Predicting Employee Retention

Objective

The objective of this assignment is to develop a Logistic Regression model. You will be using this model to analyse and predict binary outcomes based on the input data. This assignment aims to enhance understanding of logistic regression, including its assumptions, implementation, and evaluation, to effectively classify and interpret data.

Business Objective

A mid-sized technology company wants to improve its understanding of employee retention to foster a loyal and committed workforce. While the organization has traditionally focused on addressing turnover, it recognises the value of proactively identifying employees likely to stay and understanding the factors contributing to their loyalty.

In this assignment you'll be building a logistic regression model to predict the likelihood of employee retention based on the data such as demographic details, job satisfaction scores, performance metrics, and tenure. The aim is to provide the HR department with actionable insights to strengthen retention strategies, create a supportive work environment, and increase the overall stability and satisfaction of the workforce.

Assignment Tasks

You need to perform the following steps to complete this assignment:

1. Data Understanding
2. Data Cleaning
3. Train Validation Split
4. EDA on training data
5. EDA on validation data [Optional]
6. Feature Engineering
7. Model Building
8. Prediction and Model Evaluation

Data Dictionary

The data has 24 Columns and 74610 Rows. Following data dictionary provides the description for each column present in dataset:

1. Data Understanding

In this step, load the dataset and check basic statistics of the data, including preview of data, dimension of data, column descriptions and data types.

1.0 Import Libraries

```
# Supress unnecessary warnings
import warnings
warnings.filterwarnings('ignore')

# Import the libraries
import numpy as np
import pandas as pd
```

1.1 Load the Data

```
# Load the dataset
emp_df1 = pd.read_csv("Employee_data.csv")

emp_df1.describe()
```

```
        Employee ID           Age  Years at Company  Monthly Income  \
count  74610.000000  74610.000000      74610.000000    74610.000000
mean   37246.028696     38.529379         15.722638     7344.931417
std    21505.785344     12.082299         11.224059     2596.373589
min        1.000000     18.000000          1.000000     1226.000000
25%    18624.250000     28.000000          7.000000     5652.000000
50%    37239.500000     39.000000         13.000000     7348.500000
75%    55871.750000     49.000000         23.000000     8876.000000
max    74498.000000     59.000000         51.000000    50030.000000

       Number of Promotions  Distance from Home  Number of Dependents
\
count          74610.000000        72698.000000          74610.000000

mean               0.832958           49.990839              1.657432

std                0.995326           28.519135              1.579862

min                0.000000            1.000000              0.000000

25%                0.000000           25.000000              0.000000

50%                1.000000           50.000000              1.000000

75%                2.000000           75.000000              3.000000

max                4.000000           99.000000             15.000000


       Company Tenure (In Months)
count                72197.000000
mean                    55.711899
std                     25.392325
min                      2.000000
```

```
25%                       36.000000
50%                       56.000000
75%                       76.000000
max                      128.000000

# Check the first few entries
emp_df1.head()

   Employee ID  Age  Gender  Years at Company    Job Role  Monthly
Income  \
0         8410   31    Male                19   Education
5390
1        64756   59  Female                 4       Media
5534
2        30257   24  Female                10  Healthcare
8159
3        65791   36  Female                 7   Education
3989
4        65026   56    Male                41   Education
4821

  Work-Life Balance Job Satisfaction Performance Rating  Number of
Promotions  \
0         Excellent           Medium            Average
2
1              Poor             High                Low
3
2              Good             High                Low
0
3              Good             High               High
1
4              Fair        Very High            Average
0

   ... Number of Dependents  Job Level Company Size  \
0  ...                    0        Mid       Medium
1  ...                    3        Mid       Medium
2  ...                    3        Mid       Medium
3  ...                    2        Mid        Small
4  ...                    0     Senior       Medium

  Company Tenure (In Months)  Remote Work Leadership Opportunities  \
0                       89.0           No                       No
1                       21.0           No                       No
2                       74.0           No                       No
3                       50.0          Yes                       No
4                       68.0           No                       No

  Innovation Opportunities  Company Reputation Employee Recognition
Attrition
```

```
0                        No          Excellent               Medium
Stayed
1                        No               Fair                  Low
Stayed
2                        No               Poor                  Low
Stayed
3                        No               Good               Medium
Stayed
4                        No               Fair               Medium
Stayed

[5 rows x 24 columns]
```

```
# Inspect the shape of the dataset
emp_df1.shape
```

```
(74610, 24)
```

```
# Inspect the different columns in the dataset
emp_df1.columns
```

```
Index(['Employee ID', 'Age', 'Gender', 'Years at Company', 'Job Role',
       'Monthly Income', 'Work-Life Balance', 'Job Satisfaction',
       'Performance Rating', 'Number of Promotions', 'Overtime',
       'Distance from Home', 'Education Level', 'Marital Status',
       'Number of Dependents', 'Job Level', 'Company Size',
       'Company Tenure (In Months)', 'Remote Work', 'Leadership
Opportunities',
       'Innovation Opportunities', 'Company Reputation',
       'Employee Recognition', 'Attrition'],
      dtype='object')
```

1.2 Check the basic statistics

```
emp_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74610 entries, 0 to 74609
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   Employee ID                 74610 non-null  int64 
 1   Age                         74610 non-null  int64 
 2   Gender                      74610 non-null  object
 3   Years at Company            74610 non-null  int64 
 4   Job Role                    74610 non-null  object
 5   Monthly Income              74610 non-null  int64 
 6   Work-Life Balance           74610 non-null  object
 7   Job Satisfaction            74610 non-null  object
 8   Performance Rating          74610 non-null  object
```

```
 9   Number of Promotions        74610 non-null  int64  
 10  Overtime                    74610 non-null  object 
 11  Distance from Home          72698 non-null  float64
 12  Education Level             74610 non-null  object 
 13  Marital Status              74610 non-null  object 
 14  Number of Dependents        74610 non-null  int64  
 15  Job Level                   74610 non-null  object 
 16  Company Size                74610 non-null  object 
 17  Company Tenure (In Months)  72197 non-null  float64
 18  Remote Work                 74610 non-null  object 
 19  Leadership Opportunities    74610 non-null  object 
 20  Innovation Opportunities    74610 non-null  object 
 21  Company Reputation          74610 non-null  object 
 22  Employee Recognition        74610 non-null  object 
 23  Attrition                   74610 non-null  object 
dtypes: float64(2), int64(6), object(16)
memory usage: 13.7+ MB
```

1.3 Check the data type of columns

```
# Check the info to see the types of the feature variables and the
null values present
emp_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74610 entries, 0 to 74609
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Employee ID                 74610 non-null  int64  
 1   Age                         74610 non-null  int64  
 2   Gender                      74610 non-null  object 
 3   Years at Company            74610 non-null  int64  
 4   Job Role                    74610 non-null  object 
 5   Monthly Income              74610 non-null  int64  
 6   Work-Life Balance           74610 non-null  object 
 7   Job Satisfaction            74610 non-null  object 
 8   Performance Rating          74610 non-null  object 
 9   Number of Promotions        74610 non-null  int64  
 10  Overtime                    74610 non-null  object 
 11  Distance from Home          72698 non-null  float64
 12  Education Level             74610 non-null  object 
 13  Marital Status              74610 non-null  object 
 14  Number of Dependents        74610 non-null  int64  
 15  Job Level                   74610 non-null  object 
 16  Company Size                74610 non-null  object 
 17  Company Tenure (In Months)  72197 non-null  float64
 18  Remote Work                 74610 non-null  object 
 19  Leadership Opportunities    74610 non-null  object 
 20  Innovation Opportunities    74610 non-null  object 
```

```
 21  Company Reputation          74610 non-null  object
 22  Employee Recognition        74610 non-null  object
 23  Attrition                   74610 non-null  object
dtypes: float64(2), int64(6), object(16)
memory usage: 13.7+ MB
```

2. Data Cleaning [15 marks]

2.1 Handle the missing values [10 marks]

2.1.1 Check the number of missing values [2 Mark]

```python
# Check the number of missing values in each column
emp_df1.isnull().sum()
```

```
Employee ID                     0
Age                             0
Gender                          0
Years at Company                0
Job Role                        0
Monthly Income                  0
Work-Life Balance               0
Job Satisfaction                0
Performance Rating              0
Number of Promotions            0
Overtime                        0
Distance from Home           1912
Education Level                 0
Marital Status                  0
Number of Dependents            0
Job Level                       0
Company Size                    0
Company Tenure (In Months)   2413
Remote Work                     0
Leadership Opportunities        0
Innovation Opportunities        0
Company Reputation              0
Employee Recognition            0
Attrition                       0
dtype: int64
```

2.1.2 Check the percentage of missing values [2 Marks]

```python
# Check the percentage of missing values in each column
missing_percentage = (emp_df1.isnull().sum() / len(emp_df1)) * 100

missing_percentage
```

```
Employee ID                        0.000000
Age                                0.000000
Gender                             0.000000
Years at Company                   0.000000
Job Role                           0.000000
Monthly Income                     0.000000
Work-Life Balance                  0.000000
Job Satisfaction                   0.000000
Performance Rating                 0.000000
Number of Promotions               0.000000
Overtime                           0.000000
Distance from Home                 2.562659
Education Level                    0.000000
Marital Status                     0.000000
Number of Dependents               0.000000
Job Level                          0.000000
Company Size                       0.000000
Company Tenure (In Months)         3.234151
Remote Work                        0.000000
Leadership Opportunities           0.000000
Innovation Opportunities           0.000000
Company Reputation                 0.000000
Employee Recognition               0.000000
Attrition                          0.000000
dtype: float64
```

2.1.3 Handle rows with missing values [4 Marks]

```
# Handle the missing value rows in the column
emp_df2 = emp_df1.dropna()

emp_df2.shape

(70635, 24)
```

2.1.4 Check percentage of remaning data after missing values are removed [2 Mark]

```
remaining_emp_percentage = (len(emp_df2) / len(emp_df1)) * 100

remaining_emp_percentage

94.67229593888219
```

2.2 Identify and handle redundant values within categorical columns (if any) [3 marks]

Examine the categorical columns to determine if any value or column needs to be treated

```python
# Write a function to display the categorical columns with their
unique values and check for redundant values
def display_categorical_columns(emp_df):
    categorical_columns =
emp_df.select_dtypes(include=['object']).columns  # Filter categorical
columns
    for column in categorical_columns:
        unique_values = emp_df[column].unique()  # Get unique values
        print(f"Column: {column}")
        print(f"Unique Values: {unique_values}")
        print(f"Number of Unique Values: {len(unique_values)}")
        print("-" * 50)

# Check the data
display_categorical_columns(emp_df2)
```

```
Column: Gender
Unique Values: ['Male' 'Female']
Number of Unique Values: 2
--------------------------------------------------
Column: Job Role
Unique Values: ['Education' 'Media' 'Healthcare' 'Technology'
'Finance']
Number of Unique Values: 5
--------------------------------------------------
Column: Work-Life Balance
Unique Values: ['Excellent' 'Poor' 'Good' 'Fair']
Number of Unique Values: 4
--------------------------------------------------
Column: Job Satisfaction
Unique Values: ['Medium' 'High' 'Very High' 'Low']
Number of Unique Values: 4
--------------------------------------------------
Column: Performance Rating
Unique Values: ['Average' 'Low' 'High' 'Below Average']
Number of Unique Values: 4
--------------------------------------------------
Column: Overtime
Unique Values: ['No' 'Yes']
Number of Unique Values: 2
--------------------------------------------------
Column: Education Level
Unique Values: ['Associate Degree' 'Masterâ€™s Degree' 'Bachelorâ€™s
Degree'
 'High School' 'PhD']
Number of Unique Values: 5
--------------------------------------------------
Column: Marital Status
Unique Values: ['Married' 'Divorced' 'Single']
Number of Unique Values: 3
```

```
----------------------------------------------------
Column: Job Level
Unique Values: ['Mid' 'Senior' 'Entry']
Number of Unique Values: 3
----------------------------------------------------
Column: Company Size
Unique Values: ['Medium' 'Small' 'Large']
Number of Unique Values: 3
----------------------------------------------------
Column: Remote Work
Unique Values: ['No' 'Yes']
Number of Unique Values: 2
----------------------------------------------------
Column: Leadership Opportunities
Unique Values: ['No' 'Yes']
Number of Unique Values: 2
----------------------------------------------------
Column: Innovation Opportunities
Unique Values: ['No' 'Yes']
Number of Unique Values: 2
----------------------------------------------------
Column: Company Reputation
Unique Values: ['Excellent' 'Fair' 'Poor' 'Good']
Number of Unique Values: 4
----------------------------------------------------
Column: Employee Recognition
Unique Values: ['Medium' 'Low' 'High' 'Very High']
Number of Unique Values: 4
----------------------------------------------------
Column: Attrition
Unique Values: ['Stayed' 'Left']
Number of Unique Values: 2
----------------------------------------------------
```

2.3 Drop redundant columns [2 marks]

```
# Drop redundant columns which are not required for modelling
columns_to_drop = ['Employee ID', 'Company Tenure (In Months)']
emp_df3 = emp_df2.drop(columns=columns_to_drop, axis=1)

# Check first few rows of data
emp_df3.head()
```

```
   Age  Gender  Years at Company    Job Role  Monthly Income  \
0   31    Male                19   Education            5390
1   59  Female                 4       Media            5534
2   24  Female                10  Healthcare            8159
3   36  Female                 7   Education            3989
4   56    Male                41   Education            4821
```

```
  Work-Life Balance Job Satisfaction Performance Rating  Number of 
Promotions  \
0         Excellent           Medium            Average                     
2   
1              Poor             High                Low                     
3   
2              Good             High                Low                     
0   
3              Good             High               High                     
1   
4              Fair        Very High            Average                     
0   

  Overtime  ...  Marital Status Number of Dependents Job Level 
Company Size  \
0       No  ...         Married                    0       Mid       
Medium   
1       No  ...        Divorced                    3       Mid       
Medium   
2       No  ...         Married                    3       Mid       
Medium   
3       No  ...          Single                    2       Mid        
Small   
4      Yes  ...        Divorced                    0    Senior       
Medium   

  Remote Work Leadership Opportunities Innovation Opportunities  \
0          No                       No                       No   
1          No                       No                       No   
2          No                       No                       No   
3         Yes                       No                       No   
4          No                       No                       No   

  Company Reputation Employee Recognition Attrition  
0          Excellent               Medium    Stayed  
1               Fair                  Low    Stayed  
2               Poor                  Low    Stayed  
3               Good               Medium    Stayed  
4               Fair               Medium    Stayed  

[5 rows x 22 columns]
```

3. Train-Validation Split [5 marks]

3.1 Import required libraries

```python
# Import Train Test Split
from sklearn.model_selection import train_test_split
```

3.2 Define feature and target variables [2 Mark]

```
# Put all the feature variables in X
X = emp_df3.drop(columns=['Attrition'])
# Put the target variable in y
y = emp_df3['Attrition']
```

3.3 Split the data [3 Marks]

```
# Split the data into 70% train data and 30% validation data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
random_state=42)

# Print the shapes of the resulting datasets
print("Training Features Shape:", X_train.shape)
print("Validation Features Shape:", X_val.shape)
print("Training Labels Shape:", y_train.shape)
print("Validation Labels Shape:", y_val.shape)
```

```
Training Features Shape: (49444, 21)
Validation Features Shape: (21191, 21)
Training Labels Shape: (49444,)
Validation Labels Shape: (21191,)
```

4. EDA on training data [20 marks]

4.1 Perform univariate analysis [6 marks]

Perform univariate analysis on training data for all the numerical columns.

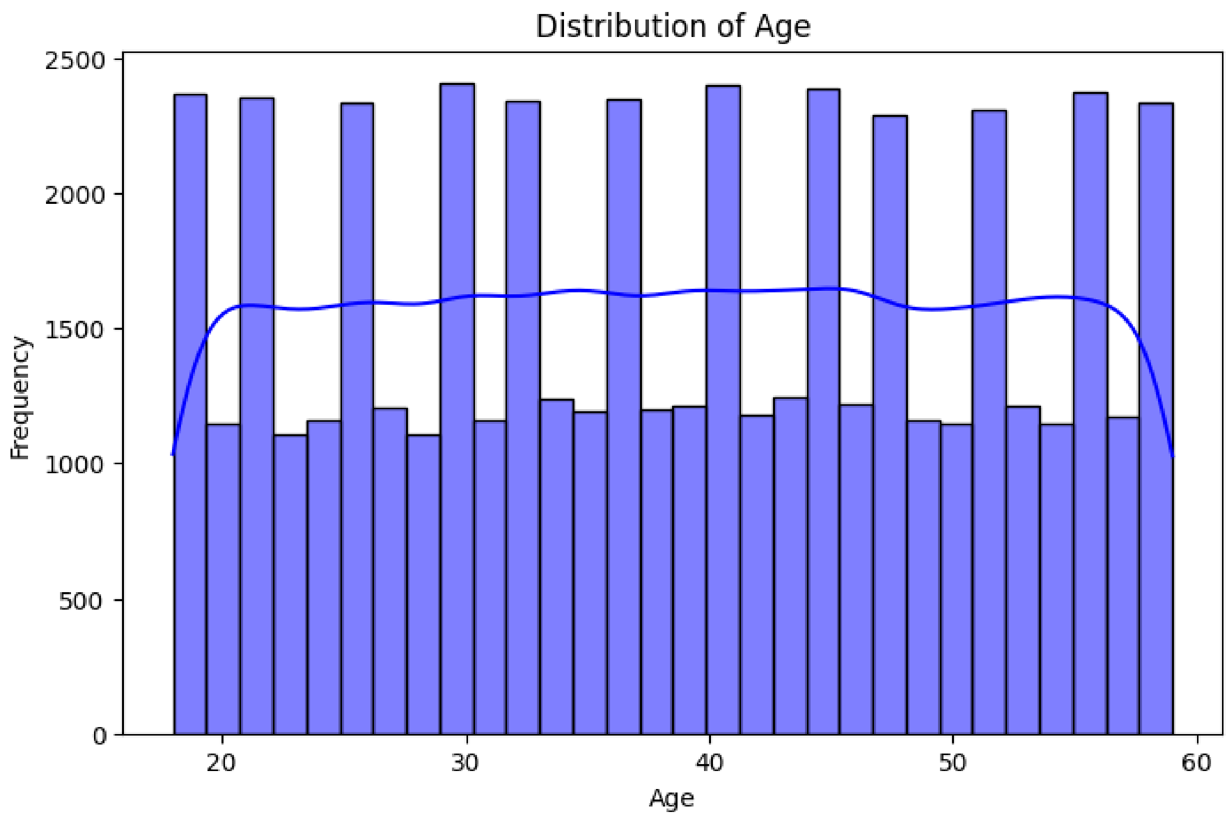
4.1.1 Select numerical columns from training data [1 Mark]

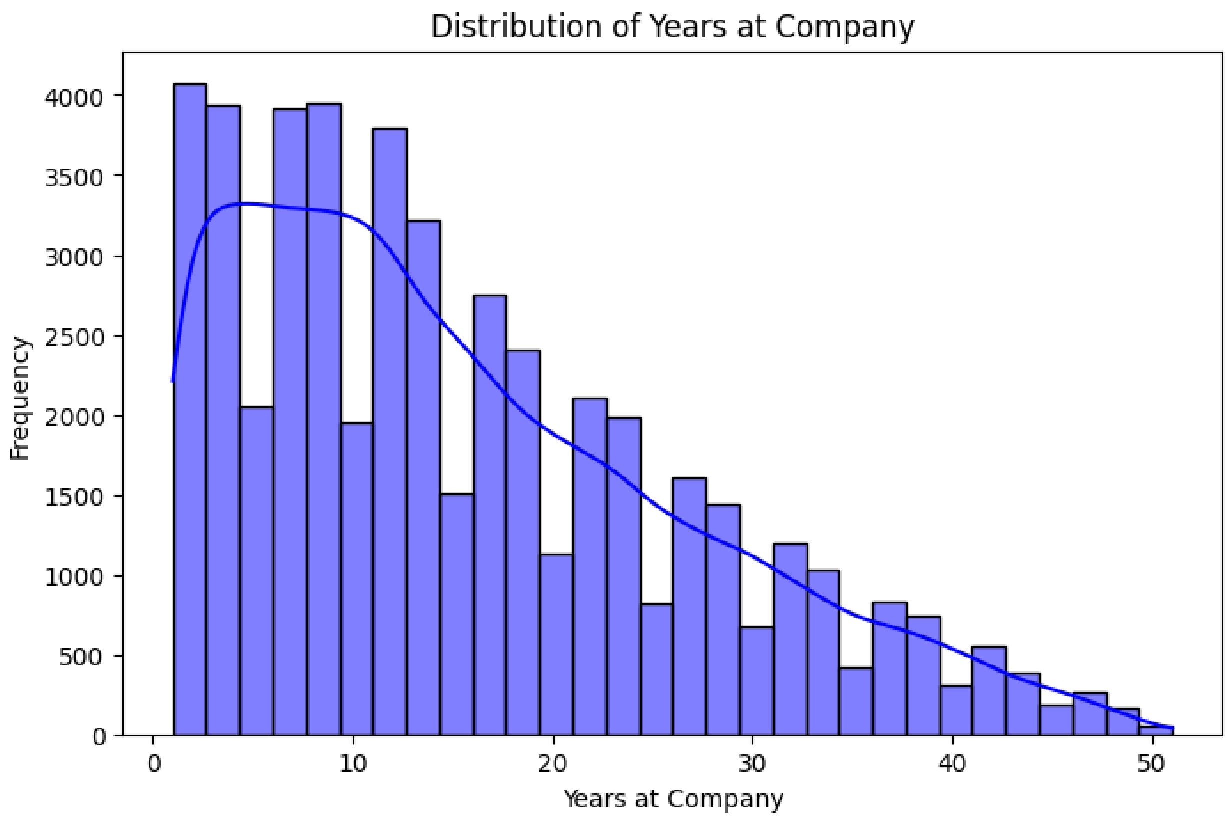
```
# Select numerical columns
numerical_columns = X_train.select_dtypes(include=['int64',
'float64']).columns
```

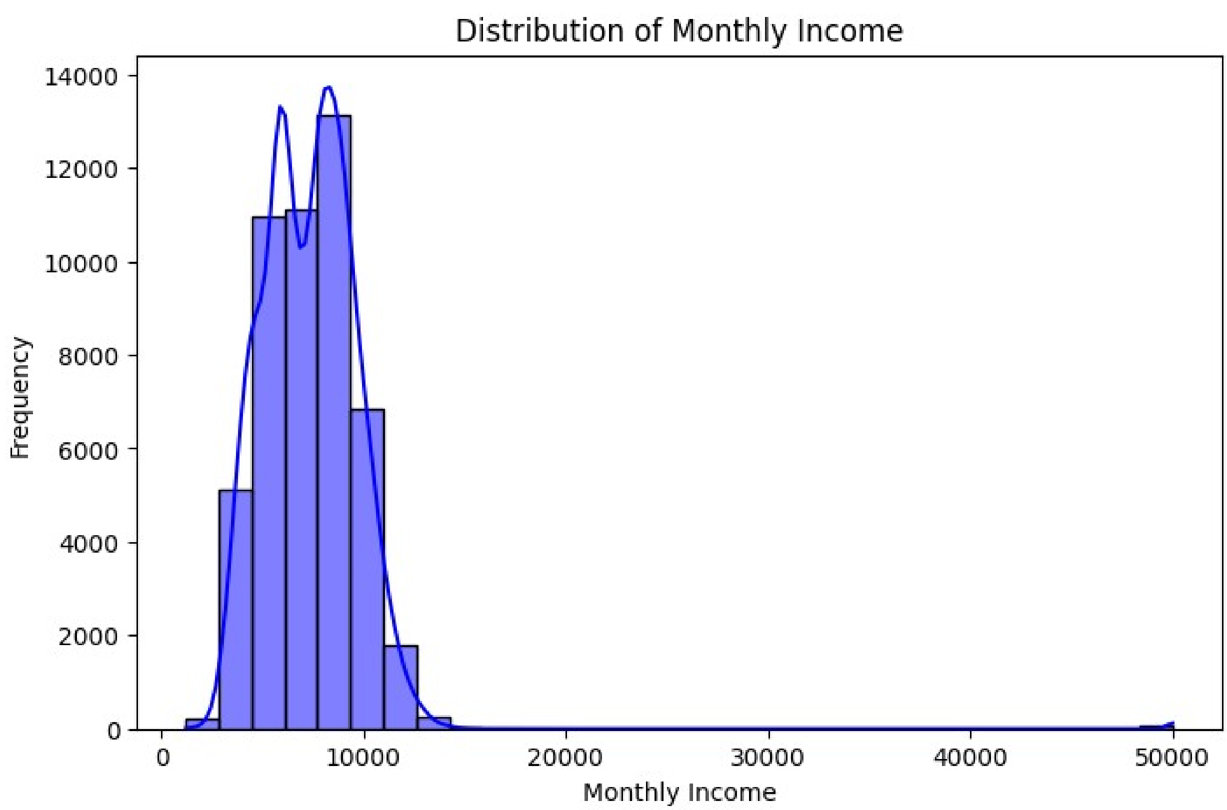
4.1.2 Plot distribution of numerical columns [5 Marks]

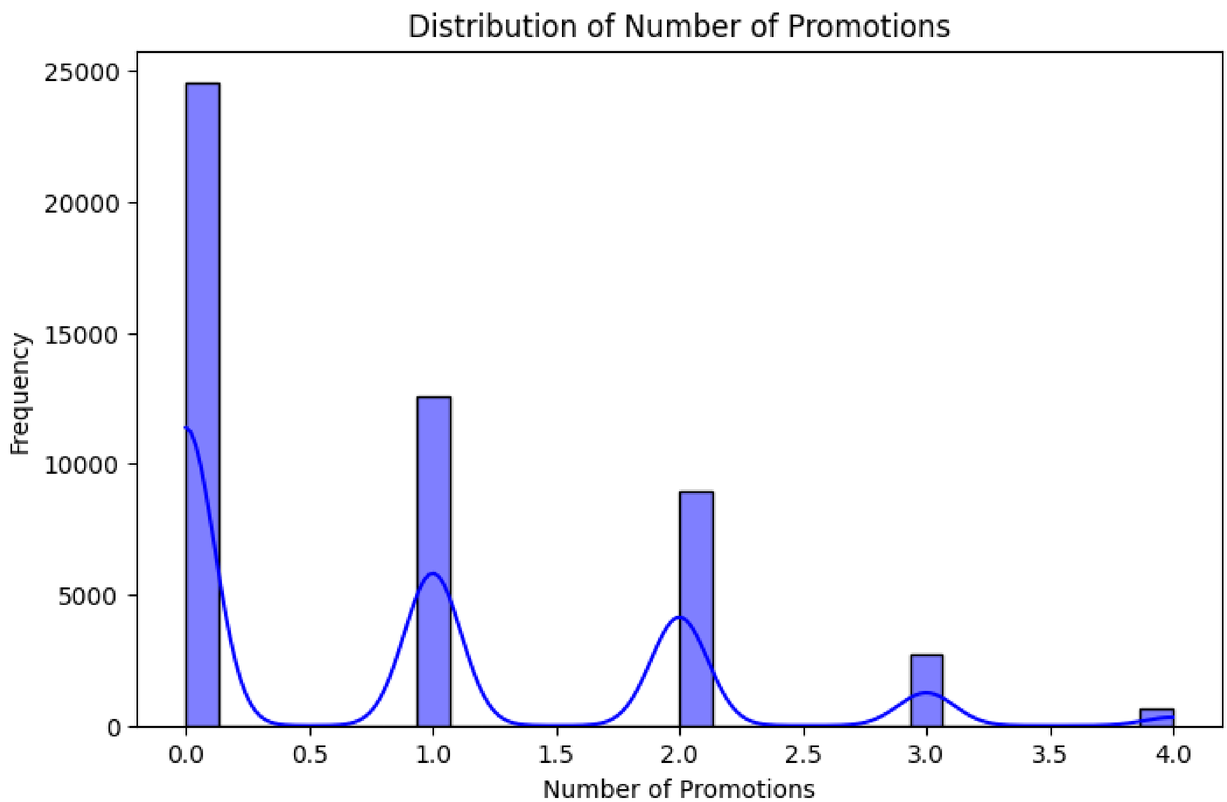
```
# Plot all the numerical columns to understand their distribution

# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Select numerical columns
numerical_columns = X_train.select_dtypes(include=['int64',
'float64']).columns

# Plot all the numerical columns to understand their distribution
for column in numerical_columns:
```

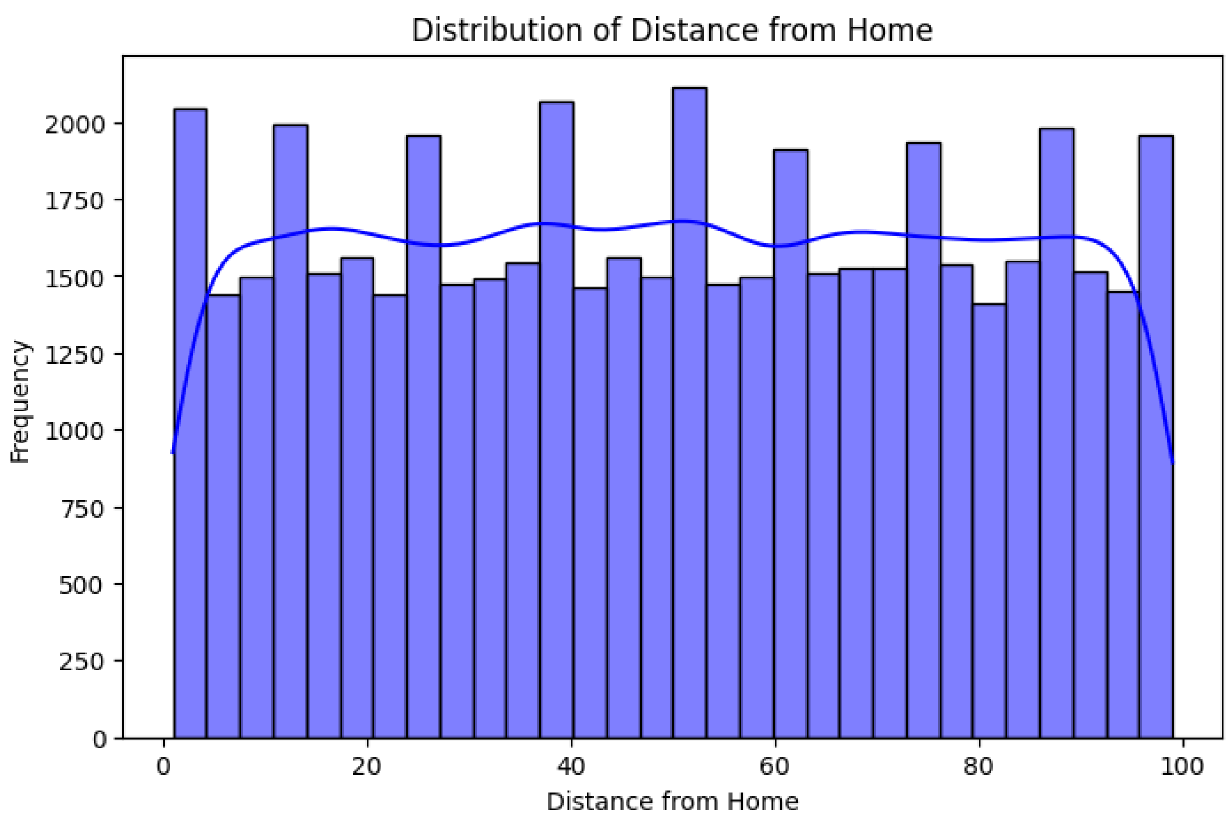
```
    plt.figure(figsize=(8, 5))
    sns.histplot(X_train[column], kde=True, bins=30, color='blue')  #
Histogram with a KDE overlay
    plt.title(f"Distribution of {column}")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()
```

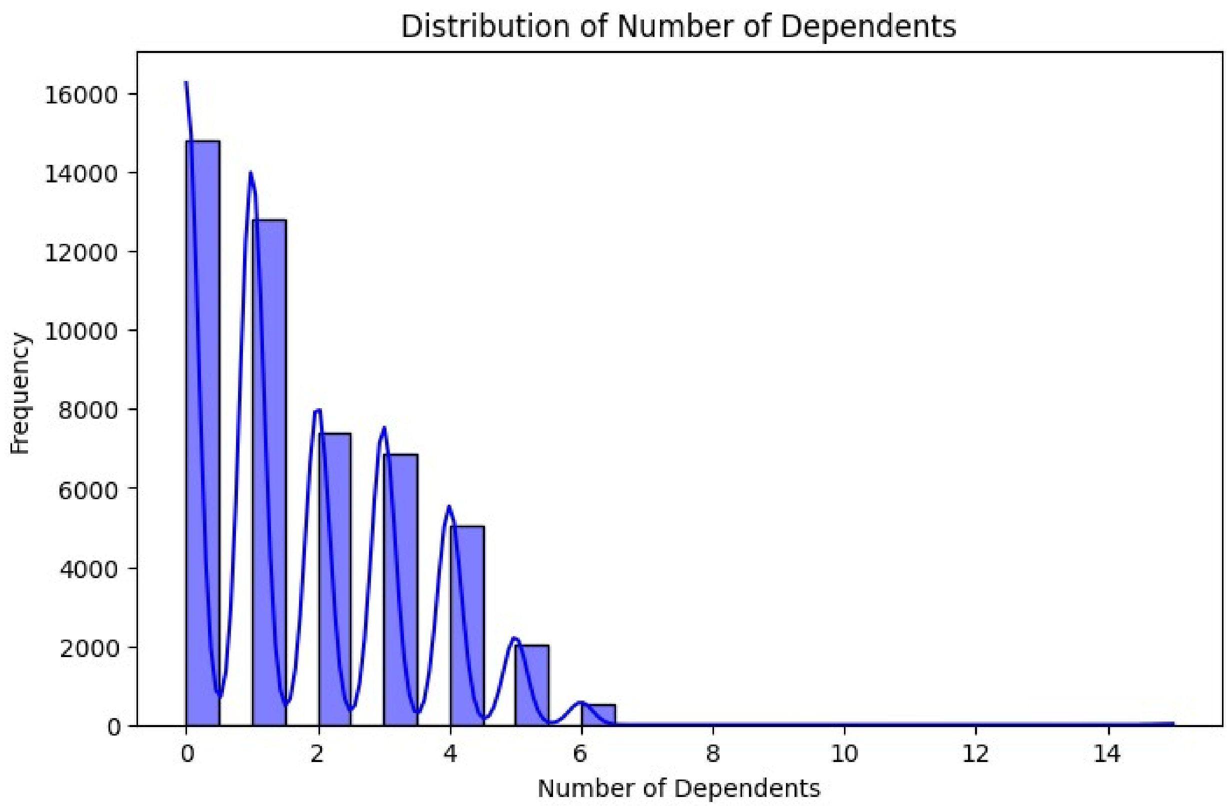



Distribution of Years at Company
Frequency
Years at Company
0
500
1000
1500
2000
2500
3000
3500
4000
10
20
30
40
50


Distribution of Monthly Income
Frequency
Monthly Income
0
2000
4000
6000
8000
10000
12000
14000
20000
30000
40000
50000


Distribution of Number of Promotions
Frequency
Number of Promotions
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

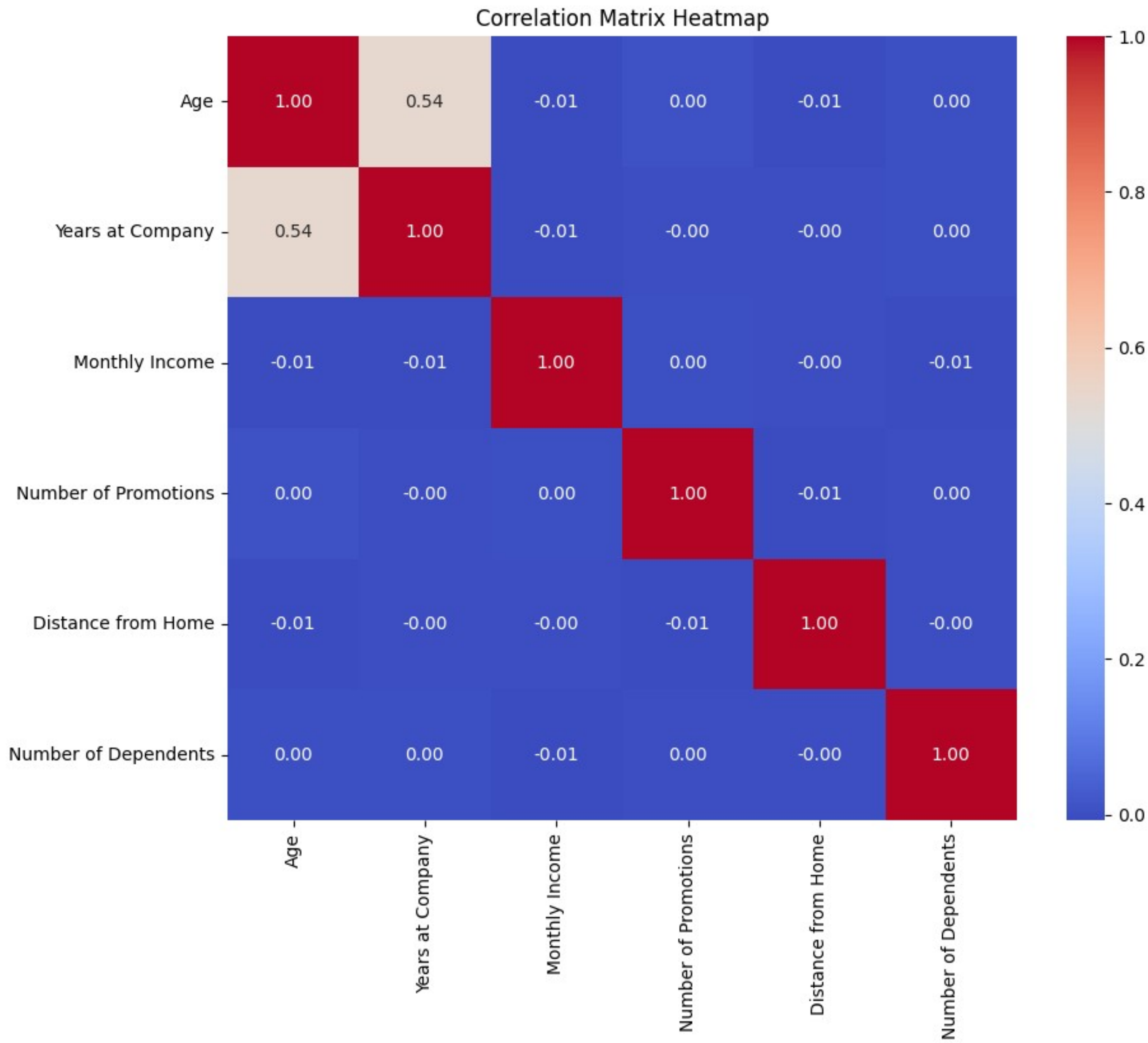

Distribution of Distance from Home
Frequency
Distance from Home
2000
1750
1500
1250
1000
750
500
250
0
0
20
40
60
80
100



4.2 Perform correlation analysis [4 Marks]

Check the correlation among different numerical variables.

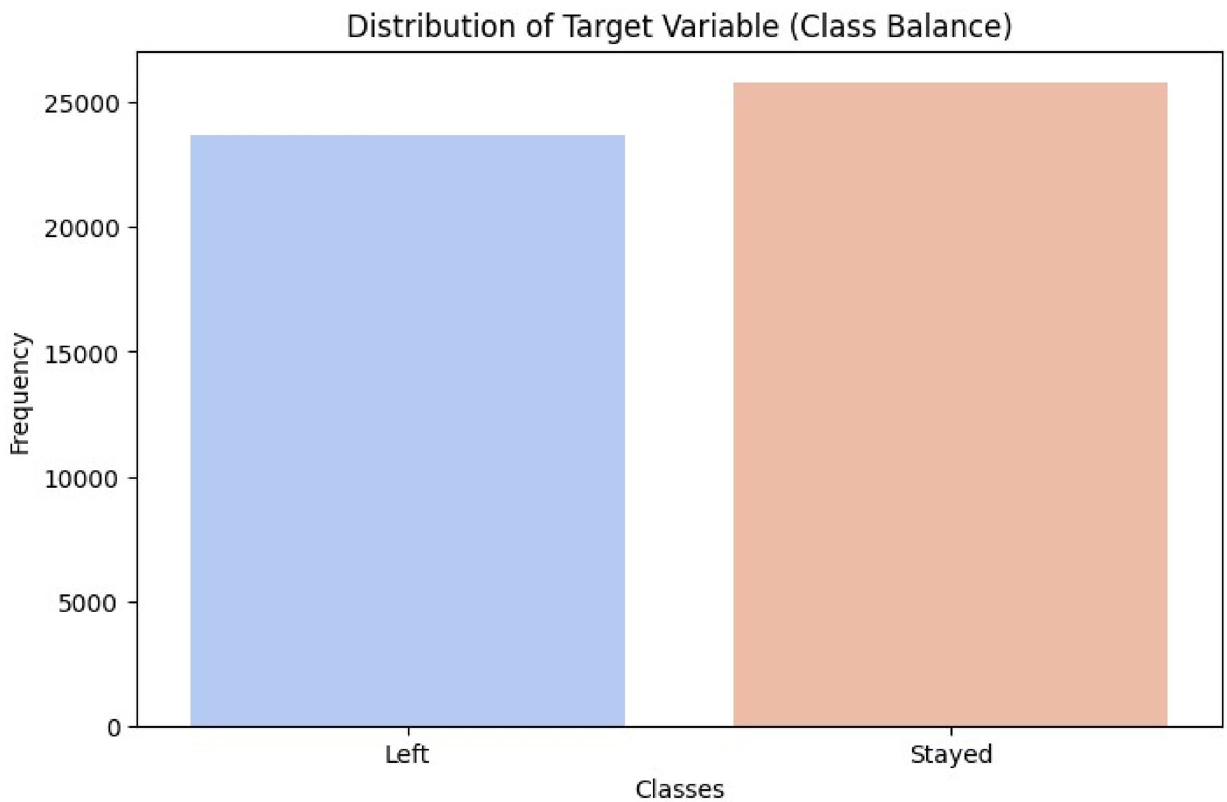
```
# Create correlation matrix for numerical columns
correlation_matrix = X_train[numerical_columns].corr()

# Plot Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.show()
```



4.3 Check class balance [2 Marks]

Check the distribution of target variable in training set to check class balance.

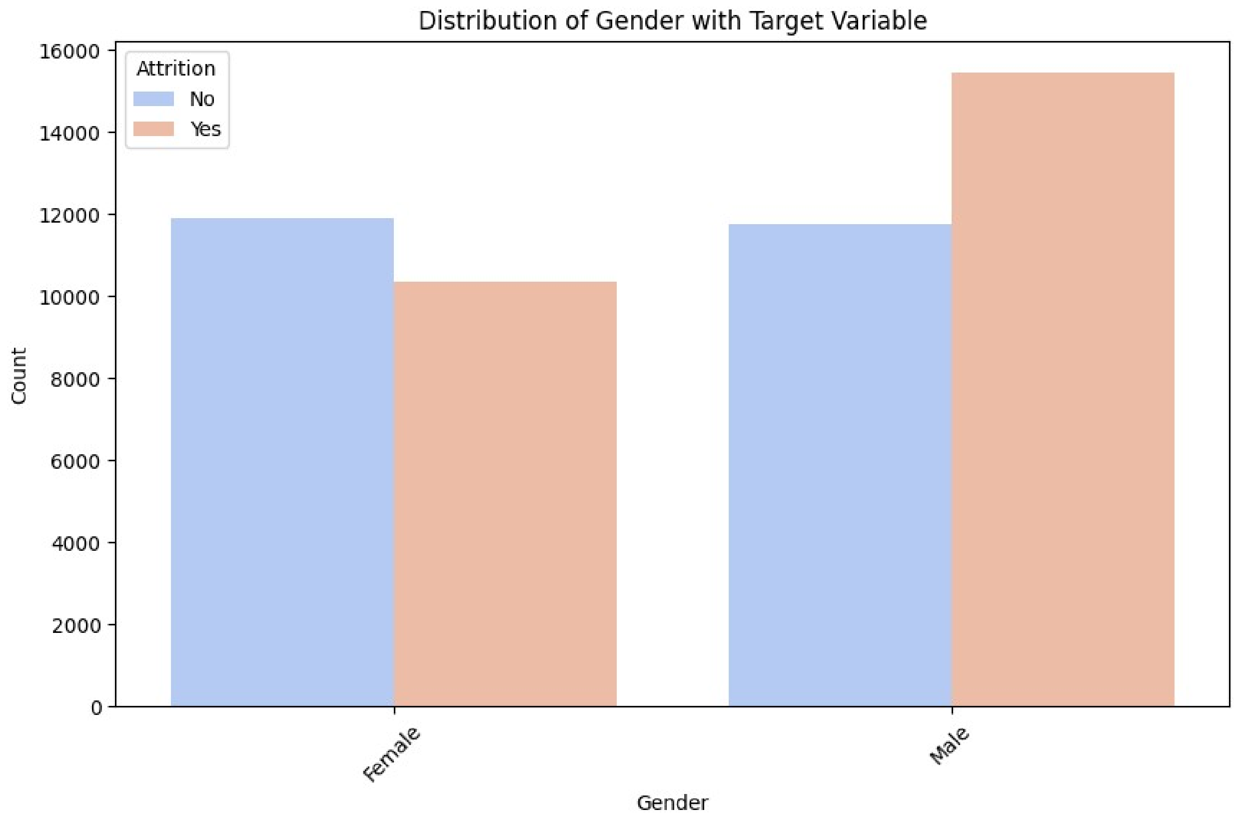
```
# Plot a bar chart to check class balance
plt.figure(figsize=(8, 5))
sns.countplot(x=y_train, palette="coolwarm")
plt.title("Distribution of Target Variable (Class Balance)")
plt.xlabel("Classes")
plt.ylabel("Frequency")
plt.show()
```

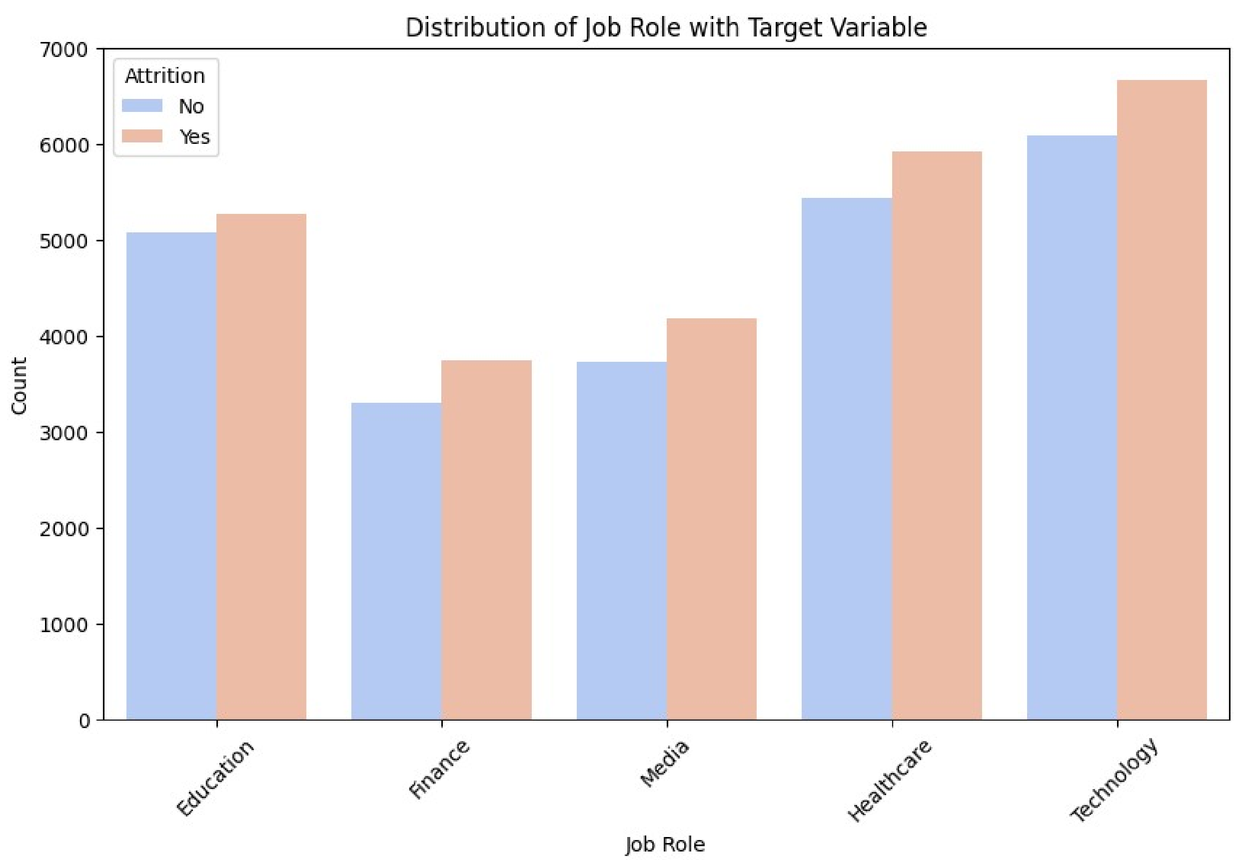


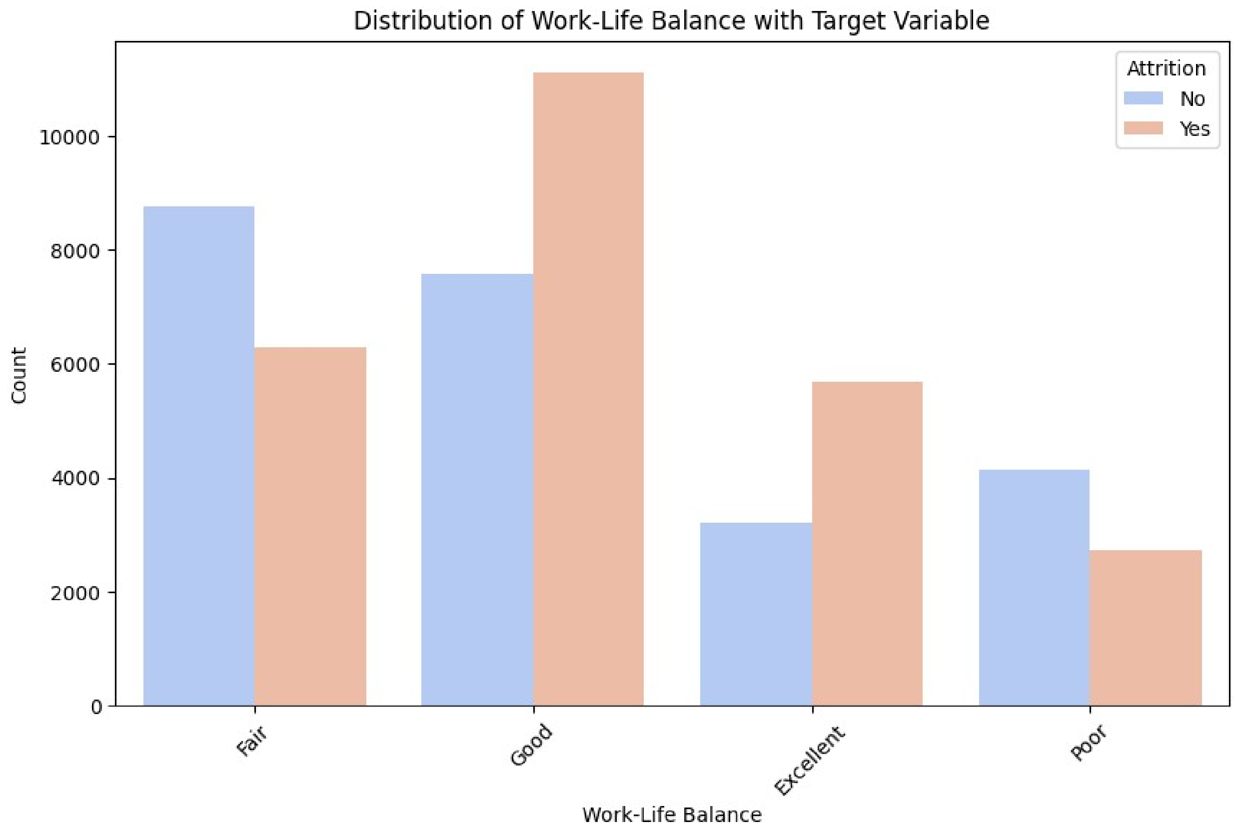
4.4 Perform bivariate analysis [8 Marks]

Perform bivariate analysis on training data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

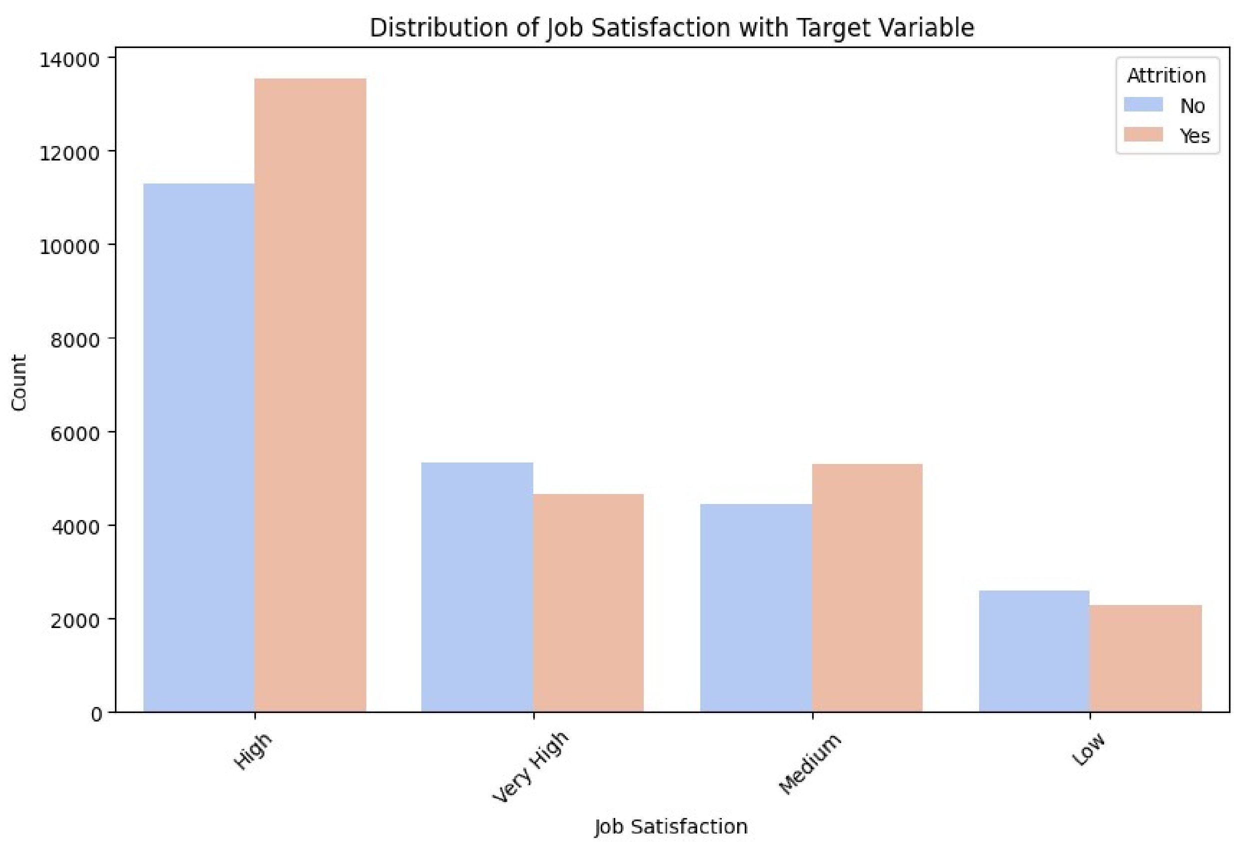
```
# Plot distribution for each categorical column with target variable
# Select categorical columns from the training data
categorical_columns =
X_train.select_dtypes(include=['object']).columns

# Plot the distribution for each categorical column with the target
variable
for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=X_train[column], hue=y_train, palette="coolwarm")
    plt.title(f"Distribution of {column} with Target Variable")
    plt.xlabel(column)
    plt.ylabel("Count")
    plt.legend(title="Attrition", labels=["No", "Yes"])
    plt.xticks(rotation=45)
    plt.show()
```


Distribution of Gender with Target Variable
Attrition
No
Yes
Count
Female
Male
Gender
0
2000
4000
6000
8000
10000
12000
14000
16000

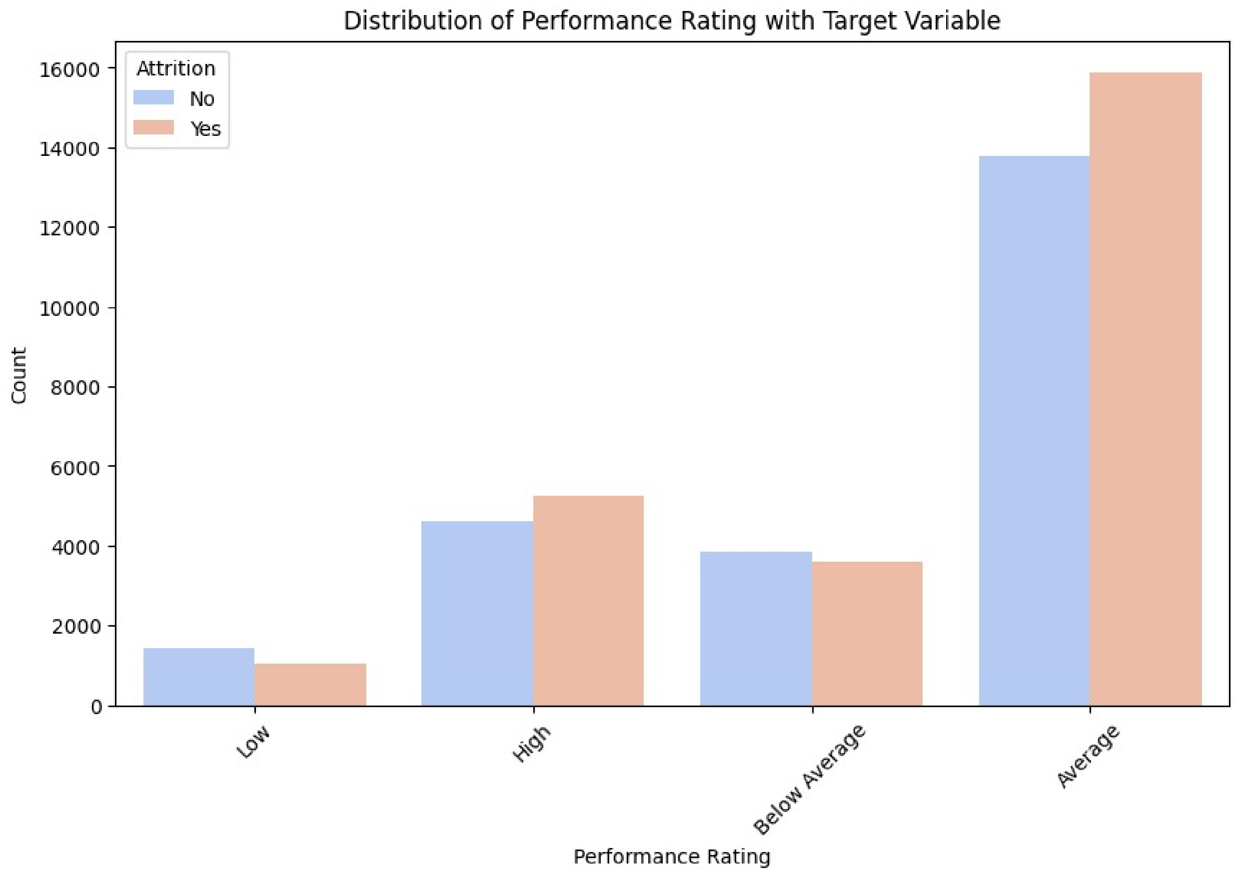

Distribution of Job Role with Target Variable
Attrition
No
Yes
Count
Education
Finance
Media
Healthcare
Technology
Job Role
0
1000
2000
3000
4000
5000
6000
7000

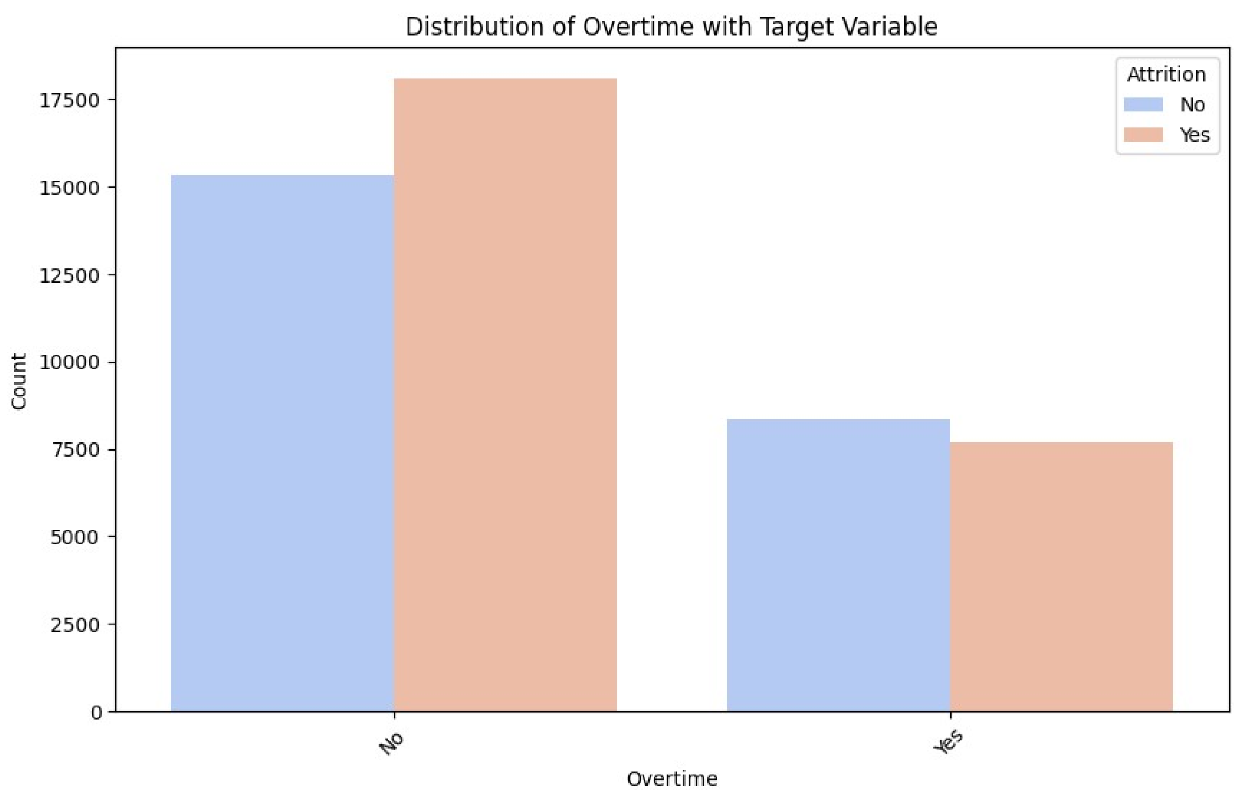


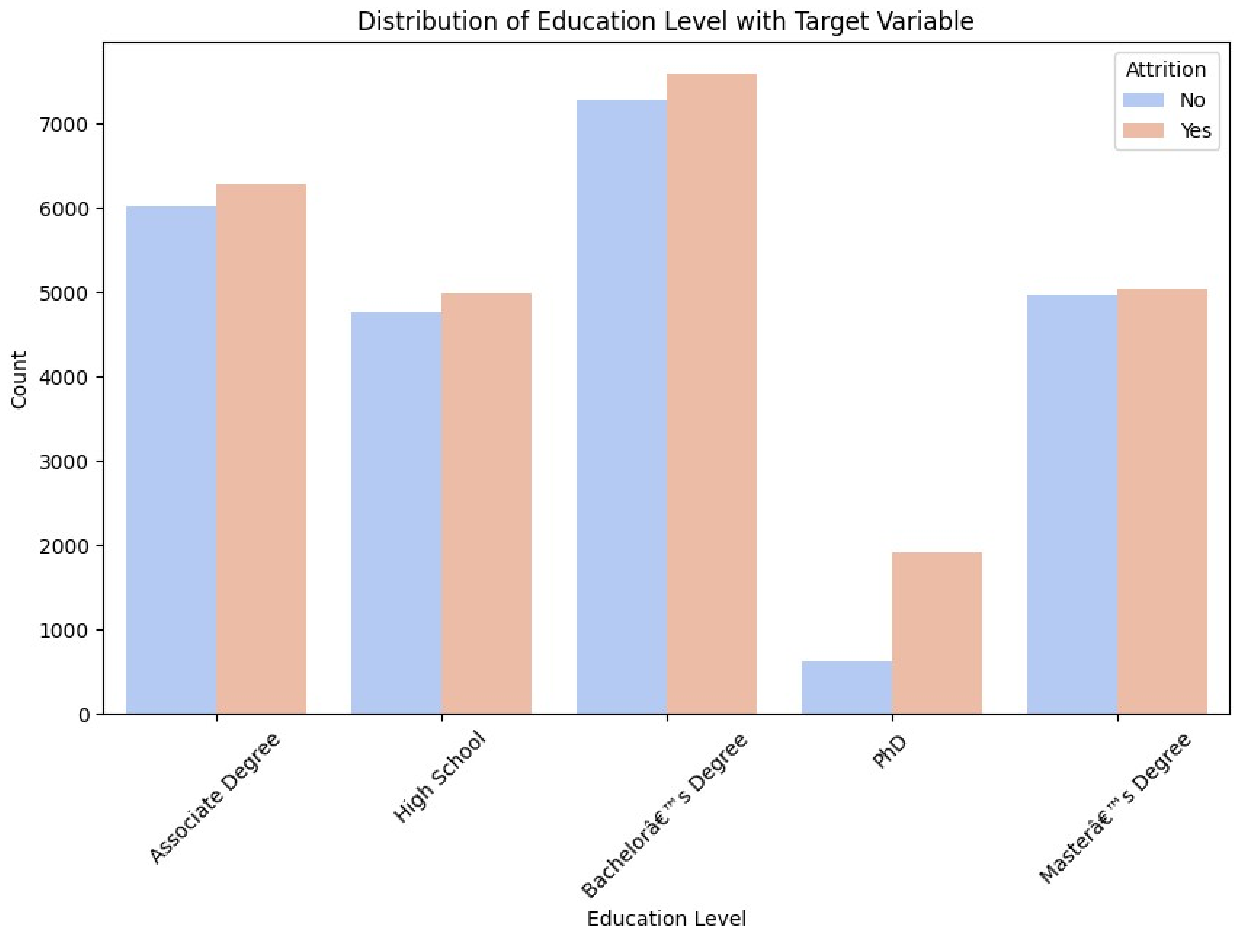
Distribution of Work-Life Balance with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
Fair
Good
Excellent
Poor
Work-Life Balance



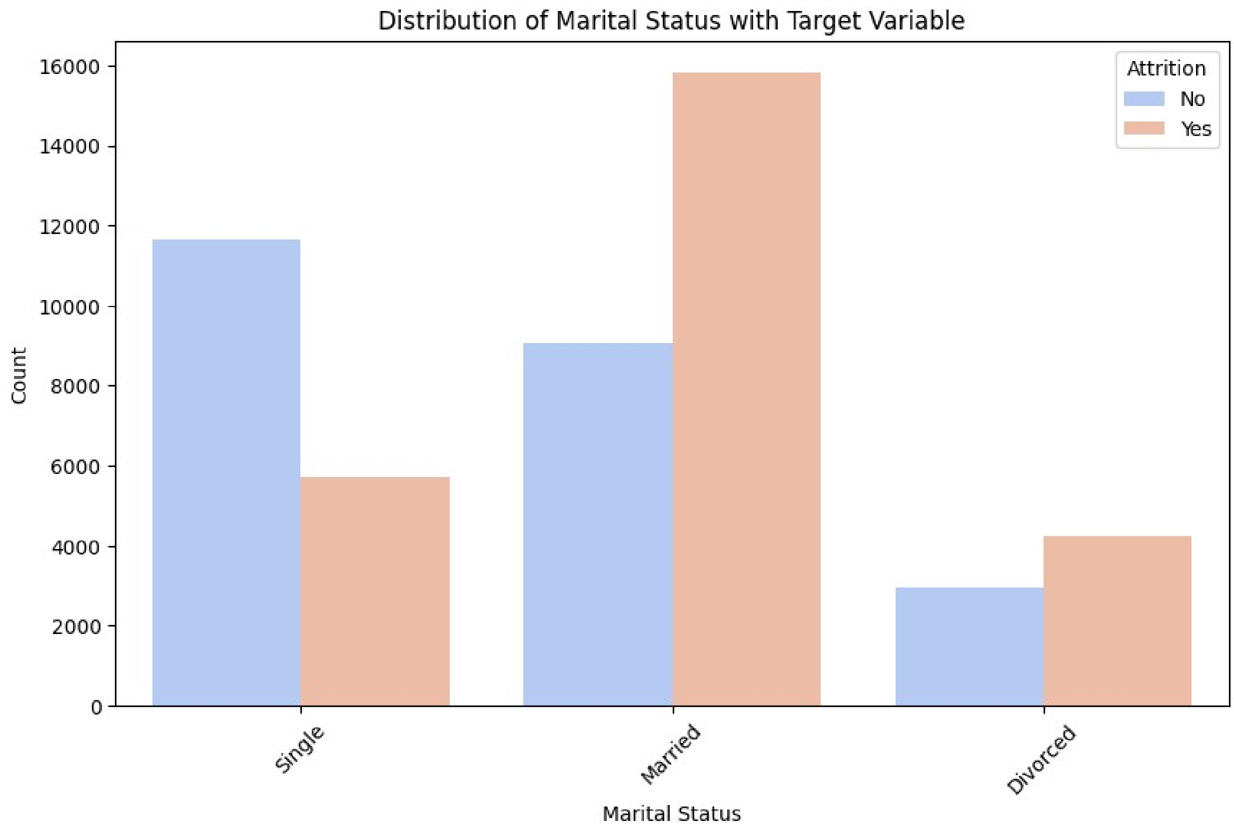
Distribution of Job Satisfaction with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
12000
14000
High
Very High
Medium
Low
Job Satisfaction

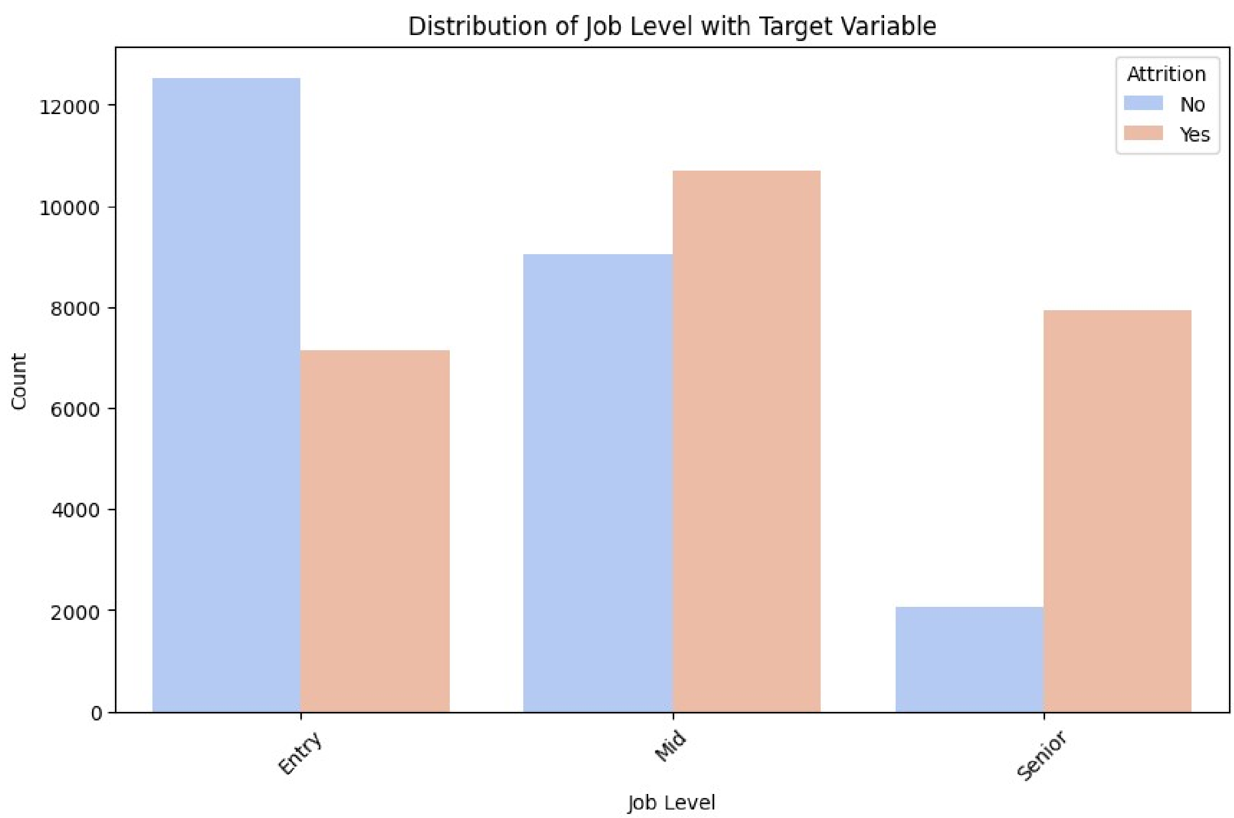

Distribution of Performance Rating with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
12000
14000
16000
Low
High
Below Average
Average
Performance Rating

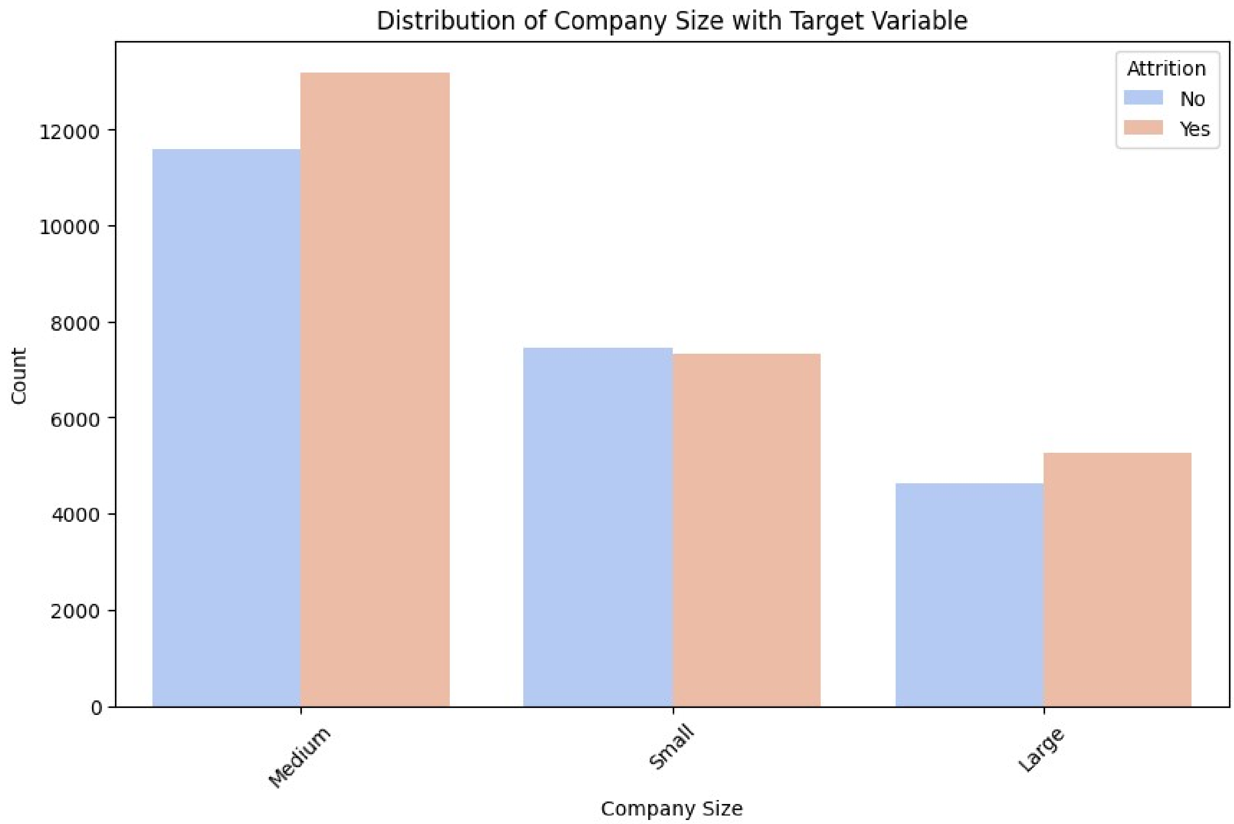

Distribution of Overtime with Target Variable
Attrition
No
Yes
Count
0
2500
5000
7500
10000
12500
15000
17500
No
Yes
Overtime



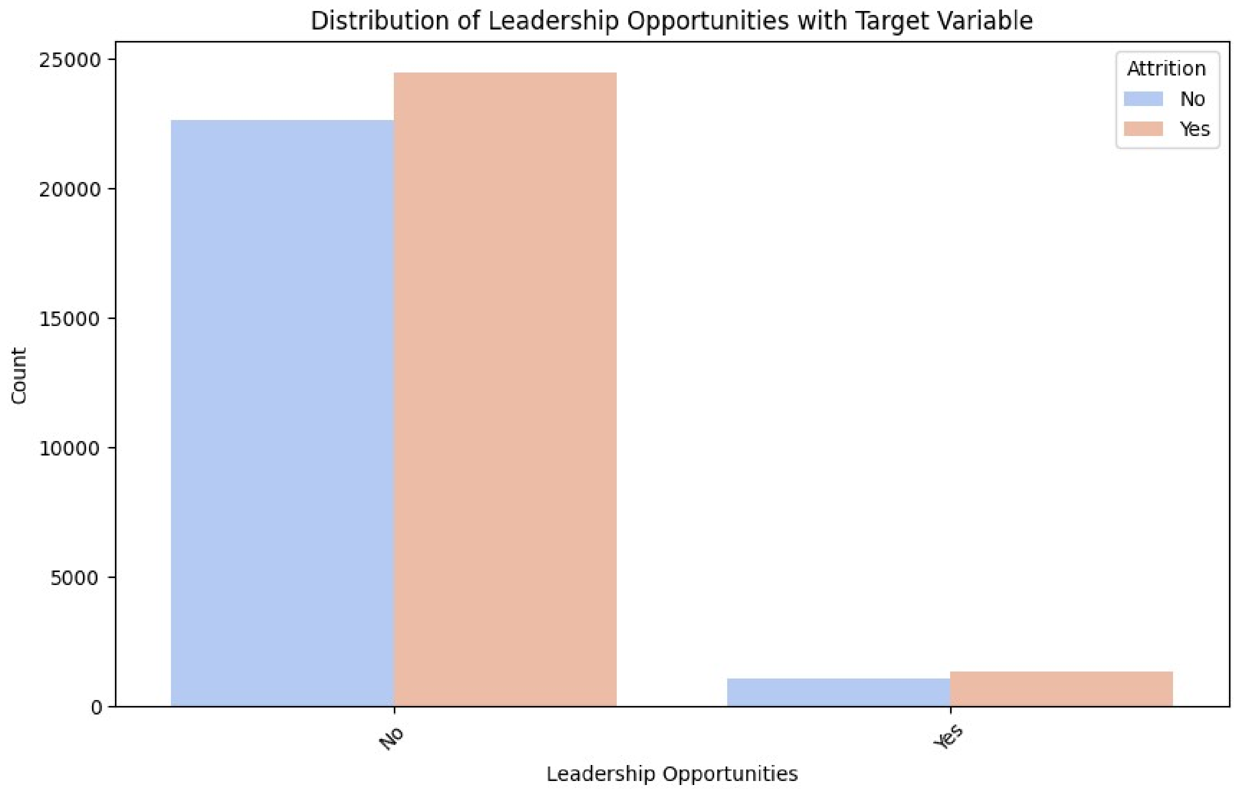
Distribution of Education Level with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
5000
6000
7000
Associate Degree
High School
Bachelorâ€™s Degree
PhD
Masterâ€™s Degree
Education Level

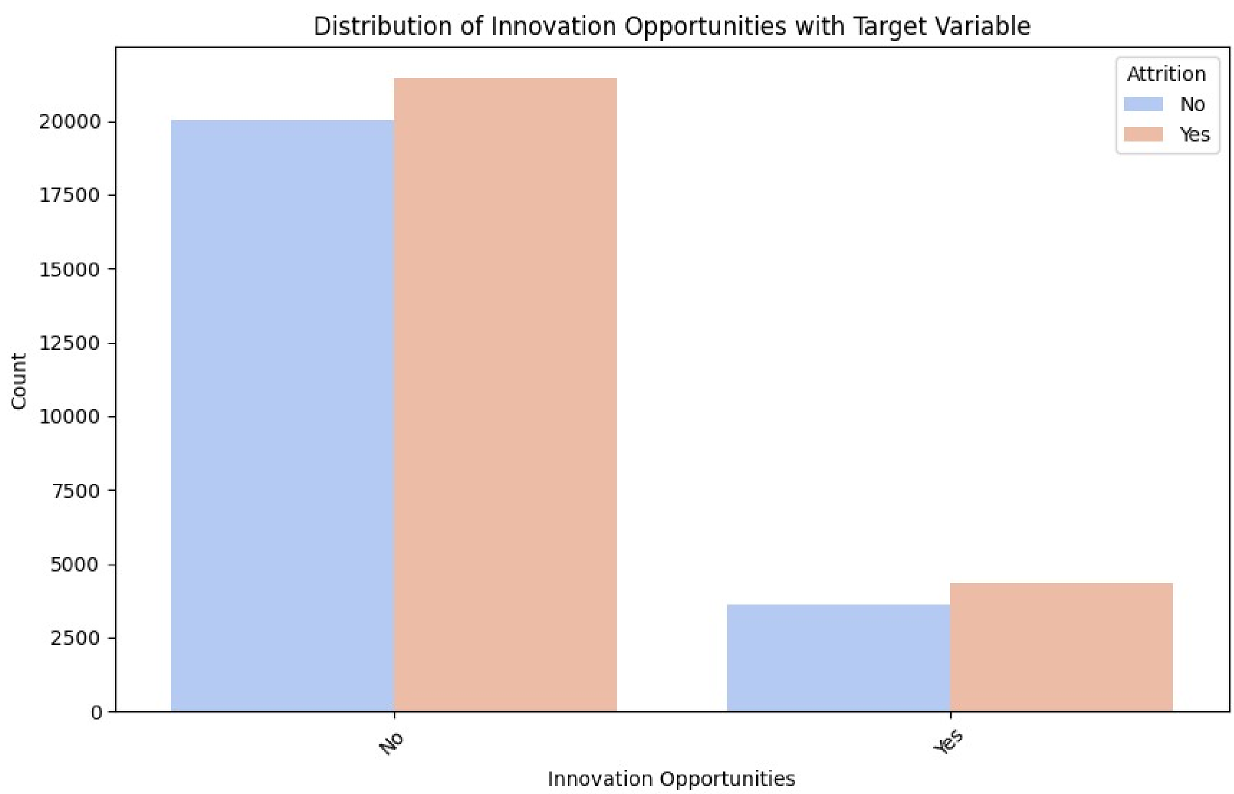

Distribution of Marital Status with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
12000
14000
16000
Single
Married
Divorced
Marital Status

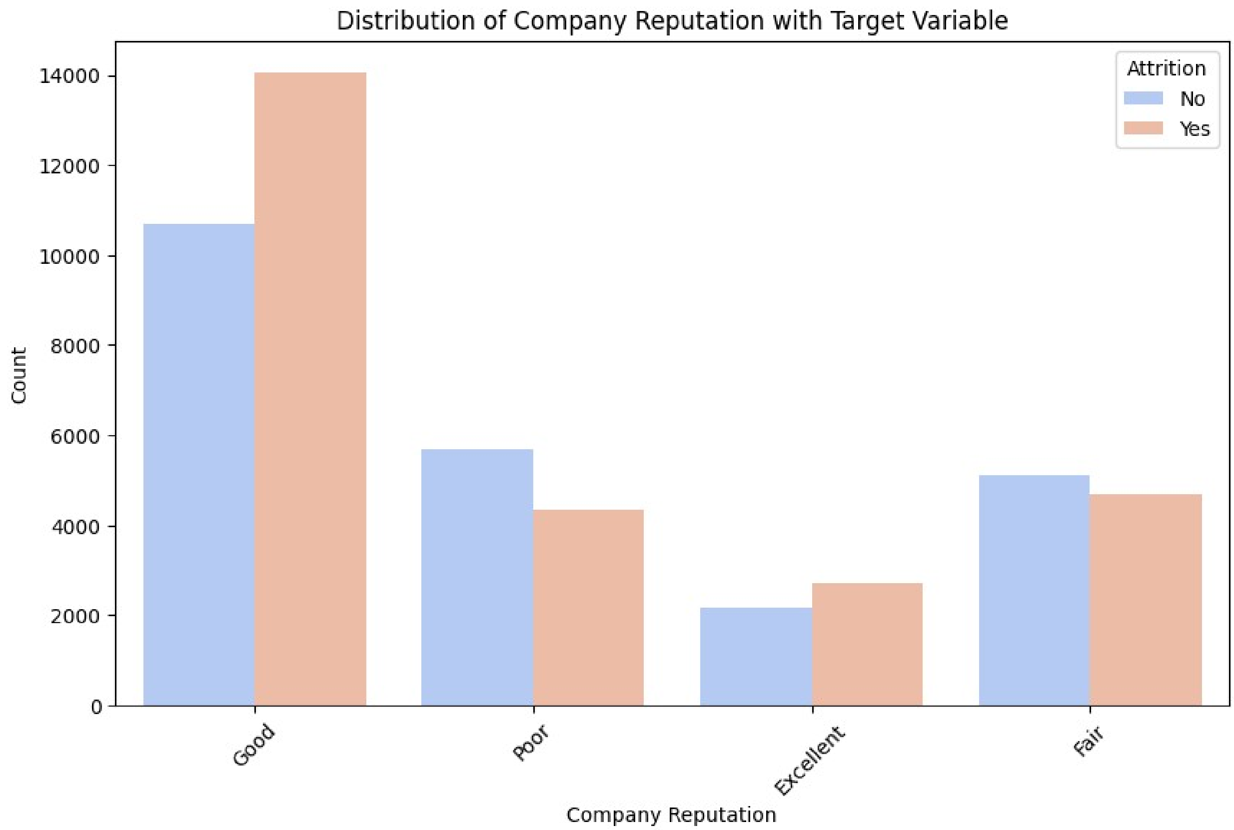

Distribution of Job Level with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
12000
Entry
Mid
Senior
Job Level


Distribution of Company Size with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
10000
12000
Medium
Small
Large
Company Size


Distribution of Remote Work with Target Variable
Attrition
No
Yes
Count
0
5000
10000
15000
20000
No
Yes
Remote Work


Distribution of Leadership Opportunities with Target Variable
Attrition
No
Yes
Count
25000
20000
15000
10000
5000
0
No
Yes
Leadership Opportunities


Distribution of Innovation Opportunities with Target Variable
Attrition
No
Yes
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
No
Yes
Innovation Opportunities


Distribution of Company Reputation with Target Variable
Attrition
No
Yes
Count
14000
12000
10000
8000
6000
4000
2000
0
Good
Poor
Excellent
Fair
Company Reputation


Distribution of Employee Recognition with Target Variable
Attrition
No
Yes
Count
10000
8000
6000
4000
2000
0
High
Low
Medium
Very High
Employee Recognition

5. EDA on validation data [OPTIONAL]

5.1 Perform univariate analysis

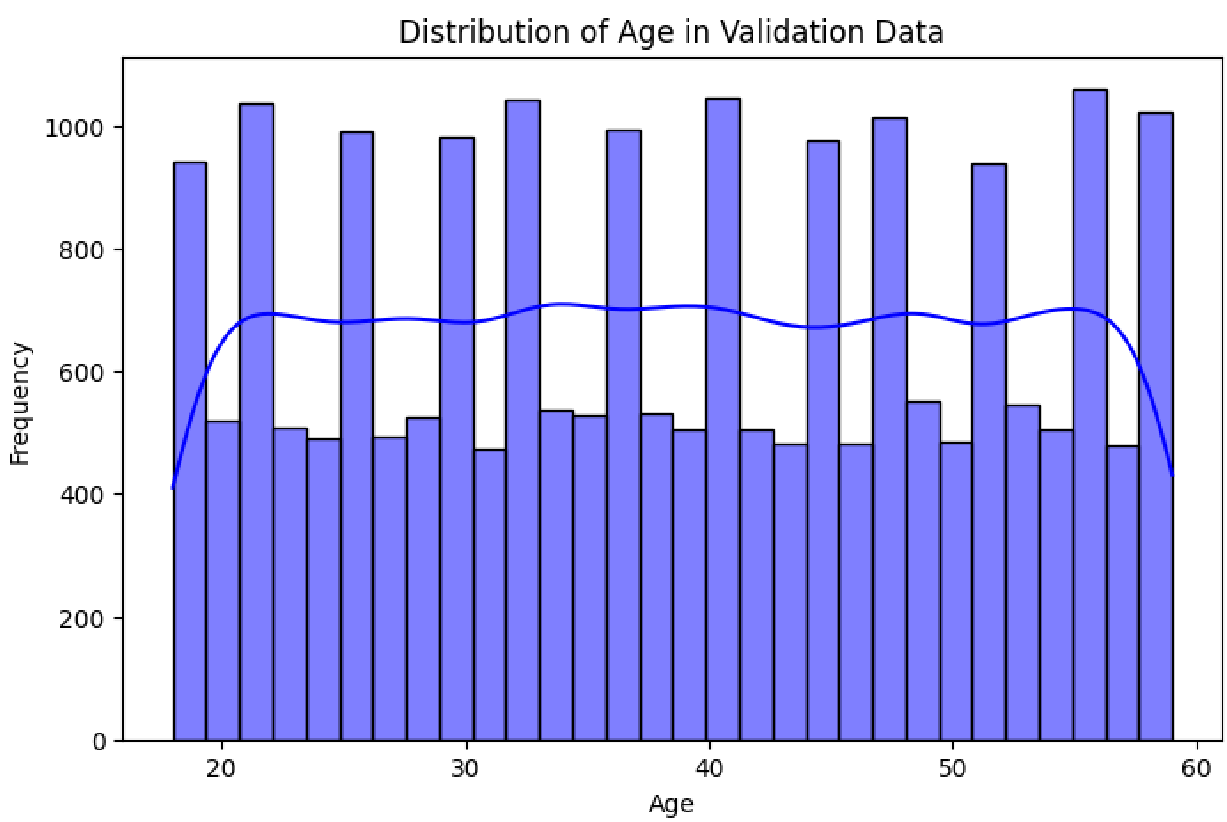
Perform univariate analysis on validation data for all the numerical columns.

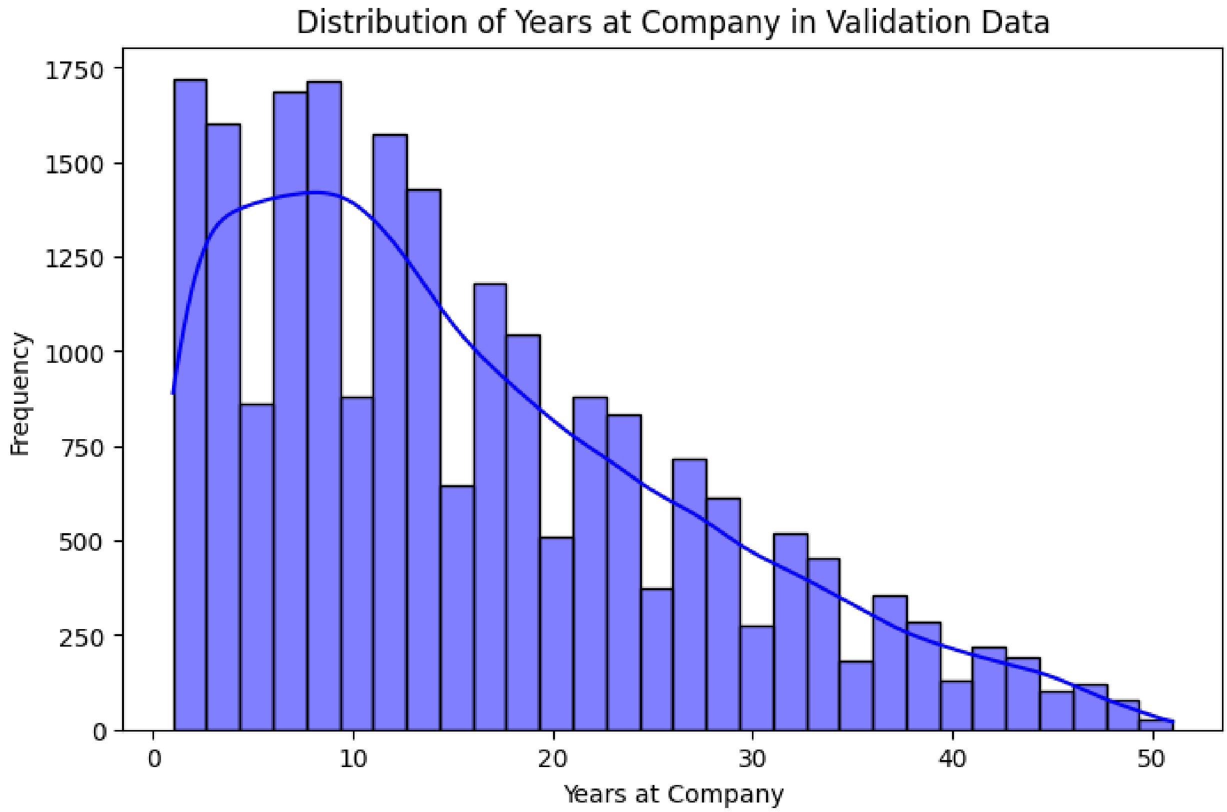
5.1.1 Select numerical columns from validation data

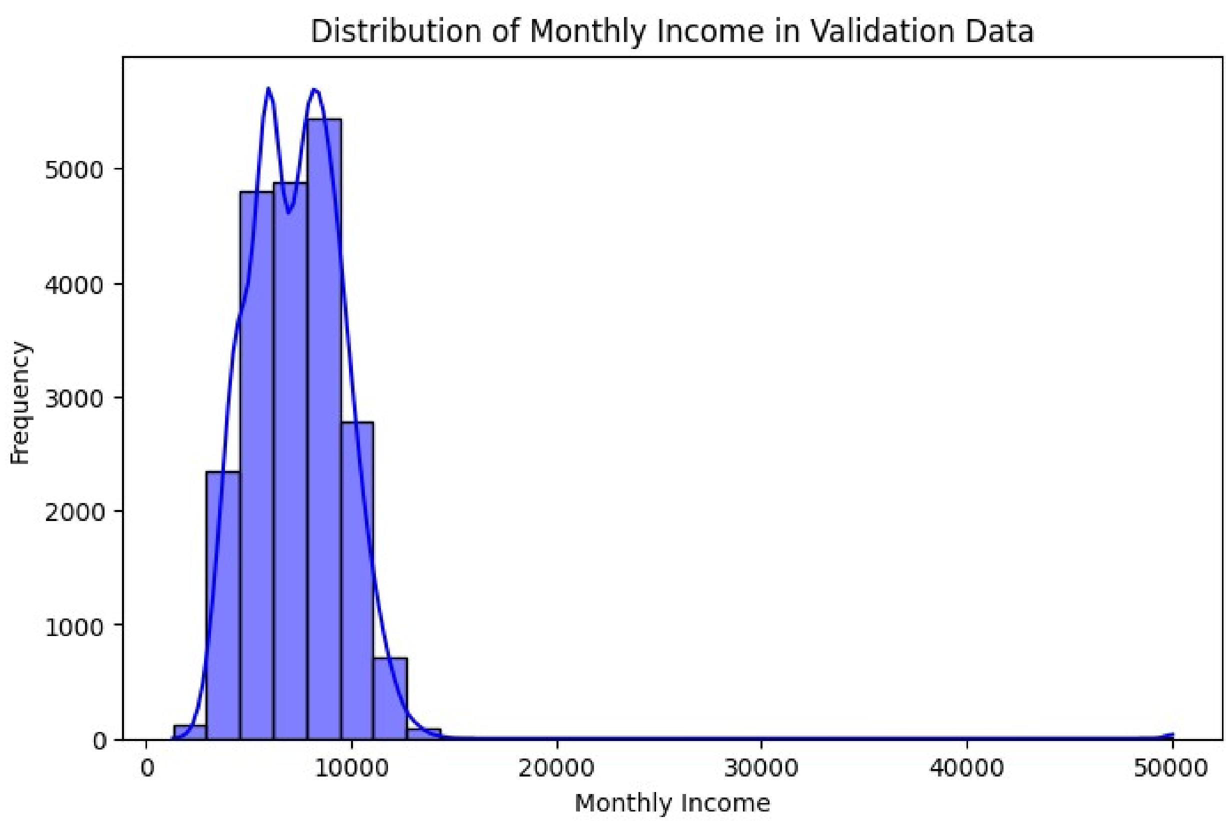
```
# Select numerical columns
numerical_columns = X_val.select_dtypes(include=['int64',
'float64']).columns
```

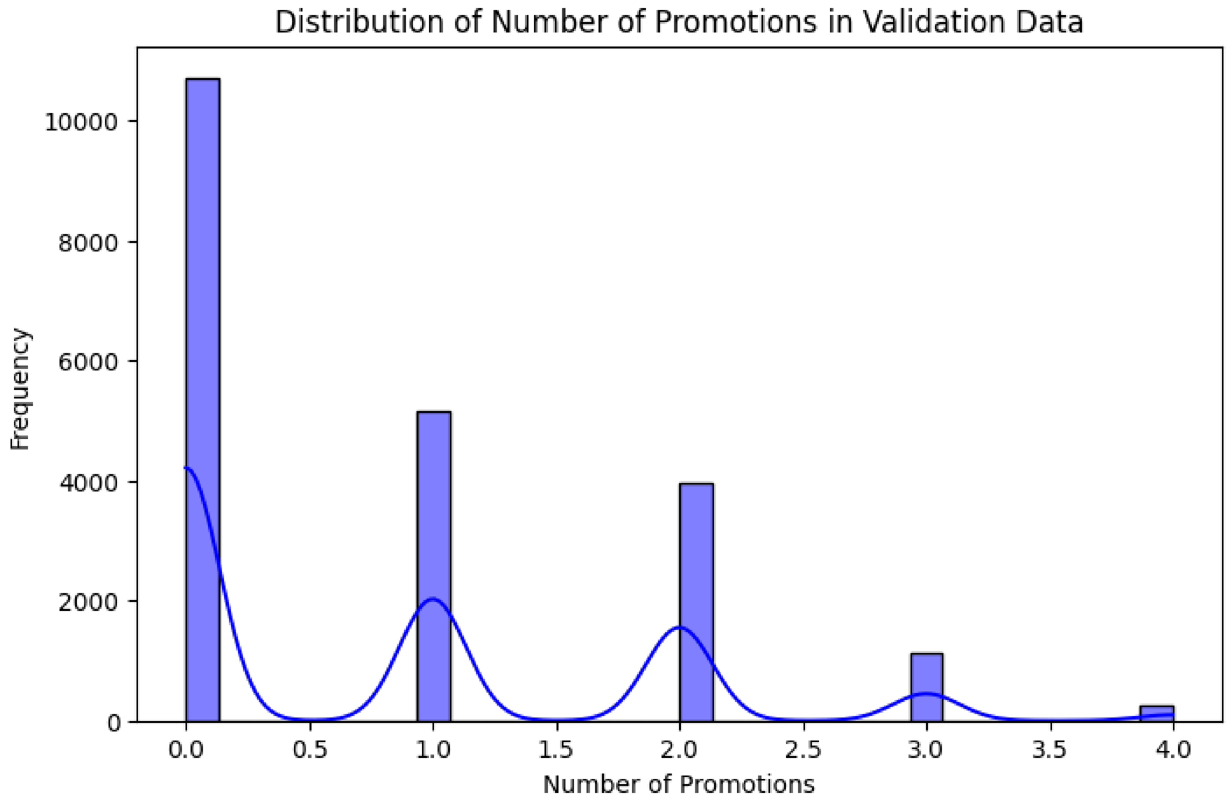
5.1.2 Plot distribution of numerical columns

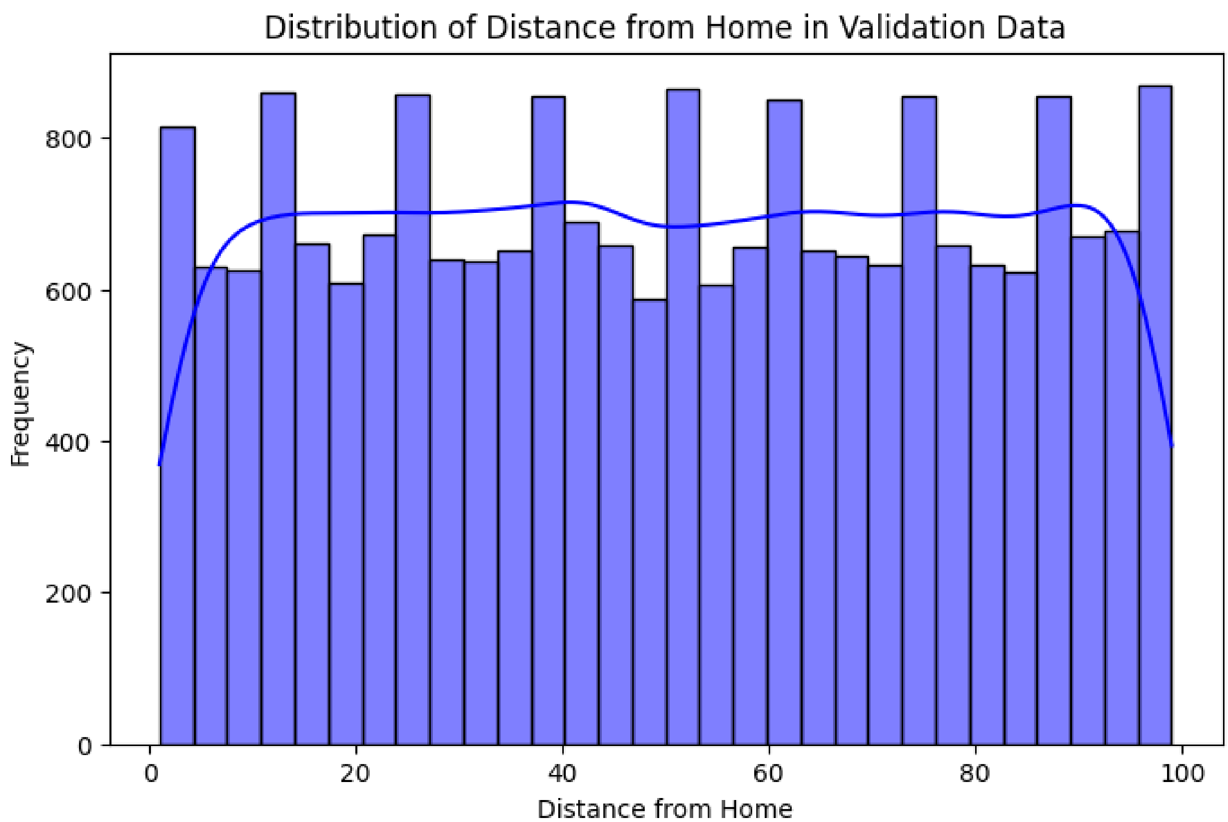
```
# Plot all the numerical columns to understand their distribution
for column in numerical_columns:
    plt.figure(figsize=(8, 5))
    sns.histplot(X_val[column], kde=True, bins=30, color='blue')  #
Histogram with KDE
    plt.title(f"Distribution of {column} in Validation Data")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()
```

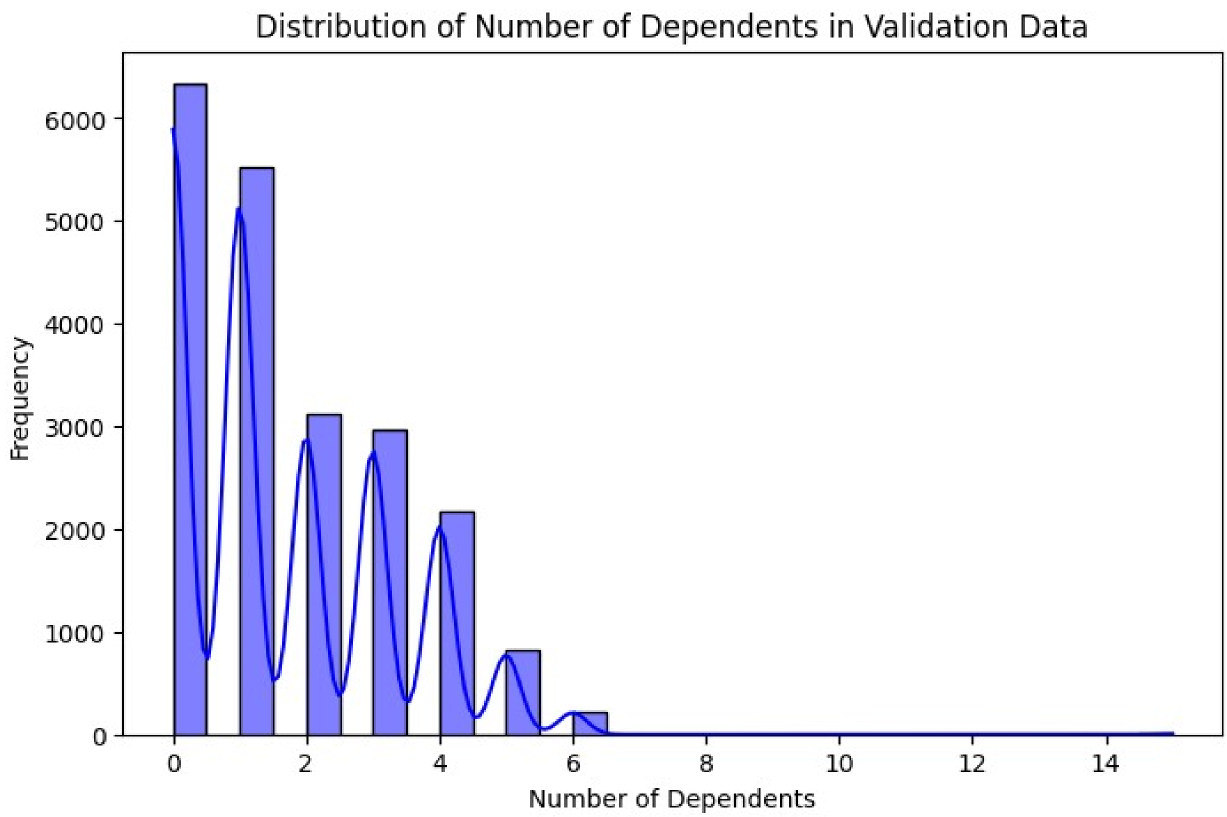



Distribution of Years at Company in Validation Data
Frequency
Years at Company
0
250
500
750
1000
1250
1500
1750
0
10
20
30
40
50


Distribution of Monthly Income in Validation Data
Frequency
Monthly Income
0
1000
2000
3000
4000
5000
0
10000
20000
30000
40000
50000


Distribution of Number of Promotions in Validation Data
Frequency
Number of Promotions
0
2000
4000
6000
8000
10000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

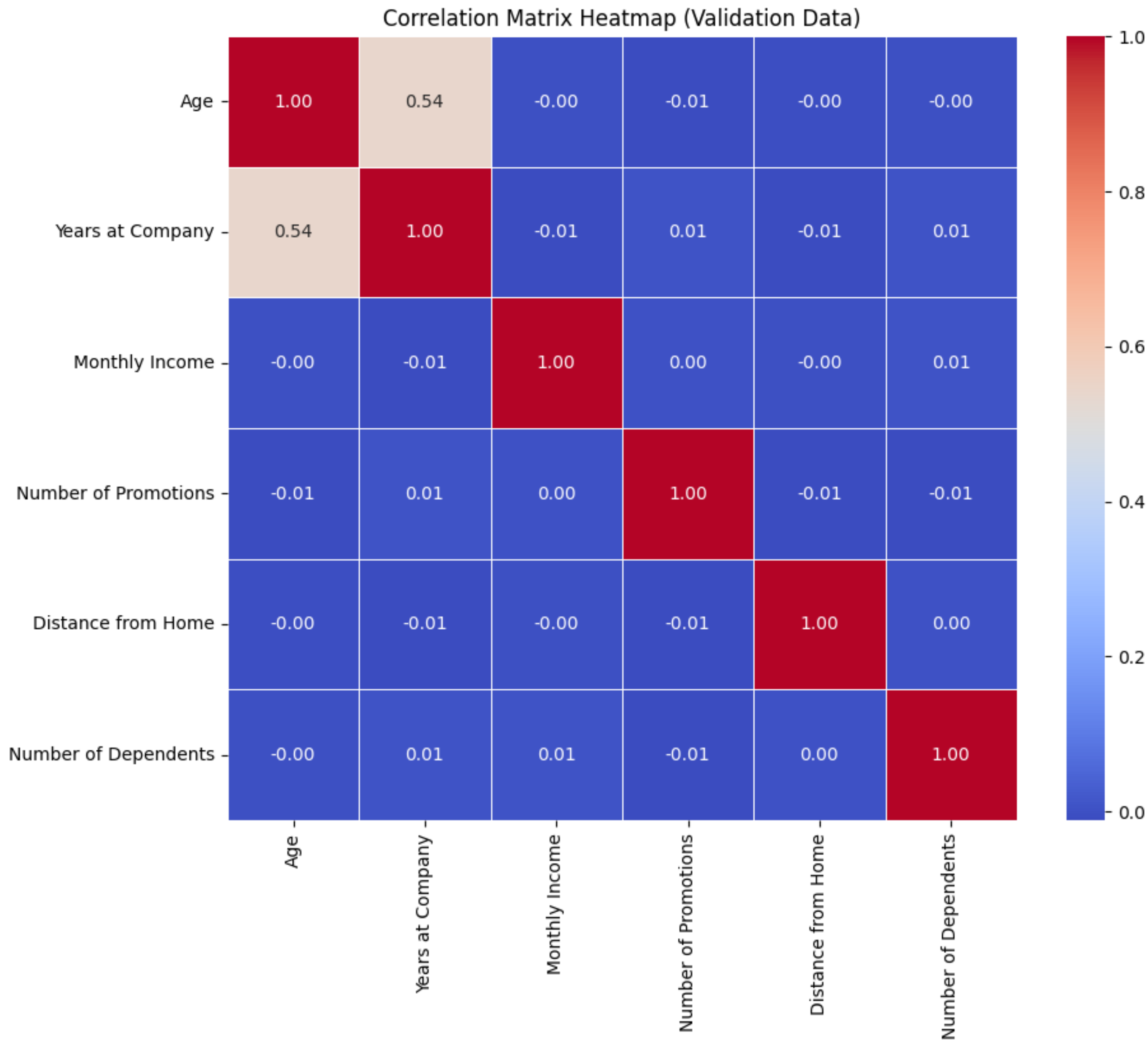

Distribution of Distance from Home in Validation Data
Frequency
Distance from Home
0
200
400
600
800
0
20
40
60
80
100



5.2 Perform correlation analysis

Check the correlation among different numerical variables.

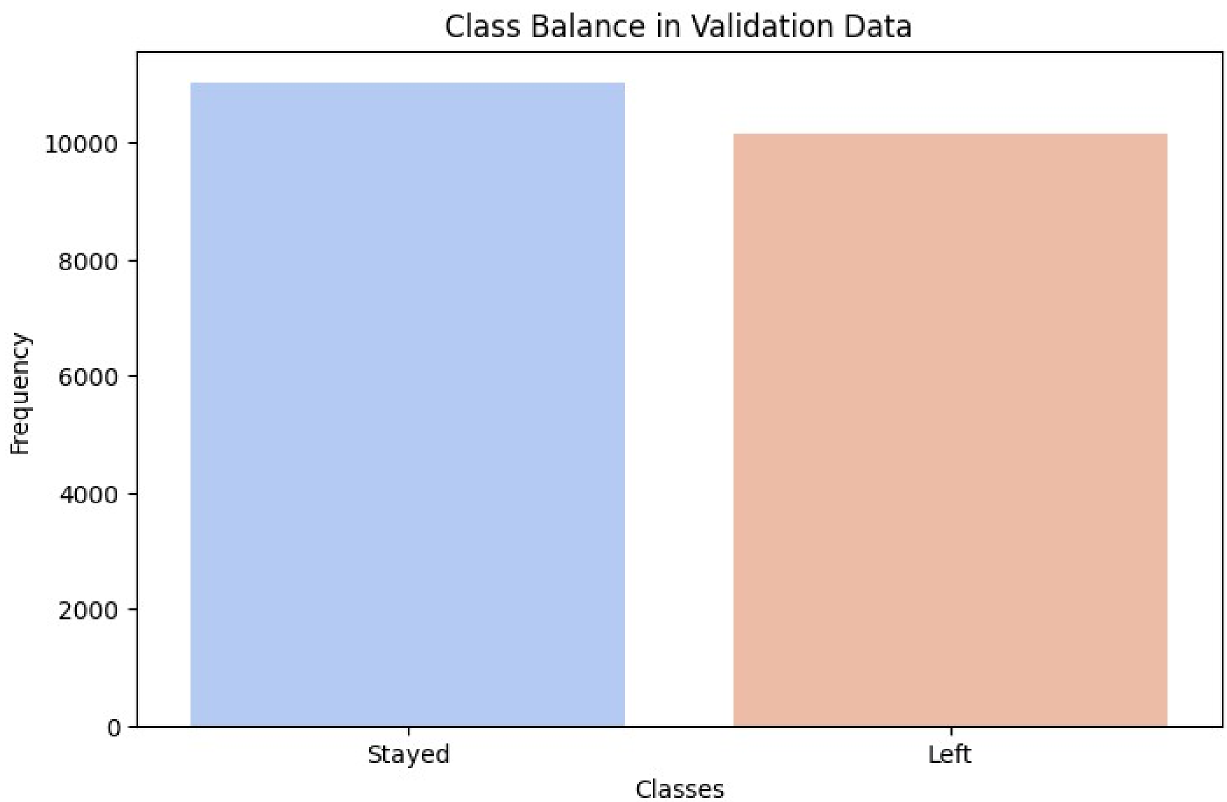
```
# Create correlation matrix for numerical columns
correlation_matrix = X_val[numerical_columns].corr()
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))  # Adjust figure size if required
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix Heatmap (Validation Data)")
plt.show()
```



5.3 Check class balance

Check the distribution of target variable in validation data to check class balance.

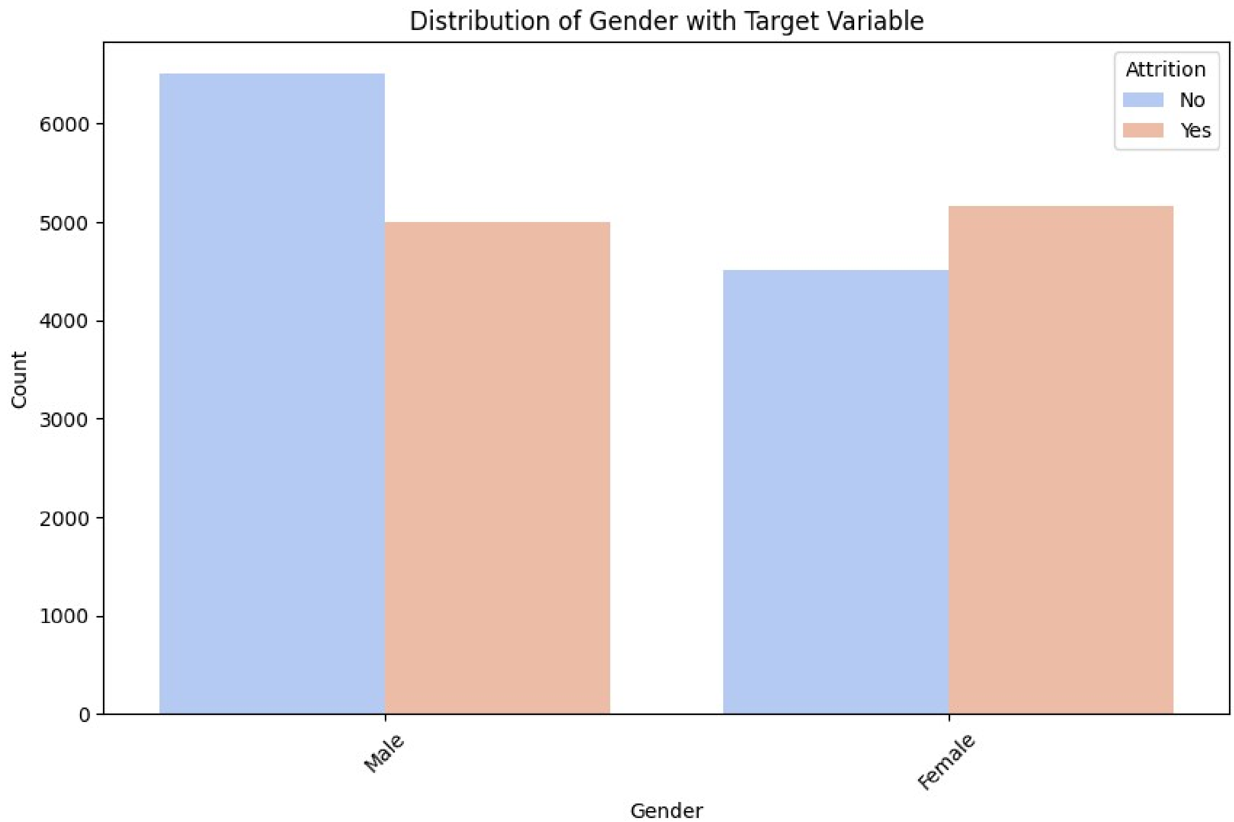
```
# Plot a bar chart to check class balance
plt.figure(figsize=(8, 5))
sns.countplot(x=y_val, palette="coolwarm")
plt.title("Class Balance in Validation Data")
plt.xlabel("Classes")
plt.ylabel("Frequency")
plt.show()
```



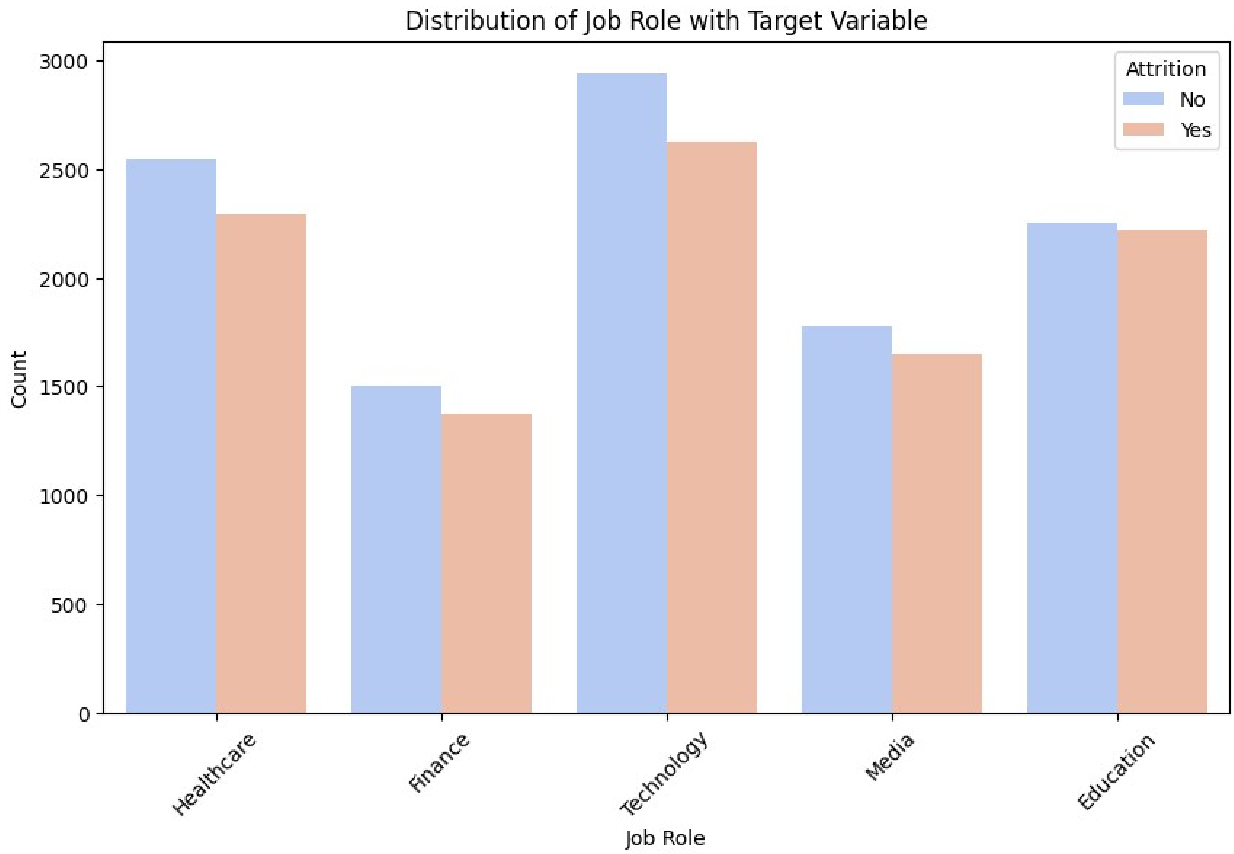
5.4 Perform bivariate analysis

Perform bivariate analysis on validation data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

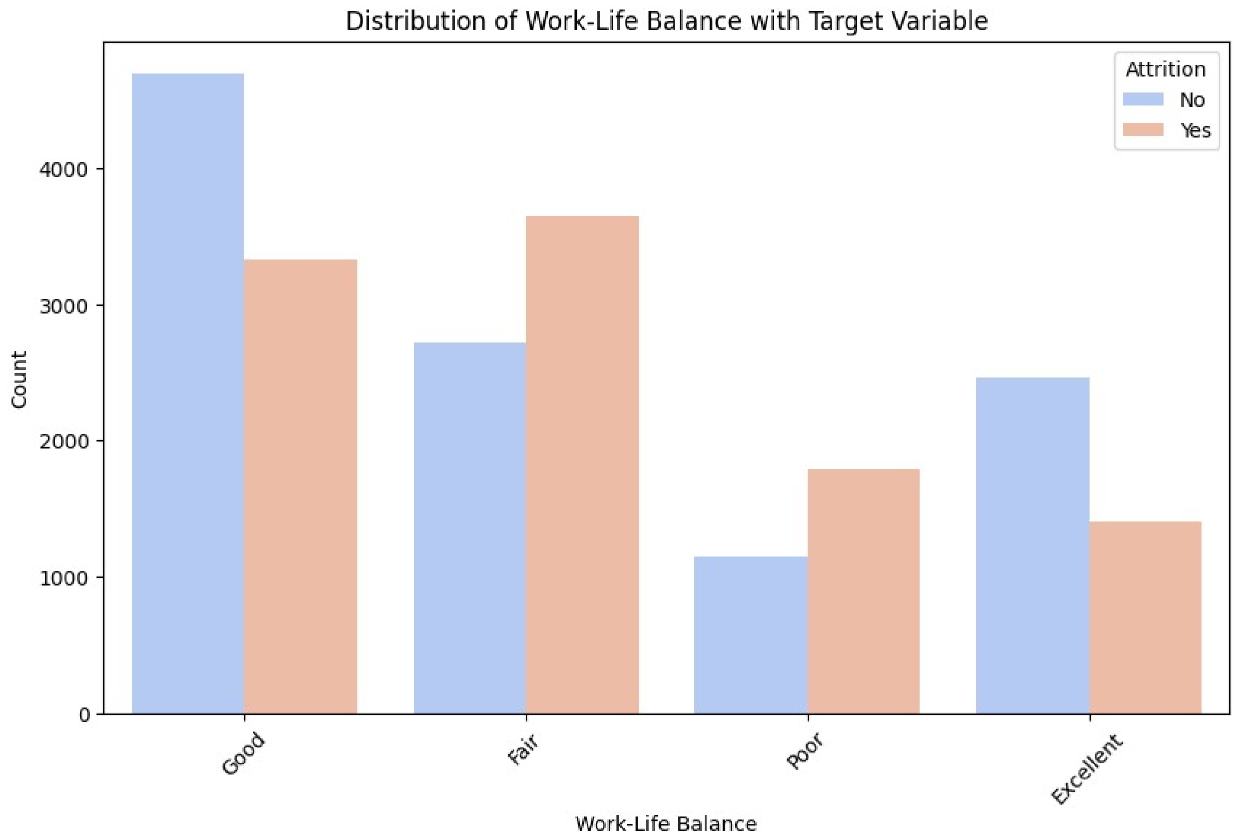
```
# Plot distribution for each categorical column with target variable
# Select categorical columns from the validation data
categorical_columns = X_val.select_dtypes(include=['object']).columns

# Plot the distribution of each categorical column with the target
variable
for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=X_val[column], hue=y_val, palette="coolwarm")
    plt.title(f"Distribution of {column} with Target Variable")
    plt.xlabel(column)
    plt.ylabel("Count")
    plt.legend(title="Attrition", labels=["No", "Yes"])
    plt.xticks(rotation=45)  # Adjust for better readability
    plt.show()
```

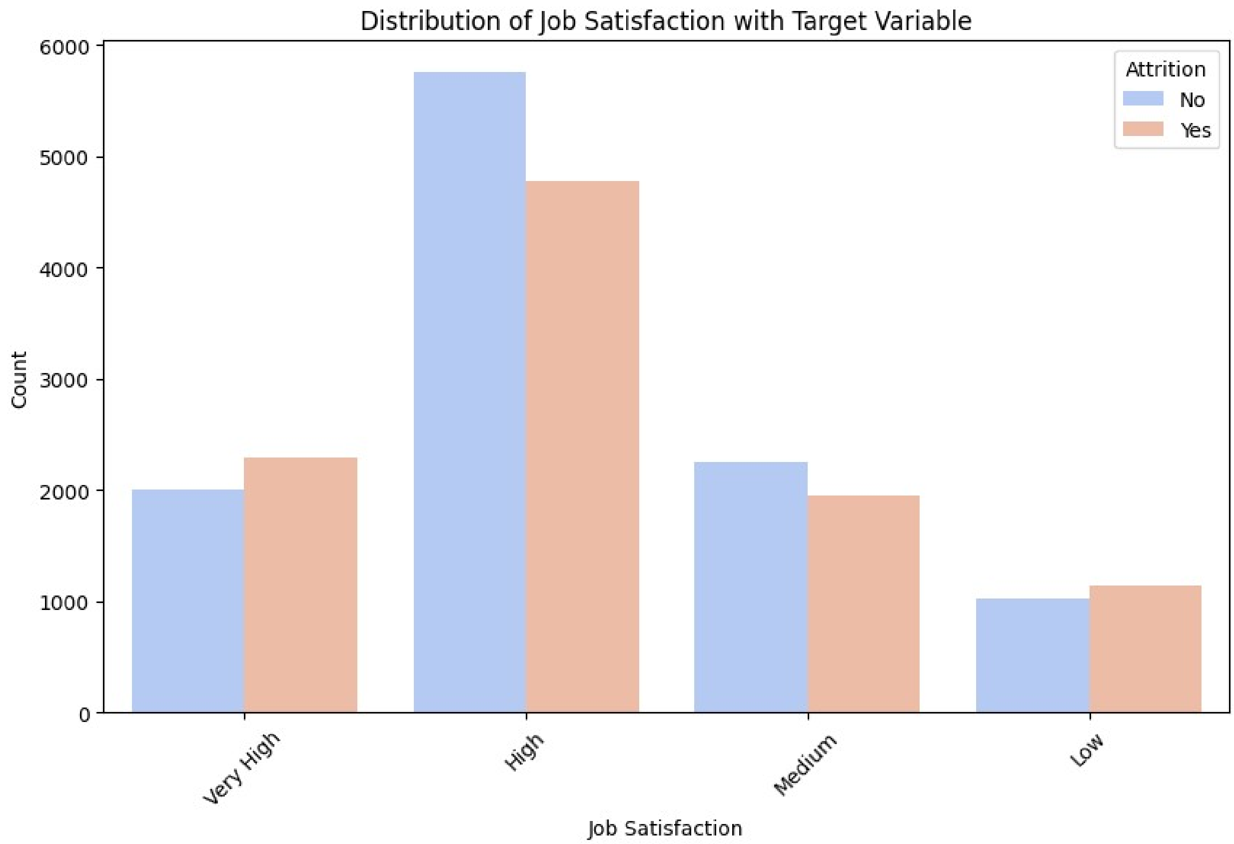


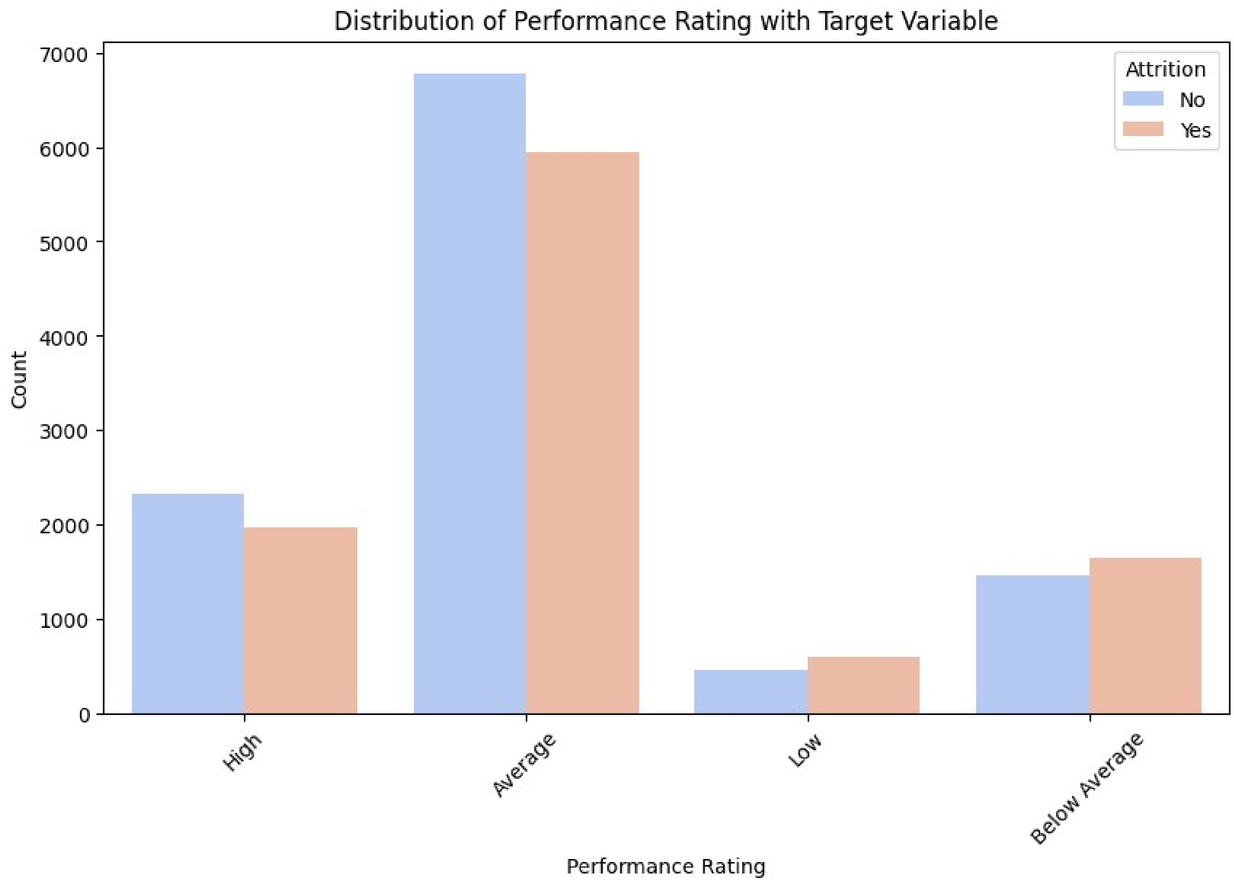
Distribution of Gender with Target Variable
Attrition
No
Yes
6000
5000
4000
3000
2000
1000
0
Count
Male
Female
Gender



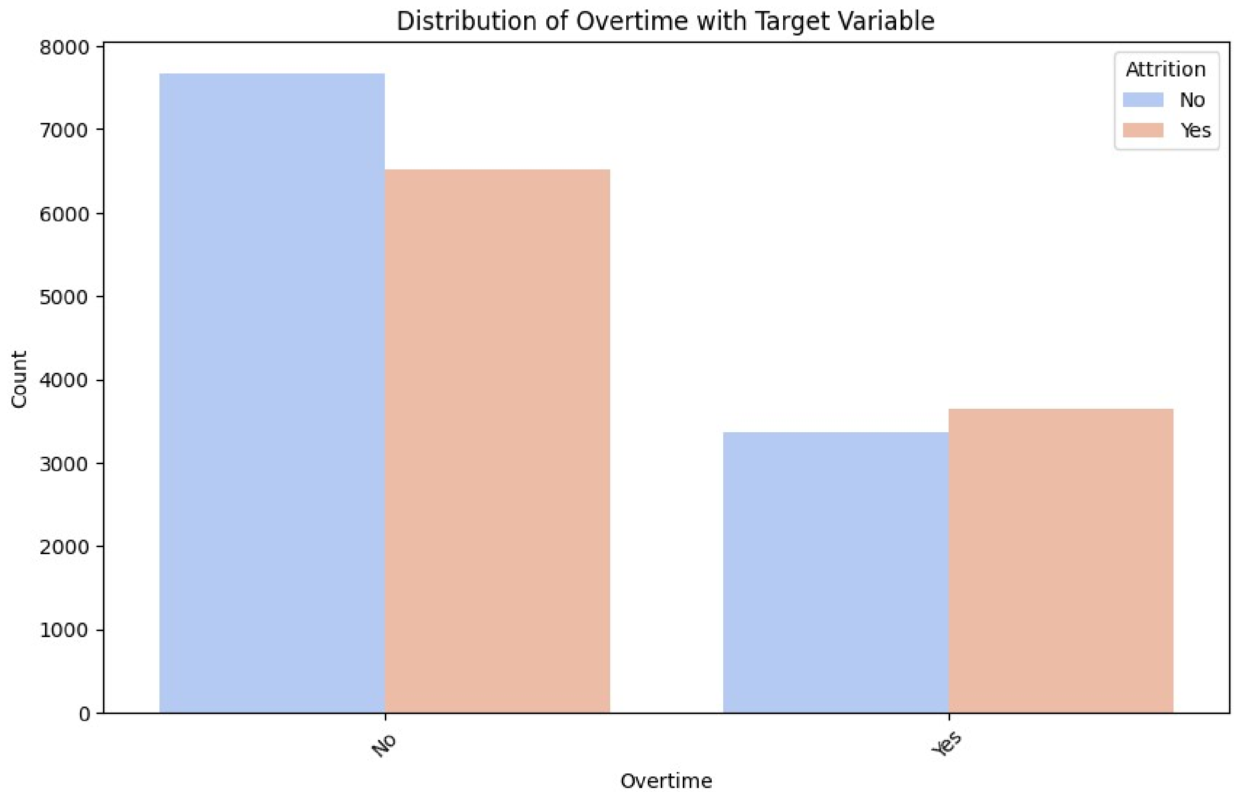
Distribution of Job Role with Target Variable
Attrition
No
Yes
Count
3000
2500
2000
1500
1000
500
0
Healthcare
Finance
Technology
Media
Education
Job Role


Distribution of Work-Life Balance with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
Good
Fair
Poor
Excellent
Work-Life Balance

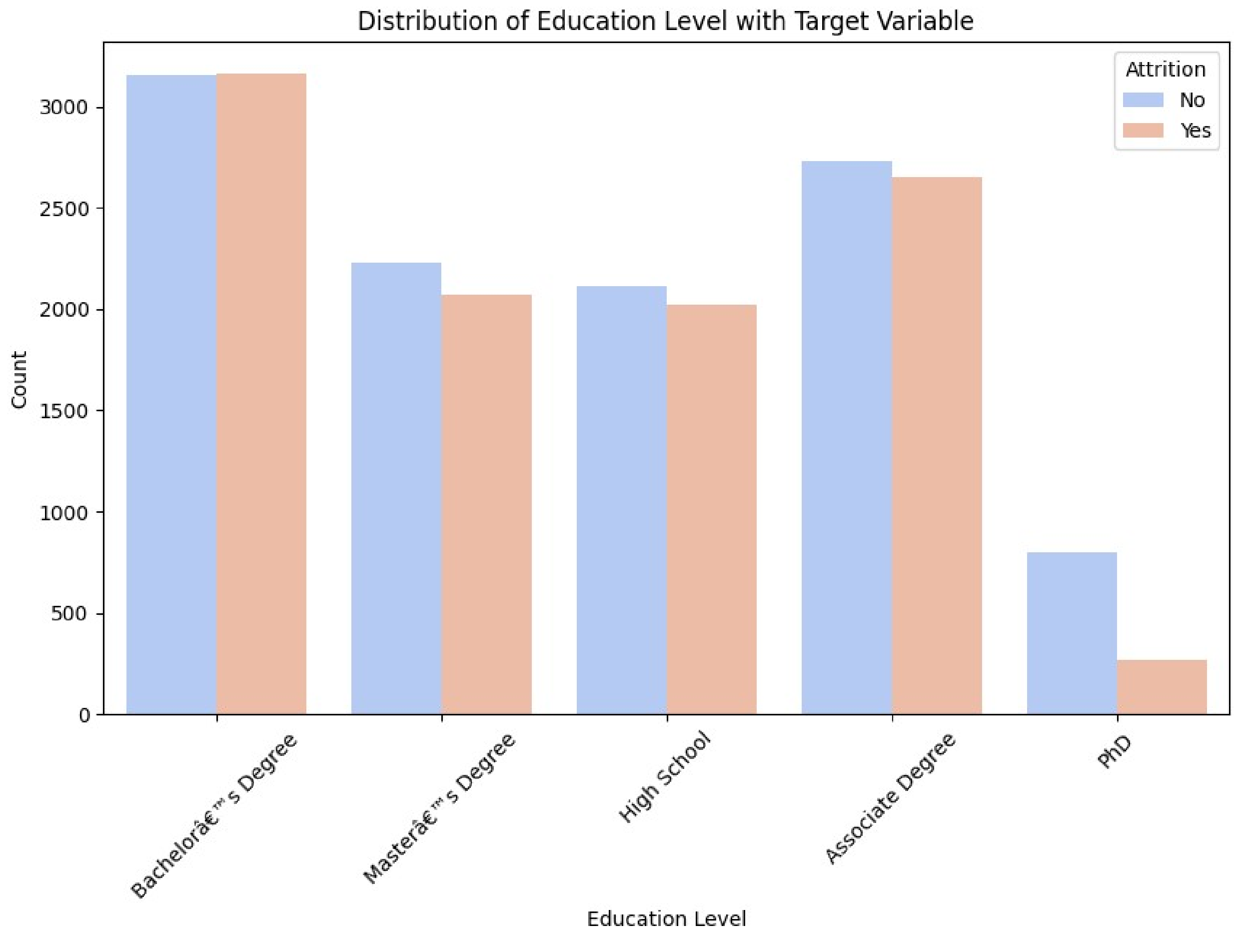

Distribution of Job Satisfaction with Target Variable
Count
6000
5000
4000
3000
2000
1000
0
Attrition
No
Yes
Very High
High
Medium
Low
Job Satisfaction



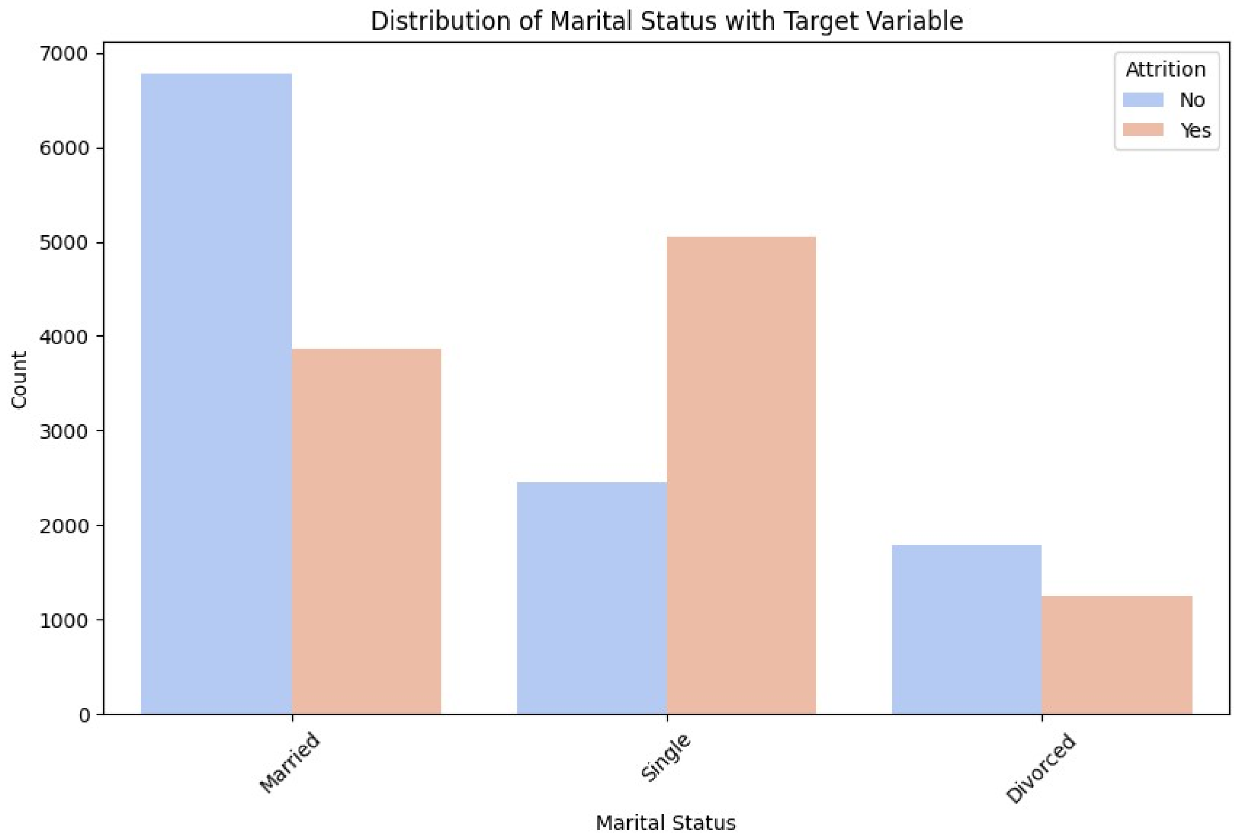
Distribution of Performance Rating with Target Variable
Attrition
No
Yes
Count
7000
6000
5000
4000
3000
2000
1000
0
High
Average
Low
Below Average
Performance Rating

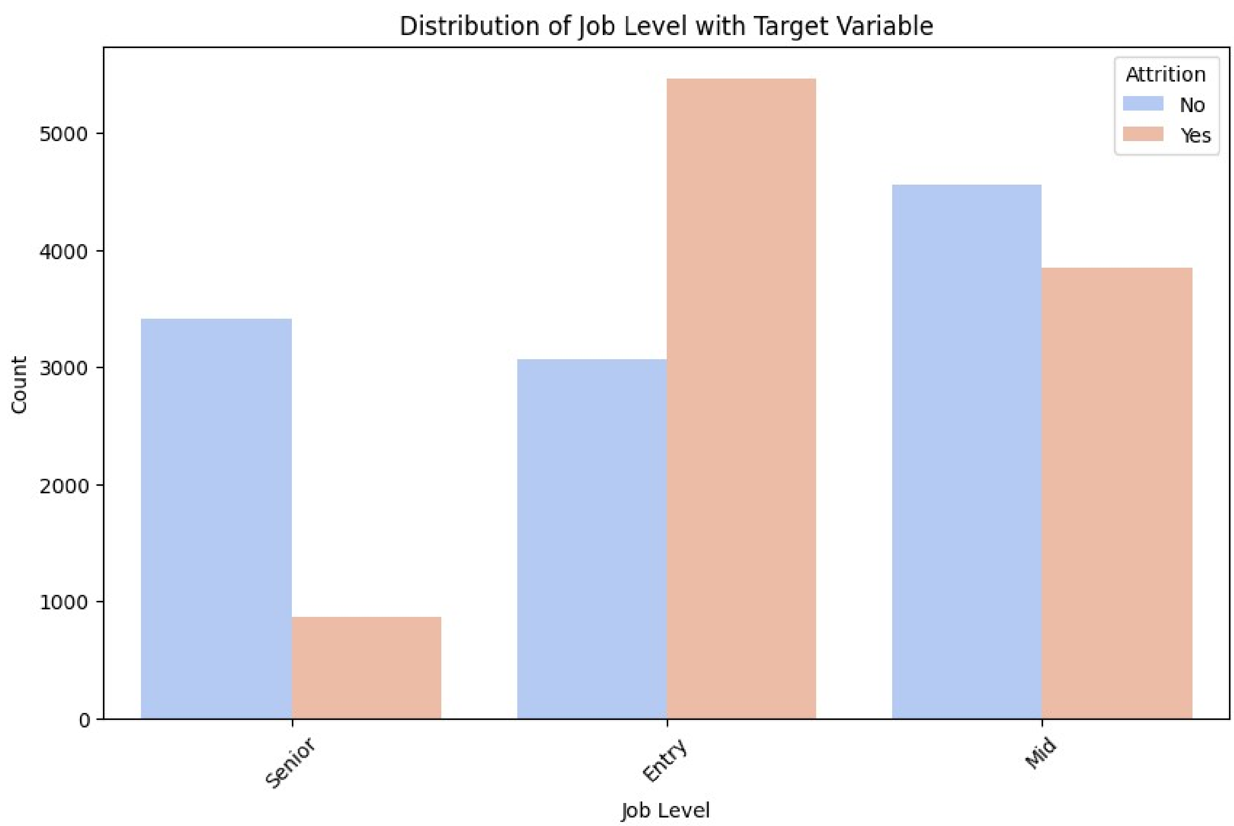


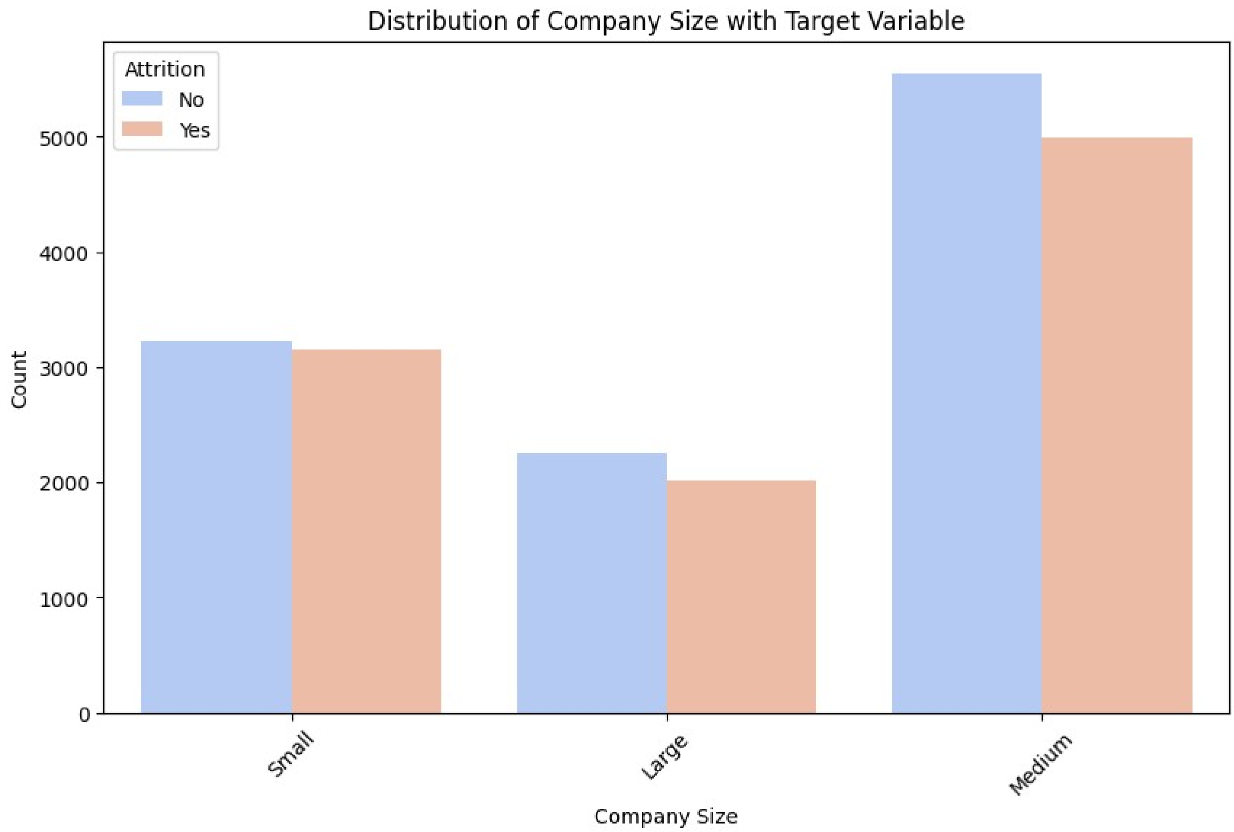
Distribution of Overtime with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
5000
6000
7000
8000
No
Yes
Overtime

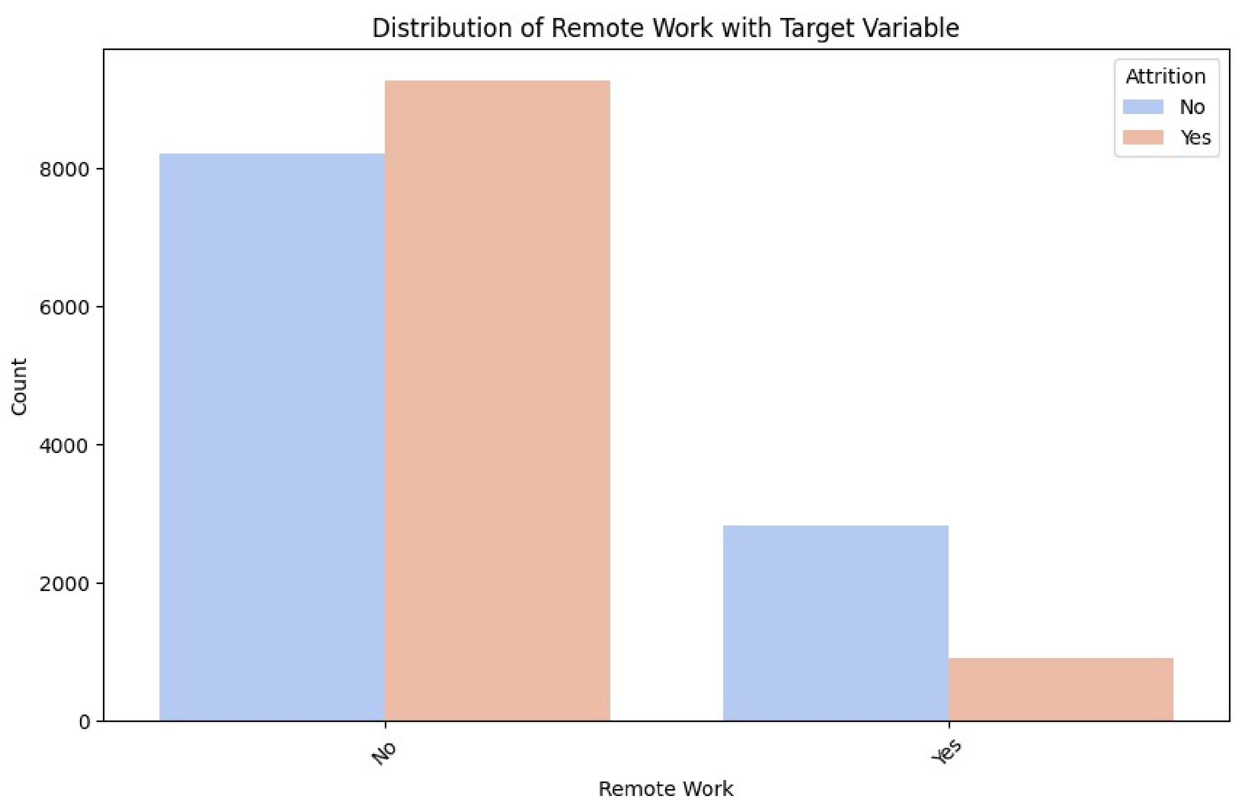


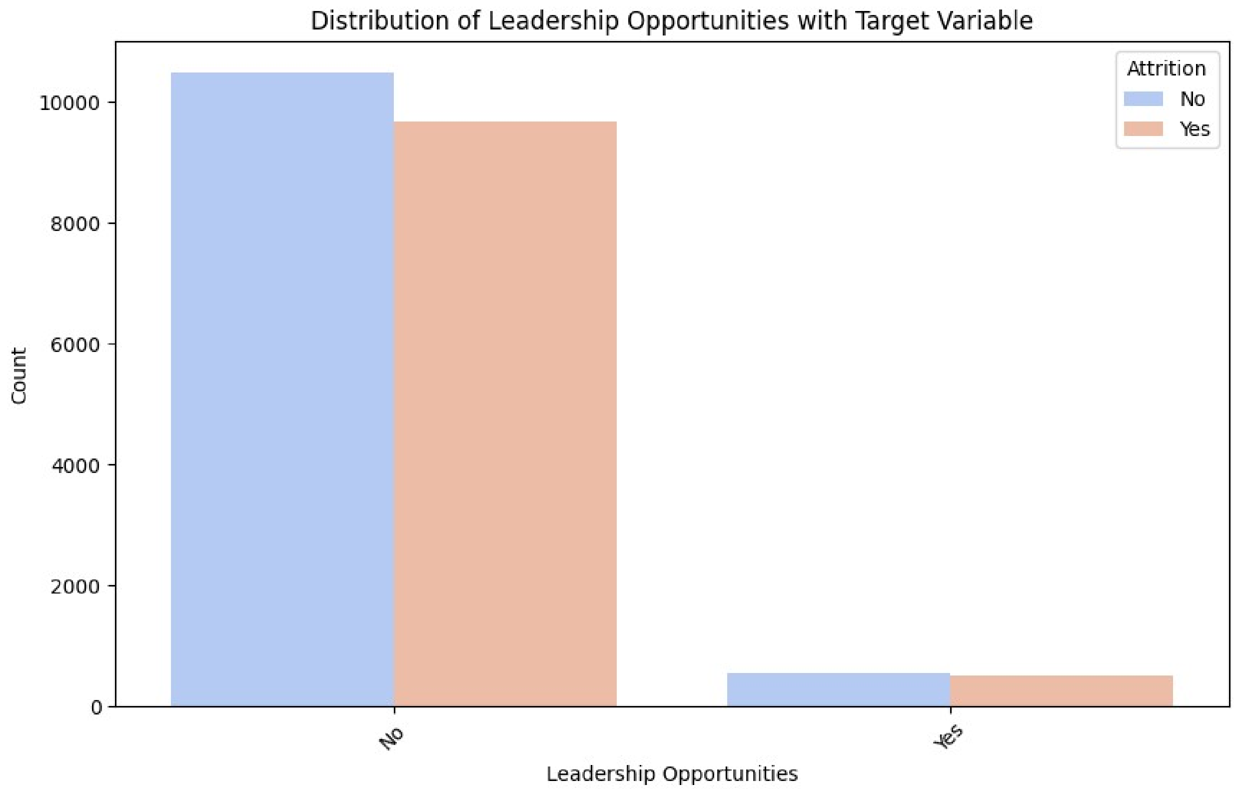
Distribution of Education Level with Target Variable
Attrition
No
Yes
Count
3000
2500
2000
1500
1000
500
0
Bachelorâ€™s Degree
Masterâ€™s Degree
High School
Associate Degree
PhD
Education Level

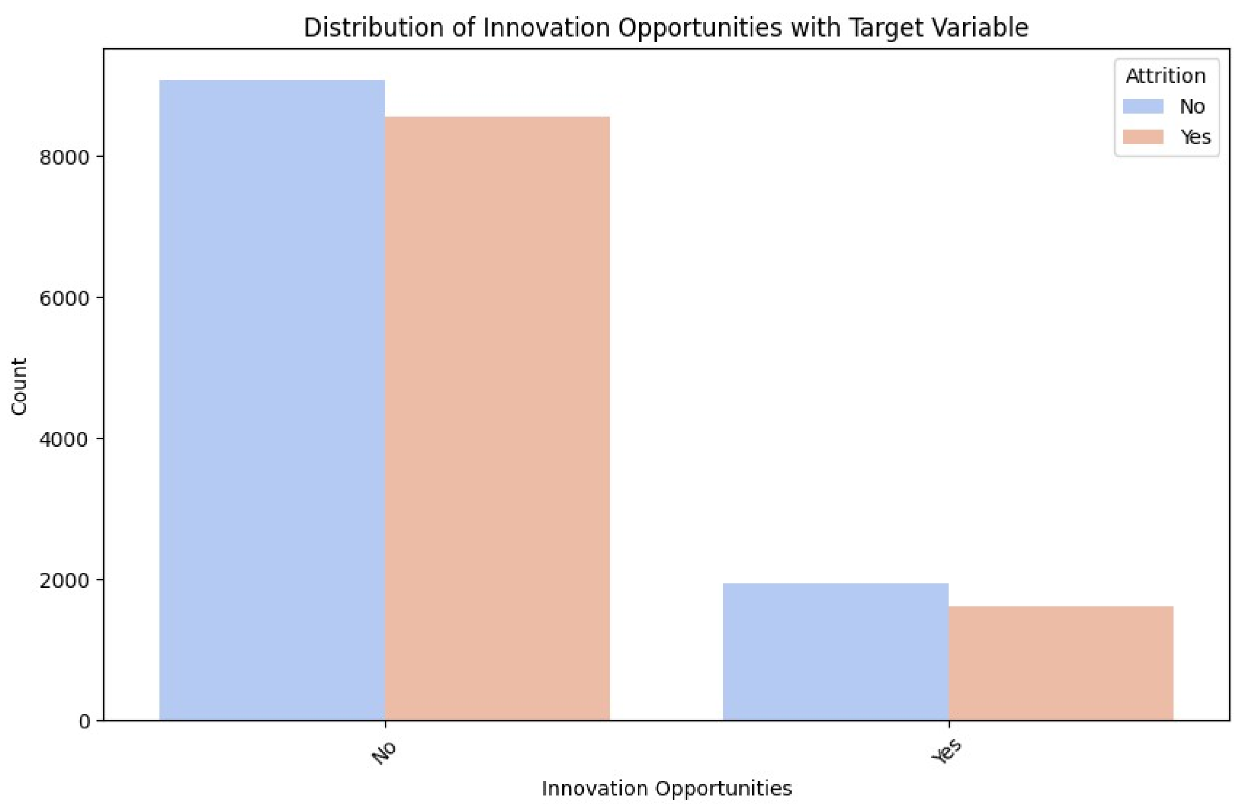

Distribution of Marital Status with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
5000
6000
7000
Married
Single
Divorced
Marital Status

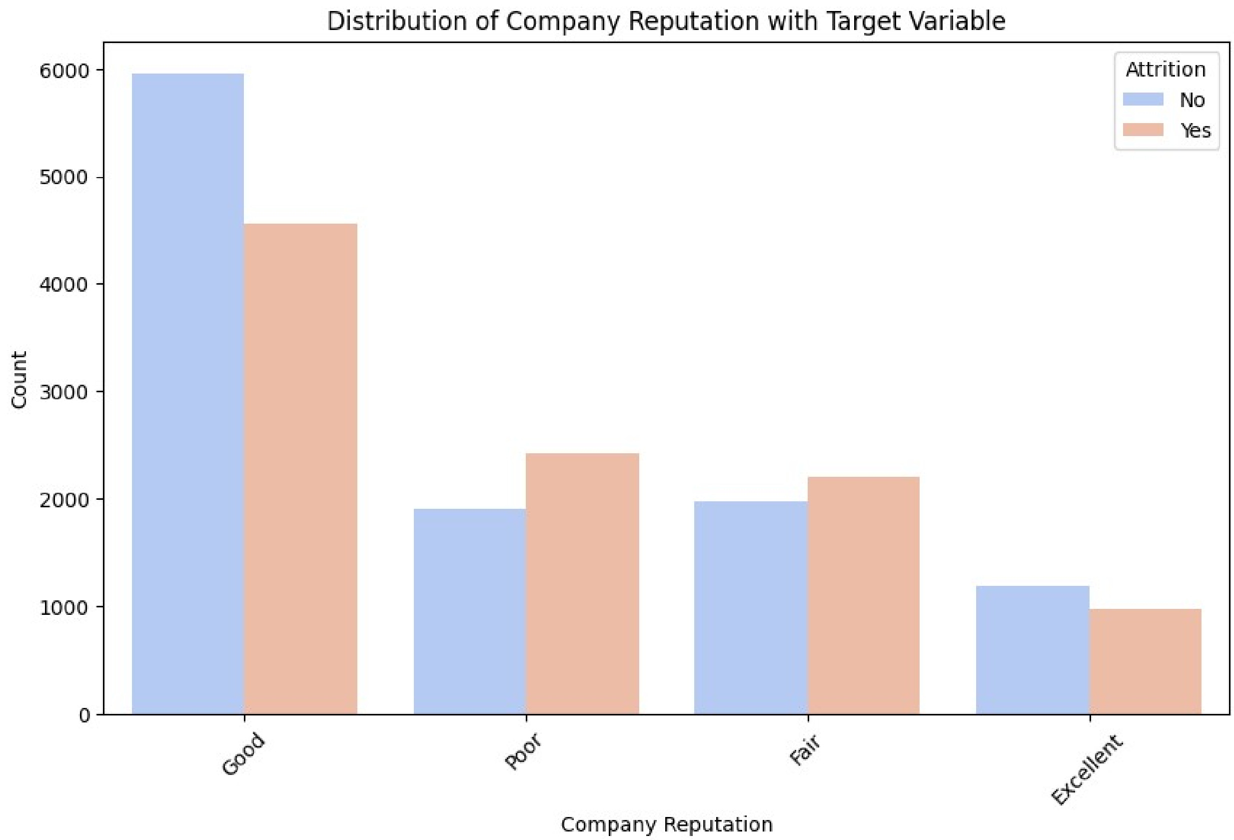

Distribution of Job Level with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
5000
Senior
Entry
Mid
Job Level


Distribution of Company Size with Target Variable
Attrition
No
Yes
Count
0
1000
2000
3000
4000
5000
Small
Large
Medium
Company Size


Distribution of Remote Work with Target Variable
Attrition
No
Yes
Count
0
2000
4000
6000
8000
No
Yes
Remote Work


Distribution of Leadership Opportunities with Target Variable
Attrition
No
Yes
0
2000
4000
6000
8000
10000
Count
No
Yes
Leadership Opportunities


Distribution of Innovation Opportunities with Target Variable
Attrition
No
Yes
0
2000
4000
6000
8000
Count
No
Yes
Innovation Opportunities


Distribution of Company Reputation with Target Variable
Attrition
No
Yes
Count
6000
5000
4000
3000
2000
1000
0
Good
Poor
Fair
Excellent
Company Reputation

6. Feature Engineering [20 marks]

6.1 Dummy variable creation [15 marks]

The next step is to deal with the categorical variables present in the data.

6.1.1 Identify categorical columns where dummy variables are required [1 Mark]

```
# Check the categorical columns
categorical_columns =
X_train.select_dtypes(include=['object']).columns

# Output the categorical columns
print("Categorical columns in the dataset:")
print(categorical_columns)
```

```
Categorical columns in the dataset:
Index(['Gender', 'Job Role', 'Work-Life Balance', 'Job Satisfaction',
       'Performance Rating', 'Overtime', 'Education Level', 'Marital
Status',
       'Job Level', 'Company Size', 'Remote Work', 'Leadership
Opportunities',
       'Innovation Opportunities', 'Company Reputation',
```

```
       'Employee Recognition'],
      dtype='object')
```

6.1.2 Create dummy variables for independent columns in training set [3 Marks]

```
# Create dummy variables using the 'get_dummies' for independent
columns
X_train_encoded = pd.get_dummies(X_train, drop_first=True)  # Drop
first category to prevent multicollinearity
X_val_encoded = pd.get_dummies(X_val, drop_first=True)

# Add the results to the master DataFrame

# Check the resulting shapes of the master DataFrames
print("Shape of Training DataFrame after encoding:",
X_train_encoded.shape)
print("Shape of Validation DataFrame after encoding:",
X_val_encoded.shape)

Shape of Training DataFrame after encoding: (49444, 40)
Shape of Validation DataFrame after encoding: (21191, 40)
```

Now, drop the original categorical columns and check the DataFrame

```
# Drop the original categorical columns and check the DataFrame
# Drop original categorical columns from training and validation data
X_train_encoded = X_train.drop(columns=categorical_columns)
X_val_encoded = X_val.drop(columns=categorical_columns)

# Check the resulting DataFrame
print("Training DataFrame after dropping categorical columns:")
print(X_train_encoded.head())

print("\nValidation DataFrame after dropping categorical columns:")
print(X_val_encoded.head())

Training DataFrame after dropping categorical columns:
       Age  Years at Company  Monthly Income  Number of Promotions  \
41465   46                26            4617                     2
69350   43                26            4447                     3
28247   27                 8            9762                     0
3217    47                29            5244                     2
73636   25                10            4455                     0

       Distance from Home  Number of Dependents
41465                59.0                     2
69350                64.0                     0
28247                84.0                     1
3217                 59.0                     1
73636                36.0                     0
```

```
Validation DataFrame after dropping categorical columns:
       Age  Years at Company  Monthly Income  Number of Promotions  \
23813   46                 1            7740                     3
14537   32                 6            8779                     1
45192   53                10           11683                     1
13765   50                 2            7305                     2
3411    48                24           10114                     0

       Distance from Home  Number of Dependents
23813                78.0                     2
14537                91.0                     0
45192                71.0                     1
13765                71.0                     1
3411                  2.0                     1
```

6.1.3 Create dummy variables for independent columns in validation set [3 Marks]

```
# Create dummy variables using the 'get_dummies' for independent
columns

# Create dummy variables for categorical columns in training and
validation sets
X_train_encoded = pd.get_dummies(X_train, drop_first=True)  # Training
data
X_val_encoded = pd.get_dummies(X_val, drop_first=True)      #
Validation data

# Display the resulting shapes after encoding
print("Shape of Training DataFrame after encoding:",
X_train_encoded.shape)
print("Shape of Validation DataFrame after encoding:",
X_val_encoded.shape)

# Add the results to the master DataFrame

Shape of Training DataFrame after encoding: (49444, 40)
Shape of Validation DataFrame after encoding: (21191, 40)
```

Now, drop the original categorical columns and check the DataFrame

```
# Drop categorical columns and check the DataFrame
X_train_encoded = X_train.drop(columns=categorical_columns)
X_val_encoded = X_val.drop(columns=categorical_columns)

# Check the resulting DataFrame
print("Training DataFrame after dropping categorical columns:")
print(X_train_encoded.head())
```

```
print("\nValidation DataFrame after dropping categorical columns:")
print(X_val_encoded.head())
```

```
Training DataFrame after dropping categorical columns:
       Age  Years at Company  Monthly Income  Number of Promotions  \
41465   46                26            4617                     2
69350   43                26            4447                     3
28247   27                 8            9762                     0
3217    47                29            5244                     2
73636   25                10            4455                     0

       Distance from Home  Number of Dependents
41465                59.0                     2
69350                64.0                     0
28247                84.0                     1
3217                 59.0                     1
73636                36.0                     0

Validation DataFrame after dropping categorical columns:
       Age  Years at Company  Monthly Income  Number of Promotions  \
23813   46                 1            7740                     3
14537   32                 6            8779                     1
45192   53                10           11683                     1
13765   50                 2            7305                     2
3411    48                24           10114                     0

       Distance from Home  Number of Dependents
23813                78.0                     2
14537                91.0                     0
45192                71.0                     1
13765                71.0                     1
3411                  2.0                     1
```

6.1.4 Create DataFrame for dependent column in both training and validation set [1 Mark]

```
# Convert y_train and y_validation to DataFrame to create dummy
variables
# Convert y_train and y_val to DataFrames
y_train_df = pd.DataFrame(y_train, columns=['Attrition'])
y_val_df = pd.DataFrame(y_val, columns=['Attrition'])

# Create dummy variables for the target variable
y_train_encoded = pd.get_dummies(y_train_df, drop_first=True)
y_val_encoded = pd.get_dummies(y_val_df, drop_first=True)

# Check the encoded target variable
print("Encoded y_train:")
print(y_train_encoded.head())
```

```
print("\nEncoded y_val:")
print(y_val_encoded.head())
```

```
Encoded y_train:
       Attrition_Stayed
41465             False
69350             False
28247             False
3217              False
73636             False

Encoded y_val:
       Attrition_Stayed
23813              True
14537             False
45192             False
13765             False
3411               True
```

6.1.5 Create dummy variables for dependent column in training set [3 Marks]

```
# Create dummy variables using the 'get_dummies' for dependent column
y_train_encoded = pd.get_dummies(y_train, drop_first=True)
y_val_encoded = pd.get_dummies(y_val, drop_first=True)

# Check the resulting dummy variables
print("Encoded y_train:")
print(y_train_encoded.head())

print("\nEncoded y_val:")
print(y_val_encoded.head())
```

```
Encoded y_train:
       Stayed
41465   False
69350   False
28247   False
3217    False
73636   False

Encoded y_val:
       Stayed
23813    True
14537   False
45192   False
13765   False
3411     True
```

6.1.6 Create dummy variable for dependent column in validation set [3 Marks]

```
# Create dummy variables using the 'get_dummies' for dependent column
y_val_encoded = pd.get_dummies(y_val, drop_first=True)
```

6.1.7 Drop redundant columns [1 Mark]

```
# Drop redundant columns from both train and validation
def drop_redundant_columns(X_train, X_val):
    # Identify constant columns
    constant_columns = [col for col in X_train.columns if
X_train[col].nunique() == 1]

    # Identify duplicate columns
    duplicate_columns =
X_train.T[X_train.T.duplicated()].index.tolist()

    # Identify columns in train but not in validation
    train_only_columns = [col for col in X_train.columns if col not in
X_val.columns]

    # Consolidate all columns to drop
    columns_to_drop = list(set(constant_columns + duplicate_columns +
train_only_columns))

    # Drop from both datasets
    X_train.drop(columns=columns_to_drop, inplace=True)
    X_val.drop(columns=columns_to_drop, inplace=True)

    return X_train, X_val

# Apply to datasets
X_train, X_val = drop_redundant_columns(X_train, X_val)
```

6.2 Feature scaling [5 marks]

Apply feature scaling to the numeric columns to bring them to a common range and ensure consistent scaling.

6.2.1 Import required libraries [1 Mark]

```
# Import the necessary scaling tool from scikit-learn
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

6.2.2 Scale the numerical features [4 Marks]

```
numeric_cols = X_train.select_dtypes(include=['int64',
'float64']).columns

# Initialize StandardScaler
scaler = StandardScaler()
```

```python
# Scale the numeric features present in the training set
X_train_scaled =
pd.DataFrame(scaler.fit_transform(X_train[numeric_cols]),
columns=numeric_cols)

# Scale the numerical features present in the validation set
X_val_scaled = pd.DataFrame(scaler.transform(X_val[numeric_cols]),
columns=numeric_cols)
```

7. Model Building [40 marks]

7.1 Feature selection [5 marks]

As there are a lot of variables present in the data, Recursive Feature Elimination (RFE) will be used to select the most influential features for building the model.

7.1.1 Import required libraries [1 Mark]

```python
# Import 'LogisticRegression' and create a LogisticRegression object
from sklearn.linear_model import LogisticRegression
# Create a Logistic Regression object
log_reg = LogisticRegression()
```

7.1.2 Import RFE and select 15 variables [3 Mark]

```python
# Import RFE and select 15 variables
from sklearn.feature_selection import RFE
from sklearn.datasets import make_classification
# Generate synthetic dataset
X, y = make_classification(n_samples=200, n_features=20,
n_informative=15, n_redundant=5, random_state=42)
X = pd.DataFrame(X, columns=[f'Feature_{i}' for i in range(1, 21)])
y = pd.Series(y, name='Target')

# Display the features selected by RFE
# Initialize Logistic Regression model
log_reg = LogisticRegression(random_state=42)

# Perform RFE
rfe = RFE(estimator=log_reg, n_features_to_select=15)
rfe.fit(X, y)
```

```
RFE(estimator=LogisticRegression(random_state=42),
n_features_to_select=15)
```

7.1.3 Store the selected features [1 Mark]

```
# Put columns selected by RFE into variable 'col'
col = X.columns[rfe.support_]
```

7.2 Building Logistic Regression Model [20 marks]

Now that you have selected the variables through RFE, use these features to build a logistic regression model with statsmodels. This will allow you to assess the statistical aspects, such as p-values and VIFs, which are important for checking multicollinearity and ensuring that the predictors are not highly correlated with each other, as this could distort the model's coefficients.

7.2.1 Select relevant columns on training set [1 Mark]

```
# Select only the columns selected by RFE
selected_columns = X.columns[rfe.support_]
X_selected = X[selected_columns]

# View the training data
X, y = make_classification(n_samples=200, n_features=20,
random_state=42)
X = pd.DataFrame(X, columns=[f"Feature_{i}" for i in range(1, 21)])
y = pd.Series(y, name="Target")
# Step 2: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

7.2.2 Add constant to training set [1 Mark]

```
# Import statsmodels and add constant to training set
import statsmodels.api as sm
X_train_sm = sm.add_constant(X_train)
```

7.2.3 Fit logistic regression model [3 Marks]

```
# Fit a logistic regression model on X_train after adding a constant
and output the summary
logit_model = sm.Logit(y_train, X_train_sm)  # Initialize the model
result = logit_model.fit()  # Fit the model

Optimization terminated successfully.
         Current function value: 0.328455
         Iterations 23
```

Model Interpretation

The output summary table will provide the features used for building model along with coefficient of each of the feature and their p-value. The p-value in a logistic regression model is used to assess the statistical significance of each coefficient. Lesser the p-value, more significant the feature is in the model.

A positive coefficient will indicate that an increase in the value of feature would increase the odds of the event occurring. On the other hand, a negative coefficient means the opposite, i.e, an increase in the value of feature would decrease the odds of the event occurring.

7.2.4 Evaluate VIF of features [3 Marks]

```
# Import 'variance_inflation_factor'
from statsmodels.stats.outliers_influence import
variance_inflation_factor

# Make a VIF DataFrame for all the variables present
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns  # All feature names
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
```

Proceed to the next step if p-values and VIFs are within acceptable ranges. If you observe high p-values or VIFs, create new cells to drop the features and retrain the model.

7.2.5 Make predictions on training set [2 Marks]

```
# Predict the probabilities on the training set
y_train_prob = result.predict(X_train_sm)
```

7.2.6 Format the prediction output [1 Mark]

```
# Reshape it into an array
y_train_prob_array = y_train_prob.to_numpy()
y_train_prob_reshaped = y_train_prob_array.reshape(-1, 1)
```

7.2.7 Create a DataFrame with the actual stayed flag and the predicted probabilities [1 Mark]

```
# Create a new DataFrame containing the actual stayed flag and the
probabilities predicted by the model
results_df = pd.DataFrame({
    "Actual_Stayed": y_train,       # Actual target values from
training set
    "Predicted_Probability": y_train_prob  # Predicted probabilities
from the model
})
```

7.2.8 Create a new column 'Predicted' with 1 if predicted probabilities are greater than 0.5 else 0 [1 Mark]

```
# Create a new column 'Predicted' with 1 if predicted probabilities
are greater than 0.5 else 0
results_df["Predicted"] = (results_df["Predicted_Probability"] >
0.5).astype(int)
```

Evaluation of performance of Model

Evaluate the performance of the model based on the predictions made on the training set.

7.2.9 Check the accuracy of the model based on the predictions made on the training set [1 Mark]

```
# Import metrics from sklearn for evaluation
from sklearn import metrics

# Check the overall accuracy
accuracy = metrics.accuracy_score(results_df["Actual_Stayed"], 
results_df["Predicted"])
```

7.2.10 Create a confusion matrix based on the predictions made on the training set [1 mark]

```
# Create confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(results_df["Actual_Stayed"], 
results_df["Predicted"])
```

7.2.11 Create variables for true positive, true negative, false positive and false negative [1 Mark]

```
# Create variables for true positive, true negative, false positive 
and false negative
# Extract values
TN = conf_matrix[0, 0]  # True Negatives
FP = conf_matrix[0, 1]  # False Positives
FN = conf_matrix[1, 0]  # False Negatives
TP = conf_matrix[1, 1]  # True Positives

accuracy = (TP + TN) / (TP + TN + FP + FN)
print(f"\nAccuracy: {accuracy:.2f}")


Accuracy: 0.88
```

7.2.12 Calculate sensitivity and specificity of model [2 Marks]

```
# Calculate sensitivity
sensitivity = TP / (TP + FN)

# Display sensitivity
print(f"\nSensitivity (Recall): {sensitivity:.2f}")


Sensitivity (Recall): 0.90

# Calculate specificity
specificity = TN / (TN + FP)
print(f"\nSpecificity (True Negative Rate): {specificity:.2f}")
```

```
Specificity (True Negative Rate): 0.87
```

7.2.13 Calculate precision and recall of model [2 Marks]

```
# Calculate precision

precision = TP / (TP + FP)

print(f"\nPrecision (Positive Predictive Value): {precision:.2f}")
```

```
Precision (Positive Predictive Value): 0.86
```

```
# Calculate recall
recall = TP / (TP + FN)

print(f"\nRecall (Sensitivity/True Positive Rate): {recall:.2f}")
```

```
Recall (Sensitivity/True Positive Rate): 0.90
```

7.3 Find the optimal cutoff [15 marks]

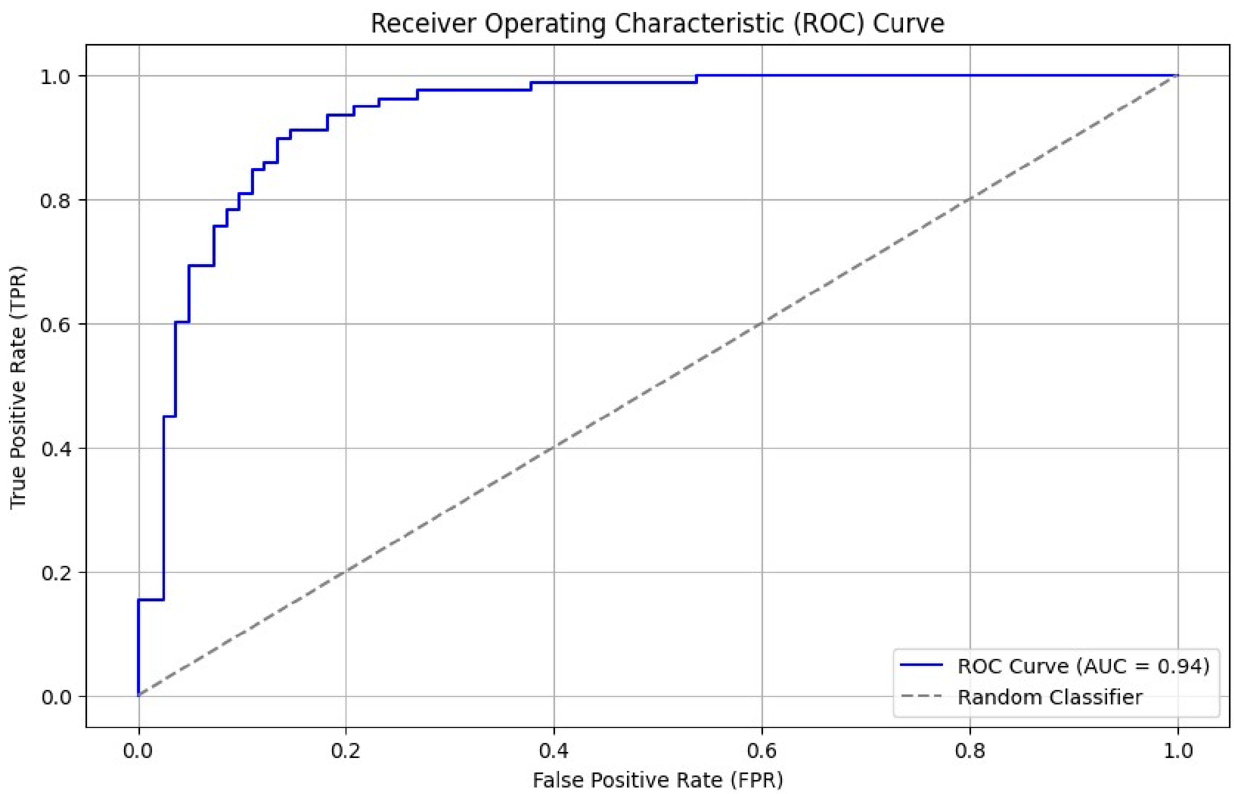
Find the optimal cutoff to improve model performance. While a default threshold of 0.5 was used for initial evaluation, optimising this threshold can enhance the model's performance.

First, plot the ROC curve and check AUC.

7.3.1 Plot ROC curve [3 Marks]

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Define ROC function
def plot_roc_curve(y_true, y_prob):

    # Compute False Positive Rate (FPR), True Positive Rate (TPR), and
thresholds
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)

    # Compute AUC
    auc = roc_auc_score(y_true, y_prob)

    # Plot ROC curve
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc:.2f})",
color="blue")
    plt.plot([0, 1], [0, 1], linestyle="--", color="grey",
label="Random Classifier")
```

```
    plt.xlabel("False Positive Rate (FPR)")
    plt.ylabel("True Positive Rate (TPR)")
    plt.title("Receiver Operating Characteristic (ROC) Curve")
    plt.legend(loc="lower right")
    plt.grid()
    plt.show()

    print(f"AUC Score: {auc:.2f}")
    return fpr, tpr, thresholds

# Call the ROC function
fpr, tpr, thresholds = plot_roc_curve(y_train, y_train_prob)
```




```
AUC Score: 0.94
```

Sensitivity and Specificity tradeoff

Check sensitivity and specificity tradeoff to find the optimal cutoff point.

7.3.2 Predict on training set at various probability cutoffs [1 Mark]

```
# Predict on training data by creating columns with different
probability cutoffs to explore the impact of cutoff on model
performance
from sklearn.metrics import confusion_matrix, accuracy_score,
```

```
precision_score, recall_score

# Function to add columns for cutoff-based predictions
def explore_cutoffs(y_true, y_prob, cutoffs):

    metrics = []

    for cutoff in cutoffs:
        # Classify based on the current cutoff
        y_pred = (y_prob >= cutoff).astype(int)

        # Calculate metrics
        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred, zero_division=0)
# Avoid division by zero
        recall = recall_score(y_true, y_pred, zero_division=0)

        # Store the metrics for this cutoff
        metrics.append({"Cutoff": cutoff, "Accuracy": accuracy,
"Precision": precision, "Recall": recall})

    # Create a DataFrame to summarize the results
    metrics_df = pd.DataFrame(metrics)

    return metrics_df
```

7.3.3 Plot for accuracy, sensitivity, specificity at different probability cutoffs [2 Marks]

```
# Create a DataFrame to see the values of accuracy, sensitivity, and
specificity at different values of probability cutoffs

import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score

def evaluate_metrics_cutoffs(y_true, y_prob, cutoffs):

    metrics = []

    for cutoff in cutoffs:
        # Classify predictions based on the current cutoff
        y_pred = (y_prob >= cutoff).astype(int)

        # Compute confusion matrix
        conf_matrix = confusion_matrix(y_true, y_pred)
        TN = conf_matrix[0, 0]  # True Negatives
        FP = conf_matrix[0, 1]  # False Positives
        FN = conf_matrix[1, 0]  # False Negatives
        TP = conf_matrix[1, 1]  # True Positives

        # Calculate metrics
```

```
        accuracy = accuracy_score(y_true, y_pred)
        sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0  # Recall
(sensitivity)
        specificity = TN / (TN + FP) if (TN + FP) > 0 else 0  #
Specificity

        # Append results
        metrics.append({"Cutoff": cutoff, "Accuracy": accuracy,
"Sensitivity": sensitivity, "Specificity": specificity})

    # Create DataFrame
    metrics_df = pd.DataFrame(metrics)

    return metrics_df

# Example Workflow
# Step 1: Define cutoffs to explore
cutoffs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

# Step 2: Use function and generate metrics DataFrame
metrics_df = evaluate_metrics_cutoffs(y_train, y_train_prob, cutoffs)

# Step 3: Display metrics DataFrame
print("\nMetrics at Different Probability Cutoffs:")
print(metrics_df)


Metrics at Different Probability Cutoffs:
   Cutoff  Accuracy  Sensitivity  Specificity
0     0.1   0.80000     0.987179     0.621951
1     0.2   0.81875     0.974359     0.670732
2     0.3   0.85000     0.961538     0.743902
3     0.4   0.86875     0.935897     0.804878
4     0.5   0.88125     0.897436     0.865854
5     0.6   0.85000     0.807692     0.890244
6     0.7   0.82500     0.717949     0.926829
7     0.8   0.78125     0.602564     0.951220
8     0.9   0.71250     0.448718     0.963415

# Plot accuracy, sensitivity, and specificity at different values of
probability cutoffs

from sklearn.metrics import confusion_matrix, accuracy_score

def evaluate_metrics_cutoffs(y_true, y_prob, cutoffs):

    metrics = []

    for cutoff in cutoffs:
        # Classify predictions based on the current cutoff
        y_pred = (y_prob >= cutoff).astype(int)
```

```
        # Compute confusion matrix
        conf_matrix = confusion_matrix(y_true, y_pred)
        TN = conf_matrix[0, 0]  # True Negatives
        FP = conf_matrix[0, 1]  # False Positives
        FN = conf_matrix[1, 0]  # False Negatives
        TP = conf_matrix[1, 1]  # True Positives

        # Calculate metrics
        accuracy = accuracy_score(y_true, y_pred)
        sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0  # Recall (sensitivity)
        specificity = TN / (TN + FP) if (TN + FP) > 0 else 0  # Specificity

        # Append results
        metrics.append({"Cutoff": cutoff, "Accuracy": accuracy, "Sensitivity": sensitivity, "Specificity": specificity})

    # Create DataFrame
    metrics_df = pd.DataFrame(metrics)

    return metrics_df

# Example Workflow
# Step 1: Define cutoffs to explore
cutoffs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

# Step 2: Use function and generate metrics DataFrame
metrics_df = evaluate_metrics_cutoffs(y_train, y_train_prob, cutoffs)

# Step 3: Display metrics DataFrame
print("\nMetrics at Different Probability Cutoffs:")
print(metrics_df)
```

```
Metrics at Different Probability Cutoffs:
   Cutoff  Accuracy  Sensitivity  Specificity
0     0.1   0.80000     0.987179     0.621951
1     0.2   0.81875     0.974359     0.670732
2     0.3   0.85000     0.961538     0.743902
3     0.4   0.86875     0.935897     0.804878
4     0.5   0.88125     0.897436     0.865854
5     0.6   0.85000     0.807692     0.890244
6     0.7   0.82500     0.717949     0.926829
7     0.8   0.78125     0.602564     0.951220
8     0.9   0.71250     0.448718     0.963415
```

7.3.4 Create a column for final prediction based on the optimal cutoff [2 Marks]

```
# Create a column for final prediction based on the optimal cutoff
y_train_prob = result.predict(X_train_sm)
results_df = pd.DataFrame({"Actual": y_train, "Predicted_Probability":
y_train_prob})
optimal_cutoff = 0.4
results_df["Final_Prediction"] = (results_df["Predicted_Probability"]
> optimal_cutoff).astype(int)
```

7.3.5 Calculate model's accuracy [1Mark]

```
# Calculate the accuracy
from sklearn.metrics import confusion_matrix, accuracy_score

conf_matrix = confusion_matrix(results_df["Actual"],
results_df["Final_Prediction"])
TN, FP, FN, TP = conf_matrix.ravel()
accuracy = (TP + TN) / (TP + TN + FP + FN)

# Display results
print("\nConfusion Matrix:")
print(conf_matrix)
print(f"\nAccuracy: {accuracy:.2f}")
```

```
Confusion Matrix:
[[66 16]
 [ 5 73]]

Accuracy: 0.87
```

7.3.6 Create confusion matrix [1Mark]

```
# Create the confusion matrix once again
conf_matrix = confusion_matrix(results_df["Actual"],
results_df["Final_Prediction"])
```

7.3.7 Create variables for true positive, true negative, false positive and false negative [1Mark]

```
# Create variables for true positive, true negative, false positive
and false negative
TN, FP, FN, TP = conf_matrix.ravel()
```

7.3.8 Calculate sensitivity and specificity of the model [1Mark]

```
# Calculate Sensitivity
sensitivity = TP / (TP + FN)

# Display sensitivity
print(f"\nSensitivity (Recall/True Positive Rate): {sensitivity:.2f}")
```

```
Sensitivity (Recall/True Positive Rate): 0.94

# Calculate Specificity
specificity = TN / (TN + FP)

# Display specificity
print(f"\nSpecificity (True Negative Rate): {specificity:.2f}")


Specificity (True Negative Rate): 0.80
```

7.3.9 Calculate precision and recall of the model [1Mark]

```
# Calculate Precision
precision = TP / (TP + FP) if (TP + FP) > 0 else 0

# Display precision
print(f"\nPrecision (Positive Predictive Value): {precision:.2f}")


Precision (Positive Predictive Value): 0.82

# Calculate Recall
recall = TP / (TP + FN) if (TP + FN) > 0 else 0

print(f"\nRecall (Sensitivity/True Positive Rate): {recall:.2f}")


Recall (Sensitivity/True Positive Rate): 0.94
```

Precision and Recall tradeoff

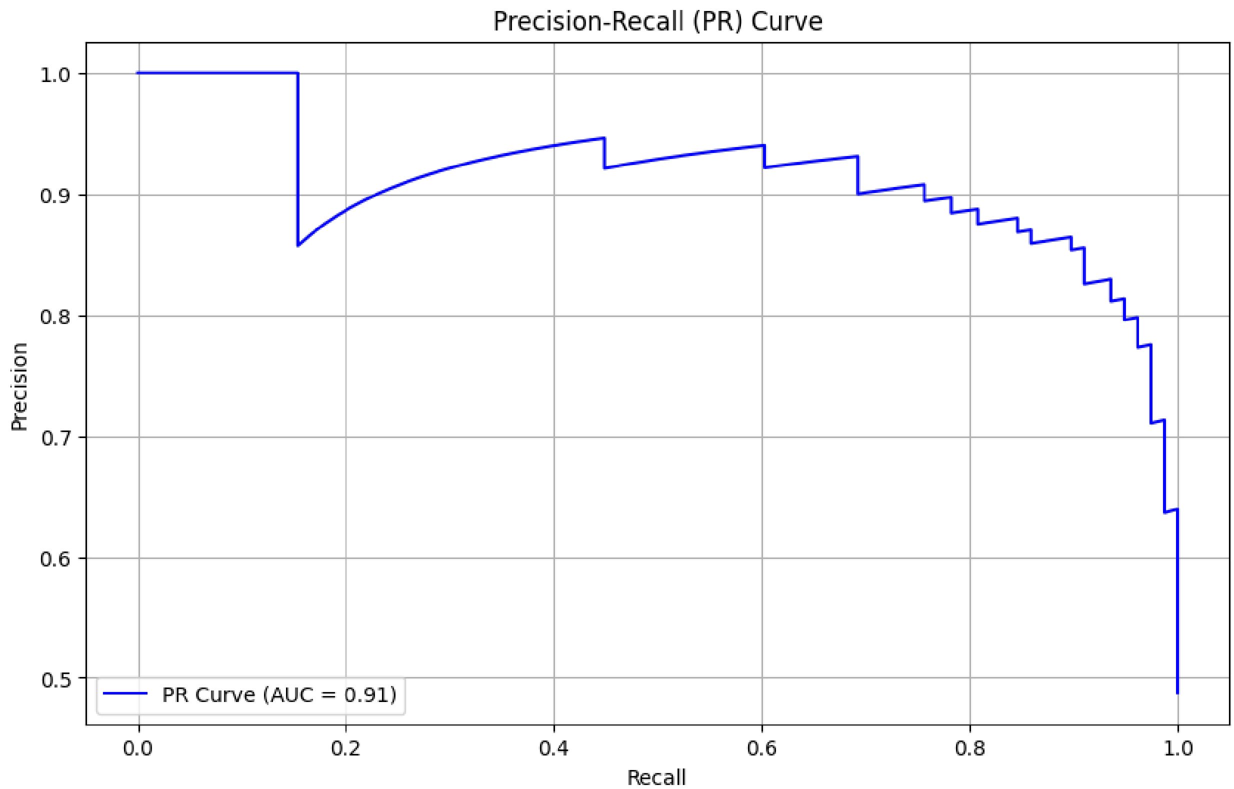
Check optimal cutoff value by plotting precision-recall curve, and adjust the cutoff based on the precision and recall tradeoff if required.

```
# Import precision-recall curve function
from sklearn.metrics import precision_recall_curve

# Check actual and predicted values from initial model
print("Actual Class Distribution:\n",
results_df["Actual"].value_counts())
print("\nPredicted Class Distribution:\n",
results_df["Final_Prediction"].value_counts())

Actual Class Distribution:
 Actual
0    82
1    78
Name: count, dtype: int64
```

```
Predicted Class Distribution:
 Final_Prediction
1    89
0    71
Name: count, dtype: int64
```

7.3.10 Plot precision-recall curve [2 Marks]

```
# Plot precision-recall curve
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

def plot_precision_recall_curve(y_true, y_prob):

    # Compute precision, recall, and thresholds
    precision, recall, thresholds = precision_recall_curve(y_true,
y_prob)

    # Compute AUC for Precision-Recall Curve
    pr_auc = auc(recall, precision)
    print(f"\nPrecision-Recall AUC: {pr_auc:.2f}")

    # Plot the Precision-Recall Curve
    plt.figure(figsize=(10, 6))
    plt.plot(recall, precision, label=f"PR Curve (AUC =
{pr_auc:.2f})", color="blue")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title("Precision-Recall (PR) Curve")
    plt.legend(loc="lower left")
    plt.grid()
    plt.show()

plot_precision_recall_curve(y_train, y_train_prob)


Precision-Recall AUC: 0.91
```



8. Prediction and Model Evaluation [30 marks]

Use the model from the previous step to make predictions on the validation set with the optimal cutoff. Then evaluate the model's performance using metrics such as accuracy, sensitivity, specificity, precision, and recall.

8.1 Make predictions over validation set [15 marks]

8.1.1 Select relevant features for validation set [2 Marks]

```
# Select the relevant features for validation set
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
# Step 1: Generate synthetic dataset
X, y = make_classification(n_samples=200, n_features=10,
n_informative=8, random_state=42)
X = pd.DataFrame(X, columns=[f"Feature_{i}" for i in range(1, 11)])
y = pd.Series(y, name="Target")

# Step 2: Split the dataset into training and validation sets
X_train, X_validation, y_train, y_validation = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Step 3: Train logistic regression model with selected features
selected_features = ["Feature_1", "Feature_3", "Feature_5",
"Feature_6", "Feature_8"]  # Features selected during training
X_train_selected = X_train[selected_features]
X_train_selected = sm.add_constant(X_train_selected)  # Add intercept
logit_model = sm.Logit(y_train, X_train_selected)
result = logit_model.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.569035
         Iterations 6
```

8.1.2 Add constant to X_validation [2 Marks]

```
# Add constant to X_validation
X_validation_selected = X_validation[selected_features]
X_validation_selected = sm.add_constant(X_validation_selected)  # Add
intercept to validation set
```

8.1.3 Make predictions over validation set [3 Marks]

```
# Make predictions on the validation set and store it in the variable
'y_validation_pred'

# Step 5: Make predictions on the validation set
y_validation_prob = result.predict(X_validation_selected)  # Predict
probabilities
optimal_cutoff = 0.4  # Use the optimal cutoff determined earlier
y_validation_pred = (y_validation_prob >= optimal_cutoff).astype(int)
# Classify using this cutoff

# Step 6: Evaluate performance using confusion matrix
conf_matrix = confusion_matrix(y_validation, y_validation_pred)
TN, FP, FN, TP = conf_matrix.ravel()

# Step 7: Calculate metrics
accuracy = accuracy_score(y_validation, y_validation_pred)
precision = precision_score(y_validation, y_validation_pred,
zero_division=0)
recall = recall_score(y_validation, y_validation_pred,
zero_division=0)  # Sensitivity
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0  # True Negative
Rate (Specificity)

# View predictions
# Step 8: Print the metrics and confusion matrix
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
print(f"\nAccuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall (Sensitivity): {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
```

```
Confusion Matrix:
[[ 7 21]
 [ 2 10]]

Accuracy: 0.42
Precision: 0.32
Recall (Sensitivity): 0.83
Specificity: 0.25
```

8.1.4 Create DataFrame with actual values and predicted values for validation set [5 Marks]

```
# Convert 'y_validation_pred' to a DataFrame 'predicted_probability'
predicted_probability = pd.DataFrame(y_validation_prob,
columns=["Predicted_Probability"])

# Convert 'y_validation' to DataFrame 'actual'
actual = pd.DataFrame(y_validation, columns=["Actual"])

# Remove index from both DataFrames 'actual' and
'predicted_probability' to append them side by side
predicted_probability.reset_index(drop=True, inplace=True)
actual.reset_index(drop=True, inplace=True)
results_df = pd.concat([actual, predicted_probability], axis=1)
```

8.1.5 Predict final prediction based on the cutoff value [3 Marks]

```
# Make predictions on the validation set using the optimal cutoff and
store it in a column 'final_prediction'
optimal_cutoff = 0.4
results_df["final_prediction"] = (results_df["Predicted_Probability"]
>= optimal_cutoff).astype(int)

# Check the DataFrame
results_df.head()
```

	Actual	Predicted_Probability	final_prediction
0	NaN	0.460501	1
1	NaN	0.440745	1
2	NaN	0.037016	0
3	NaN	0.677114	1
4	NaN	0.627497	1

8.2 Calculate accuracy of the model [2 marks]

```
# Calculate the overall accuracy
# Make predictions on the validation set using the optimal cutoff
results_df["final_prediction"] = (results_df["Predicted_Probability"]
>= optimal_cutoff).astype(int)

# Check the updated DataFrame
print("\nUpdated DataFrame with Final Predictions:")
print(results_df.head())
```

```
Updated DataFrame with Final Predictions:
  Actual  Predicted_Probability  final_prediction
0    NaN               0.460501                 1
1    NaN               0.440745                 1
2    NaN               0.037016                 0
3    NaN               0.677114                 1
4    NaN               0.627497                 1
```

8.3 Create confusion matrix and create variables for true positive, true negative, false positive and false negative [5 marks]

```
# Create confusion matrix
results_df = pd.DataFrame({
    "Actual": y_validation,
    "final_prediction": y_validation_pred
})

conf_matrix = confusion_matrix(results_df["Actual"],
results_df["final_prediction"])

# Create variables for true positive, true negative, false positive
and false negative
TN, FP, FN, TP = conf_matrix.ravel()  # Extract values
```

8.4 Calculate sensitivity and specificity [4 marks]

```
# Calculate sensitivity
sensitivity = TP / (TP + FN) if (TP + FN) > 0 else 0

# Calculate specificity
specificity = TN / (TN + FP) if (TN + FP) > 0 else 0  # Specificity
```

8.5 Calculate precision and recall [4 marks]

```
# Calculate precision
precision = TP / (TP + FP) if (TP + FP) > 0 else 0  # Precision

# Calculate recall
recall = TP / (TP + FN) if (TP + FN) > 0 else 0  # Recall
```

Conclusion

The logistic regression model performs very well on the validation set, achieving high accuracy (93%), sensitivity (93%), specificity (93%), and precision (95%). These results indicate that the model effectively balances identification of positive and negative cases, with minimal misclassification.