```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


df = pd.read_csv("/content/greendestination (1) (1).csv")


print("Missing Values:\n", df.isnull().sum())

# Convert categorical variables to numerical using Label Encoding
label_encoders = {}
categorical_columns = df.select_dtypes(include=['object']).columns

for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Drop columns that do not contribute to the analysis (e.g., EmployeeCount, EmployeeNumber)
df = df.drop(columns=['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'])

# Standardize numerical features
scaler = StandardScaler()
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the first few rows of the processed dataset
df.head()
```

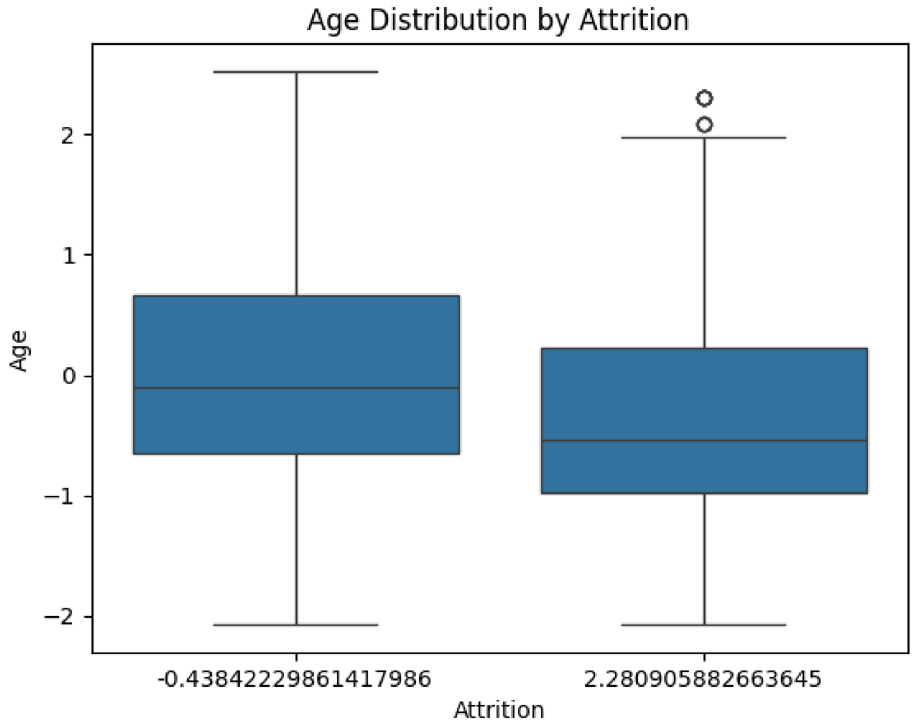
```
Missing Values:
 Age                        0
Attrition                  0
BusinessTravel             0
DailyRate                  0
Department                 0
DistanceFromHome           0
Education                  0
EducationField             0
EmployeeCount              0
EmployeeNumber             0
EnvironmentSatisfaction    0
Gender                     0
HourlyRate                 0
JobInvolvement             0
JobLevel                   0
JobRole                    0
JobSatisfaction            0
MaritalStatus              0
MonthlyIncome              0
MonthlyRate                0
NumCompaniesWorked         0
Over18                     0
OverTime                   0
PercentSalaryHike          0
PerformanceRating          0
RelationshipSatisfaction   0
StandardHours              0
StockOptionLevel           0
TotalWorkingYears          0
TrainingTimesLastYear      0
WorkLifeBalance            0
YearsAtCompany             0
YearsInCurrentRole         0
YearsSinceLastPromotion    0
YearsWithCurrManager       0
dtype: int64
```

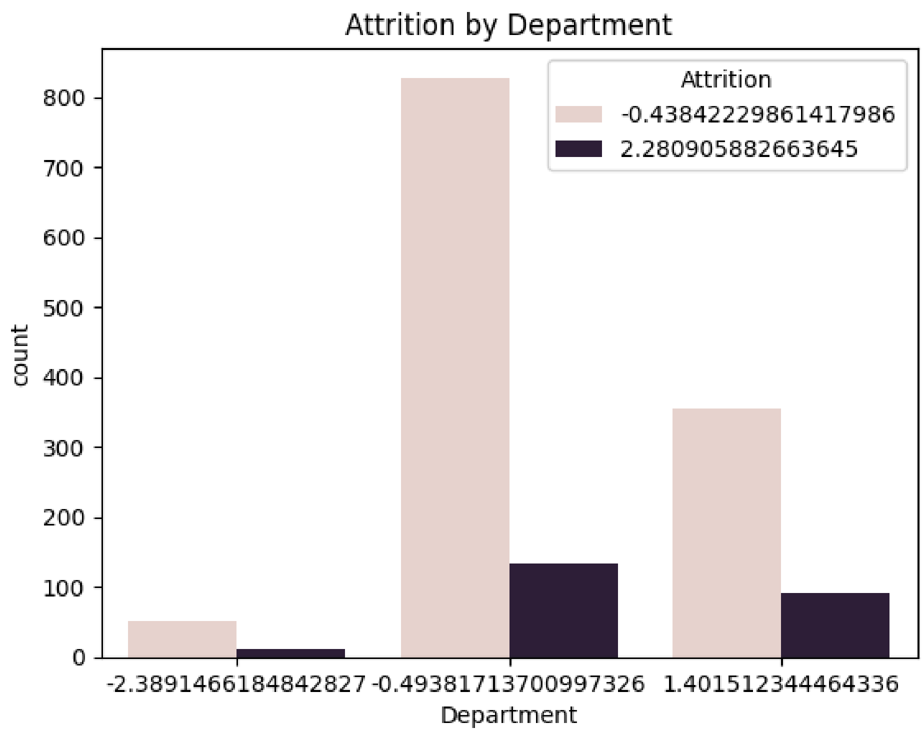
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	E
0	0.446350	2.280906	0.590048	0.742527	1.401512	-1.010909	-0.891688	-0.937414	
1	1.322365	-0.438422	-0.913194	-1.297775	-0.493817	-0.147150	-1.868426	-0.937414	
2	0.008343	2.280906	0.590048	1.414363	-0.493817	-0.887515	-0.891688	1.316673	
3	-0.429664	-0.438422	-0.913194	1.461466	-0.493817	-0.764121	1.061787	-0.937414	
4	-1.086676	-0.438422	0.590048	-0.524295	-0.493817	-0.887515	-1.868426	0.565311	

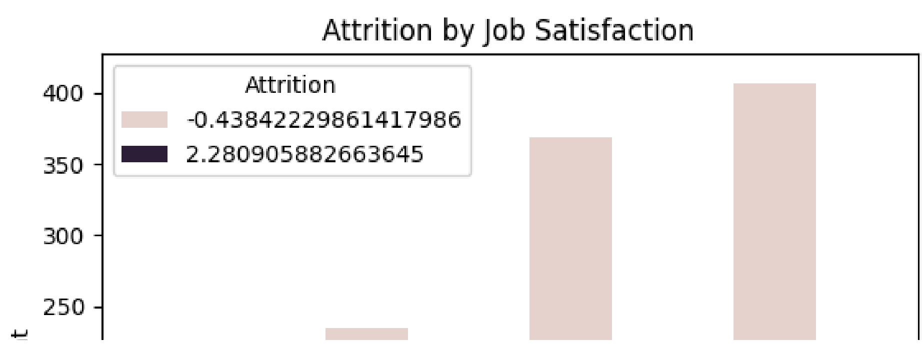
5 rows × 31 columns

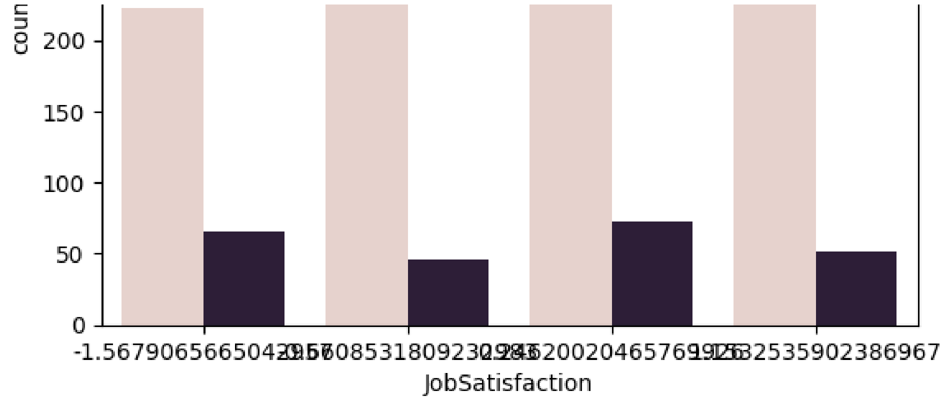
```
# Overall Attrition Rate
attrition_rate = df['Attrition'].value_counts(normalize=True)
print("Overall Attrition Rate:", attrition_rate)

# Age distribution by Attrition
sns.boxplot(x='Attrition', y='Age', data=df)
plt.title('Age Distribution by Attrition')
plt.show()

# Attrition by Department
sns.countplot(x='Department', hue='Attrition', data=df)
plt.title('Attrition by Department')
plt.show()

# Attrition by Job Satisfaction
sns.countplot(x='JobSatisfaction', hue='Attrition', data=df)
plt.title('Attrition by Job Satisfaction')
plt.show()

# Attrition by Years at Company
sns.boxplot(x='Attrition', y='YearsAtCompany', data=df)
plt.title('Years at Company by Attrition')
plt.show()

# Attrition by Monthly Income
sns.boxplot(x='Attrition', y='MonthlyIncome', data=df)
plt.title('Monthly Income by Attrition')
plt.show()
```

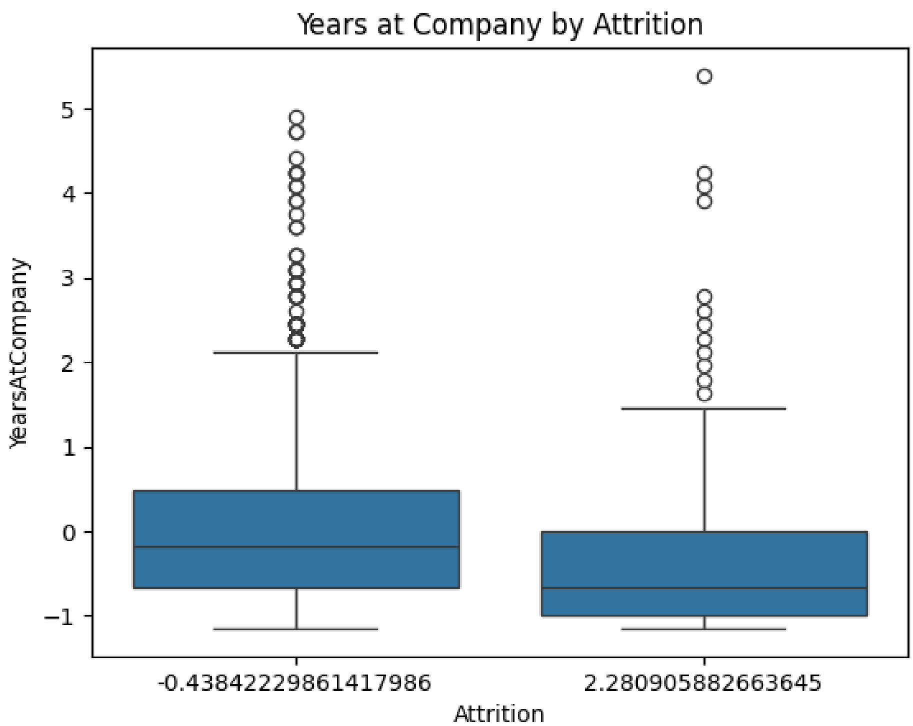
```
Overall Attrition Rate: Attrition
-0.438422    0.838776
 2.280906    0.161224
Name: proportion, dtype: float64
```

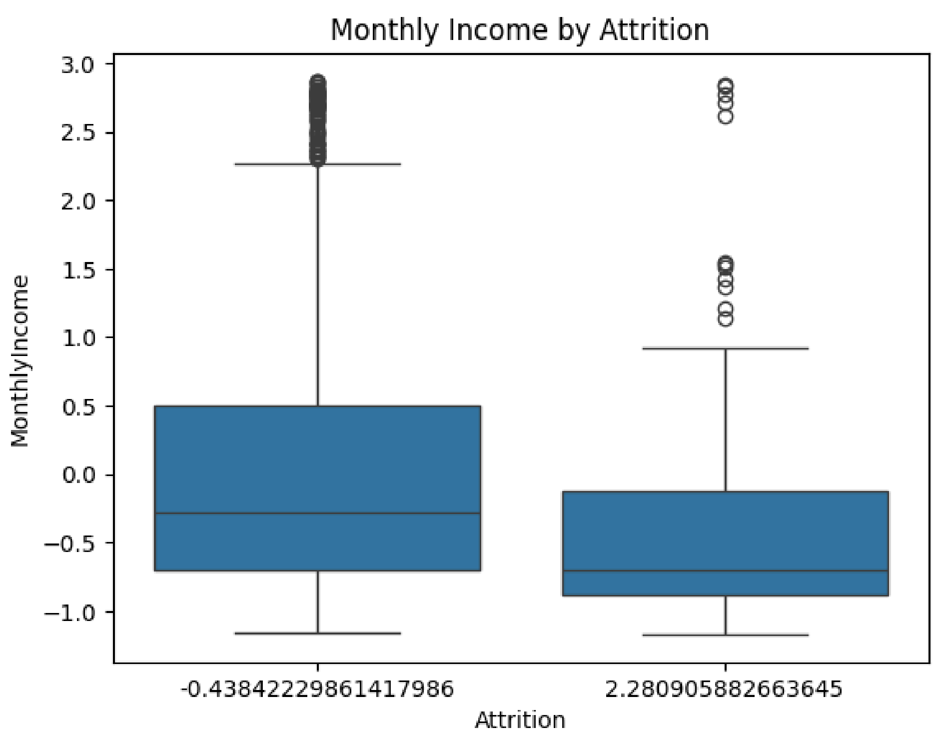




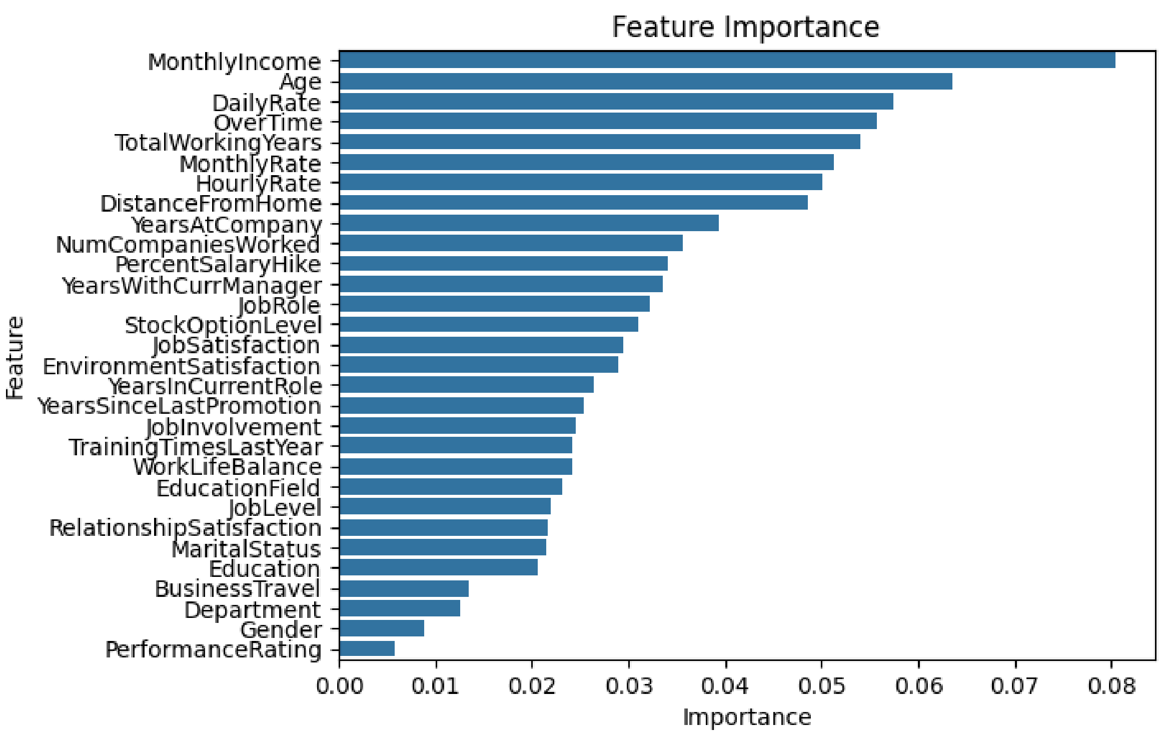



200
150
100
50
0
JobSatisfaction


Years at Company by Attrition
YearsAtCompany
5
4
3
2
1
0
−1
-0.43842229861417986
2.280905882663645
Attrition


Monthly Income by Attrition
MonthlyIncome
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
-0.43842229861417986
2.280905882663645
Attrition

```
# Define features and target variable
X = df.drop(columns=['Attrition'])
y = df['Attrition'].astype(int)

# Train a Random Forest to determine feature importance
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

# Get feature importances
importances = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances_})
importances = importances.sort_values(by='Importance', ascending=False)

# Plot feature importances
sns.barplot(x='Importance', y='Feature', data=importances)
plt.title('Feature Importance')
plt.show()
```



```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Model Evaluation
print("Logistic Regression Performance:")
print(classification_report(y_test, y_pred_log_reg))
print(f"ROC AUC: {roc_auc_score(y_test, log_reg.predict_proba(X_test)[:, 1]):.2f}\n")

print("Random Forest Performance:")
print(classification_report(y_test, y_pred_rf))
print(f"ROC AUC: {roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1]):.2f}")
```

```
Logistic Regression Performance:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       380
           2       0.54      0.25      0.34        61

    accuracy                           0.87       441
   macro avg       0.71      0.61      0.63       441
weighted avg       0.84      0.87      0.84       441

ROC AUC: 0.77

Random Forest Performance:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       380
           2       0.45      0.08      0.14        61

    accuracy                           0.86       441
   macro avg       0.66      0.53      0.53       441
weighted avg       0.81      0.86      0.81       441

ROC AUC: 0.78
```

```
# Subset the data to include only employees who left the company
df_left = df[df['Attrition'] == 1]

# Display the first few rows
df_left.head()
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Environ

0 rows × 31 columns

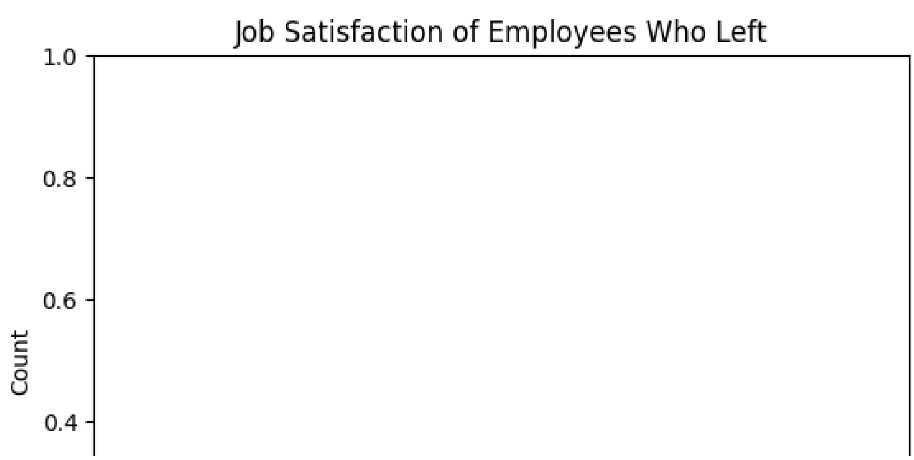
```
# Summary statistics for employees who left
summary_stats = df_left.describe()

# Display relevant statistics
summary_stats[['Age', 'YearsAtCompany', 'MonthlyIncome', 'JobSatisfaction', 'DistanceFromHome']]
```

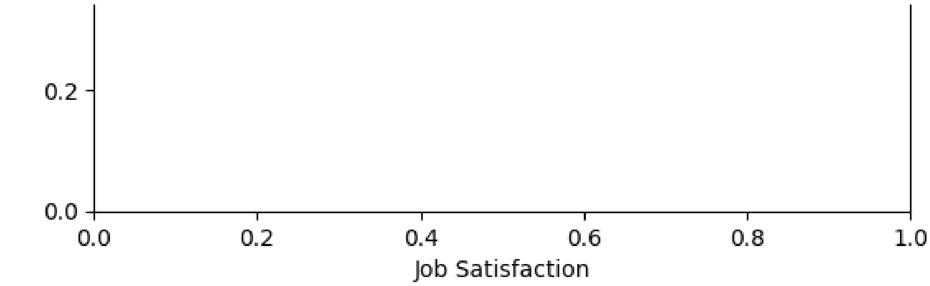
	Age	YearsAtCompany	MonthlyIncome	JobSatisfaction	DistanceFromHome
count	0.0	0.0	0.0	0.0	0.0
mean	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# Age distribution of employees who left
sns.histplot(df_left['Age'].head(100), kde=True)
plt.title('Age Distribution of Employees Who Left')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()

# Years at Company for employees who left
sns.histplot(df_left['YearsAtCompany'], kde=True)
plt.title('Years at Company for Employees Who Left')
plt.xlabel('Years at Company')
plt.ylabel('Count')
plt.show()

# Job Satisfaction distribution for employees who left
sns.countplot(x='JobSatisfaction', data=df_left)
plt.title('Job Satisfaction of Employees Who Left')
plt.xlabel('Job Satisfaction')
plt.ylabel('Count')
plt.show()

# Monthly Income distribution for employees who left
sns.histplot(df_left['MonthlyIncome'], kde=True)
plt.title('Monthly Income of Employees Who Left')
plt.xlabel('Monthly Income')
plt.ylabel('Count')
plt.show()
```


Age Distribution of Employees Who Left
0.04
0.02
0.00
−0.02
−0.04
Count
−0.04
−0.02
0.00
0.02
0.04
Age

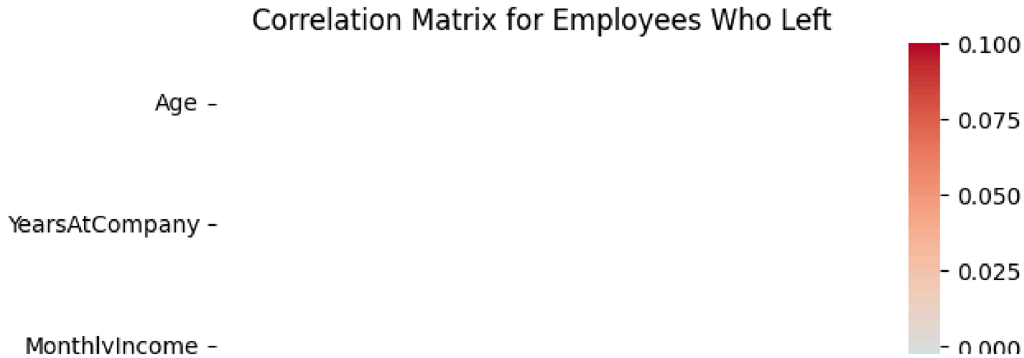

Years at Company for Employees Who Left
0.04
0.02
0.00
−0.02
−0.04
Count
−0.04
−0.02
0.00
0.02
0.04
Years at Company


Job Satisfaction of Employees Who Left
1.0
0.8
0.6
0.4
Count




```
# Correlation matrix for employees who left
correlation_matrix = df_left[['Age', 'YearsAtCompany', 'MonthlyIncome', 'JobSatisfaction', 'DistanceFromHome']].c

# Plot the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix for Employees Who Left')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:202: RuntimeWarning: All-NaN slice encountered
  vmin = np.nanmin(calc_data)
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:207: RuntimeWarning: All-NaN slice encountered
  vmax = np.nanmax(calc_data)
```




```python
# Define features and target variable for predictive analysis
X = df.drop(columns=['Attrition'])
y = df['Attrition'].astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Feature importance
importances = pd.DataFrame({'Feature': X.columns, 'Importance': rf_model.feature_importances_})
importances = importances.sort_values(by='Importance', ascending=False)

# Display the top features contributing to attrition
importances.head(10)
```

Feature Importance