Mini Project Report :Defaulter Prediction

BATCH : B1
Student Names:
Prajakta Jadhav-UCE2022516
Sanjana Inapakolla-UCE2022510
Ishani Deshmukh-UCE2022512

Synopsis :

1)Title: Attendance-Based Defaulter Prediction System

2)Problem Statement:

Our project aims to develop an efficient system for predicting student defaulters based on their attendance records. By analyzing attendance patterns, we seek to forecast whether a student is likely to default on academic obligations.

3) Introduction:

Our project focuses on developing a predictive system to identify student defaulters based on their attendance records using supervised learning techniques, specifically employing the Random Forest classification model. The dataset provided is in CSV format, containing labels denoted as YES or NO, indicating whether a student is likely to default.

The core aim of our project is to provide educators with a tool that can predict student defaulters for any given month by analyzing their attendance data. By inputting the attendance for the upcoming month, our system generates predictions for all students, aiding teachers in proactive intervention and support.

Key Features:

1)Random Forest Model: Our system employs a Random Forest classification model to predict student defaulters. Random Forest is a powerful algorithm that works by creating multiple decision trees and combining their predictions. It's known for its ability to handle complex datasets effectively and produce accurate results.
2)Real-time Prediction: Educators can input the attendance data for the next month, and our system generates predictions for all students promptly, enabling timely intervention.
3)Data Integrity Check: Our system includes functionality to identify missing values in the CSV file, ensuring data accuracy and completeness.Which helps teacher to modify CSV file easily.
4)Outlier Detection: Using outlier detection techniques, our system flags instances where incorrect attendance entries have been made by teachers, such as attendance exceeding the total number of lectures.
5)Visualization: Our system offers visualizations depicting student defaulter rates, allowing educators to gain insights into attendance patterns and potential risk factors.
6)Model Comparison: Additionally, our project analyzes the performance of different classification models, including K-Nearest Neighbors (KNN) and Logistic Regression, evaluating their accuracy and potential for overfitting. Among these models, Random Forest emerges as the most effective choice for predicting student defaulters.

By combining machine learning with data visualization and model analysis, our project aims to provide educators with a comprehensive tool for proactive student support, ultimately enhancing academic success and retention rates.

3)Data set information (link,few data samples etc)

The dataset contains the following attributes:

CNumber: Unique identification number assigned to each student.
Name of the Student: Name of the student.
Attendance for each subject: IoT , SE, HS-OB, OE1-SC, OEI-ICCF, IOTL.
Defaulter Status for Each Subject: Indicates whether the student is a defaulter for each subject.
Overall Defaulter Status: Indicates whether the student is an overall defaulter, based on being a defaulter in at least 3 subjects.

[Google Collab]
[Datasets]

4) Code and output:

Mount Google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

o/p:

```
Mounted at /content/drive
```

```
Import the necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from scipy import stats
import matplotlib.pyplot as plt
```

K-NN algorithm It giving wrong output

```
# Load the dataset
df = pd.read_csv("/content/drive/MyDrive/ML datasets/DefaulterList -
Sheet1 (1).csv")

# Encode the target variable
le = LabelEncoder()
df['Defaulter'] = le.fit_transform(df['Defaulter'])
print("Columns in the DataFrame:")
print(df.columns)
# Selecting only the specified columns for features
selected_columns = ['IoT', 'SE', 'HS-OB', 'OE1- SC', 'OEI-ICCF', 'IOTL']
X = df[selected_columns]
# Target variable
y = df['Defaulter']
```

O/P:

```
Columns in the DataFrame:
Index(['CNumber', 'Name of the Student', 'IoT', 'SE', 'HS-OB', 'OE1- SC',
       'OEI-ICCF', 'IOTL', 'Defaulter(IoT)', 'Defaulter(SE)',
```

```
       'Defaulter(HS-OB)', 'Defaulter(OE1- SC)', 'Defaulter(OEI-ICCF)',
       'Defaulter(IOTL)', 'Defaulter'],
      dtype='object')
```

Training and Testing

```
# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

print("Missing values in X_test:", X_test.isnull().sum())
```

O/P:

```
Missing values in X_test: IoT         0
SE           0
HS-OB        0
OE1- SC      0
OEI-ICCF     0
IOTL         0
dtype: int64
```

Remove the outliers

```
outlier_indices = []
for column in selected_columns:
    outlier_index = df[df[column] > 30].index
    outlier_indices.extend(outlier_index)
    if len(outlier_index) > 0:
        print("Outliers for", column, ":")
        for idx in outlier_index:
            print("CNumber:", df.at[idx, 'CNumber'], ", Name:", df.at[idx,
'Name of the Student'], ", Subject:", column, ", Value:", df.at[idx,
column])

# Remove outliers from the dataset
df.drop(outlier_indices, inplace=True)
```

O/P:

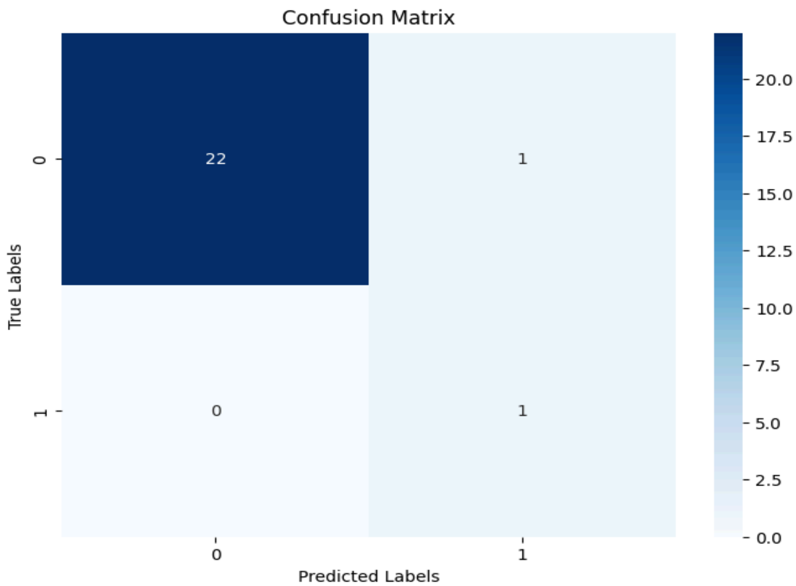
```
Outliers for IoT :
CNumber: C22018221430 , Name: AGARWAL MUSKAN , Subject: IoT , Value: 100
CNumber: C22018221431 , Name: JOSHI ADITI ANANT , Subject: IoT , Value:
100
```

Implementing Random Forest

```
from sklearn.ensemble import RandomForestClassifier
# Initialize the Random Forest classifier
```

```python
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

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

# Predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
print("Classification Report:")
print(classification_report(y_test, y_pred))
```
O/P:
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        23
           1       0.50      1.00      0.67         1

    accuracy                           0.96        24
   macro avg       0.75      0.98      0.82        24
weighted avg       0.98      0.96      0.96        24
```

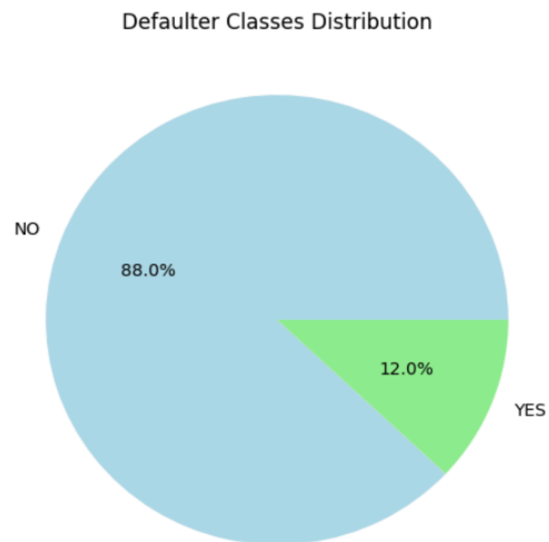
Confusion Matrix
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=rf_classifier.classes_,
yticklabels=rf_classifier.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



Pie chart

```
# Map the label values to 'YES' and 'NO'
df['Defaulter'] = df['Defaulter'].map({1: 'YES', 0: 'NO'})

# Count the occurrences of each label in the target variable
label_counts = df['Defaulter'].value_counts()

# Plotting the pie chart
plt.figure(figsize=(8, 6))
plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
colors=['lightblue', 'lightgreen'])
plt.title('Defaulter Classes Distribution')
plt.show()
```

O/P

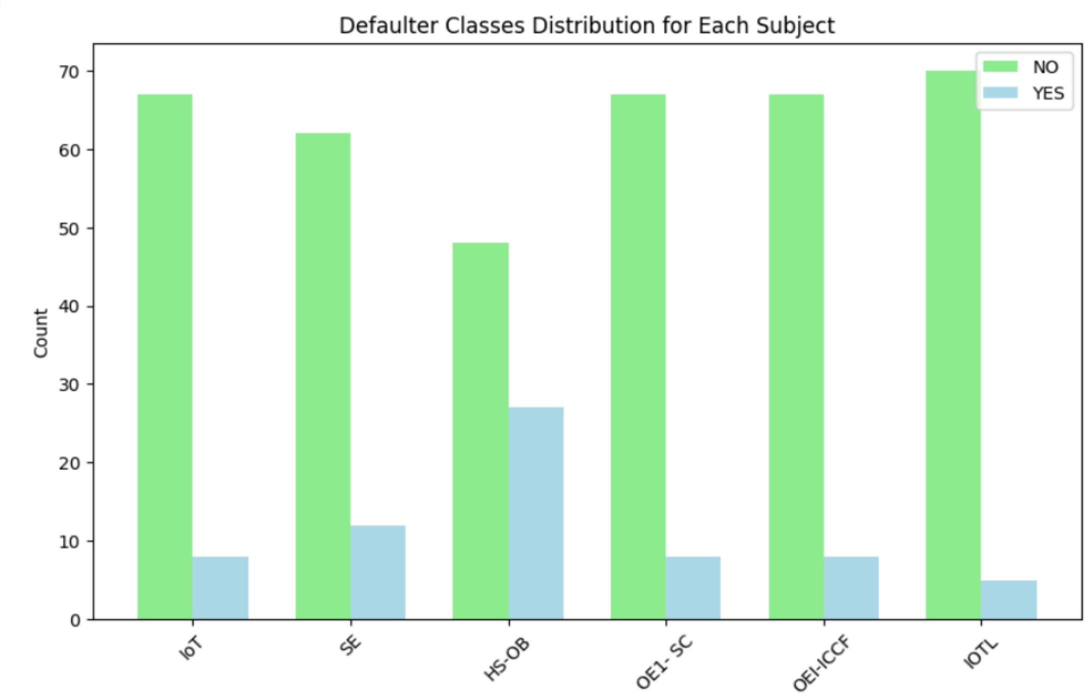


Side-by-side chart

```
import matplotlib.pyplot as plt

# Count the occurrences of "YES" and "NO" for each subject column
defaulter_counts = {}
subjects = ['IoT', 'SE', 'HS-OB', 'OE1- SC', 'OEI-ICCF', 'IOTL']

for subject in subjects:
    counts = df['Defaulter(' + subject + ')'].value_counts()
    defaulter_counts[subject] = counts

# Extract counts for YES and NO
yes_counts = [defaulter_counts[subject].get('YES', 0) for subject in
subjects]
no_counts = [defaulter_counts[subject].get('NO', 0) for subject in
subjects]

# Plotting
fig, ax = plt.subplots(figsize=(10, 6))

# Bar width
bar_width = 0.35

# Bar positions
bar_positions = range(len(subjects))
```

```
# Plotting "NO" counts
ax.bar([pos - bar_width/2 for pos in bar_positions], no_counts, bar_width,
label='NO', color='lightgreen')

# Plotting "YES" counts
ax.bar([pos + bar_width/2 for pos in bar_positions], yes_counts,
bar_width, label='YES', color='lightblue')

# Adding labels and title
ax.set_ylabel('Count')
ax.set_title('Defaulter Classes Distribution for Each Subject')
ax.set_xticks(bar_positions)
ax.set_xticklabels(subjects)
ax.legend()

plt.xticks(rotation=45)
plt.show()
```

O/P:



Testing

```
#USER INPUT TESTING
student_attendance = {
```

```
    'IoT': 1,
    'SE': 1,
    'HS-OB':10,
    'OE1- SC': 12,
    'OEI-ICCF': 0,
    'IOTL': 0
}

# Convert the student's attendance into DataFrame
student_df = pd.DataFrame([student_attendance])

# Predict whether the student is a defaulter or not using Random Forest
classifier
defaulter_prediction = rf_classifier.predict(student_df)
defaulter_prediction_label = le.inverse_transform(defaulter_prediction)
print("Predicted Defaulter Label:", defaulter_prediction_label[0])
```

O/P:

```
Predicted Defaulter Label: YES
```

Predict Defaulter for next month data

```
# Load the dataset
df1 = pd.read_csv("/content/Defaulter_NextMonth.csv")

# Initialize the LabelEncoder
le = LabelEncoder()

# Fit the LabelEncoder on the target variable 'Defaulter'
le.fit(df1['Defaulter'])

# Selecting only the specified columns for features
selected_columns = ['IoT', 'SE', 'HS-OB', 'OE1- SC', 'OEI-ICCF', 'IOTL']
X1 = df1[selected_columns]

# Predict using the Random Forest classifier
y_pred = rf_classifier.predict(X1)

# Map predictions to 'Yes' and 'No' labels
```

```
y_pred_labels = ['YES' if label == 1 else 'NO' for label in y_pred]

# Assign predicted labels to the 'Defaulter' column
df1['Defaulter'] = y_pred_labels

# Save the DataFrame back to the CSV file
df1.to_csv("/content/Defaulter_NextMonth.csv", index=False)
print(df1["Defaulter"])
```

O/P:

```
0      NO
1      NO
2      NO
3      NO
4      NO
     ...
72     NO
73     NO
74     NO
75    YES
76     NO
Name: Defaulter, Length: 77, dtype: object
```

Logistic regression is also giving wrong output

```
from sklearn.linear_model import LogisticRegression

# Load the dataset
df2 = pd.read_csv("/content/drive/MyDrive/ML datasets/DefaulterList - Sheet1 (1).csv")

# Encode the target variable
le = LabelEncoder()
df2['Defaulter'] = le.fit_transform(df2['Defaulter'])

# Selecting only the specified columns for features
selected_columns = ['IoT', 'SE', 'HS-OB', 'OE1- SC', 'OEI-ICCF', 'IOTL']
X = df2[selected_columns]
y = df2['Defaulter']

# Split the data 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)
```

```python
# Initialize the Logistic Regression classifier
logistic_regression = LogisticRegression()

# Train the classifier
logistic_regression.fit(X_train, y_train)

# Evaluate the classifier on the test set
y_pred = logistic_regression.predict(X_test)
print("Classification Report on Test Set:")
print(classification_report(y_test, y_pred))

# Define the student's attendance data for prediction
student_attendance = {
    'IoT': 10,
    'SE': 10,
    'HS-OB': 10,
    'OE1- SC': 12,
    'OEI-ICCF': 0,
    'IOTL': 0
}

# Convert the student's attendance into DataFrame
student_df = pd.DataFrame([student_attendance])

# Predict whether the student is a defaulter or not using Logistic
Regression classifier
defaulter_prediction = logistic_regression.predict(student_df)

# Convert the predicted label back to the original label
defaulter_prediction_label = le.inverse_transform(defaulter_prediction)

print("Predicted Defaulter Label:", defaulter_prediction_label[0])
```
O/P:
```
Classification Report on Test Set:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93        16
           1       0.00      0.00      0.00         0

    accuracy                           0.88        16
```

```
   macro avg       0.50      0.44      0.47        16
weighted avg       1.00      0.88      0.93        16

Predicted Defaulter Label: YES
```

Implementing the KNN model

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Initialize the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train, y_train)

# Predictions on the test set
y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

O/P:

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       1.00      1.00      1.00         1

    accuracy                           1.00        24
   macro avg       1.00      1.00      1.00        24
weighted avg       1.00      1.00      1.00        24
```

5) Conclusion:

By analyzing attendance data across multiple subjects, our predictive model successfully identifies students at risk of defaulting on academic obligations. Leveraging a threshold of defaulting in at least 3 subjects to determine overall defaulter status, our system provides actionable insights for educators to intervene and support at-risk students effectively, ultimately promoting academic success and retention.

6) References:

Random Forest Algorithm in Machine Learning
Random Forest Regression in Python
K Nearest Neighbors with Python | ML
Logistic Regression in Machine Learning
Matplotlib Tutorial