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:

Column Name	Description
Employee ID	A unique identifier assigned to each employee.
Age	The age of the employee, ranging from 18 to 60 years.
Gender	The gender of the employee.
Years at Company	The number of years the employee has been working at the company.
Monthly Income	The monthly salary of the employee, in dollars.
Job Role	The department or role the employee works in, encoded into categories such as Finance, Healthcare, Technology, Education, and Media.
Work-Life Balance	The employee's perceived balance between work and personal life (Poor, Below Average, Good, Excellent).
Job Satisfaction	The employee's satisfaction with their job (Very Low, Low, Medium, High).
Performance Rating	The employee's performance rating (Low, Below Average, Average, High).
Number of Promotions	The total number of promotions the employee has received.
Overtime	Number of overtime hours.
Distance from Home	The distance between the employee's home and workplace, in miles.
Education Level	The highest education level attained by the employee (High School, Associate Degree, Bachelor's Degree, Master's Degree, PhD).
Marital Status	The marital status of the employee (Divorced, Married, Single).
Number of Dependents	Number of dependents the employee has.
Job Level	The job level of the employee (Entry, Mid, Senior).
Company Size	The size of the company the employee works for (Small, Medium, Large).
Company Tenure (In Months)	The total number of years the employee has been working in the industry.
Remote Work	Whether the employee works remotely (Yes or No).
Leadership Opportunities	Whether the employee has leadership opportunities (Yes or No).

Column Name	Description
Innovation Opportunities	Whether the employee has opportunities for innovation (Yes or No).
Company Reputation	The employee's perception of the company's reputation (Very Poor, Poor, Good, Excellent).
Employee Recognition	The level of recognition the employee receives(Very Low, Low, Medium, High).
Attrition	Whether the employee has left the company.

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

1.1 Load the Data

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

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	...	Number of Dependents	Jo Leve
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	...	0	Mi
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	...	3	Mi
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	...	3	Mi
3	65791	36	Female	7	Education	3989	Good	High	High	1	...	2	Mi
4	65026	56	Male	41	Education	4821	Fair	Very High	Average	0	...	0	Senio
...	...	...	...	...	...	...	...	...	...	...	...	...	.
74605	13450	44	Female	20	Healthcare	7578	Fair	Medium	Low	2	...	3	Mi
74606	2386	45	Male	26	Technology	8342	Excellent	Very High	Below Average	3	...	4	Mi
74607	36968	28	Female	3	Technology	9763	Poor	Low	Average	0	...	3	Senio
74608	24276	37	Male	3	Education	3644	Fair	High	Average	2	...	4	Entr
74609	9839	38	Male	28	Media	6172	Good	High	High	0	...	2	Mi

74610 rows × 24 columns

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

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	...	Number of Dependents	Job Level	Co
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	...	0	Mid	M
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	...	3	Mid	M
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	...	3	Mid	M
3	65791	36	Female	7	Education	3989	Good	High	High	1	...	2	Mid	
4	65026	56	Male	41	Education	4821	Fair	Very High	Average	0	...	0	Senior	M

5 rows × 24 columns

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

```
(74610, 24)
```

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

Show hidden output

1.2 Check the basic statistics

```
# Check the summary of the dataset
df.describe(include='all')
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number o Promotion
count	74610.000000	74610.000000	74610	74610.000000	74610	74610.000000	74610	74610	74610	74610.00000
unique	NaN	NaN	2	NaN	5	NaN	4	4	4	Na
top	NaN	NaN	Male	NaN	Technology	NaN	Good	High	Average	Na
freq	NaN	NaN	40898	NaN	19350	NaN	28196	37306	44786	Na
mean	37246.028696	38.529379	NaN	15.722638	NaN	7344.931417	NaN	NaN	NaN	0.83295
std	21505.785344	12.082299	NaN	11.224059	NaN	2596.373589	NaN	NaN	NaN	0.99532
min	1.000000	18.000000	NaN	1.000000	NaN	1226.000000	NaN	NaN	NaN	0.00000
25%	18624.250000	28.000000	NaN	7.000000	NaN	5652.000000	NaN	NaN	NaN	0.00000
50%	37239.500000	39.000000	NaN	13.000000	NaN	7348.500000	NaN	NaN	NaN	1.00000
75%	55871.750000	49.000000	NaN	23.000000	NaN	8876.000000	NaN	NaN	NaN	2.00000
max	74498.000000	59.000000	NaN	51.000000	NaN	50030.000000	NaN	NaN	NaN	4.00000

11 rows × 24 columns

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

	0
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().mean() * 100
```

	0
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
df = df.dropna(axis=0).reset_index(drop=True)
df.shape
```

```
(70635, 24)
```

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
remaining_pct = len(df) / 74610 * 100   # 74610 mentioned in data dictionary
remaining_pct
```

```
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

```

```

```
# Write a function to display the categorical columns with their unique values and check for redundant values
cat_cols = df.select_dtypes(include=['object']).columns

def show_unique_values(data, cols):
    for c in cols:
        print(f'Column: {c}')
        print(data[c].unique())
        print('-' * 40)

show_unique_values(df, cat_cols)
```

```
Column: Gender
['Male' 'Female']
----------------------------------------
Column: Job Role
['Education' 'Media' 'Healthcare' 'Technology' 'Finance']
----------------------------------------
Column: Work-Life Balance
['Excellent' 'Poor' 'Good' 'Fair']
----------------------------------------
Column: Job Satisfaction
['Medium' 'High' 'Very High' 'Low']
----------------------------------------
Column: Performance Rating
['Average' 'Low' 'High' 'Below Average']
----------------------------------------
Column: Overtime
['No' 'Yes']
----------------------------------------
Column: Education Level
['Associate Degree' 'Masterâ€™s Degree' 'Bachelorâ€™s Degree'
 'High School' 'PhD']
----------------------------------------
Column: Marital Status
['Married' 'Divorced' 'Single']
----------------------------------------
Column: Job Level
['Mid' 'Senior' 'Entry']
----------------------------------------
Column: Company Size
['Medium' 'Small' 'Large']
----------------------------------------
Column: Remote Work
['No' 'Yes']
----------------------------------------
Column: Leadership Opportunities
['No' 'Yes']
----------------------------------------
Column: Innovation Opportunities
['No' 'Yes']
----------------------------------------
Column: Company Reputation
['Excellent' 'Fair' 'Poor' 'Good']
----------------------------------------
Column: Employee Recognition
['Medium' 'Low' 'High' 'Very High']
----------------------------------------
Column: Attrition
['Stayed' 'Left']
----------------------------------------
```

```
# Check the data
df[cat_cols].head()
```

	Gender	Job Role	Work-Life Balance	Job Satisfaction	Performance Rating	Overtime	Education Level	Marital Status	Job Level	Company Size	Remote Work	Leadership Opportunities
0	Male	Education	Excellent	Medium	Average	No	Associate Degree	Married	Mid	Medium	No	No
1	Female	Media	Poor	High	Low	No	Masterâ€™s Degree	Divorced	Mid	Medium	No	No
2	Female	Healthcare	Good	High	Low	No	Bachelorâ€™s Degree	Married	Mid	Medium	No	No
3	Female	Education	Good	High	High	No	High School	Single	Mid	Small	Yes	No

2.3 Drop redundant columns [2 marks]

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

	Gender	Job Role	Work-Life Balance	Job Satisfaction	Performance Rating	Overtime	Education Level	Marital Status	Job Level	Company Size	Remote Work	Leadership Opportunities
0	Male	Education	Excellent	Medium	Average	No	Associate Degree	Married	Mid	Medium	No	No
1	Female	Media	Poor	High	Low	No	Masterâ€™s Degree	Divorced	Mid	Medium	No	No
2	Female	Healthcare	Good	High	Low	No	Bachelorâ€™s Degree	Married	Mid	Medium	No	No
3	Female	Education	Good	High	High	No	High School	Single	Mid	Small	Yes	No

```
# Drop redundant columns which are not required for modelling
# 'Employee ID' is just an identifier; 'Company Tenure In Months' duplicates 'Years at Company' (similar info)
df = df.drop(columns=['Employee ID', 'Company Tenure In Months'], errors='ignore')
df.head()
```

	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	...	Number of Dependents	Job Level	Co
0	31	Male	19	Education	5390	Excellent	Medium	Average	2	No	...	0	Mid	M
1	59	Female	4	Media	5534	Poor	High	Low	3	No	...	3	Mid	M
2	24	Female	10	Healthcare	8159	Good	High	Low	0	No	...	3	Mid	M
3	36	Female	7	Education	3989	Good	High	High	1	No	...	2	Mid	
4	56	Male	41	Education	4821	Fair	Very High	Average	0	Yes	...	0	Senior	M

5 rows × 23 columns

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
# Put the target variable in y
# Target: Attrition (Stayed/Left) -> convert to binary later
X = df.drop(columns=['Attrition'])
y = df['Attrition']
```

3.3 Split the data [3 Marks]

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

X_train.shape, X_validation.shape
```

```
((49444, 22), (21191, 22))
```

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.

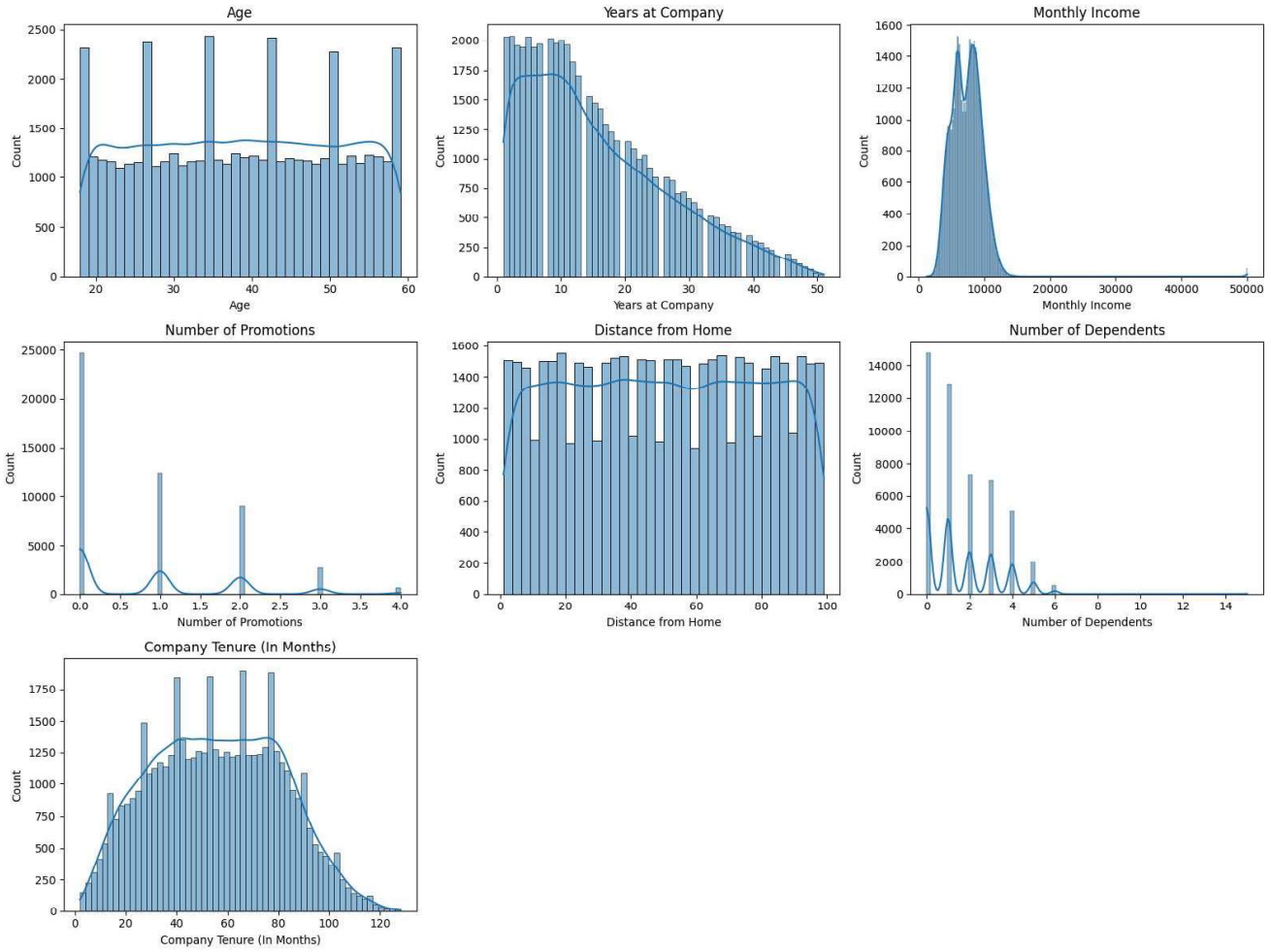
4.1.1 Select numerical columns from training data [1 Mark]

```
# Select numerical columns
num_cols = X_train.select_dtypes(include=[np.number]).columns
num_cols
```

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

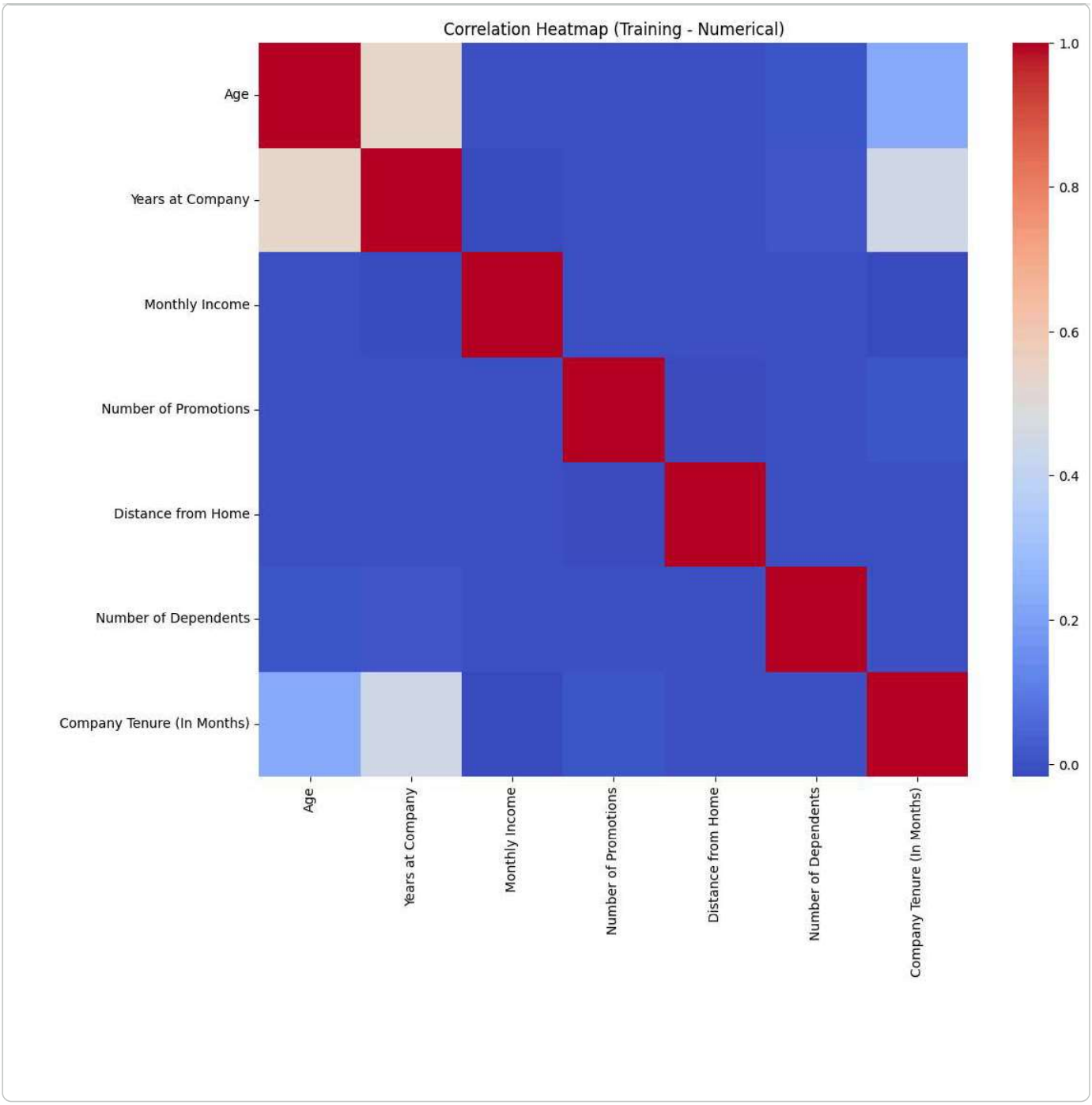
4.1.2 Plot distribution of numerical columns [5 Marks]

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

# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 12))
for i, col in enumerate(num_cols, 1):
    plt.subplot(len(num_cols)//3 + 1, 3, i)
    sns.histplot(X_train[col], kde=True)
    plt.title(col)
plt.tight_layout()
plt.show()
```



4.2 Perform correlation analysis [4 Marks]

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

# Plot Heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=False, cmap='coolwarm')
plt.title('Correlation Heatmap (Training - Numerical)')
plt.show()
```

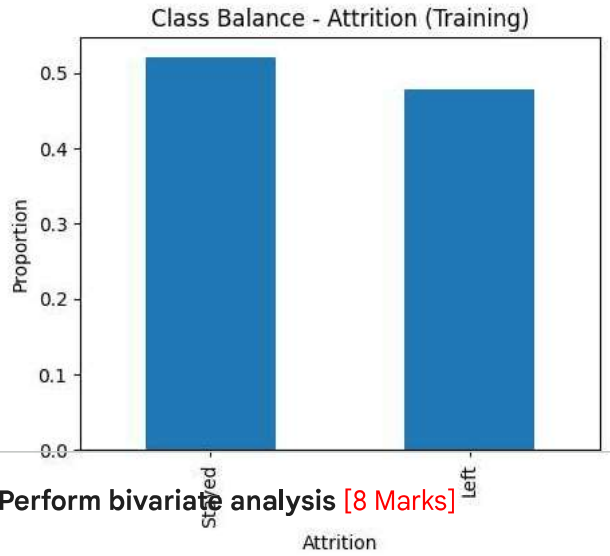


Check the correlation among different numerical variables.

4.3 Check class balance [2 Marks]

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

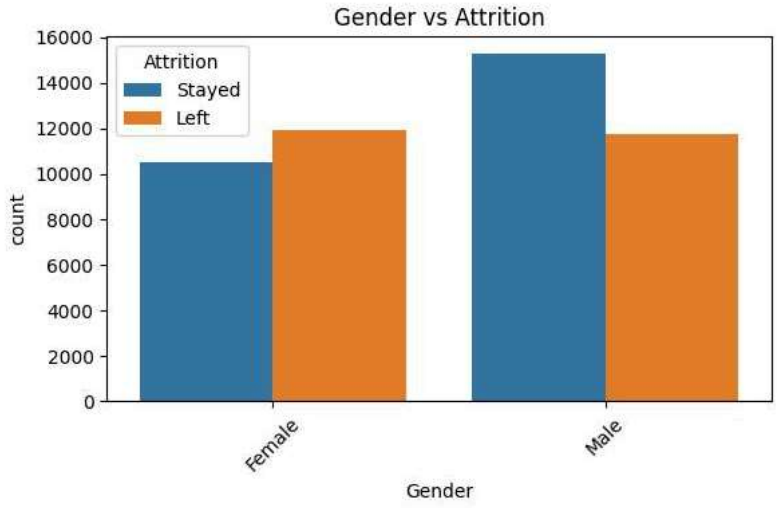
```
# Plot a bar chart to check class balance
plt.figure(figsize=(5,4))
y_train.value_counts(normalize=True).plot(kind='bar')
plt.title('Class Balance - Attrition (Training)')
plt.xlabel('Attrition')
plt.ylabel('Proportion')
plt.show()
```

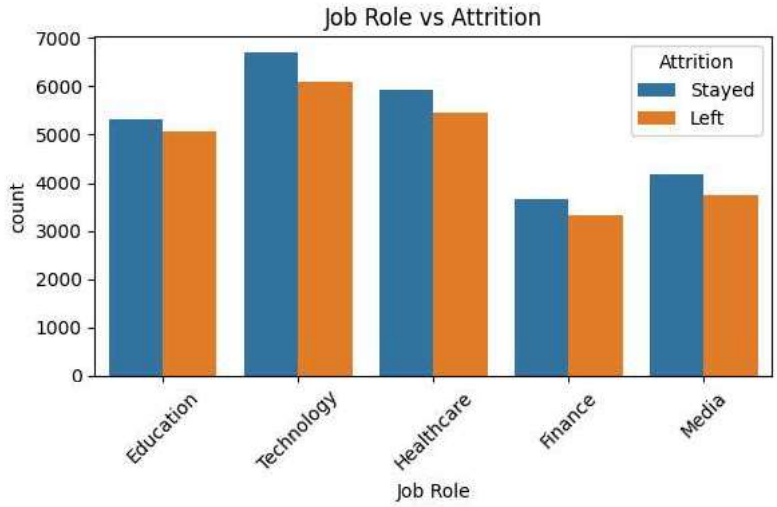


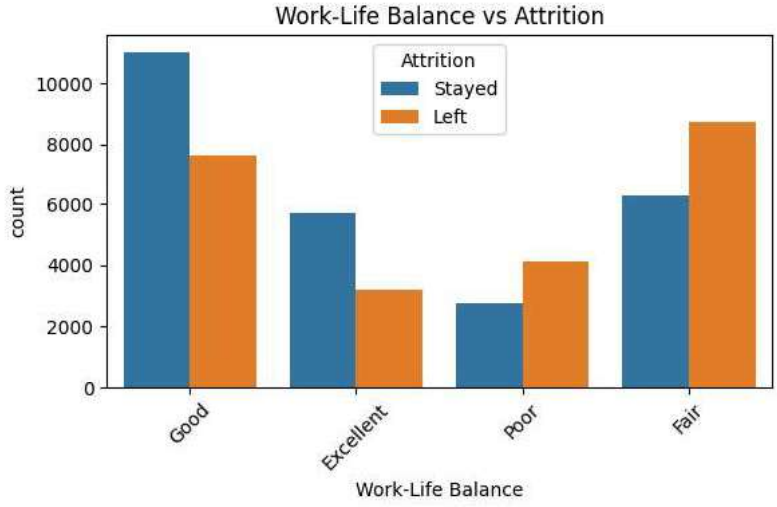
4.4 Perform bivariate analysis [8 Marks]

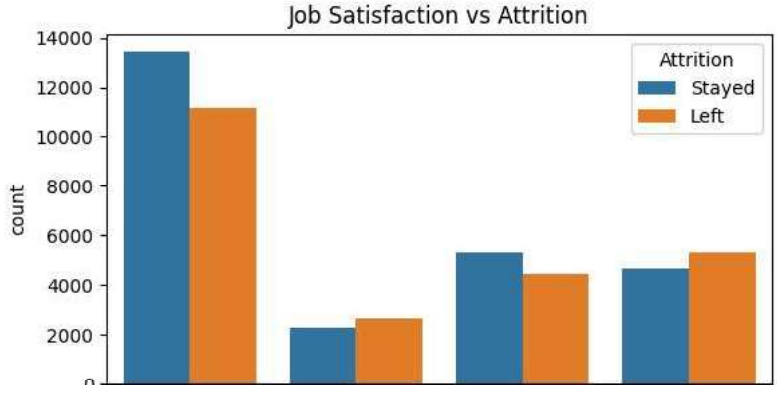
Perform bivariate analysis on training data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

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

for col in cat_cols_train:
    plt.figure(figsize=(6,4))
    sns.countplot(data=X_train.join(y_train), x=col, hue='Attrition')
    plt.title(f'{col} vs Attrition')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```


Gender vs Attrition
Attrition
Stayed
Left
count
16000
14000
12000
10000
8000
6000
4000
2000
0
Female
Male
Gender

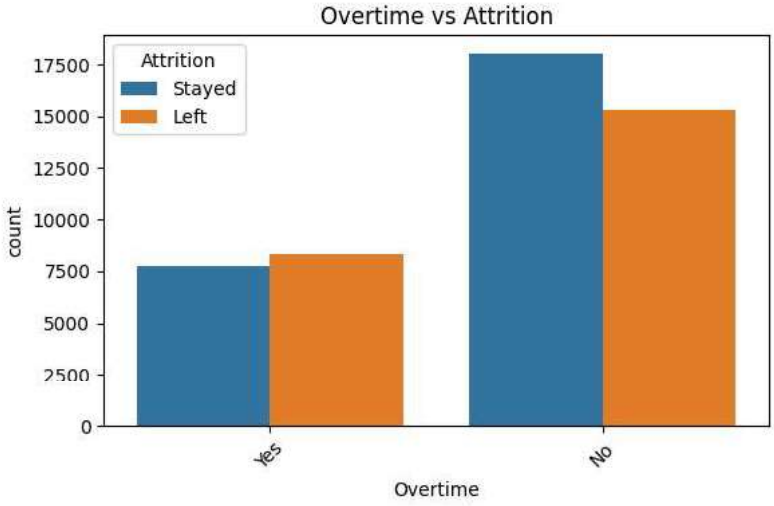

Job Role vs Attrition
Attrition
Stayed
Left
count
7000
6000
5000
4000
3000
2000
1000
0
Education
Technology
Healthcare
Finance
Media
Job Role

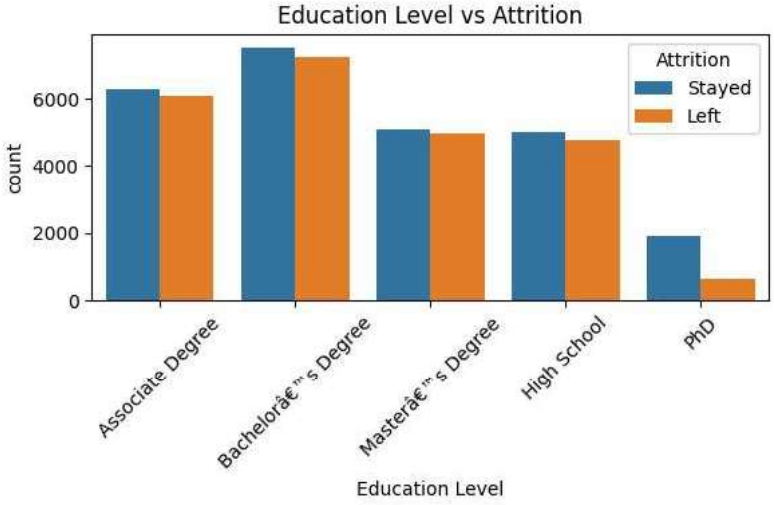

Work-Life Balance vs Attrition
Attrition
Stayed
Left
count
10000
8000
6000
4000
2000
0
Good
Excellent
Poor
Fair
Work-Life Balance


Job Satisfaction vs Attrition
Attrition
Stayed
Left
count
14000
12000
10000
8000
6000
4000
2000

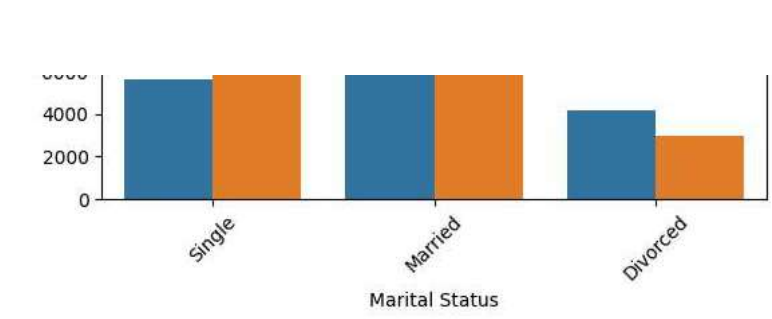

High
Low
Medium
Very High
Job Satisfaction

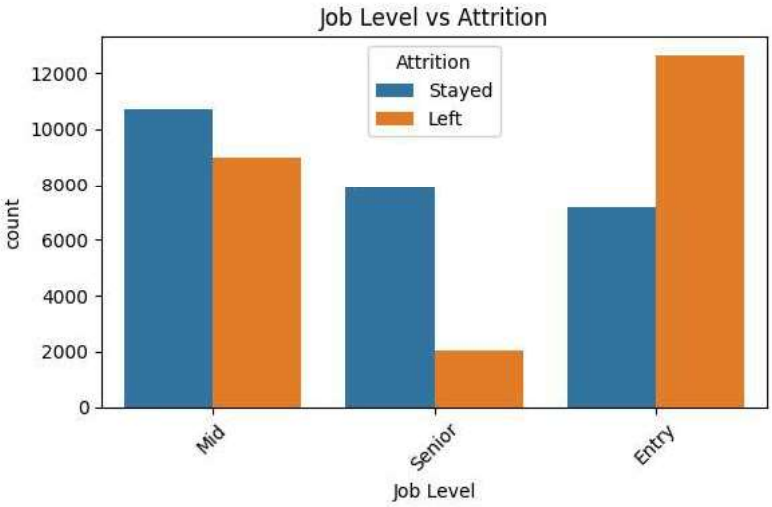

Performance Rating vs Attrition
Attrition
Stayed
Left
count
15000
12500
10000
7500
5000
2500
0
Average
Below Average
High
Low
Performance Rating

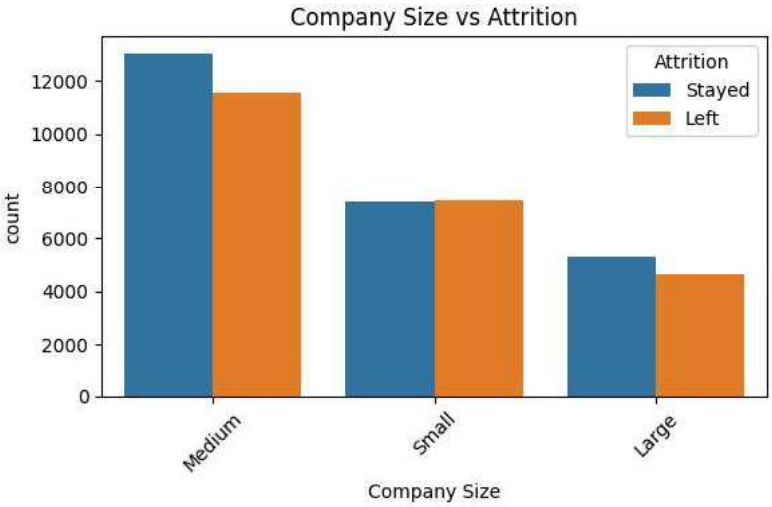

Overtime vs Attrition
Attrition
Stayed
Left
count
17500
15000
12500
10000
7500
5000
2500
0
Yes
No
Overtime

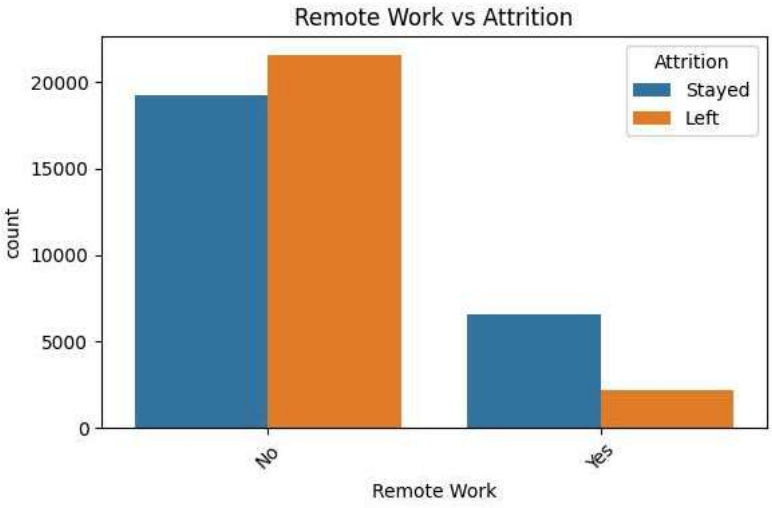

Education Level vs Attrition
Attrition
Stayed
Left
count
6000
4000
2000
0
Associate Degree
Bachelorâ€™s Degree
Masterâ€™s Degree
High School
PhD
Education Level


Marital Status vs Attrition
Attrition
Stayed
Left
count
16000
14000
12000
10000
8000
6000


4000
2000
0
Single
Married
Divorced
Marital Status


Job Level vs Attrition
Attrition
Stayed
Left
12000
10000
8000
6000
4000
2000
0
count
Mid
Senior
Entry
Job Level


Company Size vs Attrition
Attrition
Stayed
Left
12000
10000
8000
6000
4000
2000
0
count
Medium
Small
Large
Company Size


Remote Work vs Attrition
Attrition
Stayed
Left
20000
15000
10000
5000
0
count
No
Yes
Remote Work


Leadership Opportunities vs Attrition
25000
20000
Attrition
Stayed
Left

5. EDA on validation data [OPTIONAL]

5.1 Perform univariate analysis

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

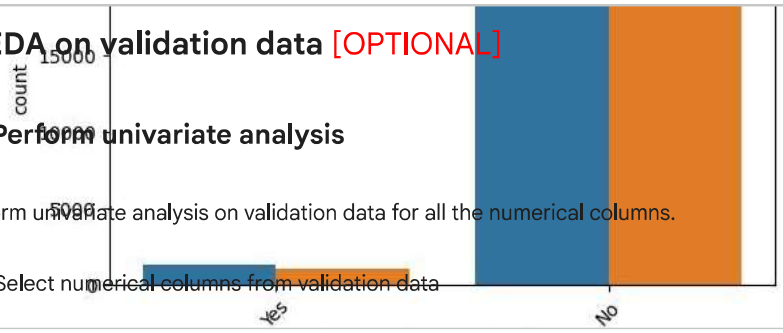
5.1.1 Select numerical columns from validation data

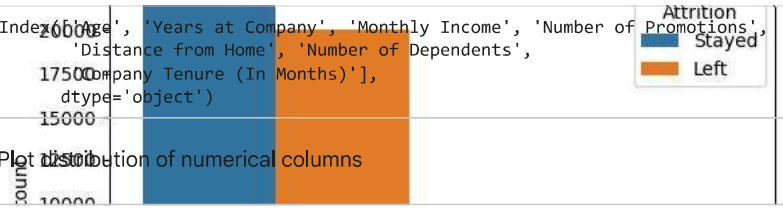
```
# Select numerical columns
num_cols_val = X_validation.select_dtypes(include=[np.number]).columns
num_cols_val
```

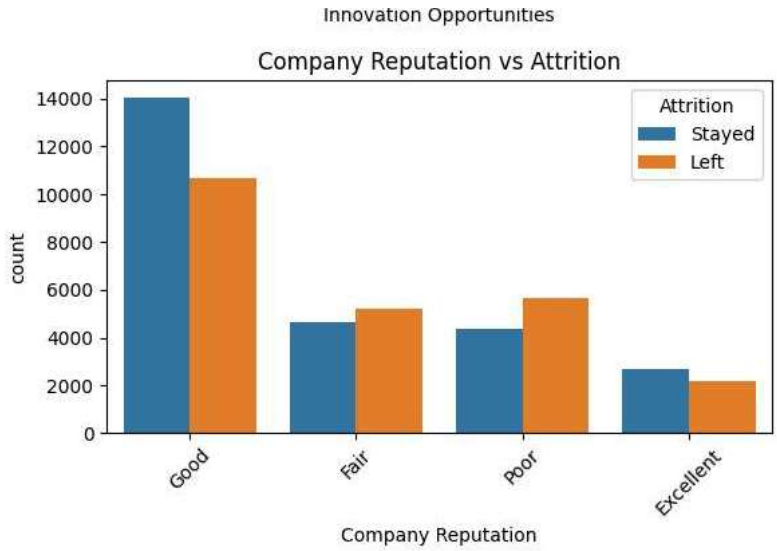
```
Index(['Age', 'Years at Company', 'Monthly Income', 'Number of Promotions',
       'Distance from Home', 'Number of Dependents',
       'Company Tenure (In Months)'],
      dtype='object')
```

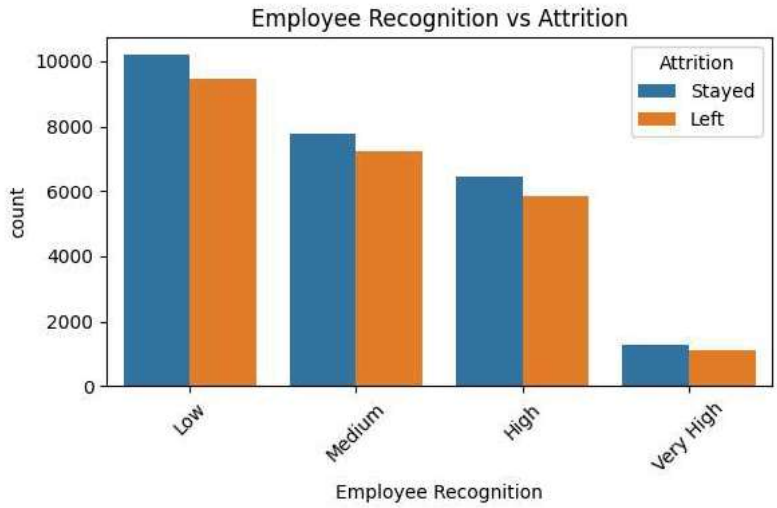
5.1.2 Plot distribution of numerical columns

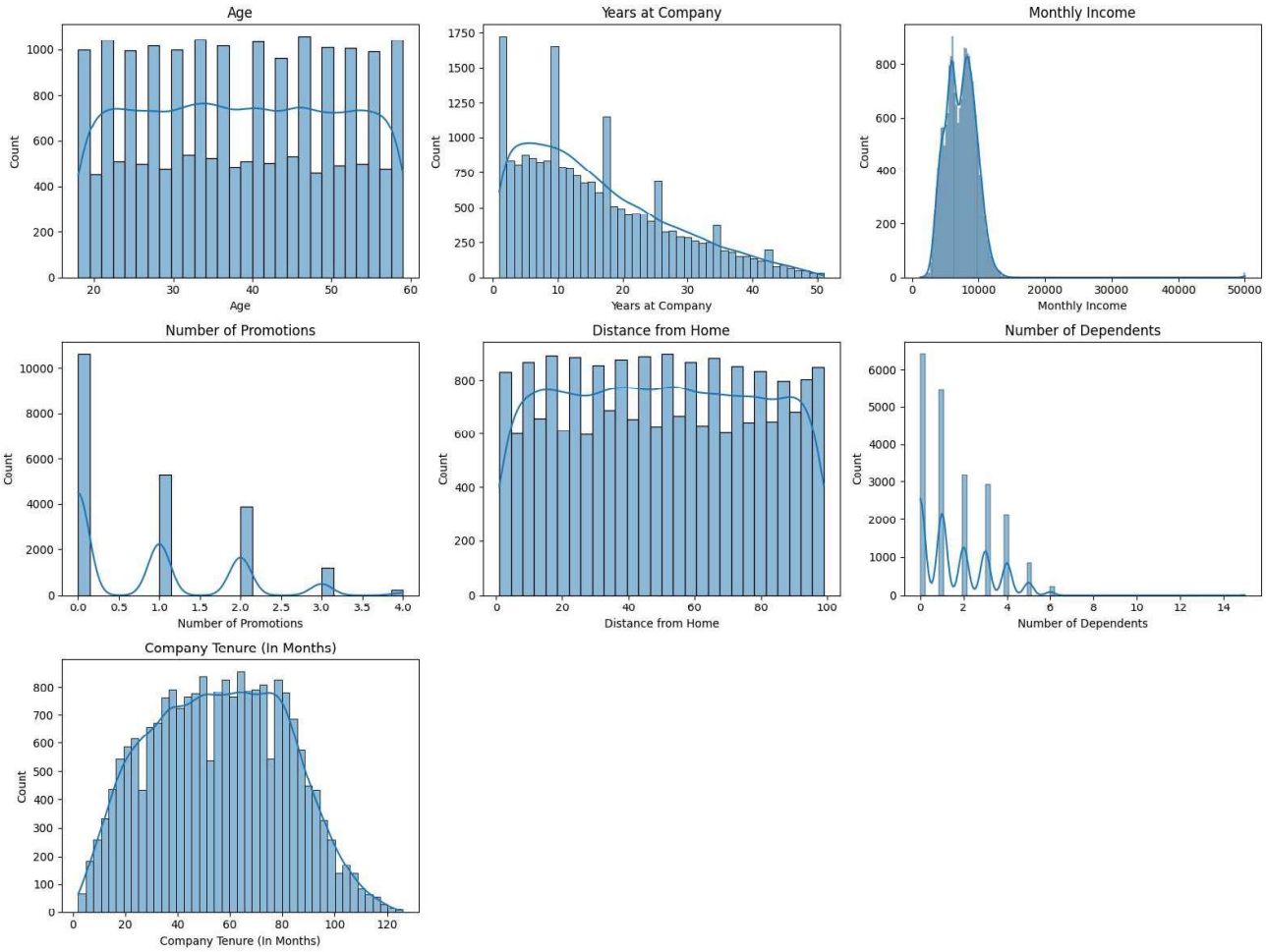
```
# Plot all the numerical columns to understand their distribution
plt.figure(figsize=(16, 12))
for i, col in enumerate(num_cols_val, 1):
    plt.subplot(len(num_cols_val)//3 + 1, 3, i)
    sns.histplot(X_validation[col], kde=True)
    plt.title(col)
plt.tight_layout()
plt.show()
```









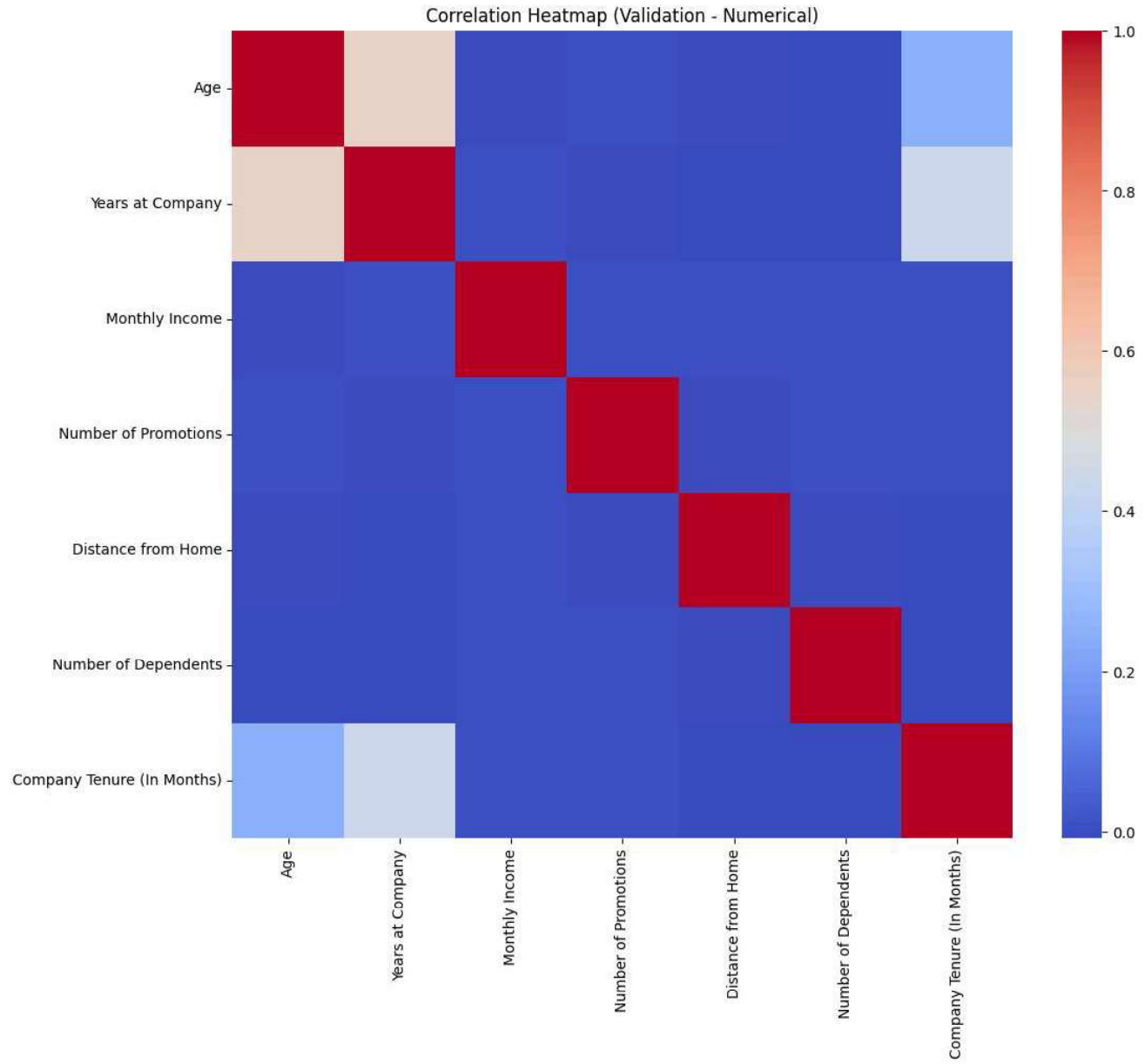


5.2 Perform correlation analysis

Check the correlation among different numerical variables.

```
# Create correlation matrix for numerical columns

# Plot Heatmap of the correlation matrix
corr_val = X_validation[num_cols_val].corr()

plt.figure(figsize=(12,10))
```

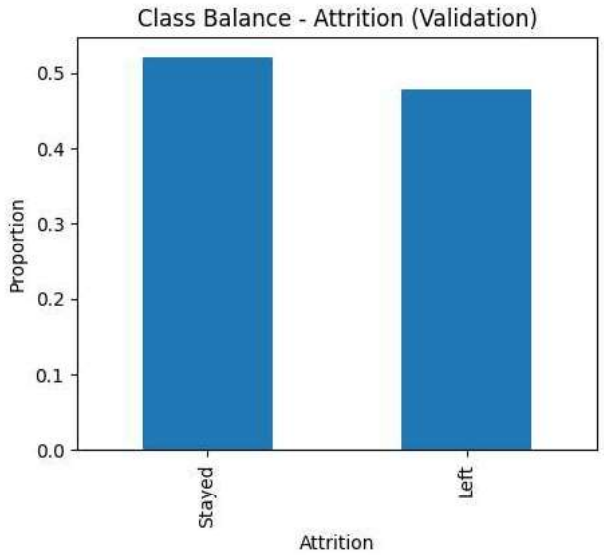
```
sns.heatmap(corr_val, annot=False, cmap='coolwarm')
plt.title('Correlation Heatmap (Validation - Numerical)')
plt.show()
```



5.3 Check class balance

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

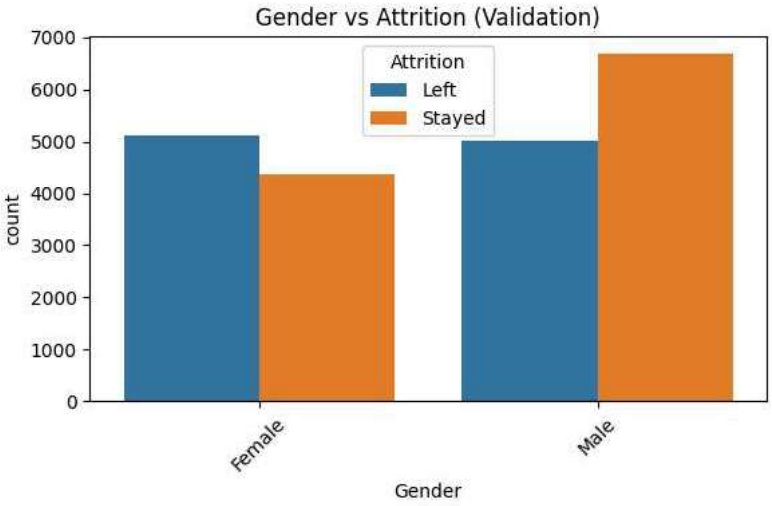
```
# Plot a bar chart to check class balance
plt.figure(figsize=(5,4))
y_validation.value_counts(normalize=True).plot(kind='bar')
plt.title('Class Balance - Attrition (Validation)')
plt.xlabel('Attrition')
plt.ylabel('Proportion')
plt.show()
```

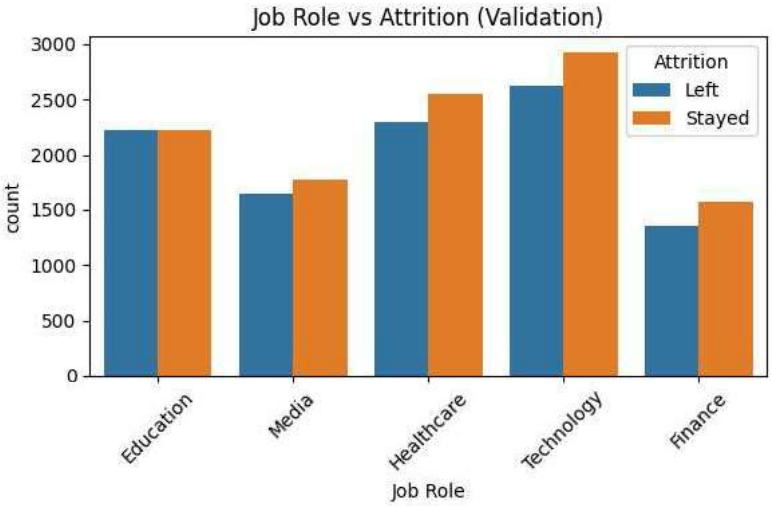


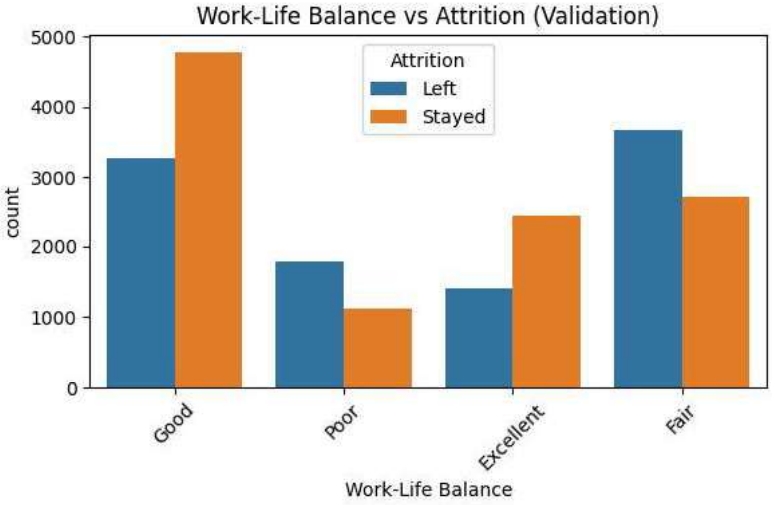
5.4 Perform bivariate analysis

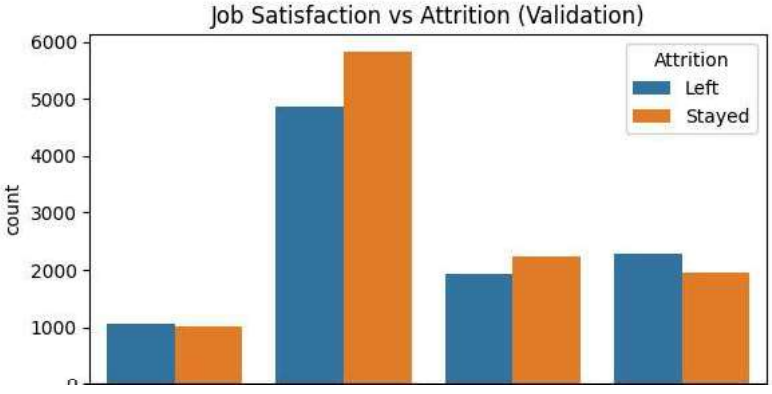
Perform bivariate analysis on validation data between all the categorical columns and target variable to analyse how the categorical variables influence the target variable.

```
# Plot distribution for each categorical column with target variable
cat_cols_val = X_validation.select_dtypes(include=['object']).columns

for col in cat_cols_val:
    plt.figure(figsize=(6,4))
    sns.countplot(data=X_validation.join(y_validation), x=col, hue='Attrition')
    plt.title(f'{col} vs Attrition (Validation)')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

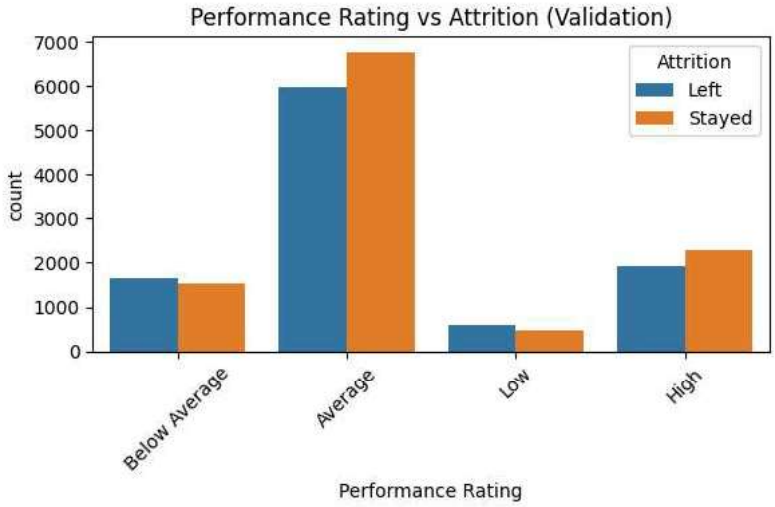

Gender vs Attrition (Validation)
Attrition
Left
Stayed
count
7000
6000
5000
4000
3000
2000
1000
0
Female
Male
Gender

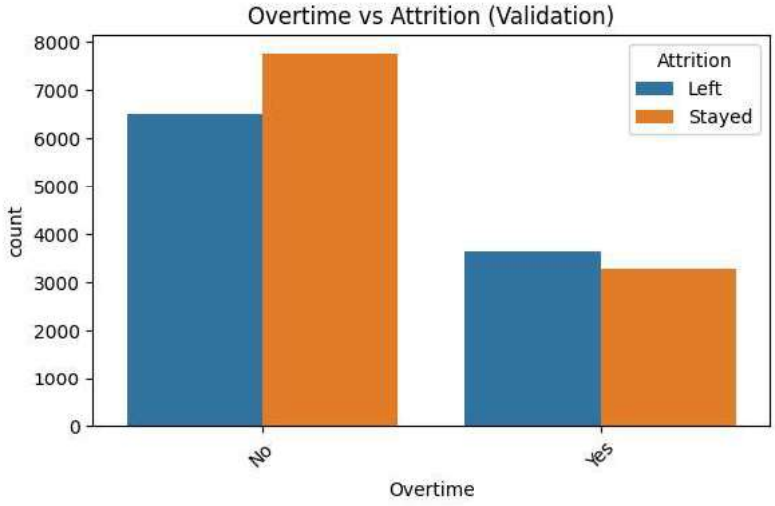

Job Role vs Attrition (Validation)
Attrition
Left
Stayed
count
3000
2500
2000
1500
1000
500
0
Education
Media
Healthcare
Technology
Finance
Job Role

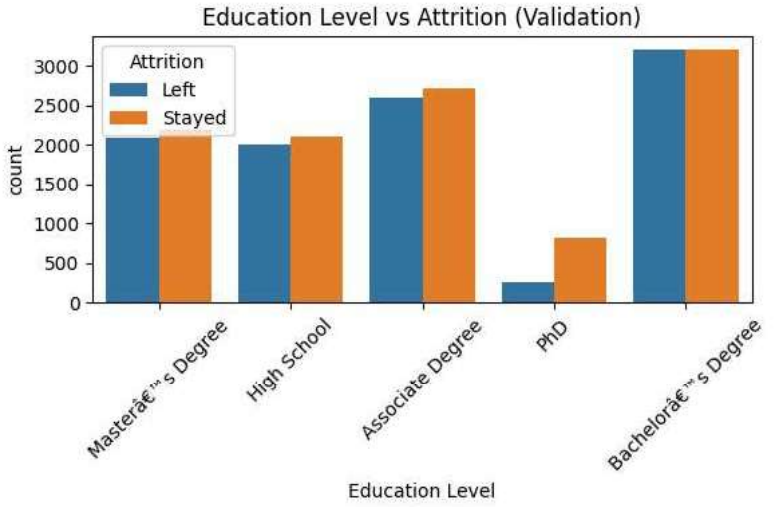

Work-Life Balance vs Attrition (Validation)
Attrition
Left
Stayed
count
5000
4000
3000
2000
1000
0
Good
Poor
Excellent
Fair
Work-Life Balance

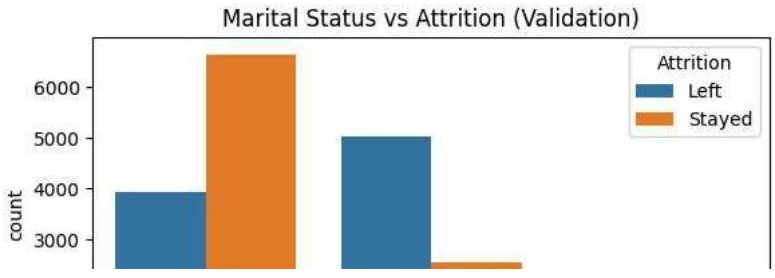

Job Satisfaction vs Attrition (Validation)
Attrition
Left
Stayed
count
6000
5000
4000
3000
2000
1000

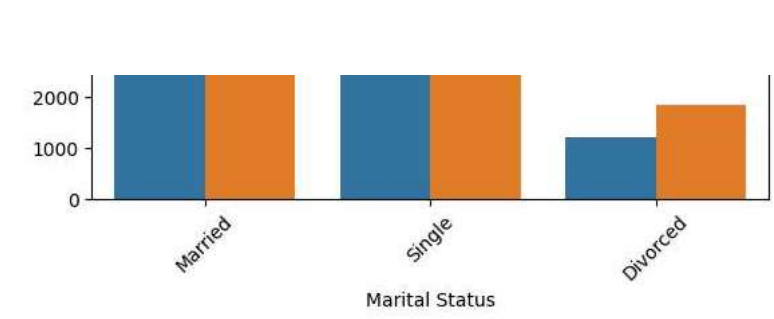

Low
High
Medium
Very High
Job Satisfaction

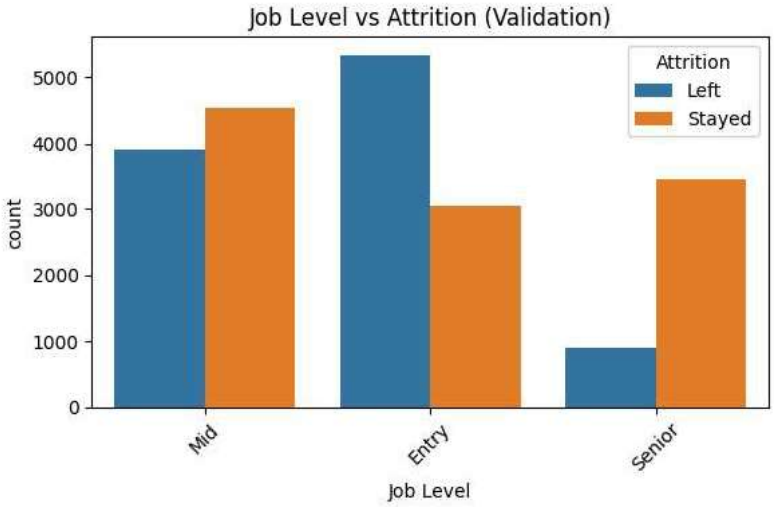

Performance Rating vs Attrition (Validation)
Attrition
Left
Stayed
count
Below Average
Average
Low
High
Performance Rating

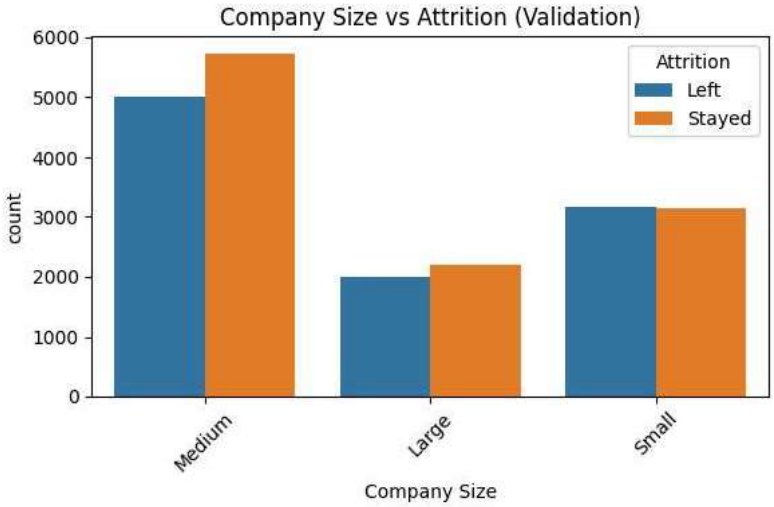

Overtime vs Attrition (Validation)
Attrition
Left
Stayed
count
No
Yes
Overtime

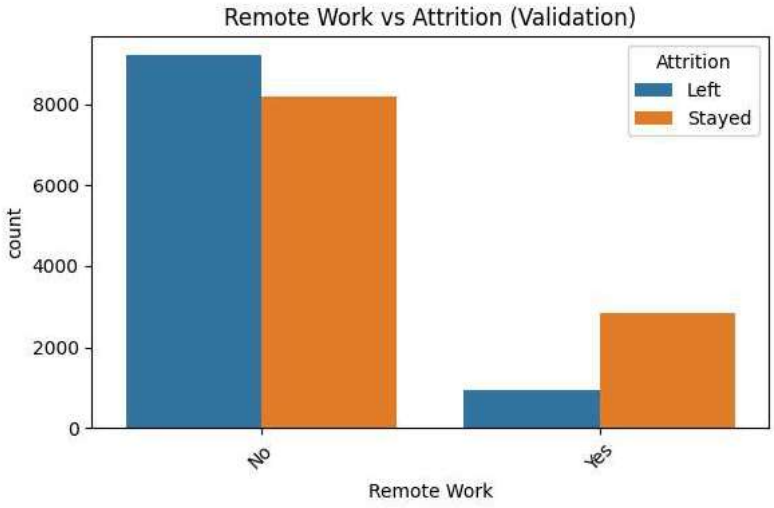

Education Level vs Attrition (Validation)
Attrition
Left
Stayed
count
Masteraé™s Degree
High School
Associate Degree
PhD
Bachelораé™s Degree
Education Level

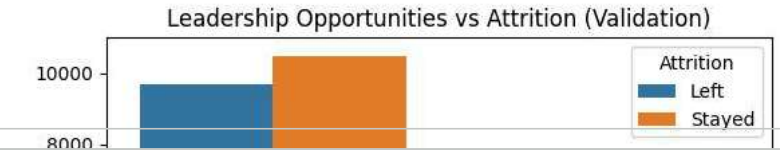

Marital Status vs Attrition (Validation)
Attrition
Left
Stayed
count


2000
1000
0
Married
Single
Divorced
Marital Status


Job Level vs Attrition (Validation)
Attrition
Left
Stayed
count
5000
4000
3000
2000
1000
0
Mid
Entry
Senior
Job Level


Company Size vs Attrition (Validation)
Attrition
Left
Stayed
count
6000
5000
4000
3000
2000
1000
0
Medium
Large
Small
Company Size


Remote Work vs Attrition (Validation)
Attrition
Left
Stayed
count
8000
6000
4000
2000
0
No
Yes
Remote Work


Leadership Opportunities vs Attrition (Validation)
Attrition
Left
Stayed
10000
8000

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
cat_cols = X_train.select_dtypes(include=['object']).columns
cat_cols
```

```
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

# Add the results to the master DataFrame
X_train_dummies = pd.get_dummies(X_train, drop_first=True)
X_train_dummies.head()
```

	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)	Gender_Male	Job Role_Finance	Job Role_Healthcare	...	Comp Size_Sm
55208	57	44	5381	2	27.0	4	90.0	False	False	False	...	Fa
58299	20	9	9466	0	92.0	1	11.0	False	False	False	...	Fa
28222	48	3	11420	0	98.0	2	51.0	True	False	False	...	Fa
49504	41	13	10092	2	53.0	1	82.0	False	False	False	...	T
1146	22	2	7434	0	6.0	0	64.0	False	False	True	...	Fa

5 rows × 41 columns

Now, drop the original categorical columns and check the DataFrame

```
# Drop the original categorical columns and check the DataFrame
X_train_dummies.head()
```

	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)	Gender_Male	Job Role_Finance	Job Role_Healthcare	...	Comp Size_Sm
55208	57	44	5381	2	27.0	4	90.0	False	False	False	...	Fa
58299	20	9	9466	0	92.0	1	11.0	False	False	False	...	Fa
28222	48	3	11420	0	98.0	2	51.0	True	False	False	...	Fa
49504	41	13	10092	2	53.0	1	82.0	False	False	False	...	T
1146	22	2	7434	0	6.0	0	64.0	False	False	True	...	Fa

5 rows × 41 columns

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

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

```
# Add the results to the master DataFrame
X_validation_dummies = pd.get_dummies(X_validation, drop_first=True)

# Align the validation columns to match the training columns
X_validation_dummies = X_validation_dummies.reindex(columns=X_train_dummies.columns, fill_value=0)
X_validation_dummies.head()
```

	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)	Gender_Male	Job Role_Finance	Job Role_Healthcare	...	Comp Size_Sm
4944	42	16	4922	0	52.0	1	70.0	False	False	False	...	Fa
66942	38	3	6388	0	67.0	1	56.0	True	False	False	...	Fa
31609	39	23	7547	0	88.0	1	50.0	False	False	True	...	Fa
51753	28	12	6036	3	52.0	0	37.0	False	False	False	...	Fa
44446	31	14	7813	4	93.0	5	15.0	True	False	True	...	Fa

5 rows × 41 columns

Now, drop the original categorical columns and check the DataFrame

```
# Drop categorical columns and check the DataFrame
X_validation_dummies.head()
```

	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)	Gender_Male	Job Role_Finance	Job Role_Healthcare	...	Comp Size_Sm
4944	42	16	4922	0	52.0	1	70.0	False	False	False	...	Fa
66942	38	3	6388	0	67.0	1	56.0	True	False	False	...	Fa
31609	39	23	7547	0	88.0	1	50.0	False	False	True	...	Fa
51753	28	12	6036	3	52.0	0	37.0	False	False	False	...	Fa
44446	31	14	7813	4	93.0	5	15.0	True	False	True	...	Fa

5 rows × 41 columns

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_df = pd.DataFrame(y_train)
y_validation_df = pd.DataFrame(y_validation)
y_train_df.head(), y_validation_df.head()
```

```
(      Attrition
 55208    Stayed
 58299    Stayed
 28222      Left
 49504    Stayed
 1146       Left,
       Attrition
 4944       Left
 66942      Left
 31609    Stayed
 51753    Stayed
 44446      Left)
```

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_dummies = pd.get_dummies(y_train_df['Attrition'], drop_first=True)
y_train_dummies.columns = ['Attrition_Stayed']  # if 'Left' is base
y_train_dummies.head()
```

	Attrition_Stayed
55208	True
58299	True
28222	False
49504	True
1146	False

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_validation_dummies = pd.get_dummies(y_validation_df['Attrition'], drop_first=True)
y_validation_dummies.columns = ['Attrition_Stayed']
y_validation_dummies.head()
```

	Attrition_Stayed
4944	False
66942	False
31609	True
51753	True
44446	False

6.1.7 Drop redundant columns [1 Mark]

```
# Drop redundant columns from both train and validation
# Here we will use y_train_dummies and y_validation_dummies as target going forward
y_train_final = y_train_dummies.copy()
y_validation_final = y_validation_dummies.copy()
```

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

# Scale the numerical features present in the validation set
scaler = StandardScaler()

X_train_scaled = X_train_dummies.copy()
X_validation_scaled = X_validation_dummies.copy()

num_cols_for_scale = X_train.select_dtypes(include=[np.number]).columns

X_train_scaled[num_cols_for_scale] = scaler.fit_transform(X_train_scaled[num_cols_for_scale])
X_validation_scaled[num_cols_for_scale] = scaler.transform(X_validation_scaled[num_cols_for_scale])

X_train_scaled.head()
```

	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure (In Months)	Gender_Male	Job Role_Finance	Job Role_Healthcare	...
55208	1.528049	2.526735	-0.765453	1.172861	-0.806784	1.489475	1.351007	False	False	False	...
58299	-1.534625	-0.596692	0.828294	-0.835796	1.467305	-0.417685	-1.763741	False	False	False	...
28222	0.783074	-1.132136	1.590639	-0.835796	1.677221	0.218035	-0.186653	True	False	False	...
49504	0.203649	-0.239729	1.072525	1.172861	0.102851	-0.417685	1.035590	False	False	False	...
1146	-1.369075	-1.221377	0.035517	-0.835796	-1.541490	-1.053404	0.325900	False	False	True	...

5 rows × 41 columns

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(max_iter=1000, solver='liblinear')
```

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_scaled, y_train_final.values.ravel())
```

```
# Display the features selected by RFE
selected_cols = X_train_scaled.columns[rfe.support_]
selected_cols
```

```
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 = selected_cols.tolist()
col
```

```
['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']
```

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_rfe = X_train_scaled[col]
```

```
# View the training data
X_train_rfe.head()
```

	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	Edu Lev
55208	False	False	False	False	False	False	False	True	
58299	False	False	False	False	False	True	False	False	
28222	True	False	False	False	False	False	False	True	
49504	False	False	False	False	False	False	False	False	
1146	False	False	False	True	False	False	False	False	

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_rfe)
X_train_sm.head()
```

	const	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_Y
55208	1.0	False	False	False	False	False	False	False	T
58299	1.0	False	False	False	False	False	True	False	Fa
28222	1.0	True	False	False	False	False	False	False	T
49504	1.0	False	False	False	False	False	False	False	Fa
1146	1.0	False	False	False	True	False	False	False	Fa

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
# Convert boolean columns in X_train_sm to integers (0 or 1)
X_train_sm_numeric = X_train_sm.astype(int)

# Convert the target variable to integers (0 or 1)
y_train_final_numeric = y_train_final['Attrition_Stayed'].astype(int)

logit_model = sm.Logit(y_train_final_numeric, X_train_sm_numeric)
result = logit_model.fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.506545
         Iterations 6
```

Logit Regression Results

Dep. Variable:	Attrition_Stayed	**No. Observations:**	49444
Model:	Logit	**Df Residuals:**	49428
Method:	MLE	**Df Model:**	15
Date:	Sat, 20 Dec 2025	**Pseudo R-squ.:**	0.2683
Time:	14:38:57	**Log-Likelihood:**	-25046.
converged:	True	**LL-Null:**	-34228.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.2568	0.028	9.081	0.000	0.201	0.312
Gender_Male	0.5724	0.022	25.795	0.000	0.529	0.616
Work-Life Balance_Fair	-1.0797	0.025	-42.843	0.000	-1.129	-1.030
Work-Life Balance_Poor	-1.2145	0.033	-36.261	0.000	-1.280	-1.149
Job Satisfaction_Low	-0.4946	0.037	-13.326	0.000	-0.567	-0.422
Job Satisfaction_Very High	-0.4663	0.028	-16.863	0.000	-0.520	-0.412
Performance Rating_Below Average	-0.3246	0.031	-10.515	0.000	-0.385	-0.264
Performance Rating_Low	-0.5816	0.051	-11.343	0.000	-0.682	-0.481
Overtime_Yes	-0.3313	0.023	-14.137	0.000	-0.377	-0.285
Education Level_PhD	1.5135	0.056	27.231	0.000	1.405	1.622
Marital Status_Single	-1.7083	0.025	-69.283	0.000	-1.757	-1.660
Job Level_Mid	0.9664	0.024	39.963	0.000	0.919	1.014
Job Level_Senior	2.4948	0.034	72.332	0.000	2.427	2.562
Remote Work_Yes	1.7120	0.032	53.268	0.000	1.649	1.775
Company Reputation_Fair	-0.5427	0.028	-19.061	0.000	-0.598	-0.487
Company Reputation_Poor	-0.7437	0.028	-26.110	0.000	-0.800	-0.688

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

vif_data = pd.DataFrame()
vif_data['feature'] = X_train_rfe.columns
vif_data['VIF'] = [variance_inflation_factor(X_train_rfe.values, i)
                   for i in range(X_train_rfe.shape[1])]
```

```
# Make a VIF DataFrame for all the variables present
vif_data
```

	feature	VIF
0	Gender_Male	0.000068
1	Work-Life Balance_Fair	0.000094
2	Work-Life Balance_Poor	0.000172
3	Job Satisfaction_Low	0.000224
4	Job Satisfaction_Very High	0.000123
5	Performance Rating_Below Average	0.000157
6	Performance Rating_Low	0.000425
7	Overtime_Yes	0.000086
8	Education Level_PhD	0.000414
9	Marital Status_Single	0.000082
10	Job Level_Mid	0.000084
11	Job Level_Senior	0.000133
12	Remote Work_Yes	0.000135
13	Company Reputation_Fair	0.000127
14	Company Reputation_Poor	0.000126

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_prob = result.predict(X_train_sm_numeric)
y_train_pred_prob.head()
```

	0
55208	0.306523
58299	0.543879
28222	0.488797
49504	0.772961
1146	0.314219

dtype: float64

7.2.6 Format the prediction output [1 Mark]

```
# Reshape it into an array
y_train_pred_prob_arr = np.array(y_train_pred_prob)
y_train_pred_prob_arr[:5]
```

```
array([[0.30652318],
       [0.54387905],
       [0.48879672],
       [0.77296091],
       [0.31421893]])
```

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
train_pred_df = pd.DataFrame({
    'Actual': y_train_final['Attrition_Stayed'].values,
    'Predicted_Prob': y_train_pred_prob_arr.flatten() # Flatten the 2D array to 1D
})
train_pred_df.head()
```

	Actual	Predicted_Prob
0	True	0.306523
1	True	0.543879
2	False	0.488797
3	True	0.772961
4	False	0.314219

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
train_pred_df['Predicted'] = (train_pred_df['Predicted_Prob'] >= 0.5).astype(int)
train_pred_df.head()
```

	Actual	Predicted_Prob	Predicted	Final_Pred
0	True	0.306523	0	0
1	True	0.543879	1	1
2	False	0.488797	0	0
3	True	0.772961	1	1
4	False	0.314219	0	0

```
Start coding or generate with AI.
```

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_train_05 = metrics.accuracy_score(train_pred_df['Actual'], train_pred_df['Predicted'])
accuracy_train_05
```

```
0.7375212361459429
```

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

```
# Create confusion matrix
conf_matrix = metrics.confusion_matrix(train_pred_df['Actual'], train_pred_df['Predicted'])
conf_matrix
```

```
array([[16996,  6681],
       [ 6297, 19470]])
```

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 = conf_matrix.ravel()
tn, fp, fn, tp
```

```
(np.int64(16996), np.int64(6681), np.int64(6297), np.int64(19470))
```

7.2.12 Calculate sensitivity and specificity of model [2 Marks]

```
# Calculate sensitivity
sensitivity = tp / (tp + fn)
sensitivity
```

```
np.float64(0.7556176504831762)
```

```
# Calculate specificity
specificity = tn / (tn + fp)
specificity
```

```
np.float64(0.7178274274612493)
```

7.2.13 Calculate precision and recall of model [2 Marks]

```
# Calculate precision
precision = tp / (tp + fp)
recall = tp / (tp + fn)
precision
```

```
np.float64(0.7445221980039004)
```

```
# Calculate recall
recall = tp / (tp + fn)
recall
```

```
np.float64(0.7556176504831762)
```

7.3 Find the optimal cutoff [15 marks]

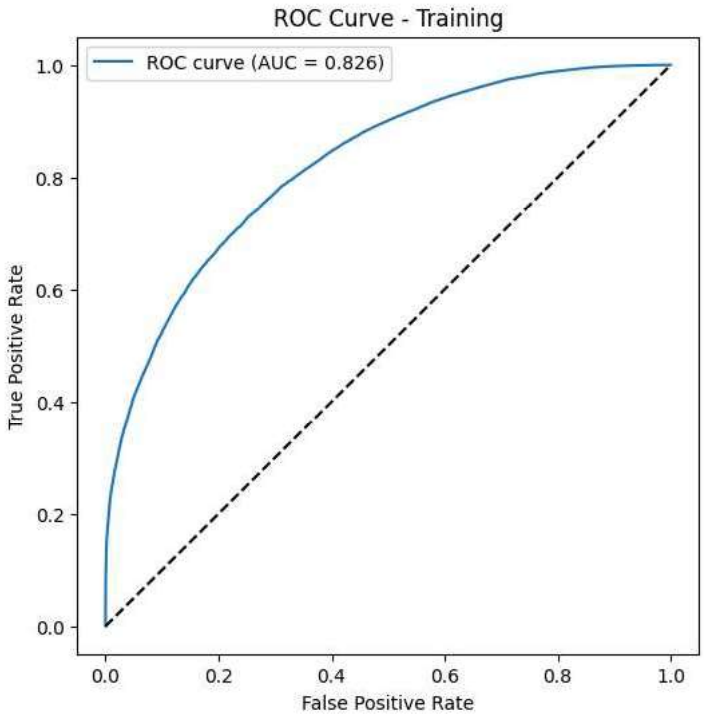
Find the optimal cutoff to improve model performance. While a default threshold of 0.5 was used for initial evaluation, optimising this threshold can enhance the model's performance.

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

fpr, tpr, thresholds = roc_curve(train_pred_df['Actual'], train_pred_df['Predicted_Prob'])
auc_score = roc_auc_score(train_pred_df['Actual'], train_pred_df['Predicted_Prob'])
```

```
# Call the ROC function
plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc_score:.3f})')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Training')
plt.legend()
plt.show()
```



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 perf
cutoffs = np.linspace(0.1, 0.9, 9)
cutoffs
```

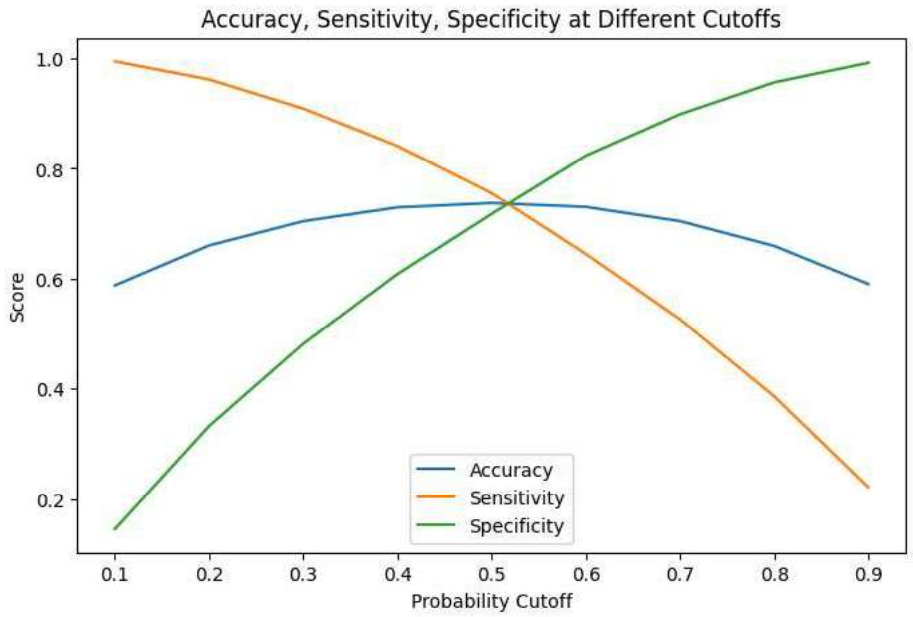
```
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

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_stats = []

for c in cutoffs:
    predicted = (train_pred_df['Predicted_Prob'] >= c).astype(int)
    tn_c, fp_c, fn_c, tp_c = metrics.confusion_matrix(train_pred_df['Actual'], predicted).ravel()
    sensitivity_c = tp_c / (tp_c + fn_c)
    specificity_c = tn_c / (tn_c + fp_c)
    accuracy_c = metrics.accuracy_score(train_pred_df['Actual'], predicted)
    cutoff_stats.append([c, accuracy_c, sensitivity_c, specificity_c])

cutoff_df = pd.DataFrame(cutoff_stats, columns=['Cutoff', 'Accuracy', 'Sensitivity', 'Specificity'])
cutoff_df
```

	Cutoff	Accuracy	Sensitivity	Specificity
0	0.1	0.587938	0.994916	0.145035
1	0.2	0.660464	0.962083	0.332221
2	0.3	0.704494	0.909225	0.481691
3	0.4	0.729654	0.841037	0.608439
4	0.5	0.737521	0.755618	0.717827
5	0.6	0.730544	0.645166	0.823457
6	0.7	0.704656	0.526177	0.898889
7	0.8	0.659534	0.386386	0.956794
8	0.9	0.590345	0.220903	0.992398

```
# Plot accuracy, sensitivity, and specificity at different values of probability cutoffs
plt.figure(figsize=(8,5))
plt.plot(cutoff_df['Cutoff'], cutoff_df['Accuracy'], label='Accuracy')
plt.plot(cutoff_df['Cutoff'], cutoff_df['Sensitivity'], label='Sensitivity')
plt.plot(cutoff_df['Cutoff'], cutoff_df['Specificity'], label='Specificity')
plt.xlabel('Probability Cutoff')
plt.ylabel('Score')
plt.title('Accuracy, Sensitivity, Specificity at Different Cutoffs')
plt.legend()
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 = cutoff_df.iloc[(cutoff_df['Sensitivity'] - cutoff_df['Specificity']).abs().argmin()]['Cutoff']
optimal_cutoff
```

```
np.float64(0.5)
```

7.3.5 Calculate model's accuracy [1Mark]

```
# Calculate the accuracy
train_pred_df['Final_Pred'] = (train_pred_df['Predicted_Prob'] >= optimal_cutoff).astype(int)
final_accuracy_train = metrics.accuracy_score(train_pred_df['Actual'], train_pred_df['Final_Pred'])
final_accuracy_train
```

```
0.7375212361459429
```

7.3.6 Create confusion matrix [1Mark]

```
# Create the confusion matrix once again
conf_matrix_opt = metrics.confusion_matrix(train_pred_df['Actual'], train_pred_df['Final_Pred'])
conf_matrix_opt
```

```
array([[16996,  6681],
       [ 6297, 19470]])
```

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_o, fp_o, fn_o, tp_o = conf_matrix_opt.ravel()
tn_o, fp_o, fn_o, tp_o
```

```
(np.int64(16996), np.int64(6681), np.int64(6297), np.int64(19470))
```

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

```
# Calculate Sensitivity
sens_opt = tp_o / (tp_o + fn_o)
sens_opt
```

```
np.float64(0.7556176504831762)
```

```
# Calculate Specificity
spec_opt = tn_o / (tn_o + fp_o)
spec_opt
```

```
np.float64(0.7178274274612493)
```

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

```
# Calculate Precision
prec_opt = tp_o / (tp_o + fp_o)
prec_opt
```

```
np.float64(0.7445221980039004)
```

```
# Calculate Recall
rec_opt = tp_o / (tp_o + fn_o)
rec_opt
```

```
np.float64(0.7556176504831762)
```

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
precisions, recalls, pr_thresholds = precision_recall_curve(
```