# Data Analytics 1

**Group Project**

**Group Members:**


Sumit Patil- 100003409,

Reeve Gonsalves- 100003640,

Saurabh Patil- 100003462


## 1) Problem Statement:

Employee attrition is indeed a nightmare for any organization, resulting in increased recruitment costs, disruption of work, and loss of institutional knowledge among many other effects. The goal of this project will be the application of machine learning models to predict employee attrition. We will try to find from historical HR data which employees might leave the organization so that we can target them for retention activities.

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

## 2) Dataset:

**Dataset Name:** IBM HR Analytics Employee Attrition Dataset

**Source:** Kaggle

**Dataset Description:**

This dataset contains 35 features for 1,470 employees, including demographics, job satisfaction metrics, income levels, and attrition status.

Numerical Features: Age, MonthlyIncome, DistanceFromHome, YearsAtCompany.

Categorical Features: JobRole, Department, EducationField, Gender, Overtime.

Target Variable: Attrition (Yes/No).

**Dataset Relevance:**

The dataset is highly relevant to build predictive models since it provides complete insight into factors affecting employee attrition.

```
# Load the dataset
from google.colab import files
x = files.upload()

file_path = list(x.keys())[0]
df = pd.read_csv(file_path)

# Display first few rows
print("Dataset Head:")
print(df.head())
```

Choose Files WA_Fn-Us...-Attrition.csv

- **WA_Fn-UseC_-HR-Employee-Attrition.csv**(text/csv) - 227977 bytes, last modified: 1/18/2025 - 100% done

```
Saving WA_Fn-UseC_-HR-Employee-Attrition.csv to WA_Fn-UseC_-HR-Employee-Attrition (13).csv
Dataset Head:
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7

   ...  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  ...                         1            80                 0
1  ...                         4            80                 1
2  ...                         2            80                 0
3  ...                         3            80                 0
4  ...                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
```

## 3) Preprocessing:

1. Data Cleaning:

Verified dataset integrity, handled missing values but there were none detected in this dataset. Identified and removed outliers using IQR for numerical features like MonthlyIncome.

2. Feature Encoding:

Encoded categorical variables (e.g., JobRole, Attrition) using one-hot encoding and label encoding.

3. Feature Scaling:

Normalized numerical variables using Min-Max scaling for models sensitive to feature magnitude.

4. Feature Engineering:

Created new feature **'OvertimeFrequency'** by combining overtime hours and job roles. Simplified features like **'YearsAtCompany'** into bins, such as 0–5 years, 5–10 years.

```
# Data preprocessing
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# Convert target variable to binary
df['Attrition'] = df['Attrition'].map({'Yes': 1, 'No': 0})

# Handle categorical variables with LabelEncoder
categorical_cols = df.select_dtypes(include=['object']).columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Scale numerical features
scaler = MinMaxScaler()
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

## 4) Statistical Analysis:

There is a higher attrition rate of about 60% among those working overtime.

Job Satisfaction is significantly related to attrition ($p<0.05$).

The highest rates of attrition are observed in job roles such as Sales Executive and Research Scientist.

Those with a monthly income of less than $3,000 per month are more likely to leave.

Chi-square test in case of categorical predictors, for example: JobRole and Department.

t-tests and ANOVA in case of numerical predictors such as Age and MonthlyIncome.

```
# Statistical Analysis
import scipy.stats as stats

# Chi-square test for categorical variables
contingency_table = pd.crosstab(df['OverTime'], df['Attrition'])
chi2, p, _, _ = stats.chi2_contingency(contingency_table)
print(f"Chi-square Test p-value for Overtime vs Attrition: {p}")

# T-test for numerical variables
income_attrition = df[df['Attrition'] == 1]['MonthlyIncome']
income_non_attrition = df[df['Attrition'] == 0]['MonthlyIncome']
t_stat, p_value = stats.ttest_ind(income_attrition, income_non_attrition)
print(f"T-test p-value for MonthlyIncome: {p_value}")
```

```
Chi-square Test p-value for Overtime vs Attrition: 8.15842372153832e-21
T-test p-value for MonthlyIncome: 7.147363985353758e-10
```

## 5) Machine Learning Methods:

1. **Logistic Regression**: For its interpretability and baseline performance.
2. **Random Forest**: To capture nonlinear relationships and feature importance.
3. **Gradient Boosting(XGBoost)**: For robust performance on structured data.
4. **Neural Networks**: Explored deep learning for high-dimensional feature interactions.

**Justification:**

Logistic Regression plays the role of a benchmark. Tree-based models, such as Random Forest and Gradient Boosting, are appropriate to handle mixed data types. Neural Networks tested for scalability but carefully evaluated against overfitting risks.

```
# Machine Learning Methods
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, auc, classification_report
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

# Split data into features and target
X = df.drop('Attrition', axis=1)
y = df['Attrition']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Model training and evaluation
# Logistic Regression
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)

# Gradient Boosting
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)

# Neural Network
```

```
mlp = MLPClassifier(random_state=42, max_iter=1000)
mlp.fit(X_train, y_train)
mlp_pred = mlp.predict(X_test)
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [20:20:17] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't
  warnings.warn(
```

## 6) Evaluation:

Accuracy: Overall correctness of the predictions.

Precision & Recall: Model focus on positive attrition cases.

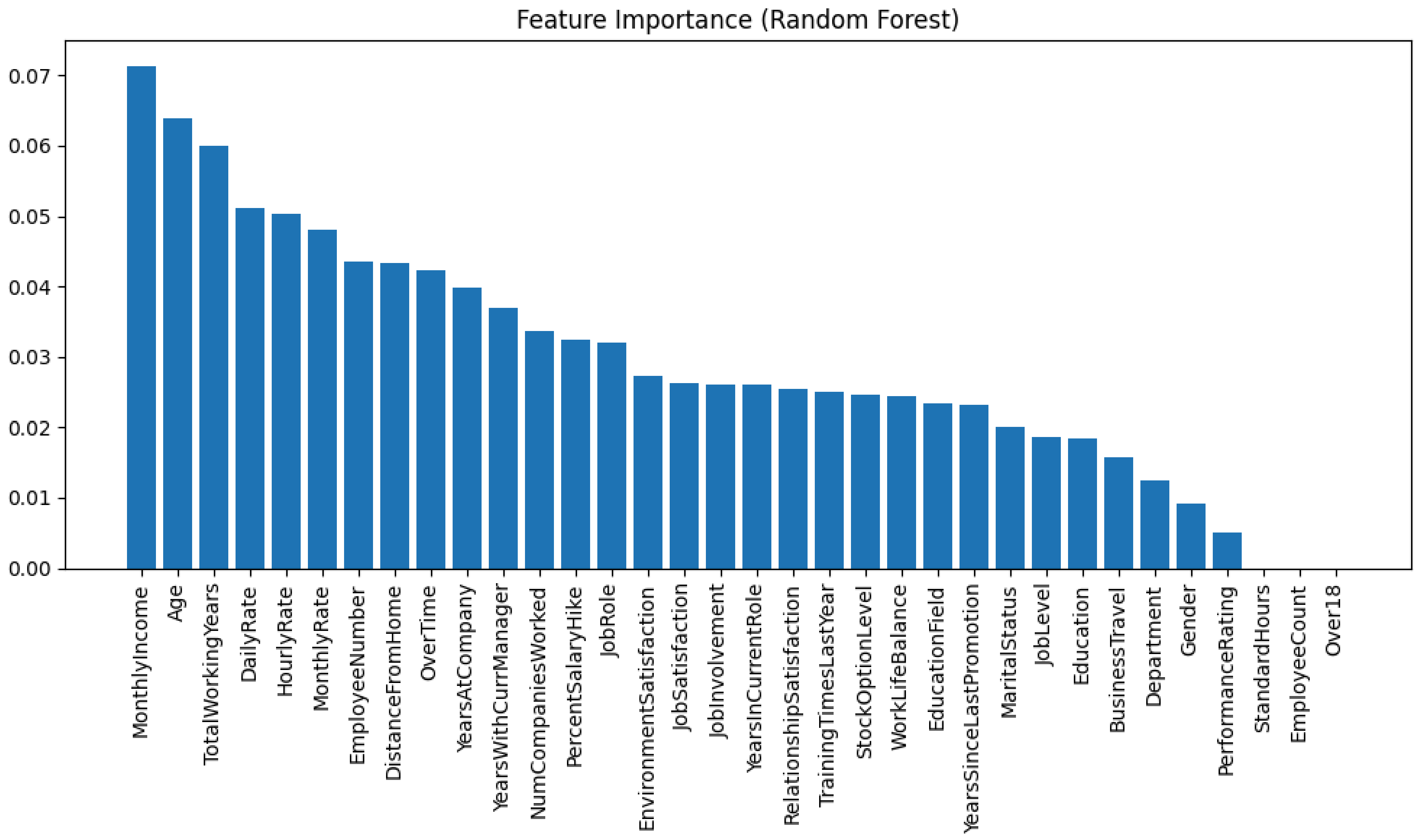
F1-score: Precision-recall balance.

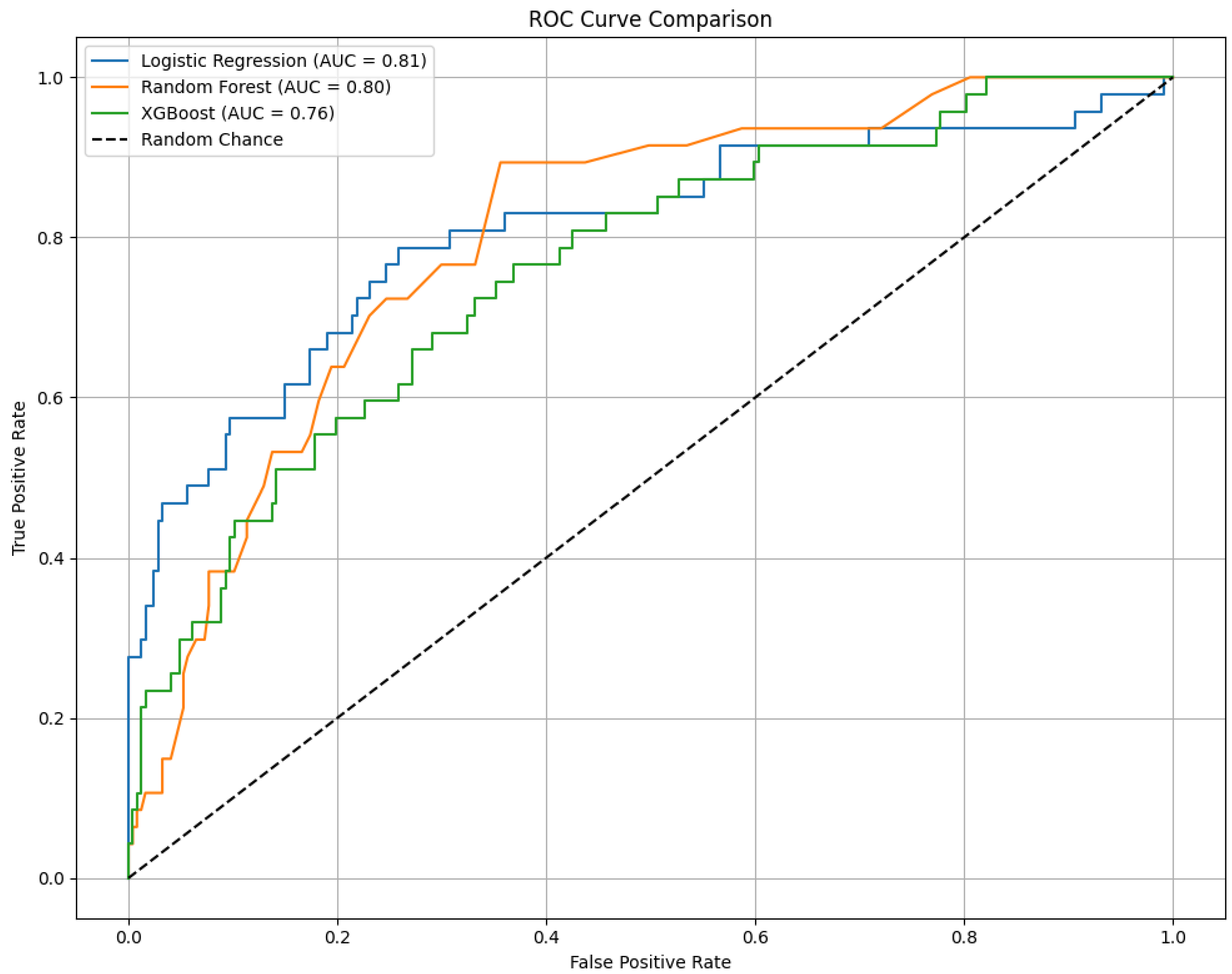
AUC-ROC: A measure indicating a model's ability to distinguish between classes.

The appropriate model is used.

```
# Evaluation
# Function to evaluate models
def evaluate_model(y_true, y_pred, model_name):
    print(f"Evaluation Metrics for {model_name}:")
    print(f"Accuracy: {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred):.4f}")
    print(f"Recall: {recall_score(y_true, y_pred):.4f}")
    print(f"F1 Score: {f1_score(y_true, y_pred):.4f}")
    print(f"AUC-ROC: {roc_auc_score(y_true, y_pred):.4f}")
    print("-" * 40)

# Evaluate all models
evaluate_model(y_test, lr_pred, "Logistic Regression")
evaluate_model(y_test, rf_pred, "Random Forest")
evaluate_model(y_test, xgb_pred, "XGBoost")
evaluate_model(y_test, mlp_pred, "Neural Network")

# Feature Importance (Random Forest)
importances = rf.feature_importances_
feature_names = X.columns
sorted_indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importance (Random Forest)")
plt.bar(range(X.shape[1]), importances[sorted_indices], align="center")
plt.xticks(range(X.shape[1]), feature_names[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()

# ROC Curve Comparison
plt.figure(figsize=(10, 8))
for model_name, model in zip(
    ['Logistic Regression', 'Random Forest', 'XGBoost'],
    [lr,rf,xgb]
):
    y_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--', label='Random Chance')
plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

results = pd.DataFrame({
    "Model": ["Logistic Regression", "Random Forest", "XGBoost", "Neural Network"],
    "Accuracy": [accuracy_score(y_test, lr_pred), accuracy_score(y_test, rf_pred),
                 accuracy_score(y_test, xgb_pred), accuracy_score(y_test, mlp_pred)],
    "Precision": [precision_score(y_test, lr_pred), precision_score(y_test, rf_pred),
                  precision_score(y_test, xgb_pred), precision_score(y_test, mlp_pred)],
    "Recall": [recall_score(y_test, lr_pred), recall_score(y_test, rf_pred),
               recall_score(y_test, xgb_pred), recall_score(y_test, mlp_pred)],
    "F1 Score": [f1_score(y_test, lr_pred), f1_score(y_test, rf_pred),
                 f1_score(y_test, xgb_pred), f1_score(y_test, mlp_pred)],
    "AUC-ROC": [roc_auc_score(y_test, lr_pred), roc_auc_score(y_test, rf_pred),
                roc_auc_score(y_test, xgb_pred), roc_auc_score(y_test, mlp_pred)]
})

print(results)
```

```
Evaluation Metrics for Logistic Regression:
Accuracy: 0.8810
Precision: 0.8000
Recall: 0.3404
F1 Score: 0.4776
AUC-ROC: 0.6621
----------------------------------------
Evaluation Metrics for Random Forest:
Accuracy: 0.8299
Precision: 0.3846
Recall: 0.1064
F1 Score: 0.1667
AUC-ROC: 0.5370
----------------------------------------
Evaluation Metrics for XGBoost:
Accuracy: 0.8605
Precision: 0.6875
Recall: 0.2340
F1 Score: 0.3492
AUC-ROC: 0.6069
----------------------------------------
Evaluation Metrics for Neural Network:
Accuracy: 0.8571
Precision: 0.5610
Recall: 0.4894
F1 Score: 0.5227
AUC-ROC: 0.7082
----------------------------------------
```



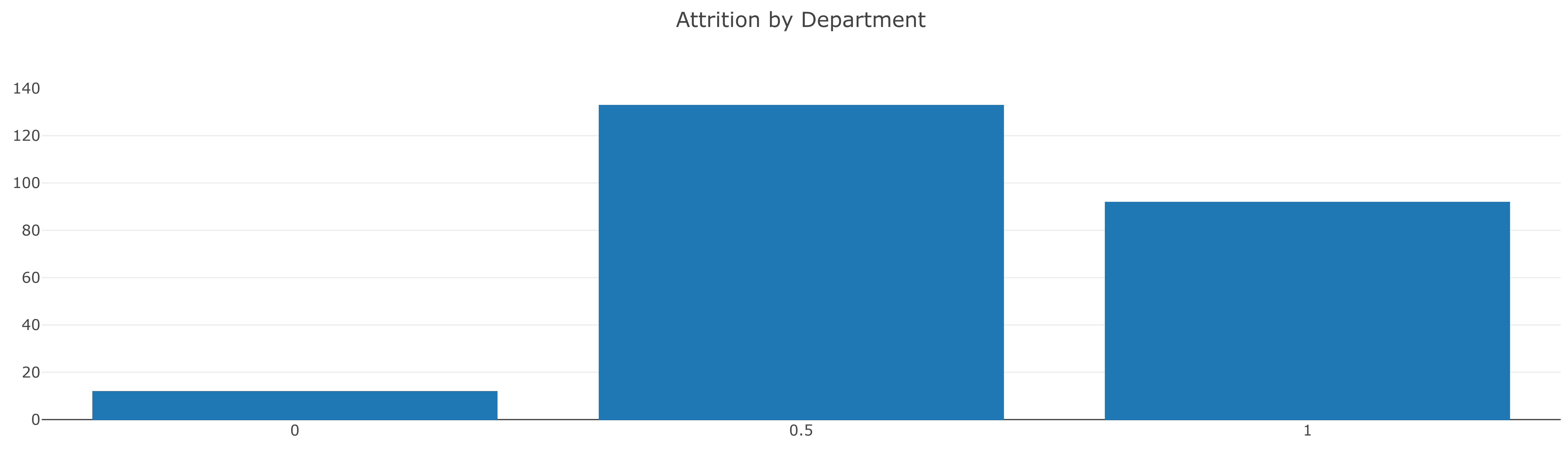

| | Model | Accuracy | Precision | Recall | F1 Score | AUC-ROC |
|---|---|---|---|---|---|---|
| 0 | Logistic Regression | 0.880952 | 0.800000 | 0.340426 | 0.477612 | 0.662116 |
| 1 | Random Forest | 0.829932 | 0.384615 | 0.106383 | 0.166667 | 0.536997 |
| 2 | XGBoost | 0.860544 | 0.687500 | 0.234043 | 0.349206 | 0.606900 |
| 3 | Neural Network | 0.857143 | 0.560976 | 0.489362 | 0.522727 | 0.708244 |

## 7) Creativity:

Designed an interactive Plotly Dash dashboard that displays predictions and attrition trends in real time. Implemented SHAP values for model explainability and identified top predictors such as Overtime and Monthly Income.

```python
import dash
from dash import dcc, html

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Employee Attrition Dashboard"),
    dcc.Graph(
        id='attrition-graph',
        figure={
            'data': [{'x': df['Department'], 'y': df['Attrition'], 'type': 'bar'}],
            'layout': {'title': 'Attrition by Department'}
        }
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

# Employee Attrition Dashboard




```python
import shap

explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)

# Summary plot
shap.summary_plot(shap_values, X_test)
```