In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df= pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
```

In [3]:
```
pd.set_option('display.max_columns', None)
```

In [4]:
```
df.head()
```

Out[4]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life
4	27	No	Travel_Rarely	591	Research & Development	2	1	

In [5]:
```
df["Attrition"].value_counts()
```

Out[5]:
```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

In [309...
```
df["EducationField"].value_counts()
```

Out[309]:
```
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64
```

In [307...
```
df["Department"].value_counts()
```

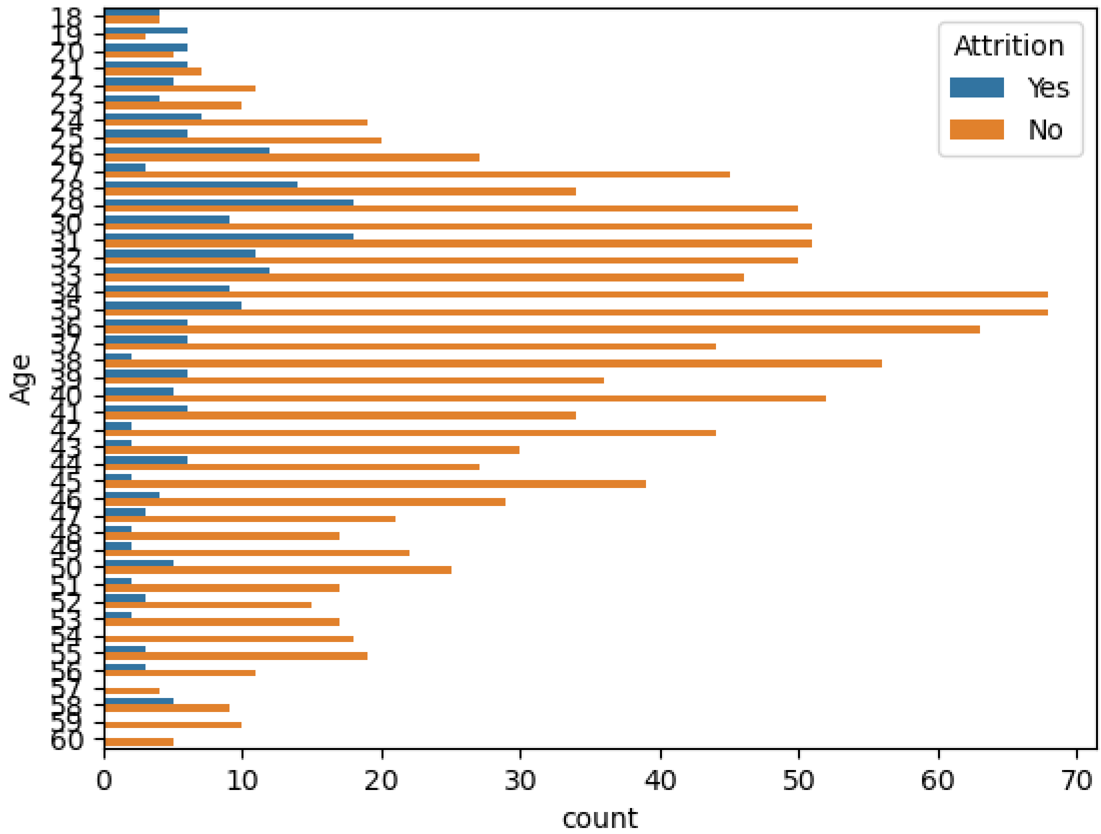
Out[307]:
```
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
```

In [221...
```
df["JobRole"].value_counts()
```

Out[221]:
```
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64
```

In [222…
```
sns.countplot(data=df,y=df["Age"], hue= df["Attrition"])
```

Out[222]:
```
<Axes: xlabel='count', ylabel='Age'>
```



In [223…
```
correlation = df[['JobInvolvement', 'JobSatisfaction']].corr()

print("Correlation Matrix:")
print(correlation)
```

```
Correlation Matrix:
                 JobInvolvement  JobSatisfaction
JobInvolvement         1.000000        -0.021476
JobSatisfaction       -0.021476         1.000000
```

In [224…
```
df.describe()
```

Out[224]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNuml
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000(
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865:
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024:
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000(
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250(
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500(
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750(
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000(

In [225…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [226…
```
df= df.drop(['Over18', 'Department', 'EducationField', 'JobRole'], axis=1)
```

In [227…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 31 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   DistanceFromHome          1470 non-null   int64 
 5   Education                 1470 non-null   int64 
 6   EmployeeCount             1470 non-null   int64 
 7   EmployeeNumber            1470 non-null   int64 
 8   EnvironmentSatisfaction   1470 non-null   int64 
 9   Gender                    1470 non-null   object
 10  HourlyRate                1470 non-null   int64 
 11  JobInvolvement            1470 non-null   int64 
 12  JobLevel                  1470 non-null   int64 
 13  JobSatisfaction           1470 non-null   int64 
 14  MaritalStatus             1470 non-null   object
 15  MonthlyIncome             1470 non-null   int64 
 16  MonthlyRate               1470 non-null   int64 
 17  NumCompaniesWorked        1470 non-null   int64 
 18  OverTime                  1470 non-null   object
 19  PercentSalaryHike         1470 non-null   int64 
 20  PerformanceRating         1470 non-null   int64 
 21  RelationshipSatisfaction  1470 non-null   int64 
 22  StandardHours             1470 non-null   int64 
 23  StockOptionLevel          1470 non-null   int64 
 24  TotalWorkingYears         1470 non-null   int64 
 25  TrainingTimesLastYear     1470 non-null   int64 
 26  WorkLifeBalance           1470 non-null   int64 
 27  YearsAtCompany            1470 non-null   int64 
 28  YearsInCurrentRole        1470 non-null   int64 
 29  YearsSinceLastPromotion   1470 non-null   int64 
 30  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(5)
memory usage: 356.1+ KB
```

Creating a dummy variable for some of the categorical variables and dropping the first one.

dummy1 = pd.get_dummies(df[['Attrition','BusinessTravel', 'Gender', 'MaritalStatus', 'OverTime']], drop_first=True) dummy1.head()

df_dummy = pd.concat([df, dummy1], axis=1) df_dummy = df_dummy.drop(['BusinessTravel', 'Gender', 'MaritalStatus', 'OverTime'], axis =1) df_dummy.head()

Balancing

In [228…

```
# UNDERSAMPLING
# # Separate the majority and minority classes
majority_class = 'No'
minority_class = 'Yes'

majority_df = df[df['Attrition'] == majority_class]
minority_df = df[df['Attrition'] == minority_class]

# # Randomly sample rows from the majority class to make it balanced
# # You can adjust the sample size based on your preference
sample_size = len(minority_df)

# # Randomly select rows from the majority class
majority_sampled = majority_df.sample(n=sample_size, random_state=42)

# # Concatenate the minority class DataFrame with the sampled majority class DataFr
df = pd.concat([majority_sampled, minority_df])

# # Shuffle the DataFrame to ensure randomness
df= df.sample(frac=1, random_state=42).reset_index(drop=True)
df.shape
```

Out[228]:

```
(474, 31)
```

In [229…

```
# # OVERSAMPLING

# import pandas as pd
# from imblearn.over_sampling import RandomOverSampler
# from sklearn.model_selection import train_test_split

# # Assuming you have a DataFrame named 'df' and the dependent variable is 'Attriti
# # Replace 'Attrition' with the actual column name

# # Separate the features and the target variable
# X = df.drop('Attrition', axis=1)
# y = df['Attrition']

# # Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# # Concatenate the training features and target variable
# train_data = pd.concat([X_train, y_train], axis=1)

# # Count the number of occurrences of each class
# class_counts = train_data['Attrition'].value_counts()

# # Find the class with fewer occurrences
# minority_class = class_counts.idxmin()

# # Get the number of occurrences of the minority class
# minority_class_count = class_counts[minority_class]

# # Set the desired number of samples for each class (e.g., the number of the major
# desired_count = class_counts.max()

# # Calculate the number of samples to add for each class
# samples_to_add = desired_count - minority_class_count

# # Create a RandomOverSampler
# oversampler = RandomOverSampler(sampling_strategy={minority_class: samples_to_add

# # Fit and apply the oversampler to the training data
# X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)
```

```
# # Concatenate the resampled features and target variable
# df = pd.concat([X_train_resampled, y_train_resampled], axis=1)

# # Now 'train_data_resampled' contains the balanced dataset

# # If you want to use the resampled data for training a model, you can use 'X_trai
# df.shape
```

In [230…

```
# from imblearn.over_sampling import SMOTE
# from sklearn.model_selection import train_test_split

# # Assuming you have a DataFrame named 'df' and the dependent variable is 'Attriti
# # Replace 'Attrition' with the actual column name

# # Separate the features and the target variable
# X = df_dummy.drop('Attrition', axis=1)
# y = df_dummy['Attrition']

# # Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

# # Concatenate the training features and target variable
# train_data = pd.concat([X_train, y_train], axis=1)

# # Count the number of occurrences of each class
# class_counts = train_data['Attrition'].value_counts()

# # Find the class with fewer occurrences
# minority_class = class_counts.idxmin()

# # Create a SMOTE object
# smote = SMOTE(sampling_strategy='auto', random_state=42)

# # Fit and apply SMOTE to the training data
# X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# # Concatenate the resampled features and target variable
# df = pd.concat([X_train_resampled, y_train_resampled], axis=1)

# # Now 'train_data_resampled' contains the balanced dataset
# df.shape
```

Model 1

between attrition and WorkLifeBalance

In [231…

```
df.head(2)
```

Out[231]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [232…

```
df1= df[["Attrition","BusinessTravel", "DistanceFromHome", "OverTime", "WorkLifeBal
df1
```

Out[232]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance
0	Yes	Travel_Rarely	10	No	3
1	Yes	Travel_Frequently	8	No	2
2	No	Travel_Frequently	18	No	2
3	Yes	Travel_Rarely	24	Yes	2
4	No	Travel_Rarely	2	No	3
...	...	...	...	...	...
469	No	Travel_Frequently	1	No	3
470	Yes	Travel_Rarely	3	Yes	3
471	Yes	Travel_Rarely	2	No	3
472	Yes	Travel_Rarely	7	Yes	3
473	No	Travel_Rarely	14	No	4

474 rows × 5 columns

In [233…

```
df1["BusinessTravel"].value_counts()
```

Out[233]:

```
Travel_Rarely        326
Travel_Frequently    109
Non-Travel            39
Name: BusinessTravel, dtype: int64
```

In [234…

```
# Create a mapping dictionary
mapping1 = {'Yes': 1, 'No': 0}
mapping2= {'Travel_Frequently':2, 'Travel_Rarely':1, 'Non-Travel':0 }

# Apply the mapping to the specified column
df1['OverTime'] = df1['OverTime'].map(mapping1)
df1['Attrition'] = df1['Attrition'].map(mapping1)
df1['BusinessTravel']= df1["BusinessTravel"].map(mapping2)
# Save the first 5 rows to an Excel file
df1.head(5)
```

```
C:\Users\acer\AppData\Local\Temp\ipykernel_18884\3738258657.py:6: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['OverTime'] = df1['OverTime'].map(mapping1)
C:\Users\acer\AppData\Local\Temp\ipykernel_18884\3738258657.py:7: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['Attrition'] = df1['Attrition'].map(mapping1)
C:\Users\acer\AppData\Local\Temp\ipykernel_18884\3738258657.py:8: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['BusinessTravel']= df1["BusinessTravel"].map(mapping2)
```

Out[234]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance
0	1	1	10	0	3
1	1	2	8	0	2
2	0	2	18	0	2
3	1	1	24	1	2
4	0	1	2	0	3

In [235…

```
X = df1.drop(["Attrition"], axis=1)
y = df1["Attrition"]
```

In [236…

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [237…

```
X_train.columns
```

Out[237]:

```
Index(['BusinessTravel', 'DistanceFromHome', 'OverTime', 'WorkLifeBalance'], dtype
='object')
```

In [239…

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train[['BusinessTravel', 'DistanceFromHome', 'OverTime', 'WorkLifeBalance']] = sc

X_train.head()
# Save the first 5 rows to a CSV file
```

Out[239]:

	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance
155	-0.258384	-0.133780	1.242427	-0.926767
453	1.624836	-0.621303	1.242427	0.403705
22	-0.258384	1.206910	1.242427	-0.926767
310	1.624836	1.572553	-0.804876	-0.926767
46	1.624836	-0.986946	-0.804876	-0.926767

In [240…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[ 2 50]
 [ 2 41]]

Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.04      0.07        52
           1       0.45      0.95      0.61        43

    accuracy                           0.45        95
   macro avg       0.48      0.50      0.34        95
weighted avg       0.48      0.45      0.32        95
```

In [241…

```
# Logistic regression model
import statsmodels.api as sm
logm1 = sm.Logit(y_train,(sm.add_constant(X_train)))
logm1.fit().summary()
```

```
Optimization terminated successfully.
         Current function value: 0.621045
         Iterations 5
```

Out[241]:

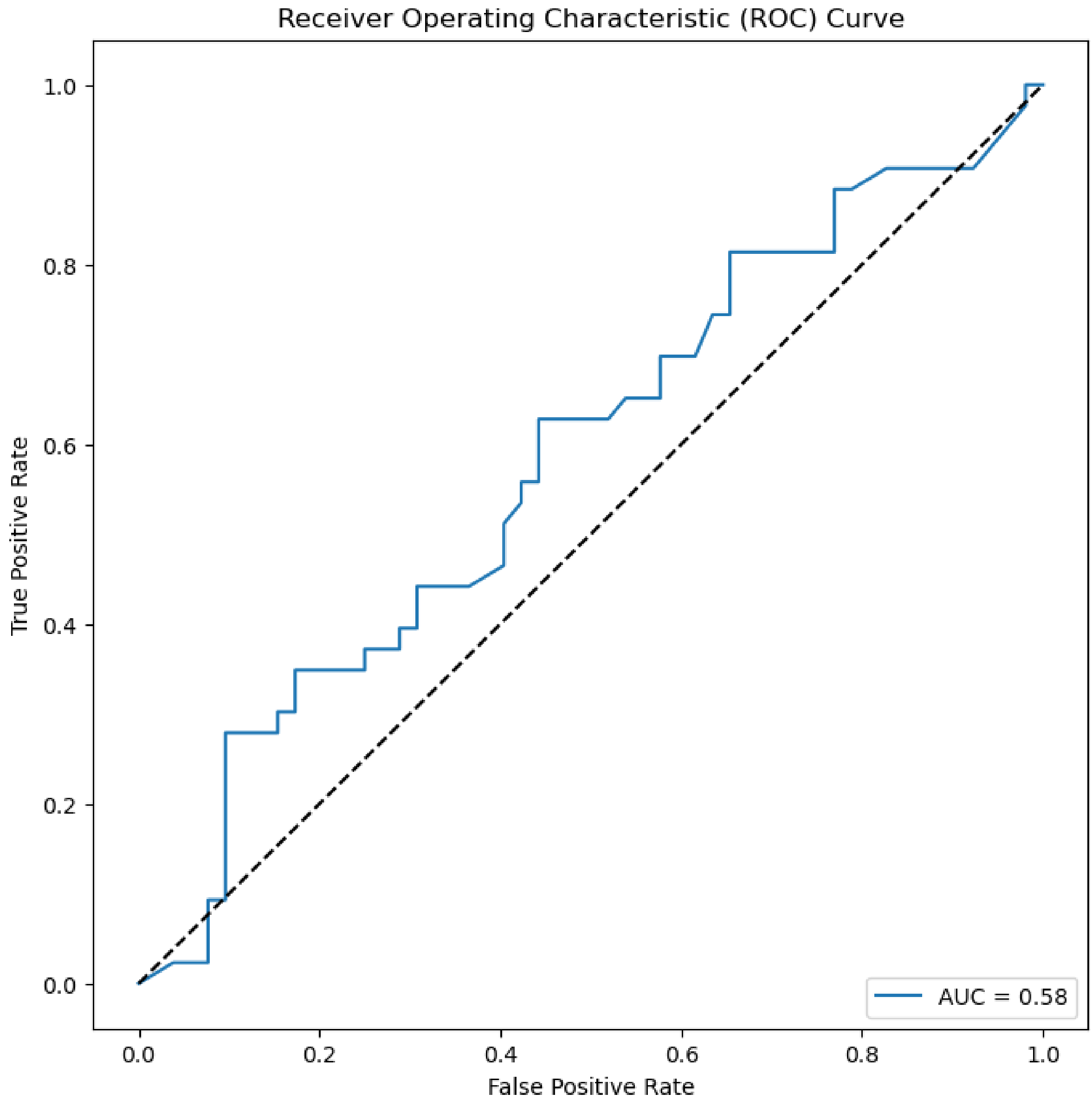
Logit Regression Results

Dep. Variable:	Attrition	**No. Observations:**	379
Model:	Logit	**Df Residuals:**	374
Method:	MLE	**Df Model:**	4
Date:	Mon, 13 Nov 2023	**Pseudo R-squ.:**	0.1037
Time:	16:16:03	**Log-Likelihood:**	-235.38
converged:	True	**LL-Null:**	-262.60
Covariance Type:	nonrobust	**LLR p-value:**	4.256e-11

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0671	0.111	0.605	0.545	-0.150	0.284
BusinessTravel	0.3364	0.114	2.953	0.003	0.113	0.560
DistanceFromHome	0.1364	0.111	1.228	0.219	-0.081	0.354
OverTime	0.6743	0.113	5.983	0.000	0.453	0.895
WorkLifeBalance	-0.2094	0.111	-1.878	0.060	-0.428	0.009

In [242…

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

# Assuming you have your X_train, X_test, y_train, and y_test

# Train LogisticRegressionCV model
logm1 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm1.fit(X_train, y_train)

y_probs = logm1.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.5843
```

In [243…

```
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.620879
         Iterations 5
AIC: 480.6265
```

Model 2

Including Personal Factors

In [244…

```
df.head(2)
```

Out[244]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [245…

```
df2 = pd.concat([df1, df[["Age", "Gender", "MaritalStatus", "Education"]]], axis=1)
df2
```

Out[245]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Ma
0	1	1	10	0	3	35	Male	
1	1	2	8	0	2	30	Female	
2	0	2	18	0	2	35	Male	
3	1	1	24	1	2	53	Male	
4	0	1	2	0	3	32	Male	
...	...	...	...	...	...	...	...	
469	0	2	1	0	3	26	Female	
470	1	1	3	1	3	30	Female	
471	1	1	2	0	3	58	Male	
472	1	1	7	1	3	23	Male	
473	0	1	14	0	4	40	Male	

474 rows × 9 columns

In [246…

```
# Create a mapping dictionary
mapping1 = {'Male': 1, 'Female': 0}
mapping2= {'Single':2, 'Married':1, 'Divorced':0 }

# Apply the mapping to the specified column
df2['Gender'] = df2['Gender'].map(mapping1)
df2['MaritalStatus'] = df2['MaritalStatus'].map(mapping2)
df2
```

Out[246]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Ma
0	1	1	10	0	3	35	1	
1	1	2	8	0	2	30	0	
2	0	2	18	0	2	35	1	
3	1	1	24	1	2	53	1	
4	0	1	2	0	3	32	1	
...	...	...	...	...	...	...	...	
469	0	2	1	0	3	26	0	
470	1	1	3	1	3	30	0	
471	1	1	2	0	3	58	1	
472	1	1	7	1	3	23	1	
473	0	1	14	0	4	40	1	

474 rows × 9 columns

In [247…

```
X = df2.drop(["Attrition"], axis=1)
y = df2["Attrition"]
```

In [248…

```
# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [249…

```
df2.columns
```

Out[249]:

```
Index(['Attrition', 'BusinessTravel', 'DistanceFromHome', 'OverTime',
       'WorkLifeBalance', 'Age', 'Gender', 'MaritalStatus', 'Education'],
      dtype='object')
```

In [250…

```
X_train[['BusinessTravel', 'DistanceFromHome', 'OverTime', 'WorkLifeBalance', 'Age'

X_train.head()
```

Out[250]:

	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Marita
155	-0.258384	-0.133780	1.242427	-0.926767	0.082623	0.878082	-0.
453	1.624836	-0.621303	1.242427	0.403705	-1.854342	0.878082	1.
22	-0.258384	1.206910	1.242427	-0.926767	0.620670	-1.138846	1.
310	1.624836	1.572553	-0.804876	-0.926767	-0.563032	0.878082	-1.
46	1.624836	-0.986946	-0.804876	-0.926767	-0.778250	0.878082	1.

In [251…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)
```

```
# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[52  0]
 [43  0]]

Classification Report:
               precision    recall  f1-score   support

           0       0.55      1.00      0.71        52
           1       0.00      0.00      0.00        43

    accuracy                           0.55        95
   macro avg       0.27      0.50      0.35        95
weighted avg       0.30      0.55      0.39        95
```

```
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [252…

```
# Logistic regression model
import statsmodels.api as sm
logm2 = sm.GLM(y_train,(sm.add_constant(X_train)))
logm2.fit().summary()
```

Out[252]:

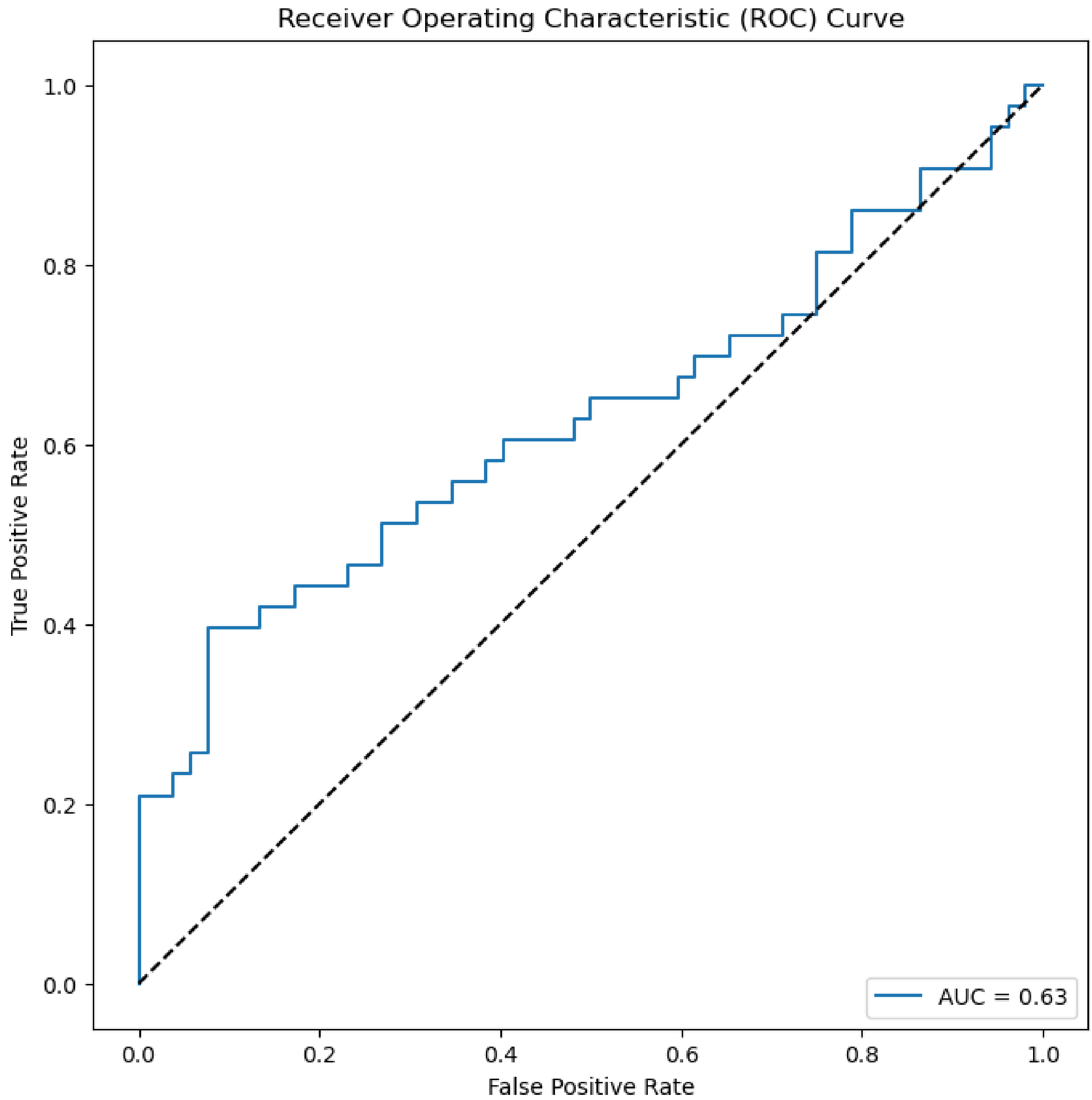
Generalized Linear Model Regression Results

Dep. Variable:	Attrition	**No. Observations:**	379
Model:	GLM	**Df Residuals:**	370
Model Family:	Gaussian	**Df Model:**	8
Link Function:	identity	**Scale:**	0.19912
Method:	IRLS	**Log-Likelihood:**	-227.40
Date:	Mon, 13 Nov 2023	**Deviance:**	73.673
Time:	16:16:05	**Pearson chi2:**	73.7
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.2434
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5119	0.023	22.332	0.000	0.467	0.557
BusinessTravel	0.0756	0.023	3.264	0.001	0.030	0.121
DistanceFromHome	0.0271	0.023	1.174	0.240	-0.018	0.072
OverTime	0.1536	0.023	6.676	0.000	0.109	0.199
WorkLifeBalance	-0.0455	0.023	-1.968	0.049	-0.091	-0.000
Age	-0.0807	0.024	-3.357	0.001	-0.128	-0.034
Gender	0.0586	0.023	2.530	0.011	0.013	0.104
MaritalStatus	0.0950	0.023	4.064	0.000	0.049	0.141
Education	0.0115	0.024	0.483	0.629	-0.035	0.058

In [253…

```
# Train LogisticRegressionCV model
logm2 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm2.fit(X_train, y_train)


# Predict probabilities
y_probs = logm2.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.6275
```

In [254…

```python
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.569084
         Iterations 5
AIC: 449.3656
```

Model 3

Including Overall Satsifaction

In [255…

```python
df.head(2)
```

Out[255]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [256…

```
df3 = pd.concat([df2, df[["JobSatisfaction", "EnvironmentSatisfaction", "Relationsh
df3 = df3.drop(["Education"], axis= 1)
df3
```

Out[256]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Ma
0	1	1	10	0	3	35	1	
1	1	2	8	0	2	30	0	
2	0	2	18	0	2	35	1	
3	1	1	24	1	2	53	1	
4	0	1	2	0	3	32	1	
...	...	...	...	...	...	...	...	
469	0	2	1	0	3	26	0	
470	1	1	3	1	3	30	0	
471	1	1	2	0	3	58	1	
472	1	1	7	1	3	23	1	
473	0	1	14	0	4	40	1	

474 rows × 11 columns

In [257…

```
X = df3.drop(["Attrition"], axis=1)
y = df3["Attrition"]
```

No mapping required

In [258…

```
# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [259…

```
df3.columns
```

Out[259]:

```
Index(['Attrition', 'BusinessTravel', 'DistanceFromHome', 'OverTime',
       'WorkLifeBalance', 'Age', 'Gender', 'MaritalStatus', 'JobSatisfaction',
       'EnvironmentSatisfaction', 'RelationshipSatisfaction'],
      dtype='object')
```

In [260…

```
X_train[['BusinessTravel', 'DistanceFromHome', 'OverTime', 'WorkLifeBalance', 'Age'

X_train.head()
```

Out[260]:

	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Marita
155	-0.258384	-0.133780	1.242427	-0.926767	0.082623	0.878082	-0.:
453	1.624836	-0.621303	1.242427	0.403705	-1.854342	0.878082	1.(
22	-0.258384	1.206910	1.242427	-0.926767	0.620670	-1.138846	1.(
310	1.624836	1.572553	-0.804876	-0.926767	-0.563032	0.878082	-1.(
46	1.624836	-0.986946	-0.804876	-0.926767	-0.778250	0.878082	1.(

In [261…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[52  0]
 [43  0]]

Classification Report:
               precision    recall  f1-score   support

           0       0.55      1.00      0.71        52
           1       0.00      0.00      0.00        43

    accuracy                           0.55        95
   macro avg       0.27      0.50      0.35        95
weighted avg       0.30      0.55      0.39        95
```

```
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\acer\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [262…

```
# Logistic regression model
import statsmodels.api as sm
logm3 = sm.GLM(y_train,(sm.add_constant(X_train)))
logm3.fit().summary()
```

Out[262]:

Generalized Linear Model Regression Results

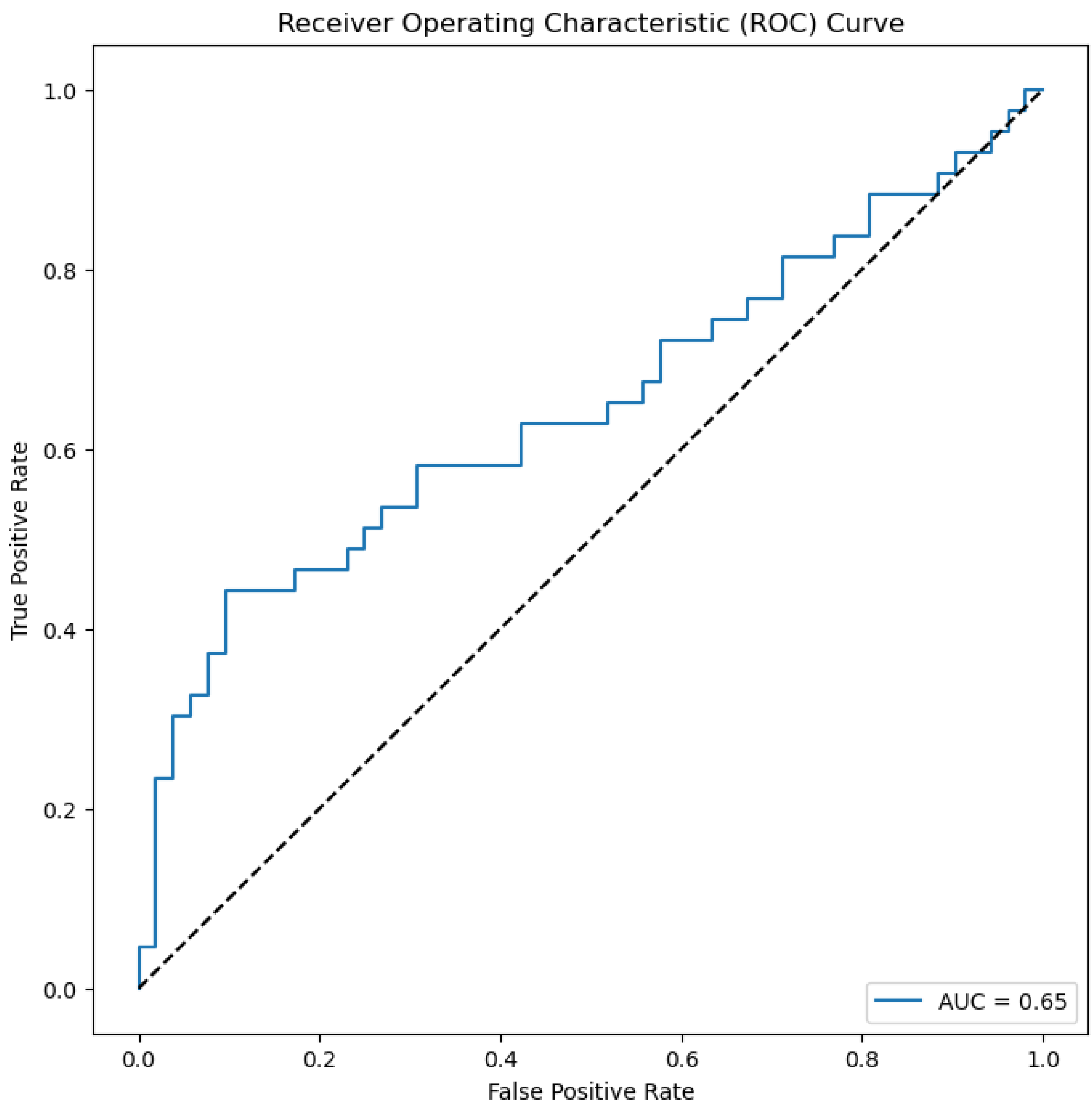
Dep. Variable:	Attrition	**No. Observations:**	379
Model:	GLM	**Df Residuals:**	368
Model Family:	Gaussian	**Df Model:**	10
Link Function:	identity	**Scale:**	0.19349
Method:	IRLS	**Log-Likelihood:**	-220.94
Date:	Mon, 13 Nov 2023	**Deviance:**	71.205
Time:	16:16:06	**Pearson chi2:**	71.2
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.2744
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5119	0.023	22.654	0.000	0.468	0.556
BusinessTravel	0.0786	0.023	3.440	0.001	0.034	0.123
DistanceFromHome	0.0293	0.023	1.288	0.198	-0.015	0.074
OverTime	0.1630	0.023	7.115	0.000	0.118	0.208
WorkLifeBalance	-0.0530	0.023	-2.306	0.021	-0.098	-0.008
Age	-0.0667	0.023	-2.862	0.004	-0.112	-0.021
Gender	0.0615	0.023	2.691	0.007	0.017	0.106
MaritalStatus	0.1004	0.023	4.339	0.000	0.055	0.146
JobSatisfaction	-0.0403	0.023	-1.764	0.078	-0.085	0.004
EnvironmentSatisfaction	-0.0485	0.023	-2.123	0.034	-0.093	-0.004
RelationshipSatisfaction	-0.0524	0.023	-2.275	0.023	-0.098	-0.007

In [263…

```
# Train LogisticRegressionCV model
logm3 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm3.fit(X_train, y_train)


# Predict probabilities
y_probs = logm3.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.6471
```

In [264…

```
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.553296
         Iterations 5
AIC: 441.3982
```

Model 4

Including Employee Engagement

In [265…
```
df.head(2)
```

Out[265]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [266…
```
df4 = pd.concat([df3, df[["TotalWorkingYears", "NumCompaniesWorked","YearsAtCompany
df4 = df4.drop(["RelationshipSatisfaction"], axis= 1)
df4
```

Out[266]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Ma
0	1	1	10	0	3	35	1	
1	1	2	8	0	2	30	0	
2	0	2	18	0	2	35	1	
3	1	1	24	1	2	53	1	
4	0	1	2	0	3	32	1	
...	...	...	...	...	...	...	...	
469	0	2	1	0	3	26	0	
470	1	1	3	1	3	30	0	
471	1	1	2	0	3	58	1	
472	1	1	7	1	3	23	1	
473	0	1	14	0	4	40	1	

474 rows × 15 columns

In [267…
```
X = df4.drop(["Attrition"], axis=1)
y = df4["Attrition"]
```

No Mapping Required

In [268…
```
# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [269…
```
df4.columns
```

Out[269]:
```
Index(['Attrition', 'BusinessTravel', 'DistanceFromHome', 'OverTime',
       'WorkLifeBalance', 'Age', 'Gender', 'MaritalStatus', 'JobSatisfaction',
       'EnvironmentSatisfaction', 'TotalWorkingYears', 'NumCompaniesWorked',
       'YearsAtCompany', 'YearsWithCurrManager', 'YearsInCurrentRole'],
      dtype='object')
```

In [270…
```
X_train[['BusinessTravel', 'DistanceFromHome', 'OverTime', 'WorkLifeBalance', 'Age'
X_train.head()
```

Out[270]:

	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance	Age	Gender	Marita
155	-0.258384	-0.133780	1.242427	-0.926767	0.082623	0.878082	-0.
453	1.624836	-0.621303	1.242427	0.403705	-1.854342	0.878082	1.
22	-0.258384	1.206910	1.242427	-0.926767	0.620670	-1.138846	1.
310	1.624836	1.572553	-0.804876	-0.926767	-0.563032	0.878082	-1.
46	1.624836	-0.986946	-0.804876	-0.926767	-0.778250	0.878082	1.

In [271…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[52  0]
 [41  2]]

Classification Report:
               precision    recall  f1-score   support

           0       0.56      1.00      0.72        52
           1       1.00      0.05      0.09        43

    accuracy                           0.57        95
   macro avg       0.78      0.52      0.40        95
weighted avg       0.76      0.57      0.43        95
```

In [272…

```
# Logistic regression model
import statsmodels.api as sm
logm4 = sm.GLM(y_train,(sm.add_constant(X_train)))
logm4.fit().summary()
```

Out[272]:

Generalized Linear Model Regression Results

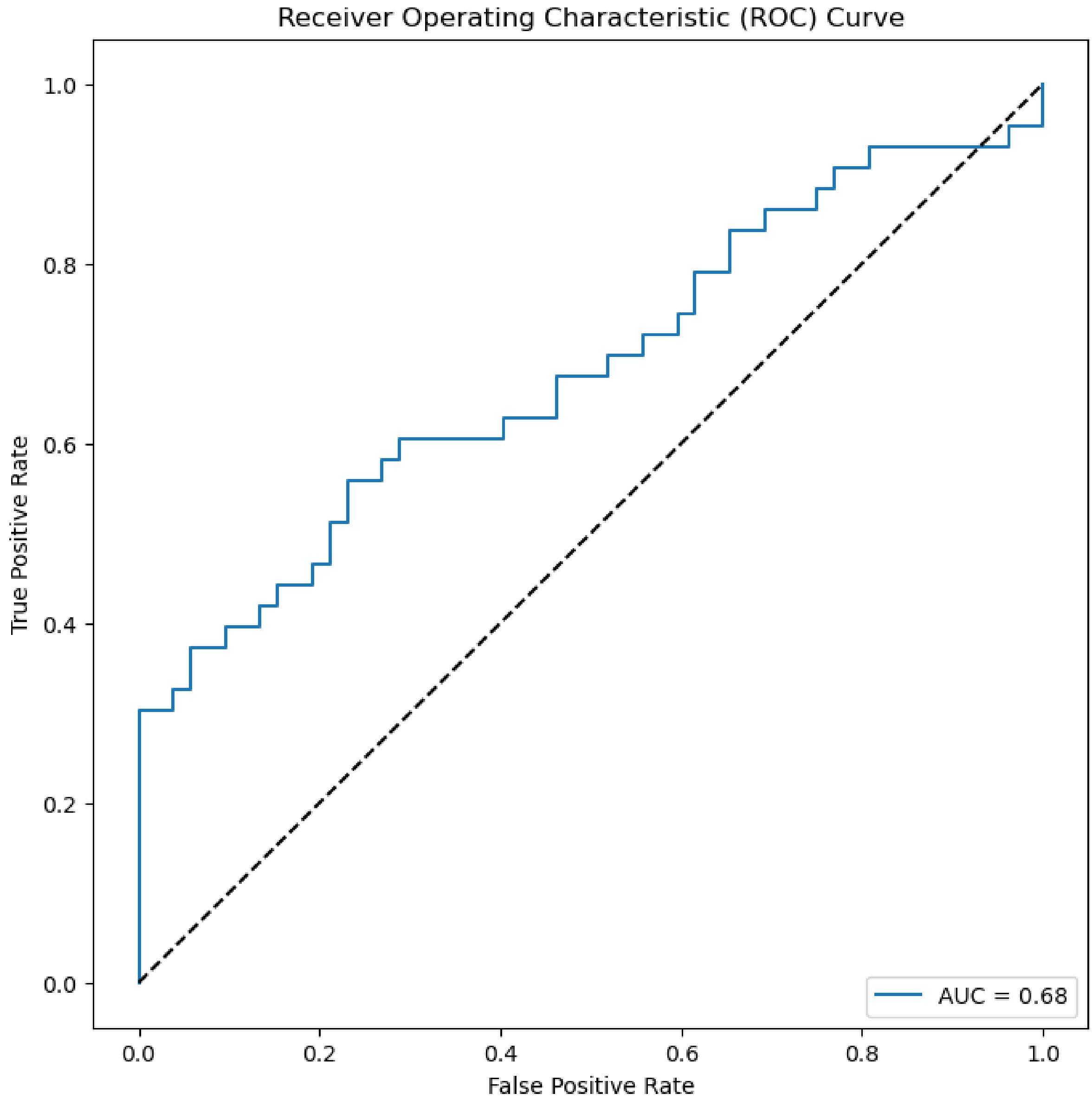
Dep. Variable:	Attrition	**No. Observations:**	379
Model:	GLM	**Df Residuals:**	364
Model Family:	Gaussian	**Df Model:**	14
Link Function:	identity	**Scale:**	0.17952
Method:	IRLS	**Log-Likelihood:**	-204.66
Date:	Mon, 13 Nov 2023	**Deviance:**	65.344
Time:	16:16:08	**Pearson chi2:**	65.3
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.3509
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5119	0.022	23.520	0.000	0.469	0.555
BusinessTravel	0.0779	0.022	3.530	0.000	0.035	0.121
DistanceFromHome	0.0375	0.022	1.693	0.091	-0.006	0.081
OverTime	0.1576	0.022	7.117	0.000	0.114	0.201
WorkLifeBalance	-0.0361	0.022	-1.625	0.104	-0.080	0.007
Age	-0.0411	0.031	-1.337	0.181	-0.101	0.019
Gender	0.0437	0.022	1.970	0.049	0.000	0.087
MaritalStatus	0.0920	0.022	4.125	0.000	0.048	0.136
JobSatisfaction	-0.0339	0.022	-1.521	0.128	-0.077	0.010
EnvironmentSatisfaction	-0.0494	0.022	-2.233	0.026	-0.093	-0.006
TotalWorkingYears	-0.1133	0.039	-2.905	0.004	-0.190	-0.037
NumCompaniesWorked	0.1037	0.025	4.202	0.000	0.055	0.152
YearsAtCompany	0.1329	0.045	2.952	0.003	0.045	0.221
YearsWithCurrManager	-0.0989	0.041	-2.402	0.016	-0.180	-0.018
YearsInCurrentRole	-0.0433	0.039	-1.119	0.263	-0.119	0.033

In [273…

```
# Train LogisticRegressionCV model
logm4 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm4.fit(X_train, y_train)


# Predict probabilities
y_probs = logm4.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
```

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

# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.6816
```

In [274…

```
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.512013
         Iterations 6
AIC: 418.1057
```

In [275…

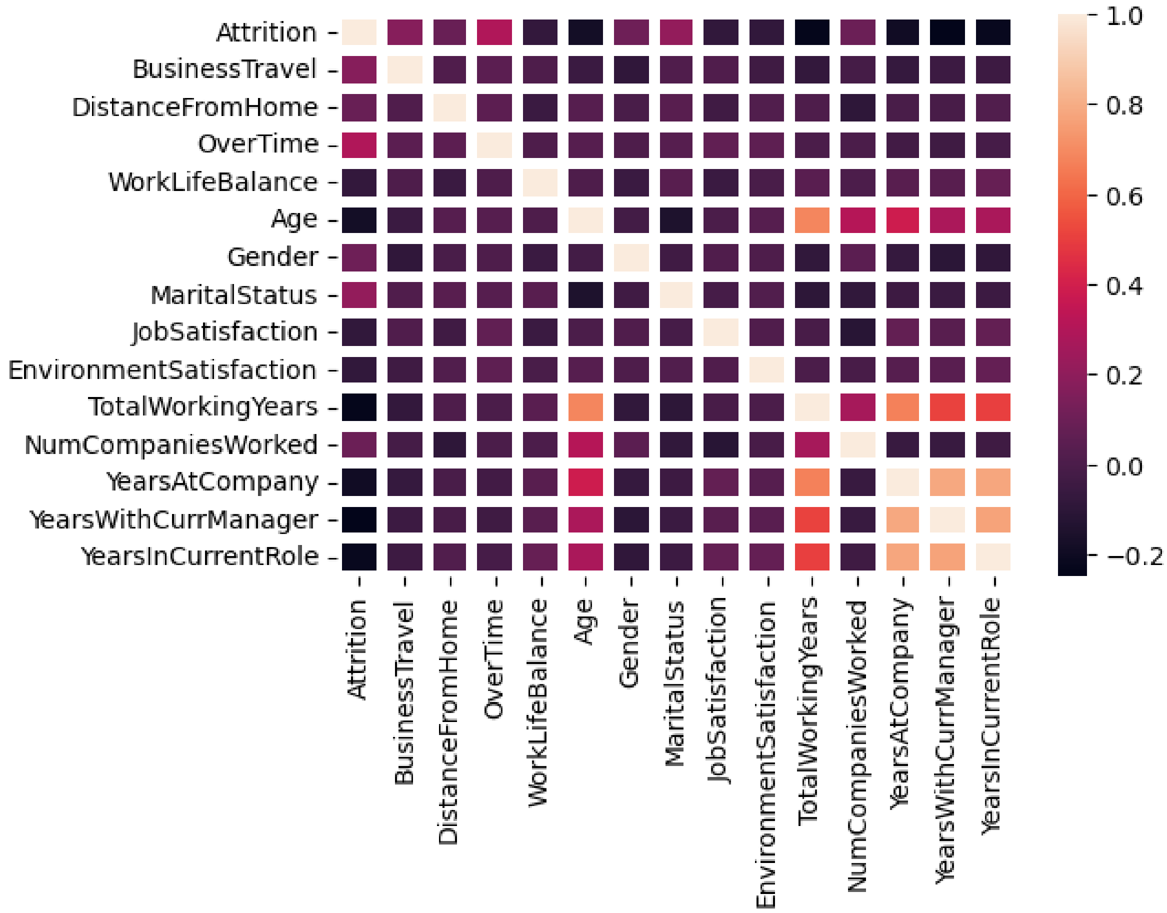
```
df4.corr(numeric_only=True)
```

Out[275]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	WorkLifeBalance
Attrition	1.000000	0.172249	0.084048	0.293803	-0.082342
BusinessTravel	0.172249	1.000000	0.012902	0.044349	0.008648
DistanceFromHome	0.084048	0.012902	1.000000	0.051422	-0.055475
OverTime	0.293803	0.044349	0.051422	1.000000	0.006919
WorkLifeBalance	-0.082342	0.008648	-0.055475	0.006919	1.000000
Age	-0.181864	-0.056005	0.030702	0.031401	0.006121
Gender	0.106868	-0.092106	-0.007661	0.009530	-0.056597
MaritalStatus	0.213730	0.011963	0.036451	0.034068	0.035214
JobSatisfaction	-0.085050	0.012519	-0.037930	0.066684	-0.056171
EnvironmentSatisfaction	-0.086241	-0.042323	0.015773	0.057224	-0.007612
TotalWorkingYears	-0.234247	-0.079355	0.007161	-0.002400	0.042227
NumCompaniesWorked	0.093153	-0.022974	-0.096496	-0.001471	0.004876
YearsAtCompany	-0.191977	-0.072465	-0.009068	-0.033154	0.036868
YearsWithCurrManager	-0.248757	-0.051867	-0.011397	-0.041218	0.034864
YearsInCurrentRole	-0.222822	-0.046202	0.014924	-0.014928	0.082539

In [276…

```
# Create a heatmap
plt.figure(figsize=(6, 4))  # Adjust the overall size of the plot
sns.heatmap(df4.corr(), linewidths=4 )
# Save the heatmap to an image file
plt.savefig('final_heatmap.png')
```



In [277…

```
from sklearn.metrics import confusion_matrix

# Assuming you have your logistic regression model 'logm1' and test data 'X_test' a
y_pred = logm4.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# True Positives, False Positives, False Negatives
tp = conf_matrix[1, 1]
fp = conf_matrix[0, 1]
fn = conf_matrix[1, 0]

# Sensitivity (Recall)
sensitivity = tp / (tp + fn)

# Precision
precision = tp / (tp + fp)

print(f"Sensitivity (Recall): {sensitivity:.4f}")
print(f"Precision: {precision:.4f}")
```

```
Sensitivity (Recall): 0.0233
Precision: 1.0000
```

Model 5

Dropping Insignificant Variables

In [278…

```
df.head(2)
```

Out[278]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [279…

```
df5=df4.drop(["WorkLifeBalance", "Age", "JobSatisfaction", "YearsInCurrentRole"], a
df5
```

Out[279]:

	Attrition	BusinessTravel	DistanceFromHome	OverTime	Gender	MaritalStatus	Environment
0	1	1	10	0	1	1	
1	1	2	8	0	0	0	
2	0	2	18	0	1	1	
3	1	1	24	1	1	2	
4	0	1	2	0	1	2	
...	...	...	...	...	...	...	
469	0	2	1	0	0	0	
470	1	1	3	1	0	2	
471	1	1	2	0	1	2	
472	1	1	7	1	1	0	
473	0	1	14	0	1	2	

474 rows × 11 columns

In [280…

```
X = df5.drop(["Attrition"], axis=1)
y = df5["Attrition"]
```

In [281…

```
# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [282…

```
df5.columns
```

Out[282]:

```
Index(['Attrition', 'BusinessTravel', 'DistanceFromHome', 'OverTime', 'Gender',
       'MaritalStatus', 'EnvironmentSatisfaction', 'TotalWorkingYears',
       'NumCompaniesWorked', 'YearsAtCompany', 'YearsWithCurrManager'],
      dtype='object')
```

In [283…

```
X_train[['BusinessTravel', 'DistanceFromHome', 'OverTime', 'Gender', 'MaritalStatus
X_train.head()
```

Out[283]:

	BusinessTravel	DistanceFromHome	OverTime	Gender	MaritalStatus	EnvironmentSatisfacti
155	-0.258384	-0.133780	1.242427	0.878082	-0.313632	1.290
453	1.624836	-0.621303	1.242427	0.878082	1.052650	-0.4718
22	-0.258384	1.206910	1.242427	-1.138846	1.052650	-0.4718
310	1.624836	1.572553	-0.804876	0.878082	-1.679915	-1.352
46	1.624836	-0.986946	-0.804876	0.878082	1.052650	0.409

In [284…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[41 11]
 [22 21]]

Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.79      0.71        52
           1       0.66      0.49      0.56        43

    accuracy                           0.65        95
   macro avg       0.65      0.64      0.64        95
weighted avg       0.65      0.65      0.64        95
```

In [285…

```
# Logistic regression model
import statsmodels.api as sm
logm5 = sm.GLM(y_train,(sm.add_constant(X_train)))
logm5.fit().summary()
```

Out[285]:

Generalized Linear Model Regression Results

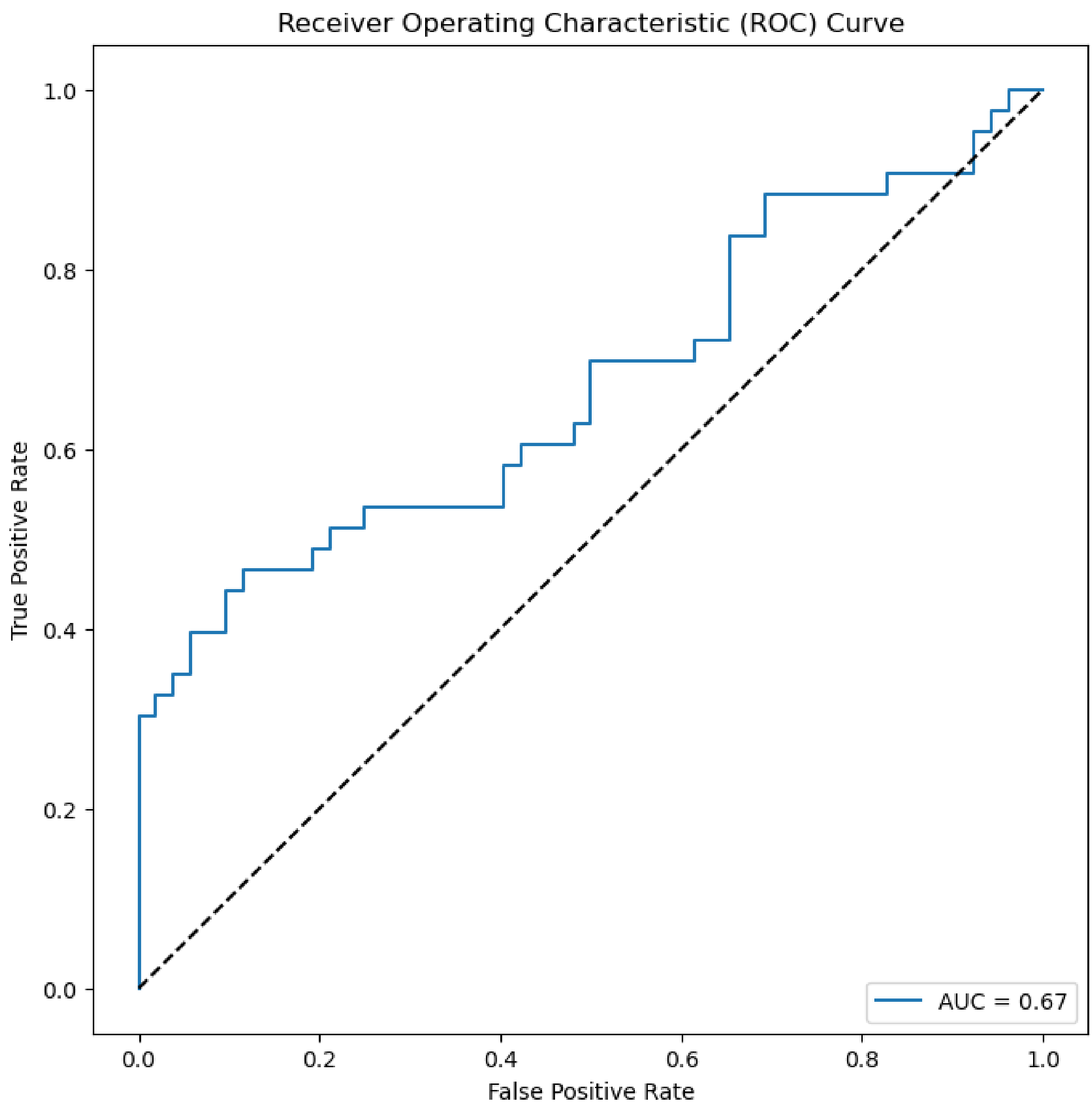
Dep. Variable:	Attrition	**No. Observations:**	379
Model:	GLM	**Df Residuals:**	368
Model Family:	Gaussian	**Df Model:**	10
Link Function:	identity	**Scale:**	0.18163
Method:	IRLS	**Log-Likelihood:**	-208.95
Date:	Mon, 13 Nov 2023	**Deviance:**	66.841
Time:	16:16:11	**Pearson chi2:**	66.8
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.3331
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.5119	0.022	23.382	0.000	0.469	0.555
BusinessTravel	0.0747	0.022	3.369	0.001	0.031	0.118
DistanceFromHome	0.0375	0.022	1.694	0.090	-0.006	0.081
OverTime	0.1527	0.022	6.898	0.000	0.109	0.196
Gender	0.0465	0.022	2.096	0.036	0.003	0.090
MaritalStatus	0.0942	0.022	4.230	0.000	0.051	0.138
EnvironmentSatisfaction	-0.0540	0.022	-2.441	0.015	-0.097	-0.011
TotalWorkingYears	-0.1378	0.033	-4.221	0.000	-0.202	-0.074
NumCompaniesWorked	0.1015	0.024	4.155	0.000	0.054	0.149
YearsAtCompany	0.1153	0.043	2.707	0.007	0.032	0.199
YearsWithCurrManager	-0.1215	0.036	-3.369	0.001	-0.192	-0.051

In [286…

```
# Train LogisticRegressionCV model
logm4 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm4.fit(X_train, y_train)


# Predict probabilities
y_probs = logm4.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.6699
```

In [287…

```
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.519656
         Iterations 6
AIC: 415.8994
```

Model 6

Dropping Distance from home variable

In [288…
```
df.head(2)
```

Out[288]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
0	35	Yes	Travel_Rarely	737	10	3	1	
1	30	Yes	Travel_Frequently	600	8	3	1	

In [289…
```
df6=df5.drop(["DistanceFromHome"], axis =1)
df6
```

Out[289]:

	Attrition	BusinessTravel	OverTime	Gender	MaritalStatus	EnvironmentSatisfaction	TotalWo
0	1	1	0	1	1	4	
1	1	2	0	0	0	3	
2	0	2	0	1	1	2	
3	1	1	1	1	2	1	
4	0	1	0	1	2	4	
...	...	...	...	...	...	...	
469	0	2	0	0	0	1	
470	1	1	1	0	2	4	
471	1	1	0	1	2	4	
472	1	1	1	1	0	3	
473	0	1	0	1	2	3	

474 rows × 10 columns

In [290…
```
df6.shape
```

Out[290]:
```
(474, 10)
```

In [291…
```
X = df6.drop(["Attrition"], axis=1)
y = df6["Attrition"]
```

In [292…
```
# Assume X is your feature matrix and y is your target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [293…
```
df6.columns
```

Out[293]:
```
Index(['Attrition', 'BusinessTravel', 'OverTime', 'Gender', 'MaritalStatus',
       'EnvironmentSatisfaction', 'TotalWorkingYears', 'NumCompaniesWorked',
       'YearsAtCompany', 'YearsWithCurrManager'],
      dtype='object')
```

In [294…
```
X_train[['BusinessTravel', 'OverTime', 'Gender', 'MaritalStatus','EnvironmentSatisf
X_train.head()
```

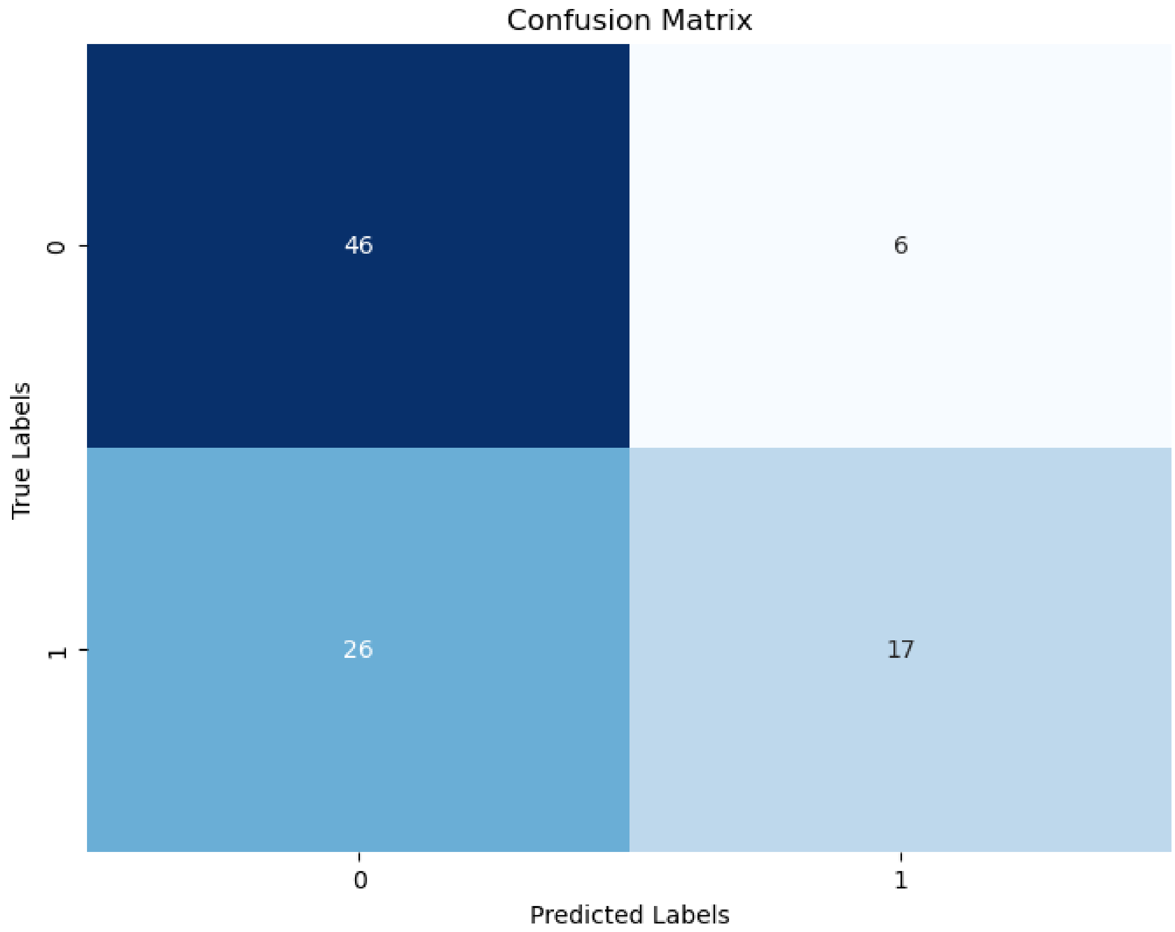
Out[294]:

	BusinessTravel	OverTime	Gender	MaritalStatus	EnvironmentSatisfaction	TotalWorkingYea
155	-0.258384	1.242427	0.878082	-0.313632	1.290161	-1.04940
453	1.624836	1.242427	0.878082	1.052650	-0.471897	-1.31859
22	-0.258384	1.242427	-1.138846	1.052650	-0.471897	-0.78021
310	1.624836	-0.804876	0.878082	-1.679915	-1.352925	0.02734
46	1.624836	-0.804876	0.878082	1.052650	0.409132	-0.24184

In [295…

```
# Create a Logistic Regression model with balanced class weights
clf = LogisticRegression(random_state=42)

# Fit the model
clf.fit(X_train, y_train)

# Predictions on the test set
y_pred = clf.predict(X_test)
cm= confusion_matrix(y_test, y_pred)
# Evaluate the model
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, annot_kws={"size": 1
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
Confusion Matrix:
 [[46  6]
 [26 17]]

Classification Report:
               precision    recall  f1-score   support

           0       0.64      0.88      0.74        52
           1       0.74      0.40      0.52        43

    accuracy                           0.66        95
   macro avg       0.69      0.64      0.63        95
weighted avg       0.68      0.66      0.64        95
```



In [296…

```
# Logistic regression model
import statsmodels.api as sm
logm5 = sm.GLM(y_train,(sm.add_constant(X_train)))
logm5.fit().summary()
```

Out[296]:

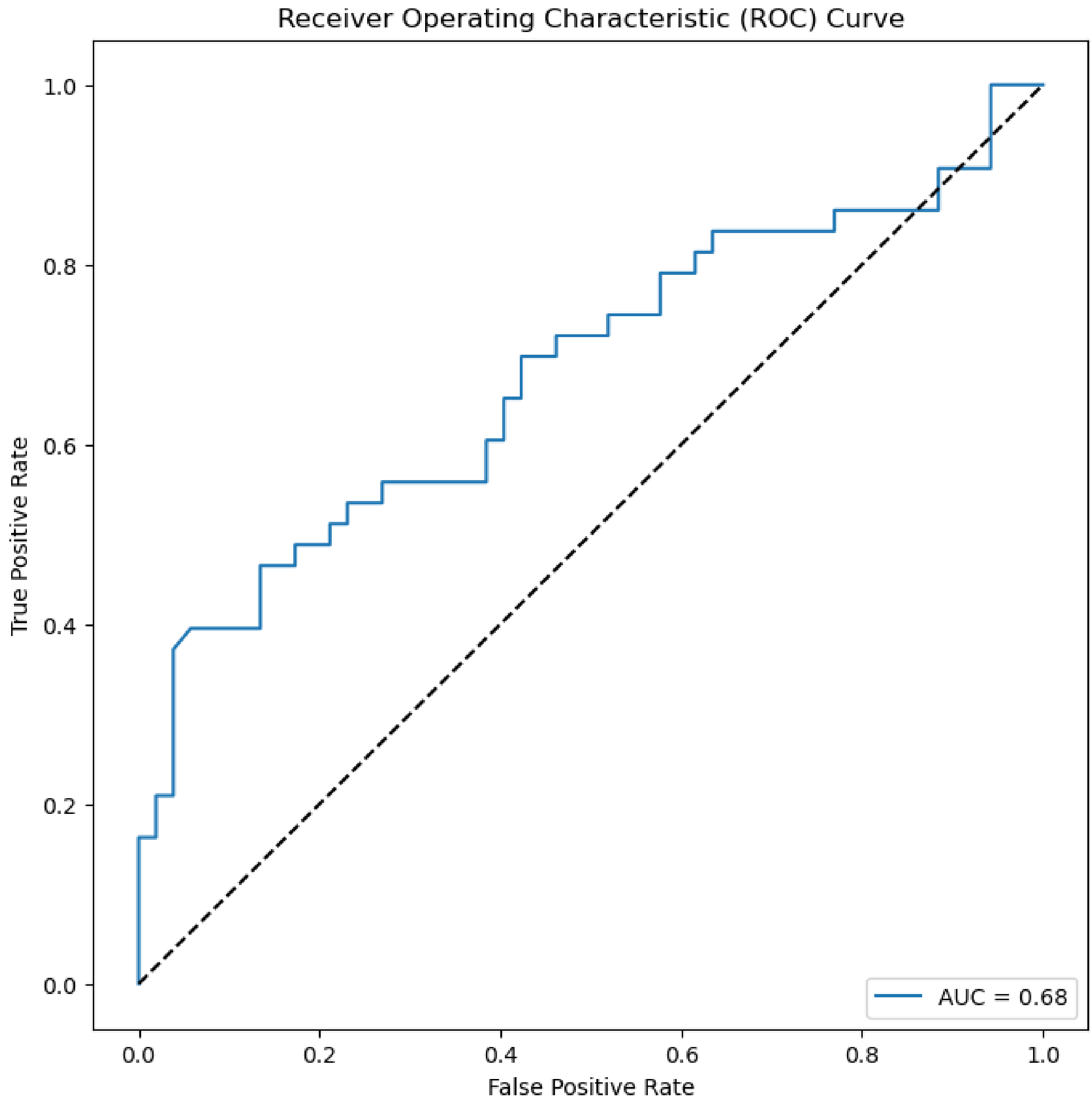
Generalized Linear Model Regression Results

Dep. Variable:	Attrition	**No. Observations:**	379
Model:	GLM	**Df Residuals:**	369
Model Family:	Gaussian	**Df Model:**	9
Link Function:	identity	**Scale:**	0.18255
Method:	IRLS	**Log-Likelihood:**	-210.43
Date:	Mon, 13 Nov 2023	**Deviance:**	67.363
Time:	16:16:13	**Pearson chi2:**	67.4
No. Iterations:	3	**Pseudo R-squ. (CS):**	0.3266
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.5119	0.022	23.323	0.000	0.469	0.555
BusinessTravel	0.0761	0.022	3.425	0.001	0.033	0.120
OverTime	0.1544	0.022	6.963	0.000	0.111	0.198
Gender	0.0465	0.022	2.093	0.036	0.003	0.090
MaritalStatus	0.0959	0.022	4.302	0.000	0.052	0.140
EnvironmentSatisfaction	-0.0542	0.022	-2.444	0.015	-0.098	-0.011
TotalWorkingYears	-0.1358	0.033	-4.154	0.000	-0.200	-0.072
NumCompaniesWorked	0.0968	0.024	3.979	0.000	0.049	0.145
YearsAtCompany	0.1162	0.043	2.722	0.006	0.033	0.200
YearsWithCurrManager	-0.1230	0.036	-3.403	0.001	-0.194	-0.052

In [297…

```
# Train LogisticRegressionCV model
logm4 = LogisticRegressionCV(cv=5)  # You can adjust the number of cross-validatior
logm4.fit(X_train, y_train)


# Predict probabilities
y_probs = logm4.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Compute AUC
auc_value = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'AUC = {auc_value:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random chance
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print AUC value
print(f'AUC: {auc_value:.4f}')
```




```
AUC: 0.6800
```

In [298…

```
# Add a constant term to the design matrix (required for statsmodels)
X = sm.add_constant(X_train)

# Fit your logistic regression model
model = sm.Probit(y_train, X).fit()

# Calculate AIC
aic_value = model.aic

print(f"AIC: {aic_value:.4f}")
```

```
Optimization terminated successfully.
         Current function value: 0.524266
         Iterations 6
AIC: 417.3938
```

In [299…

```
# Assuming you have your logistic regression model 'logm1' and test data 'X_test' a
y_pred = logm4.predict(X_test)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# True Positives, False Positives, False Negatives
tp = conf_matrix[1, 1]
fp = conf_matrix[0, 1]
```

```
fn = conf_matrix[1, 0]

# Sensitivity (Recall)
sensitivity = tp / (tp + fn)

# Precision
precision = tp / (tp + fp)

print(f"Sensitivity (Recall): {sensitivity:.4f}")
print(f"Precision: {precision:.4f}")
```

```
Sensitivity (Recall): 0.3953
Precision: 0.7391
```

In [300…

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
# Assuming you have a DataFrame named 'df' with your data
# and 'y' is your binary dependent variable
# and 'X' is a DataFrame containing your independent variables



# Create a DataFrame to store the VIF values
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1]

# Display the VIF DataFrame
print(vif_data)
```

```
                  Variable       VIF
0                    const  1.000000
1           BusinessTravel  1.023510
2                 OverTime  1.021068
3                   Gender  1.026315
4            MaritalStatus  1.032006
5  EnvironmentSatisfaction  1.019870
6        TotalWorkingYears  2.219991
7       NumCompaniesWorked  1.229104
8           YearsAtCompany  3.783904
9     YearsWithCurrManager  2.710849
```

In [301…

```
beta= 0.5119
beta0 = 0.0761
beta1 =0.1544
beta2=-0.0645
beta3=0.0955
beta4=-0.0542
beta5=-0.1358
beta6=0.0968
beta7=0.1162
beta8=-0.1230







# Display the logistic regression equation
equation = "P(Attrition=1) = 1 / (1 + e^(-({} + {} * BusinessTravel + {} * OverTime
    beta,beta0, beta1, beta2,beta3,beta4,beta5,beta6,beta7,beta8)  # Replace ... wi

print("Logistic Regression Equation:")
print(equation)
```

```
Logistic Regression Equation:
P(Attrition=1) = 1 / (1 + e^(-(0.5119 + 0.0761 * BusinessTravel + 0.1544 * OverTim
e + -0.0645 * Age + 0.0955 * MaritalStatus + -0.0542 * EnvironmentSatisfaction + -
0.1358 * TotalWorkingYEar + 0.0968 * NumCompWork + 0.1162 * YearsAtcomp + -0.123 *
YearsCurrManager  )))
```

In [302…

```
# Create a heatmap
plt.figure(figsize=(6, 4))  # Adjust the overall size of the plot
sns.heatmap(df6.corr(), linewidths=4 )
# Save the heatmap to an image file
# plt.savefig('final_heatmap6.png')
```

Out[302]: <Axes: >

In [ ]: