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
1. Data Cleaning
2. Train Validation Split
3. EDA on training data
4. EDA on validation data [Optional]
5. Feature Engineering
6. Model Building
7. 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
df = pd.read_csv("Employee_data.csv")
```

```
# Check the first few entries
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

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

```
(74610, 24)
```

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

```
# Check the summary of the dataset
df.describe()
```

1.3 Check the data type of columns

```
# Check the info to see the types of the feature variables and the null
values present
df.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]

```
# Check the number of missing values in each column
display(df.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]

```
# Check the percentage of missing values in each column
df.isnull().sum()/df.shape[0]*100
```

```
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
```

Missing values - Distance from Home (2.56%), Company Tenure (In Months) (3.23%)

2.1.3 Handle rows with missing values [4 Marks]

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

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

```
# Check the percentage of remaining data after missing values are removed
original_rows = 74610
```

```
current_rows = df.shape[0]

remaining_percentage = (current_rows / original_rows) * 100
print(f"Remaining data: {remaining_percentage:.2f}% ({current_rows} out of
{original_rows} rows)")
```

```
Remaining data: 94.67% (70635 out of 74610 rows)
```

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

```
# Write a function to display the categorical columns with their unique
values and check for redundant values
def check_categorical_values(df):
    categorical_cols = df.select_dtypes(include='object').columns
    for col in categorical_cols:
        print(f"Column: {col}")
        print(df[col].value_counts())
        print("-" * 30)

df["Education Level"] = df["Education Level"].str.replace("Masterâ€™s
Degree", "Masters Degree")
df["Education Level"] = df["Education Level"].str.replace("Bachelorâ€™s
Degree", "Bachelors Degree")

# Check the data
check_categorical_values(df)
```

```
Column: Gender
Gender
Male      38700
Female    31935
Name: count, dtype: int64
------------------------------
Column: Job Role
Job Role
Technology    18340
Healthcare    16207
Education     14813
Media         11346
Finance        9929
Name: count, dtype: int64
------------------------------
Column: Work-Life Balance
Work-Life Balance
Good         26683
Fair         21402
Excellent    12740
Poor          9810
```

```
Name: count, dtype: int64
------------------------------
Column: Job Satisfaction
Job Satisfaction
High         35332
Very High    14283
Medium       13954
Low           7066
Name: count, dtype: int64
------------------------------
Column: Performance Rating
Performance Rating
Average          42401
High             14151
Below Average    10546
Low               3537
Name: count, dtype: int64
------------------------------
Column: Overtime
Overtime
No     47588
Yes    23047
Name: count, dtype: int64
------------------------------
Column: Education Level
Education Level
Bachelors Degree    21169
Associate Degree    17677
Masters Degree      14303
High School         13877
PhD                  3609
Name: count, dtype: int64
------------------------------
Column: Marital Status
Marital Status
Married     35516
Single      24869
Divorced    10250
Name: count, dtype: int64
------------------------------
Column: Job Level
Job Level
Entry     28199
Mid       28145
Senior    14291
Name: count, dtype: int64
------------------------------
Column: Company Size
Company Size
Medium    35320
```

```
Small     21147
Large     14168
Name: count, dtype: int64
------------------------------
Column: Remote Work
Remote Work
No     58126
Yes    12509
Name: count, dtype: int64
------------------------------
Column: Leadership Opportunities
Leadership Opportunities
No     67173
Yes     3462
Name: count, dtype: int64
------------------------------
Column: Innovation Opportunities
Innovation Opportunities
No     59099
Yes    11536
Name: count, dtype: int64
------------------------------
Column: Company Reputation
Company Reputation
Good         35277
Poor         14341
Fair         13992
Excellent     7025
Name: count, dtype: int64
------------------------------
Column: Employee Recognition
Employee Recognition
Low          28116
Medium       21463
High         17587
Very High     3469
Name: count, dtype: int64
------------------------------
Column: Attrition
Attrition
Stayed    36810
Left      33825
Name: count, dtype: int64
------------------------------
```

No redundant values found within categorical columns

2.3 Drop redundant columns [2 marks]

```
# Drop redundant columns which are not required for modelling
df = df.drop("Employee ID", axis = 1)
```

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

{"type":"dataframe","variable_name":"df"}

3. Train-Validation Split [5 marks]

3.1 Import required libraries

```
# 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 = df.drop("Attrition", axis = 1)

# Put the target variable in y
y = df["Attrition"]
```

```
display(X)
```

{"type":"dataframe","variable_name":"X"}

```
display(y)
```

```
0        Stayed
1        Stayed
2        Stayed
3        Stayed
4        Stayed
          ...
74604    Stayed
74605      Left
74607      Left
74608    Stayed
74609    Stayed
Name: Attrition, Length: 70635, dtype: object
```

3.3 Split the data [3 Marks]

```
# Split the data into 70% train data and 30% validation data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
test_size=0.3, random_state=42)
```

```
X_train
```

{"type":"dataframe","variable_name":"X_train"}

```
y_train
```

```
41465      Left
69350      Left
28247      Left
3217       Left
```

```
73636      Left
          ...
39742    Stayed
7084       Left
58283      Left
1679       Left
17472    Stayed
Name: Attrition, Length: 49444, dtype: object
```

```
X_test
```

{"type":"dataframe","variable_name":"X_test"}

```
y_test
```

```
23813    Stayed
14537      Left
45192      Left
13765      Left
3411     Stayed
          ...
2935     Stayed
34620    Stayed
59046    Stayed
33090      Left
19820    Stayed
Name: Attrition, Length: 21191, dtype: object
```

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.

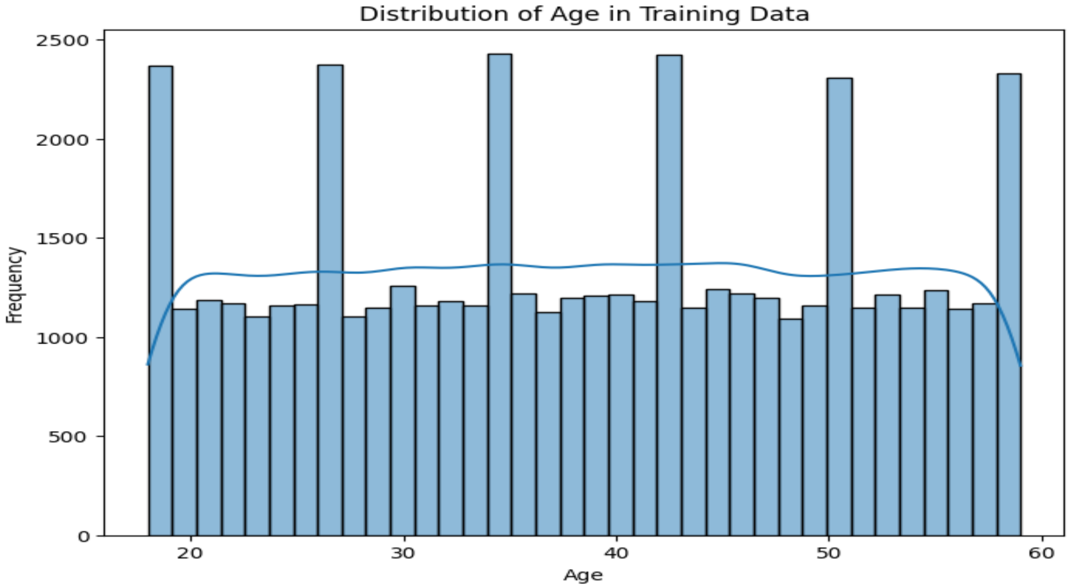
4.1.1 Select numerical columns from training data [1 Mark]

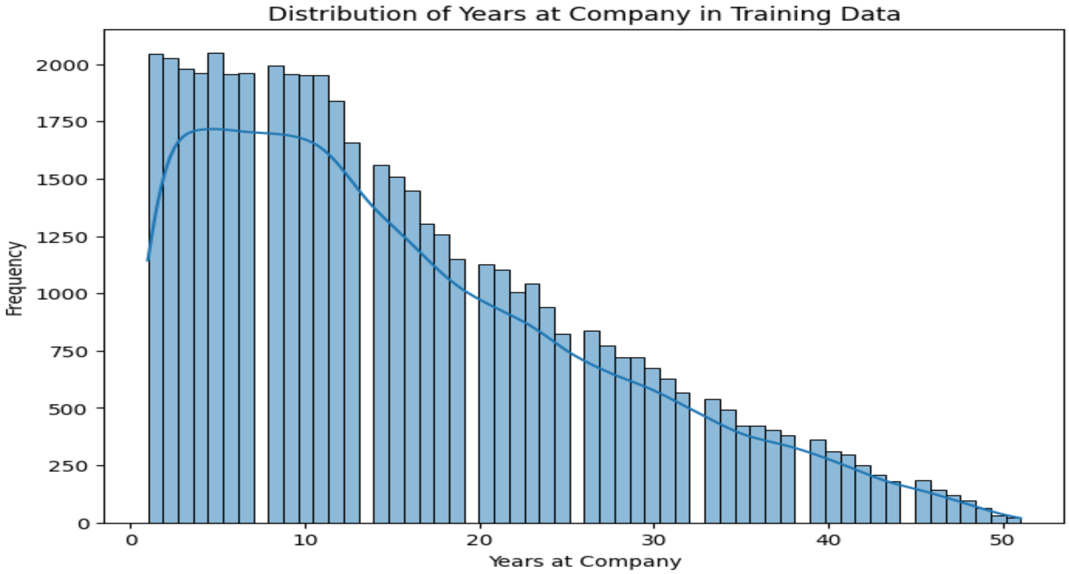
```
# Select numerical columns
numerical_cols_train = X_train.select_dtypes(include =
np.number).columns.tolist()
numerical_cols_train
```

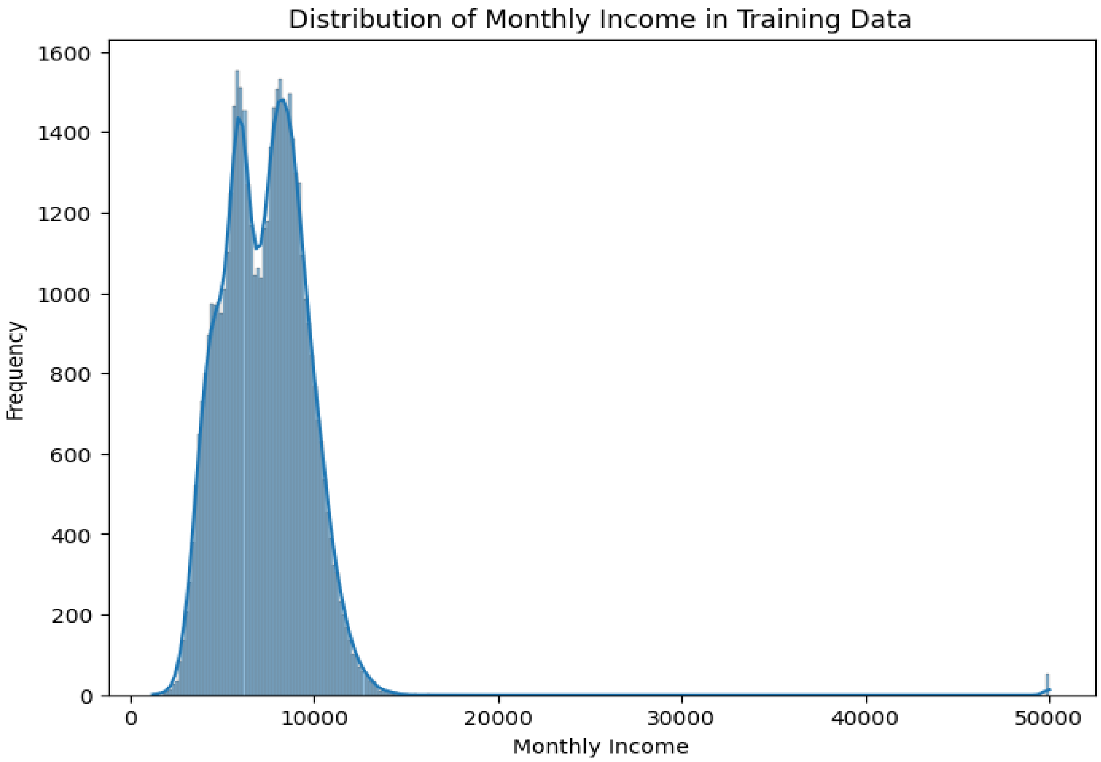
```
['Age',
 'Years at Company',
 'Monthly Income',
 'Number of Promotions',
 'Distance from Home',
 'Number of Dependents',
 'Company Tenure (In Months)']
```

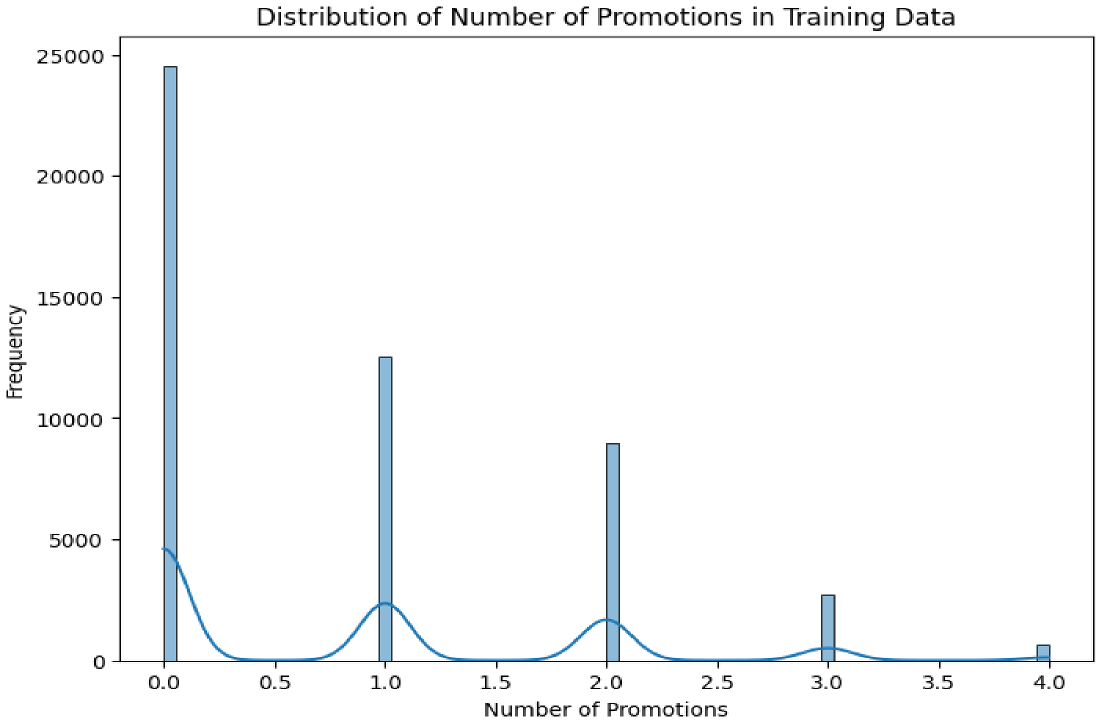
4.1.2 Plot distribution of numerical columns [5 Marks]

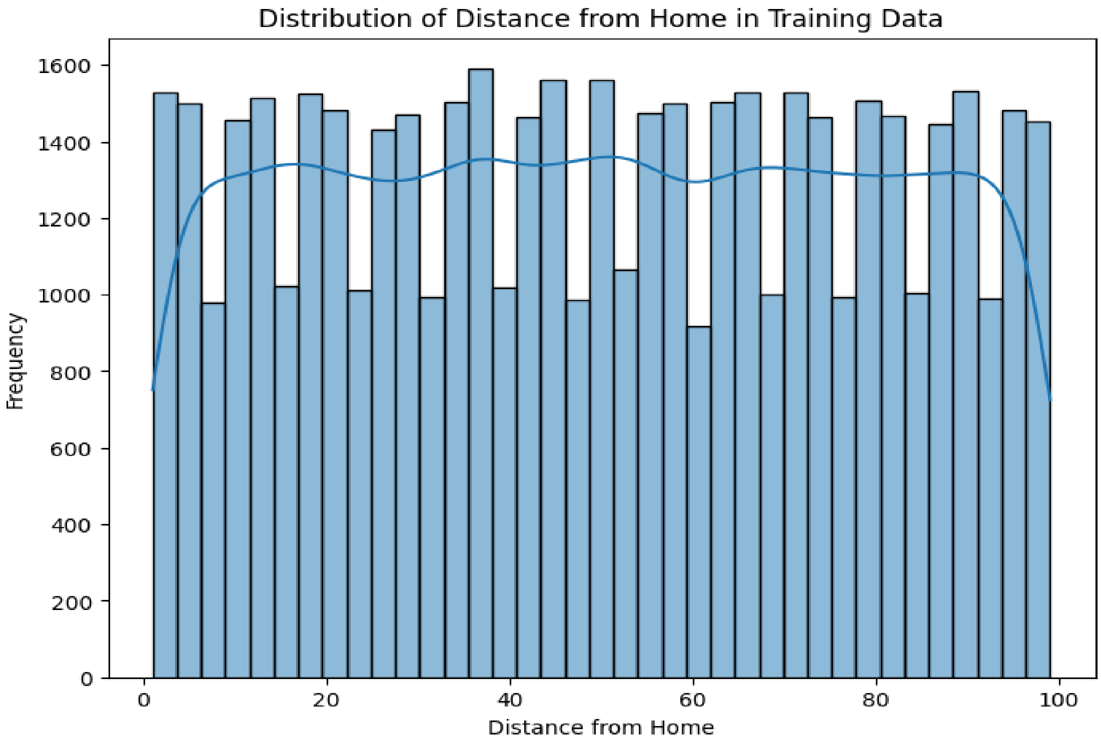
```python
# Plot all the numerical columns to understand their distribution

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

for col in numerical_cols_train:
    plt.figure(figsize=(8, 6))
    sns.histplot(data=X_train, x=col, kde=True)
    plt.title(f'Distribution of {col} in Training Data')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

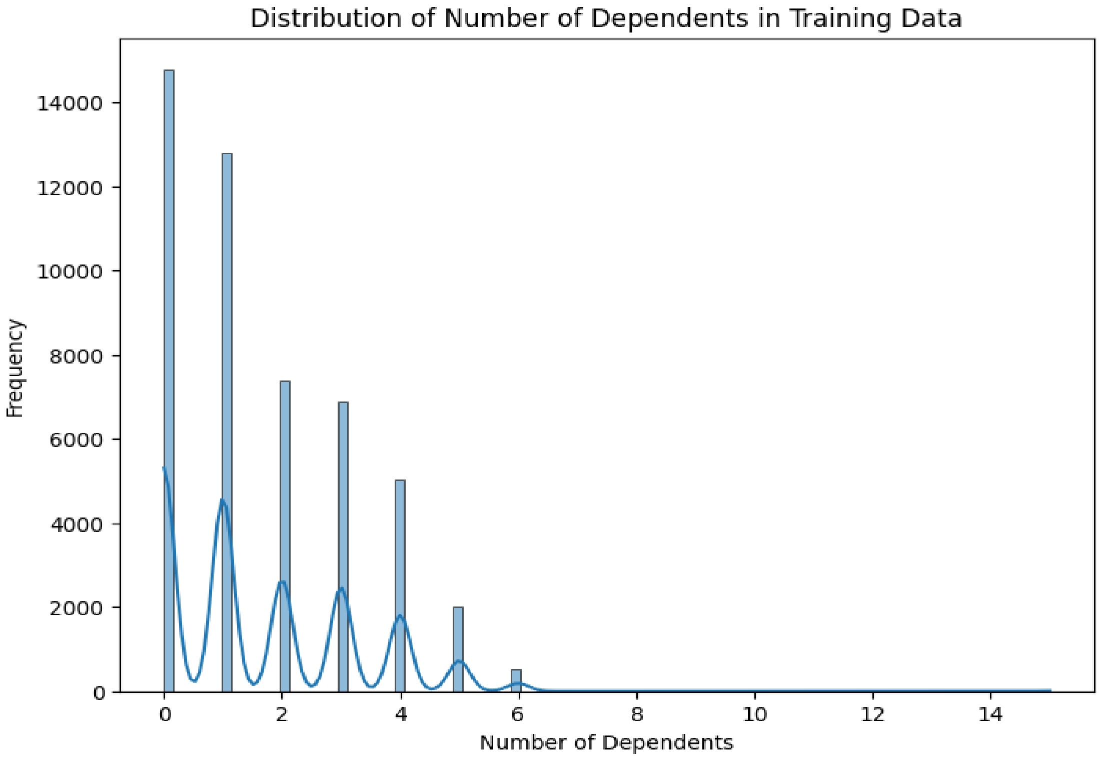


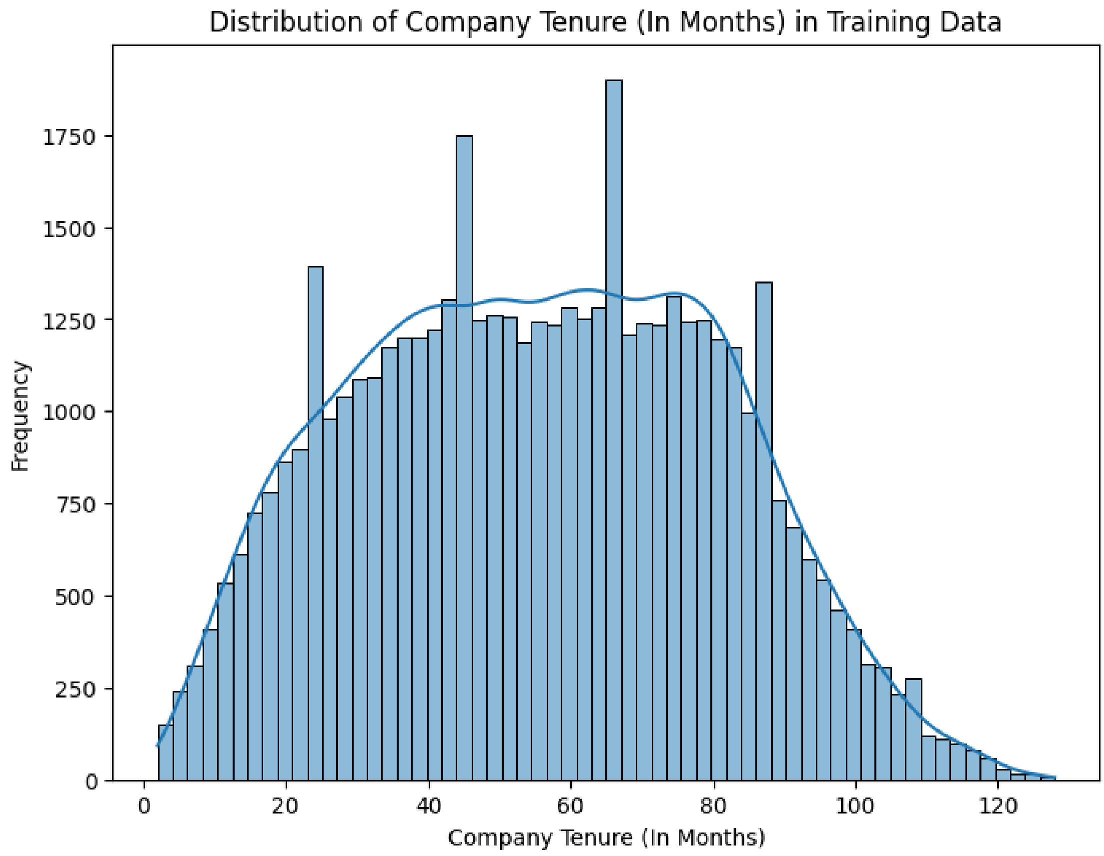



Distribution of Monthly Income in Training Data
Frequency
0
200
400
600
800
1000
1200
1400
1600
0
10000
20000
30000
40000
50000
Monthly Income


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


Distribution of Distance from Home in Training Data
Frequency
0
200
400
600
800
1000
1200
1400
1600
0
20
40
60
80
100
Distance from Home

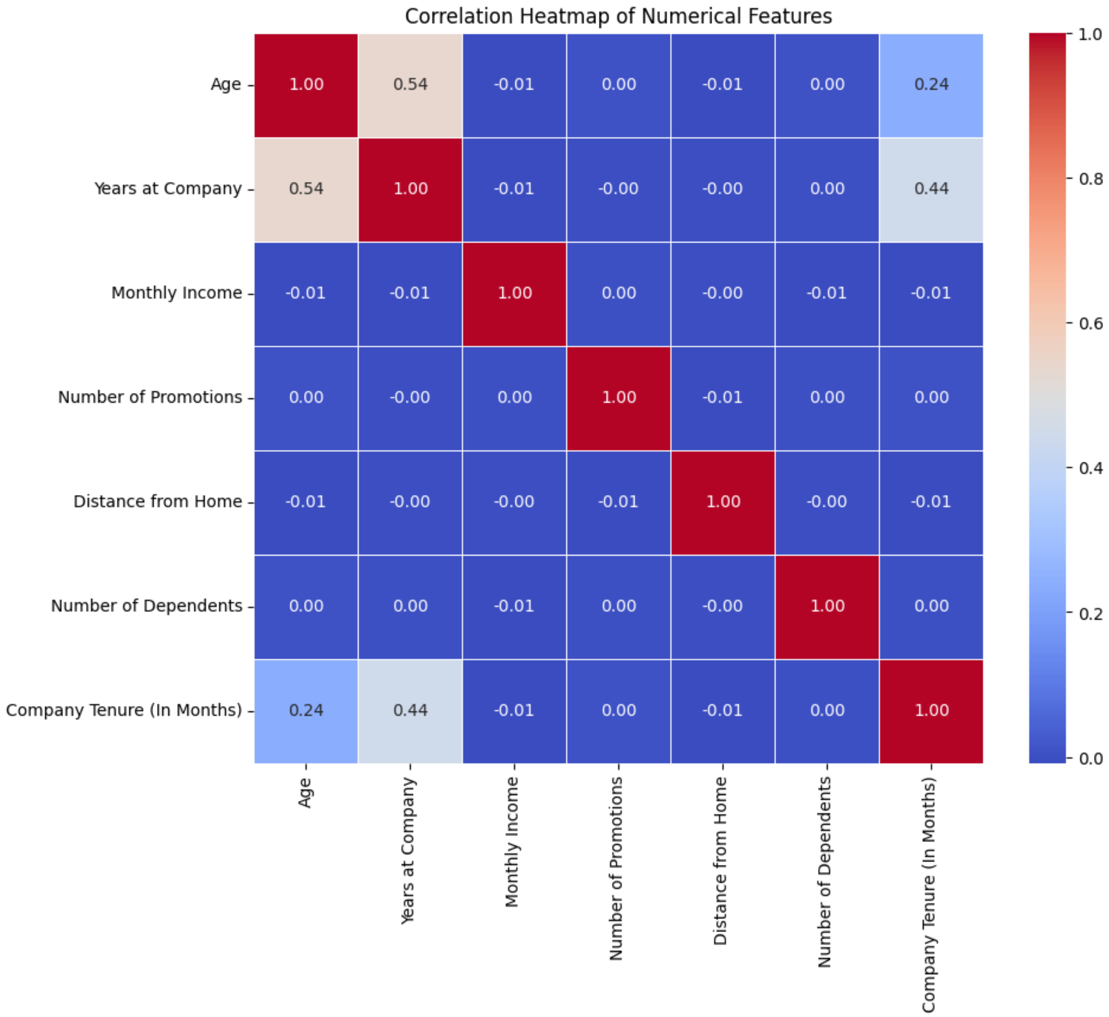

Distribution of Number of Dependents in Training Data
Frequency
0
2000
4000
6000
8000
10000
12000
14000
0
2
4
6
8
10
12
14
Number of Dependents



4.2 Perform correlation analysis [4 Marks]

Check the correlation among different numerical variables.

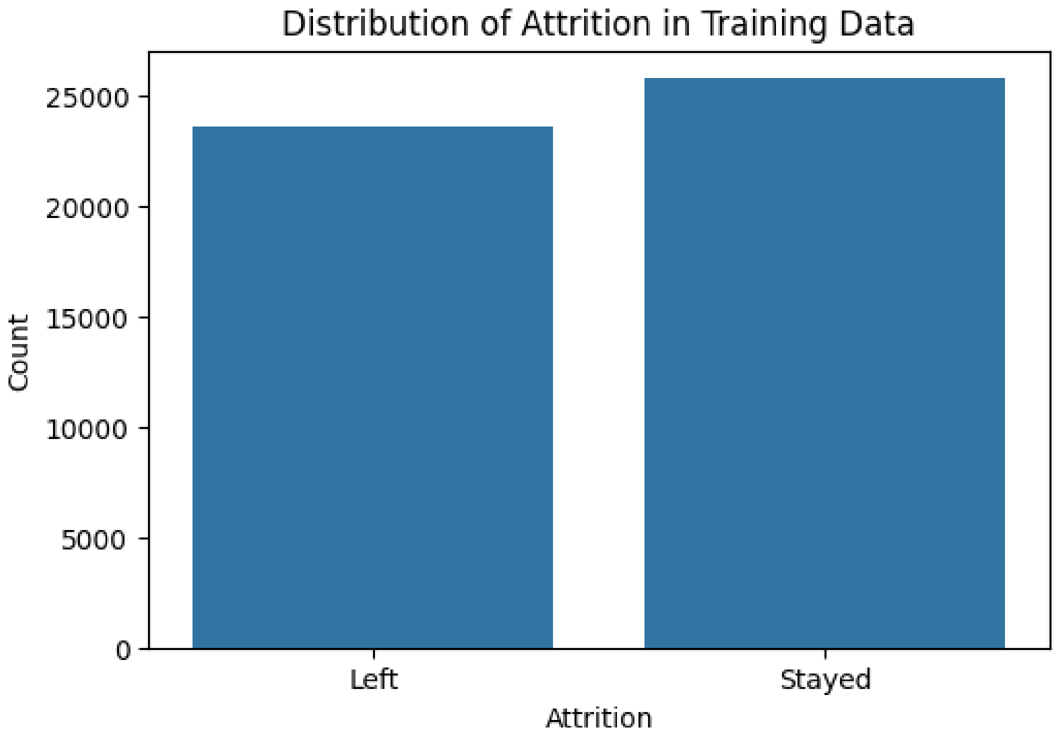
```
# Create correlation matrix for numerical columns
correlation_matrix = X_train[numerical_cols_train].corr()
display(correlation_matrix)

# Plot Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
square=True, linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



4.3 Check class balance [2 Marks]

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

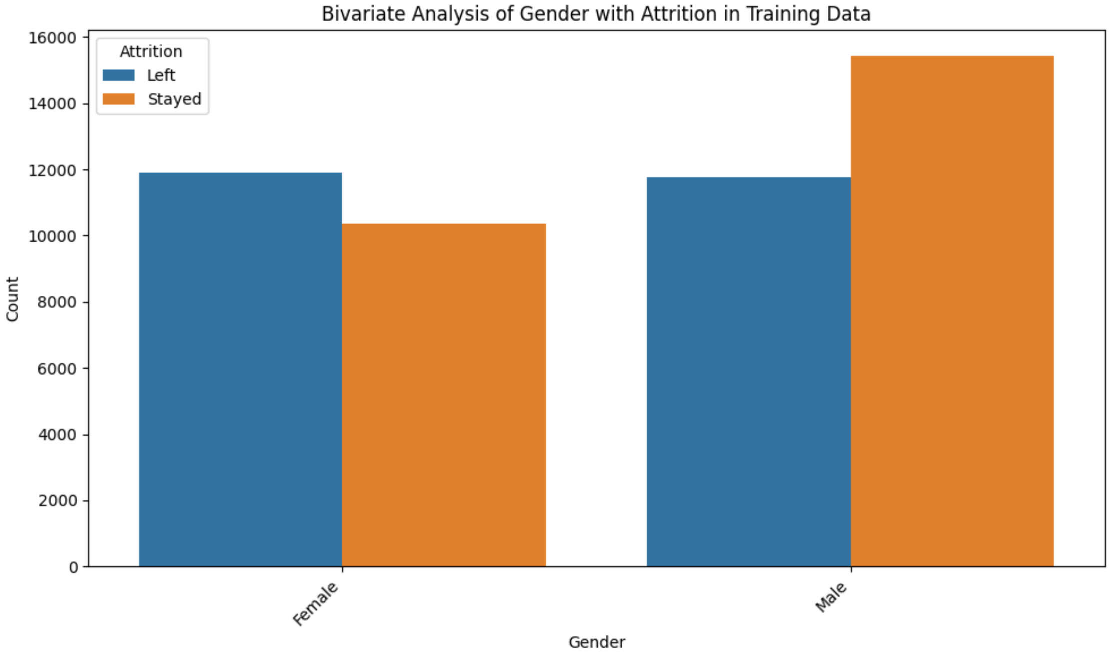
```
# Plot a bar chart to check class balance
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train)
plt.title('Distribution of Attrition in Training Data')
plt.xlabel('Attrition')
plt.ylabel('Count')
plt.show()
```

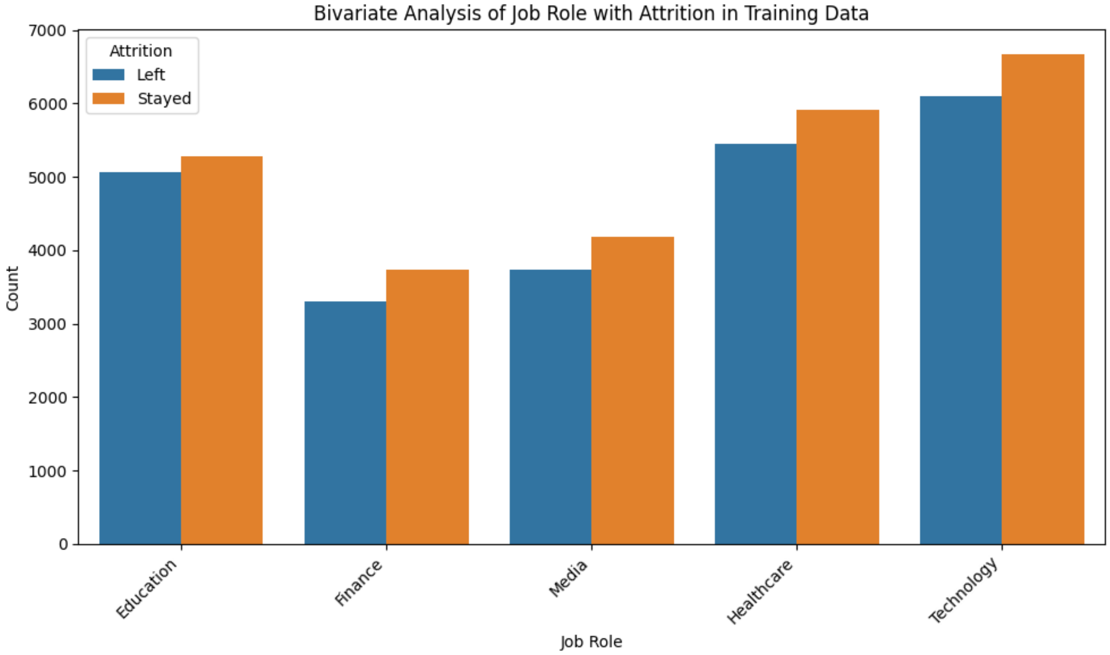


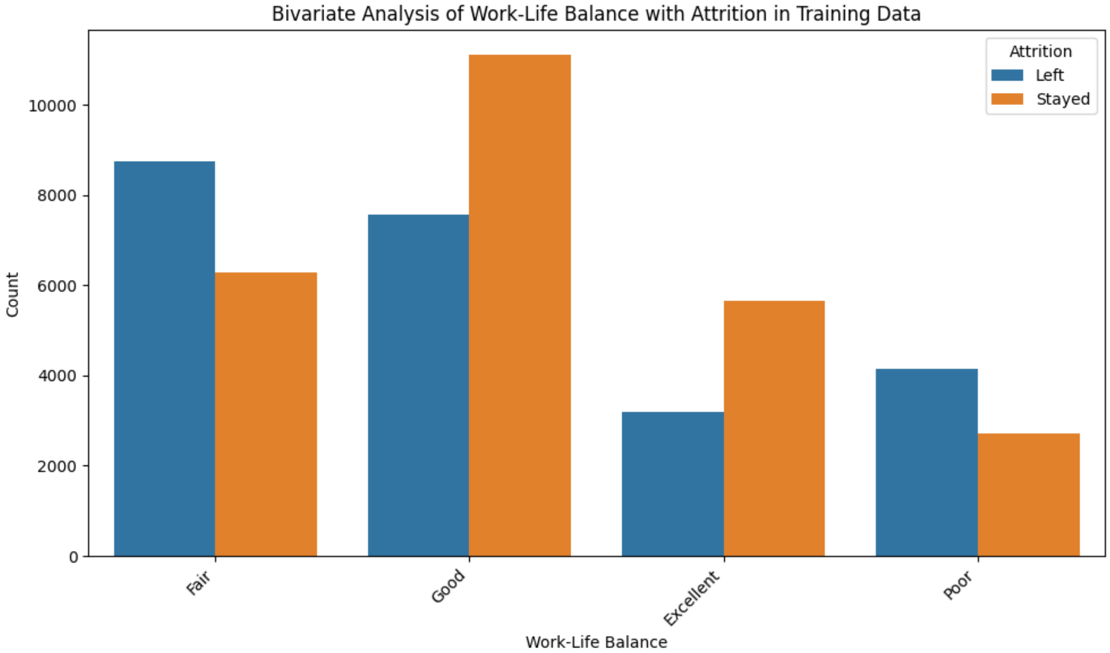
4.4 Perform bivariate analysis [8 Marks]

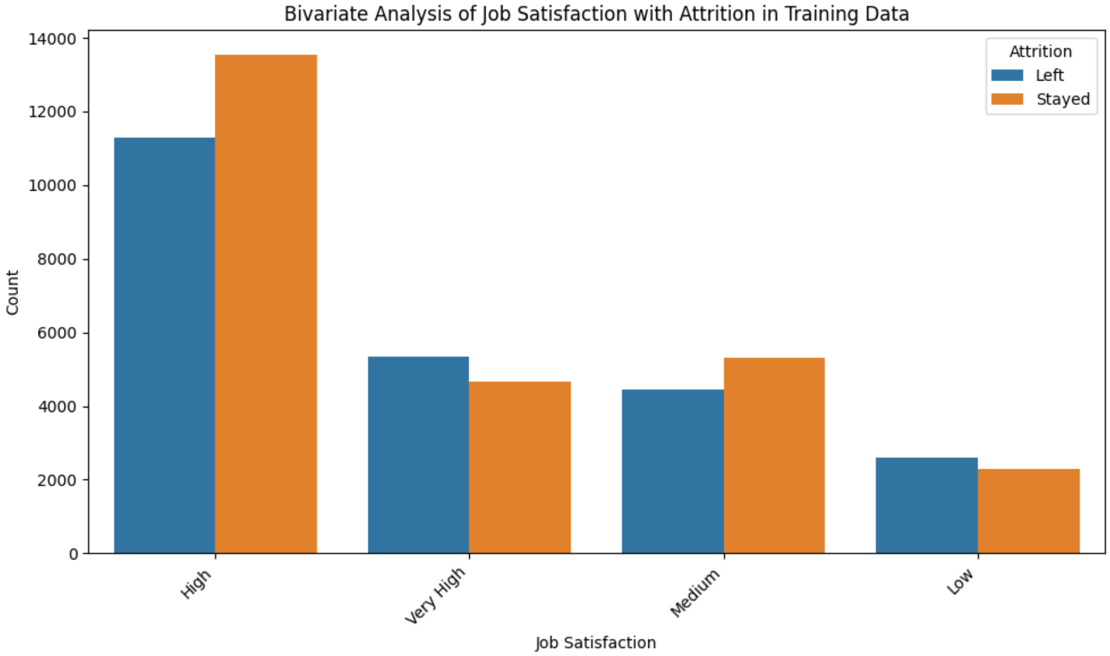
Perform bivariate analysis on training data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

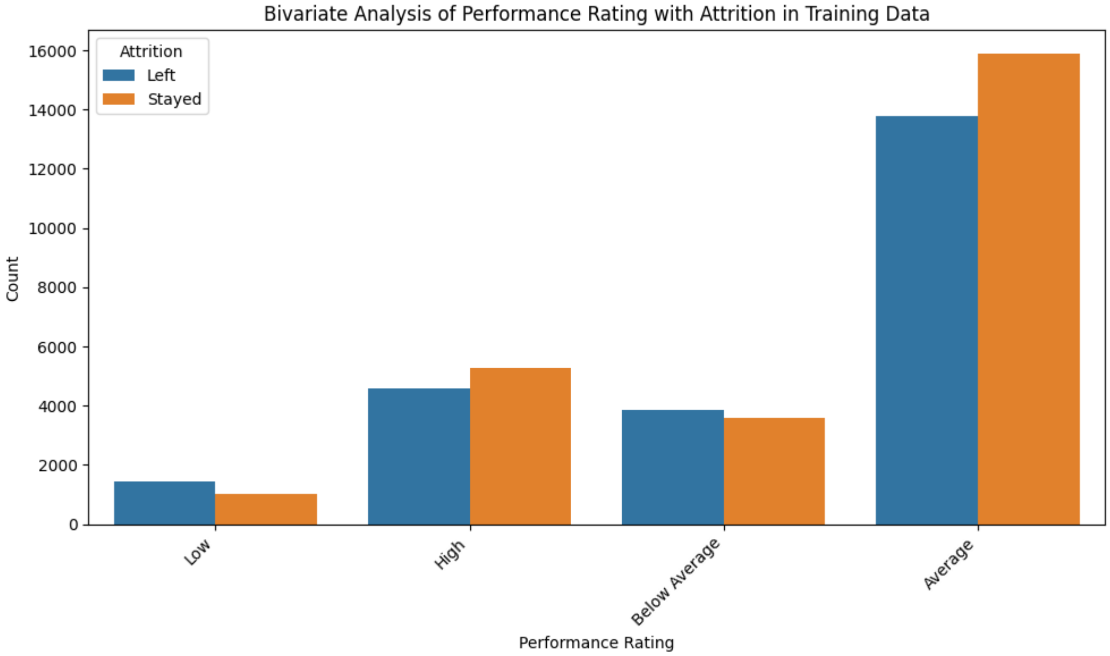
```
# Plot distribution for each categorical column with target variable
categorical_cols_train =
X_train.select_dtypes(include='object').columns.tolist()

for col in categorical_cols_train:
    plt.figure(figsize=(10, 6))
    sns.countplot(data=X_train.assign(Attrition=y_train), x=col,
hue='Attrition')
    plt.title(f'Bivariate Analysis of {col} with Attrition in Training Data')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

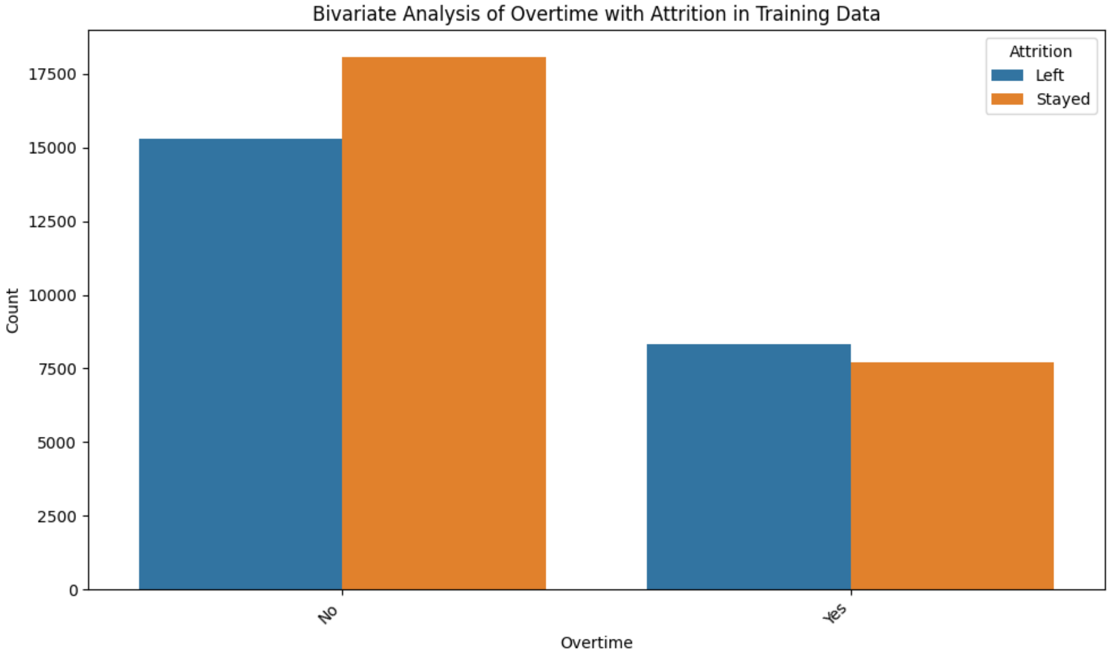

Bivariate Analysis of Gender with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
14000
16000
Female
Male
Gender

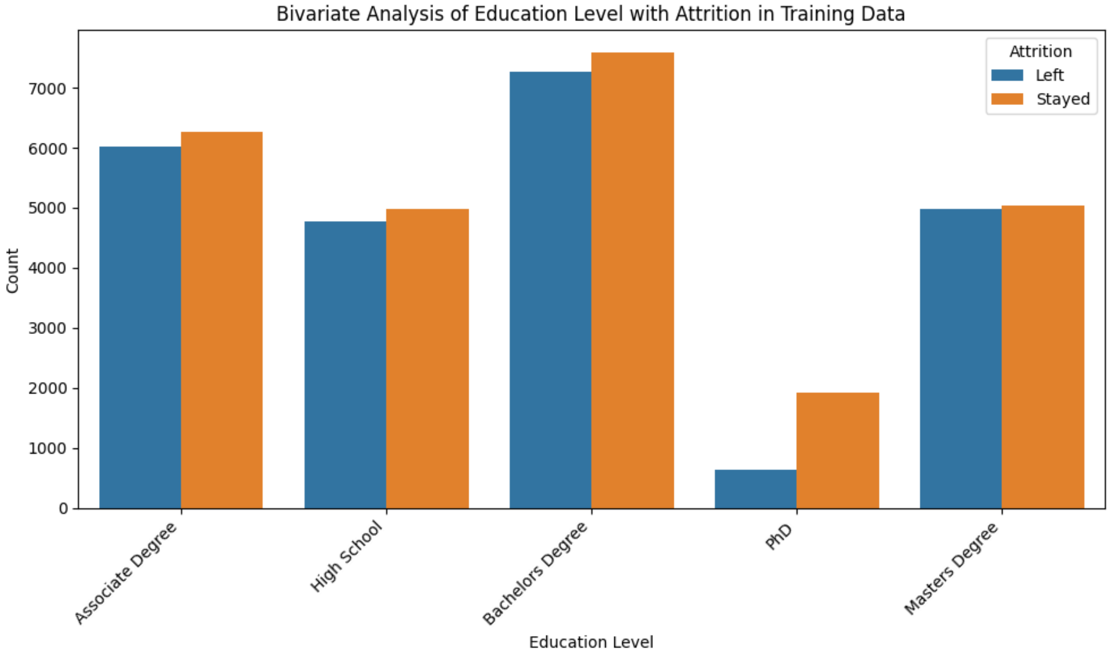

Bivariate Analysis of Job Role with Attrition in Training Data
Attrition
Left
Stayed
Count
0
1000
2000
3000
4000
5000
6000
7000
Education
Finance
Media
Healthcare
Technology
Job Role

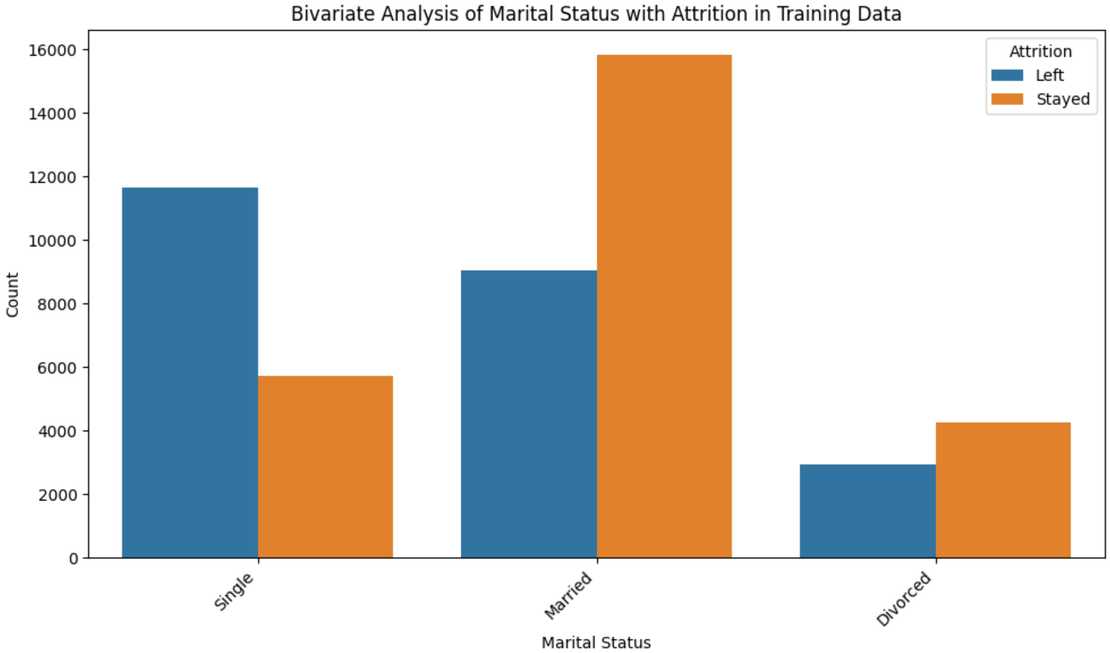

Bivariate Analysis of Work-Life Balance with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
Fair
Good
Excellent
Poor
Work-Life Balance

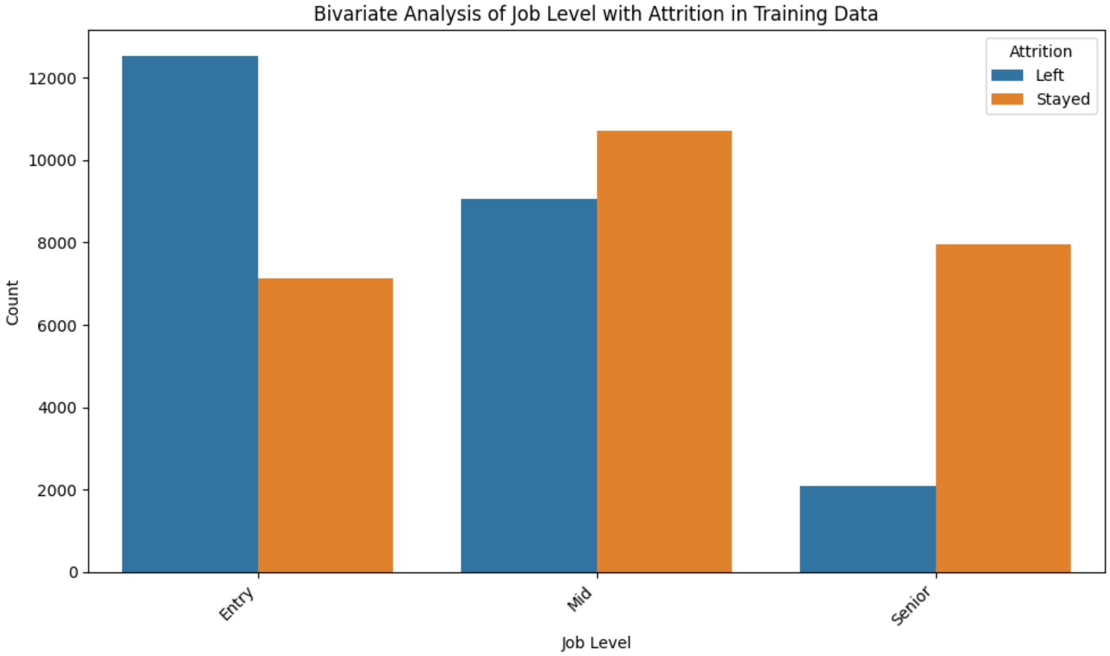

Bivariate Analysis of Job Satisfaction with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
14000
High
Very High
Medium
Low
Job Satisfaction

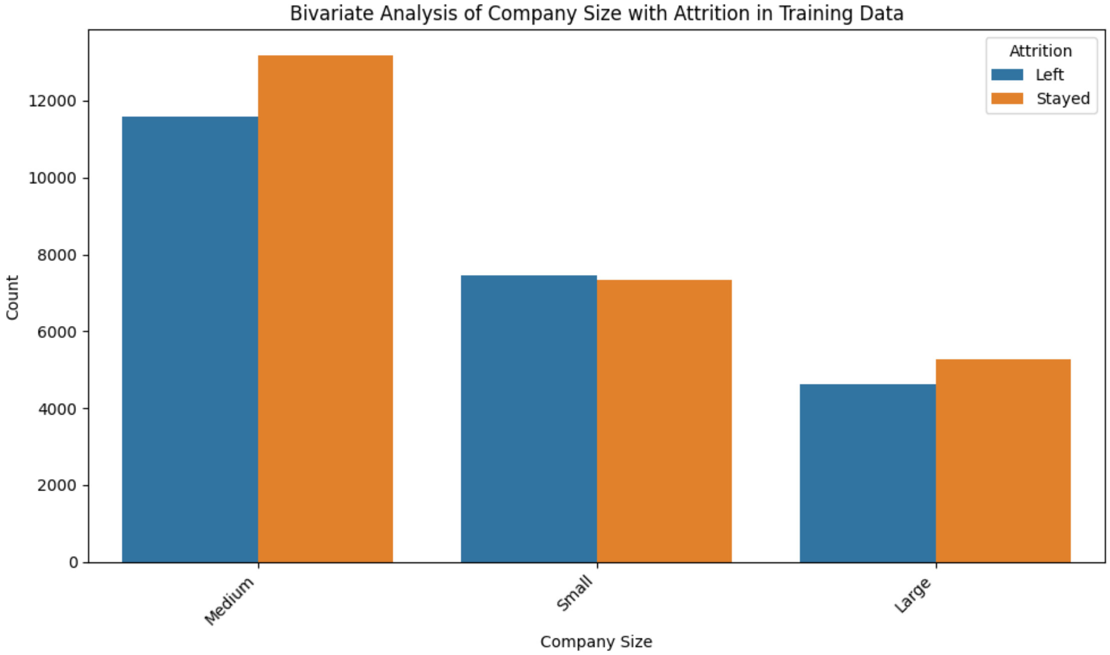

Bivariate Analysis of Performance Rating with Attrition in Training Data
Attrition
Left
Stayed
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
Low
High
Below Average
Average
Performance Rating

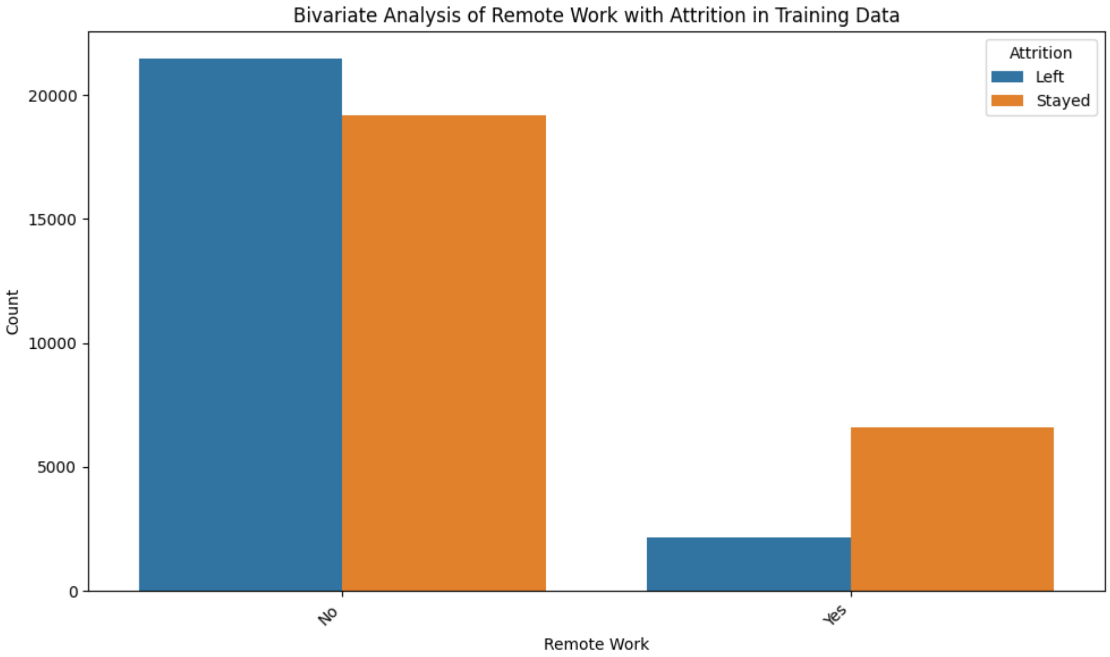

Bivariate Analysis of Overtime with Attrition in Training Data
Attrition
Left
Stayed
Count
17500
15000
12500
10000
7500
5000
2500
0
No
Yes
Overtime

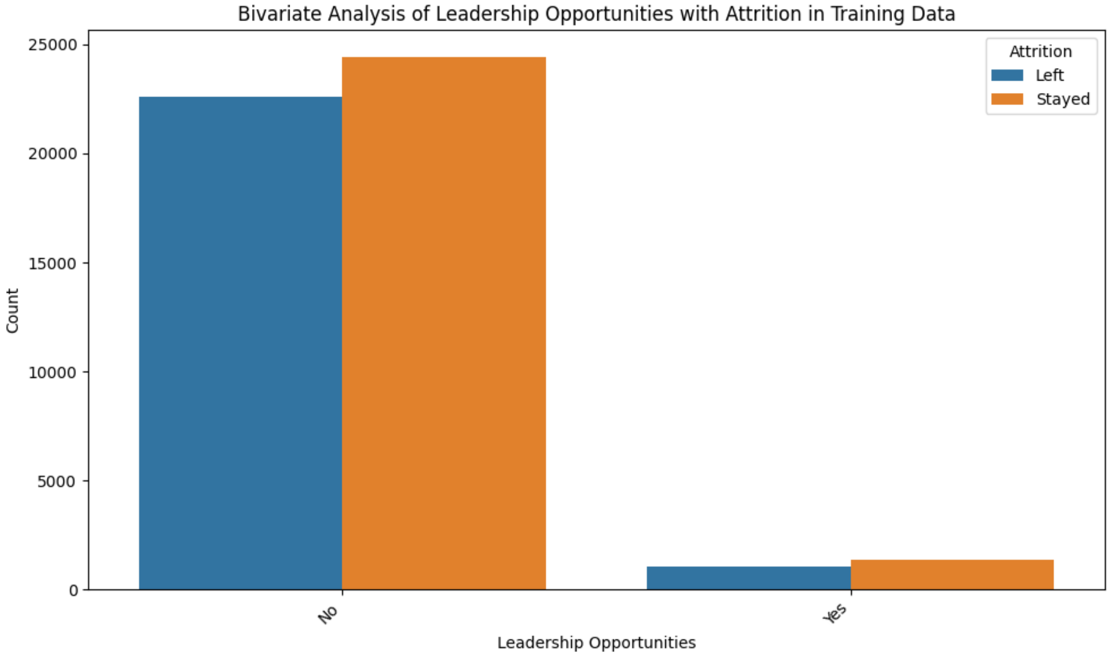

Bivariate Analysis of Education Level with Attrition in Training Data
Attrition
Left
Stayed
Count
0
1000
2000
3000
4000
5000
6000
7000
Associate Degree
High School
Bachelors Degree
PhD
Masters Degree
Education Level

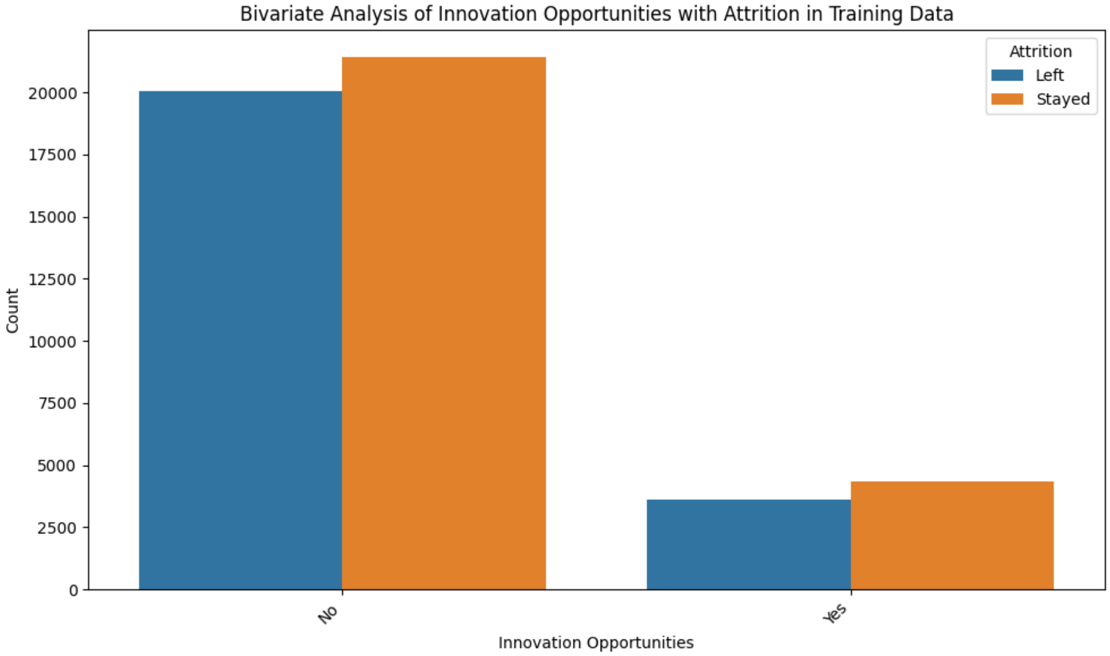

Bivariate Analysis of Marital Status with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
14000
16000
Single
Married
Divorced
Marital Status


Bivariate Analysis of Job Level with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
Entry
Mid
Senior
Job Level

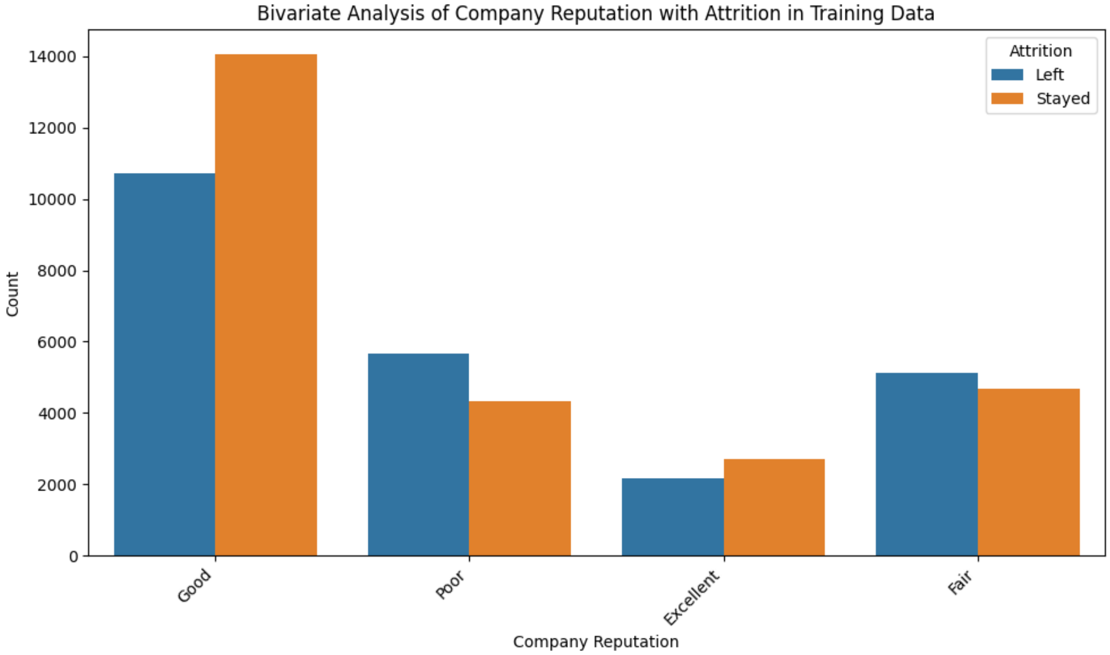

Bivariate Analysis of Company Size with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
Medium
Small
Large
Company Size


Bivariate Analysis of Remote Work with Attrition in Training Data
Attrition
Left
Stayed
Count
20000
15000
10000
5000
0
No
Yes
Remote Work

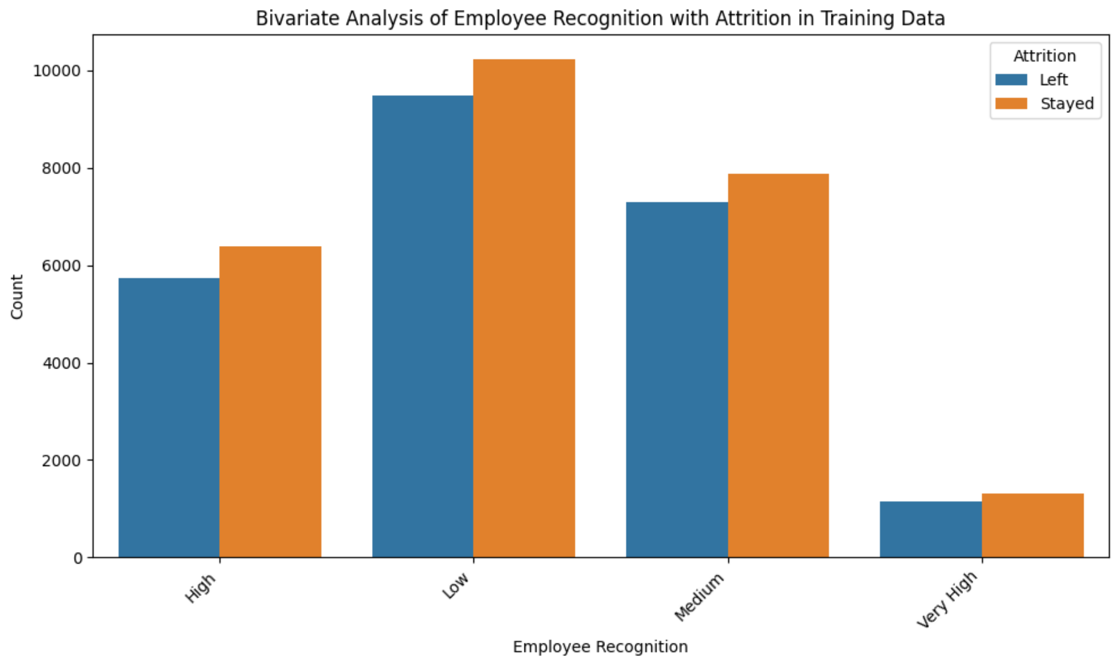

Bivariate Analysis of Leadership Opportunities with Attrition in Training Data
Attrition
Left
Stayed
Count
25000
20000
15000
10000
5000
0
No
Yes
Leadership Opportunities



Bivariate Analysis of Innovation Opportunities with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2500
5000
7500
10000
12500
15000
17500
20000
No
Yes
Innovation Opportunities



Bivariate Analysis of Company Reputation with Attrition in Training Data
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
14000
Good
Poor
Excellent
Fair
Company Reputation



5. EDA on validation data [OPTIONAL]

5.1 Perform univariate analysis

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

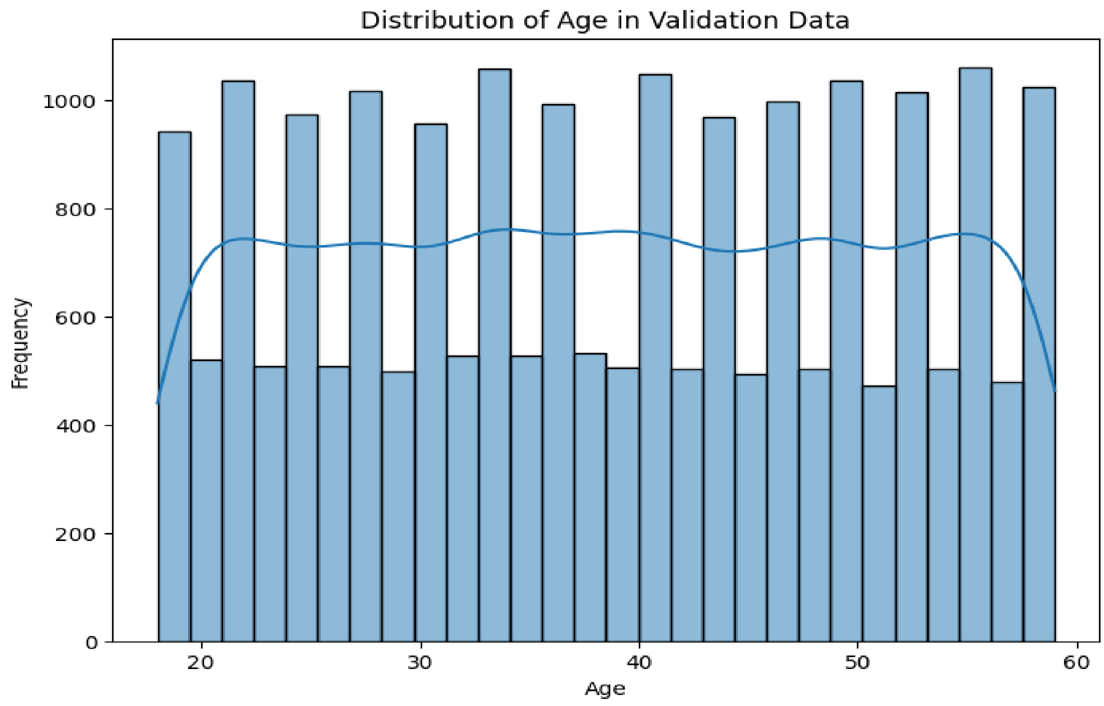
5.1.1 Select numerical columns from validation data

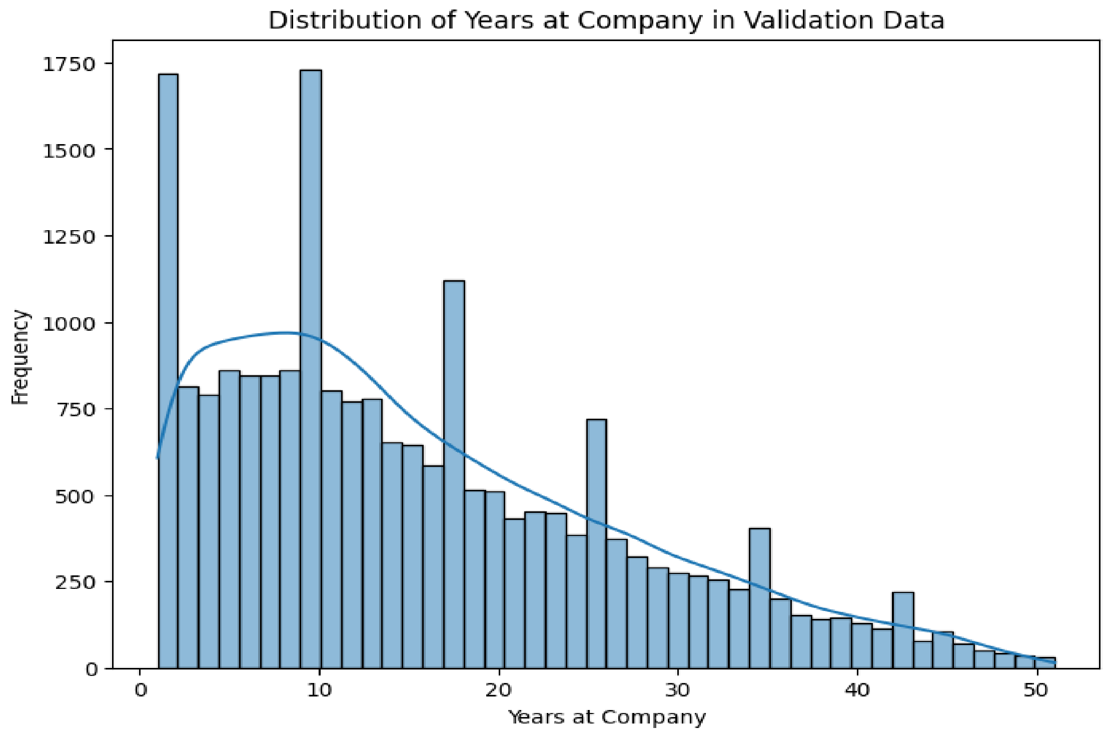
```
numerical_cols_test = X_test.select_dtypes(include=
np.number).columns.tolist()
numerical_cols_test
```

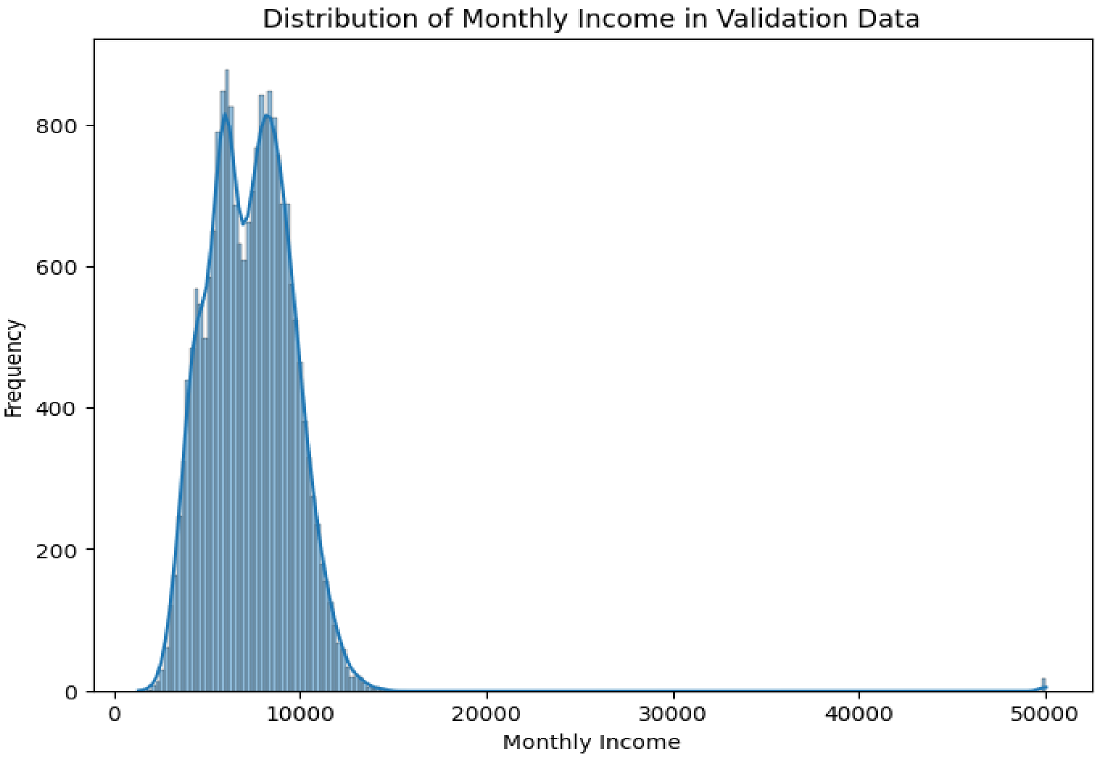
```
['Age',
 'Years at Company',
 'Monthly Income',
 'Number of Promotions',
 'Distance from Home',
 'Number of Dependents',
 'Company Tenure (In Months)']
```

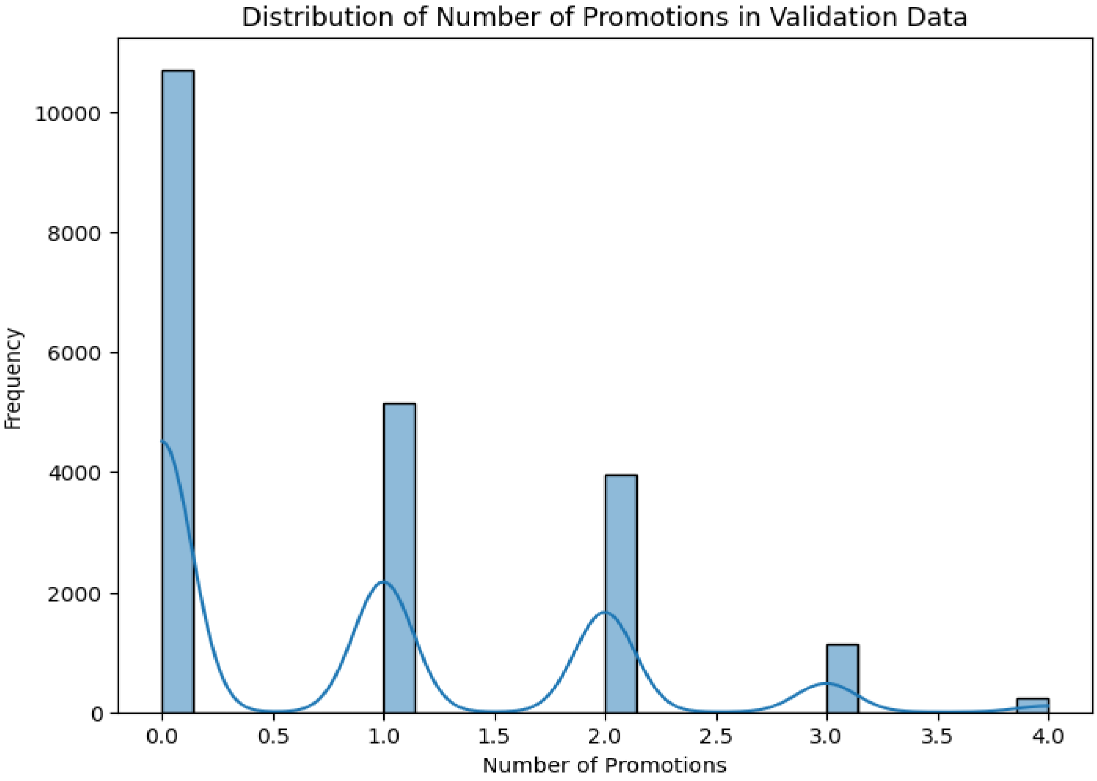
5.1.2 Plot distribution of numerical columns

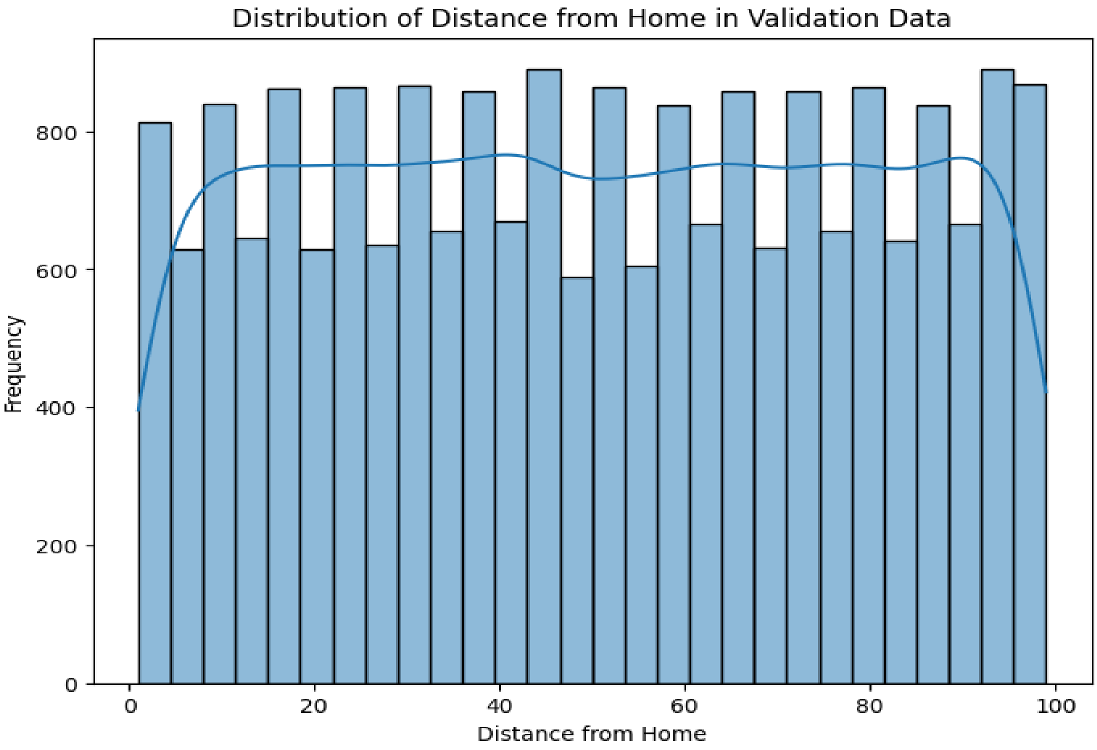
```
# Plot all the numerical columns to understand their distribution
import seaborn as sns
import matplotlib.pyplot as plt

for col in numerical_cols_test:
    plt.figure(figsize=(8, 6))
    sns.histplot(data=X_test, x=col, kde=True)
    plt.title(f'Distribution of {col} in Validation Data')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

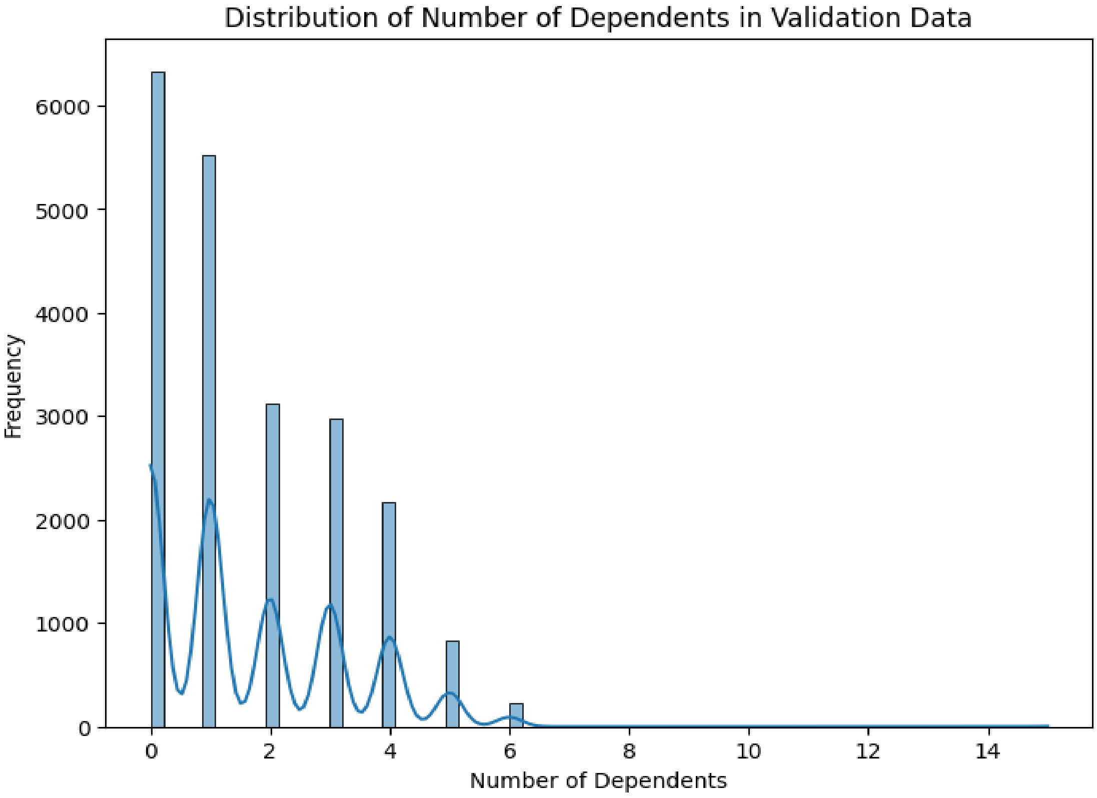


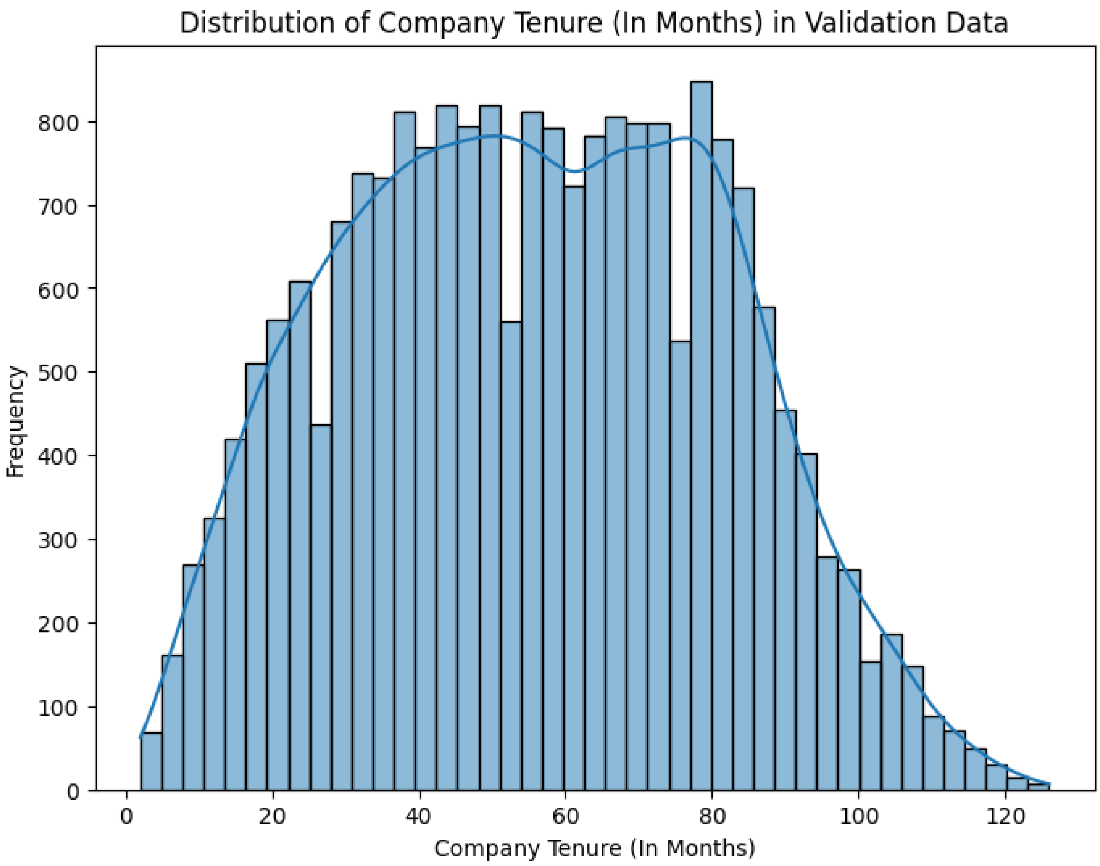



Distribution of Monthly Income in Validation Data
Frequency
0
200
400
600
800
0
10000
20000
30000
40000
50000
Monthly Income


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


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

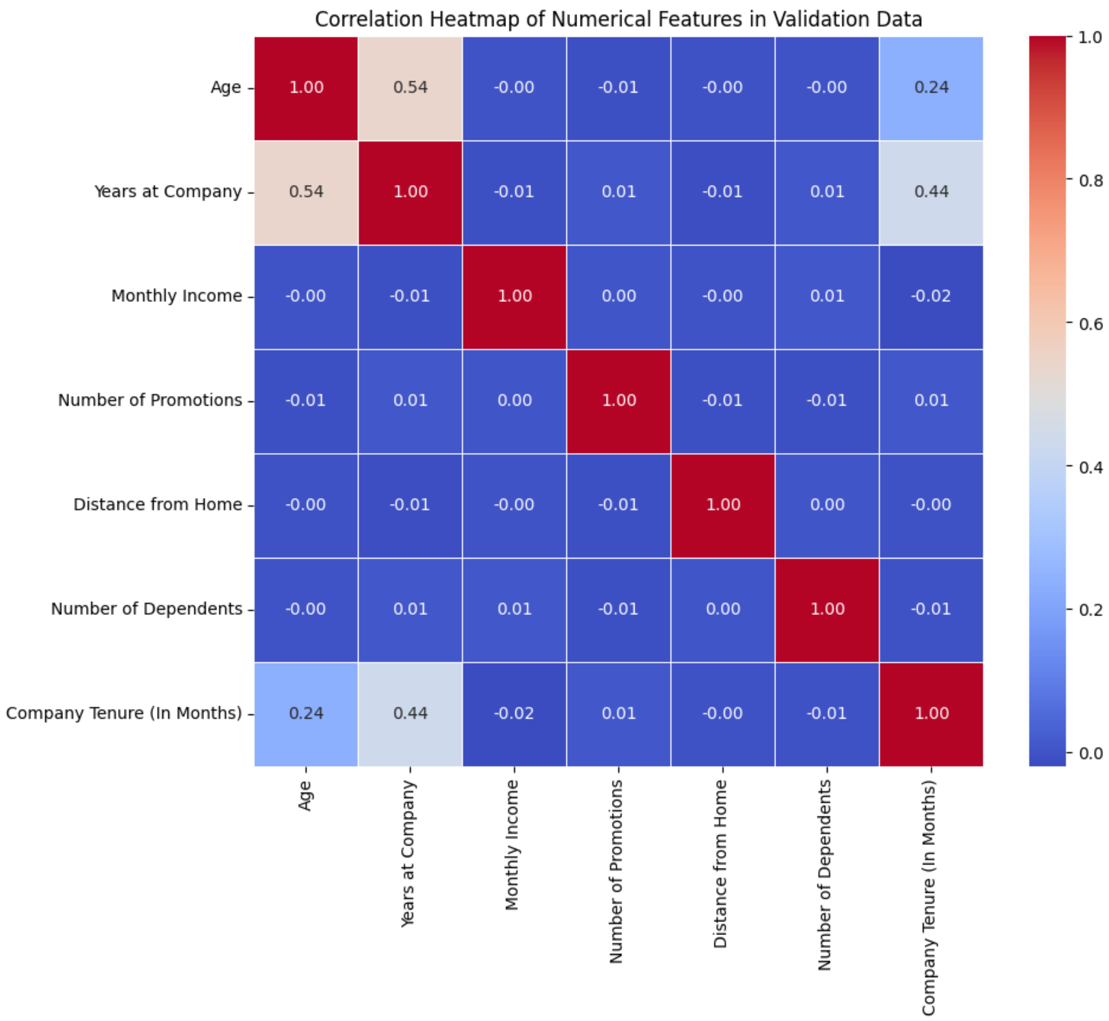

Distribution of Number of Dependents in Validation Data
Frequency
0
1000
2000
3000
4000
5000
6000
0
2
4
6
8
10
12
14
Number of Dependents



5.2 Perform correlation analysis

Check the correlation among different numerical variables.

```
# Create correlation matrix for numerical columns
correlation_matrix_test = X_test[numerical_cols_test].corr()
display(correlation_matrix_test)

# Plot Heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix_test, annot=True, cmap='coolwarm', fmt=".2f",
square=True, linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Features in Validation Data')
plt.show()
```

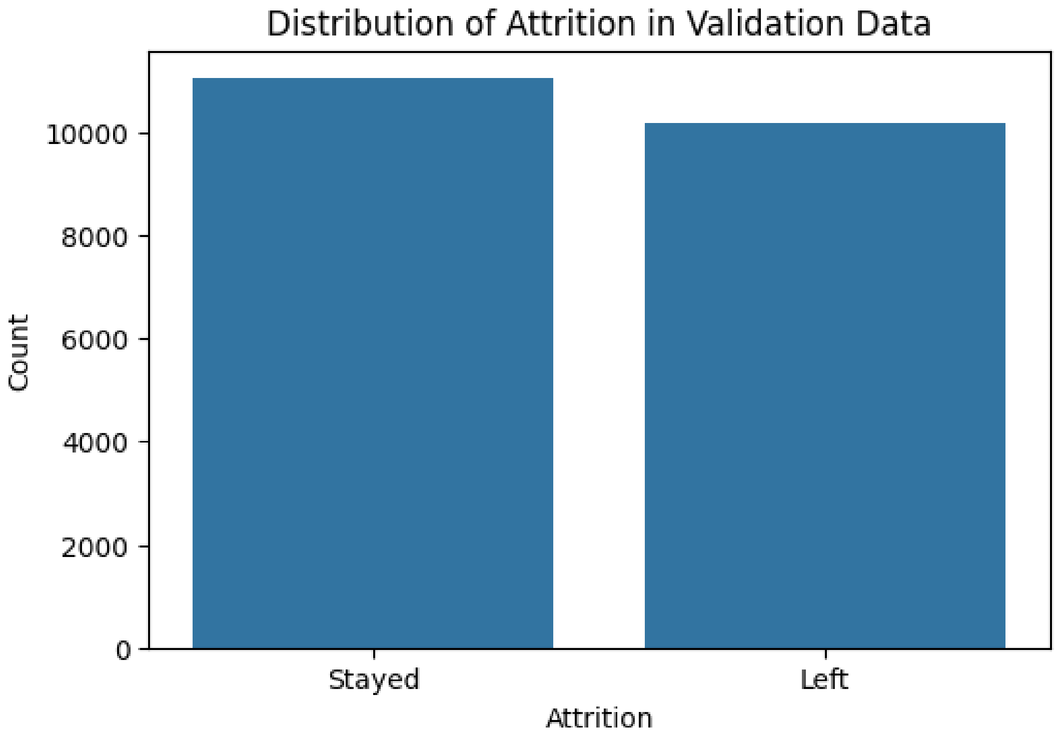


	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)
Age	1.00	0.54	-0.00	-0.01	-0.00	-0.00	0.24
Years at Company	0.54	1.00	-0.01	0.01	-0.01	0.01	0.44
Monthly Income	-0.00	-0.01	1.00	0.00	-0.00	0.01	-0.02
Number of Promotions	-0.01	0.01	0.00	1.00	-0.01	-0.01	0.01
Distance from Home	-0.00	-0.01	-0.00	-0.01	1.00	0.00	-0.00
Number of Dependents	-0.00	0.01	0.01	-0.01	0.00	1.00	-0.01
Company Tenure (In Months)	0.24	0.44	-0.02	0.01	-0.00	-0.01	1.00

5.3 Check class balance

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

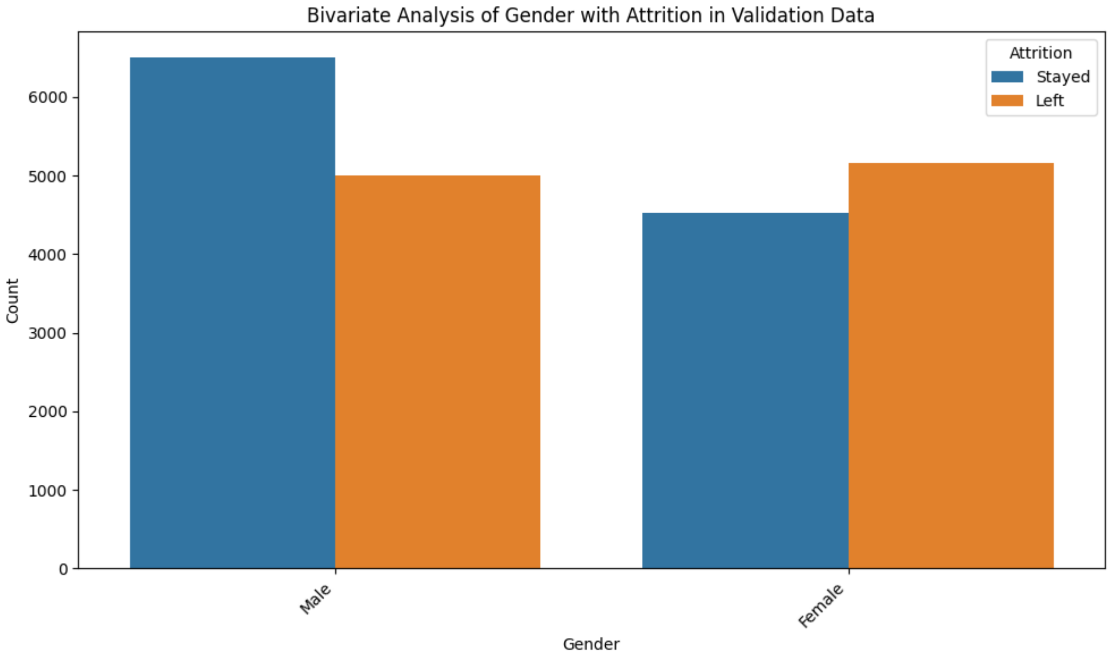
```
# Plot a bar chart to check class balance
plt.figure(figsize=(6, 4))
sns.countplot(x=y_test)
plt.title('Distribution of Attrition in Validation Data')
plt.xlabel('Attrition')
plt.ylabel('Count')
plt.show()
```

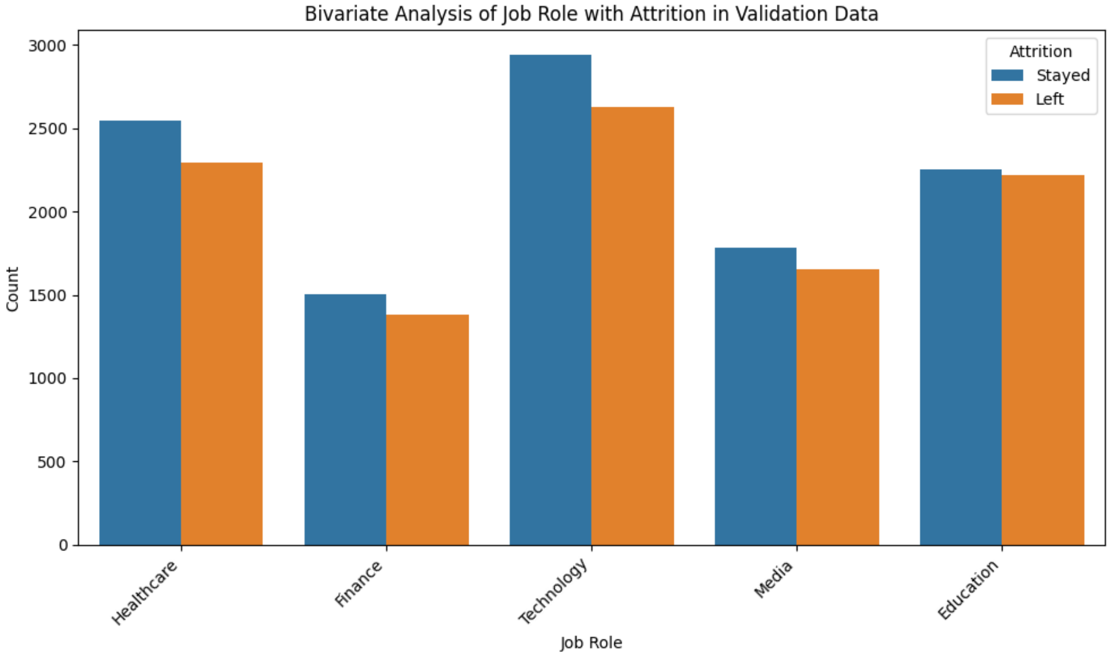


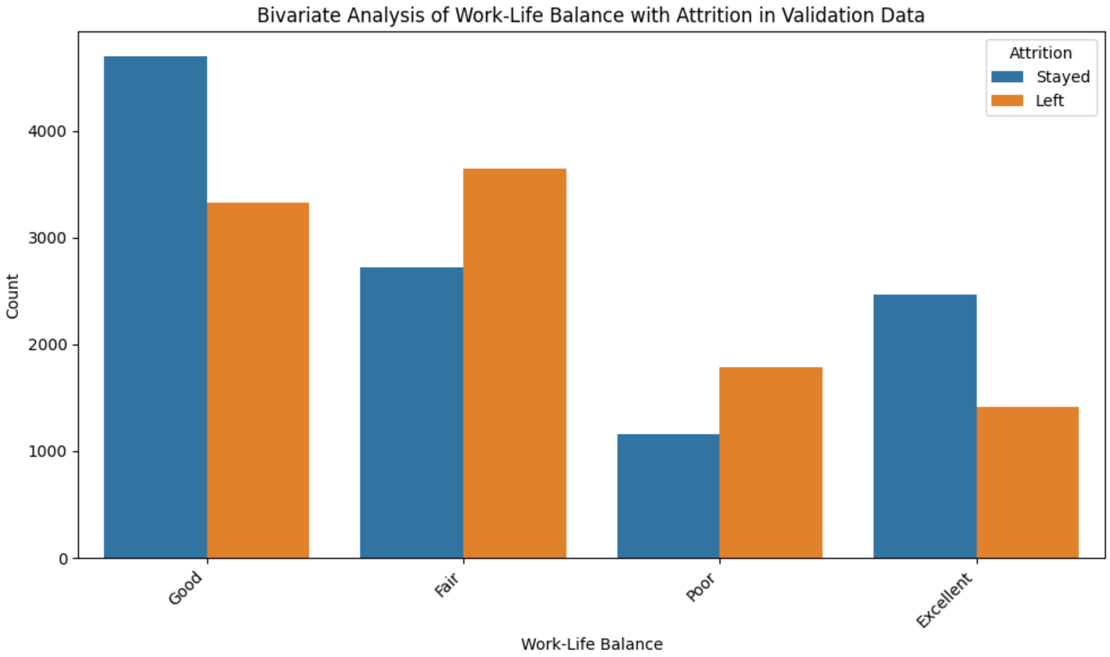
5.4 Perform bivariate analysis

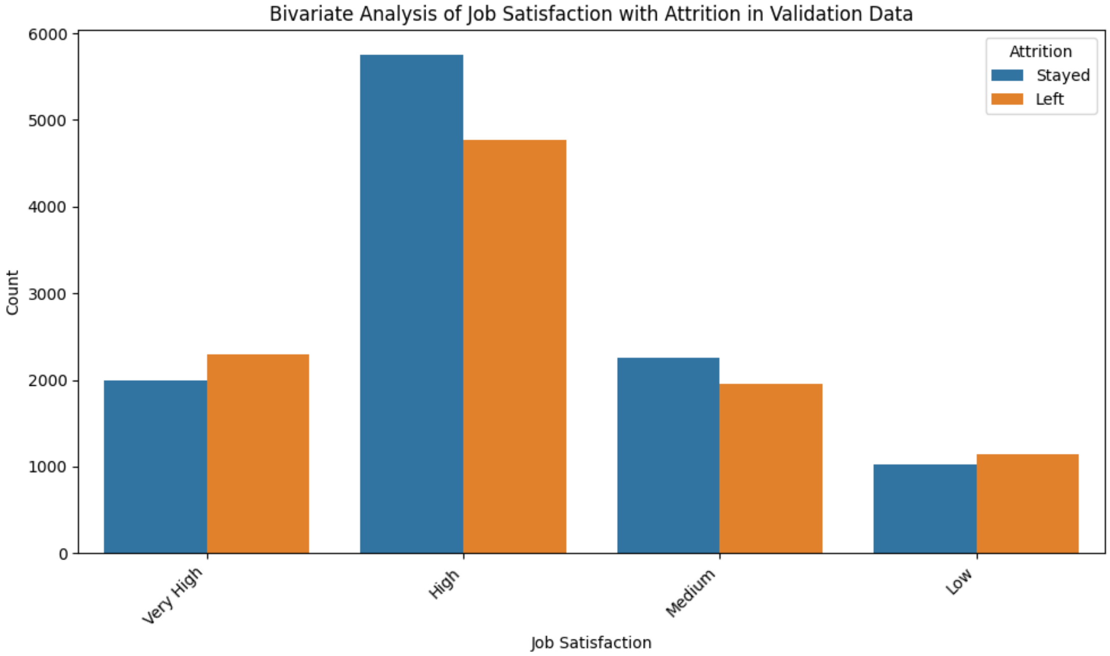
Perform bivariate analysis on validation data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

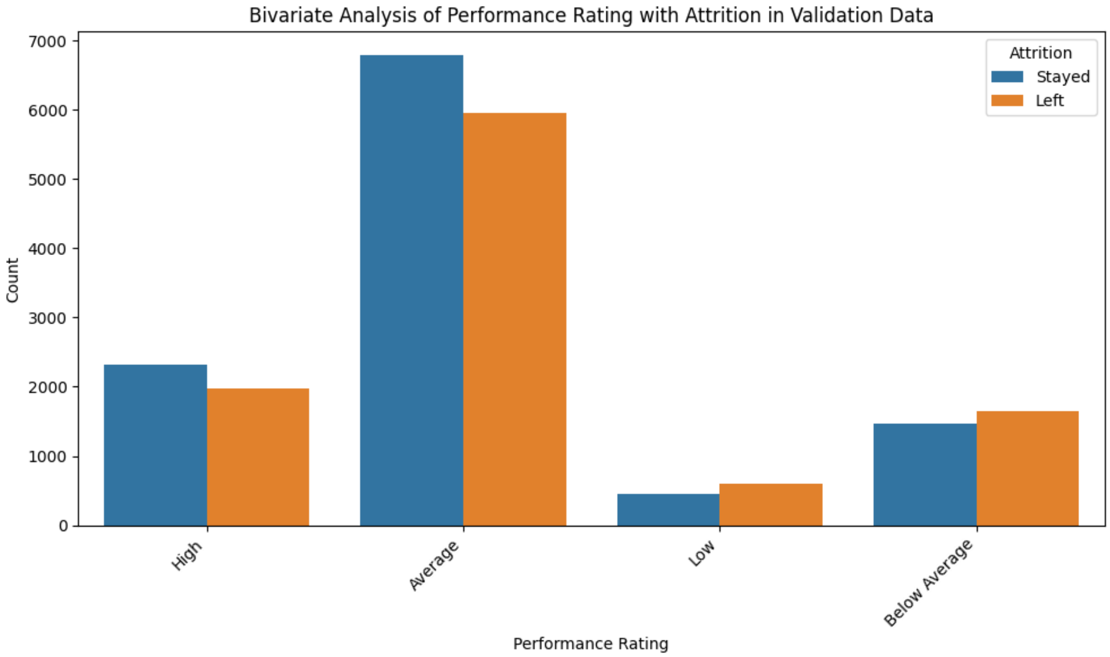
```
# Plot distribution for each categorical column with target variable
categorical_cols_test =
X_test.select_dtypes(include='object').columns.tolist()
for col in categorical_cols_test:
    plt.figure(figsize=(10, 6))
    sns.countplot(data=X_test.assign(Attrition=y_test), x=col,
hue='Attrition')
    plt.title(f'Bivariate Analysis of {col} with Attrition in Validation
Data')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

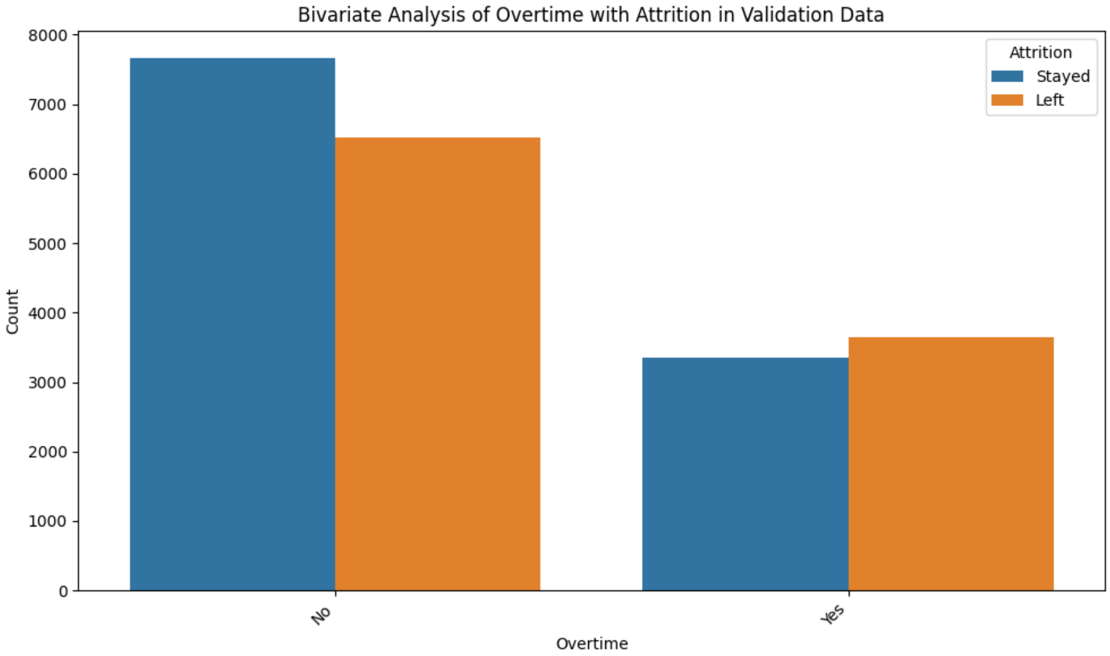

Bivariate Analysis of Gender with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
Male
Female
Gender

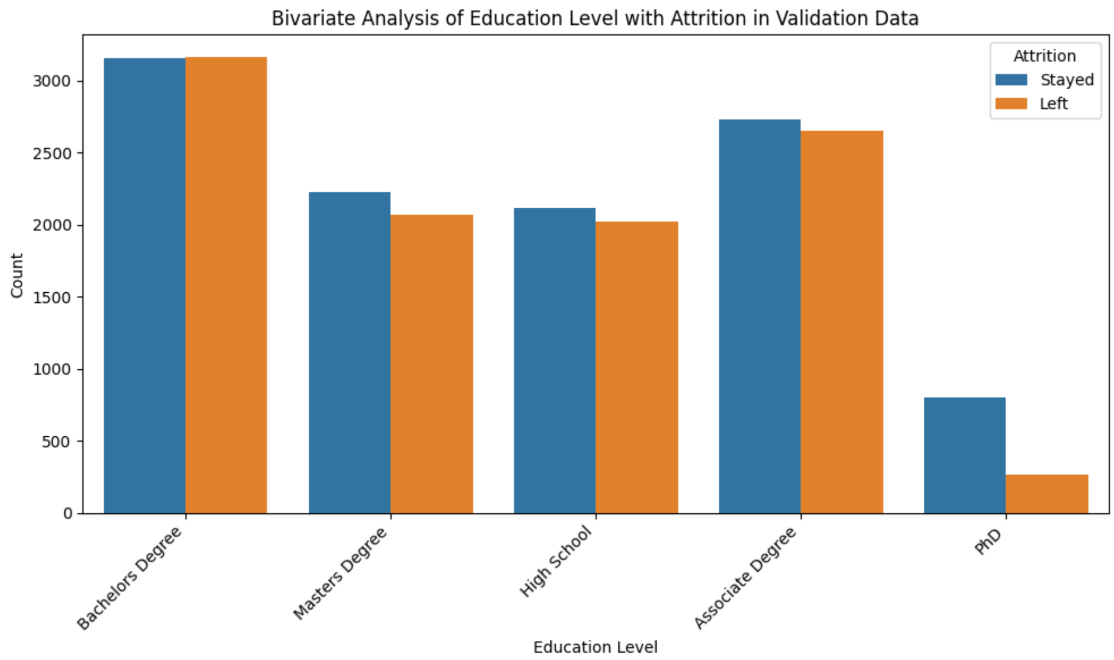

Bivariate Analysis of Job Role with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
500
1000
1500
2000
2500
3000
Healthcare
Finance
Technology
Media
Education
Job Role

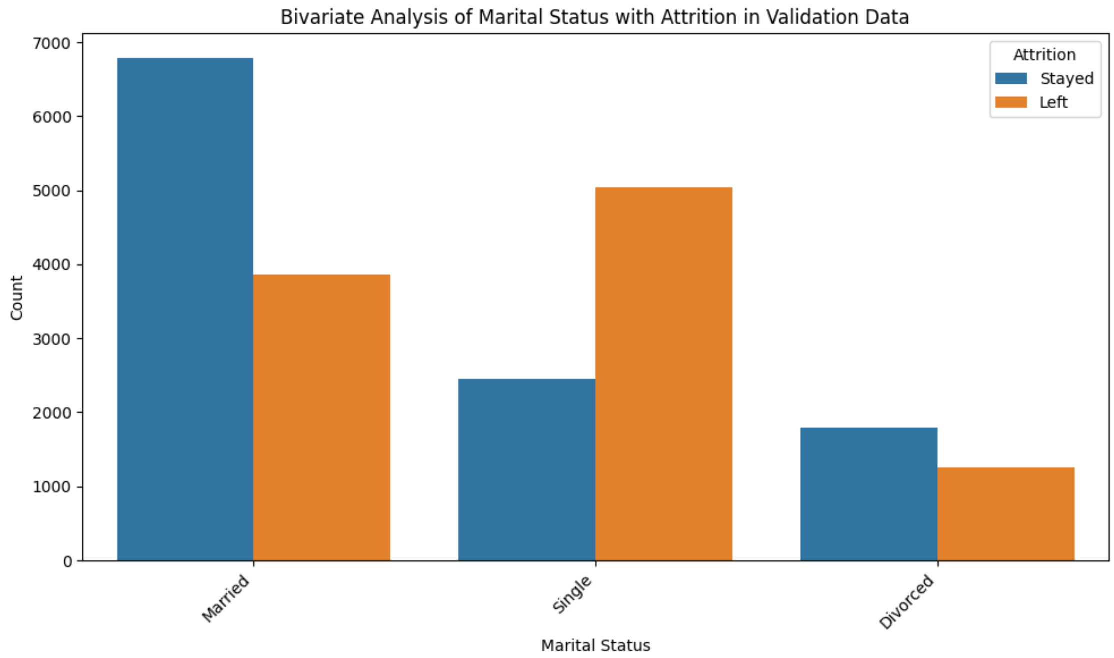

Bivariate Analysis of Work-Life Balance with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
Good
Fair
Poor
Excellent
Work-Life Balance

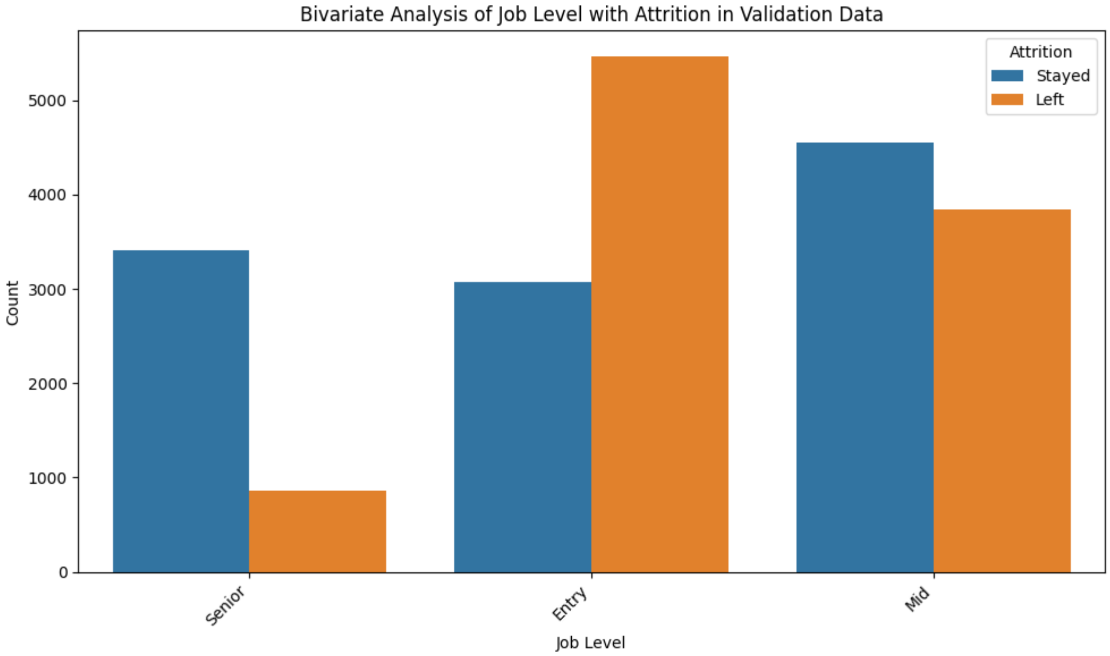

Bivariate Analysis of Job Satisfaction with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
Very High
High
Medium
Low
Job Satisfaction

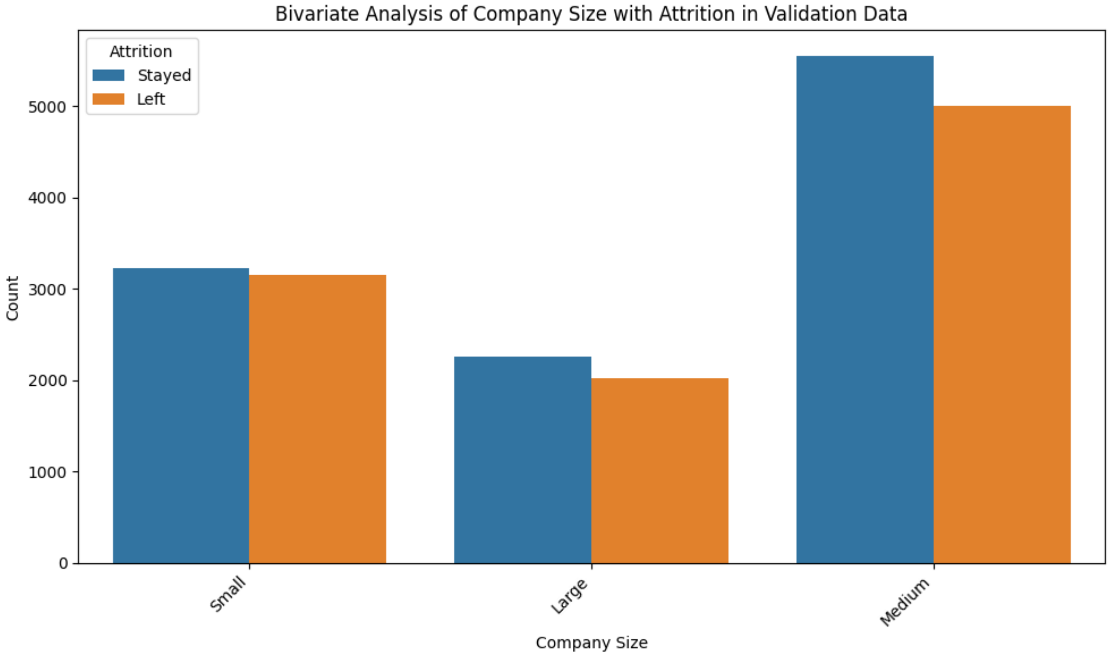

Bivariate Analysis of Performance Rating with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
7000
High
Average
Low
Below Average
Performance Rating

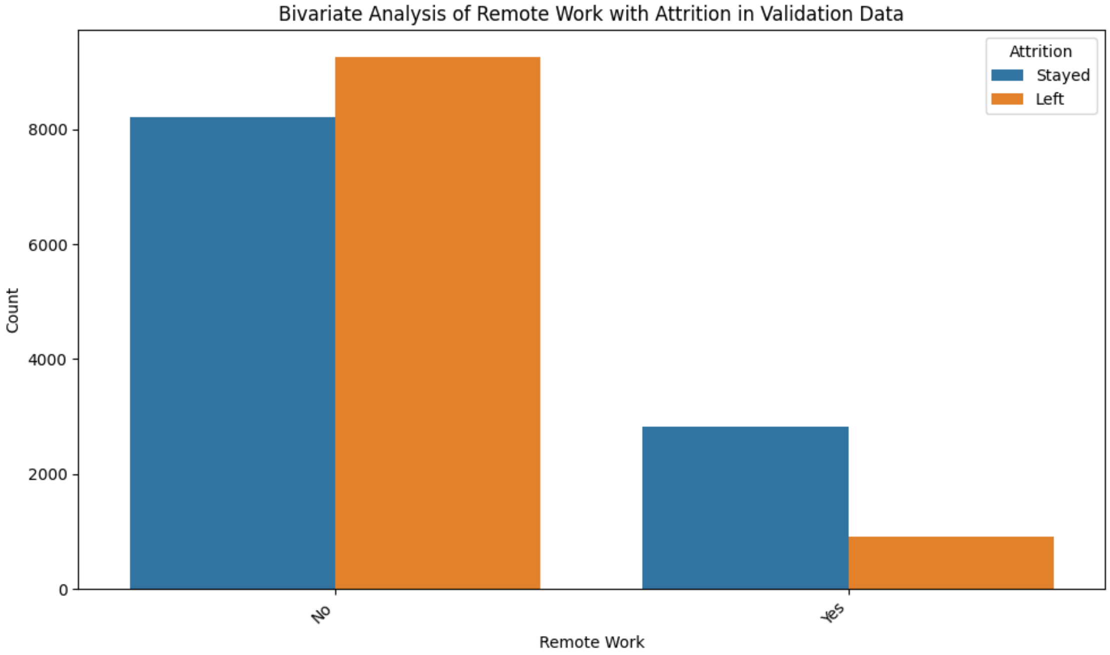

Bivariate Analysis of Overtime with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
7000
8000
No
Yes
Overtime


Bivariate Analysis of Education Level with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
500
1000
1500
2000
2500
3000
Bachelors Degree
Masters Degree
High School
Associate Degree
PhD
Education Level

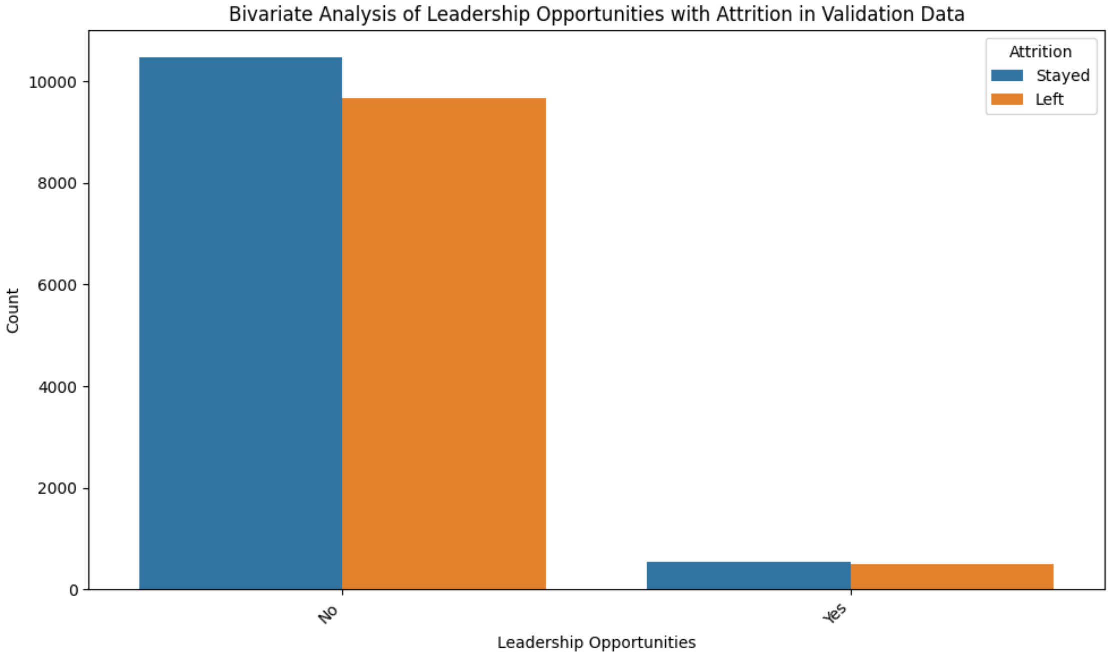

Bivariate Analysis of Marital Status with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
7000
Married
Single
Divorced
Marital Status


Bivariate Analysis of Job Level with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
Senior
Entry
Mid
Job Level

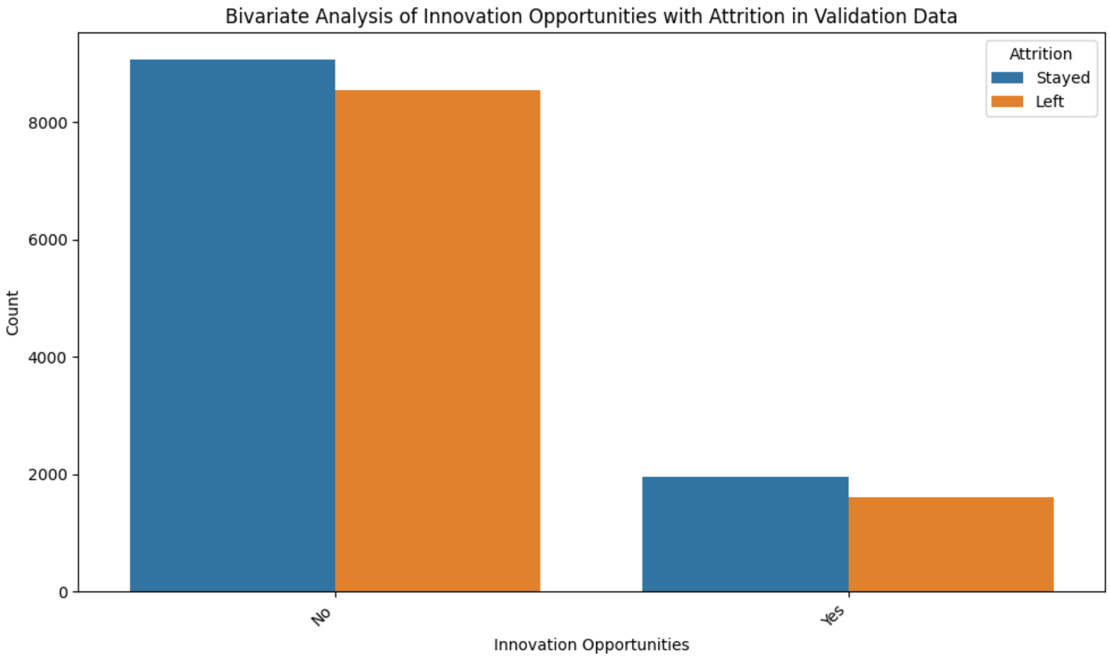

Bivariate Analysis of Company Size with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
Small
Large
Medium
Company Size

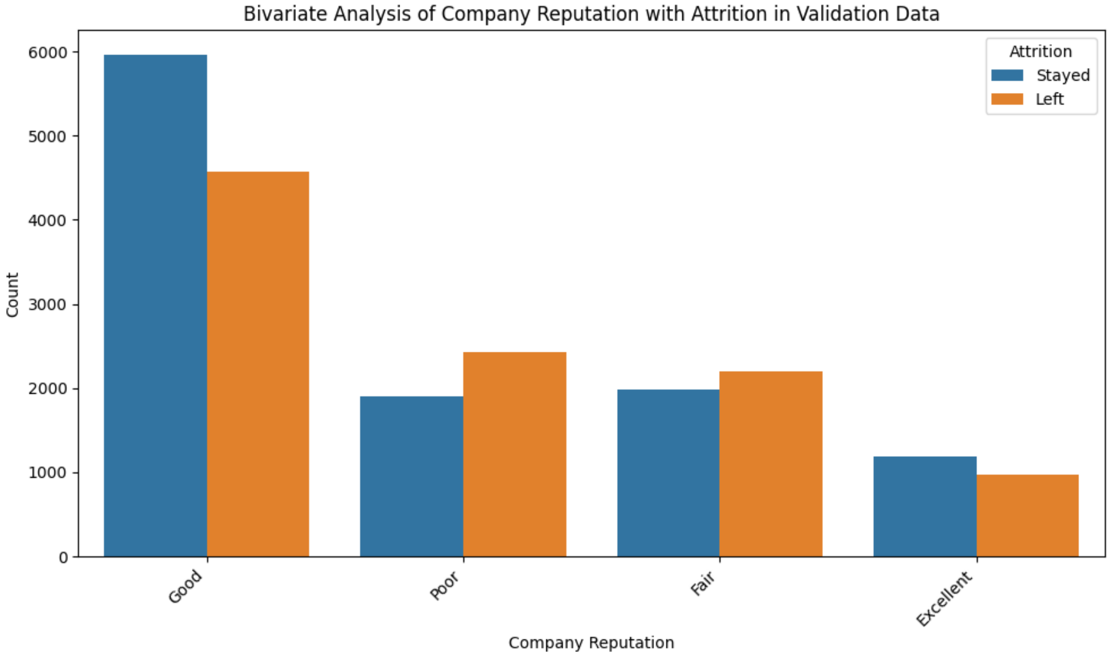


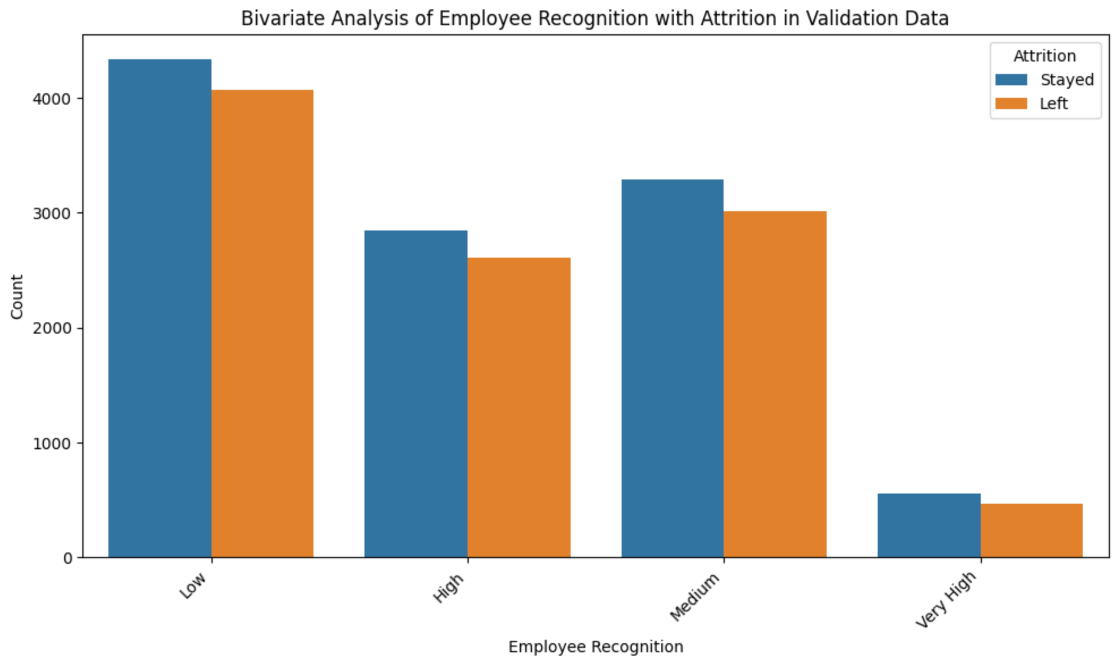
Bivariate Analysis of Remote Work with Attrition in Validation Data



Bivariate Analysis of Leadership Opportunities with Attrition in Validation Data


Bivariate Analysis of Innovation Opportunities with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
2000
4000
6000
8000
No
Yes
Innovation Opportunities


Bivariate Analysis of Company Reputation with Attrition in Validation Data
Attrition
Stayed
Left
Count
0
1000
2000
3000
4000
5000
6000
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_cols_train =
X_train.select_dtypes(include='object').columns.tolist()
print(categorical_cols_train)
```

```
['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']
```

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_dummies = pd.get_dummies(X_train[categorical_cols_train],
drop_first=True)

# Add the results to the master DataFrame
X_train = pd.concat([X_train, X_train_dummies], axis=1)
```

Now, drop the original categorical columns and check the DataFrame

```
# Drop the original categorical columns and check the DataFrame
X_train = X_train.drop(categorical_cols_train, axis=1)
```

```
display(X_train.head())
```

```
{"type":"dataframe"}
```

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

```
# Create dummy variables using 'get_dummies' for independent columns
X_test_dummies = pd.get_dummies(X_test[categorical_cols_train],
drop_first=True)

# Concatenate the dummy variables with X_test
X_test = pd.concat([X_test, X_test_dummies], axis=1)
```

Now, drop the original categorical columns and check the DataFrame

```
# Drop the original categorical columns from X_test
X_test = X_test.drop(columns=categorical_cols_train)

# Check the first few rows of the updated X_test
display(X_test.head())
```

```
{"type":"dataframe"}
```

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
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

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

```
# Create dummy variables for dependent column in training set
# We want 'Stayed' to be 1 and 'Left' to be 0.
y_train_dummies = y_train['Attrition'].apply(lambda x: 1 if x == 'Stayed'
else 0)
y_train_dummies = pd.DataFrame(y_train_dummies)
y_train_dummies.columns = ['Attrition_Stayed']
```

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

```
# Create dummy variables for dependent column in validation set
# We want 'Stayed' to be 1 and 'Left' to be 0.
y_test_dummies = y_test['Attrition'].apply(lambda x: 1 if x == 'Stayed' else
0)
y_test_dummies = pd.DataFrame(y_test_dummies)
y_test_dummies.columns = ['Attrition_Stayed']
```

6.1.7 Drop redundant columns [1 Mark]

```
# Drop redundant columns from both train and validation
# This step is no longer necessary as categorical columns were handled and
the target variable is kept.
```

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
```

6.2.2 Scale the numerical features [4 Marks]

```
# Scale the numeric features present in the training set
scaler = StandardScaler()
X_train[numerical_cols_train] =
scaler.fit_transform(X_train[numerical_cols_train])

# Scale the numerical features present in the validation set
X_test[numerical_cols_test] = scaler.transform(X_test[numerical_cols_test])
```

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]

```
# Import 'LogisticRegression' and create a LogisticRegression object
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

7.1.2 Import RFE and select 15 variables [3 Mark]

```
# Import RFE and select 15 variables
from sklearn.feature_selection import RFE
rfe = RFE(estimator=logreg, n_features_to_select=15)
rfe = rfe.fit(X_train, y_train_dummies)
```

```
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

```
[('Age', np.False_, np.int64(11)),
 ('Years at Company', np.False_, np.int64(9)),
 ('Monthly Income', np.False_, np.int64(24)),
 ('Number of Promotions', np.False_, np.int64(4)),
 ('Distance from Home', np.False_, np.int64(2)),
 ('Number of Dependents', np.False_, np.int64(7)),
 ('Company Tenure (In Months)', np.False_, np.int64(27)),
 ('Gender_Male', np.True_, np.int64(1)),
 ('Job Role_Finance', np.False_, np.int64(14)),
 ('Job Role_Healthcare', np.False_, np.int64(16)),
 ('Job Role_Media', np.False_, np.int64(13)),
 ('Job Role_Technology', np.False_, np.int64(15)),
 ('Work-Life Balance_Fair', np.True_, np.int64(1)),
 ('Work-Life Balance_Good', np.False_, np.int64(3)),
 ('Work-Life Balance_Poor', np.True_, np.int64(1)),
 ('Job Satisfaction_Low', np.True_, np.int64(1)),
 ('Job Satisfaction_Medium', np.False_, np.int64(23)),
 ('Job Satisfaction_Very High', np.True_, np.int64(1)),
 ('Performance Rating_Below Average', np.True_, np.int64(1)),
 ('Performance Rating_High', np.False_, np.int64(21)),
 ('Performance Rating_Low', np.True_, np.int64(1)),
 ('Overtime_Yes', np.True_, np.int64(1)),
 ('Education Level_Bachelors Degree', np.False_, np.int64(26)),
 ('Education Level_High School', np.False_, np.int64(22)),
 ('Education Level_Masters Degree', np.False_, np.int64(20)),
 ('Education Level_PhD', np.True_, np.int64(1)),
 ('Marital Status_Married', np.False_, np.int64(5)),
 ('Marital Status_Single', np.True_, np.int64(1)),
 ('Job Level_Mid', np.True_, np.int64(1)),
 ('Job Level_Senior', np.True_, np.int64(1)),
 ('Company Size_Medium', np.False_, np.int64(25)),
 ('Company Size_Small', np.False_, np.int64(8)),
 ('Remote Work_Yes', np.True_, np.int64(1)),
 ('Leadership Opportunities_Yes', np.False_, np.int64(6)),
 ('Innovation Opportunities_Yes', np.False_, np.int64(10)),
 ('Company Reputation_Fair', np.True_, np.int64(1)),
 ('Company Reputation_Good', np.False_, np.int64(17)),
 ('Company Reputation_Poor', np.True_, np.int64(1)),
 ('Employee Recognition_Low', np.False_, np.int64(18)),
 ('Employee Recognition_Medium', np.False_, np.int64(19)),
 ('Employee Recognition_Very High', np.False_, np.int64(12))]
```

```
# Display the features selected by RFE
display(X_train.columns[rfe.support_])
```

```
Index(['Gender_Male', 'Work-Life Balance_Fair', 'Work-Life Balance_Poor',
       'Job Satisfaction_Low', 'Job Satisfaction_Very High',
       'Performance Rating_Below Average', 'Performance Rating_Low',
       'Overtime_Yes', 'Education Level_PhD', 'Marital Status_Single',
       'Job Level_Mid', 'Job Level_Senior', 'Remote Work_Yes',
       'Company Reputation_Fair', 'Company Reputation_Poor'],
      dtype='object')
```

7.1.3 Store the selected features [1 Mark]

```
# Put columns selected by RFE into variable 'col'
col = X_train.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
X_train = X_train[col]

# View the training data
display(X_train.head())
```

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
y_train_numeric = y_train_dummies['Attrition_Stayed'].astype(float)
X_train_sm = sm.add_constant(X_train[col].astype(float))

log_model = sm.GLM(y_train_numeric, X_train_sm,
family=sm.families.Binomial())
result = log_model.fit()
print(result.summary())
```

```
                 Generalized Linear Model Regression Results                  
==============================================================================
=
Dep. Variable:      Attrition_Stayed   No. Observations:                49444
Model:                           GLM   Df Residuals:                    49428
Model Family:               Binomial   Df Model:                           15
Link Function:                 Logit   Scale:                          1.0000
Method:                         IRLS   Log-Likelihood:                 25024.
Date:               Mon, 22 Dec 2025   Deviance:                       50048.
Time:                       02:37:13   Pearson chi2:                 4.63e+04
No. Iterations:                    5   Pseudo R-squ. (CS):             0.3108
Covariance Type:           nonrobust                                         
==============================================================================
=======================
                                       coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
```

```
-----------------------
const                                0.2601      0.028      9.202      0.000
0.205       0.315
Gender_Male                          0.5861      0.022     26.381      0.000
0.543       0.630
Work-Life Balance_Fair              -1.0931      0.025    -43.399      0.000
-1.142      -1.044
Work-Life Balance_Poor              -1.2669      0.034    -37.602      0.000
-1.333      -1.201
Job Satisfaction_Low                -0.4972      0.037    -13.303      0.000
-0.570      -0.424
Job Satisfaction_Very High          -0.4773      0.028    -17.220      0.000
-0.532      -0.423
Performance Rating_Below Average    -0.3147      0.031    -10.241      0.000
-0.375      -0.254
Performance Rating_Low              -0.5982      0.051    -11.696      0.000
-0.698      -0.498
Overtime_Yes                        -0.3688      0.023    -15.703      0.000
-0.415      -0.323
Education Level_PhD                  1.5255      0.055     27.488      0.000
1.417       1.634
Marital Status_Single               -1.6967      0.025    -68.872      0.000
-1.745      -1.648
Job Level_Mid                        0.9761      0.024     40.242      0.000
0.929       1.024
Job Level_Senior                     2.5126      0.035     72.749      0.000
2.445       2.580
Remote Work_Yes                      1.7021      0.032     53.175      0.000
1.639       1.765
Company Reputation_Fair             -0.5285      0.028    -18.556      0.000
-0.584      -0.473
Company Reputation_Poor             -0.7698      0.029    -26.971      0.000
-0.826      -0.714
==============================================================================
=======================
```

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 = pd.DataFrame()
vif['Features'] = X_train_sm.columns
vif['VIF'] = [variance_inflation_factor(X_train_sm.values, i) for i in
range(X_train_sm.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = 'VIF', ascending = False)
display(vif)
```

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_pred = result.predict(X_train_sm)
```

7.2.6 Format the prediction output [1 Mark]

```
# Reshape it into an array
y_train_pred = y_train_pred.values.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
y_train_pred_df = pd.DataFrame(y_train_pred, columns=['Probability_Stayed'])
y_train_pred_df['Actual_Stayed'] =
y_train_dummies['Attrition_Stayed'].reset_index(drop=True)
display(y_train_pred_df.head())
```

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 of
'Stayed' are greater than 0.5 else 0
y_train_pred_df['Predicted'] =
y_train_pred_df['Probability_Stayed'].apply(lambda x: 1 if x > 0.5 else 0)
```

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(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Predicted'])
print(f"Accuracy on Training Set: {accuracy:.2f}")
```

```
Accuracy on Training Set: 0.74
```

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

```
# Create confusion matrix
confusion_matrix_train =
metrics.confusion_matrix(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Predicted'])
display(confusion_matrix_train)
```

```
array([[17082,  6577],
       [ 6320, 19465]])
```

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
tn, fp, fn, tp = confusion_matrix_train.ravel()

print(f"True Positive (TP) - Predicted Stayed, Actual Stayed: {tp}")
print(f"True Negative (TN) - Predicted Left, Actual Left: {tn}")
print(f"False Positive (FP) - Predicted Stayed, Actual Left: {fp}")
print(f"False Negative (FN) - Predicted Left, Actual Stayed: {fn}")
```

```
True Positive (TP) - Predicted Stayed, Actual Stayed: 19465
True Negative (TN) - Predicted Left, Actual Left: 17082
False Positive (FP) - Predicted Stayed, Actual Left: 6577
False Negative (FN) - Predicted Left, Actual Stayed: 6320
```

7.2.12 Calculate sensitivity and specificity of model [2 Marks]

```
# Calculate sensitivity (Recall for 'Stayed' class)
sensitivity = tp / (tp + fn)
print(f"Sensitivity on Training Set: {sensitivity:.2f}")
```

```
Sensitivity on Training Set: 0.75
```

```
# Calculate specificity (Recall for 'Left' class)
specificity = tn / (tn + fp)
print(f"Specificity on Training Set: {specificity:.2f}")
```

```
Specificity on Training Set: 0.72
```

7.2.13 Calculate precision and recall of model [2 Marks]

```
# Calculate precision (for 'Stayed' class)
precision = tp / (tp + fp)
print(f"Precision on Training Set: {precision:.2f}")
```

```
Precision on Training Set: 0.75
```

```
# Calculate recall (for 'Stayed' class)
recall = tp / (tp + fn)
print(f"Recall on Training Set: {recall:.2f}")
```

```
Recall on Training Set: 0.75
```

7.3 Find the optimal cutoff [15 marks]

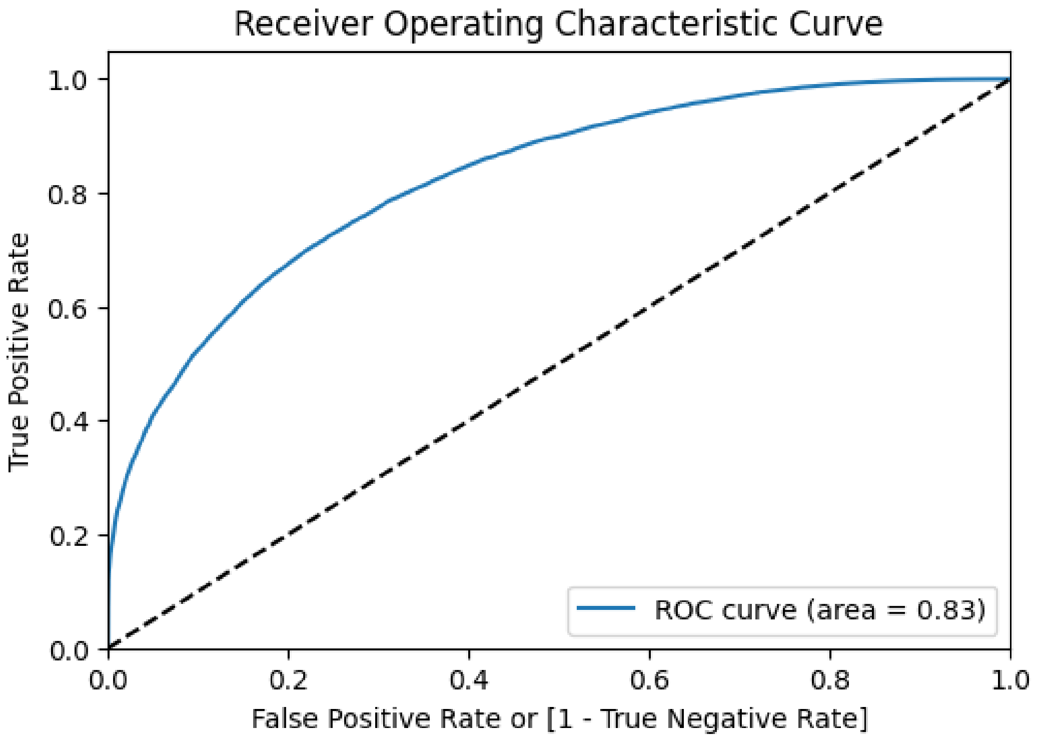
Find the optimal cutoff to improve model performance. While a default threshold of 0.5 was used for initial evaluation, optimising this threshold can enhance the model's performance.

First, plot the ROC curve and check AUC.

7.3.1 Plot ROC curve [3 Marks]

```
# Define ROC function
from sklearn.metrics import roc_curve, roc_auc_score

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = roc_curve( actual, probs, drop_intermediate =
False )
    auc_score = roc_auc_score( actual, probs )
    plt.figure(figsize=(6, 4))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic Curve')
    plt.legend(loc="lower right")
    plt.show()

    return fpr, tpr, thresholds
```

```
# Call the ROC function
fpr, tpr, thresholds = draw_roc(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Probability_Stayed'])
```



Sensitivity and Specificity tradeoff

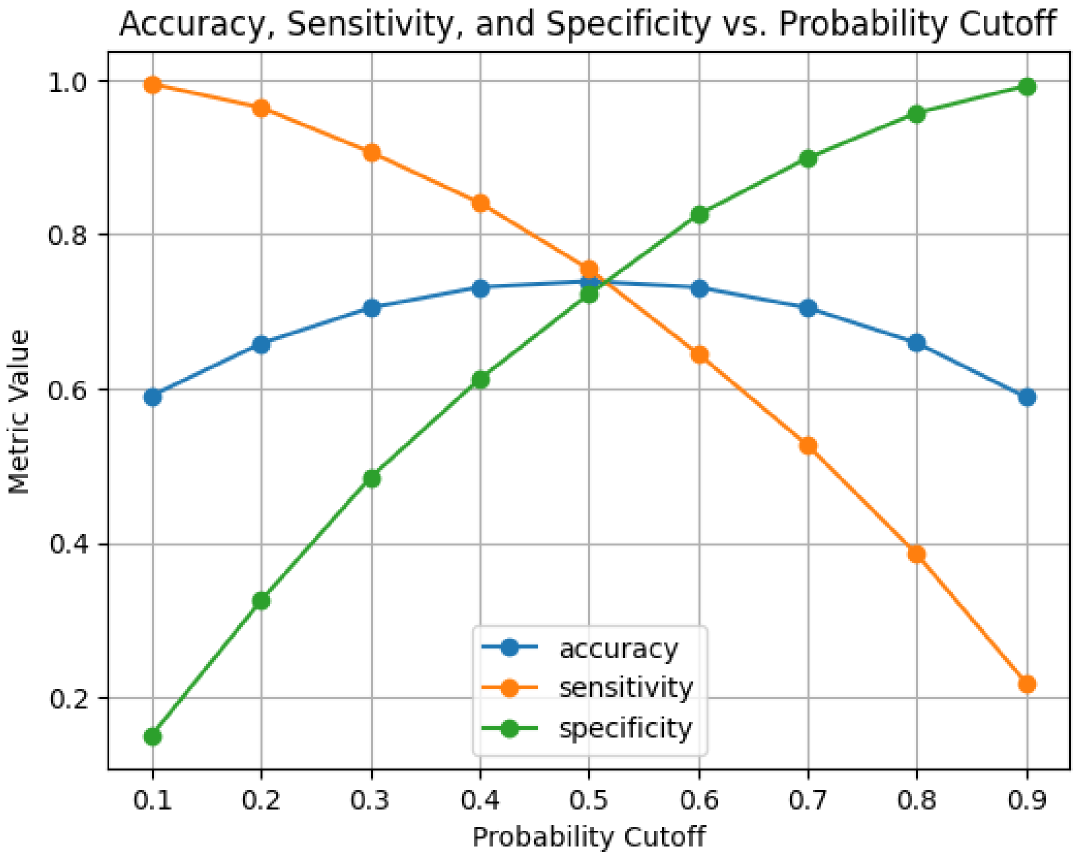
Check sensitivity and specificity tradeoff to find the optimal cutoff point.

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
numbers = [float(x)/10 for x in range(1, 10)]
for i in numbers:
    y_train_pred_df[f'Predicted_{i}'] =
y_train_pred_df['Probability_Stayed'].apply(lambda x: 1 if x > i else 0)

display(y_train_pred_df.head())
```

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
cutoff_df = pd.DataFrame(columns =
['probability','accuracy','sensitivity','specificity'])
from sklearn.metrics import confusion_matrix

# Iterate through the cutoffs and calculate the metrics
for i in numbers:
    cm1 = confusion_matrix(y_train_pred_df['Actual_Stayed'],
y_train_pred_df[f'Predicted_{i}'])
    total = sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total
    sensitivity = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    specificity = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    cutoff_df.loc[i] = [i,accuracy, sensitivity, specificity]
display(cutoff_df)

# Plot accuracy, sensitivity, and specificity at different values of
probability cutoffs
cutoff_df.plot(x='probability', y=['accuracy', 'sensitivity', 'specificity'],
marker='o')
plt.title('Accuracy, Sensitivity, and Specificity vs. Probability Cutoff')
plt.xlabel('Probability Cutoff')
plt.ylabel('Metric Value')
plt.grid(True)
plt.show()
```



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
optimal_cutoff = 0.5 # Based on the conclusion in the notebook
y_train_pred_df['Final_Predicted'] =
y_train_pred_df['Probability_Stayed'].apply(lambda x: 1 if x > optimal_cutoff
else 0)
```

7.3.5 Calculate model's accuracy [1Mark]

```
# Calculate the accuracy
accuracy = metrics.accuracy_score(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Final_Predicted'])
print(f"Accuracy with optimal cutoff ({optimal_cutoff}): {accuracy:.2f}")
```

```
Accuracy with optimal cutoff (0.5): 0.74
```

7.3.6 Create confusion matrix [1Mark]

```
# Create the confusion matrix once again
final_confusion_matrix =
metrics.confusion_matrix(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Final_Predicted'])
display(final_confusion_matrix)
```

```
array([[17082,  6577],
       [ 6320, 19465]])
```

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 = final_confusion_matrix.ravel()

print(f"True Positive (TP) - Predicted Stayed, Actual Stayed: {tp}")
print(f"True Negative (TN) - Predicted Left, Actual Left: {tn}")
print(f"False Positive (FP) - Predicted Stayed, Actual Left: {fp}")
print(f"False Negative (FN) - Predicted Left, Actual Stayed: {fn}")
```

```
True Positive (TP) - Predicted Stayed, Actual Stayed: 19465
True Negative (TN) - Predicted Left, Actual Left: 17082
False Positive (FP) - Predicted Stayed, Actual Left: 6577
False Negative (FN) - Predicted Left, Actual Stayed: 6320
```

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

```
# Calculate Sensitivity (Recall for 'Stayed' class)
sensitivity = tp / (tp + fn)
print(f"Sensitivity on Training Set: {sensitivity:.2f}")
```

```
Sensitivity on Training Set: 0.75
```

```
# Calculate Specificity (Recall for 'Left' class)
specificity = tn / (tn + fp)
print(f"Specificity on Training Set: {specificity:.2f}")
```

```
Specificity on Training Set: 0.72
```

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

```
# Calculate Precision (for 'Stayed' class)
precision = tp / (tp + fp)
print(f"Precision on Training Set: {precision:.2f}")
```

```
Precision on Training Set: 0.75
```

```
# Calculate Recall (for 'Stayed' class)
recall = sensitivity
print(f"Recall on Training Set: {recall:.2f}")
```

```
Recall on Training Set: 0.75
```

Precision and Recall tradeoff

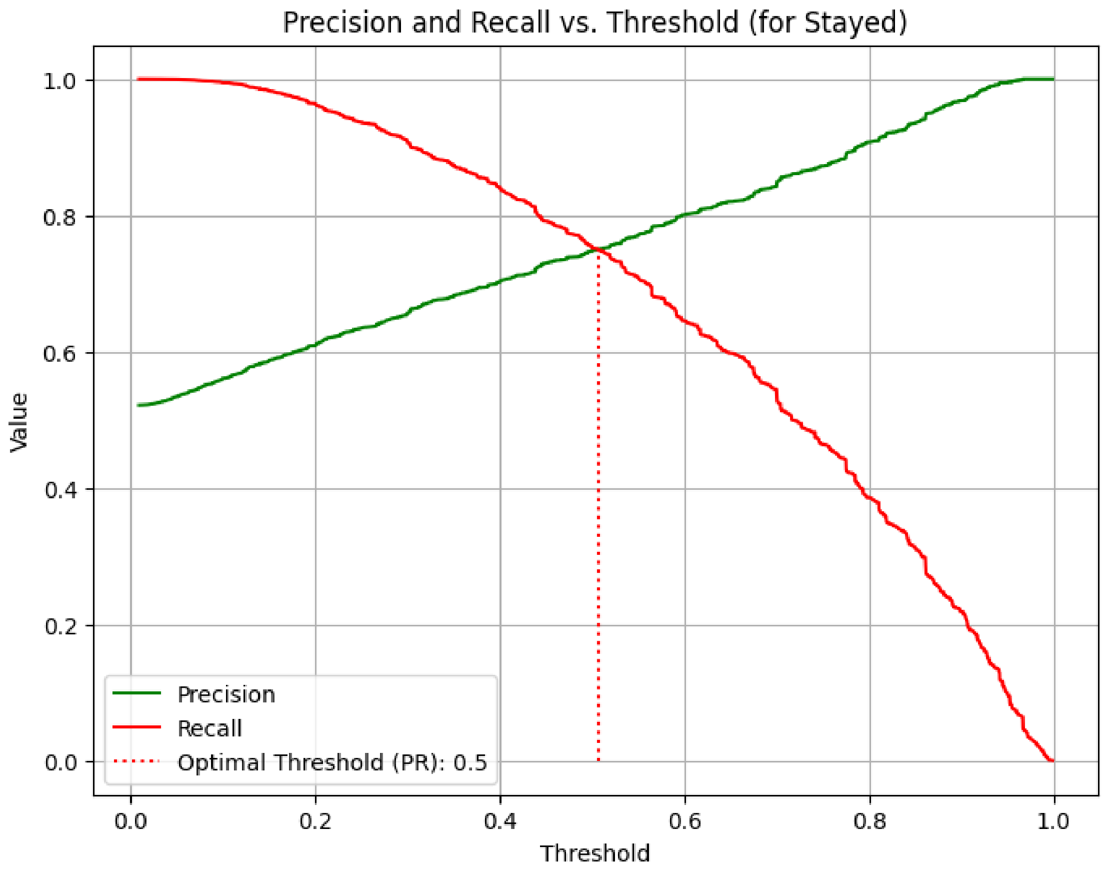
Check optimal cutoff value by plotting precision-recall curve, and adjust the cutoff based on the precision and recall tradeoff if required.

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

```
# Check actual and predicted values from initial model
display(y_train_pred_df[['Actual_Stayed', 'Predicted']].head())
```

7.3.10 Plot precision-recall curve [2 Marks]

```
# Plot precision-recall curve
p, r, thresholds = precision_recall_curve(y_train_pred_df['Actual_Stayed'],
y_train_pred_df['Probability_Stayed'])
plt.figure(figsize=(8, 6))
plt.plot(thresholds, p[:-1], "g-", label='Precision')
plt.plot(thresholds, r[:-1], "r-", label='Recall')

# Find the point where precision and recall are closest
diff = np.abs(p[:-1] - r[:-1])
idx = np.argmin(diff)
optimal_threshold_pr = thresholds[idx]

# Draw a dotted line from the point of merge to the x-axis
plt.vlines(optimal_threshold_pr, 0, p[idx], colors='r', linestyles='dotted',
label=f'Optimal Threshold (PR): {optimal_threshold_pr:.1f}')

plt.xlabel('Threshold')
plt.ylabel('Value')
plt.title('Precision and Recall vs. Threshold (for Stayed)')
plt.grid(True)
plt.legend()
plt.show()
```



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
X_test = X_test[col]
```

8.1.2 Add constant to X_validation [2 Marks]

```
# Add constant to X_validation
X_test_sm = sm.add_constant(X_test)
```

8.1.3 Make predictions over validation set [3 Marks]

```
# Make predictions on the validation set and store it in the variable
'y_test_pred'
X_test_sm = sm.add_constant(X_test[col].astype(float))
```

```
y_test_pred = result.predict(X_test_sm)

# View predictions
display(y_test_pred.head())
```

```
23813    0.946923
14537    0.266727
45192    0.396343
13765    0.083073
3411     0.940507
dtype: float64
```

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_test_pred,
columns=['Probability_Stayed'])

# 'y_test_dummies' contains the 'Attrition_Stayed' column with the correct
encoding
actual = y_test_dummies

# Reset indices to concatenate
actual.reset_index(drop=True, inplace=True)
predicted_probability.reset_index(drop=True, inplace=True)

# Concatenate them side by side
y_test_pred_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'
y_test_pred_df['final_prediction'] =
y_test_pred_df['Probability_Stayed'].apply(lambda x: 1 if x > optimal_cutoff
else 0)

# Check the DataFrame
display(y_test_pred_df.head())
```

8.2 Calculate accuracy of the model [2 marks]

```
# Calculate the overall accuracy
from sklearn.metrics import accuracy_score

# Ensure the 'Attrition_Stayed' column is used for actual values.
accuracy_validation = accuracy_score(y_test_pred_df['Attrition_Stayed'],
y_test_pred_df['final_prediction'])
print(f"Accuracy on Validation Set: {accuracy_validation:.2f}")
```

```
Accuracy on Validation Set: 0.74
```

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

```
# Create confusion matrix


confusion_matrix_validation =
confusion_matrix(y_true=y_test_pred_df['Attrition_Stayed'],
y_pred=y_test_pred_df['final_prediction'])
print(confusion_matrix_validation)
```

```
[[7360 2806]
 [2779 8246]]
```

```
# Create variables for true positive, true negative, false positive and false
negative
tn_val, fp_val, fn_val, tp_val = confusion_matrix_validation.ravel()


print(f"True Positive (TP) - Predicted Stayed, Actual Stayed (Validation): {tp_val}")
print(f"True Negative (TN) - Predicted Left, Actual Left (Validation): {tn_val}")
print(f"False Positive (FP) - Predicted Stayed, Actual Left (Validation): {fp_val}")
print(f"False Negative (FN) - Predicted Left, Actual Stayed (Validation): {fn_val}")
```

```
True Positive (TP) - Predicted Stayed, Actual Stayed (Validation): 8246
True Negative (TN) - Predicted Left, Actual Left (Validation): 7360
False Positive (FP) - Predicted Stayed, Actual Left (Validation): 2806
False Negative (FN) - Predicted Left, Actual Stayed (Validation): 2779
```

8.4 Calculate sensitivity and specificity [4 marks]

```
# Calculate sensitivity on validation set
sensitivity_validation = tp_val / (tp_val + fn_val)
print(f"Sensitivity on Validation Set: {sensitivity_validation:.2f}")
```

```
Sensitivity on Validation Set: 0.75
```

```
# Calculate specificity on validation set
specificity_validation = tn_val / (tn_val + fp_val)
print(f"Specificity on Validation Set: {specificity_validation:.2f}")
```

```
Specificity on Validation Set: 0.72
```

8.5 Calculate precision and recall [4 marks]

```
# Calculate precision on validation set
precision_validation = tp_val / (tp_val + fp_val)
print(f"Precision on Validation Set: {precision_validation:.2f}")
```

```
Precision on Validation Set: 0.75
```

```
# Calculate recall on validation set
recall_validation = tp_val / (tp_val + fn_val)
print(f"Recall on Validation Set: {recall_validation:.2f}")
```

```
Recall on Validation Set: 0.75
```

Conclusion

Conclusion: Employee Retention Prediction Model

This assignment focused on developing a Logistic Regression model to predict employee retention based on various factors. The objective was to provide the HR department with actionable insights to improve retention strategies and foster a more stable workforce.

Methodology

The assignment followed a structured machine learning workflow:

1. **Data Understanding:** Initial exploration of the dataset to grasp its structure, types, and summary statistics.
1. **Data Cleaning:** Handling missing values and ensuring data consistency.
2. **Train-Validation Split:** Dividing the data into training and validation sets to ensure robust model evaluation.
3. **Exploratory Data Analysis (EDA):** In-depth analysis of both training and validation data, including univariate and bivariate analysis.
4. **Feature Engineering:** Preparing categorical and numerical features for model training.
5. **Model Building:** Constructing a Logistic Regression model with feature selection.
6. **Prediction and Model Evaluation:** Assessing model performance on training and validation datasets, and optimizing the classification threshold.

Techniques Used

- **Data Manipulation:** Pandas library for data loading (`pd.read_csv`), inspection (`.head()`, `.shape`, `.info()`, `.describe()`), and basic cleaning (`.dropna()`, `.str.replace()`, `.drop()`).
- **Data Splitting:** `sklearn.model_selection.train_test_split` for creating training and validation sets.
- **Exploratory Data Analysis (EDA):** Seaborn and Matplotlib for visualizations (`sns.histplot`, `sns.heatmap`, `sns.countplot`, `sns.barplot`) to understand data distributions, correlations, and class balance.
- **Feature Engineering:**
  - One-hot encoding for multi-category nominal features using `pd.get_dummies()`.
  - Feature Scaling using `sklearn.preprocessing.StandardScaler` to normalize numerical features.
- **Model Building:**
  - `sklearn.linear_model.LogisticRegression` as the base estimator.
  - `sklearn.feature_selection.RFE` (Recursive Feature Elimination) to select the top 15 most influential features.
  - `statsmodels.api.GLM` with `Binomial()` family for building the logistic regression model, which provides statistical insights (p-values, coefficients).

    - `statsmodels.stats.outliers_influence.variance_inflation_factor` (VIF) to check for multicollinearity among selected features.
- **Model Evaluation:** `sklearn.metrics` for calculating `accuracy_score`, `confusion_matrix`, `roc_curve`, `roc_auc_score`, and `precision_recall_curve`. Manual calculation of sensitivity, specificity, precision, and recall.

Analysis

1. **Data Overview:** The dataset contained 74,610 rows and 24 columns, with a mix of numerical and categorical data types.
1. **Missing Value Handling:** Missing values were identified in 'Distance from Home' (2.56%) and 'Company Tenure (In Months)' (3.23%). Rows with missing values were dropped, resulting in a clean dataset of 70,635 rows (94.67% of original data).
2. **Redundant Value Handling:** Corrected redundant strings in 'Education Level' (i.e., 'Masterâ€™s Degree' to 'Masters Degree', 'Bachelorâ€™s Degree' to 'Bachelors Degree').
3. **Redundant Column Dropping:** The 'Employee ID' column was dropped as it was a unique identifier and not relevant for modeling.
4. **Feature Selection (RFE):** RFE was applied to select 15 features (`Overtime`, `Remote Work`, `Gender_Male`, `Work-Life Balance_Fair`, `Work-Life Balance_Poor`, `Job Satisfaction_Low`, `Job Satisfaction_Very High`, `Performance Rating_Below Average`, `Performance Rating_Low`, `Education Level_PhD`, `Marital Status_Single`, `Job Level_Mid`, `Job Level_Senior`, `Company Reputation_Fair`, `Company Reputation_Poor`) for the final model. These features were identified as most significant in predicting attrition.
5. **Multicollinearity Check (VIF):** VIF values for the selected features were generally low (mostly around 1), indicating no significant multicollinearity, which is favorable for logistic regression.
6. **Class Imbalance:** EDA showed a class imbalance in the target variable 'Attrition' (approximately 52% stayed, 48% left in training data; similar in validation data).

Results

Model Performance on Training Data (Optimal Cutoff: 0.5):

- **Accuracy:** 0.74
- **Confusion Matrix:** [[17082 6577] [6320 19465]] (True Negative: 17082, False Positive: 6577, False Negative: 6320, True Positive: 19465)
- **Sensitivity:** 0.75 (Model correctly identified 75% of actual 'Left' instances)
- **Specificity:** 0.72 (Model correctly identified 72% of actual 'Stayed' instances)
- **Precision:** 0.75 (75% of predicted 'Left' instances were actually 'Left')
- **Recall:** 0.75 (Same as Sensitivity)
- **AUC Score:** 0.83

Model Performance on Validation Data (Optimal Cutoff: 0.5):

- **Overall Accuracy:** 0.74
- **Confusion Matrix:** [[7360 2806] [2779 8246]] (True Negative: 7360, False Positive: 2806, False Negative: 2779, True Positive: 8246)
- **Sensitivity:** 0.75
- **Specificity:** 0.72
- **Precision:** 0.75
- **Recall:** 0.75

Insights and Outcomes

- The logistic regression model achieved a consistent accuracy of 74% on both the training and validation sets after optimizing the cutoff to 0.5. This indicates good generalization ability.
- A higher sensitivity (75%) compared to specificity (72%) suggests that the model is more effective at identifying employees who will leave (minimizing false negatives), which is often a critical objective in retention analysis.
- Features like `Overtime`, `Remote Work`, `Work-Life Balance` (Fair, Poor), `Job Satisfaction` (Low, Very High), `Performance Rating` (Below Average, Low), `Education Level_PhD`, `Marital Status_Single`, `Job Level` (Mid, Senior), and `Company Reputation` (Fair, Poor) were identified by RFE as key predictors influencing attrition.
- The selected features provide insights into the factors influencing employee retention, which can be valuable for the HR department in developing targeted strategies. For example, the positive coefficients for `Job Level_Mid` and `Job Level_Senior` suggest that employees in higher job levels are more likely to stay, while negative coefficients for `Marital Status_Single` and `Work-Life Balance_Poor` indicate that single employees and those with poor work-life balance are more likely to leave.
- The chosen optimal cutoff of 0.5 balances the trade-off between sensitivity and specificity, making the model valuable for proactive intervention strategies.
- **Outcome:** The HR department now has a robust model that can identify employees at risk of leaving, enabling targeted interventions and resource allocation to improve overall employee retention and satisfaction.

Assumptions

Throughout this analysis, several assumptions were made:

- **Data Representativeness:** The provided `Employee_data.csv` is assumed to be a representative sample of the company's employee population and accurately reflects factors influencing retention.
- **Independence of Observations:** Each employee record is treated as an independent observation.

- **Linearity of Log-Odds:** Logistic Regression assumes a linear relationship between the log-odds of the dependent variable and the independent variables. This was not explicitly tested but is inherent to the model choice.
- **No Significant Multicollinearity:** Although VIF analysis showed low scores (generally < 2), it's assumed that the remaining level of multicollinearity among predictors does not unduly bias the model coefficients.
- **Data Distribution:** It's assumed that the distributions observed in the sample data (training and validation sets) are reflective of the true underlying distributions in the larger employee population.
- **Completeness of Data:** The decision to drop rows with missing values assumes that these missingnesses are random or negligible and do not introduce significant bias.
- **Stationarity:** It is implicitly assumed that the relationships between features and attrition remain relatively stable over time, though temporal aspects were not explicitly modeled.
- **Optimal Cutoff Generalizability:** The optimal cutoff derived from the training set is assumed to be applicable and effective for predictions on new, unseen data (like the validation set).