Employee Attrition Prediction: Project Documentation

1. Dataset Analysis

- Loading Dataset: The dataset "IBM HR Analytics Employee Attrition & Performance" is loaded using pandas' read_csv function.

```
[ ] import pandas as pd

    df = pd.read_csv("/content/IBM HR Analytics Employee Attrition & Performance.csv")
```

- Exploratory Data Analysis (EDA):
  - head(): Displays the first few rows of the dataset.

```
df.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1

5 rows × 35 columns

  - info(): Provides information about the dataset including data types and missing values.

```
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
```

  - describe(): Generates summary statistics for numerical columns.

```
df.describe()
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000

8 rows × 26 columns

- isnull().sum(): Calculates the number of missing values in each column.

```
df.isnull().sum()

Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
```

- Number of unique values in each column is printed.

```python
print("Number of unique values in each column")
for column in df.columns:
    print(f"{column}: {df[column].nunique()}")
```

```
Number of unique values in each column
Age: 43
Attrition: 2
BusinessTravel: 3
DailyRate: 886
Department: 3
DistanceFromHome: 29
Education: 5
EducationField: 6
EmployeeCount: 1
EmployeeNumber: 1470
EnvironmentSatisfaction: 4
Gender: 2
HourlyRate: 71
JobInvolvement: 4
JobLevel: 5
JobRole: 9
JobSatisfaction: 4
MaritalStatus: 3
MonthlyIncome: 1349
MonthlyRate: 1427
NumCompaniesWorked: 10
Over18: 1
OverTime: 2
PercentSalaryHike: 15
PerformanceRating: 2
RelationshipSatisfaction: 4
StandardHours: 1
StockOptionLevel: 4
TotalWorkingYears: 40
TrainingTimesLastYear: 7
WorkLifeBalance: 4
```

- Data Preprocessing:
    - We notice that 'EmployeeCount', 'Over18', 'StandardHours' have only one unique values and 'EmployeeNumber' has 1470 unique values. This features aren't useful for us, So we are going to drop those columns.
    - Non-essential columns ('EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours') are dropped.

```python
df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis="columns", inplace=True)
```

    - One-hot encoding is performed for categorical variables.
    - Label encoding is applied to the 'Attrition' column.
    - Numeric and categorical columns are segregated.

```
# Perform one-hot encoding for categorical variables
df_encoded = pd.get_dummies(df)
```

```
object_col = []
for column in df.columns:
    if df[column].dtype == object and len(df[column].unique()) <= 30:
        object_col.append(column)
        print(f"{column} : {df[column].unique()}")
        print(df[column].value_counts())
        print("====================================")
object_col.remove('Attrition')
```

```
Attrition : ['Yes' 'No']
Attrition
No     1233
Yes     237
Name: count, dtype: int64
====================================
BusinessTravel : ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
BusinessTravel
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: count, dtype: int64
====================================
Department : ['Sales' 'Research & Development' 'Human Resources']
Department
Research & Development    961
Sales                     446
Human Resources            63
Name: count, dtype: int64
====================================
EducationField : ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']
EducationField
```

```
len(object_col)
```

7

```
from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
df["Attrition"] = label.fit_transform(df.Attrition)
```

```
disc_col = []
for column in df.columns:
    if df[column].dtypes != object and df[column].nunique() < 30:
        print(f"{column} : {df[column].unique()}")
        disc_col.append(column)
        print("====================================")
disc_col.remove('Attrition')
```

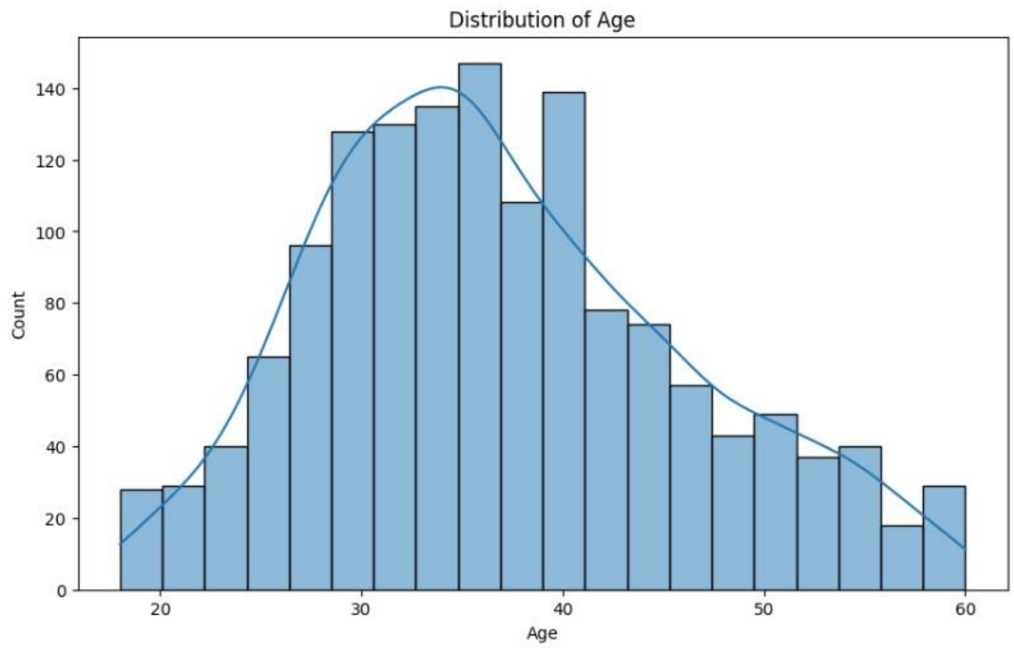
```
Attrition : [1 0]
====================================
DistanceFromHome : [ 1  8  2  3 24 23 27 16 15 26 19 21  5 11  9  7  6 10  4 25 12 18 29 22
 14 20 28 17 13]
====================================
Education : [2 1 4 3 5]
====================================
EnvironmentSatisfaction : [2 3 4 1]
====================================
JobInvolvement : [3 2 4 1]
====================================
JobLevel : [2 1 3 4 5]
====================================
JobSatisfaction : [4 2 3 1]
====================================
NumCompaniesWorked : [8 1 6 9 0 4 5 2 7 3]
====================================
PercentSalaryHike : [11 23 15 12 13 20 22 21 17 14 16 18 19 24 25]
```

```python
cont_col = []
for column in df.columns:
    if df[column].dtypes != object and df[column].nunique() > 30:
        print(f"{column} : Minimum: {df[column].min()}, Maximum: {df[column].max()}")
        cont_col.append(column)
        print("====================================")
```

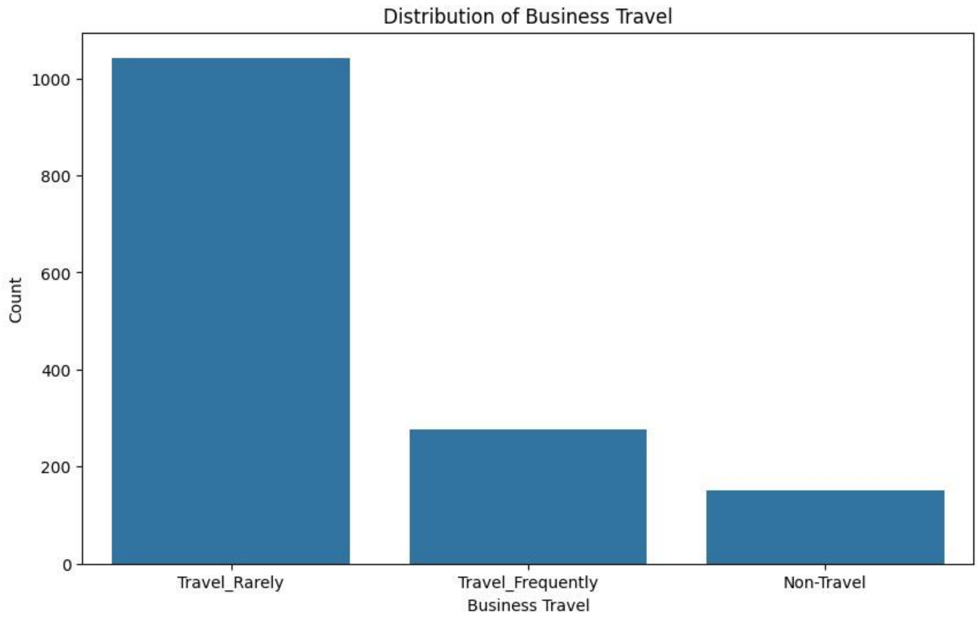
```
Age : Minimum: 18, Maximum: 60
====================================
DailyRate : Minimum: 102, Maximum: 1499
====================================
HourlyRate : Minimum: 30, Maximum: 100
====================================
MonthlyIncome : Minimum: 1009, Maximum: 19999
====================================
MonthlyRate : Minimum: 2094, Maximum: 26999
====================================
TotalWorkingYears : Minimum: 0, Maximum: 40
====================================
YearsAtCompany : Minimum: 0, Maximum: 40
====================================
```

- Data Visualization:
    - Histogram of 'Age', bar plot of 'BusinessTravel', and box plot of 'MonthlyIncome' across 'JobRole' are plotted.
    - Histogram of 'YearsAtCompany' with respect to 'Attrition' is visualized.
    - Correlation heatmap of numerical features is generated.

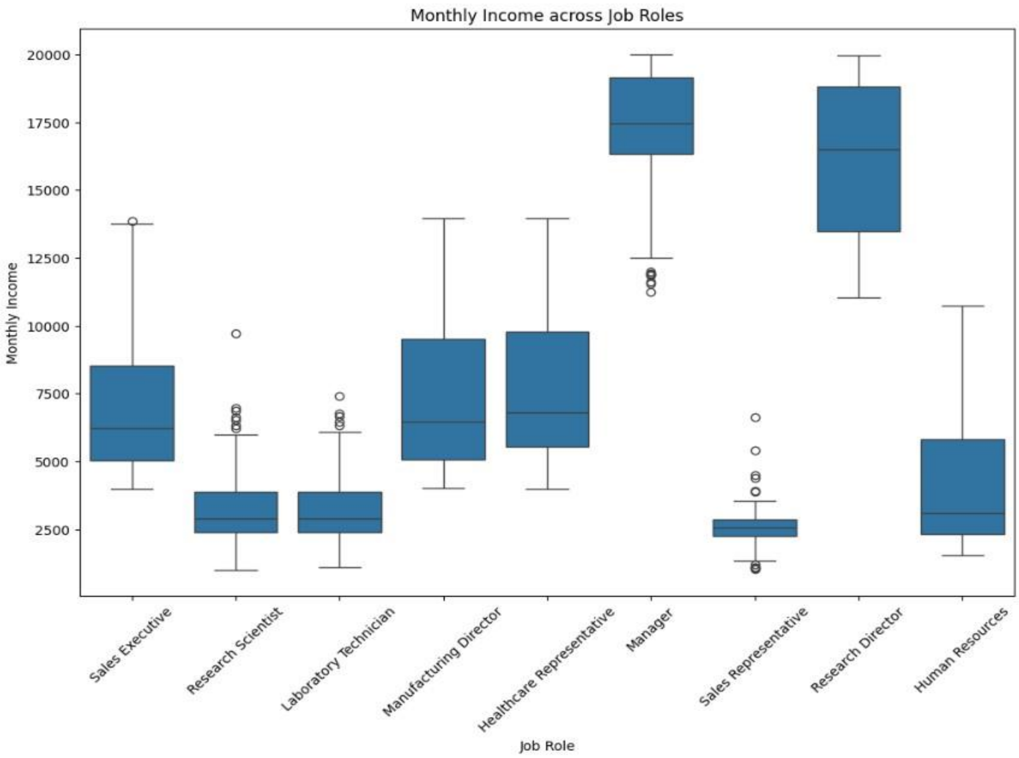
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram of Age
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', bins=20, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



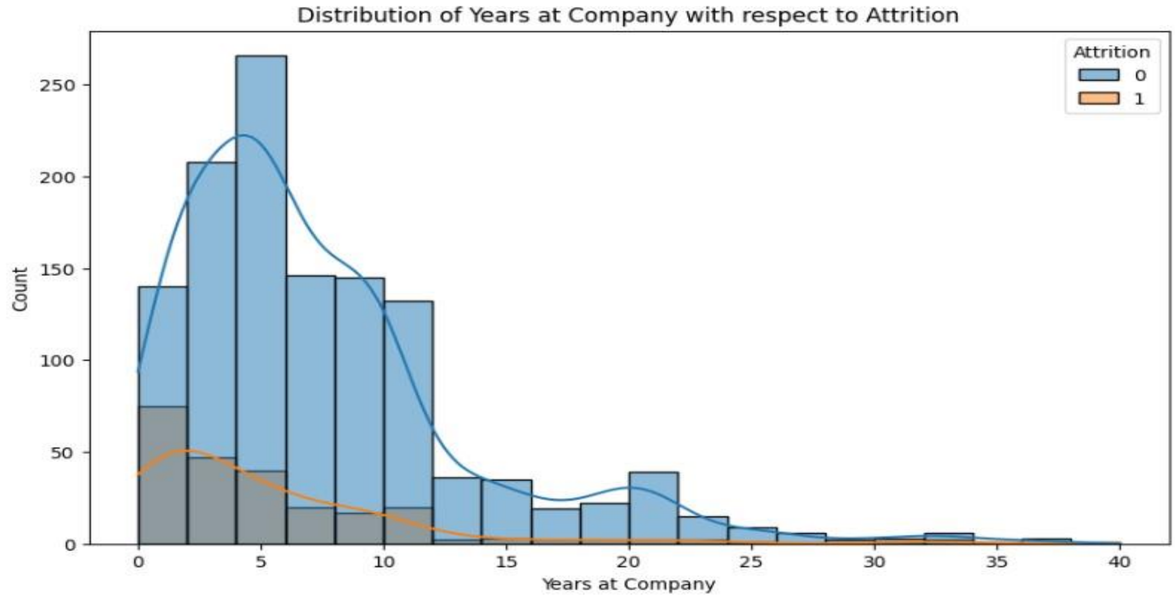
```python
# Bar plot of BusinessTravel
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='BusinessTravel')
plt.title('Distribution of Business Travel')
plt.xlabel('Business Travel')
plt.ylabel('Count')
plt.show()
```



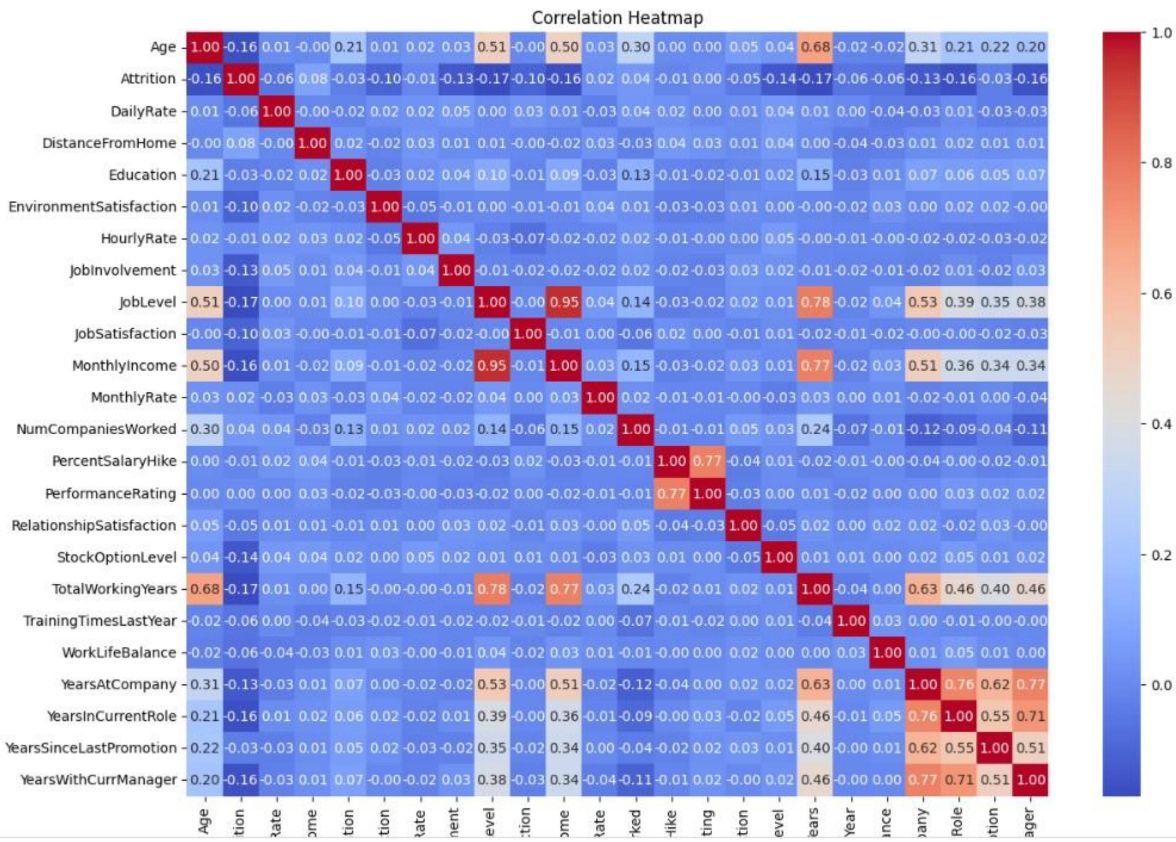

```python
# Box plot of MonthlyIncome across JobRole
plt.figure(figsize=(12, 8))
sns.boxplot(data=df, x='JobRole', y='MonthlyIncome')
plt.xticks(rotation=45)
plt.title('Monthly Income across Job Roles')
plt.xlabel('Job Role')
plt.ylabel('Monthly Income')
plt.show()
```



```python
# Histogram of YearsAtCompany with respect to Attrition
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='YearsAtCompany', hue='Attrition', bins=20, kde=True)
plt.title('Distribution of Years at Company with respect to Attrition')
plt.xlabel('Years at Company')
plt.ylabel('Count')
plt.show()
```




```python
# Drop non-numeric columns
df_numeric = df.select_dtypes(include=['number'])

# Create the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(df_numeric.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



2. Model Development

- Feature Selection:

    - Features with low correlation to the target variable 'Attrition' are removed.

```
import numpy as np

feature_correlation = data.drop('Attrition', axis=1).corrwith(data.Attrition).sort_values()
model_col = feature_correlation[np.abs(feature_correlation) > 0.02].index
len(model_col)
```

```
92
```

- Model Training:

    - Data is split into training and testing sets.
    - Standard scaling is applied to the features.
    - Logistic Regression and AdaBoost classifiers are trained on the dataset.

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler

X = data.drop('Attrition', axis=1)
y = data.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
                                                    stratify=y)

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
X_std = scaler.transform(X)
```

```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

```
y_test.value_counts()[0] / y_test.shape[0]
```

```
0.8390022675736961
```

```python
stay = (y_train.value_counts()[0] / y_train.shape)[0]
leave = (y_train.value_counts()[1] / y_train.shape)[0]

print("===============TRAIN=================")
print(f"Staying Rate: {stay * 100:.2f}%")
print(f"Leaving Rate: {leave * 100 :.2f}%")

stay = (y_test.value_counts()[0] / y_test.shape)[0]
leave = (y_test.value_counts()[1] / y_test.shape)[0]

print("===============TEST=================")
print(f"Staying Rate: {stay * 100:.2f}%")
print(f"Leaving Rate: {leave * 100 :.2f}%")
```

```
===============TRAIN=================
Staying Rate: 83.87%
Leaving Rate: 16.13%
===============TEST=================
Staying Rate: 83.90%
Leaving Rate: 16.10%
```

3. Model Evaluation

- Evaluation Metrics:
  - Confusion matrix, accuracy score, precision, recall, and F1-score are computed for both training and testing sets.
- ROC AUC Scores:
  - ROC AUC scores are calculated for Logistic Regression and AdaBoost classifiers.

```python
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, roc_auc_score

def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```python
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(solver='liblinear', penalty='l1')
lr_clf.fit(X_train_std, y_train)

evaluate(lr_clf, X_train_std, X_test_std, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[849  14]
 [ 59 107]]
ACCURACY SCORE:
0.9291
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.935022    0.884298  0.929057     0.909660      0.926839
recall       0.983778    0.644578  0.929057     0.814178      0.929057
f1-score     0.958780    0.745645  0.929057     0.852212      0.924397
support    863.000000  166.000000  0.929057  1029.000000   1029.000000
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[348  22]
 [ 43  28]]
ACCURACY SCORE:
0.8526
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.890026   0.560000  0.852608    0.725013      0.836892
recall       0.940541   0.394366  0.852608    0.667453      0.852608
f1-score     0.914586   0.462810  0.852608    0.688698      0.841851
support    370.000000  71.000000  0.852608  441.000000    441.000000
```

```python
scores_dict = {
    'Logistic Regression': {
        'Train': roc_auc_score(y_train, lr_clf.predict(X_train)),
        'Test': roc_auc_score(y_test, lr_clf.predict(X_test)),
    },
}
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

```python
from sklearn.ensemble import AdaBoostClassifier

ab_clf = AdaBoostClassifier()
ab_clf.fit(X_train, y_train)

evaluate(ab_clf, X_train, X_test, y_train, y_test)
```

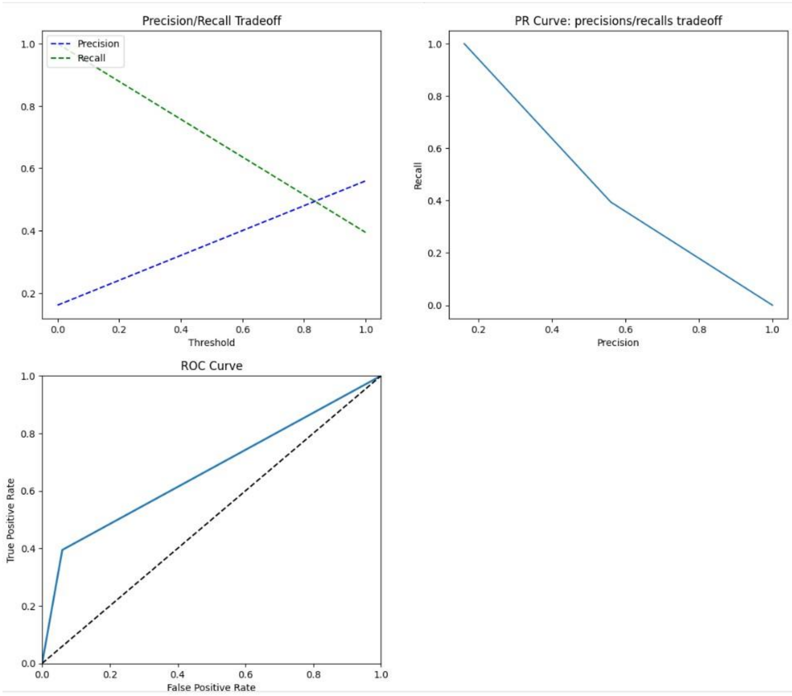
```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[952  26]
 [101  97]]
ACCURACY SCORE:
0.8920
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.904084    0.788618  0.892007     0.846351      0.884643
recall       0.973415    0.489899  0.892007     0.731657      0.892007
f1-score     0.937469    0.604361  0.892007     0.770915      0.881385
support    978.000000  198.000000  0.892007  1176.000000   1176.000000
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[233  22]
 [ 28  11]]
ACCURACY SCORE:
0.8299
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.892720   0.333333  0.829932    0.613027      0.818516
recall       0.913725   0.282051  0.829932    0.597888      0.829932
f1-score     0.903101   0.305556  0.829932    0.604328      0.823835
support    255.000000  39.000000  0.829932  294.000000    294.000000
```

```
scores_dict['AdaBoost'] = {
        'Train': roc_auc_score(y_train, ab_clf.predict(X_train)),
        'Test': roc_auc_score(y_test, ab_clf.predict(X_test)),
    }
```

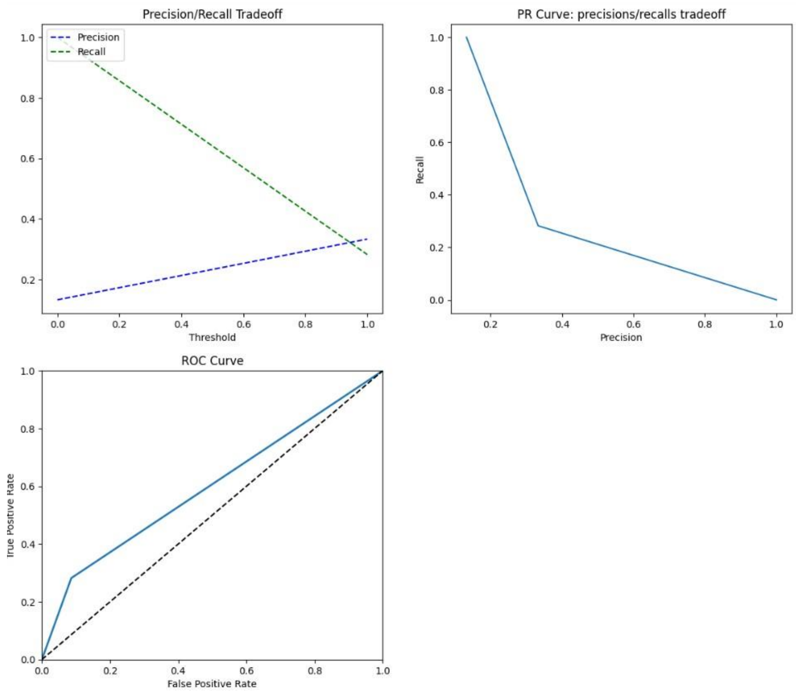
4. Optimization Techniques

- Further Evaluation:
    - Precision-recall curves and ROC curves are plotted to visualize model performance.
- Model Comparison:
    - Performance metrics and ROC AUC scores are compared between the Logistic Regression and AdaBoost models.
- Model Scores Visualization:
    - Model scores are visualized using a horizontal bar plot.

```
from sklearn.metrics import precision_recall_curve, roc_curve

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.title("Precision/Recall Tradeoff")


def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')


precisions, recalls, thresholds = precision_recall_curve(y_test, lr_clf.predict(X_test_std))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, lr_clf.predict(X_test_std))
plot_roc_curve(fpr, tpr)
```




```python
precisions, recalls, thresholds = precision_recall_curve(y_test, ab_clf.predict(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, ab_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```

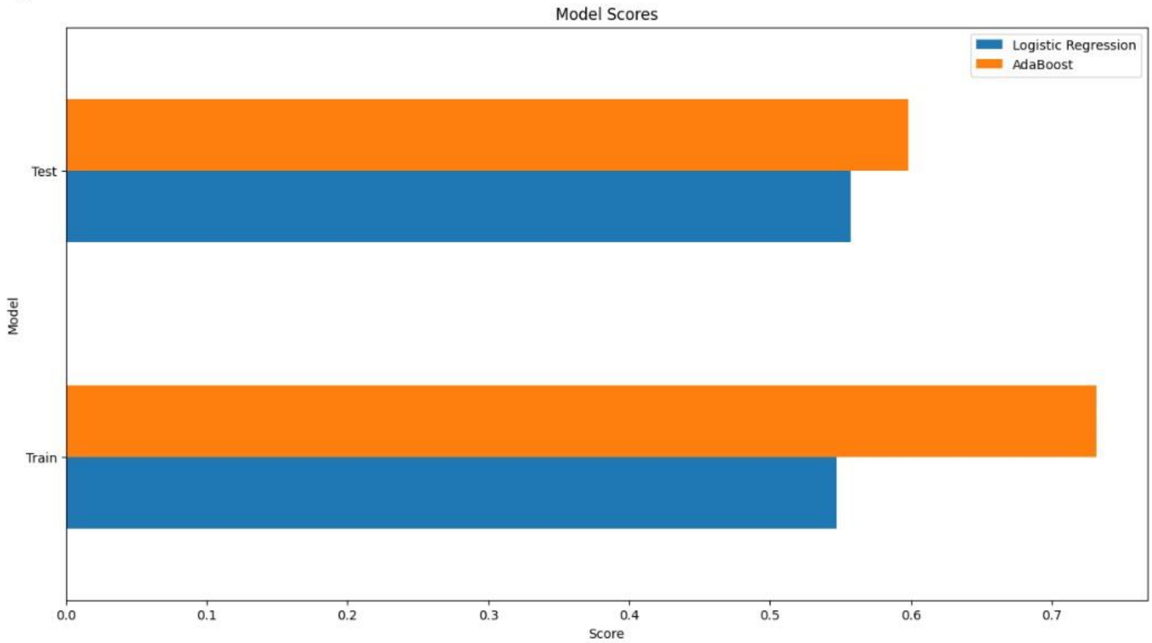


```
ml_models = {
    'Logistic Regression': lr_clf,
    'AdaBoost': ab_clf
}

for model in ml_models:
    print(f"{model.upper():{30}} roc_auc_score: {roc_auc_score(y_test, ml_models[model].predict(X_test)):.3f}")
```

```
LOGISTIC REGRESSION            roc_auc_score: 0.557
ADABOOST                       roc_auc_score: 0.598
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

```
# Convert scores_dict to a DataFrame
scores_df = pd.DataFrame(scores_dict)

# Create a horizontal bar plot using Matplotlib
plt.figure(figsize=(10, 6))
scores_df.plot(kind='barh', figsize=(15, 8))
plt.xlabel('Score')
plt.ylabel('Model')
plt.title('Model Scores')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



5. Summary

- Findings:
    - The dataset is imbalanced with approximately 84% of employees staying and 16% leaving.
    - Both Logistic Regression and AdaBoost models achieved reasonable accuracy, but their ROC AUC scores suggest room for improvement.
- Challenges:
    - Dealing with imbalanced data and interpreting complex model results were major challenges encountered.
- Recommendations:
    - Addressing imbalanced data, exploring feature importance, and further evaluation of models are recommended for improving performance and insights.