Name: Dhinesh Kandra

University and campus: VIT Vellore

Branch: BTECH CSE with Specialization in Data Science

Registration number: 21BDS0030

Course name: Smart Bridge – Artificial Intelligence & Machine Learning in collaboration with Google

Batch: Morning

Assignment 4:September 22 2023

AIM:

To perform data preprocessing on the Employee Attrition dataset and to prepare it for model building using logistic regression, decision tree, and random forest algorithms.

Dataset Introduction :

The Employee Attrition dataset contains information about employees, including whether they left the company or not. The aim is to prepare the dataset for further analysis by handling missing values, encoding categorical variables, scaling continuous variables, and splitting the dataset into training and test sets.

Data Collection :

Downloading The Employee Attrition Dataset :

In [1]:

```
import gdown
file_id = '1w9qAqyddPcpZSXlF600YhEG8brvSyuBx'
output_file = '/content/Employee-Attrition.csv'
gdown.download(f'https://drive.google.com/uc?id={file_id}', output_file, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1w9qAqyddPcpZSXlF600YhEG8brvSyuBx
To: /content/Employee-Attrition.csv
100%|██████████| 228k/228k [00:00<00:00, 5.48MB/s]
```

Out[1]:

```
'/content/Employee-Attrition.csv'
```

Data Preprocessing :

Importing Necessary Libraries :

In [2]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Loading The Employee Attrition Dataset :

In [3]:

```
data = pd.read_csv('/content/Employee-Attrition.csv')
```

Displaying all Column Names :

In [4]:

```
column_names = data.columns
print("Column Names:")
for column in column_names:
    print(column)
```

```
Column Names:
Age
Attrition
BusinessTravel
DailyRate
Department
DistanceFromHome
Education
EducationField
EmployeeCount
EmployeeNumber
EnvironmentSatisfaction
Gender
HourlyRate
JobInvolvement
JobLevel
JobRole
JobSatisfaction
MaritalStatus
MonthlyIncome
MonthlyRate
NumCompaniesWorked
Over18
OverTime
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StandardHours
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
```

Checking For Missing Values :

In [5]:

```
print(data.isnull().any())
```

```
Age                         False
Attrition                   False
BusinessTravel              False
DailyRate                   False
Department                  False
DistanceFromHome            False
Education                   False
EducationField              False
EmployeeCount               False
EmployeeNumber              False
EnvironmentSatisfaction     False
Gender                      False
HourlyRate                  False
JobInvolvement              False
JobLevel                    False
JobRole                     False
JobSatisfaction             False
MaritalStatus               False
MonthlyIncome               False
MonthlyRate                 False
NumCompaniesWorked          False
Over18                      False
OverTime                    False
PercentSalaryHike           False
PerformanceRating           False
RelationshipSatisfaction    False
StandardHours               False
StockOptionLevel            False
TotalWorkingYears           False
TrainingTimesLastYear       False
WorkLifeBalance             False
YearsAtCompany              False
YearsInCurrentRole          False
YearsSinceLastPromotion     False
YearsWithCurrManager        False
dtype: bool
```

Checking For Outliers :

In [6]:

```
all_columns = data.columns
outliers_count = {}
for column in all_columns:
    if pd.api.types.is_numeric_dtype(data[column]):
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        count = len(data[(data[column] < lower_bound) | (data[column] > upper_bound)])
        outliers_count[column] = count
    else:
        count = len(data[column].unique())
        outliers_count[column] = count
for column in all_columns:
    print(f'Number of outliers in {column}: {outliers_count[column]}')
```

```
Number of outliers in Age: 0
Number of outliers in Attrition: 2
Number of outliers in BusinessTravel: 3
Number of outliers in DailyRate: 0
Number of outliers in Department: 3
Number of outliers in DistanceFromHome: 0
Number of outliers in Education: 0
Number of outliers in EducationField: 6
Number of outliers in EmployeeCount: 0
Number of outliers in EmployeeNumber: 0
Number of outliers in EnvironmentSatisfaction: 0
Number of outliers in Gender: 2
Number of outliers in HourlyRate: 0
Number of outliers in JobInvolvement: 0
Number of outliers in JobLevel: 0
Number of outliers in JobRole: 9
Number of outliers in JobSatisfaction: 0
Number of outliers in MaritalStatus: 3
Number of outliers in MonthlyIncome: 114
Number of outliers in MonthlyRate: 0
Number of outliers in NumCompaniesWorked: 52
Number of outliers in Over18: 1
Number of outliers in OverTime: 2
Number of outliers in PercentSalaryHike: 0
Number of outliers in PerformanceRating: 226
Number of outliers in RelationshipSatisfaction: 0
Number of outliers in StandardHours: 0
Number of outliers in StockOptionLevel: 85
Number of outliers in TotalWorkingYears: 63
Number of outliers in TrainingTimesLastYear: 238
Number of outliers in WorkLifeBalance: 0
Number of outliers in YearsAtCompany: 104
Number of outliers in YearsInCurrentRole: 21
Number of outliers in YearsSinceLastPromotion: 107
Number of outliers in YearsWithCurrManager: 14
```

There is no need to handle above outliers as the ones which have outliers are logically genuine

Displaying Data Types of Columns :

In [7]:

```
print(data.dtypes)
```

```
Age                          int64
Attrition                   object
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
MonthlyRate                  int64
NumCompaniesWorked           int64
Over18                      object
OverTime                    object
PercentSalaryHike            int64
PerformanceRating            int64
RelationshipSatisfaction     int64
StandardHours                int64
StockOptionLevel             int64
TotalWorkingYears            int64
TrainingTimesLastYear        int64
WorkLifeBalance              int64
YearsAtCompany               int64
YearsInCurrentRole           int64
YearsSinceLastPromotion      int64
YearsWithCurrManager         int64
dtype: object
```

Encoding Categorical Columns :

In [8]:

```
clmns = ['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'OverTime', 'Edu
cation', 'EnvironmentSatisfaction', 'JobInvolvement', 'JobSatisfaction', 'PerformanceRating', 'RelationshipSatisf
action', 'WorkLifeBalance']
label_encoders = {}
for col in clmns:
    label_encoder = LabelEncoder()
    data[col] = label_encoder.fit_transform(data[col])
    label_encoders[col] = label_encoder
```

Splitting Features and Targets :

In [9]:

```
X_features = data.drop(columns=['Attrition'])
y_target = data['Attrition']
```

Scalling Numerical Values :

In [10]:

```
numerical_features = data.select_dtypes(include=np.number).columns
scaler = StandardScaler()
scaler.fit(data[numerical_features])
data[numerical_features] = scaler.transform(data[numerical_features])
```

Splitting Data Into Testing And Training sets :

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.2, random_state=42)
```

Model Building :

Training a Logistic Regression Model :

In [12]:

```
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_encoded)
X_test_scaled = scaler.transform(X_test_encoded)
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_scaled, y_train)
logistic_pred = logistic_model.predict(X_test_scaled)
```

Training a Decision Tree Model :

In [13]:

```
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train_encoded, y_train)
decision_tree_pred = decision_tree_model.predict(X_test_encoded)
```

Training a Random Forest Model :

In [14]:

```
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)
random_forest_model = RandomForestClassifier()
random_forest_model.fit(X_train_encoded, y_train)
random_forest_pred = random_forest_model.predict(X_test_encoded)
```

Calculating Performance Metrics:

Performance Metrics for Logistic Regression :

In [15]:

```
logistic_accuracy = accuracy_score(y_test, logistic_pred)
logistic_report = classification_report(y_test, logistic_pred)
logistic_confusion_matrix = confusion_matrix(y_test, logistic_pred)
print("Accuracy:\n", logistic_accuracy)
print("Report:\n", logistic_report)
```

```
Accuracy:
 0.891156462585034
Report:
               precision    recall  f1-score   support

          No       0.91      0.98      0.94       255
         Yes       0.68      0.33      0.45        39

    accuracy                           0.89       294
   macro avg       0.79      0.65      0.69       294
weighted avg       0.88      0.89      0.87       294
```

Performance Metrics for Decision Tree :

In [16]:

```
decision_tree_accuracy = accuracy_score(y_test, decision_tree_pred)
decision_tree_report = classification_report(y_test, decision_tree_pred)
decision_tree_confusion_matrix = confusion_matrix(y_test, decision_tree_pred)
print("Accuracy:\n", decision_tree_accuracy)
print("Report:\n", decision_tree_report)
```

```
Accuracy:
 0.7721088435374149
Report:
               precision    recall  f1-score   support

          No       0.87      0.86      0.87       255
         Yes       0.17      0.18      0.17        39

    accuracy                           0.77       294
   macro avg       0.52      0.52      0.52       294
weighted avg       0.78      0.77      0.78       294
```

Performance Metrics for Random Forest :

In [17]:

```
random_forest_accuracy = accuracy_score(y_test, random_forest_pred)
random_forest_report = classification_report(y_test, random_forest_pred)
random_forest_confusion_matrix = confusion_matrix(y_test, random_forest_pred)
print("Accuracy:\n", random_forest_accuracy)
print("Report:\n", random_forest_report)
```

```
Accuracy:
 0.8673469387755102
Report:
               precision    recall  f1-score   support

          No       0.87      0.99      0.93       255
         Yes       0.50      0.05      0.09        39

    accuracy                           0.87       294
   macro avg       0.69      0.52      0.51       294
weighted avg       0.82      0.87      0.82       294
```

Best Model: Logistic Regression

Thank You