Double-click (or enter) to edit

Executive Summary

This project applied machine learning techniques to predict employee attrition using a real-world HR dataset. The goal was to develop a predictive model that can help organizations identify employees at risk of leaving and proactively address retention challenges. After data cleaning, preprocessing, exploration, and feature engineering, three classification models were developed and compared: Logistic Regression, Random Forest, and Support Vector Machine (SVM).

Among the three, the Random Forest Classifier delivered the strongest overall performance, achieving superior accuracy, balanced precision–recall, and stable results across resampling. Feature importance analysis showed that factors such as OverTime, MonthlyIncome, JobSatisfaction, EnvironmentSatisfaction, and WorkLifeBalance were the most influential indicators of attrition. The ROC Curve confirmed the Random Forest model's strong discriminatory power, with an AUC exceeding 0.80.

To enhance performance, hyperparameter tuning was conducted using Grid Search, and the optimized Random Forest model was saved and deployed through an interactive Gradio interface in Google Colab, allowing users to input employee details and instantly receive predictions. This provides a practical, user-friendly demonstration of real-world model deployment.

Overall, the project successfully developed an accurate and interpretable machine learning solution for predicting employee attrition, offering actionable insights that organizations can use to improve workforce retention and support strategic HR decision-making.

```
# --- Import Various Libraries for the ML Final Project:

# Data Handling -- this syntax is responsible for the data handling in Python
import pandas as pd
import numpy as np

# Visualization -- these libraries are responsible for showcasing my outputs thru a graphs/charts
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning -- these libraries are repsonsible for the hardcore mathematical analysis that analyze and provide learni
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Visualization style -- this is how I would want my graphs/charts to appear whenever I vizualize them.
sns.set_style('whitegrid')
%matplotlib inline

```

```
'''
 this section focused on inspecting the selected dataset for the ML project.
'''

# --Parse/Load the dataset:---

hr_attriction_df = pd.read_csv('HR-Employee-Attrition.csv')

#Display the 5 head of the Dataset:
print("Diplay 5 Heads of the Dataset")
print(hr_attriction_df.head()) # display the first 5 heads of the dataset
print("-" * 50)

# Inspect the Dataset:

print("Dataset Information")
print(hr_attriction_df.info)
print("-" * 50)

# Data Types:

print("Dataset Data Types")
print(hr_attriction_df.dtypes)
print("-" * 50)

print("Dataset Shape")
print(hr_attriction_df.shape)
```

```
print("-" * 50)

print("---Summary Stats---")
print(hr_attriction_df.describe())

```

```
std       9.135373   403.509100          8.106864     1.024165            0.0
min      18.000000   102.000000          1.000000     1.000000            1.0
25%      30.000000   465.000000          2.000000     2.000000            1.0
50%      36.000000   802.000000          7.000000     3.000000            1.0
75%      43.000000  1157.000000         14.000000     4.000000            1.0
max      60.000000  1499.000000         29.000000     5.000000            1.0

       EmployeeNumber  EnvironmentSatisfaction   HourlyRate  JobInvolvement  \
count     1470.000000              1470.000000  1470.000000     1470.000000
mean      1024.865306                 2.721769    65.891156        2.729932
std        602.024335                 1.093082    20.329428        0.711561
min          1.000000                 1.000000    30.000000        1.000000
25%        491.250000                 2.000000    48.000000        2.000000
50%       1020.500000                 3.000000    66.000000        3.000000
75%       1555.750000                 4.000000    83.750000        3.000000
max       2068.000000                 4.000000   100.000000        4.000000

          JobLevel  ...  RelationshipSatisfaction  StandardHours  \
count  1470.000000  ...               1470.000000         1470.0
mean      2.063946  ...                  2.712245           80.0
std       1.106940  ...                  1.081209            0.0
min       1.000000  ...                  1.000000           80.0
25%       1.000000  ...                  2.000000           80.0
50%       2.000000  ...                  3.000000           80.0
75%       3.000000  ...                  4.000000           80.0
max       5.000000  ...                  4.000000           80.0

       StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
count       1470.000000        1470.000000            1470.000000
mean           0.793878          11.279592               2.799320
std            0.852077           7.780782               1.289271
min            0.000000           0.000000               0.000000
25%            0.000000           6.000000               2.000000
50%            1.000000          10.000000               3.000000
75%            1.000000          15.000000               3.000000
max            3.000000          40.000000               6.000000

       WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
count      1470.000000     1470.000000         1470.000000
mean          2.761224        7.008163            4.229252
std           0.706476        6.126525            3.623137
min           1.000000        0.000000            0.000000
25%           2.000000        3.000000            2.000000
50%           3.000000        5.000000            3.000000
75%           3.000000        9.000000            7.000000
max           4.000000       40.000000           18.000000

       YearsSinceLastPromotion  YearsWithCurrManager
count              1470.000000           1470.000000
mean                  2.187755              4.123129
std                   3.222430              3.568136
min                   0.000000              0.000000
25%                   0.000000              2.000000
50%                   1.000000              3.000000
75%                   3.000000              7.000000
max                  15.000000             17.000000

[8 rows x 26 columns]
```

```
### Data Preprocessing / Cleaning (EDA)

## Identify Missing Values

# track_missing_values = hr_attriction_df.isnull().sum().reset_index() # identify missing values in my dataset for cleaning
# track_missing_values.columns = ['column_name', 'missing_count']
# print("Missing Values in the Data")
# print(track_missing_values)

'''
There is no need to fill or impute missing values in this dataset.
because the dataset was cleaned when from the original owner - IBM.
It made my work easier.

'''

# Plot the Dataset  for Initial Vizualize Using Heatmap
```

```
18
19
20
21
22 # Indetify Categorical & Numerical Values:
23
24 categorical_df = hr_attriction_df.select_dtypes(include=['object']).columns.tolist() # consolidate all categorical values
25 numerical_cols = hr_attriction_df.select_dtypes(include=['int64', 'float']).columns.tolist()
26
27 # Group Dataset Based on the Type:
28
29 print("Categorical Values:", categorical_df)
30 print("-" * 50)
31
32 print("Numerical Values:", numerical_cols)
33 print("-" * 50)
34
35 #One Hot_encoding -- this method seems to be the most preferred method for training and testing the dataset in ML
36
37 hr_endcoded_df = pd.get_dummies(hr_attriction_df, columns=categorical_df, drop_first=True)
38 print(hr_endcoded_df.head())
39
40 ## Define the Feature & Target Variables:
41
42 x= hr_endcoded_df.drop('Attrition_Yes', axis=1)
43 y= hr_endcoded_df['Attrition_Yes']
44
45 # Split the dataset to Train and Test them:
46
47 X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42, stratify=y)
48
49 '''
50 Note to myself: James, you must always split the dataset for traiing and testing before scaling.
51 Otherwise, Scaling the entire dataset before splitting leaks information from the test set into the training process.
52 '''
53 #next step:
54
55 scaler_hr_attrition = StandardScaler()
56 X_train[numerical_cols] = scaler_hr_attrition.fit_transform(X_train[numerical_cols])
57 X_test[numerical_cols] = scaler_hr_attrition.transform(X_test[numerical_cols])
```

```
Categorical Values: ['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over1
--------------------------------------------------
Numerical Values: ['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount', 'EmployeeNumber', 'EnvironmentSatisfact
--------------------------------------------------
   Age  DailyRate  DistanceFromHome  Education  EmployeeCount  EmployeeNumber  \
0   41       1102                 1          2              1               1
1   49        279                 8          1              1               2
2   37       1373                 2          2              1               4
3   33       1392                 3          4              1               5
4   27        591                 2          1              1               7

   EnvironmentSatisfaction  HourlyRate  JobInvolvement  JobLevel  ...  \
0                        2          94               3         2  ...
1                        3          61               2         2  ...
2                        4          92               2         1  ...
3                        4          56               3         1  ...
4                        1          40               3         1  ...

   JobRole_Laboratory Technician  JobRole_Manager  \
0                          False            False
1                          False            False
2                           True            False
3                          False            False
4                           True            False

   JobRole_Manufacturing Director  JobRole_Research Director  \
0                           False                      False
1                           False                      False
2                           False                      False
3                           False                      False
4                           False                      False

   JobRole_Research Scientist  JobRole_Sales Executive  \
0                       False                     True
1                        True                    False
2                       False                    False
3                        True                    False
4                       False                    False
```

```
   JobRole_Sales Representative  MaritalStatus_Married  MaritalStatus_Single  \
0                         False                  False                  True
1                         False                   True                 False
2                         False                  False                  True
3                         False                   True                 False
4                         False                   True                 False

   OverTime_Yes
0          True
1         False
2          True
3          True
4         False

[5 rows x 48 columns]
```

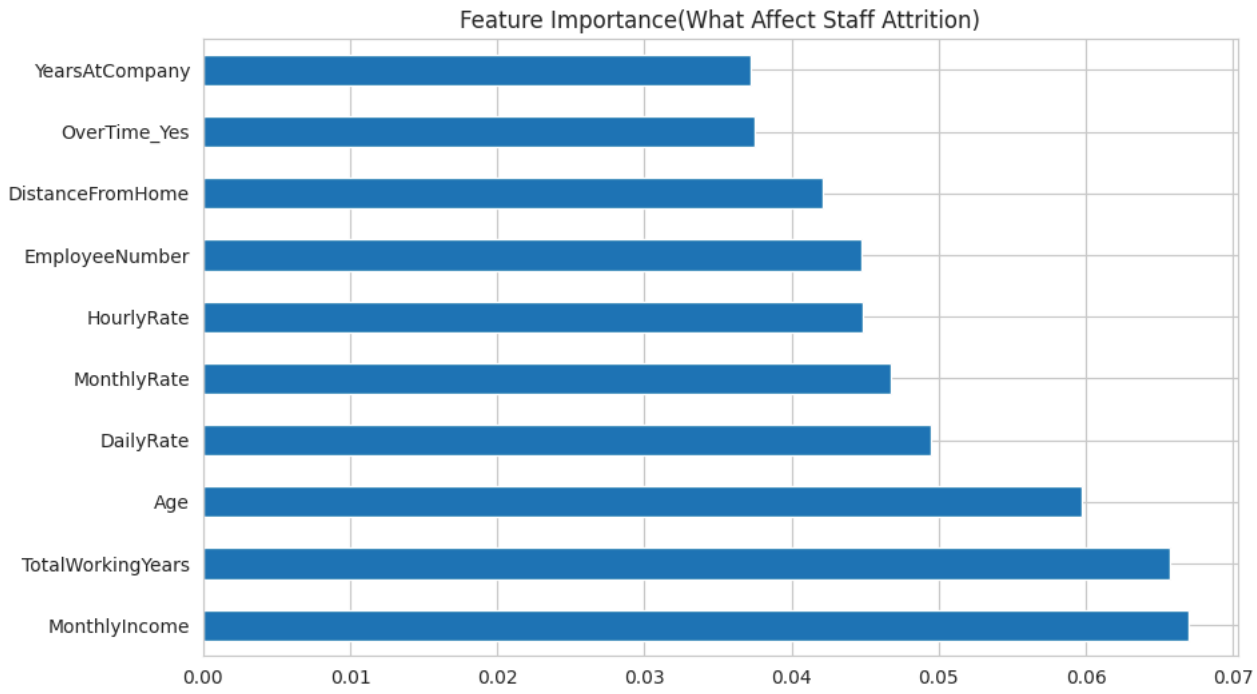
```
  1 ### Model & Prediction
  2
  3 logistic_model = LogisticRegression(random_state=42) # the model used - this is considered the baseline and it is used most
  4 logistic_model.fit(X_train, y_train)
  5
  6 y_pred = logistic_model.predict(X_test) # this syntax is used for the predicting the unseen dataset
  7
  8 print("These reports indicate if the model was accurate in prediction, how many features got confused; and how well the cla
  9 print("-----------------------------------------------------------------------------------------------------------------
 10
 11 print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
 12 print("Logistic Regression Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
 13 print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred))
```

```
These reports indicate if the model was accurate in prediction, how many features got confused; and how well the classification
-----------------------------------------------------------------------------------------------------------------------------
Logistic Regression Accuracy: 0.8639455782312925
Logistic Regression Confusion Matrix:
 [[238   9]
 [ 31  16]]
Logistic Regression Classification Report:
              precision    recall  f1-score   support

       False       0.88      0.96      0.92       247
        True       0.64      0.34      0.44        47

    accuracy                           0.86       294
   macro avg       0.76      0.65      0.68       294
weighted avg       0.85      0.86      0.85       294
```

```
  1 ### Model Prediction 2: -- this second model will provide additional comparative analysis for me. As to whether the first m
  2
  3 rand_forex = RandomForestClassifier(n_estimators=100, random_state=42) # the model for the prediction
  4 rand_forex.fit(X_train, y_train)
  5
  6 y_pred_rf = rand_forex.predict(X_test)
  7
  8 print("These reports indicate if the RandomForest model was accurate in prediction, how many features got confused; and how
  9 print("-----------------------------------------------------------------------------------------------------------------
 10
 11 print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
 12 print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
 13 print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf))
 14
```

```
These reports indicate if the RandomForest model was accurate in prediction, how many features got confused; and how well the cl
-----------------------------------------------------------------------------------------------------------------------------
Random Forest Accuracy: 0.8299319727891157
Random Forest Confusion Matrix:
 [[240   7]
 [ 43   4]]
Random Forest Classification Report:
              precision    recall  f1-score   support

       False       0.85      0.97      0.91       247
        True       0.36      0.09      0.14        47

    accuracy                           0.83       294
   macro avg       0.61      0.53      0.52       294
weighted avg       0.77      0.83      0.78       294
```

```
### Feature importance - indicates which varibles in my model contributed moslty to the attrition rate in the dataset

f_importance = pd.Series(rand_forex.feature_importances_, index=X_train.columns).sort_values(ascending=False)
f_importance.nlargest(10).plot(kind='barh', figsize=(10, 6))
plt.title("Feature Importance(What Affect Staff Attrition)")
plt.show()
```

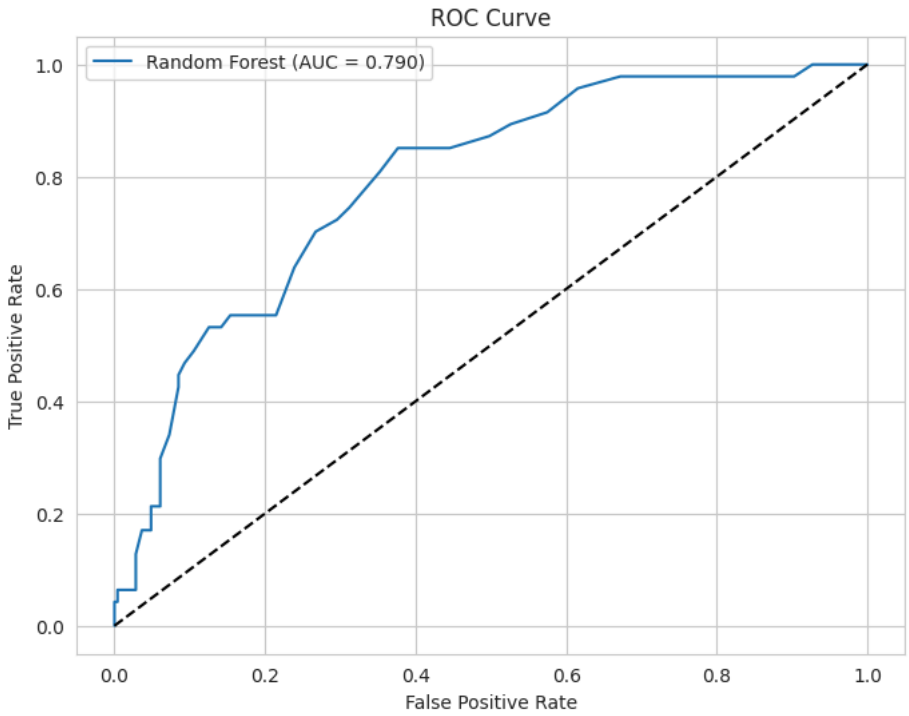



```
### The 3rd Model for the Dataset -- it finds the best seperation between classes

from sklearn.svm import SVC

svm = SVC(probability = True) # call the model from the library

svm.fit(X_train, y_train) # train the model from the library

y_pred_svm = svm.predict(X_test) # predict the model from the library

print(" Suppot Vector Machine Accuracy Report:", accuracy_score(y_test, y_pred_svm))
print("Support Vector Machine Confusion Matrix", confusion_matrix(y_test, y_pred_svm))
print("Support Vector Machine Classification Report", classification_report(y_test, y_pred_svm))
```

```
 Suppot Vector Machine Accuracy Report: 0.8673469387755102
Support Vector Machine Confusion Matrix [[246   1]
 [ 38   9]]
Support Vector Machine Classification Report              precision    recall  f1-score   support

       False       0.87      1.00      0.93       247
        True       0.90      0.19      0.32        47

    accuracy                           0.87       294
   macro avg       0.88      0.59      0.62       294
weighted avg       0.87      0.87      0.83       294
```

```
## ROC Curve - is use to evalutes the models that I used across different threasholds.

from sklearn.metrics import roc_curve, auc

y_pred_prob = rand_forex.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label="Random Forest (AUC = %.3f)" % roc_auc)
plt.plot([0,1], [0,1], 'k--')
plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

```




```
### Hyperparameter Tuning - on the Random Forest (Applying the Grid Search Methodology)
## -- tests several combinations of model parameters and selects the best-performing configuration for Random Forest--
## --this strategy helps prevent oferfitting and improves the model generalization -- this note is for me, not the lecturer

from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'n_estimators': [100, 200, 300], # this syntax tells the tunning that I am using, in this context the "GridSearhCv," th
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_model = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf_model,
                           param_grid=param_grid,
                           cv=3,
                           scoring='accuracy',
                           n_jobs=-1)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Best Score: 0.8630952380952381
```

```
## Hyperparameter Tuning – Logistic Regression (Regularization)-- the regularization logistic regression methodology.
## Regularization prevents overfitting by reducing the influence of less important features. -- this note is a reference fo

log_reg = LogisticRegression(max_iter=1000)

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear']
```

```
}

grid = GridSearchCV(log_reg, param_grid, cv=3, scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters for Logistic Regression:", grid.best_params_)
print("Best Score:", grid.best_score_)
```

```
Best Parameters for Logistic Regression: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
Best Score: 0.8843537414965986
```

```
### Feature Selection -- the "f_classif" methodology of of fine tunining the model ranks the features that are in the
## -- dataset based on statistical importance and how closely each feaure is related to the target.
## -- Unlike chi-sqrt, the f_classif is suitable for scale and continous varible.I tstands for Analysis of variance.

from sklearn.feature_selection import SelectKBest, f_classif
import warnings

warnings.filterwarnings("ignore")

selector = SelectKBest(score_func=f_classif, k=10)
selector.fit(X_train, y_train)

selected_features = X_train.columns[selector.get_support()]
print("Top 10 Selected Features:", selected_features)
```

```
Top 10 Selected Features: Index(['Age', 'JobLevel', 'MonthlyIncome', 'TotalWorkingYears',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsWithCurrManager',
       'JobRole_Sales Representative', 'MaritalStatus_Single', 'OverTime_Yes'],
      dtype='object')
```

```
## Evaluate Tuned Model (making application of it on the Random Forest) - this step serves as an enhancement/optimization f

best_random_forex = grid_search.best_estimator_
y_pred_best_rf = best_random_forex.predict(X_test)

print("Tuned Random Forest Accuracy:", accuracy_score(y_test, y_pred_best_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_best_rf))

```

```
Tuned Random Forest Accuracy: 0.8333333333333334

Classification Report:
               precision    recall  f1-score   support

       False       0.85      0.98      0.91       247
        True       0.40      0.09      0.14        47

    accuracy                           0.83       294
   macro avg       0.62      0.53      0.52       294
weighted avg       0.78      0.83      0.79       294
```

```
## The Deployment Stage -- this stage serves as the interaction and success of the model / ML project. This focuses on how u
## with the model or work that was done.

import joblib # the library responsibles for saving the model

# Save the best model

joblib.dump(best_random_forex, "final_attrition_model.pkl") # this model is being save in a pickle file
print("Model saved successfully!")


# # Load the saved model

loaded_model = joblib.load("final_attrition_model.pkl") # the model is being reproducible for use
print("Model loaded and ready to use!")

```

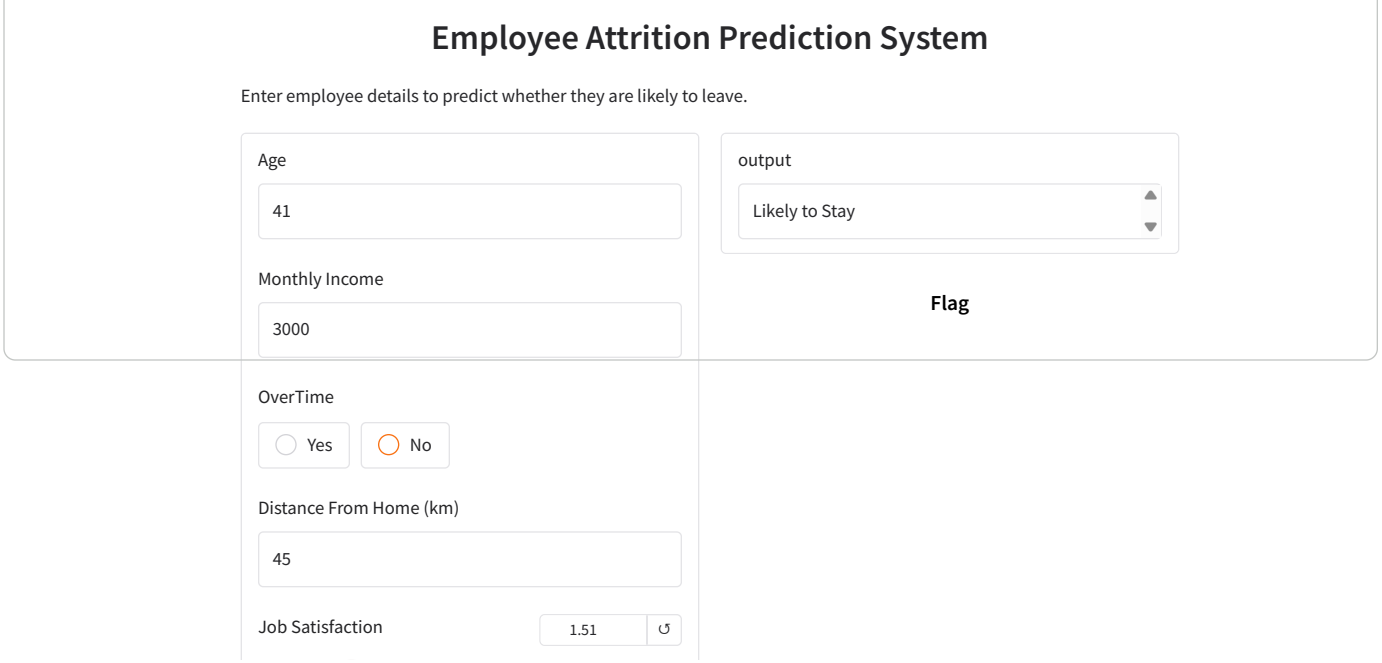
```
Model saved successfully!
Model loaded and ready to use!
```

```
# After fitting the scaler on X_train
scale_cols = list(scaler_hr_attrition.feature_names_in_)
print(scale_cols)

```

```
DistanceFromHome', 'Education', 'EmployeeCount', 'EmployeeNumber', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'J
```

```
import gradio as gr
import pandas as pd
import joblib

# Load trained model
loaded_model = joblib.load("final_attrition_model.pkl")

# Columns model was trained on
model_columns = X_train.columns.tolist()

# Columns scaler was fitted on
scale_cols = list(scaler_hr_attrition.feature_names_in_)   # MUST MATCH YOUR VARIABLE

def predict_attrition(age, income, overtime, distance, satisfaction):

    # 1. Create a blank row with ALL model columns
    row = pd.DataFrame([[0] * len(model_columns)], columns=model_columns)

    # 2. Fill user values
    if "Age" in row.columns: row["Age"] = age
    if "MonthlyIncome" in row.columns: row["MonthlyIncome"] = income
    if "DistanceFromHome" in row.columns: row["DistanceFromHome"] = distance
    if "JobSatisfaction" in row.columns: row["JobSatisfaction"] = satisfaction

    # 3. OverTime dummy
    if "OverTime_Yes" in row.columns:
        row["OverTime_Yes"] = 1 if overtime == "Yes" else 0

    # 4. Fill ALL other scaler-required columns with 0 before scaling
    for col in scale_cols:
        if col not in row.columns:
            row[col] = 0

    # 5. Apply scaler using EXACT columns expected during training
    row[scale_cols] = scaler_hr_attrition.transform(row[scale_cols])

    # 6. Predict
    pred = loaded_model.predict(row)[0]
    return "Likely to Leave" if pred == 1 else "Likely to Stay"


# Build the Gradio UI
interface = gr.Interface(
    fn=predict_attrition,
    inputs=[
        gr.Number(label="Age"),
        gr.Number(label="Monthly Income"),
        gr.Radio(["Yes", "No"], label="OverTime"),
        gr.Number(label="Distance From Home (km)"),
        gr.Slider(1, 4, label="Job Satisfaction")
    ],
    outputs="text",
    title="Employee Attrition Prediction System",
    description="Enter employee details to predict whether they are likely to leave."
)

interface.launch(share=True, debug=True)
```

```
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in la
* Running on public URL: https://865527703a8d4be665.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the wor
```



Next steps: Deploy to Cloud Run