SoluLab – Machine Learning Internship 2024 – Final Assessment Report

Problem no. 1

Employee Attrition Prediction using ML

Github Link for Notebook Repository:

https://github.com/KavinKarthik18/Employee-Attrition-Prediction-ML-SoluLab

Project Implementation and Documentation by

Kavin Karthik V

B.tech Computer Science with spl in AI and Robotics

Vellore Institute of Technology, Chennai

Problem Statement:

Implement a machine learning model to predict employee attrition by analyzing key factors such as job satisfaction, compensation, tenure, and work environment.

My Approach to the problem statement:

1. Identify key indicators of potential attrition through exploratory data analysis and visualization techniques, providing insights to HR and management.

2. Develop a predictive model that can accurately forecast the likelihood of an employee leaving the organization, using relevant features from the dataset.

3. Evaluate and refine the model's performance to ensure reliable predictions across different employee groups and departments.

A] Data Preprocessing:

1. Initial Data Exploration:
    - Examined the first few rows using df.head()
    - Checked basic information about the dataset using df.info()
    - Obtained summary statistics with df.describe()\

```
# Display summary statistics of numerical columns
print(df.describe())
```

```
               Age    DailyRate  DistanceFromHome    Education  EmployeeCount  \
count  1470.000000  1470.000000       1470.000000  1470.000000         1470.0
mean     36.923810   802.485714          9.192517     2.912925            1.0
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
...
75%              3.000000              7.000000
max             15.000000             17.000000

[8 rows x 26 columns]
```

2. Handling Missing Data:

- Checked for missing values using df.isnull().sum()
- Decided to remove unwanted columns, since they were redundant and not anywhere helpful to improve model performance on attrition prediction.

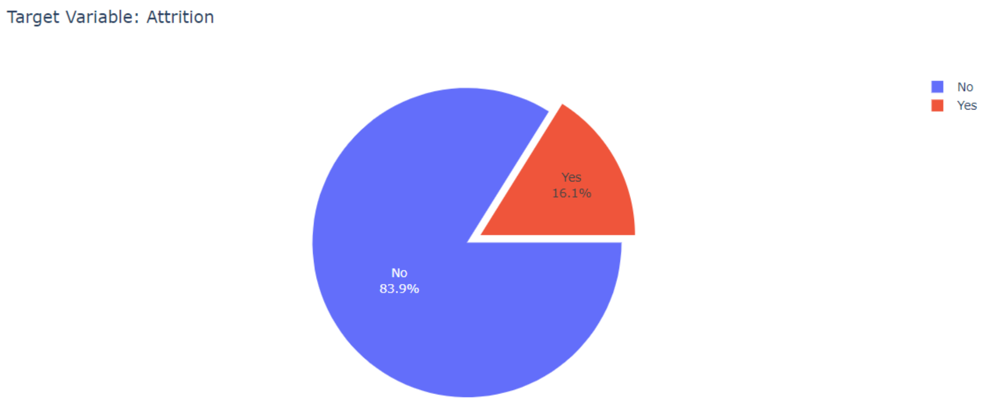
```
#Removing unwanted columns
df = df.drop(['EmployeeCount',
              'EmployeeNumber',
              'Over18',
               'StandardHours'],axis = 1)
df.columns
```

```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager'],
      dtype='object')
```

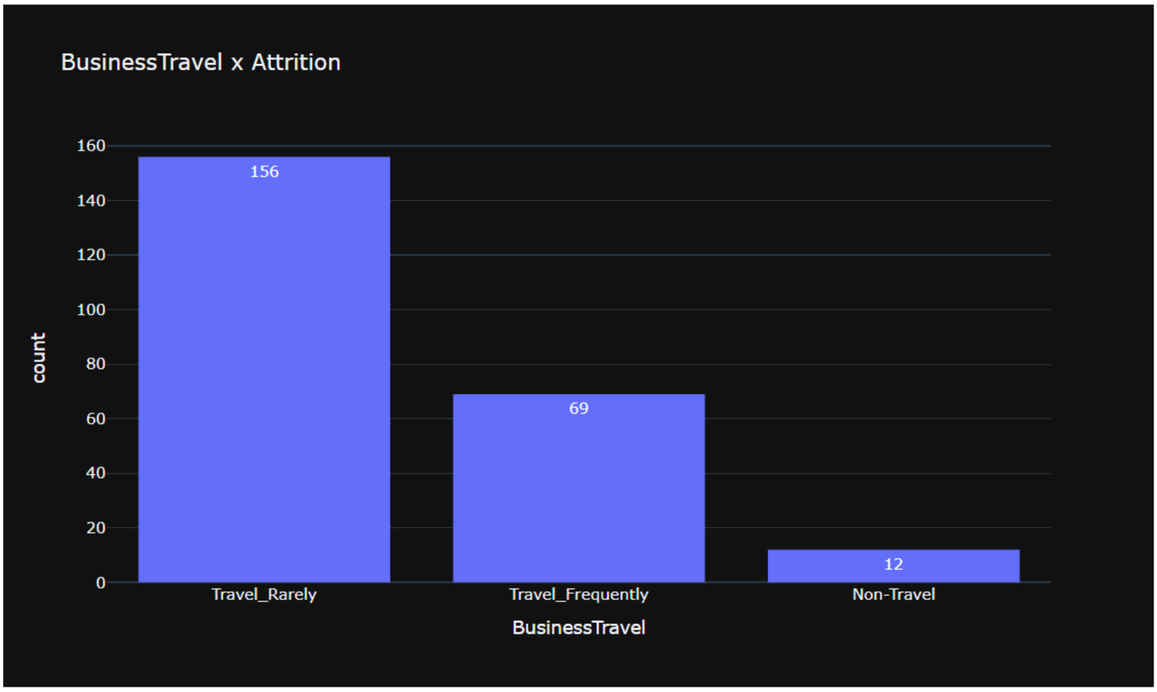
NOTE: Matplotlib graphs and charts are not visible in the GitHub preview. Kindly download the .ipynb file provided in the repository (link in Page 1) and run it locally or in Google Colab to view interactive plots which are explained below.

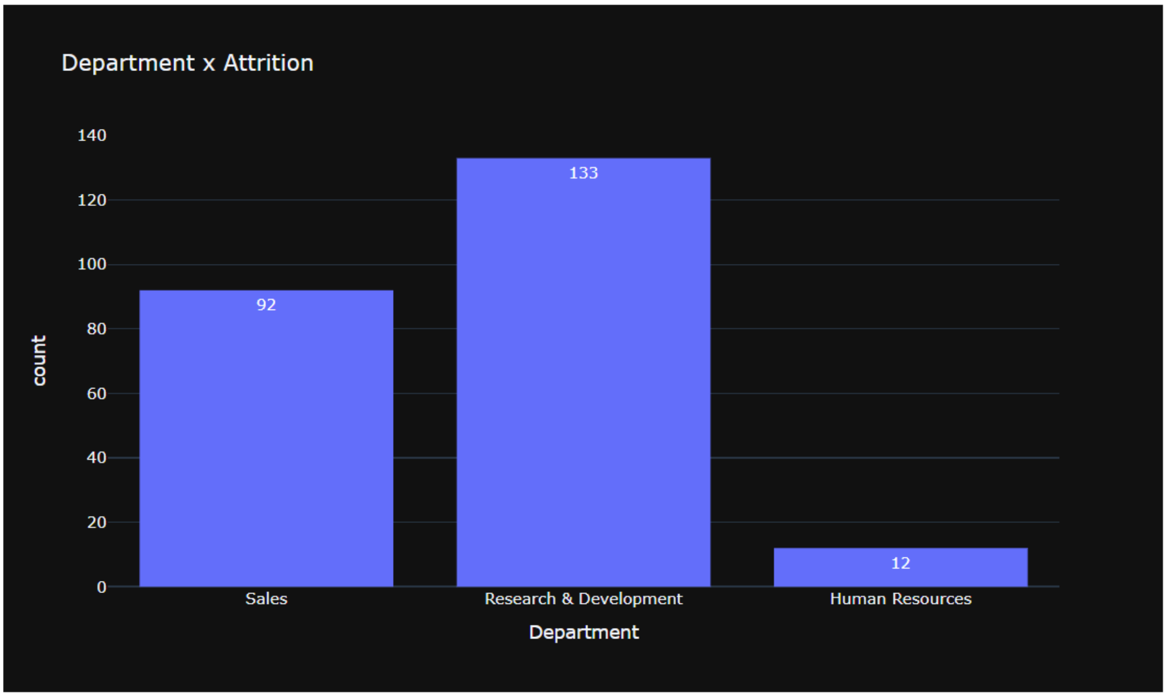
3. Exploratory Data Analysis

Visualised target variable classes is critical for identifying class imbalance, which can significantly impact model performance.

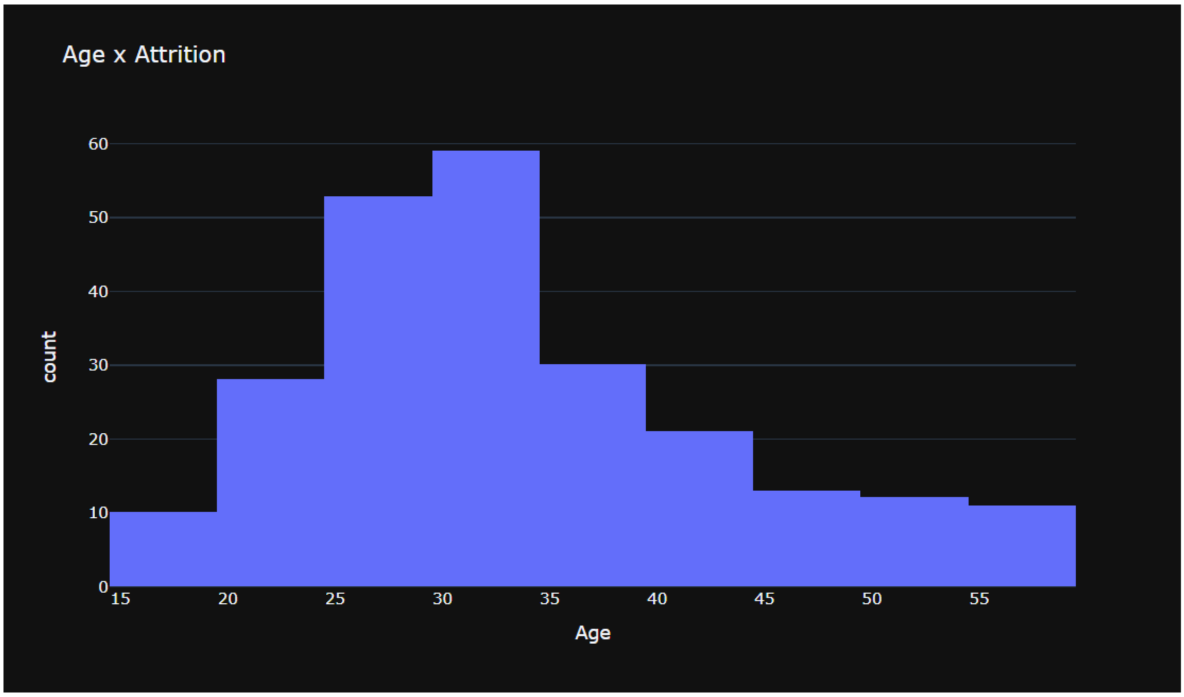


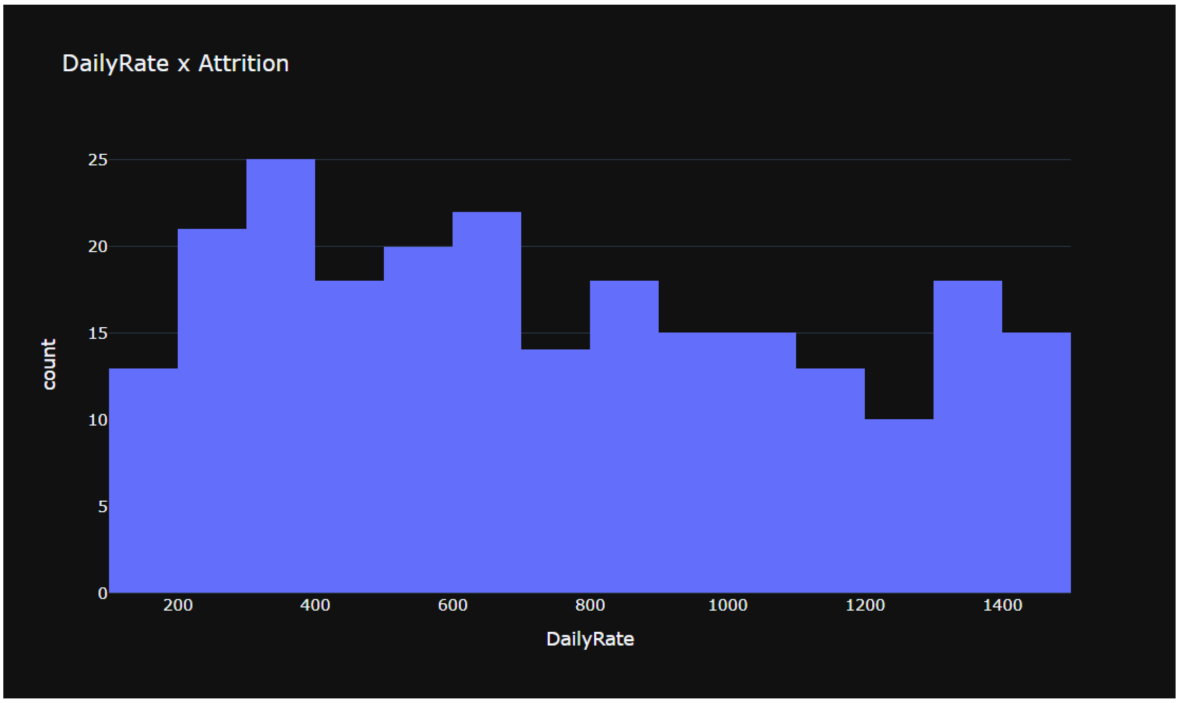
- Analyzed relationships between various categorical and numerical features and attrition using bar plots and number graphs.





The above 2 and the rest of the bar plots in the notebook show us the comparison of attrition with various other categorical attributes.





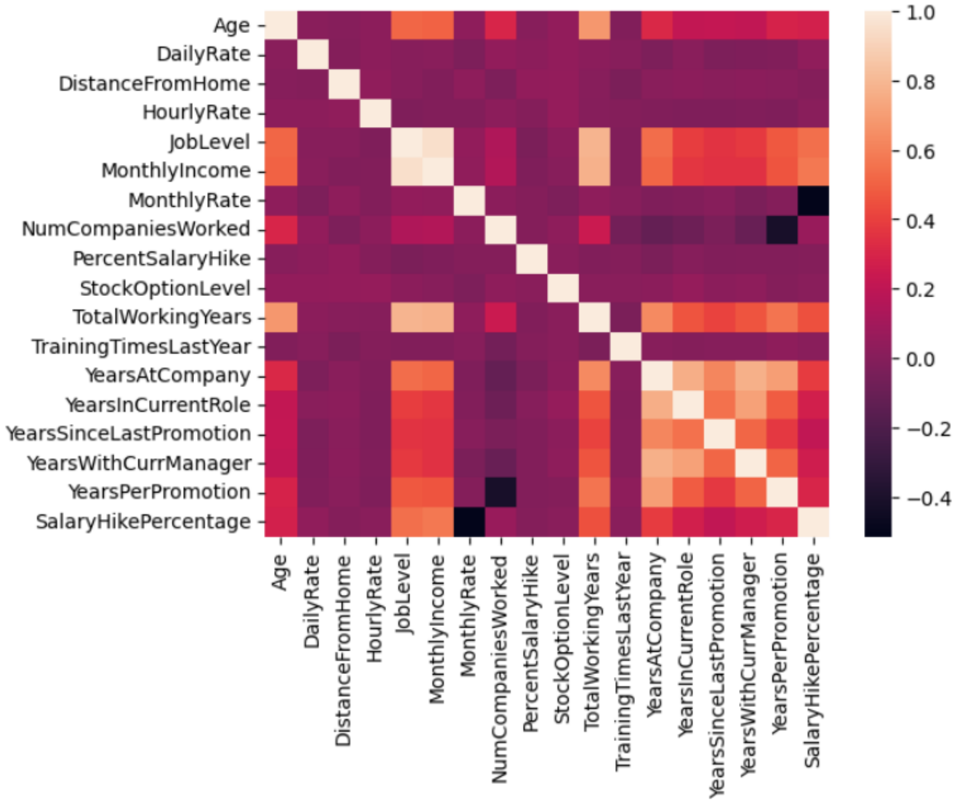
Above 2 are Some examples of numerical attributes comparison.

Conclusions from Numerical Variables Analysis:

- Age and attrition: Employees aged 25-35 have the highest attrition rates Attrition decreases as employees get older
- Tenure and attrition: Employees with fewer years at the company are more likely to leave Attrition decreases with more years in current role and total work experience

- Income and attrition: Majority of departing employees had monthly incomes between 2000-3000 Lower percentage salary hikes correlate with higher attrition
- Career stage: Most departing employees were early in their careers (less than 10 years total work experience)
- Management relationship: Many departing employees had less than 2 years working with their last manager
- **Overall pattern:**
- ***Attrition is highest among younger, less experienced employees with lower incomes and shorter tenures Attrition decreases with age, experience, higher income, and longer company tenure***

Correlation Heatmap:



Conclusions from Corelations Heatmap Analysis (Observe Diagonals)

- Age and DistanceFromHome show moderate positive correlation.
- JobSatisfaction correlates positively with EnvironmentSatisfaction.
- MonthlyRate has strong positive correlation with YearsAtCompany and YearsSinceLastPromotion.
- TrainingTimesLastYear correlates positively with YearsAtCompany and YearsSinceLastPromotion.
- StockOptionLevel shows negative correlation with JobSatisfaction.

- PercentSalaryHike has weak correlations with most variables.
- RelationshipSatisfaction shows little correlation with other factors.
- EmployeeCount appears to have no significant correlations.
- JobInvolvement has weak to moderate correlations with several variables

B] Feature Engineering:

Introduced and Calculated the following features to eliminate redundant information that combines various other attributes separately:

```python
# 1. Years per promotion
df['YearsPerPromotion'] = df['TotalWorkingYears'] / (df['NumCompaniesWorked'] + 1)

# 2. Salary hike percentage (using safe division)
df['SalaryHikePercentage'] = np.where(
    df['MonthlyRate'] != 0,
    (df['MonthlyIncome'] - df['MonthlyRate']) / df['MonthlyRate'] * 100,
    0
)
```

```python
# Display the first few rows of the dataset with new features
print(df[['YearsPerPromotion', 'SalaryHikePercentage']].head())

# Display summary statistics of the new features
print(df[['YearsPerPromotion', 'SalaryHikePercentage']].describe())
```

```
   YearsPerPromotion  SalaryHikePercentage
0           0.888889            -69.233534
1           5.000000            -79.403381
2           1.000000            -12.771285
3           4.000000            -87.439009
4           0.600000            -79.148629
       YearsPerPromotion  SalaryHikePercentage
count        1470.000000           1470.000000
mean            4.193478            -33.082105
std             4.035504             84.209929
min             0.000000            -96.262825
25%             1.600000            -77.932986
50%             3.000000            -60.319155
75%             5.000000            -21.562278
max            38.000000            817.741176
```

Justifications for the above features are mentioned in the notebook markdown.

Another important Feature Engineering aspect is **Encoding all categorical variables using both the Ordinal Encoder and the One Hot Encoder.**

```python
# Enoding categorical variables with Ordinal Encoder
OE = OrdinalEncoder()
columns_OE = ['BusinessTravel', 'Education', 'EnvironmentSatisfaction', 'JobInvolvement',
        'JobSatisfaction','WorkLifeBalance','PerformanceRating','RelationshipSatisfaction']
X_train[columns_OE] = OE.fit_transform(X_train[columns_OE])
X_test[columns_OE] = OE.transform(X_test[columns_OE])
X_train
```

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender	HourlyRate	...	StockOptionLevel	TotalWorkingYears
721	50	2.0	939	Research & Development	24	0.0	Life Sciences	3.0	Male	95	...	1	22
520	48	2.0	817	Sales	2	1.0	Marketing	2.0	Male	56	...	0	12
379	55	2.0	1311	Research & Development	2	0.0	Life Sciences	0.0	Female	97	...	0	30
184	53	2.0	1084	Research & Development	13	2.0	Medical	3.0	Female	57	...	2	5
213	51	2.0	1469	Research & Development	8	4.0	Life Sciences	2.0	Male	81	...	2	16
...	...	...	...	...	...	...	...	...	...	...	...	...	...
104	37	0.0	1040	Research & Development	2	2.0	Life Sciences	0.0	Male	100	...	1	17
1380	35	2.0	682	Sales	18	4.0	Medical	2.0	Male	71	...	1	6
798	33	2.0	1017	Research & Development	25	0.0	Medical	1.0	Male	55	...	0	5
1189	33	2.0	392	Sales	2	4.0	Medical	3.0	Male	93	...	2	6
1173	36	2.0	711	Research & Development	5	4.0	Life Sciences	2.0	Female	42	...	2	9

Justification and Observation for using the encoders:

Categories such as Education and Job Involvement can easily be encoded with the Ordinal Encoder since there is some sort of hierarchy among their values, but Department, for instance, would be better encoded with the One Hot Encoder, since there is no department lesser or more than another.

Rescaling Data:

```python
# Rescaling Data
Scaler = MinMaxScaler()
Scaling_Cols = ['TrainingTimesLastYear','YearsAtCompany','TotalWorkingYears',
                'YearsInCurrentRole','YearsSinceLastPromotion','YearsWithCurrManager',
                'PercentSalaryHike','Age','DailyRate','DistanceFromHome','HourlyRate',
                'MonthlyIncome','MonthlyRate','NumCompaniesWorked']
X_train[Scaling_Cols] = Scaler.fit_transform(X_train[Scaling_Cols])
X_test[Scaling_Cols] = Scaler.transform(X_test[Scaling_Cols])
X_train
```

Python

	Age	BusinessTravel	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	Gender	HourlyRate	JobInvolvement	JobLevel	...	11	12	13	14	15	16	17	18	19
721	0.761905	2.0	0.599141	0.821429	0.0	3.0	1	0.928571	0.0	4	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.(
520	0.714286	2.0	0.511811	0.035714	1.0	2.0	1	0.371429	3.0	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.(
379	0.880952	2.0	0.865426	0.035714	0.0	0.0	0	0.957143	0.0	4	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
184	0.833333	2.0	0.702935	0.428571	2.0	3.0	0	0.385714	3.0	2	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.(
213	0.785714	2.0	0.978525	0.250000	4.0	2.0	1	0.728571	2.0	3	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
104	0.452381	0.0	0.671439	0.035714	2.0	0.0	1	1.000000	2.0	2	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.(
1380	0.404762	2.0	0.415175	0.607143	4.0	2.0	1	0.585714	0.0	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.(
798	0.357143	2.0	0.654975	0.857143	0.0	1.0	1	0.357143	2.0	1	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.(
1189	0.357143	2.0	0.207588	0.035714	4.0	3.0	1	0.900000	0.0	2	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.(
1173	0.428571	2.0	0.435934	0.142857	4.0	2.0	0	0.171429	0.0	3	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.(

1176 rows × 49 columns

Justification:

In the employee attrition dataset, features have vastly different scales. For example, 'Age' might range from 18 to 60, while 'MonthlyIncome' could range from thousands to hundreds of thousands. Without rescaling, features with larger magnitudes (like MonthlyIncome) would dominate the model training process, potentially overshadowing the influence of equally important features with smaller scales.

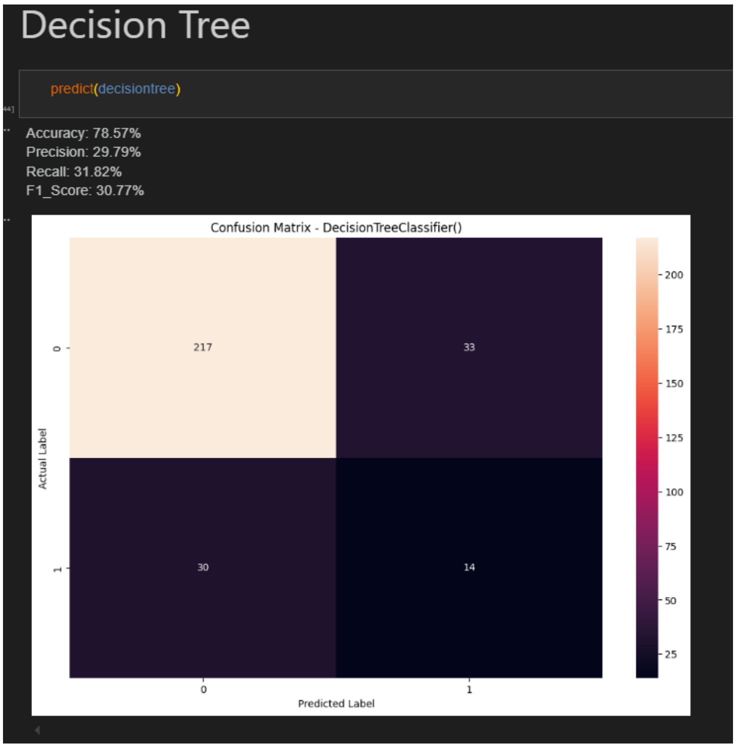
C] Model Training:

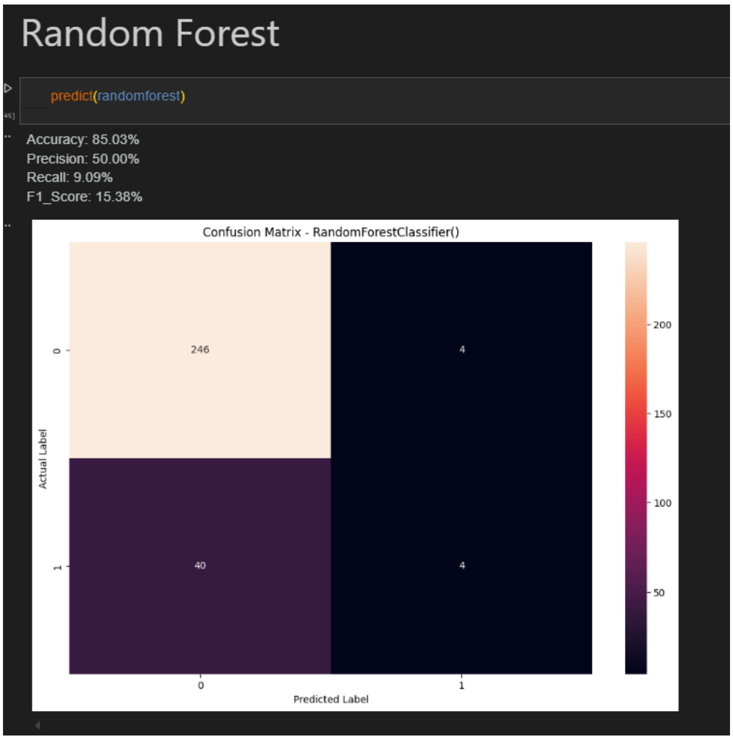
After the model was split into 20% test and 80% training data sample, the most preferable approach would be training on atleast 4 different classifier models in order to utilise the features engineered in step B].

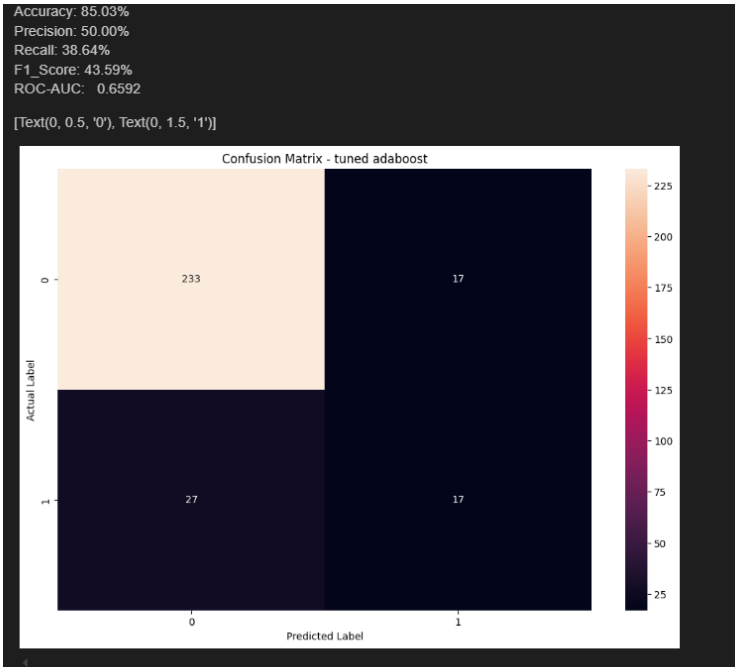
The 4 models of my choice was **DECISION TREE (initial), RANDOM FOREST, ADABOOST, GRADIENT BOOST:**

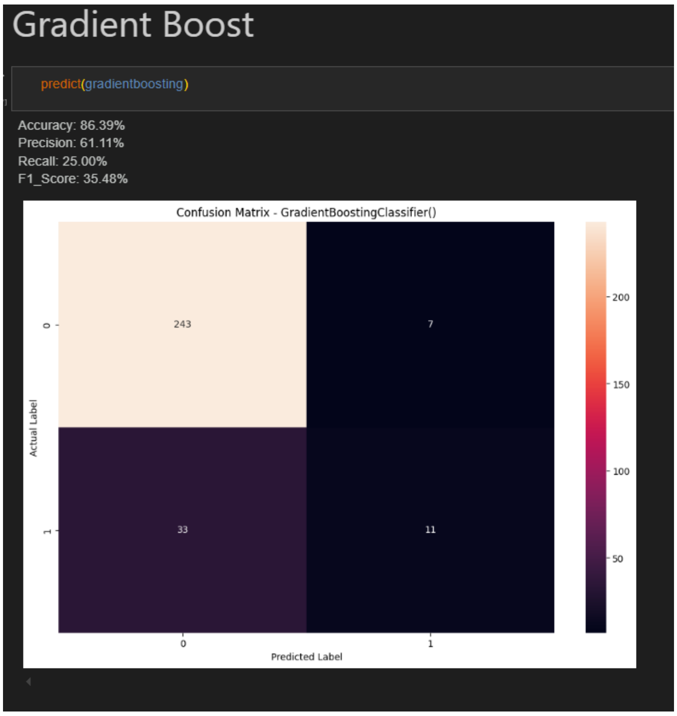
- **Chosen for its superior predictive power, ability to handle missing data, and built-in regularization, making it well-suited for the complex nature of employee attrition prediction and for its ability to handle non-linear relationships and capture complex interactions between features, which is crucial in employee attrition prediction.**

INITIAL CONFUSION MATRIX AND PEROMANCE METRICS:









For this particular employee attrition prediction the 2 important performance metrics are Accuracy and Recall scores.

In order to further improvise the model performance we perform **Hyperparameter Tuning using RandomizedSearchCV.**

D] The three major steps of this methodology of Hyperparameter tuning is:

• Set up RandomizedSearchCV: Initialize it with the model, parameter space, number of iterations, and cross-validation strategy.

• Fit the search: Apply RandomizedSearchCV to the training data, which will randomly sample parameter combinations and evaluate them using cross-validation.

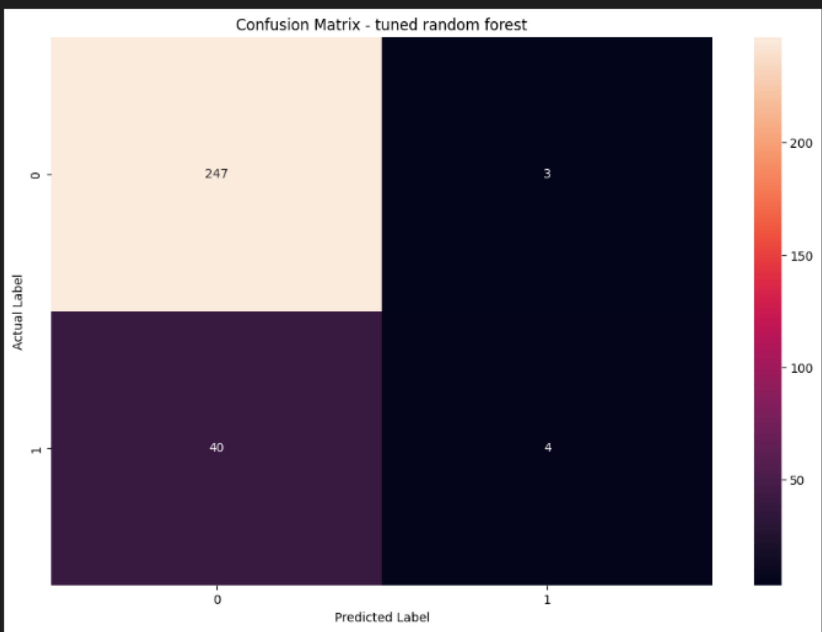
• Extract best parameters: Retrieve the best-performing parameter combination found during the search process.

```
from sklearn.model_selection import RandomizedSearchCV
randomforest_grid = {'n_estimators' : [100,150,200,250,300,350,400,450,500],
                     'max_depth':[5,1,15,20,25,30,40,50],
                     'criterion': ['gini','entropy'],
                     'min_samples_leaf':[2,5,8,10],
                     'min_samples_split': [2,5,8,10],
                     'max_features':['auto','sqrt']}
rf_tuning = RandomForestClassifier()
rf_search = RandomizedSearchCV(rf_tuning,param_distributions = randomforest_grid,
                               cv = 10, n_iter = 10, n_jobs = 1, verbose = 1, scoring = 'recall')
rf_search.fit(X_train, y_train)

Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

- **The model is fitted and evaluated 10 times (once for each fold)After all 100 fits, RandomizedSearchCV will identify the best performing hyperparameter combination based on the average performance across the 10 folds.**

POST HYPARAMETER TUNING RESULT:

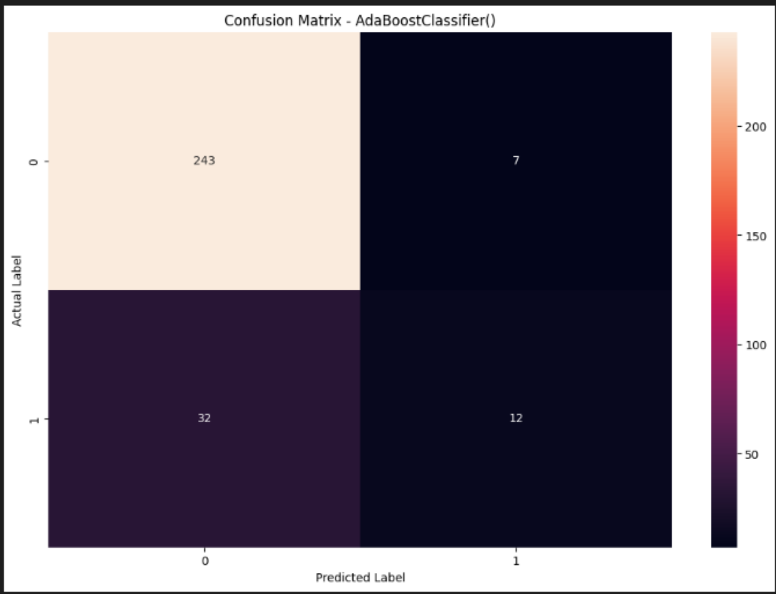
```
Accuracy: 85.37%
Precision: 57.14%
Recall: 9.09%
F1_Score: 15.69%
ROC-AUC:   0.5395

[Text(0, 0.5, '0'), Text(0, 1.5, '1')]
```



AdaBoost

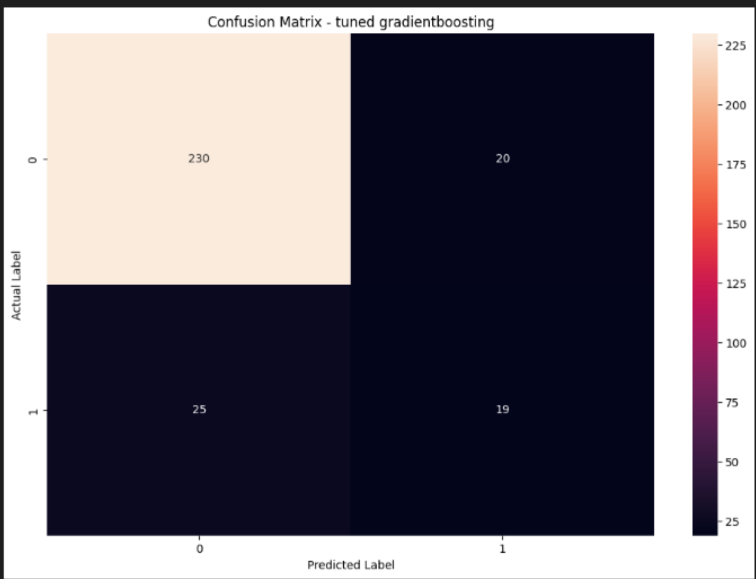
```
predict(adaboost)
```

Accuracy: 86.73%
Precision: 63.16%
Recall: 27.27%
F1_Score: 38.10%



Accuracy: 84.69%
Precision: 48.72%
Recall: 43.18%
F1_Score: 45.78%
ROC-AUC: 0.6759

[Text(0, 0.5, '0'), Text(0, 1.5, '1')]



E] MODEL INTERPRETATION:

MODEL	PRE HYPERPARAMETER TUNING	POST HYPERPARAMETER TUNING
RANDOM FOREST	**Accuracy: 85.03%** **Precision: 50.00%** **Recall: 9.09%** **F1_Score: 15.38%**	**Accuracy: 85.37%** **Precision: 57.14%** **Recall: 9.09%** **F1_Score: 15.69%** **ROC-AUC: 0.5395**
ADABOOST	**Accuracy: 86.73%** **Precision: 63.16%** **Recall: 27.2%** **F1_Score: 38.10%**	**Accuracy: 85.03%** **Precision: 50.00%** **Recall: 62.69%** **F1_Score: 43.59%** **ROC-AUC: 0.6592**
GRADIENT BOOST	**Accuracy: 84.69%** **Precision: 48.72%** **Recall: 43.18%** **F1_Score: 45.78%**	**Accuracy: 86.39%** **Precision: 61.11%** **Recall: 25.00%** **F1_Score: 35.48%** **ROC-AUC: 0.6759**

The **Random Forest** model showed minimal improvement after hyperparameter tuning. While accuracy and precision increased slightly, recall remained unchanged, resulting in only a marginal increase in F1_Score. The low recall indicates that the model is missing many actual attrition cases. The ROC-AUC of 0.5395 suggests the model is only **slightly better than random chance at distinguishing between classes.**

AdaBoost showed mixed results after tuning. While accuracy and precision decreased, recall improved significantly, leading to a better F1_Score. The increased recall means the model is now identifying more actual attrition cases. The ROC-AUC of 0.6592 indicates a **moderate ability to distinguish between classes, which is an improvement over Random Forest.**

Gradient Boost showed improvements in accuracy and precision after tuning, but a significant decrease in recall. This led to a lower F1_Score post-tuning. However, it achieved the highest ROC-AUC score of 0.6759, indicating **the best overall ability to distinguish between classes.**

Confusion Matrices:

- **Random Forest** tends to overpredict the majority class (non-attrition), missing many actual attrition cases.
- **AdaBoost** improves on this slightly, catching more attrition cases.
- **Gradient Boosting** shows the most balanced prediction, identifying the highest number of actual attrition cases, though at the cost of more false positives.

Best Overall Performing Model:

Considering all metrics, the AdaBoost model after hyperparameter tuning appears to be the best-performing model overall. It achieved the highest F1_Score (62.69%) and a good balance between precision and recall correctly predicting the largest amount of employees who were more likely to leave.

****