std                  3.160103              2.900866            1.715959
min                  0.000000              1.000000            0.000000
25%                  2.000000              3.000000            1.000000
50%                  5.000000              6.000000            3.000000
75%                  8.000000              8.000000            4.000000
max                 10.000000             10.000000            5.000000
```

File Edit View

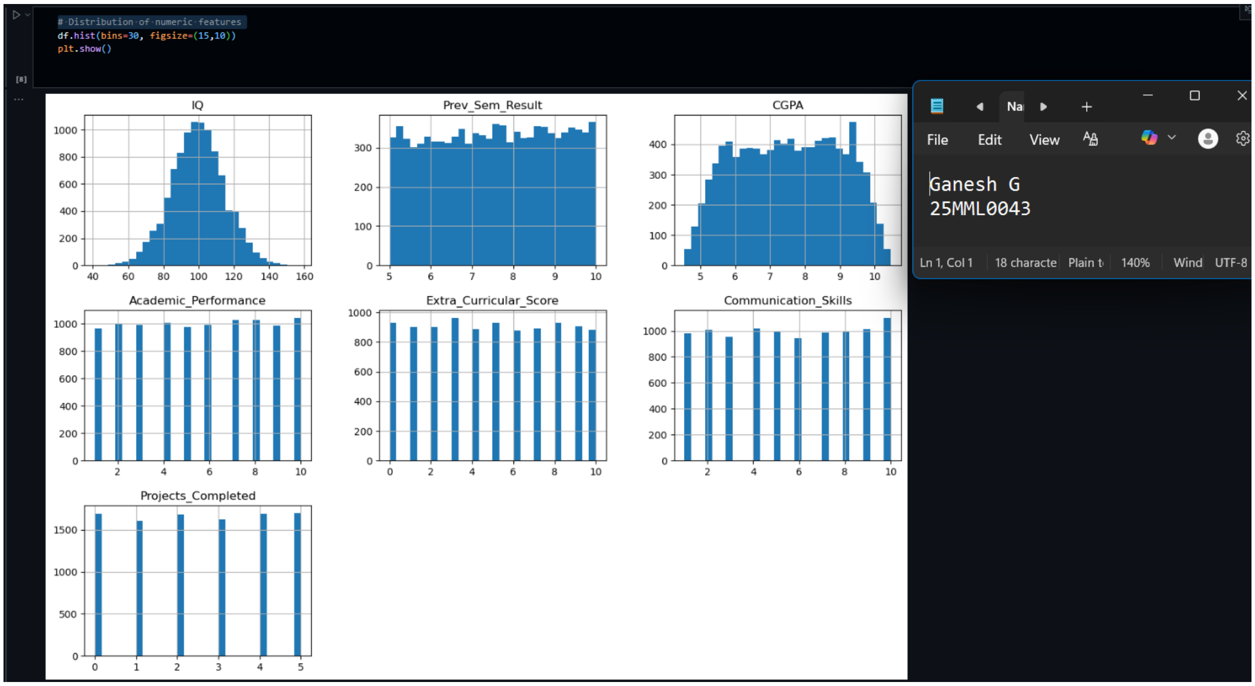
Ganesh G
25MML0043

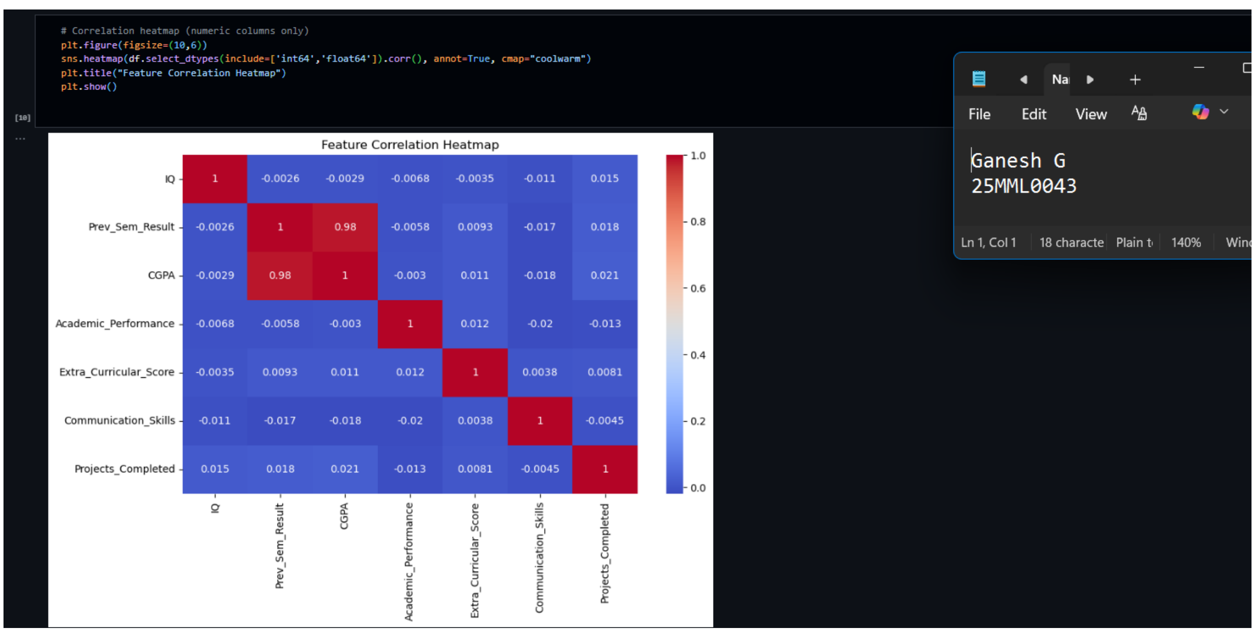
Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8

```python
print(df["Placement"].value_counts(normalize=True))
```

```
Placement
No     0.8341
Yes    0.1659
Name: proportion, dtype: float64
```

Distribution of numeric features



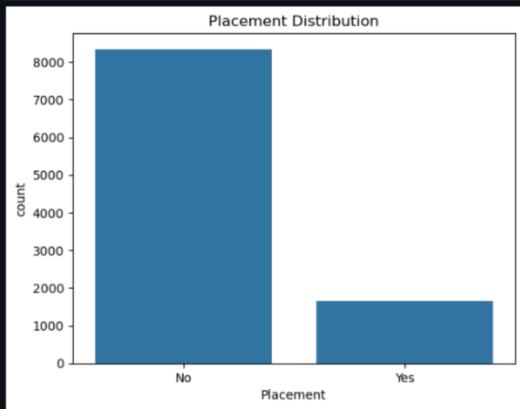


```python
# Placement counts
sns.countplot(x="Placement", data=df)
plt.title("Placement Distribution")
plt.show()
```

[11]

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8



Data Preprocessing

```python
# 4. Data Preprocessing

# Drop College_ID (not useful for prediction)
df = df.drop("College_ID", axis=1)

# Encode categorical variables
for col in df.select_dtypes(include=["object"]).columns:
    df[col] = LabelEncoder().fit_transform(df[col])

# Features and Target
X = df.drop("Placement", axis=1)
y = df["Placement"]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[12]

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8

```python
# 5. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
```

[ ]

```python
# 6. Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
```

[14]

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Model Evaluation

```
# 7. Model Evaluation
y_pred = rf.predict(X_test)

print("Random Forest Performance")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

[15]

```
Random Forest Performance
Accuracy: 0.999
Precision: 1.0
Recall: 0.9939759036144579
F1-score: 0.9969788519637462

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1668
           1       1.00      0.99      1.00       332

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8

```
# 8. Feature Importance
feature_importances = rf.feature_importances_
sorted_idx = np.argsort(feature_importances)[::-1]

plt.figure(figsize=(10,6))
sns.barplot(x=feature_importances[sorted_idx], y=X.columns[sorted_idx])
plt.title("Feature Importance - Random Forest")
plt.show()
```

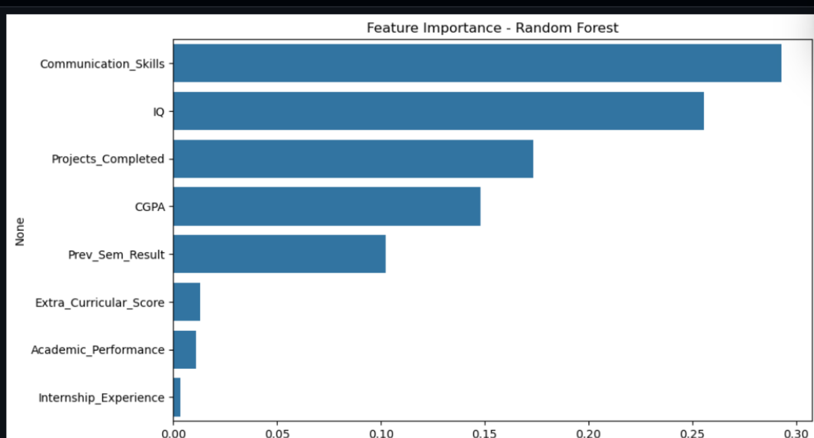
Feature Importance

```
# 8. Feature Importance
feature_importances = rf.feature_importances_
sorted_idx = np.argsort(feature_importances)[::-1]

plt.figure(figsize=(10,6))
sns.barplot(x=feature_importances[sorted_idx], y=X.columns[sorted_idx])
plt.title("Feature Importance - Random Forest")
plt.show()
```

[16]

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8



Hyperparameter Tuning

```
# 9. Hyperparameter Tuning
param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=3,
    scoring="f1",
    n_jobs=-1
)
grid_search.fit(X_train, y_train)

print("Best Hyperparameters:", grid_search.best_params_)
best_rf = grid_search.best_estimator_
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

Ganesh G
25MML0043

Evaluation of Tuned model

```
# Evaluate tuned model
y_pred_tuned = best_rf.predict(X_test)
print("Tuned Random Forest Performance")
print("Accuracy:", accuracy_score(y_test, y_pred_tuned))
print("Precision:", precision_score(y_test, y_pred_tuned))
print("Recall:", recall_score(y_test, y_pred_tuned))
print("F1-score:", f1_score(y_test, y_pred_tuned))
```

```
Tuned Random Forest Performance
Accuracy: 0.999
Precision: 1.0
Recall: 0.9939759036144579
F1-score: 0.9969788519637462
```

Ganesh G
25MML0043

predicting the new student from the given values

```
#predicting the new student from the given values
# Example new student data (change values as needed)
new_data = pd.DataFrame([{
    "IQ": 110,
    "Prev_Sem_Result": 8.2,
    "CGPA": 8.5,
    "Academic_Performance": 9,
    "Internship_Experience": "Yes",
    "Extra_Curricular_Score": 7,
    "Communication_Skills": 8,
    "Projects_Completed": 3
}])

# Encode categorical variable using same encoding
new_data["Internship_Experience"] = LabelEncoder().fit(["No","Yes"]).transform(new_data["Internship_Experience"])

# Scale using the same scaler fitted earlier
new_data_scaled = scaler.transform(new_data)

# Predict placement
prediction = best_rf.predict(new_data_scaled)
prediction_proba = best_rf.predict_proba(new_data_scaled)

print("Prediction (0=No, 1=Yes):", prediction[0])
print("Prediction Probability:", prediction_proba)
```

```
Prediction (0=No, 1=Yes): 1
Prediction Probability: [[0.05352778 0.94647222]]
```

Ganesh G
25MML0043

Models Comparison

```python
#optional
# ========================
# 10. Baseline Models Comparison
# ========================

# Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_lr = log_reg.predict(X_test)

print("\nLogistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))

# Random Forest
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_tuned))
```

[20]

```
Logistic Regression Accuracy: 0.9035
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 0.999
```

File Edit View

Ganesh G
25MML0043

Ln 1, Col 1 | 18 characte | Plain t | 140% | Wind | UTF-8