KAGGLE CODE FOR ML:

df=pd.read_csv('/kaggle/input/ibm-hr-analytics-attrition-dataset/WA_Fn-UseC_-HR-Employee-Attrition.csv')

CHECKING THE ATTRIBUTES IN THE DATASET AND ITS TYPE

df.info()

class 'pandas.core.frame.DataFrame'>

RangeIndex: 1470 entries, 0 to 1469

Data columns (total 35 columns):

```
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
```

```
 23  PercentSalaryHike        1470 non-null   int64
 24  PerformanceRating        1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours            1470 non-null   int64
 27  StockOptionLevel         1470 non-null   int64
 28  TotalWorkingYears        1470 non-null   int64
 29  TrainingTimesLastYear    1470 non-null   int64
 30  WorkLifeBalance          1470 non-null   int64
 31  YearsAtCompany           1470 non-null   int64
 32  YearsInCurrentRole       1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager     1470 non-null   int64
dtypes: int64(26), object(9)
```

DROPPING STANDARD VALUES WHICH IS CONSTANT OR IRRELEVANT TO ANALYSIS

```
df = df.drop(['Over18','EmployeeCount','EmployeeNumber','StandardHours'],axis=1)
df = df.drop(['PerformanceRating','HourlyRate','MonthlyRate'],axis=1)
```

CONVERTING ATTRIBUTES TO NUMBERS ENCODING:

```
from sklearn.preprocessing import LabelEncoder


label_encoder = LabelEncoder()
df['Attrition'] = label_encoder.fit_transform(df['Attrition'])
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['OverTime'] = label_encoder.fit_transform(df['OverTime'])
df['MaritalStatus'] = label_encoder.fit_transform(df['MaritalStatus'])
```

CHECKING UNIQUE VALUES IN SOME ATTRIBUTES

```
def get_unique_values(df, column_name):
    unique_values = df[column_name].unique()
    print(f"Unique values in '{column_name}': {unique_values}")
    return unique_values


# Call the function for 'BusinessTravel'
unique_ms = get_unique_values(df, 'MaritalStatus')
unique_business_travel = get_unique_values(df, 'BusinessTravel')
```

unique_business_travel = get_unique_values(df, 'Department')

unique_business_travel = get_unique_values(df, 'EducationField')

unique_business_travel = get_unique_values(df, 'JobRole')

Unique values in 'MaritalStatus': [2 1 0]

Unique values in 'BusinessTravel': ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']

Unique values in 'Department': ['Sales' 'Research & Development' 'Human Resources']

Unique values in 'EducationField': ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'

'Human Resources']

Unique values in 'JobRole': ['Sales Executive' 'Research Scientist' 'Laboratory Technician'

'Manufacturing Director' 'Healthcare Representative' 'Manager'

'Sales Representative' 'Research Director' 'Human Resources']

CREATING COLUMNS BY SPLITTING THE COLUMNS

df = pd.get_dummies(df, columns=['BusinessTravel'], drop_first=True)

df= pd.get_dummies(df, columns=['Department'], drop_first=True)

df = pd.get_dummies(df, columns=['EducationField'], drop_first=True)

df = pd.get_dummies(df, columns=['JobRole'], drop_first=True)

CHECKING THAT ALL ARE NUMERIC OR BOOLEAN TYPE

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1470 entries, 0 to 1469

Data columns (total 41 columns):

```
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Age                          1470 non-null   int64
 1   Attrition                    1470 non-null   int64
 2   DailyRate                    1470 non-null   int64
 3   DistanceFromHome             1470 non-null   int64
 4   Education                    1470 non-null   int64
 5   EnvironmentSatisfaction      1470 non-null   int64
 6   Gender                       1470 non-null   int64
 7   JobInvolvement               1470 non-null   int64
 8   JobLevel                     1470 non-null   int64
 9   JobSatisfaction              1470 non-null   int64
```

```
 10  MaritalStatus                   1470 non-null   int64
 11  MonthlyIncome                   1470 non-null   int64
 12  NumCompaniesWorked              1470 non-null   int64
 13  OverTime                        1470 non-null   int64
 14  PercentSalaryHike               1470 non-null   int64
 15  RelationshipSatisfaction        1470 non-null   int64
 16  StockOptionLevel                1470 non-null   int64
 17  TotalWorkingYears               1470 non-null   int64
 18  TrainingTimesLastYear           1470 non-null   int64
 19  WorkLifeBalance                 1470 non-null   int64
 20  YearsAtCompany                  1470 non-null   int64
 21  YearsInCurrentRole              1470 non-null   int64
 22  YearsSinceLastPromotion         1470 non-null   int64
 23  YearsWithCurrManager            1470 non-null   int64
 24  BusinessTravel_Travel_Frequently   1470 non-null   bool
 25  BusinessTravel_Travel_Rarely      1470 non-null   bool
 26  Department_Research & Development  1470 non-null   bool
 27  Department_Sales                1470 non-null   bool
 28  EducationField_Life Sciences      1470 non-null   bool
 29  EducationField_Marketing         1470 non-null   bool
 30  EducationField_Medical           1470 non-null   bool
 31  EducationField_Other             1470 non-null   bool
 32  EducationField_Technical Degree    1470 non-null   bool
 33  JobRole_Human Resources          1470 non-null   bool
 34  JobRole_Laboratory Technician      1470 non-null   bool
 35  JobRole_Manager                 1470 non-null   bool
 36  JobRole_Manufacturing Director     1470 non-null   bool
 37  JobRole_Research Director         1470 non-null   bool
 38  JobRole_Research Scientist        1470 non-null   bool
 39  JobRole_Sales Executive          1470 non-null   bool
 40  JobRole_Sales Representative      1470 non-null   bool
dtypes: bool(17), int64(24)
memory usage: 300.2 KB
```

UNDERSTANDING LINEAR RELATIONSHIPS

corr_matrix = df.corr()

attrition_corr = corr_matrix['Attrition'].sort_values(ascending=False)

print(attrition_corr)

```
Attrition                           1.000000
OverTime                            0.246118
MaritalStatus                       0.162070
JobRole_Sales Representative        0.157234
BusinessTravel_Travel_Frequently    0.115143
JobRole_Laboratory Technician       0.098290
Department_Sales                    0.080855
DistanceFromHome                    0.077924
EducationField_Technical Degree     0.069355
EducationField_Marketing            0.055781
NumCompaniesWorked                  0.043494
JobRole_Human Resources             0.036215
Gender                              0.029453
JobRole_Sales Executive             0.019774
JobRole_Research Scientist         -0.000360
PercentSalaryHike                  -0.013478
EducationField_Other               -0.017898
Education                          -0.031373
EducationField_Life Sciences       -0.032703
YearsSinceLastPromotion            -0.033019
RelationshipSatisfaction           -0.045872
EducationField_Medical             -0.046999
BusinessTravel_Travel_Rarely       -0.049538
DailyRate                          -0.056652
TrainingTimesLastYear              -0.059478
WorkLifeBalance                    -0.063939
JobRole_Manufacturing Director     -0.082994
JobRole_Manager                    -0.083316
Department_Research & Development  -0.085293
```

JobRole_Research Director -0.088870

EnvironmentSatisfaction -0.103369

JobSatisfaction -0.103481

JobInvolvement -0.130016

YearsAtCompany -0.134392

StockOptionLevel -0.137145

YearsWithCurrManager -0.156199

Age -0.159205

MonthlyIncome -0.159840

YearsInCurrentRole -0.160545

JobLevel -0.169105

TotalWorkingYears -0.171063

Name: Attrition, dtype: float64

DROPPING UNECESSARY IRRELEVANT ATTRIBUTES FROM CORRELATION ANALYSIS

df = df.drop(['NumCompaniesWorked','Gender','PercentSalaryHike','Education'],axis=1)

FINAL FEATURES FOR ML TRAINING

add Code**add** Markdown

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1470 entries, 0 to 1469

Data columns (total 37 columns):

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      1470 non-null   int64
 1   Attrition                1470 non-null   int64
 2   DailyRate                1470 non-null   int64
 3   DistanceFromHome         1470 non-null   int64
 4   EnvironmentSatisfaction  1470 non-null   int64
 5   JobInvolvement           1470 non-null   int64
 6   JobLevel                 1470 non-null   int64
 7   JobSatisfaction          1470 non-null   int64
 8   MaritalStatus            1470 non-null   int64
 9   MonthlyIncome            1470 non-null   int64
```

```
 10  OverTime                         1470 non-null   int64
 11  RelationshipSatisfaction         1470 non-null   int64
 12  StockOptionLevel                 1470 non-null   int64
 13  TotalWorkingYears                1470 non-null   int64
 14  TrainingTimesLastYear            1470 non-null   int64
 15  WorkLifeBalance                  1470 non-null   int64
 16  YearsAtCompany                   1470 non-null   int64
 17  YearsInCurrentRole               1470 non-null   int64
 18  YearsSinceLastPromotion          1470 non-null   int64
 19  YearsWithCurrManager             1470 non-null   int64
 20  BusinessTravel_Travel_Frequently   1470 non-null   bool
 21  BusinessTravel_Travel_Rarely      1470 non-null   bool
 22  Department_Research & Development  1470 non-null   bool
 23  Department_Sales                 1470 non-null   bool
 24  EducationField_Life Sciences      1470 non-null   bool
 25  EducationField_Marketing         1470 non-null   bool
 26  EducationField_Medical           1470 non-null   bool
 27  EducationField_Other             1470 non-null   bool
 28  EducationField_Technical Degree    1470 non-null   bool
 29  JobRole_Human Resources          1470 non-null   bool
 30  JobRole_Laboratory Technician     1470 non-null   bool
 31  JobRole_Manager                  1470 non-null   bool
 32  JobRole_Manufacturing Director    1470 non-null   bool
 33  JobRole_Research Director        1470 non-null   bool
 34  JobRole_Research Scientist       1470 non-null   bool
 35  JobRole_Sales Executive          1470 non-null   bool
 36  JobRole_Sales Representative      1470 non-null   bool
dtypes: bool(17), int64(20)
```

LABELING THE FEATURE COLUMNS AND PREDICTION

```
X = df.drop('Attrition',axis=1)

y = df['Attrition'].astype(int)

X.info()

<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 36 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Age                                1470 non-null   int64
 1   DailyRate                          1470 non-null   int64
 2   DistanceFromHome                   1470 non-null   int64
 3   EnvironmentSatisfaction            1470 non-null   int64
 4   JobInvolvement                     1470 non-null   int64
 5   JobLevel                           1470 non-null   int64
 6   JobSatisfaction                    1470 non-null   int64
 7   MaritalStatus                      1470 non-null   int64
 8   MonthlyIncome                      1470 non-null   int64
 9   OverTime                           1470 non-null   int64
 10  RelationshipSatisfaction           1470 non-null   int64
 11  StockOptionLevel                   1470 non-null   int64
 12  TotalWorkingYears                  1470 non-null   int64
 13  TrainingTimesLastYear              1470 non-null   int64
 14  WorkLifeBalance                    1470 non-null   int64
 15  YearsAtCompany                     1470 non-null   int64
 16  YearsInCurrentRole                 1470 non-null   int64
 17  YearsSinceLastPromotion            1470 non-null   int64
 18  YearsWithCurrManager               1470 non-null   int64
 19  BusinessTravel_Travel_Frequently   1470 non-null   bool
 20  BusinessTravel_Travel_Rarely       1470 non-null   bool
 21  Department_Research & Development  1470 non-null   bool
 22  Department_Sales                   1470 non-null   bool
 23  EducationField_Life Sciences       1470 non-null   bool
 24  EducationField_Marketing           1470 non-null   bool
 25  EducationField_Medical             1470 non-null   bool
 26  EducationField_Other               1470 non-null   bool
 27  EducationField_Technical Degree    1470 non-null   bool
 28  JobRole_Human Resources            1470 non-null   bool
```

```
 29  JobRole_Laboratory Technician      1470 non-null   bool
 30  JobRole_Manager                    1470 non-null   bool
 31  JobRole_Manufacturing Director     1470 non-null   bool
 32  JobRole_Research Director          1470 non-null   bool
 33  JobRole_Research Scientist         1470 non-null   bool
 34  JobRole_Sales Executive            1470 non-null   bool
 35  JobRole_Sales Representative       1470 non-null   bool
dtypes: bool(17), int64(19)
memory usage: 242.7 KB
```

RANDOM FOREST IMPLEMENTATION

```
rom sklearn.ensemble import RandomForestClassifier


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()
classifier.fit(X=X_train,y=y_train)
test_pred = classifier.predict(X_test)
model = RandomForestClassifier(class_weight='balanced')
model.fit(X_train, y_train)
train_accuracy = model.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
test_accuracy = model.score(X_test, y_test)
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
Training Accuracy: 100.00%
Testing Accuracy: 85.03%
```

SUPPORT VECTOR MACHINE IMPLEMENTATION

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Initialize the SVM classifier

classifier = SVC(class_weight='balanced') # SVM with class weight handling

classifier.fit(X_train, y_train)

Predict on the test set

test_pred = classifier.predict(X_test)

Calculate and print the training accuracy

train_accuracy = classifier.score(X_train, y_train)

print(f"Training Accuracy: {train_accuracy * 100:.2f}%")

Calculate and print the testing accuracy

test_accuracy = classifier.score(X_test, y_test)

print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")

Training Accuracy: 66.57%

Testing Accuracy: 66.89%

DECISION TREES IMPLEMENTATION

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

Initialize the Decision Tree classifier

classifier = DecisionTreeClassifier(class_weight='balanced') # Decision Tree with class weight handling

classifier.fit(X_train, y_train)

Predict on the test set

test_pred = classifier.predict(X_test)

Calculate and print the training accuracy

train_accuracy = classifier.score(X_train, y_train)

```
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")


# Calculate and print the testing accuracy
test_accuracy = classifier.score(X_test, y_test)
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
```

Training Accuracy: 100.00%

Testing Accuracy: 79.59%

X GRADIENT BOOST

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)


# Initialize the XGBoost classifier
classifier = XGBClassifier(scale_pos_weight=(y_train.value_counts()[0] /
y_train.value_counts()[1]))  # Handle class imbalance
classifier.fit(X_train, y_train)


# Predict on the test set
test_pred = classifier.predict(X_test)


# Calculate and print the training accuracy
train_accuracy = classifier.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")


# Calculate and print the testing accuracy
test_accuracy = classifier.score(X_test, y_test)
print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
```

Training Accuracy: 100.00%

Testing Accuracy: 85.94%

XG BOOST WITH SMOTE THRESHOLD AND CLASSIFICATION REPORT:

```
import joblib
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE
import numpy as np

# Assuming X and y are your original features and target
# Replace these with your actual dataset
# X = ... (Your feature data)
# y = ... (Your target labels)

# Apply SMOTE to handle class imbalance
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_res, y_res = smote.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# Initialize XGBoost model with fixed random_state
xgb_model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=False,
    random_state=42
)

# Train the model
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred = xgb_model.predict(X_test)
```

```
# Predict probabilities for custom thresholding
y_proba = xgb_model.predict_proba(X_test)[:, 1]

# Custom threshold
threshold = 0.3 # Adjust this value as needed
y_pred_custom = (y_proba >= threshold).astype(int)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred_custom)
print(f"Accuracy with custom threshold: {accuracy * 100:.2f}%")

# Detailed classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_custom))
#print(X_res)

# Save the trained model
joblib.dump(xgb_model, 'eapsnew2.pkl')
print("Model saved successfully as 'eapsnew.pkl2'")
```

```
Accuracy with custom threshold: 92.51%
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       250
           1       0.93      0.91      0.92       244

    accuracy                           0.93       494
   macro avg       0.93      0.92      0.93       494
weighted avg       0.93      0.93      0.93       494

Model saved successfully as 'eapsnew.pkl2'
```

COMPARISON OF PERFORMANCE BEFORE AND AFTER SMOTE:

```
import matplotlib.pyplot as plt

import numpy as np


# Metrics to compare

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']


# Values before and after SMOTE + custom threshold

before_smote = [86.17, 87, 90, 89]  # Normal XGBoost

after_smote = [92.51, 93, 92, 93]  # After SMOTE + Custom Threshold


# Create a bar chart

x = np.arange(len(metrics))  # the label locations

width = 0.35  # the width of the bars


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


# Plot bars for before and after SMOTE

bars1 = ax.bar(x - width/2, before_smote, width, label='Before SMOTE', color='lightblue')

bars2 = ax.bar(x + width/2, after_smote, width, label='After SMOTE + Threshold',
color='lightgreen')


# Add some text for labels, title and custom x-axis tick labels, etc.

ax.set_xlabel('Metrics')

ax.set_ylabel('Scores')

ax.set_title('Comparison of XGBoost Performance Before and After SMOTE + Custom Threshold')

ax.set_xticks(x)

ax.set_xticklabels(metrics)

ax.legend()


# Display the chart

plt.show()

import matplotlib.pyplot as plt

import numpy as np
```

```
# Metrics to compare
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']

# Values before and after SMOTE + custom threshold
before_smote = [86.17, 87, 90, 89]  # Normal XGBoost
after_smote = [92.51, 93, 92, 93]  # After SMOTE + Custom Threshold

# Create a bar chart
x = np.arange(len(metrics))  # the label locations
width = 0.35  # the width of the bars

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

# Plot bars for before and after SMOTE
bars1 = ax.bar(x - width/2, before_smote, width, label='Before SMOTE', color='lightblue')
bars2 = ax.bar(x + width/2, after_smote, width, label='After SMOTE + Threshold',
color='lightgreen')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Comparison of XGBoost Performance Before and After SMOTE + Custom Threshold')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Display the chart
plt.show()
```

PERFORMANCE METRICS ACROSS MODELS:

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

```
# Metrics for each model
metrics = ['precision', 'recall', 'f1-score']

# Extracted values from the classification reports
results = {
    'Random Forest': [0.94, 0.92, 0.93],  # precision, recall, f1-score for class 0 and 1
    'LightGBM': [0.92, 0.91, 0.92],
    'XGBoost': [0.92, 0.94, 0.93]
}

# Create a DataFrame for better visualization
df_results = pd.DataFrame(results, index=metrics)

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df_results, annot=True, cmap='Blues', fmt='.2f', cbar=True)
plt.title('Performance Metrics Heatmap: Random Forest, LightGBM, and XGBoost')
plt.xlabel('Models')
plt.ylabel('Metrics')
plt.show()
```

UNSEEN DATA VALIDATION:

```
import numpy as np
import joblib  # Import joblib to load the model

# Load the trained model (replace 'eapsnew2.pkl' with the actual path to your model file)
xgb_model = joblib.load('eapsnew2.pkl')  # Ensure the correct path to your saved model

# Define the hypothetical input for the employee (36 features)
X_hypothetical = np.array([[
    30, 1200, 5, 3, 2, 3, 4, 1, 5000, 1, 3, 0, 8, 2, 3, 3, 2, 2, 1, 0,
    1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0
]])
```

```
# Now X_hypothetical has 36 features, which matches the model's expectation



# Make a prediction using the trained model
y_pred = xgb_model.predict(X_hypothetical)

# If you used custom thresholding, you can apply that to the probabilities
y_proba = xgb_model.predict_proba(X_hypothetical)[:, 1]
threshold = 0.3  # Example threshold
y_pred_custom = (y_proba >= threshold).astype(int)

# Print the results
print(f"Predicted Attrition (0 = Stay, 1 = Leave): {y_pred[0]}")
print(f"Predicted Attrition with Custom Threshold (0 = Stay, 1 = Leave): {y_pred_custom[0]}")
print(f"Predicted Probability of Attrition: {y_proba[0]}")

# Example output explanation:
if y_pred[0] == 1:
    print("The model predicts that the employee will leave the company.")
else:
    print("The model predicts that the employee will stay with the company.")
import numpy as np

# Define the hypothetical input for the employee (36 features with adjusted values)
X_hypothetical = np.array([[
    30, 3000, 5, 1, 1, 2, 1, 1, 3000, 1, 1, 0, 1, 5, 1, 1, 1, 1, 1, 1,
    1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0
]])

# Now X_hypothetical has 36 features, which matches the model's expectation
```

```
# Make a prediction using the trained model
y_pred = xgb_model.predict(X_hypothetical)

# If you used custom thresholding, you can apply that to the probabilities
y_proba = xgb_model.predict_proba(X_hypothetical)[:, 1]
threshold = 0.3  # Example threshold
y_pred_custom = (y_proba >= threshold).astype(int)

# Print the results
print(f"Predicted Attrition (0 = Stay, 1 = Leave): {y_pred[0]}")
print(f"Predicted Attrition with Custom Threshold (0 = Stay, 1 = Leave): {y_pred_custom[0]}")
print(f"Predicted Probability of Attrition: {y_proba[0]}")

# Example output explanation:
if y_pred[0] == 1:
    print("The model predicts that the employee will leave the company.")
else:
    print("The model predicts that the employee will stay with the company.")
```

STREAMLIT CODE:

```
!pip install streamlit
!pip install pyngrok
from google.colab import files
uploaded = files.upload()
import joblib

# Assuming the file is uploaded and named 'eapsnew2.pkl'
model_path = 'eapsnew2.pkl'

# Load the model
xgb_model = joblib.load(model_path)

!pip install google-genai
```

```
%%writefile eapbest.py
import streamlit as st
import pandas as pd
import numpy as np
import shap
import matplotlib.pyplot as plt
import seaborn as sns
import pickle


# ========== 1. Load Model ==========
def load_model(model_path="eapsnew2.pkl"):
    with open(model_path, "rb") as f:
        return pickle.load(f)

# ========== 2. Individual Prediction ==========




import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
from transformers import pipeline

def individual_prediction_mode(model):
    st.header("Individual Employee Attrition Prediction")
    st.sidebar.subheader("Enter Employee Information")

    # Collect user input
    input_data = collect_individual_inputs()
    predict_button = st.sidebar.button("Predict Attrition")
```

```
if predict_button:
    # Convert input to DataFrame
    df = pd.DataFrame([input_data])

    # Ensure numeric types
    numeric_cols = [
        'Age','DailyRate','DistanceFromHome','MonthlyIncome','TotalWorkingYears',
        'TrainingTimesLastYear','YearsAtCompany','YearsInCurrentRole',
        'YearsSinceLastPromotion','YearsWithCurrManager'
    ]
    for col in numeric_cols:
        df[col] = pd.to_numeric(df[col])

    # Align features with model
    model_features = model.get_booster().feature_names
    for col in model_features:
        if col not in df.columns:
            df[col] = 0
    df = df[model_features]

    # Prediction
    prediction = model.predict(df)[0]
    probability = model.predict_proba(df)[0][1]

    # Display prediction
    st.subheader("Prediction Output")
    st.metric("Prediction", "ATTRITION" if prediction == 1 else "RETENTION")
    st.progress(int(probability * 100))
    display_employee_profile(input_data)

    # Pie chart
    fig, ax = plt.subplots()
    ax.pie(
```

```
    [probability, 1 - probability],
    labels=["Attrition Risk", "Retention"],
    autopct="%1.1f%%",
    colors=["#f28b82", "#81c995"]
  )
  st.pyplot(fig)

  # ===== Personalized Factor-Based Recommendations =====
  st.subheader("Personalized Retention Suggestions")

  # Generate recommendations using Google Gemini API
  def generate_factor_based_recommendations(input_data):
    from google import genai
    client = genai.Client(api_key="AIzaSyDrdesGe7-FE7kAMEHpMu8ngWXb6YU378w")  #
Replace with your actual API key

    # Create a detailed text summary of all input features
    factors_text = "Employee Attributes:\n"
    for key, value in input_data.items():
      if isinstance(value, int) and value in [0, 1] and '_' in key:
        # Display categorical one-hot fields as Yes/No
        factors_text += f"- {key.replace('_', ' ')}: {'Yes' if value == 1 else 'No'}\n"
      else:
        factors_text += f"- {key.replace('_', ' ')}: {value}\n"

    prompt = f"""
You are an HR analyst. Analyze the employee attributes above.

The predictive model has calculated an attrition probability of {probability:.2f}
(where 0 means very low risk of leaving and 1 means very high risk of leaving).

Based on this probability and the employee attributes, provide the following:
```

```
1. Determine the attrition risk: High, Medium, or Low.

2. Extract the top 3 features contributing most to this prediction (e.g., Job Satisfaction, Work-Life Balance, Overtime).

3. If the employee is at risk of leaving (high or medium risk), list the key reasons why they might leave.

4. If the employee is likely to stay (low risk), explain why they are retaining and what factors should be addressed to prevent future attrition.

5. Provide 3 actionable retention strategies, numbered.


Format your response as:


Attrition Risk: <High/Medium/Low>

Top Contributing Factors:

1. ...

2. ...

3. ...

Reasons / Retention Insights:

1. ...

2. ...

3. ...

Recommendations:

1. ...

2. ...

3. ...

"""


    # Generate text with Google Gemini API

    response = client.models.generate_content(

        model="gemini-2.5-flash",

        contents=[factors_text + prompt]

    )


    output = response.text
```

```
        # Parse numbered lists
        lines = [line.strip() for line in output.splitlines() if line.strip()]
        result = {"AttritionRisk": "", "Reasons": [], "Recommendations": []}
        current_section = None
        for line in lines:
            if line.lower().startswith("attrition risk"):
                result["AttritionRisk"] = line.split(":")[1].strip()
                current_section = None
            elif line.lower().startswith("reasons"):
                current_section = "Reasons"
            elif line.lower().startswith("recommendations"):
                current_section = "Recommendations"
            elif current_section:
                line_clean = line.split(".", 1)[-1].strip()
                if line_clean:
                    result[current_section].append(line_clean)
        return result

    # Generate and display recommendations
    recommendations = generate_factor_based_recommendations(input_data)
    st.markdown(f"**Attrition Risk:** {recommendations['AttritionRisk']}")
    st.markdown("**Reasons:**")
    for reason in recommendations['Reasons']:
        st.markdown(f"- {reason}")
    st.markdown("**Retention Strategies:**")
    for rec in recommendations['Recommendations']:
        st.markdown(f"- {rec}")

def collect_individual_inputs():
    st.sidebar.title("Employee Details")

    # ---------- Numeric Inputs ----------
    input_data = {
```

```
        'Age': st.sidebar.number_input("Age", min_value=18, max_value=60, value=30),
        'DailyRate': st.sidebar.number_input("Daily Rate", min_value=100, max_value=1500,
value=800),
        'DistanceFromHome': st.sidebar.slider("Distance From Home (km)", 1, 30, 10),
        'EnvironmentSatisfaction': st.sidebar.selectbox("Environment Satisfaction", [1, 2, 3, 4]),
        'JobInvolvement': st.sidebar.selectbox("Job Involvement", [1, 2, 3, 4]),
        'JobLevel': st.sidebar.selectbox("Job Level", [1, 2, 3, 4, 5]),
        'JobSatisfaction': st.sidebar.selectbox("Job Satisfaction", [1, 2, 3, 4]),
        'RelationshipSatisfaction': st.sidebar.selectbox("Relationship Satisfaction", [1, 2, 3, 4]),
        'WorkLifeBalance': st.sidebar.selectbox("Work Life Balance", [1, 2, 3, 4]),
        'MaritalStatus': st.sidebar.selectbox(
            "Marital Status", [0, 1, 2],
            format_func=lambda x: {0: "Single", 1: "Married", 2: "Divorced"}[x]
        ),
        'MonthlyIncome': st.sidebar.number_input("Monthly Income", min_value=1000,
max_value=30000, value=8000),
        'StockOptionLevel': st.sidebar.selectbox("Stock Option Level", [0, 1, 2, 3]),
        'TotalWorkingYears': st.sidebar.number_input("Total Working Years", min_value=0,
max_value=40, value=5),
        'TrainingTimesLastYear': st.sidebar.number_input("Training Times Last Year", min_value=0,
max_value=10, value=2),
        'YearsAtCompany': st.sidebar.number_input("Years At Company", min_value=0,
max_value=40, value=3),
        'YearsInCurrentRole': st.sidebar.number_input("Years In Current Role", min_value=0,
max_value=40, value=2),
        'YearsSinceLastPromotion': st.sidebar.number_input("Years Since Last Promotion",
min_value=0, max_value=40, value=1),
        'YearsWithCurrManager': st.sidebar.number_input("Years With Current Manager",
min_value=0, max_value=40, value=2),
        'OverTime': 1 if st.sidebar.selectbox("OverTime", ["Yes", "No"]) == "Yes" else 0
    }


    # ---------- Categorical Inputs ----------
    business_travel = st.sidebar.selectbox(
        "Business Travel", ["Travel_Frequently", "Travel_Rarely", "Non-Travel"]
```

```
    )
    department = st.sidebar.selectbox(
        "Department", ["Sales", "Research & Development", "Human Resources"]
    )
    education_field = st.sidebar.selectbox(
        "Education Field", ["Life Sciences", "Medical", "Marketing", "Technical Degree", "Other"]
    )
    job_role = st.sidebar.selectbox(
        "Job Role", [
            "Human Resources", "Laboratory Technician", "Manager", "Manufacturing Director",
            "Research Director", "Research Scientist", "Sales Executive", "Sales Representative"
        ]
    )

    # ---------- One-Hot Encoding ----------
    categorical_columns = [
        "BusinessTravel_Travel_Frequently", "BusinessTravel_Travel_Rarely", "BusinessTravel_Non-Travel",
        "Department_Research & Development", "Department_Sales", "Department_Human Resources",
        "EducationField_Life Sciences", "EducationField_Medical", "EducationField_Marketing",
        "EducationField_Technical Degree", "EducationField_Other",
        "JobRole_Human Resources", "JobRole_Laboratory Technician", "JobRole_Manager",
        "JobRole_Manufacturing Director", "JobRole_Research Director", "JobRole_Research Scientist",
        "JobRole_Sales Executive", "JobRole_Sales Representative"
    ]

    for col in categorical_columns:
        if col.startswith("BusinessTravel_"):
            input_data[col] = 1 if col.split("_", 1)[1] == business_travel else 0
        elif col.startswith("Department_"):
            input_data[col] = 1 if col.split("_", 1)[1] == department else 0
        elif col.startswith("EducationField_"):
```

```
            input_data[col] = 1 if col.split("_", 1)[1] == education_field else 0

        elif col.startswith("JobRole_"):

            input_data[col] = 1 if col.split("_", 1)[1] == job_role else 0


    return input_data


def display_employee_profile(input_data):

    st.markdown("<h3 style='color: #000000;'>EMPLOYEE PROFILE</h3>",
unsafe_allow_html=True)


    # Basic Information

    with st.container():

        st.markdown("<h4 style='color: #2196F3;'>BASIC INFORMATION</h4>",
unsafe_allow_html=True)

        st.markdown(f"""

            <div style="border: 1px solid #e0e0e0; padding: 15px; border-radius: 10px; background-
color: #e3f2fd;">

                <p><strong>AGE</strong>: {input_data['Age']}</p>

                <p><strong>WORKED FOR</strong>: {input_data['TotalWorkingYears']}</p>

                <p><strong>DISTANCE FROM HOME</strong>: {input_data['DistanceFromHome']}</p>

            </div>

        """, unsafe_allow_html=True)


    # Satisfaction Metrics

    with st.container():

        st.markdown("<h4 style='color: #2196F3;'>SATISFACTION METRICS</h4>",
unsafe_allow_html=True)

        st.markdown(f"""

            <div style="border: 1px solid #e0e0e0; padding: 15px; border-radius: 10px; background-
color: #e3f2fd;">

                <p><strong>Job Satisfaction</strong>: {input_data['JobSatisfaction']}</p>

                <p><strong>Job Involvement</strong>: {input_data['JobInvolvement']}</p>

                <p><strong>Relationship Satisfaction</strong>:
{input_data['RelationshipSatisfaction']}</p>
```

```
            <p><strong>Environment Satisfaction</strong>:
{input_data['EnvironmentSatisfaction']}</p>

            <p><strong>Work-Life Balance</strong>: {input_data['WorkLifeBalance']}</p>

            <p><strong>Overtime</strong>: {'Yes' if input_data['OverTime'] == 1 else 'No'}</p>

          </div>

        """, unsafe_allow_html=True)


# ========== 3. Batch Prediction ==========

def batch_prediction_mode(model):

    st.header("Batch Attrition Prediction")

    uploaded_file = st.file_uploader("Upload Employee Data CSV", type=["csv"])


    if uploaded_file:

        df = pd.read_csv(uploaded_file)

        df["Attrition_Pred"] = model.predict(df)

        df["Attrition_Proba"] = model.predict_proba(df)[:, 1]


        st.subheader("Predictions Overview")

        st.dataframe(df.head())


        # Pie chart for attrition vs retention

        counts = df["Attrition_Pred"].value_counts()

        fig, ax = plt.subplots()

        ax.pie(counts, labels=["Retention", "Attrition"], autopct="%1.1f%%", colors=["#81c995",
"#f28b82"])

        st.pyplot(fig)


        st.download_button("Download Predictions CSV", df.to_csv(index=False),
"attrition_predictions.csv")


# ========== 4. Factor Insights ==========

def factor_insights_mode():

    st.header("Attrition Factor Insights")

    uploaded_file = st.file_uploader("Upload Employee Dataset", type=["csv"])
```

```
    if uploaded_file:
        df = pd.read_csv(uploaded_file)
        if "Attrition" not in df.columns:
            st.error("Dataset must contain an 'Attrition' column.")
            return

        feature_groups = [
            'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
            'Department_Research & Development', 'Department_Sales',
            'EducationField_Life Sciences', 'EducationField_Marketing',
            'EducationField_Medical', 'EducationField_Other',
            'EducationField_Technical Degree', 'JobRole_Human Resources',
            'JobRole_Laboratory Technician', 'JobRole_Manager',
            'JobRole_Manufacturing Director', 'JobRole_Research Director',
            'JobRole_Research Scientist', 'JobRole_Sales Executive',
            'JobRole_Sales Representative'
        ]

        for feature in feature_groups:
            if feature in df.columns:
                st.subheader(f"Attrition by {feature}")
                counts = df.groupby(feature)["Attrition"].mean()
                st.bar_chart(counts)

        st.subheader("Feature Correlation Heatmap")
        fig, ax = plt.subplots()
        sns.heatmap(df.corr(), cmap="coolwarm", annot=False, ax=ax)
        st.pyplot(fig)

# ========== 5. Main App ==========
def main():
```

```
    st.title("📊 ATTRI-SHIELD: PROACTIVE WORKFORCE MANAGEMENT DASHBOARD")
    model = load_model()

    mode = st.sidebar.radio(
        "Choose a mode:",
        ["Individual Prediction", "Batch Prediction", "Factor Insights"]
    )

    if mode == "Individual Prediction":
        individual_prediction_mode(model)
    elif mode == "Batch Prediction":
        batch_prediction_mode(model)
    elif mode == "Factor Insights":
        factor_insights_mode()

if __name__ == "__main__":
    main()


rom pyngrok import ngrok

# Connect to the correct port (Streamlit default is 8501)
public_url = ngrok.connect(8501)
print("Public URL:", public_url)

!streamlit run eapbest.py & npx ngrok http 8501
```