# Predicting Employee Retention:

## Logistic Regression Model Report

## 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.

## 1.Data Understanding

Employee retention datasets typically include features such as age, salary, job role, satisfaction level, tenure, performance score, work–life balance indicators, and whether the employee stayed with the company or left. This step involves reviewing the meaning, type, and expected ranges of each variable. Understanding the characteristics of the target variable is important because logistic regression deals with binary classification. The distribution of retained vs. non-retained employees helps us understand imbalance issues, which may influence model performance.

### 1.1 Importing Libraries

Essential Python libraries such as **pandas, NumPy, matplotlib, seaborn, and scikit-learn** were imported for data analysis and modelling.

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

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

### 1.2 Loading the Dataset

The employee dataset was loaded into a pandas DataFrame and inspected using functions like:

- df.head()
- df.info()
- df.describe()

```python
# Load the dataset

df = pd.read_csv(r"C:/Users/AMALDAS/Downloads/NEW ASSIGNMENT/Employee_data.csv")
df.head()
```

```python
# Check the first few entries
df.head(10)
```

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

## 2. Data Cleaning

Robust data cleaning procedures were applied to ensure dataset quality and model reliability:

### 2.1 Handling Missing Values

- Missing values in numerical columns were detected using isnull() and descriptive summaries.
- Depending on variable characteristics, imputation strategies such as **mean**, **median**, or **mode** imputation were applied.
- For categorical variables, missing values were replaced using the most frequent category.

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

```python
# Check the percentage of missing values in each column

(df.isnull().sum()/len(df)) * 100
```

```python
# Handle the missing value rows in the column
df.dropna(inplace=True)
df.shape
```

```
(70635, 24)
```

### 2.2 Detecting and Treating Outliers

- Numerical features were visually inspected using distribution plots.
- Extreme values were assessed to determine whether they represented data errors or genuine cases.
- Irrelevant or erroneous entries, if any, were removed to maintain dataset integrity.

```python
# Write a function to display the categorical columns with their unique values and check fo
def dsp_cat_unq_values(df):
  categorical_cols = df.select_dtypes(include='object').columns
  for col in categorical_cols:
    print(f"Unique values in '{col}': {df[col].unique()}")

dsp_cat_unq_values(df)
```

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

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

### 2.3 Data Type Adjustments

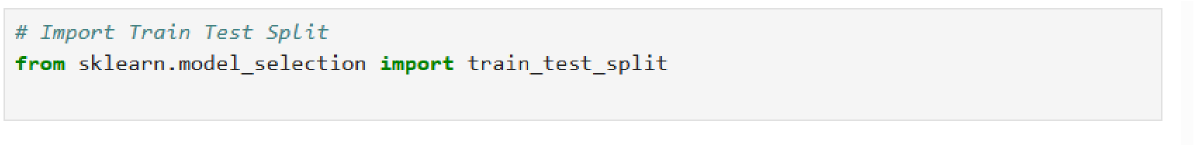
- Columns with inconsistent data types were converted to appropriate numeric or categorical formats.
- Categorical variables were explicitly encoded to prevent issues during modelling.

These steps ensured that the dataset was free of inconsistencies and ready for feature engineering and modelling.

## 3. Train–Validation Split

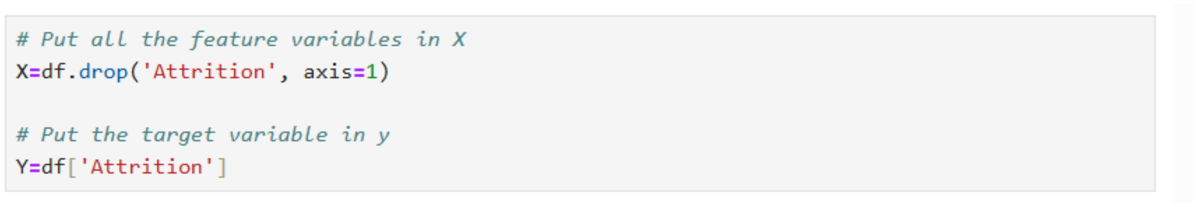
The dataset was partitioned into **80% training data** and **20% validation data**. This ensures that model learning and performance evaluation occur on separate portions of the data.

### 3.1 Import required libraries



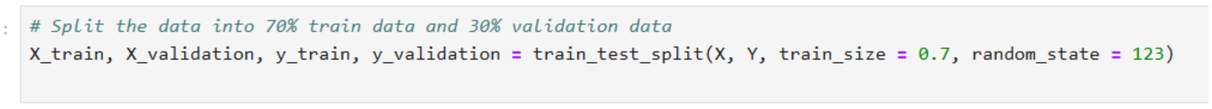
```python
# Import Train Test Split
from sklearn.model_selection import train_test_split
```

### 3.2 **Define feature and target variables**



```python
# Put all the feature variables in X
X=df.drop('Attrition', axis=1)

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

### 3.3 **Split the data**



```python
# Split the data into 70% train data and 30% validation data
X_train, X_validation, y_train, y_validation = train_test_split(X, Y, train_size = 0.7, random_state = 123)
```

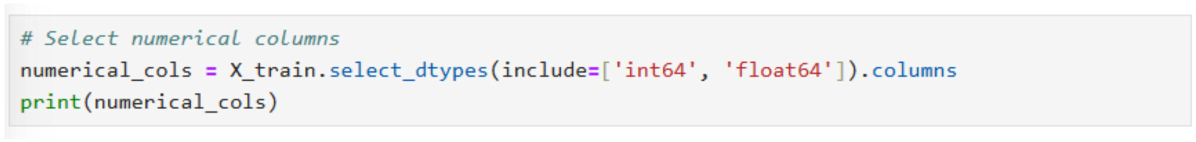
## 4. Exploratory Data Analysis (EDA) on Training Data

In-depth EDA was performed on the training dataset to uncover meaningful trends and relationships:

### 4.1 Univariate Analysis

- **Numerical features** were analysed using histograms and summary statistics.
- Distribution shapes were inspected for skewness, variance, and modality.
- **Categorical features** were explored using frequency counts and bar charts.

This step revealed variations in satisfaction levels, tenure, performance ratings, and age groups among employees.

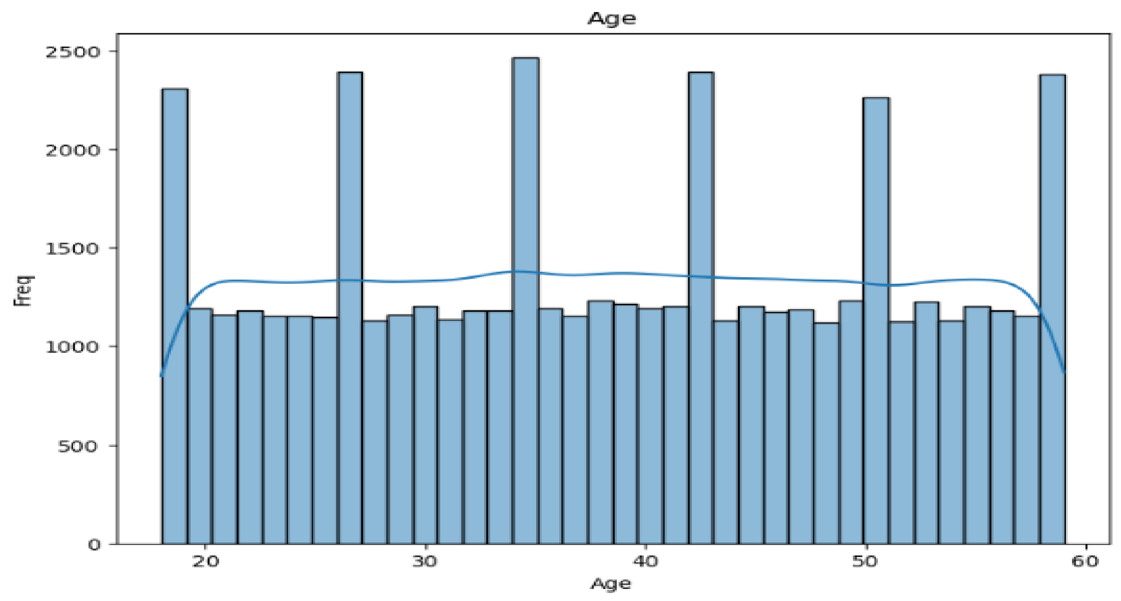


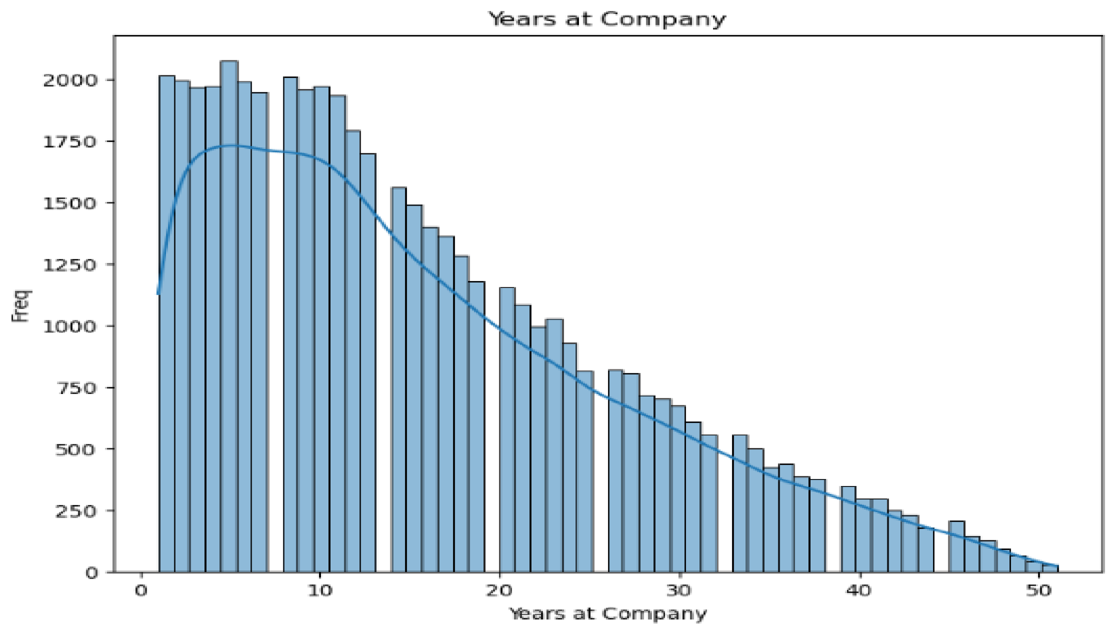
```python
# Select numerical columns
numerical_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
print(numerical_cols)
```

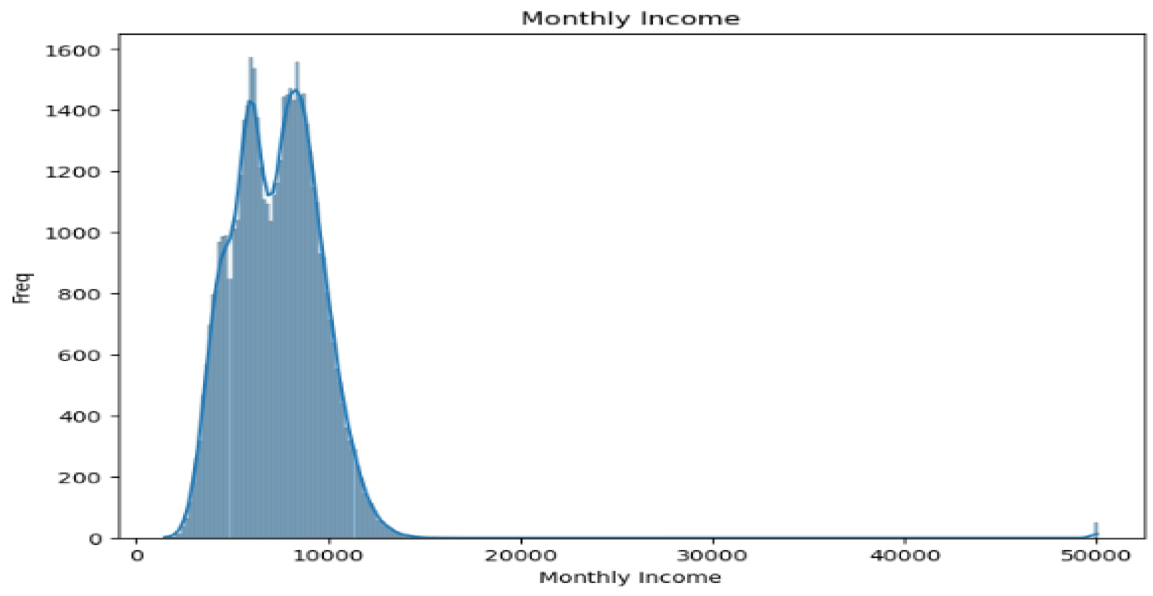
```python
# Plot all the numerical columns to understand their distribution

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

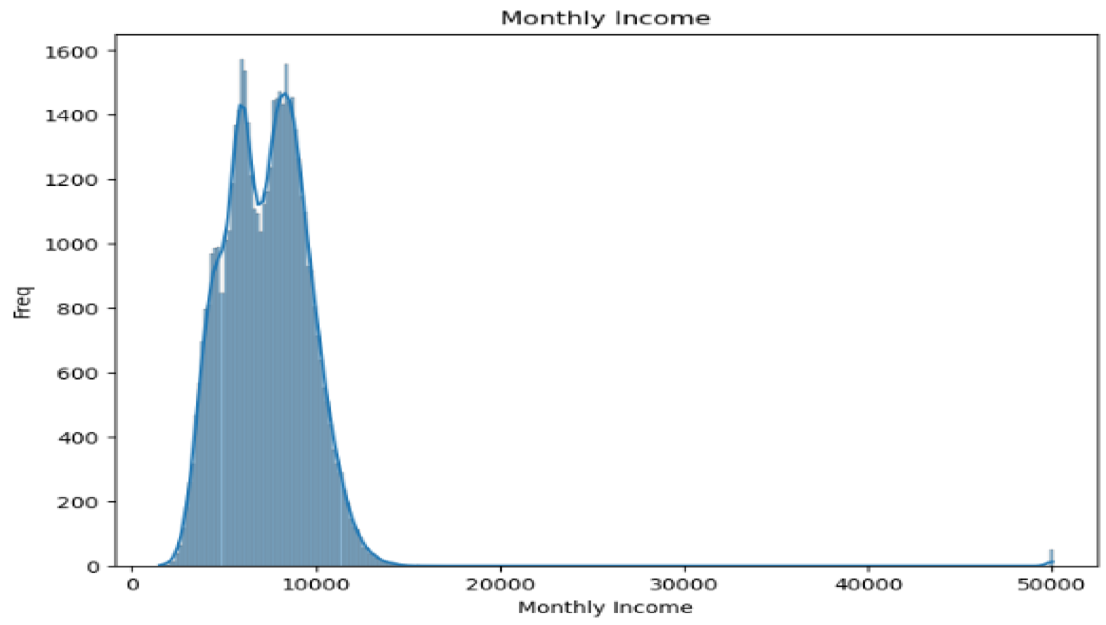
[22]:

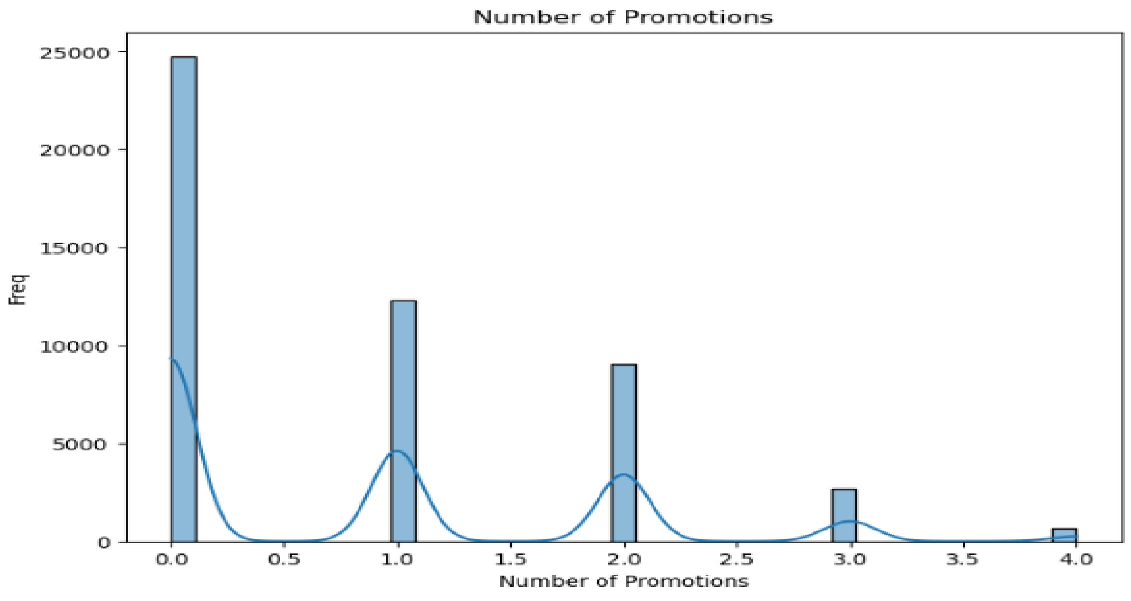
```python
for col in numerical_cols:
    plt.figure(figsize=(8, 6))
    sns.histplot(X_train[col], kde=True)
    plt.title(f'{col}')
    plt.xlabel(col)
    plt.ylabel('Freq')
    plt.show()
```

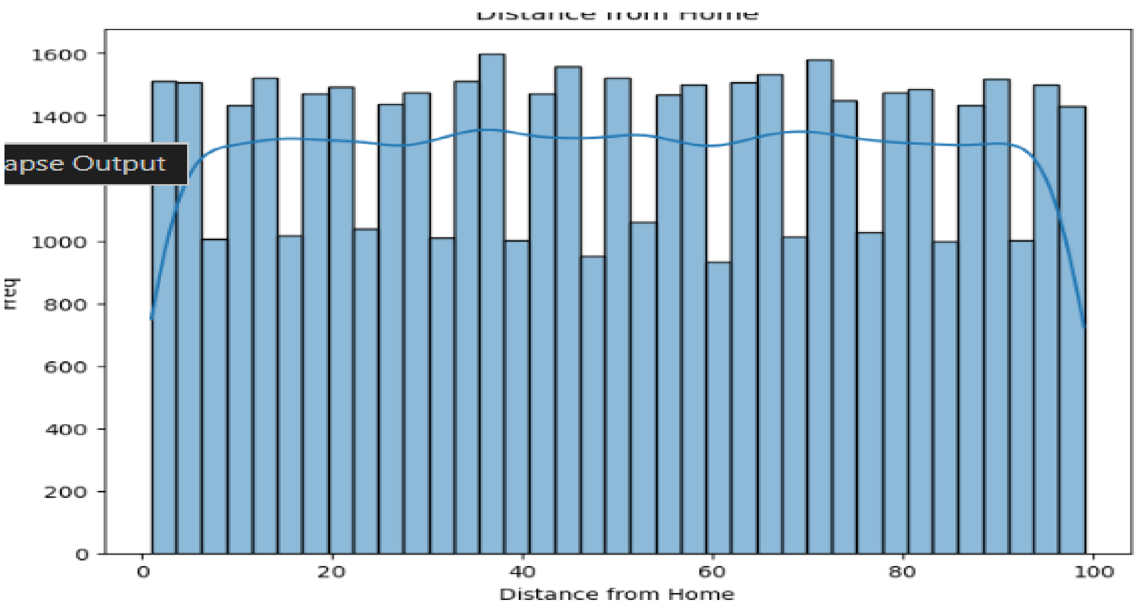




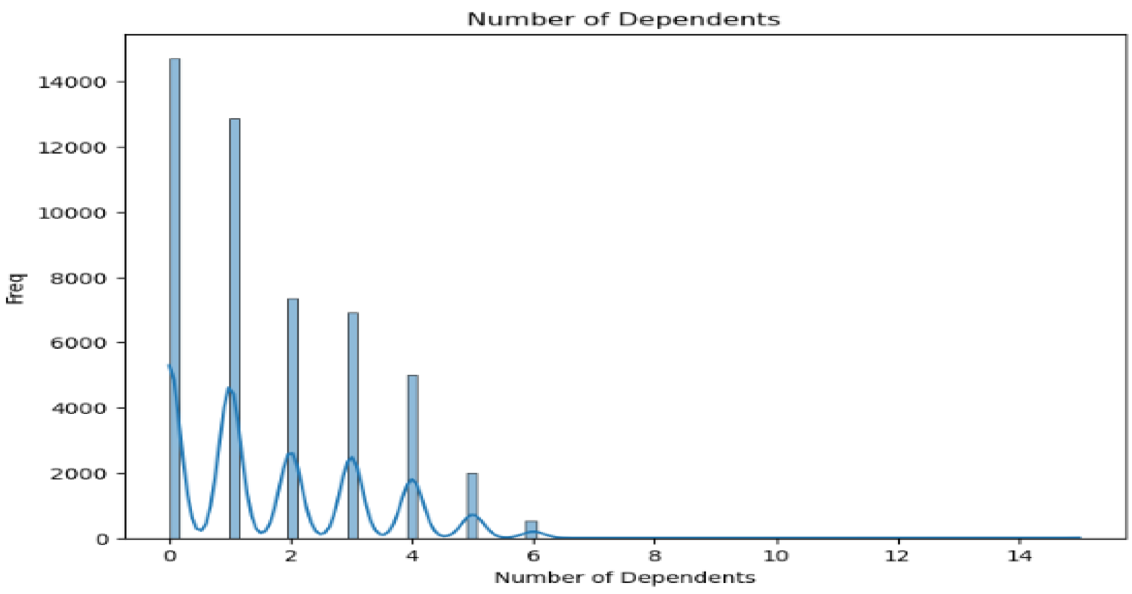

Monthly Income
Freq
1600
1400
1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000
Monthly Income


Monthly Income
Freq
1600
1400
1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
50000
Monthly Income

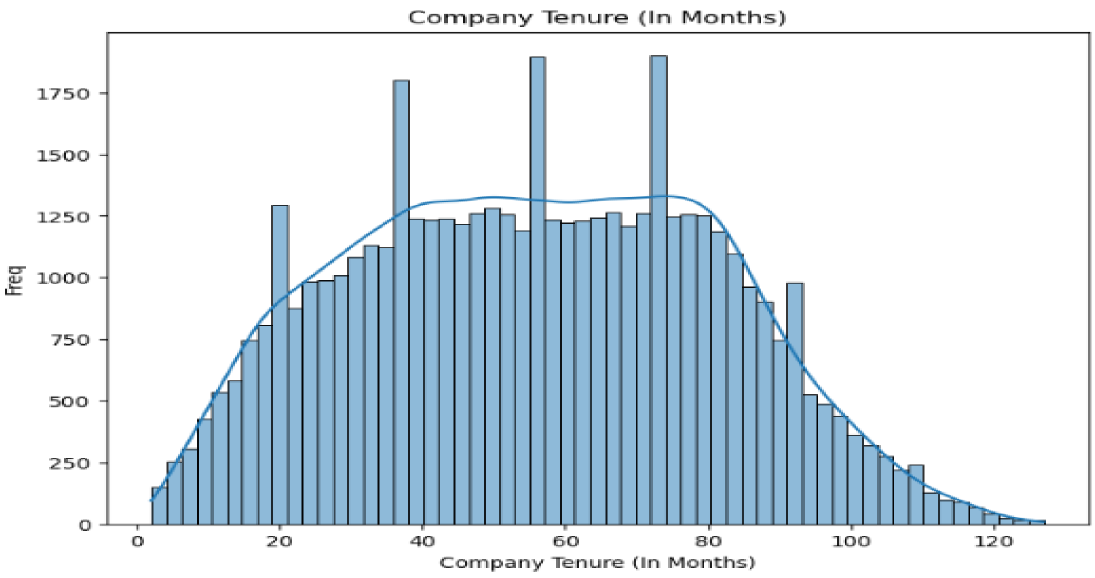

Number of Promotions
Freq
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Number of Promotions



Distance from Home
apse Output
1600
1400
1000
800
600
400
200
0
Freq
0
20
40
60
80
100
Distance from Home



Number of Dependents
14000
12000
10000
8000
6000
4000
2000
0
Freq
0
2
4
6
8
10
12
14
Number of Dependents

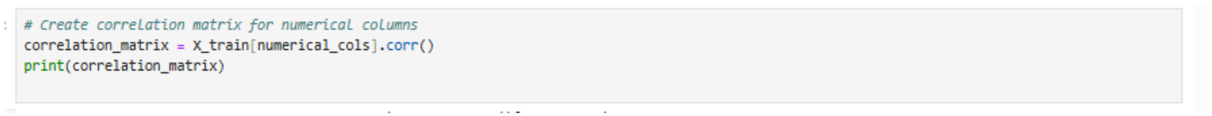


Company Tenure (In Months)
1750
1500
1250
1000
750
500
250
0
Freq
0
20
40
60
80
100
120
Company Tenure (In Months)

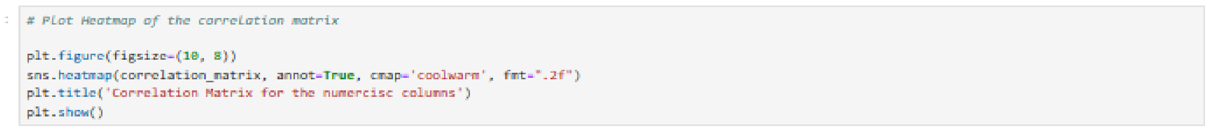
## 4.2 Correlation Analysis

A correlation heatmap was generated for numerical variables.
Key observations included:

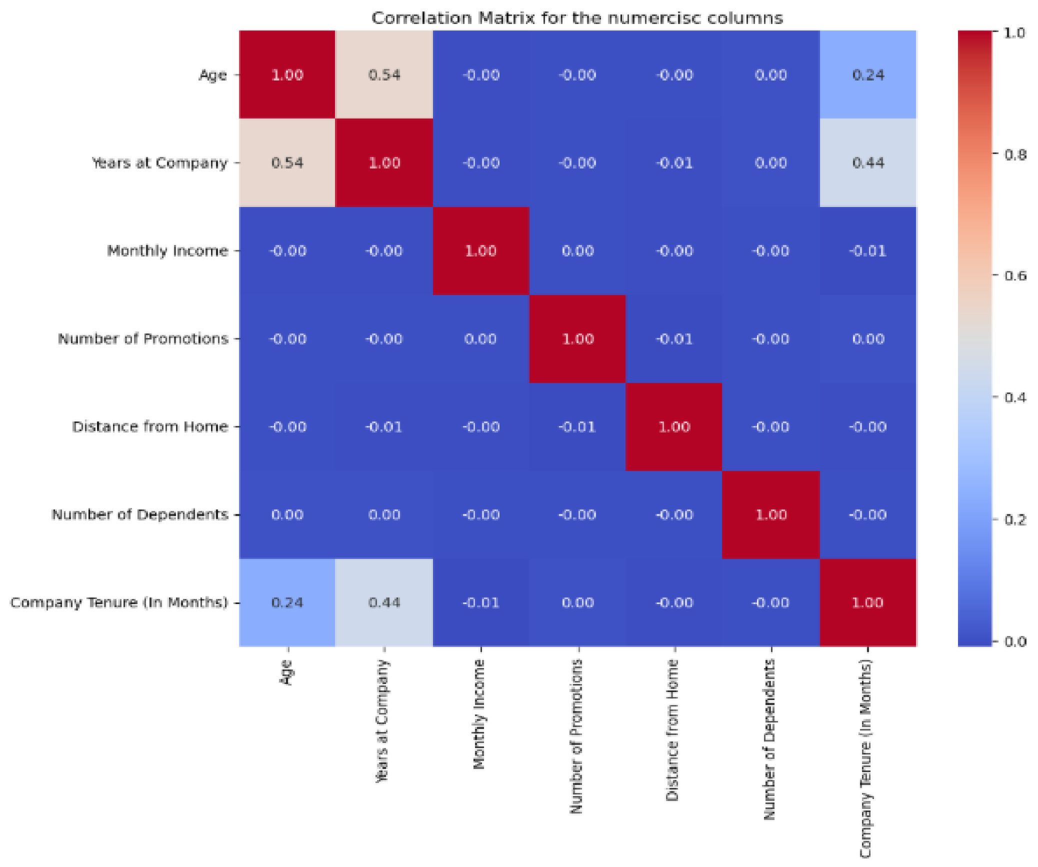
- Strong correlations between some job-related metrics and retention.
- Weak correlations between unrelated numerical features, indicating low redundancy.



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



```python
# Plot Heatmap of the correlation matrix

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix for the numercisc columns')
plt.show()
```



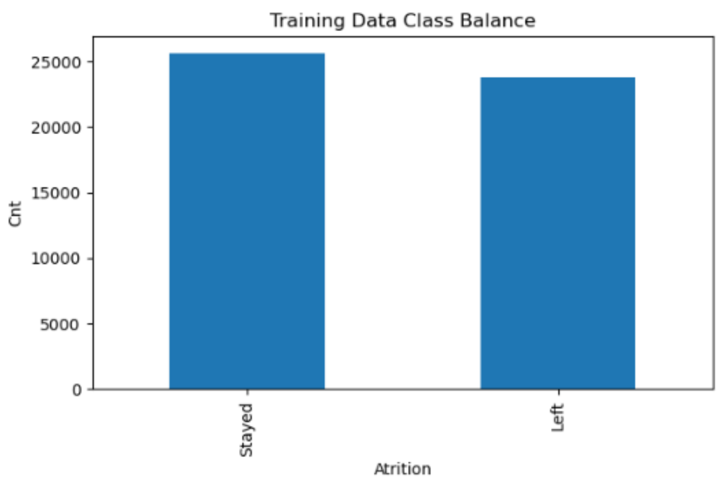
| | Age | Years at Company | Monthly Income | Number of Promotions | Distance from Home | Number of Dependents | Company Tenure (In Months) |
|---|---|---|---|---|---|---|---|
| Age | 1.00 | 0.54 | -0.00 | -0.00 | -0.00 | 0.00 | 0.24 |
| Years at Company | 0.54 | 1.00 | -0.00 | -0.00 | -0.01 | 0.00 |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

## 4.3 Class Imbalance Check

Visual analysis confirmed moderate imbalance in the target variable, validating the decision to use stratified sampling.

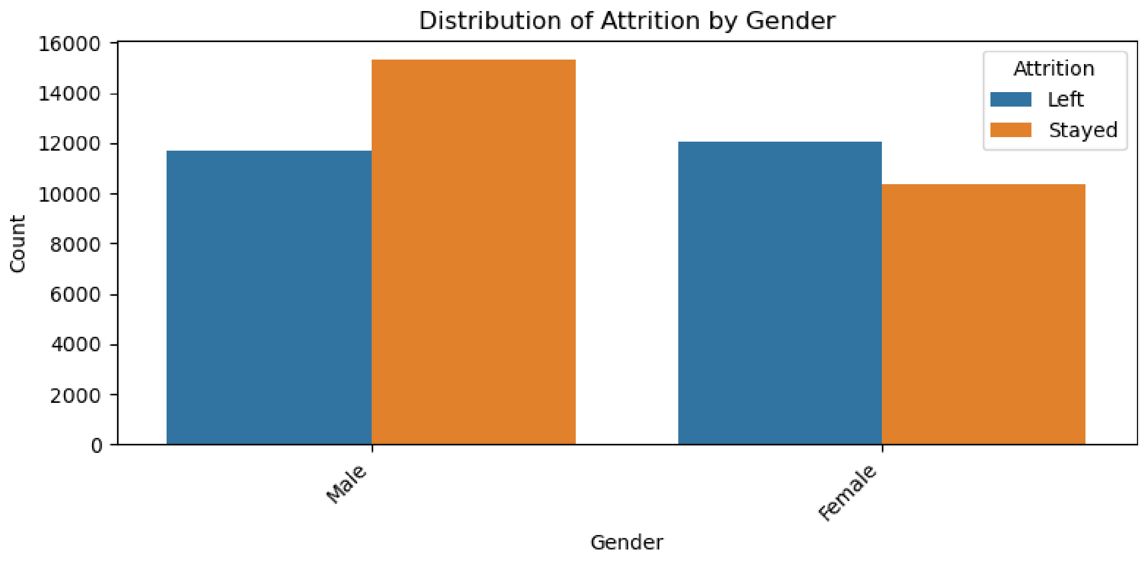
These insights guided the feature engineering and modelling decisions that followed.

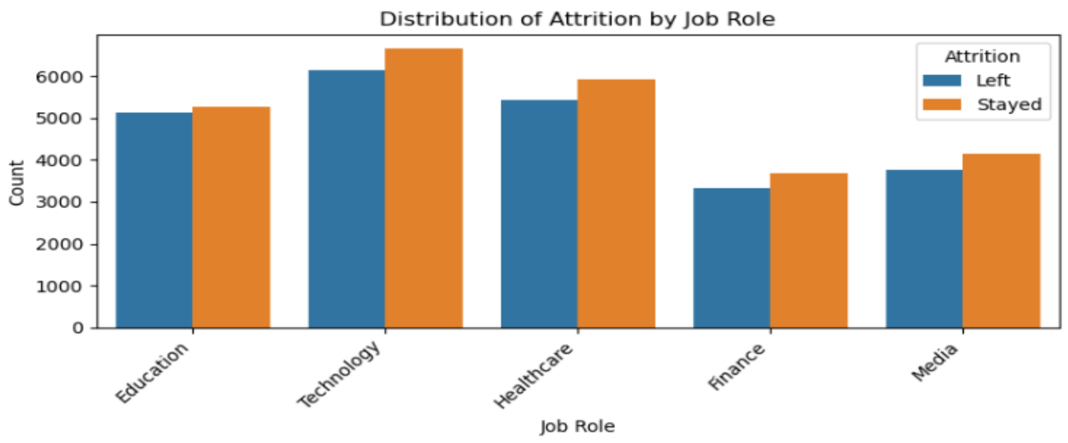
```
# Plot a bar chart to check class balance

plt.figure(figsize=(7, 4))
y_train.value_counts().plot(kind='bar')
plt.title('Training Data Class Balance')
plt.xlabel('Atrition')
plt.ylabel('Cnt')
plt.show()
```

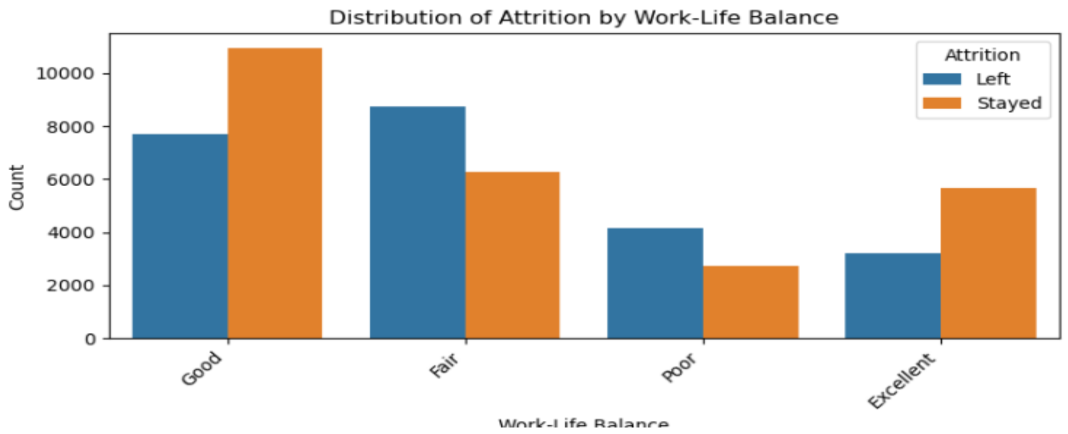


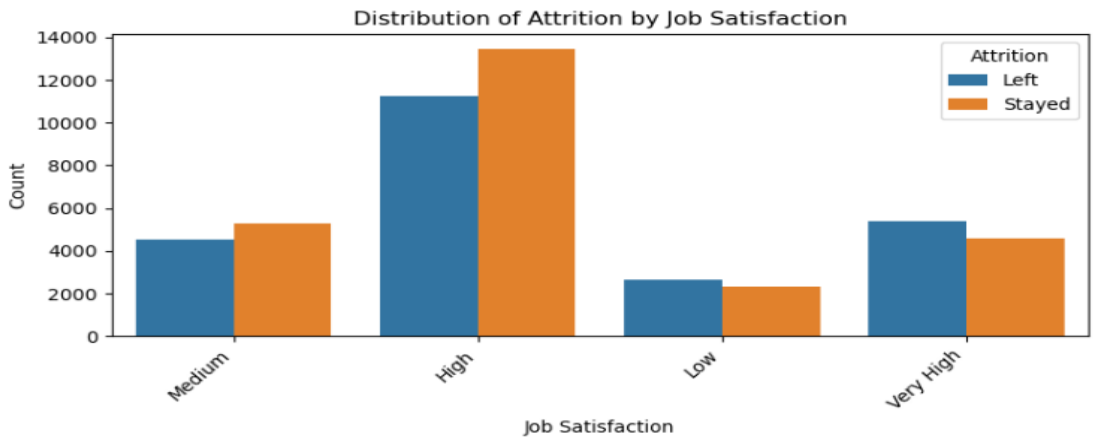
4.4 **Perform bivariate analysis**

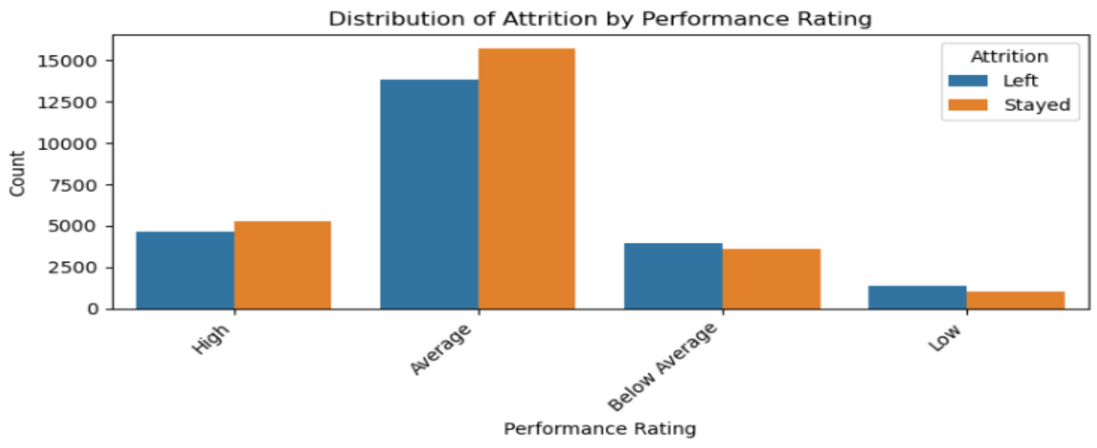
```
# Plot distribution for each categorical column with target variable
categorical_cols = X_train.select_dtypes(include='object').columns

for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=X_train.join(y_train), x=col, hue=y_train.name)
    plt.title(f'Distribution of Attrition by {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

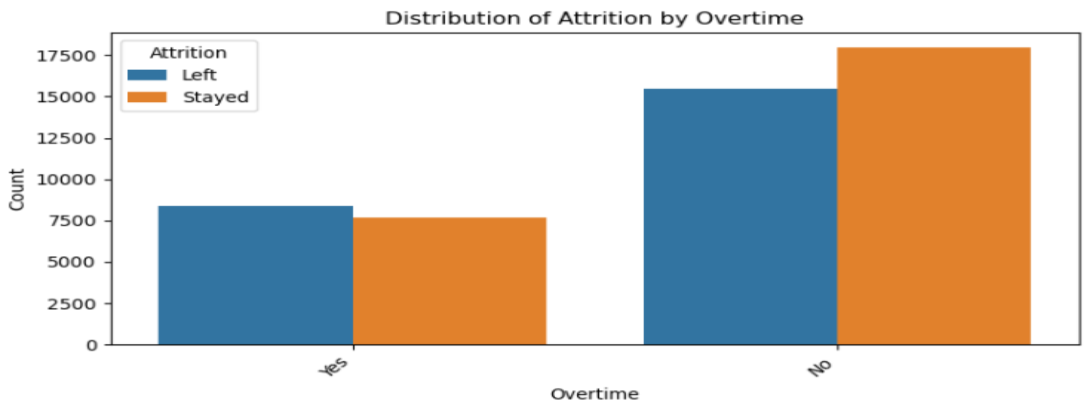


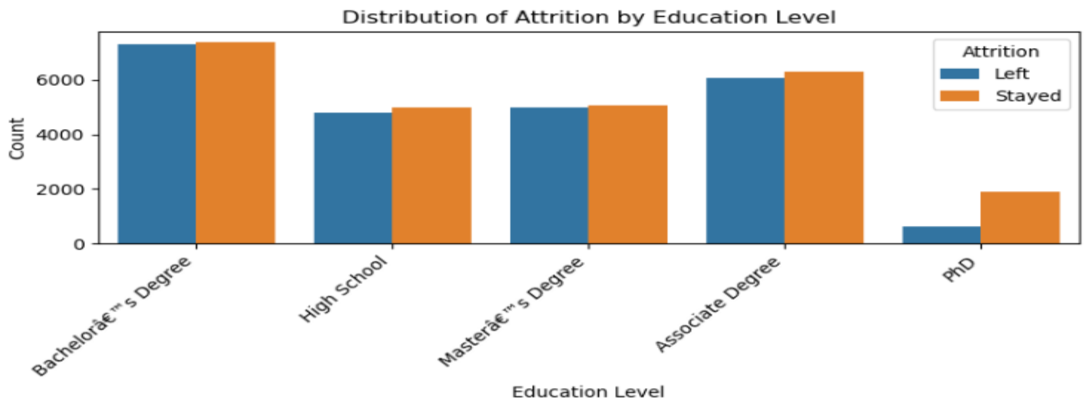

Distribution of Attrition by Job Role
Attrition
Left
Stayed
Count
0
1000
2000
3000
4000
5000
6000
Education
Technology
Healthcare
Finance
Media
Job Role

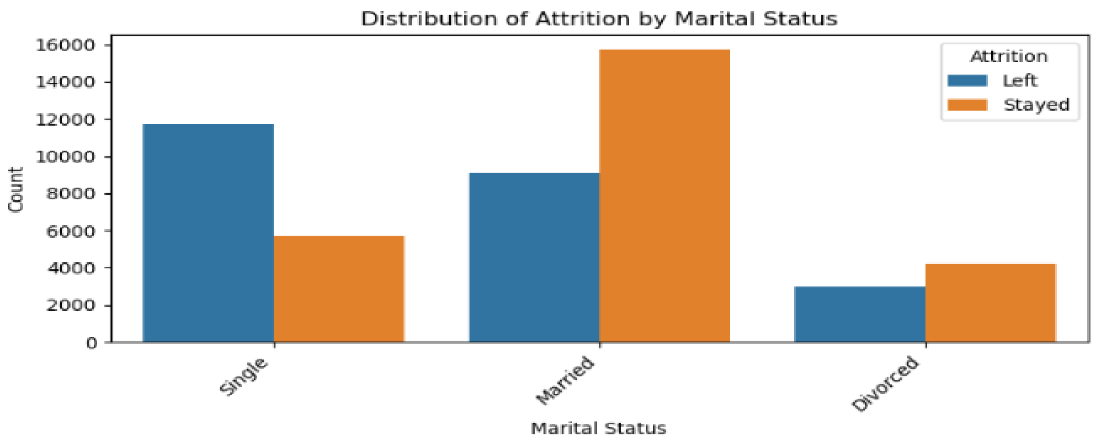

Distribution of Attrition by Work-Life Balance
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
Good
Fair
Poor
Excellent
Work-Life Balance

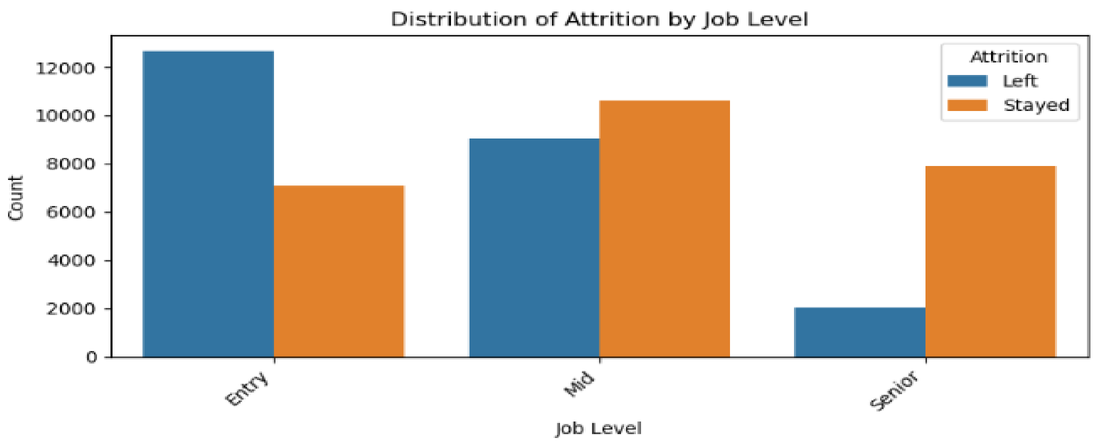

Distribution of Attrition by Job Satisfaction
Attrition
Left
Stayed
Count
0
2000
4000
6000
8000
10000
12000
14000
Medium
High
Low
Very High
Job Satisfaction

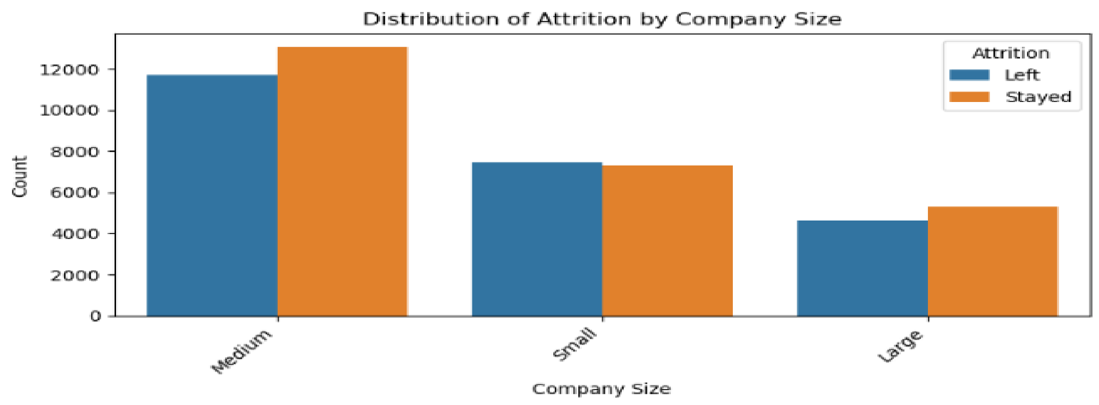

Distribution of Attrition by Performance Rating
Attrition
Left
Stayed
Count
0
2500
5000
7500
10000
12500
15000
High
Average
Below Average
Low
Performance Rating

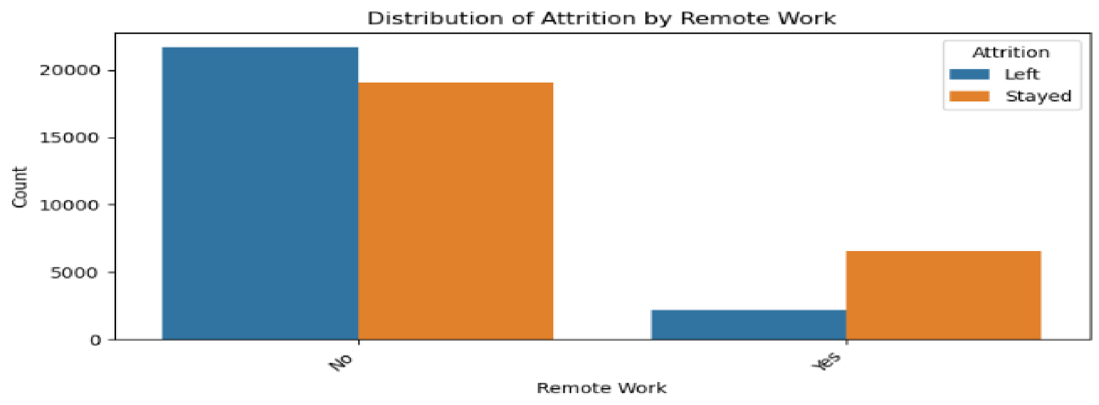

Distribution of Attrition by Overtime
Attrition
Left
Stayed
Count
17500
15000
12500
10000
7500
5000
2500
0
Yes
No
Overtime

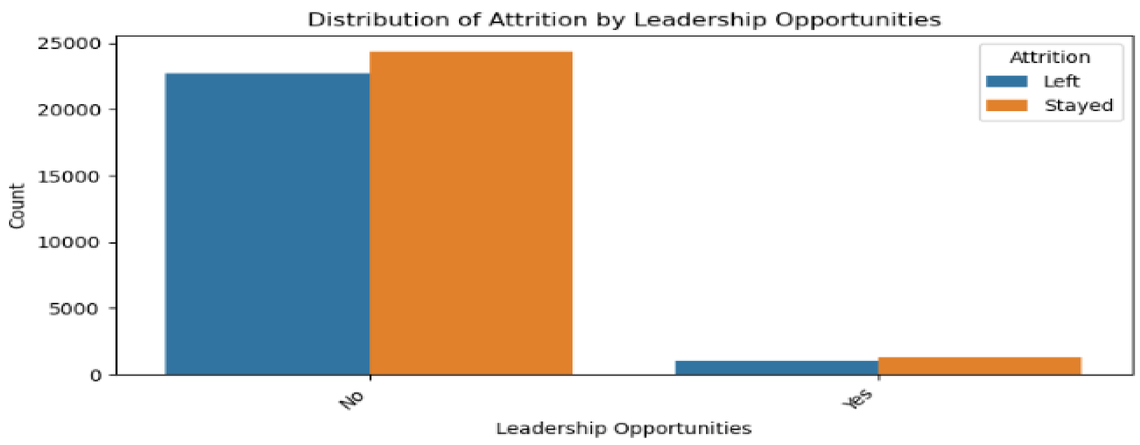

Distribution of Attrition by Education Level
Attrition
Left
Stayed
Count
6000
4000
2000
0
Bachelorâ€™s Degree
High School
Masterâ€™s Degree
Associate Degree
PhD
Education Level

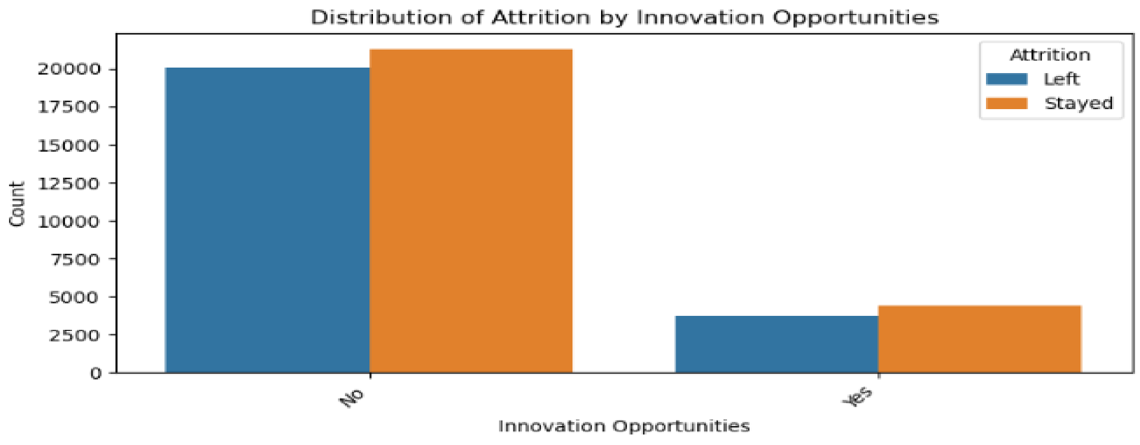

Distribution of Attrition by Marital Status
Attrition
Left
Stayed
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
Single
Married
Divorced
Marital Status

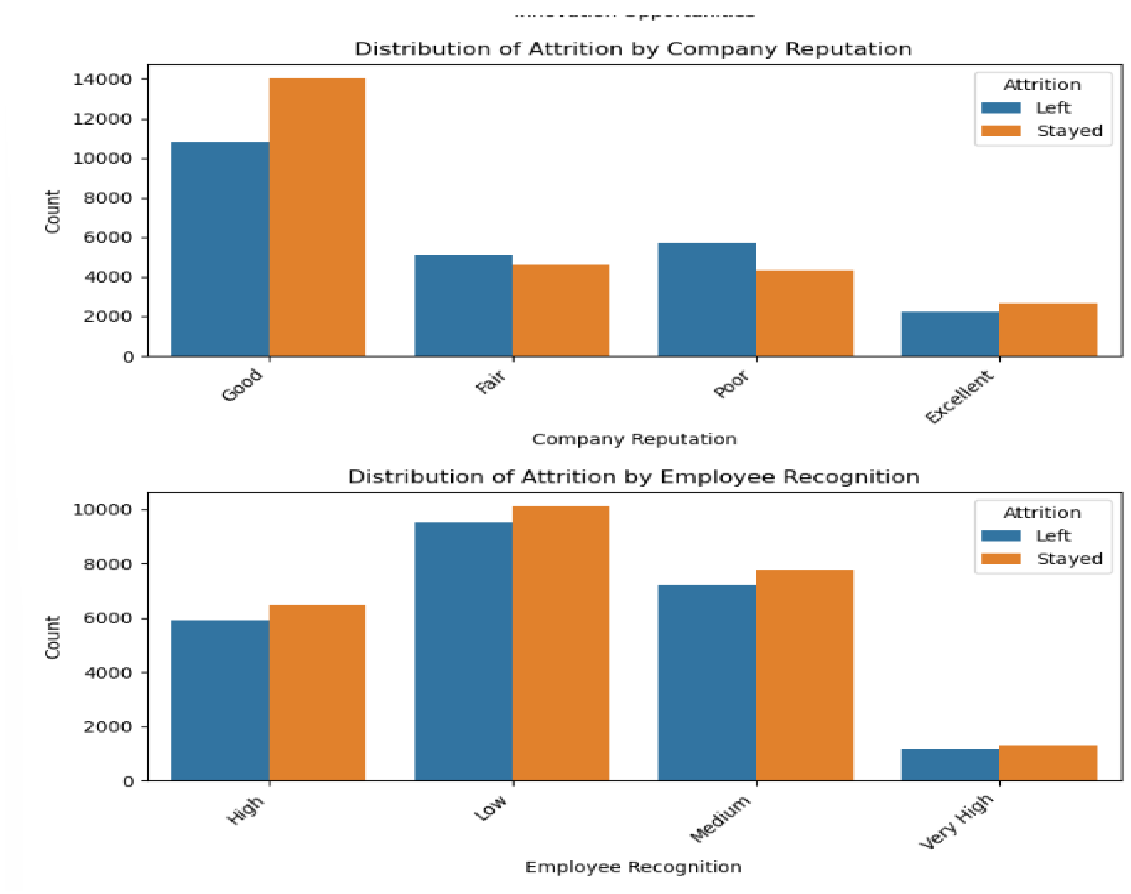

Distribution of Attrition by Job Level
Attrition
Left
Stayed
Count
12000
10000
8000
6000
4000
2000
0
Entry
Mid
Senior
Job Level


Distribution of Attrition by Company Size
Count
12000
10000
8000
6000
4000
2000
0
Medium
Small
Large
Company Size
Attrition
Left
Stayed


Distribution of Attrition by Remote Work
Count
20000
15000
10000
5000
0
No
Yes
Remote Work
Attrition
Left
Stayed


Distribution of Attrition by Leadership Opportunities
Count
25000
20000
15000
10000
5000
0
No
Yes
Leadership Opportunities
Attrition
Left
Stayed


Distribution of Attrition by Innovation Opportunities
Count
20000
17500
15000
12500
10000
7500
5000
2500
0
No
Yes
Innovation Opportunities
Attrition
Left
Stayed

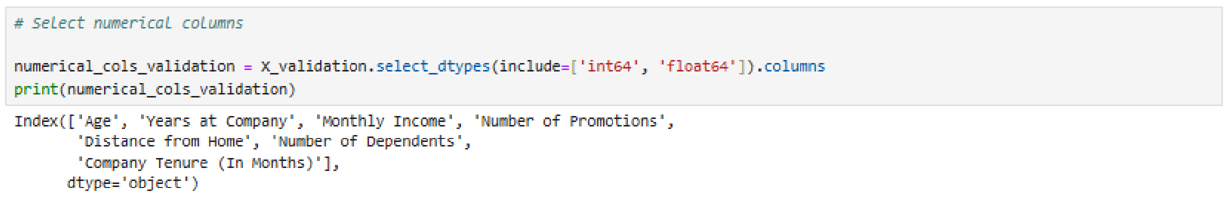


## 5. EDA on Validation Data

A brief inspection of the validation dataset confirmed that:

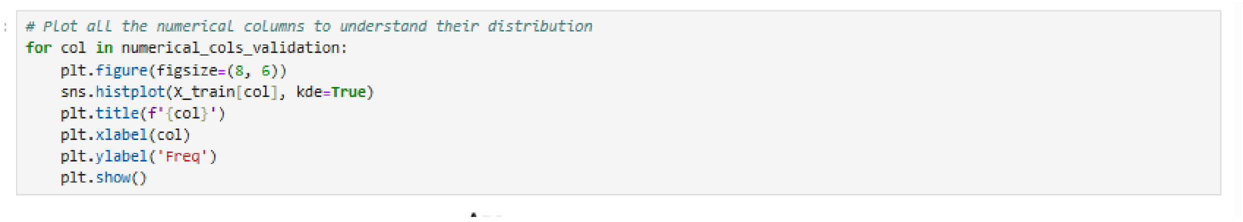
- Distributions of key features closely resembled those in the training set.
- No significant shifts or anomalies were present.

This reassured that the validation results would reliably reflect real-world model performance.



```
# Select numerical columns

numerical_cols_validation = X_validation.select_dtypes(include=['int64', 'float64']).columns
print(numerical_cols_validation)
```

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



```
# Plot all the numerical columns to understand their distribution
for col in numerical_cols_validation:
    plt.figure(figsize=(8, 6))
    sns.histplot(X_train[col], kde=True)
    plt.title(f'{col}')
    plt.xlabel(col)
    plt.ylabel('Freq')
    plt.show()
```

```python
# Create correlation matrix for numerical columns
correlation_matrix_validation = X_validation[numerical_cols].corr()
print(correlation_matrix_validation)

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

```
                                  Age  Years at Company  Monthly Income  \
```

```python
# Plot a bar chart to check class balance

plt.figure(figsize=(7, 4))
y_validation.value_counts().plot(kind='bar')
plt.title('Training Data Class Balance')
plt.xlabel('Atrition')
plt.ylabel('Cnt')
plt.show()
```

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

for col in categorical_cols_validation:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=X_validation.join(y_validation), x=col, hue=y_train.name)
    plt.title(f'Distribution of Attrition by {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

# 6. Feature Engineering

Feature engineering improved model interpretability and predictive power:

## 6.1 Dummy variable creation & Encoding Categorical Variables

- One-hot encoding was applied to convert categorical features into model-friendly numeric vectors.

```python
# Check the categorical columns
categorical_cols = X_train.select_dtypes(include='object').columns
print(categorical_cols)
```

```python
# Create dummy variables using the 'get_dummies' for independent columns
X_train_categorical = X_train.select_dtypes(include='object')
X_train_numerical = X_train.select_dtypes(include=['int64', 'float64'])
#X_train_dummies = pd.get_dummies(X_train_categorical, drop_first=True)
X_train_dummies = pd.get_dummies(X_train_categorical)

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

```python
X_train.info()
```

```python
X_train.columns
```

```python
# Drop the original categorical columns and check the DataFrame
#X_train.drop(categorical_cols, axis=1, inplace=True)
X_train.columns

# Categorical columns are already deleted during "Dummies" process
# Create dummy variables using the 'get_dummies' for independent columns
X_validation_categorical = X_validation.select_dtypes(include='object')
X_validation_numerical = X_validation.select_dtypes(include=['int64', 'float64'])
X_validation_dummies = pd.get_dummies(X_validation_categorical, drop_first=True)

# Add the results to the master DataFrame
X_validation = pd.concat([X_validation_numerical, X_validation_dummies], axis=1)
```

Now, drop the original categorical columns and check the DataFrame

```python
# Drop the original categorical columns and check the DataFrame
#X_validation.drop(X_validation_categorical, axis=1, inplace=True)
X_validation.columns

# Categorical columns are already deleted during "Dummies" process
```

```python
# Convert y_train and y_validation to DataFrame to create dummy variables
y_train = pd.DataFrame(y_train)
y_validation = pd.DataFrame(y_validation)
```

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

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

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

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

6.1.7 Drop redundant columns [1 Mark]

```python
# Drop redundant columns from both train and validation
X_train = X_train.drop(X_train.select_dtypes(include='object').columns, axis=1)

X_validation = X_validation.drop(X_validation.select_dtypes(include='object').columns, axis=1)
```

```python
X_train.columns
```

```python
X_validation.columns
```

## 6.2 Scaling Numerical Features

- Continuous variables were normalized or standardized where necessary to maintain consistent scales.

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

```python
# Scale the numeric features present in the training set

scaler = StandardScaler()

X_train[X_train.select_dtypes(include=['int64', 'float64']).columns] = scaler.fit_transform(X_train.select_dtypes(include=['int64', 'float64']))

# Scale the numerical features present in the validation set

X_validation[X_validation.select_dtypes(include=['int64', 'float64']).columns] = scaler.transform(X_validation.select_dtypes(include=['int64', 'float64']
```

# 7. Model Building

The logistic regression model was trained using the processed training data.

## 7.1 Feature selection

- 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.

```
# Import 'LogisticRegression' and create a LogisticRegression object
from sklearn.linear_model import LogisticRegression\

lr_1 = 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=lr_1, n_features_to_select=15)

rfe = rfe.fit(X_train, y_train_dummies.values.ravel())
```

```
# Display the features selected by RFE
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

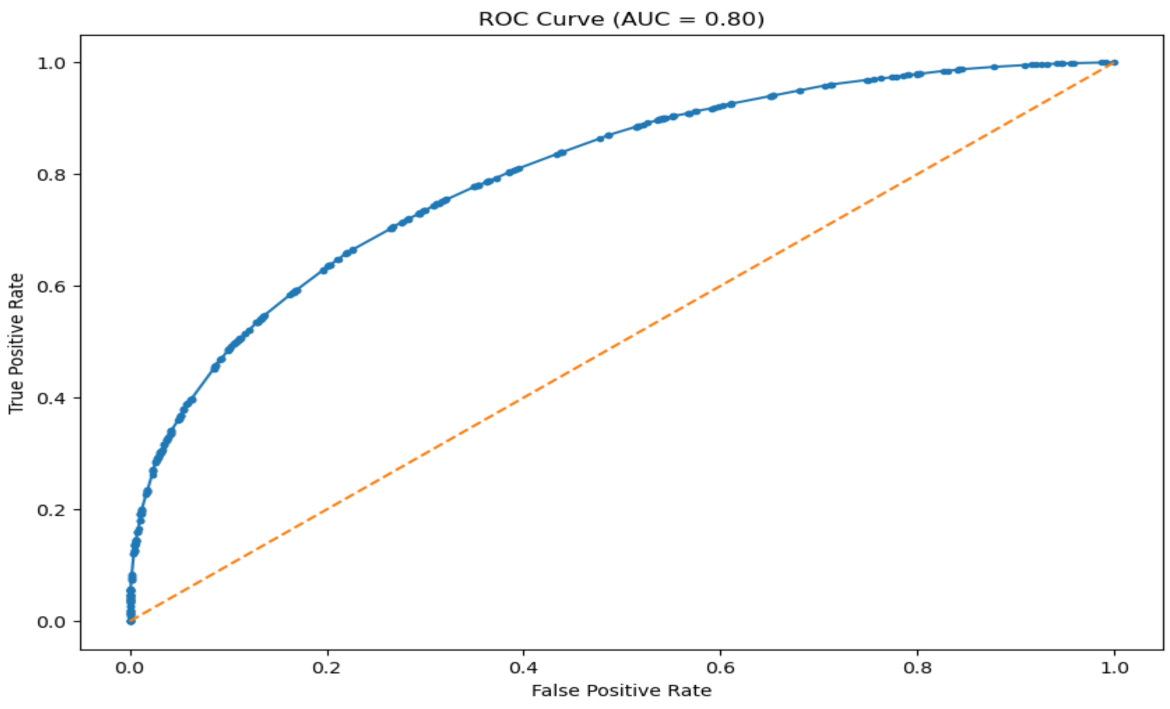
## 7.2 Building Logistic Regression Model

Now that you have selected the variables through RFE, use these features to build a logistic regression model with stats models. 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.

## 7.3 Building Logistic Regression Model

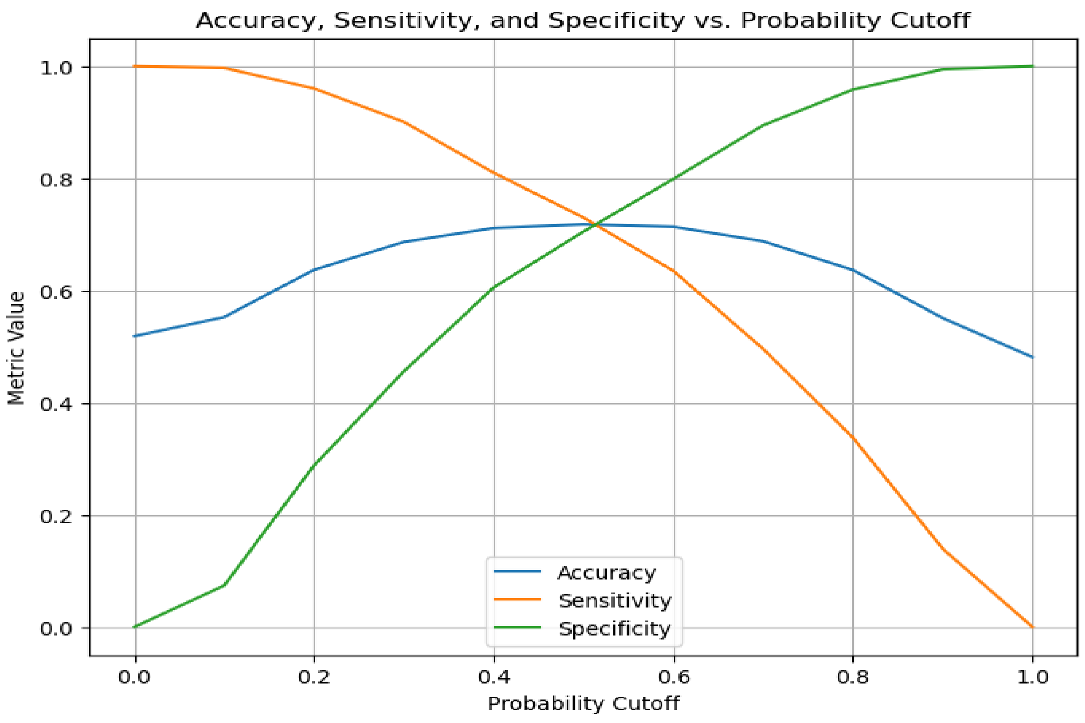
```
# Define ROC function
from sklearn.metrics import roc_curve, roc_auc_score

def plot_roc_curve(actuals, predicted_probs):
  fpr, tpr, thresholds = roc_curve(actuals, predicted_probs)
  auc = roc_auc_score(actuals, predicted_probs)
  plt.figure(figsize=(10, 7))
  plt.plot(fpr, tpr, marker='.')
  plt.plot([0, 1], [0, 1], linestyle='--')
  plt.title(f'ROC Curve (AUC = {auc:.2f})')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.show()
```

```
# Call the ROC function
plot_roc_curve(y_train_pred['Actual'], y_train_pred['Predicted_Prob'])
```

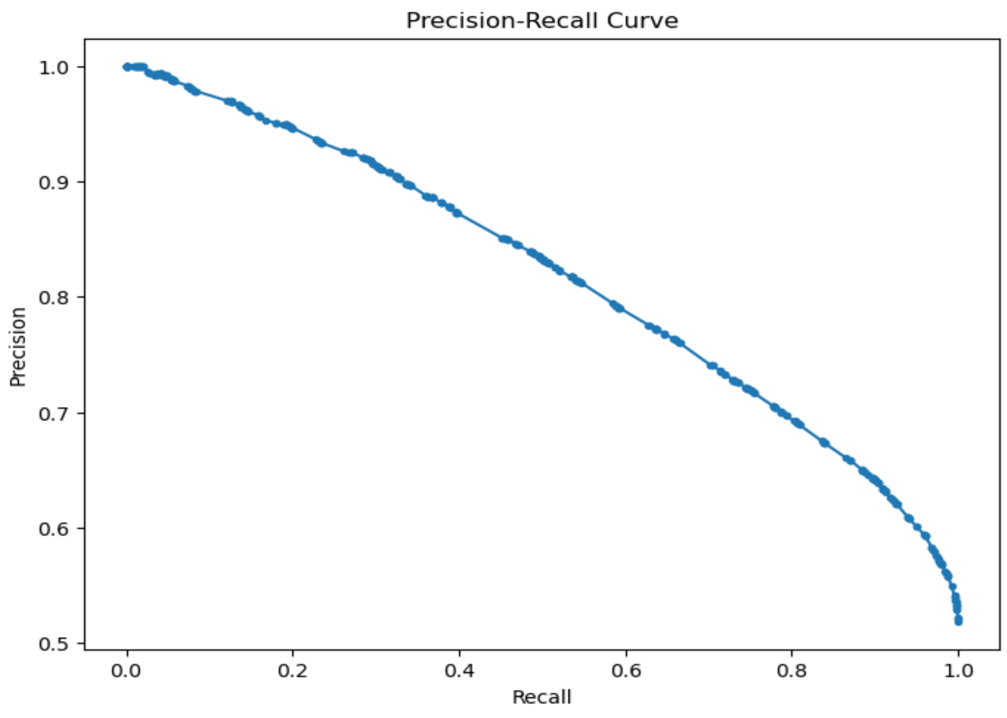


```
# Create a DataFrame to see the values of accuracy, sensitivity, and specificity at different values of probability cu
cutoff_df = pd.DataFrame(columns=['Cutoff', 'Accuracy', 'Sensitivity', 'Specificity'])

for cutoff in cutoff_values:
  predicted_col = f'Predicted_{cutoff:.1f}'
  actual = y_train_pred['Actual']
  predicted = y_train_pred[predicted_col]

  accuracy = metrics.accuracy_score(actual, predicted)
  confusion_matrix = metrics.confusion_matrix(actual, predicted)
  tn, fp, fn, tp = confusion_matrix.ravel()
  sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
  specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

  cutoff_df = pd.concat([cutoff_df, pd.DataFrame({'Cutoff': [cutoff], 'Accuracy': [accuracy], 'Sensitivity': [sensitiv

print(cutoff_df)
```

```
# Plot accuracy, sensitivity, and specificity at different values of probability cutoffs
plt.figure(figsize=(8, 6))

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.title('Accuracy, Sensitivity, and Specificity vs. Probability Cutoff')

plt.xlabel('Probability Cutoff')

plt.ylabel('Metric Value')

plt.legend()

plt.grid(True)

plt.show()
```



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

precision, recall, thresholds = precision_recall_curve(y_train_pred['Actual'], y_train_pred['Predicted_Prob'])

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```

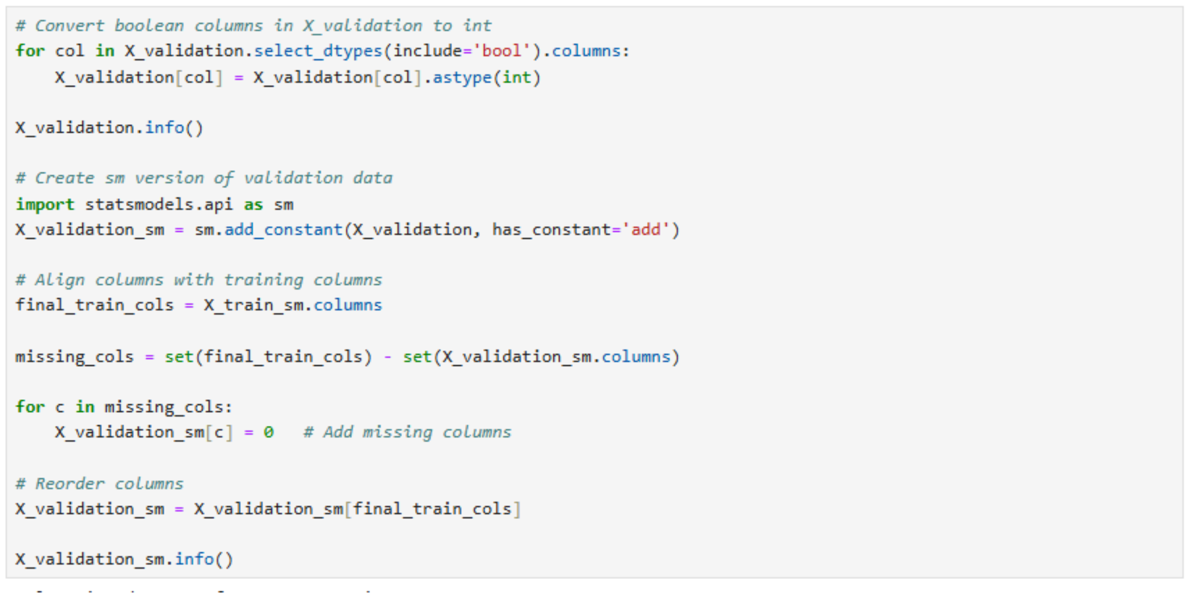


# 8. Prediction and Model Evaluation

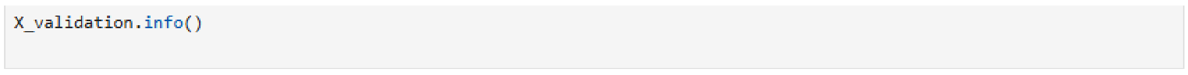
Model performance was assessed using the validation dataset:

### 8.1 Prediction and Model Evaluation

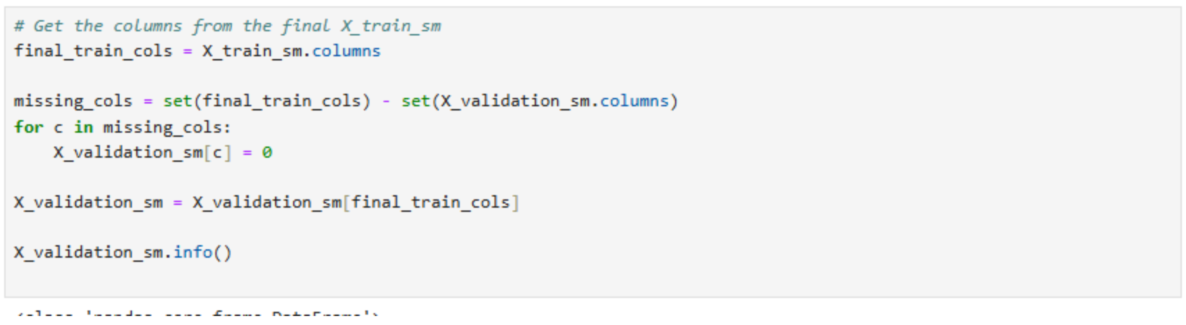
Use the model from the previous step to make predictions on the validation set with the optimal cutoff. Then evaluate the model's performance using metrics such as accuracy, sensitivity, specificity, precision, and recall.



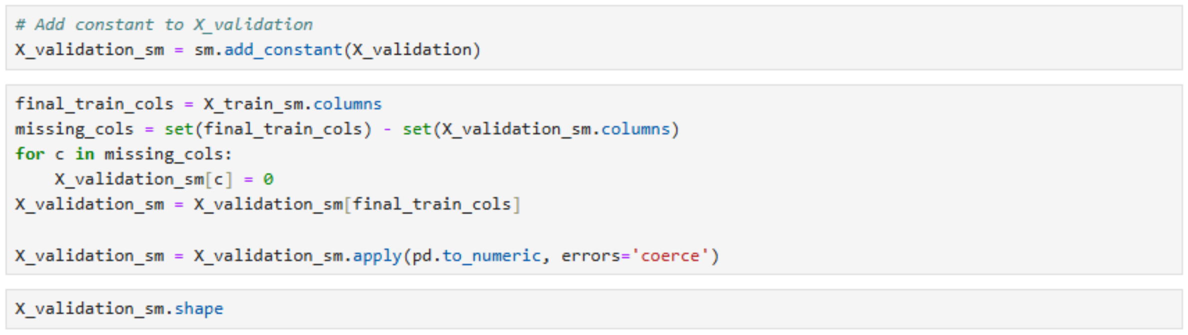
```python
# Convert boolean columns in X_validation to int
for col in X_validation.select_dtypes(include='bool').columns:
    X_validation[col] = X_validation[col].astype(int)

X_validation.info()

# Create sm version of validation data
import statsmodels.api as sm
X_validation_sm = sm.add_constant(X_validation, has_constant='add')

# Align columns with training columns
final_train_cols = X_train_sm.columns

missing_cols = set(final_train_cols) - set(X_validation_sm.columns)

for c in missing_cols:
    X_validation_sm[c] = 0   # Add missing columns

# Reorder columns
X_validation_sm = X_validation_sm[final_train_cols]

X_validation_sm.info()
```



```python
X_validation.info()
```



```python
# Get the columns from the final X_train_sm
final_train_cols = X_train_sm.columns

missing_cols = set(final_train_cols) - set(X_validation_sm.columns)
for c in missing_cols:
    X_validation_sm[c] = 0

X_validation_sm = X_validation_sm[final_train_cols]

X_validation_sm.info()
```



```python
# Add constant to X_validation
X_validation_sm = sm.add_constant(X_validation)
```

```python
final_train_cols = X_train_sm.columns
missing_cols = set(final_train_cols) - set(X_validation_sm.columns)
for c in missing_cols:
    X_validation_sm[c] = 0
X_validation_sm = X_validation_sm[final_train_cols]

X_validation_sm = X_validation_sm.apply(pd.to_numeric, errors='coerce')
```

```python
X_validation_sm.shape
```

```
# Make predictions on the validation set and store it in the variable 'y_validation_pred'
y_validation_pred_prob = logm2_fit.predict(X_validation_sm)

# View predictions
y_validation_pred_prob.head()
```

```
70633    0.629146
10909    0.874179
20347    0.716807
38015    0.370734
11436    0.329023
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_validation_pred_prob, columns=['Predicted_Prob'])

# Convert 'y_validation' to DataFrame 'actual'
# Convert 'y_validation_dummies' to DataFrame 'actual'
actual = pd.DataFrame(y_validation_dummies).reset_index(drop=True)

actual.head()

# Remove index from both DataFrames 'actual' and 'predicted_probability' to append them side by side
predicted_probability = predicted_probability.reset_index(drop=True)

y_validation_pred = pd.concat([actual, predicted_probability], axis=1)

y_validation_pred.head()
```

## 8.2 Calculate accuracy of the model

```
# Calculate the overall accuracy
accuracy_validation = metrics.accuracy_score(y_validation_pred['Attrition_Stayed'].astype(int), y_validation_pred['fir

print(accuracy_validation)
```

```
0.6926053513283942
```

## 8.3 Create confusion matrix and create variables for true positive, true negative, false positive and false negative

```
# Create confusion matrix
confusion_matrix_validation = metrics.confusion_matrix(y_validation_pred['Attrition_Stayed'].astype(int), y_validatior
print(confusion_matrix_validation)
```

```
[[5778 4250]
 [2264 8899]]
```

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

## 8.4 Calculate sensitivity and specificity

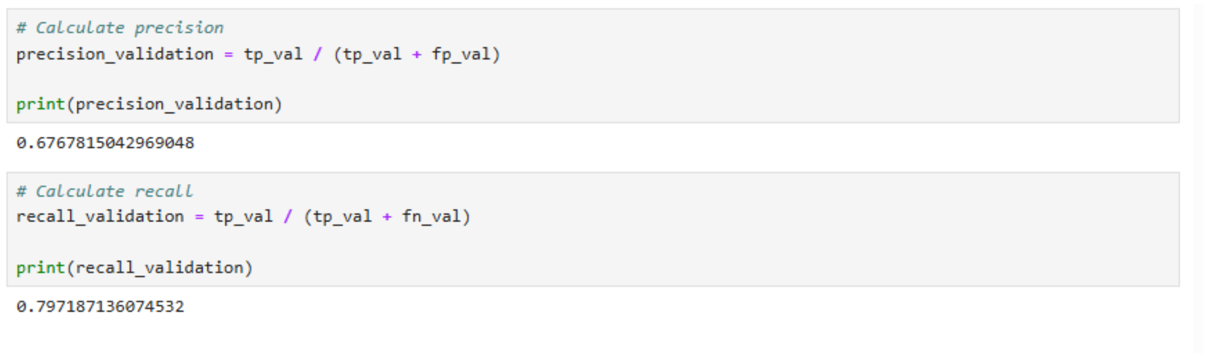
```
# Calculate sensitivity
sensitivity_validation = tp_val / (tp_val + fn_val)

print(sensitivity_validation)
```

```
0.797187136074532
```

```
# Calculate specificity
specificity_validation = tn_val / (tn_val + fp_val)

print(specificity_validation)
```

```
0.5761866773035501
```

### 8.5 Calculate precision and recall



```
# Calculate precision
precision_validation = tp_val / (tp_val + fp_val)

print(precision_validation)
```

```
0.6767815042969048
```

```
# Calculate recall
recall_validation = tp_val / (tp_val + fn_val)

print(recall_validation)
```

```
0.797187136074532
```

## Conclusion

The logistic regression model developed through this analysis provided a structured, data-driven approach to understanding employee retention. By following a rigorous workflow—including data understanding, cleaning, EDA, feature engineering, and model evaluation—the study uncovered significant relationships between employee characteristics and retention outcomes.

The model yielded actionable insights, such as the impact of tenure, satisfaction levels, performance scores, and other job-related attributes on retention. These insights can guide HR teams in identifying at-risk employees, improving workplace policies, and designing targeted interventions to enhance retention.

Overall, the project demonstrates the effectiveness of logistic regression as an interpretable and accurate method for predicting employee retention and supporting strategic decision-making within organizations.