Introduction

Overview of the Problem

Employee attrition is a critical issue faced by many organizations, as it can lead to increased costs related to hiring, training, and lost productivity. Understanding the factors that contribute to employee turnover can help businesses develop strategies to improve retention.

Dataset Description

In this project, we will be using an employee attrition dataset that contains various features related to employee demographics, job roles, compensation, and satisfaction levels. The dataset consists of `1470` records and includes key features such as `JobLevel`, `MonthlyIncome`, `YearsAtCompany`, `JobSatisfaction`, and `Attrition` (which indicates whether an employee has left the company).

Objective

The objective of this project is to build a machine learning model that can predict employee attrition based on the available features. By identifying the factors that contribute most to attrition, we can provide actionable insights to the organization to help reduce turnover rates.

Approach

The approach will involve the following steps:

1. Data exploration and preprocessing to clean and prepare the data for modeling.
2. Feature engineering to select and transform relevant features.
3. Model training and evaluation using techniques such as logistic regression, random forest, and grid search for hyperparameter tuning.
4. Interpretation of the model using SHAP and LIME to understand the impact of different features on attrition predictions.

Data Import and Exploration

In [51]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import roc_curve, auc, classification_report, roc
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


import warnings
warnings.filterwarnings('ignore')



df = pd.read_csv('/Users/tgisakia/Desktop/EmployeeAttrition.csv')
```

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

Out[4]:

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

5 rows × 35 columns

In [5]:
```
df.shape
```

Out[5]:
```
(1470, 35)
```

In [6]:
```
df.dtypes # Data Types of Each Column in the DataFrame
```

Out[6]:
```
Age                          int64
Attrition                   object
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
```

Data Preprocessing

In [7]:
```python
df.isnull().sum() # Checking for Missing Values in the DataFrame
```

Out[7]:
```
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
```

In [8]:
```python
df.describe() # Summary Statistics of the DataFrame
```

Out[8]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeN
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.
std	9.135373	403.509100	8.106864	1.024165	0.0	602.
min	18.000000	102.000000	1.000000	1.000000	1.0	1.
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.

8 rows × 26 columns

In [9]:
```python
df.columns # Listing All Column Names in the DataFrame
```

Out[9]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department
',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCo
unt',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'Hourly
Rate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompanies
Worked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating'
,
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel
',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance
',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromoti
on',
       'YearsWithCurrManager'],
      dtype='object')
```

In [10]:
```python
df['Attrition'].value_counts() # Count of Unique Values in the Attriti
```

Out[10]:
```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

In [11]:
```python
# Count of Unique Values in the BusinessTravel Column
df['BusinessTravel'].value_counts()
```

Out[11]:
```
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: BusinessTravel, dtype: int64
```

In [12]:
```python
# Count of Unique Values in Specific Feature Columns
df['JobInvolvement'].value_counts(), df['JobLevel'].value_counts(), df
df['JobSatisfaction'].value_counts(), df['EducationField'].value_count
```

Out[12]:
```
(3    868
 2    375
 4    144
 1     83
 Name: JobInvolvement, dtype: int64,
 1    543
 2    534
 3    218
 4    106
 5     69
 Name: JobLevel, dtype: int64,
 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,
 4    459
 3    442
 1    289
 2    280
 Name: JobSatisfaction, dtype: int64,
 Life Sciences       606
 Medical             464
 Marketing           159
 Technical Degree    132
 Other                82
 Human Resources      27
 Name: EducationField, dtype: int64)
```

In [15]:
```
# Count of Unique Values in different Columns
df['Over18'].value_counts(), df['StandardHours'].value_counts(), df['S
df['TrainingTimesLastYear'].value_counts(), df['Department'].value_cou
```

Out[15]:
```
(Y    1470
 Name: Over18, dtype: int64,
 80    1470
 Name: StandardHours, dtype: int64,
 0    631
 1    596
 2    158
 3     85
 Name: StockOptionLevel, dtype: int64,
 2    547
 3    491
 4    123
 5    119
 1     71
 6     65
 0     54
 Name: TrainingTimesLastYear, dtype: int64,
 Research & Development    961
 Sales                     446
 Human Resources            63
 Name: Department, dtype: int64)
```

Feature Engineering

In [41]:
```
# Encoding Categorical Variables and Displaying the First Few Rows of

df['Attrition'] = df['Attrition'].astype('category')
df['Gender'] = df['Gender'].astype('category')
df['BusinessTravel'] = df['BusinessTravel'].astype('category')
df['Department'] = df['Department'].astype('category')

df_encoded = pd.get_dummies(df, columns=['Attrition', 'Gender', 'Busir

df_encoded.head()
```

Out[41]:

	Age	DailyRate	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeN
0	41	1102	1	2	Life Sciences	1	
1	49	279	8	1	Life Sciences	1	
2	37	1373	2	2	Other	1	
3	33	1392	3	4	Life Sciences	1	
4	27	591	2	1	Medical	1	

5 rows × 45 columns

In [54]:
```
# Concatonating One-HOt Encoded Features into DF
df = pd.concat([df, df_encoded], axis=1)
```

In [55]:
```
# Correlation Matrix for Selected Features
df[['Department', 'Gender',
    'DistanceFromHome', 'JobInvolvement',
    'WorkLifeBalance', 'YearsSinceLastPromotion', 'YearsInCurrentRole'
    'JobRole',
    'TrainingTimesLastYear' ]].corr()
```

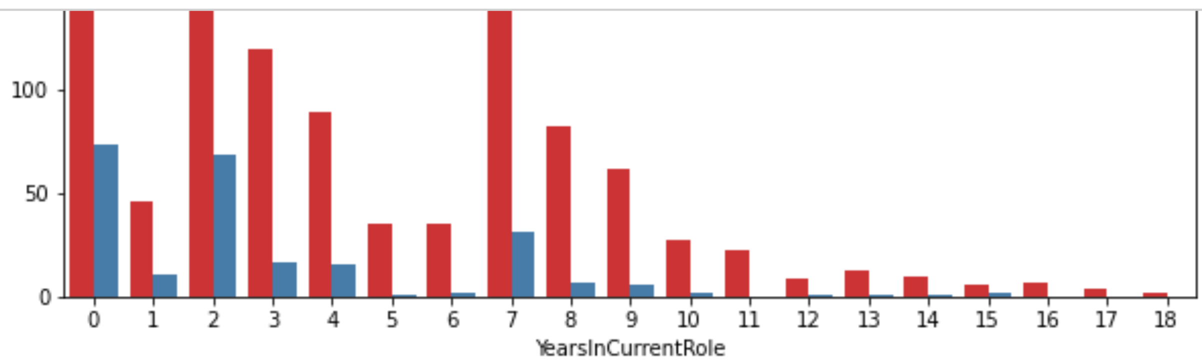
DistanceFromHome	1.000000	1.000000	0.008783	0.008
JobInvolvement	0.008783	0.008783	1.000000	1.000
JobInvolvement	0.008783	0.008783	1.000000	1.000
WorkLifeBalance	-0.026556	-0.026556	-0.014617	-0.014
WorkLifeBalance	-0.026556	-0.026556	-0.014617	-0.014
YearsSinceLastPromotion	0.010029	0.010029	-0.024184	-0.024
YearsSinceLastPromotion	0.010029	0.010029	-0.024184	-0.024
YearsInCurrentRole	0.018845	0.018845	0.008717	0.008
YearsInCurrentRole	0.018845	0.018845	0.008717	0.008
JobRole	-0.001015	-0.001015	0.006616	0.006
TrainingTimesLastYear	-0.036942	-0.036942	-0.015338	-0.015
TrainingTimesLastYear	-0.036942	-0.036942	-0.015338	-0.015

In [ ]:
```
# Notes on data above
most of these variables have very weak correlations with each other,
The only notable correlation is between "YearsInCurrentRole" and "Year
where a moderate positive relationship exists.
```

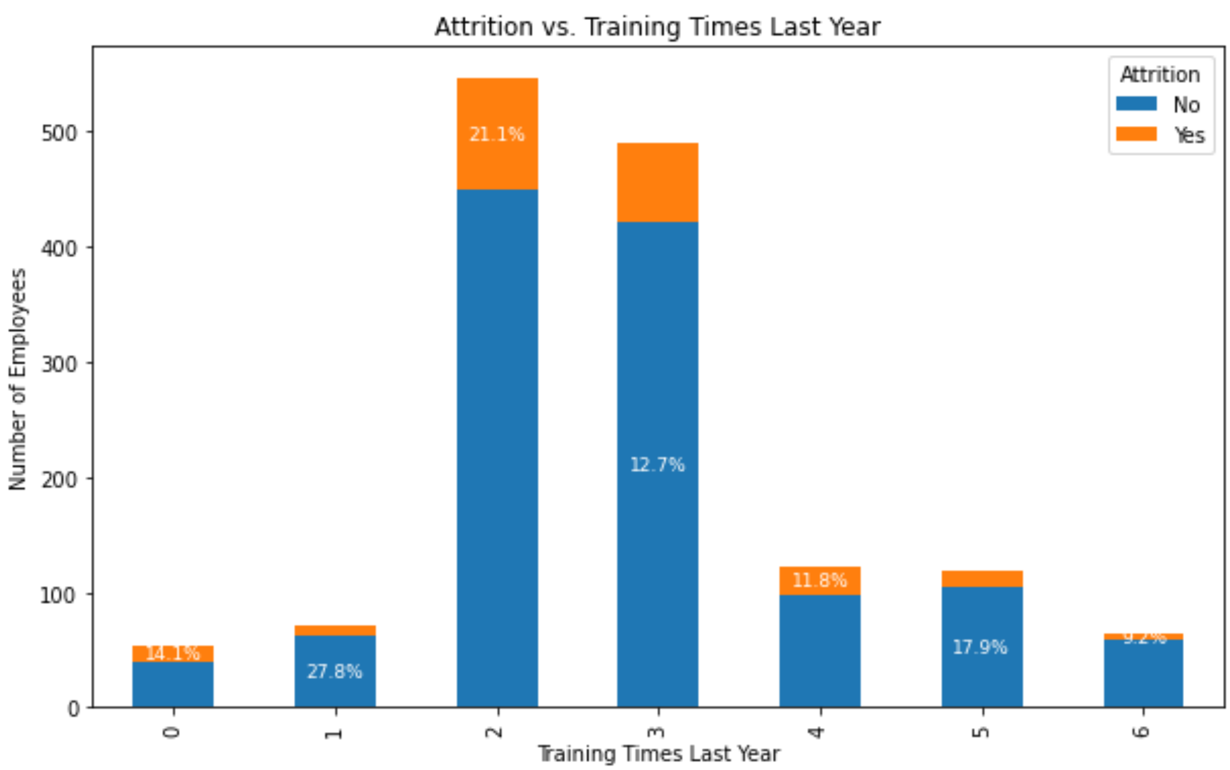
Exploratory Data Analysis (EDA)

In [18]:

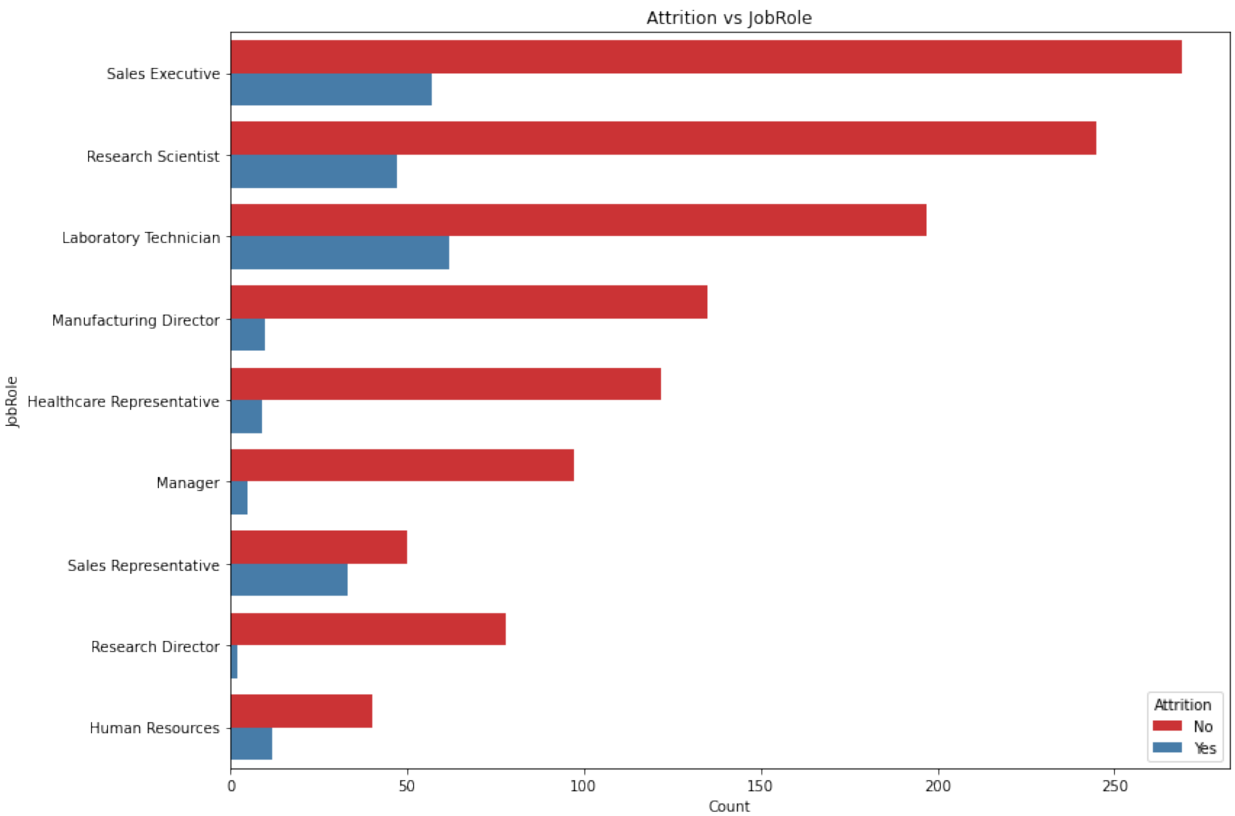
```
# Loop Through Each Column in the DataFrame
for col in df:

    # Set the Size of the Plot
    plt.figure(figsize=(10, 6))

    # Create a Count Plot for Each Column with Attrition as the Hue
    sns.countplot(x=col, hue='Attrition', data=df, palette='Set1')

    # Set the Title for Each Plot
    plt.title(f'Attrition vs {col}')

    # Label the X-Axis
    plt.xlabel(col)

    # Label the Y-Axis
    plt.ylabel('Count')

    # Add a Legend with the Title 'Attrition'
    plt.legend(title='Attrition')

    # Display the Plot
    plt.show()
```



In [17]:
```
# Calculate the counts
attrition_vs_training = df.groupby(['TrainingTimesLastYear', 'Attritic

# Calculate percentages for "Yes"
attrition_percentage = attrition_vs_training.div(attrition_vs_training

# Plot the stacked bar chart
ax = attrition_vs_training.plot(kind='bar', stacked=True, color=['#1f7

# Annotate the percentages on the bars
for idx, rect in enumerate(ax.patches):
    width, height = rect.get_width(), rect.get_height()
    x, y = rect.get_xy()

    if height > 0 and idx % 2 == 1:  # Only label the 'Yes' bars (ever
        percentage = f'{attrition_percentage.values[int(idx/2)][1]:.1f
        ax.text(x + width / 2, y + height / 2, percentage, ha='center'

plt.title('Attrition vs. Training Times Last Year')
plt.xlabel('Training Times Last Year')
plt.ylabel('Number of Employees')
plt.show()
```

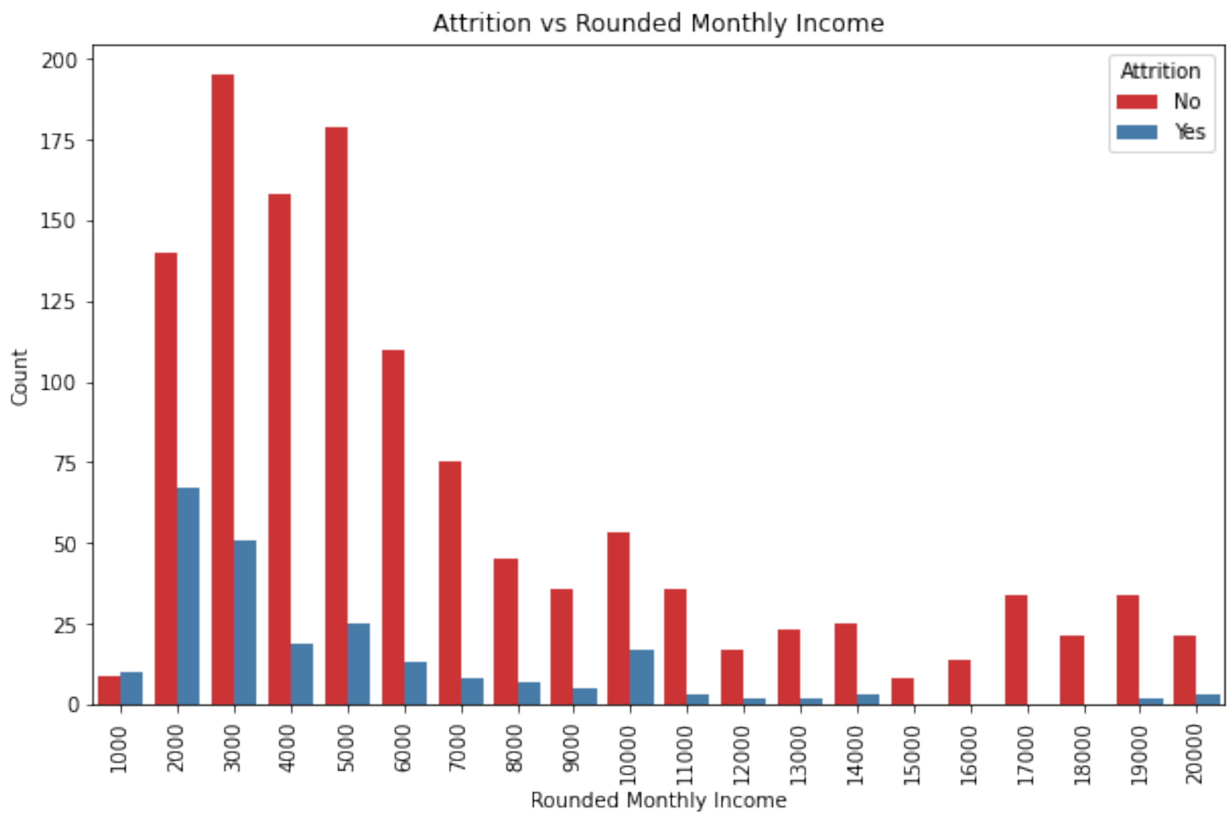


In [19]:
```
plt.figure(figsize=(12, 8))
sns.countplot(y='JobRole', hue='Attrition', data=df, palette='Set1')
plt.title('Attrition vs JobRole')
plt.xlabel('Count')
plt.ylabel('JobRole')
plt.legend(title='Attrition')
plt.tight_layout()
plt.show()
```



In [ ]:
```
#Creating a New DF with Significant Features
df.columns[['Department', 'Gender', 'DistanceFromHome', 'JobInvolvemen
'YearsSinceLastPromotion', 'YearsInCurrentRole','JobRole', 'TrainingTi
```

In [21]:
```
df['MonthlyIncomeRounded'] = df['MonthlyIncome'].round(-3)

plt.figure(figsize=(10, 6))
sns.countplot(x='MonthlyIncomeRounded', hue='Attrition', data=df, pale
plt.title('Attrition vs Rounded Monthly Income')
plt.xlabel('Rounded Monthly Income')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.legend(title='Attrition')
plt.show()
```



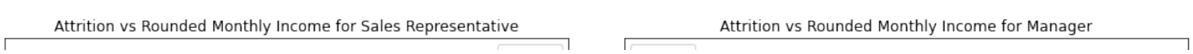
In [22]:
```
# Calculate average MonthlyIncome for each JobRole
avg_income_by_role = df.groupby('JobRole')['MonthlyIncome'].mean().sor

# Select the top and bottom paying job roles
lowest_paying_roles = avg_income_by_role.head(1).index.tolist()
highest_paying_roles = avg_income_by_role.tail(1).index.tolist()

print("Lowest Paying Role:", lowest_paying_roles)
print("Highest Paying Role:", highest_paying_roles)
```

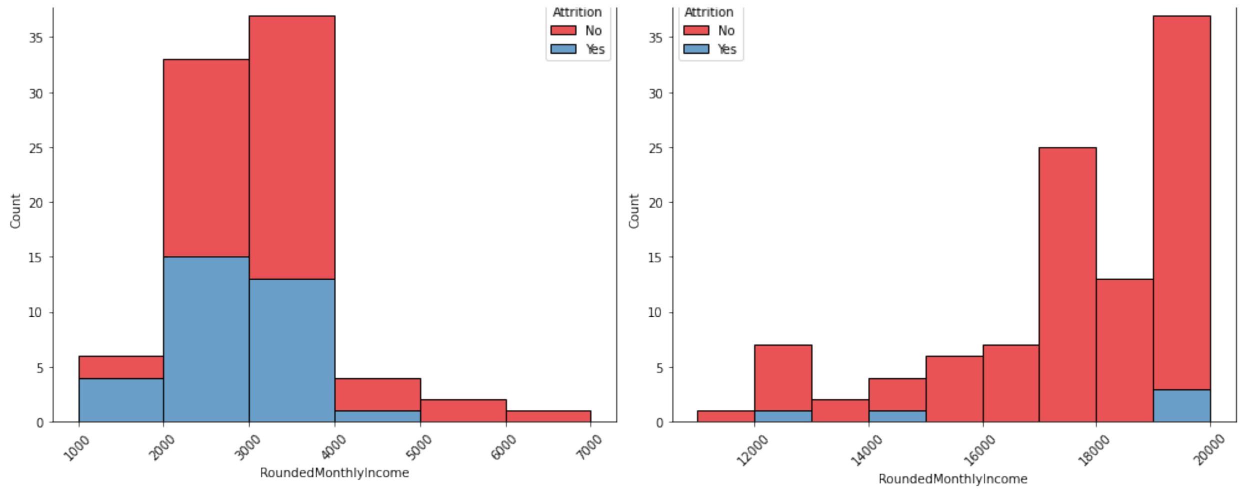
```
Lowest Paying Role: ['Sales Representative']
Highest Paying Role: ['Manager']
```

In [27]:

```python
# Filter the DataFrame to include only the relevant data
filtered_df = df[df['JobRole'].isin(lowest_paying_roles + highest_payi

# Round the MonthlyIncome to the nearest thousand for binning
filtered_df['RoundedMonthlyIncome'] = filtered_df['MonthlyIncome'].rou

# Create the plot
plt.figure(figsize=(14, 6))

# Plot for the lowest paying role
plt.subplot(1, 2, 1)
sns.histplot(
    data=filtered_df[filtered_df['JobRole'].isin(lowest_paying_roles)]
    x='RoundedMonthlyIncome',
    hue='Attrition',
    multiple='stack',
    kde=False,
    binwidth=1000,
    palette="Set1"
)
plt.title(f'Attrition vs Rounded Monthly Income for {lowest_paying_rol
plt.xticks(rotation=45)

# Plot for the highest paying role
plt.subplot(1, 2, 2)
sns.histplot(
    data=filtered_df[filtered_df['JobRole'].isin(highest_paying_roles)
    x='RoundedMonthlyIncome',
    hue='Attrition',
    multiple='stack',
    kde=False,
    binwidth=1000,
    palette="Set1"
)
plt.title(f'Attrition vs Rounded Monthly Income for {highest_paying_rc
plt.xticks(rotation=45)

# Adjust layout
plt.tight_layout()
plt.show()
```

```
<ipython-input-27-8f3791a057a0>:5: SettingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  filtered_df['RoundedMonthlyIncome'] = filtered_df['MonthlyIncome'].
round(-3)
```





In [30]:
```python
# Filter for Sales Representatives who have attrited
attrited_sales_reps = df[(df['JobRole'] == 'Sales Representative') & (

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

Out[30]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	E
21	36	Yes	Travel_Rarely	1218	Sales	9	4	
33	39	Yes	Travel_Rarely	895	Sales	5	3	
36	50	Yes	Travel_Rarely	869	Sales	3	2	
127	19	Yes	Travel_Rarely	528	Sales	22	1	
171	19	Yes	Travel_Frequently	602	Sales	1	1	

5 rows × 36 columns

In [34]:
```python
# Get descriptive statistics for the filtered data
attrited_sales_reps.describe()
```

Out[34]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumb
count	33.000000	33.000000	33.000000	33.000000	33.0	33.000C
mean	27.787879	734.090909	8.151515	2.454545	1.0	993.2727
std	8.192060	370.205886	7.319158	0.938446	0.0	621.6094
min	18.000000	156.000000	1.000000	1.000000	1.0	27.000C
25%	21.000000	428.000000	2.000000	2.000000	1.0	494.000C
50%	27.000000	667.000000	7.000000	3.000000	1.0	952.000C
75%	31.000000	895.000000	9.000000	3.000000	1.0	1486.000C
max	50.000000	1496.000000	24.000000	4.000000	1.0	2023.000C

8 rows × 27 columns

In [36]:
```python

# Filter the data for those who have attrited
attrited_df = df[df['Attrition'] == 'Yes']

# Get descriptive statistics for the attrited employees
attrited_stats = attrited_df.describe()

attrited_stats
```

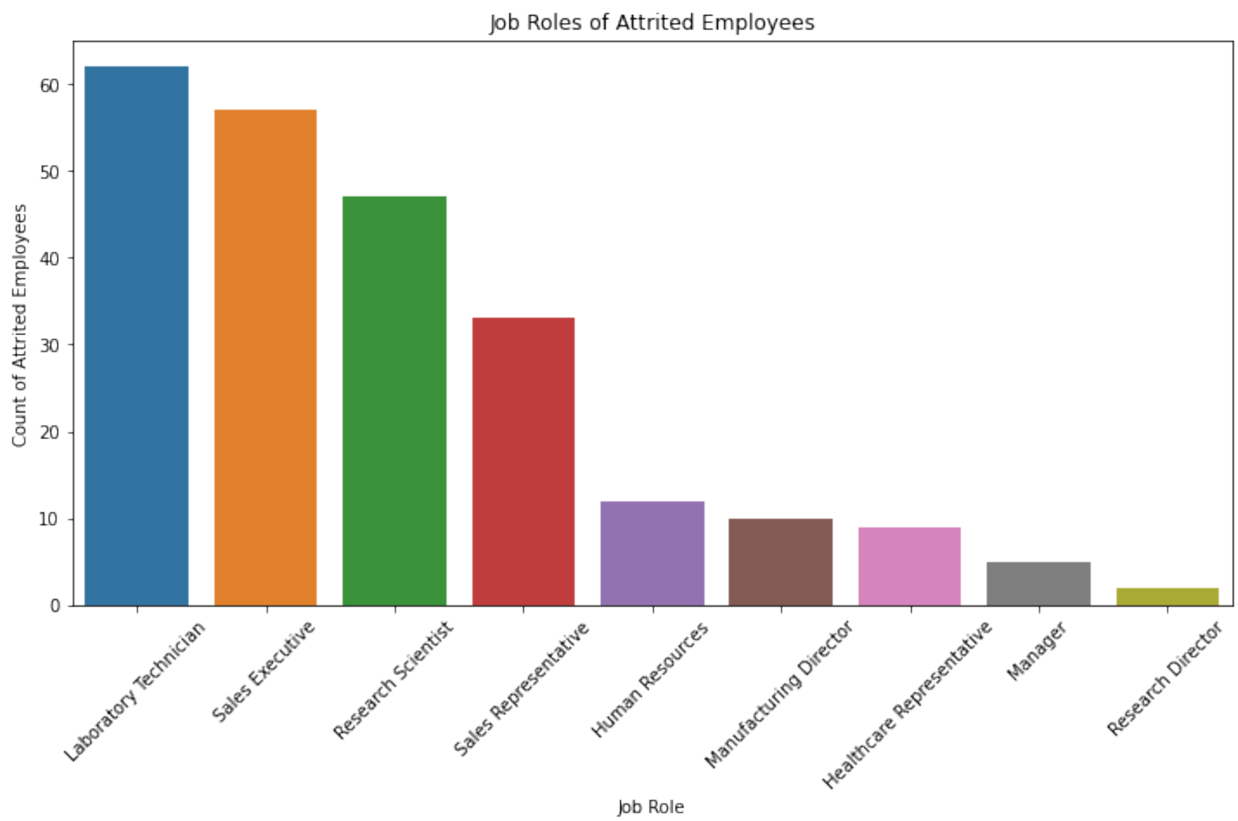
Out[36]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNu
count	237.000000	237.000000	237.000000	237.000000	237.0	237.00
mean	33.607595	750.362869	10.632911	2.839662	1.0	1010.34
std	9.689350	401.899519	8.452525	1.008244	0.0	580.75
min	18.000000	103.000000	1.000000	1.000000	1.0	1.00
25%	28.000000	408.000000	3.000000	2.000000	1.0	514.00
50%	32.000000	699.000000	9.000000	3.000000	1.0	1017.00
75%	39.000000	1092.000000	17.000000	4.000000	1.0	1486.00
max	58.000000	1496.000000	29.000000	5.000000	1.0	2055.00

8 rows × 27 columns

In [37]:

```python
# Set the Size of the Figure
plt.figure(figsize=(12, 6))

# Create a Count Plot for Job Roles of Attrited Employees
sns.countplot(data=attrited_df, x='JobRole', order=attrited_df['JobRol

# Set the Title of the Plot
plt.title('Job Roles of Attrited Employees')

# Label the X-Axis as Job Role
plt.xlabel('Job Role')

# Label the Y-Axis as Count of Attrited Employees
plt.ylabel('Count of Attrited Employees')

# Rotate the X-Axis Labels for Better Readability
plt.xticks(rotation=45)

# Display the Plot
plt.show()
```



In [38]:

```
# Filter the DataFrame for Attrited Employees
attrited_df = df[df['Attrition'] == 'Yes']

# Group by JobRole and count the number of attrited employees in each
job_role_counts = attrited_df['JobRole'].value_counts()

# Calculate the total number of attrited employees
total_attrited = job_role_counts.sum()

total_attrited
```

Out[38]:

```
237
```

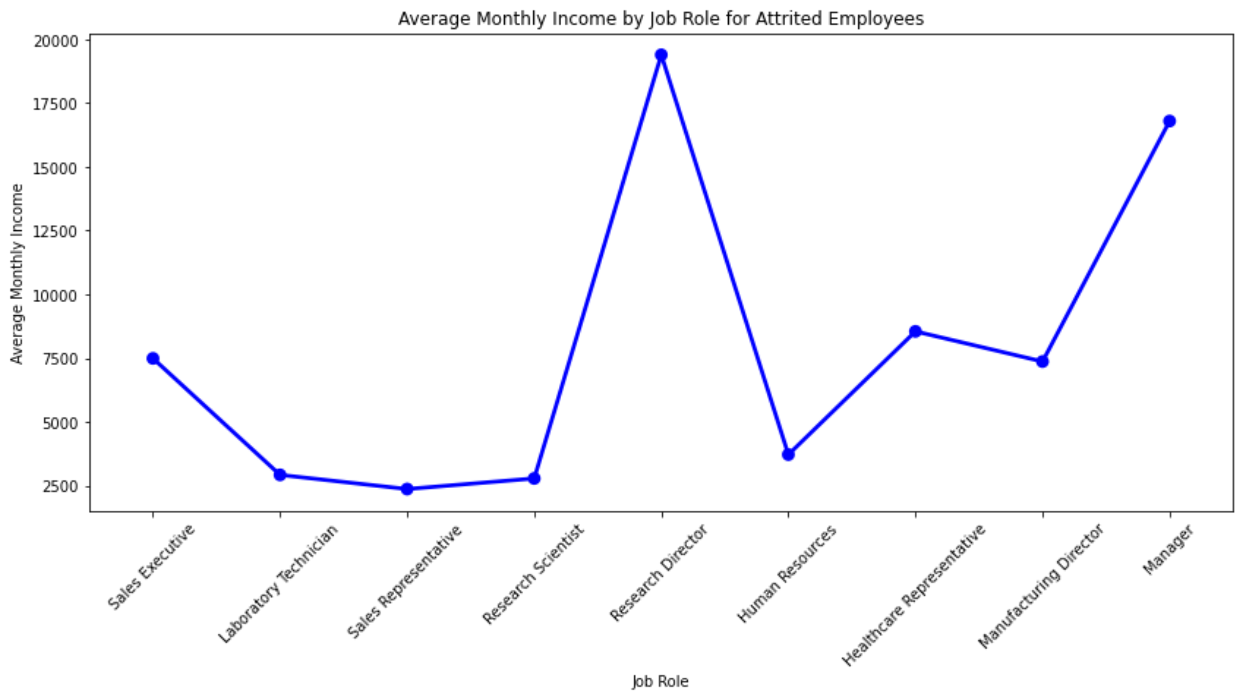
In [39]:

```
df['Attrition'].value_counts() # Count of Unique Values in the Attriti
```

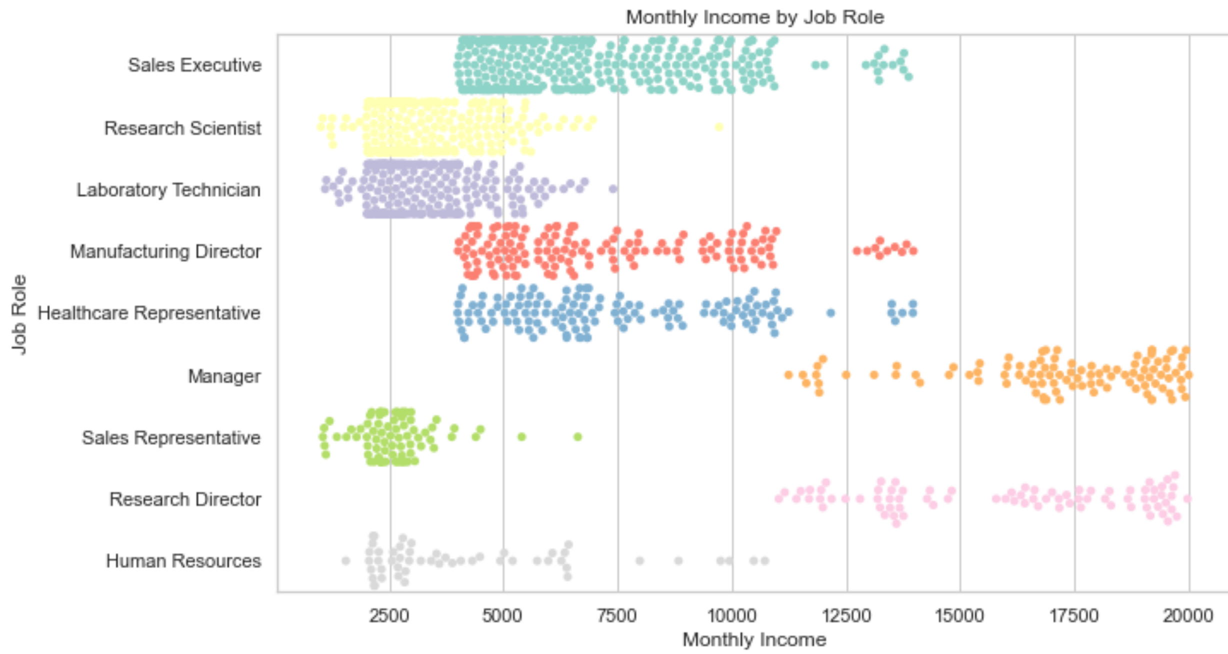
Out[39]:

```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

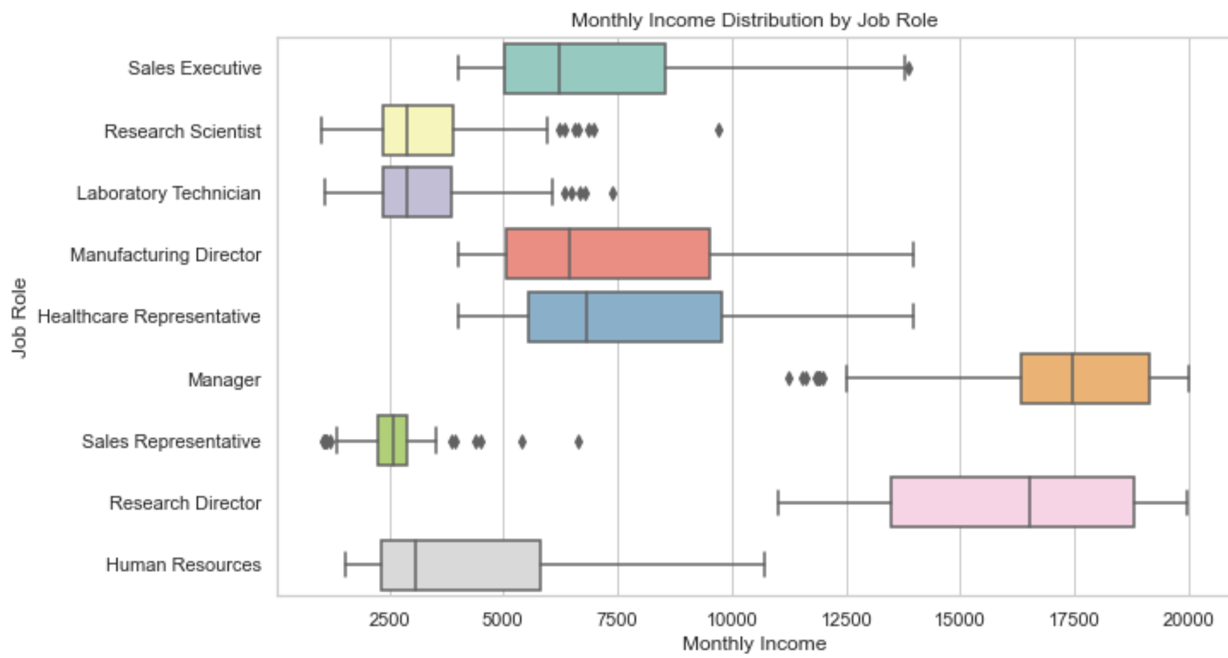
In [40]:
```
# Filter the data for attrited employees only
attrited_df = df[df['Attrition'] == 'Yes']

# Example: Point plot for Rounded Monthly Income by Job Role for attri
plt.figure(figsize=(14, 6))

sns.pointplot(data=attrited_df, x='JobRole', y='MonthlyIncome', ci=Non
plt.title('Average Monthly Income by Job Role for Attrited Employees')
plt.xlabel('Job Role')
plt.ylabel('Average Monthly Income')
plt.xticks(rotation=45)
plt.show()
```



In [52]:
```
plt.figure(figsize=(10, 6))
sns.swarmplot(x='MonthlyIncome', y='JobRole', data=df, palette='Set3')
plt.title('Monthly Income by Job Role')
plt.xlabel('Monthly Income')
plt.ylabel('Job Role')
plt.show()
```

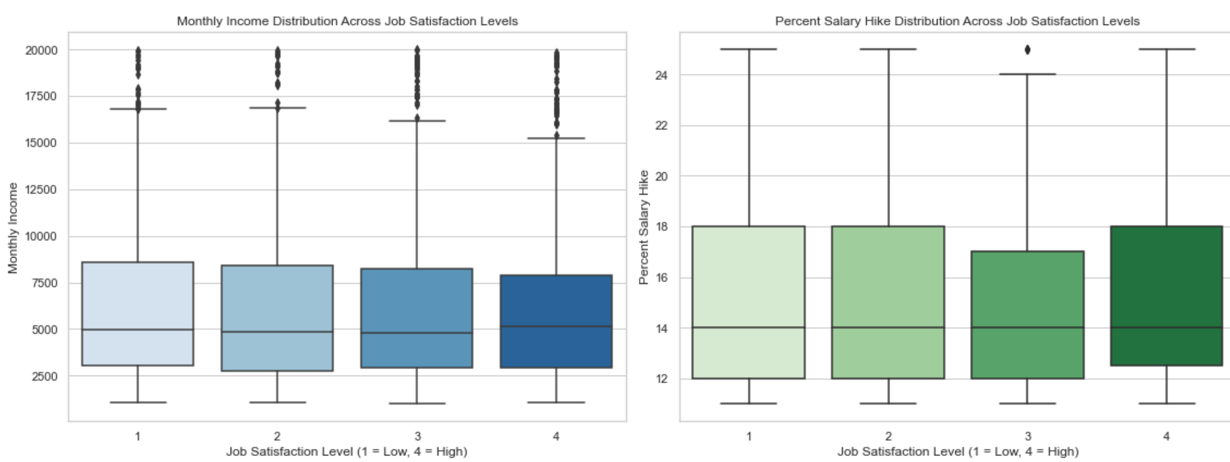


In [53]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='MonthlyIncome', y='JobRole', data=df, palette='Set3')
plt.title('Monthly Income Distribution by Job Role')
plt.xlabel('Monthly Income')
plt.ylabel('Job Role')
plt.show()
```



In [45]:

```
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Create a figure for the box plots
plt.figure(figsize=(16, 6))

# Box Plot for Monthly Income
plt.subplot(1, 2, 1)
sns.boxplot(x='JobSatisfaction', y='MonthlyIncome', data=df, palette='
plt.title('Monthly Income Distribution Across Job Satisfaction Levels'
plt.xlabel('Job Satisfaction Level (1 = Low, 4 = High)')
plt.ylabel('Monthly Income')

# Box Plot for Percent Salary Hike
plt.subplot(1, 2, 2)
sns.boxplot(x='JobSatisfaction', y='PercentSalaryHike', data=df, palet
plt.title('Percent Salary Hike Distribution Across Job Satisfaction Le
plt.xlabel('Job Satisfaction Level (1 = Low, 4 = High)')
plt.ylabel('Percent Salary Hike')

plt.tight_layout()
plt.show()
```

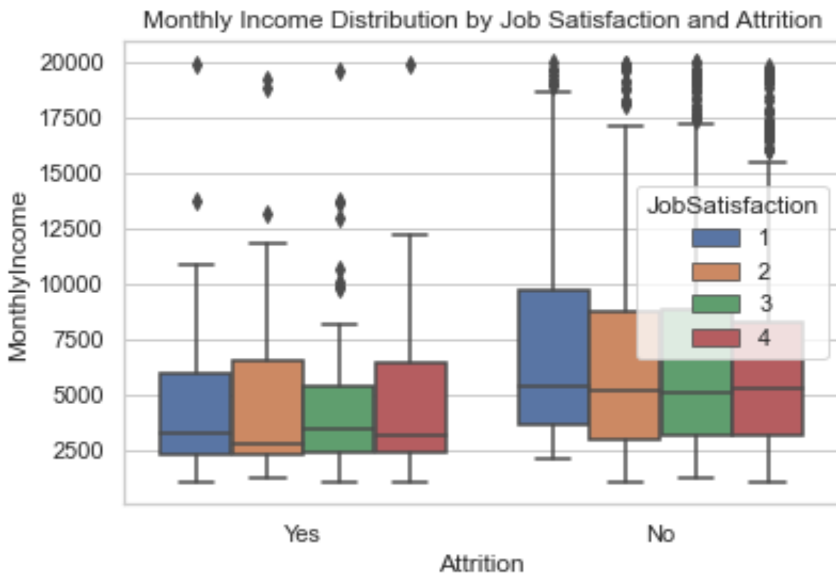


In [76]:
```
# Cross-tabulation between JobSatisfaction, StockOptionLevel, and Attr
pd.crosstab([df['JobSatisfaction'], df['StockOptionLevel']], df['Attri
```

Out[76]:

JobSatisfaction	StockOptionLevel	No	Yes	All
1	0	77	46	123
	1	107	15	122
	2	24	2	26
	3	15	3	18
2	0	87	33	120
	1	107	6	113
	2	28	2	30
	3	12	5	17
3	0	151	46	197
	1	157	18	175
	2	42	6	48
	3	19	3	22
4	0	162	29	191
	1	169	17	186
	2	52	2	54
	3	24	4	28
All		1233	237	1470

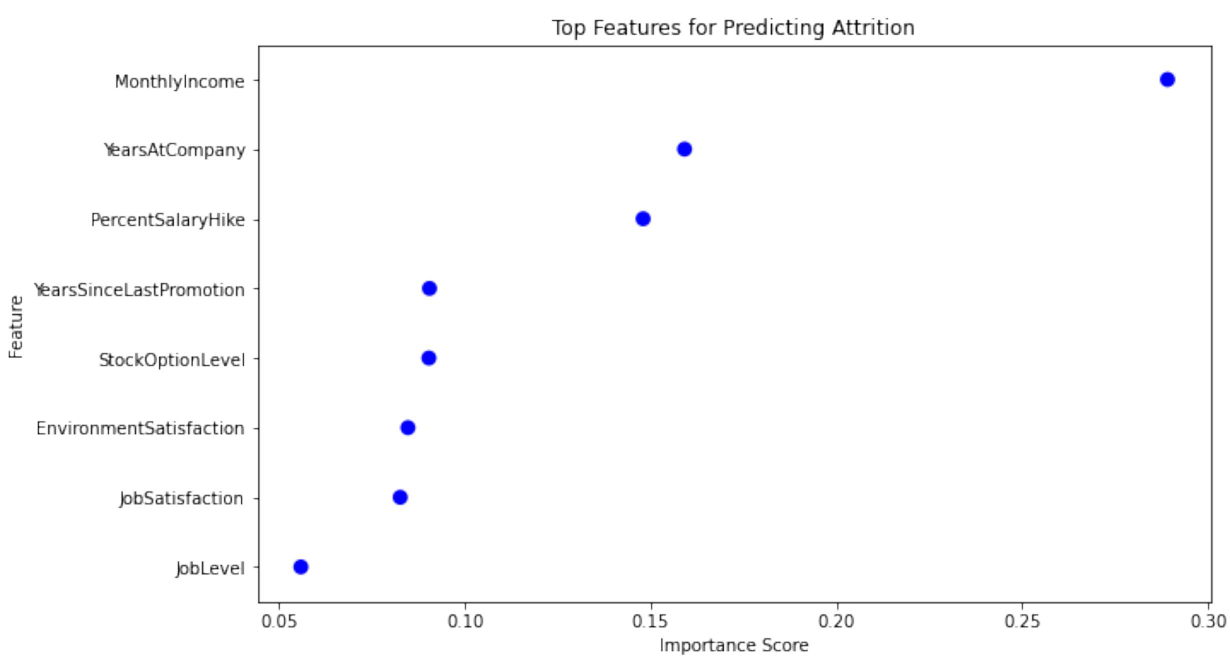
In [77]:
```
# Visualize JobSatisfaction vs. MonthlyIncome for those who left vs. s
sns.boxplot(x='Attrition', y='MonthlyIncome', hue='JobSatisfaction', d
plt.title('Monthly Income Distribution by Job Satisfaction and Attriti
plt.show()
```



Modeling

In [43]:

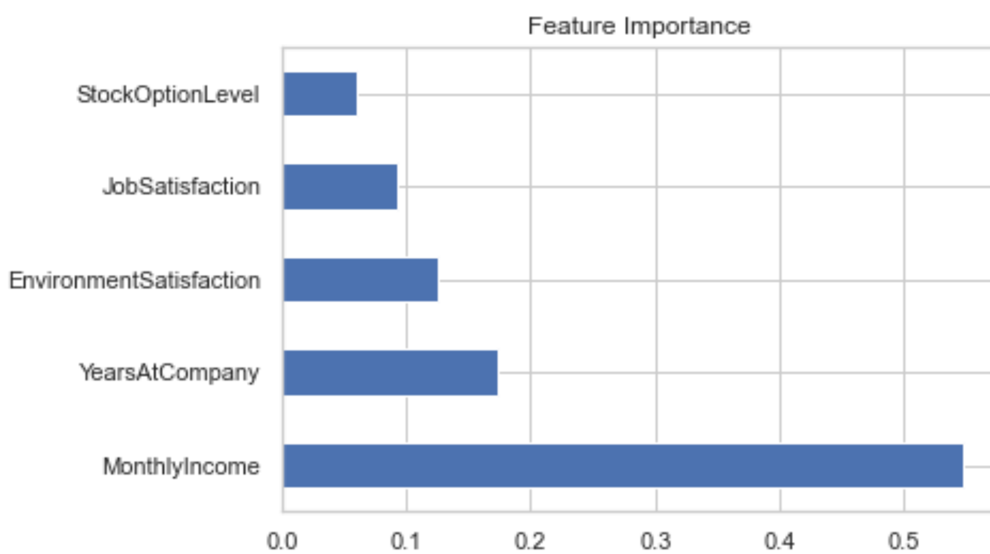
```
# Encode categorical variables
label_encoder = LabelEncoder()
df_encoded = df.copy()
for column in df_encoded.select_dtypes(include=['object']).columns:
    df_encoded[column] = label_encoder.fit_transform(df_encoded[column

# Define the features and target variable
features = ['JobLevel', 'MonthlyIncome', 'PercentSalaryHike', 'StockOp
            'YearsAtCompany', 'YearsSinceLastPromotion',
            'EnvironmentSatisfaction', 'JobSatisfaction']
X = df_encoded[features]
y = df_encoded['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.

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

# Get feature importances
importance = model.feature_importances_
feature_importance = pd.DataFrame({'Feature': features, 'Importance':
feature_importance = feature_importance.sort_values(by='Importance', a

# Plot the feature importance as a point plot
plt.figure(figsize=(10, 6))
sns.pointplot(x='Importance', y='Feature', data=feature_importance, co
plt.title('Top Features for Predicting Attrition')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()
```



In [78]:

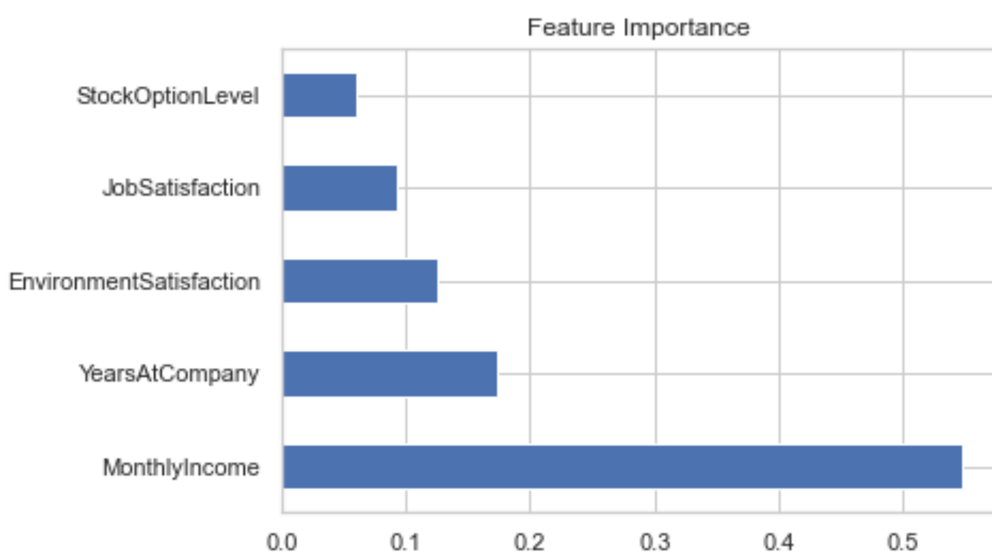
```
# Prepare data
X = df[['YearsAtCompany', 'MonthlyIncome', 'JobSatisfaction', 'StockOp
y = df['Attrition'].map({'Yes': 1, 'No': 0})  # Convert to binary

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Train a decision tree
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

# Feature importance
feature_importance = pd.Series(clf.feature_importances_, index=X.colum
feature_importance.plot(kind='barh', title='Feature Importance')
plt.show()
```

	precision	recall	f1-score	support
0	0.88	0.86	0.87	255
1	0.20	0.23	0.21	39
accuracy			0.78	294
macro avg	0.54	0.54	0.54	294
weighted avg	0.79	0.78	0.78	294



In [79]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Prepare data
X = df[['YearsAtCompany', 'MonthlyIncome', 'JobSatisfaction', 'StockOp
y = df['Attrition'].map({'Yes': 1, 'No': 0})  # Convert to binary

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Train a decision tree
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

# Feature importance
feature_importance = pd.Series(clf.feature_importances_, index=X.colum
feature_importance.plot(kind='barh', title='Feature Importance')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       255
           1       0.20      0.23      0.21        39

    accuracy                           0.78       294
   macro avg       0.54      0.54      0.54       294
weighted avg       0.79      0.78      0.78       294
```

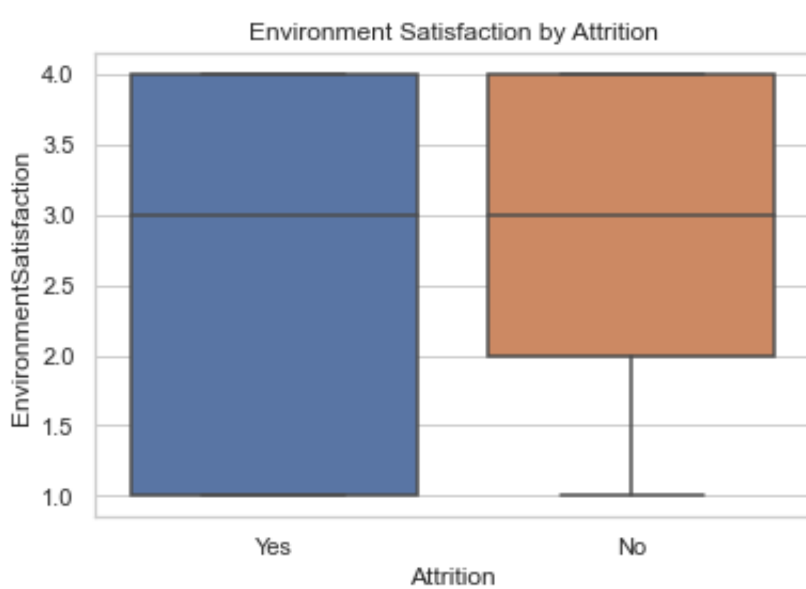


In [80]:
```
# Segment the data by Attrition and compare features
stayers = df[df['Attrition'] == 'No']
leavers = df[df['Attrition'] == 'Yes']

# Compare mean values for relevant features
comparison = pd.DataFrame({
    'Feature': ['JobSatisfaction', 'MonthlyIncome', 'StockOptionLevel'
    'Stayers': stayers[['JobSatisfaction', 'MonthlyIncome', 'StockOpti
    'Leavers': leavers[['JobSatisfaction', 'MonthlyIncome', 'StockOpti
})

print(comparison)
```

```
                            Feature      Stayers      Leavers
JobSatisfaction     JobSatisfaction     2.778589     2.468354
MonthlyIncome         MonthlyIncome  6832.739659  4787.092827
StockOptionLevel   StockOptionLevel     0.845093     0.527426
```

In [81]:
```
# Group by Attrition and calculate mean EnvironmentSatisfaction
env_satisfaction_mean = df.groupby('Attrition')['EnvironmentSatisfacti
print(env_satisfaction_mean)

# Visualize the distribution with a boxplot
sns.boxplot(x='Attrition', y='EnvironmentSatisfaction', data=df)
plt.title('Environment Satisfaction by Attrition')
plt.show()
```

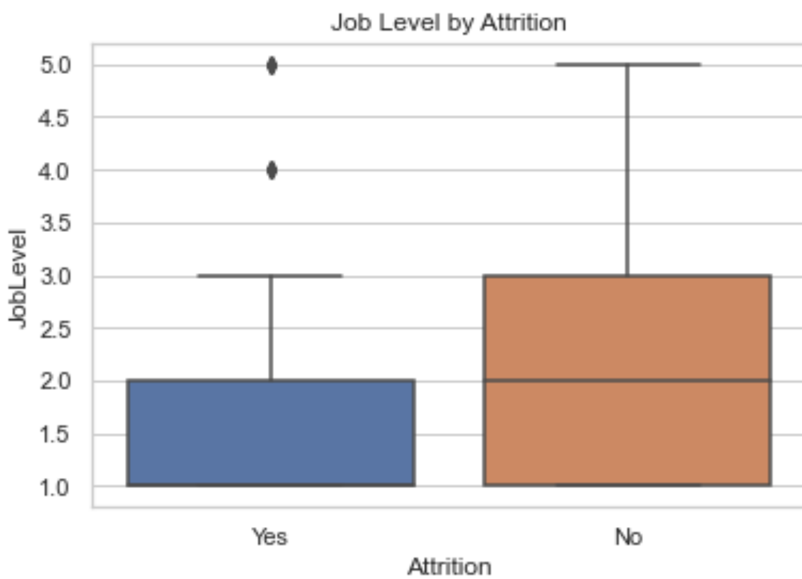
```
Attrition
No     2.771290
Yes    2.464135
Name: EnvironmentSatisfaction, dtype: float64
```



In [82]:

```python
# Group by Attrition and calculate mean JobLevel
job_level_mean = df.groupby('Attrition')['JobLevel'].mean()
print(job_level_mean)

# Visualize the distribution with a boxplot
sns.boxplot(x='Attrition', y='JobLevel', data=df)
plt.title('Job Level by Attrition')
plt.show()

# Cross-tabulation between JobLevel and Attrition
job_level_crosstab = pd.crosstab(df['JobLevel'], df['Attrition'], marg
print(job_level_crosstab)
```

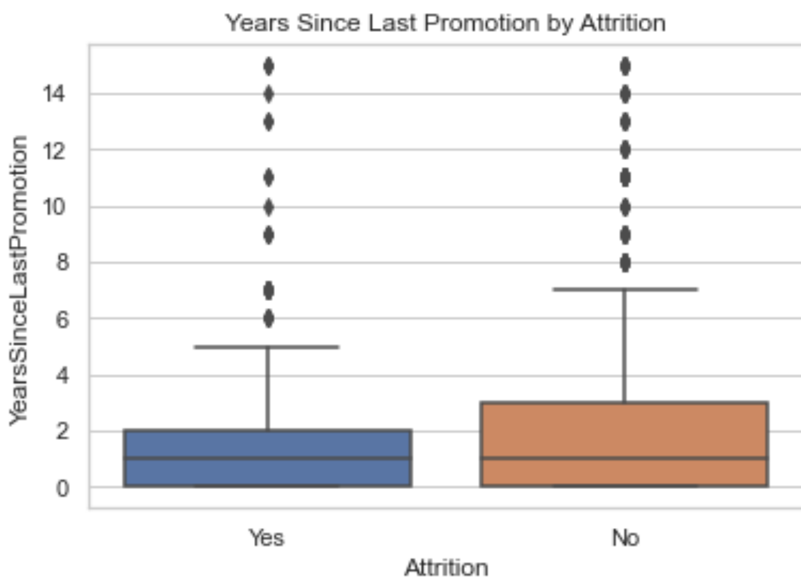
```
Attrition
No     2.145985
Yes    1.637131
Name: JobLevel, dtype: float64
```



Attrition	No	Yes	All
JobLevel			
1	400	143	543
2	482	52	534
3	186	32	218
4	101	5	106
5	64	5	69
All	1233	237	1470

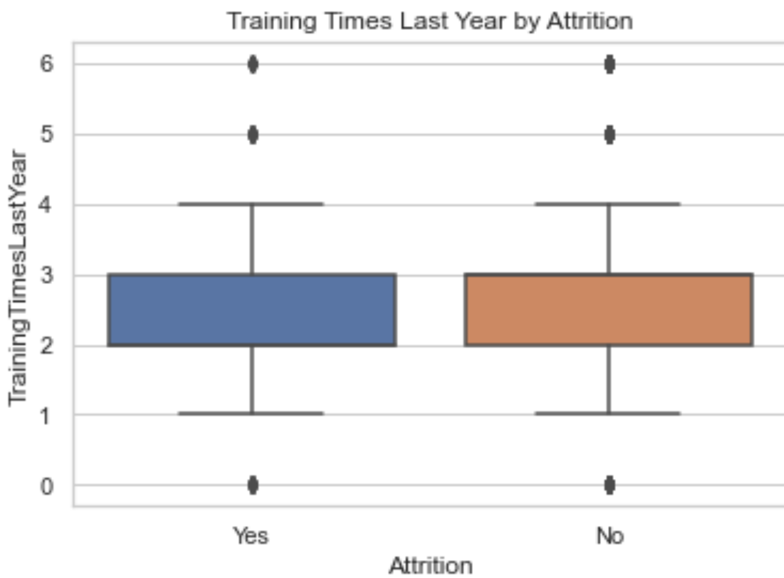
In [83]:
```python
# Group by Attrition and calculate mean YearsSinceLastPromotion
promotion_mean = df.groupby('Attrition')['YearsSinceLastPromotion'].me
print(promotion_mean)

# Visualize the distribution with a boxplot
sns.boxplot(x='Attrition', y='YearsSinceLastPromotion', data=df)
plt.title('Years Since Last Promotion by Attrition')
plt.show()
```

```
Attrition
No     2.234388
Yes    1.945148
Name: YearsSinceLastPromotion, dtype: float64
```



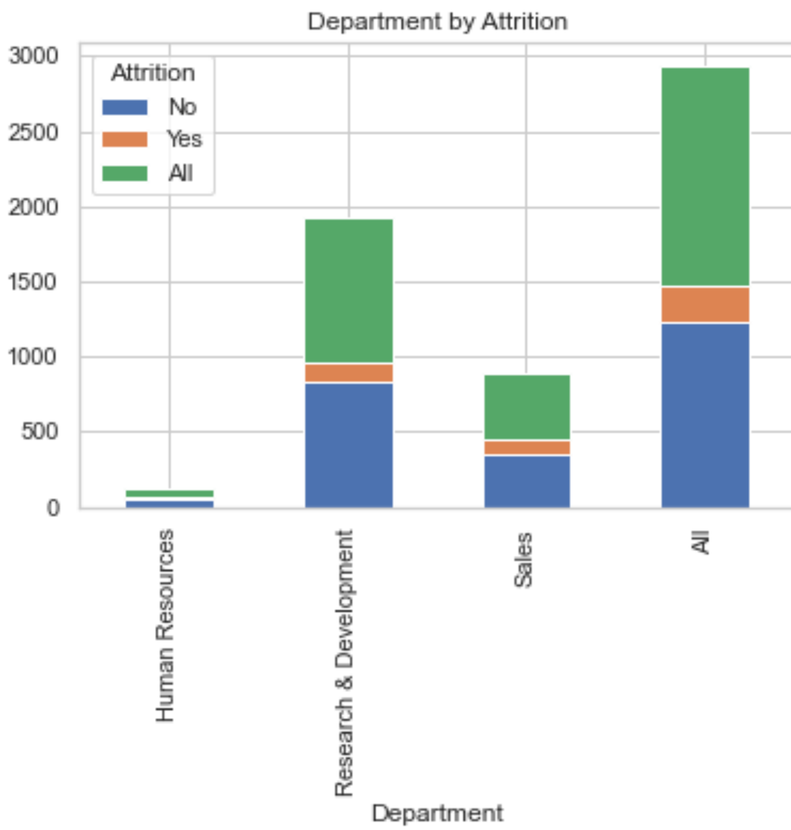
In [84]:
```
# Group by Attrition and calculate mean TrainingTimesLastYear
training_mean = df.groupby('Attrition')['TrainingTimesLastYear'].mean(
print(training_mean)

# Visualize the distribution with a boxplot
sns.boxplot(x='Attrition', y='TrainingTimesLastYear', data=df)
plt.title('Training Times Last Year by Attrition')
plt.show()
```

```
Attrition
No     2.832928
Yes    2.624473
Name: TrainingTimesLastYear, dtype: float64
```

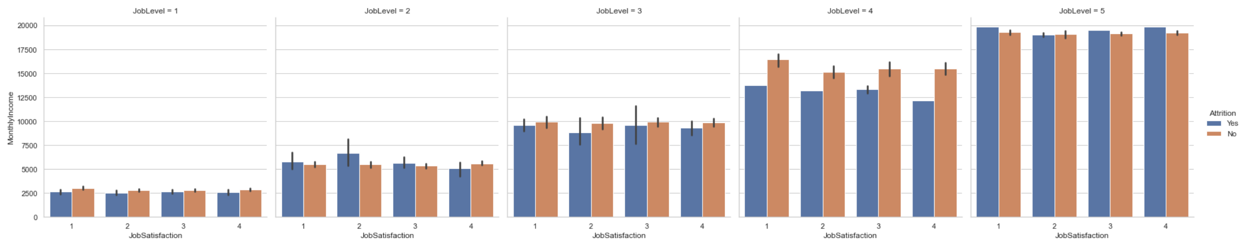


In [85]:
```
# Cross-tabulation between Department and Attrition
department_crosstab = pd.crosstab(df['Department'], df['Attrition'], m
print(department_crosstab)

# Visualize with a bar plot
department_crosstab.plot(kind='bar', stacked=True)
plt.title('Department by Attrition')
plt.show()
```

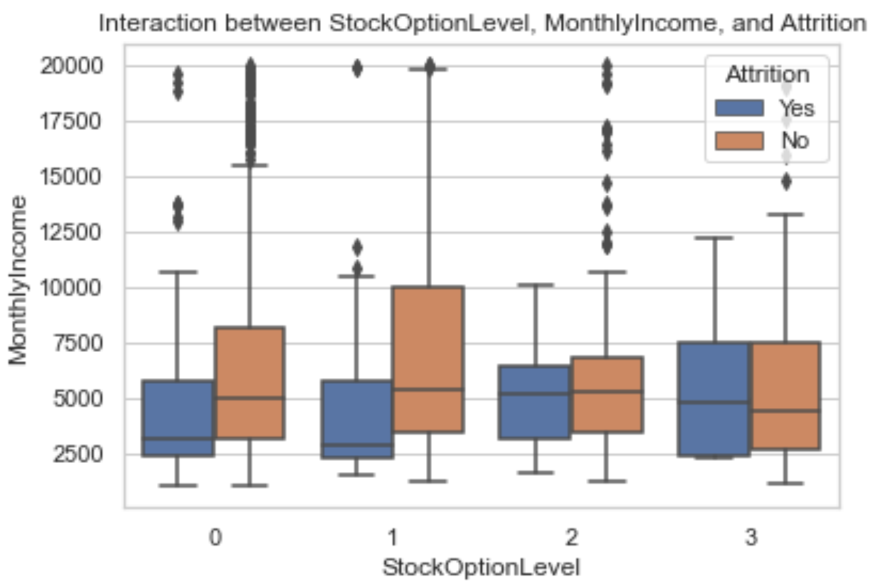
```
Attrition                 No  Yes   All
Department
Human Resources           51   12    63
Research & Development   828  133   961
Sales                    354   92   446
All                     1233  237  1470
```



In [86]:
```python
# Example: Interaction between JobSatisfaction and JobLevel, segmentec
sns.catplot(x='JobSatisfaction', y='MonthlyIncome', hue='Attrition', c
                data=df, kind='bar')
plt.show()
```

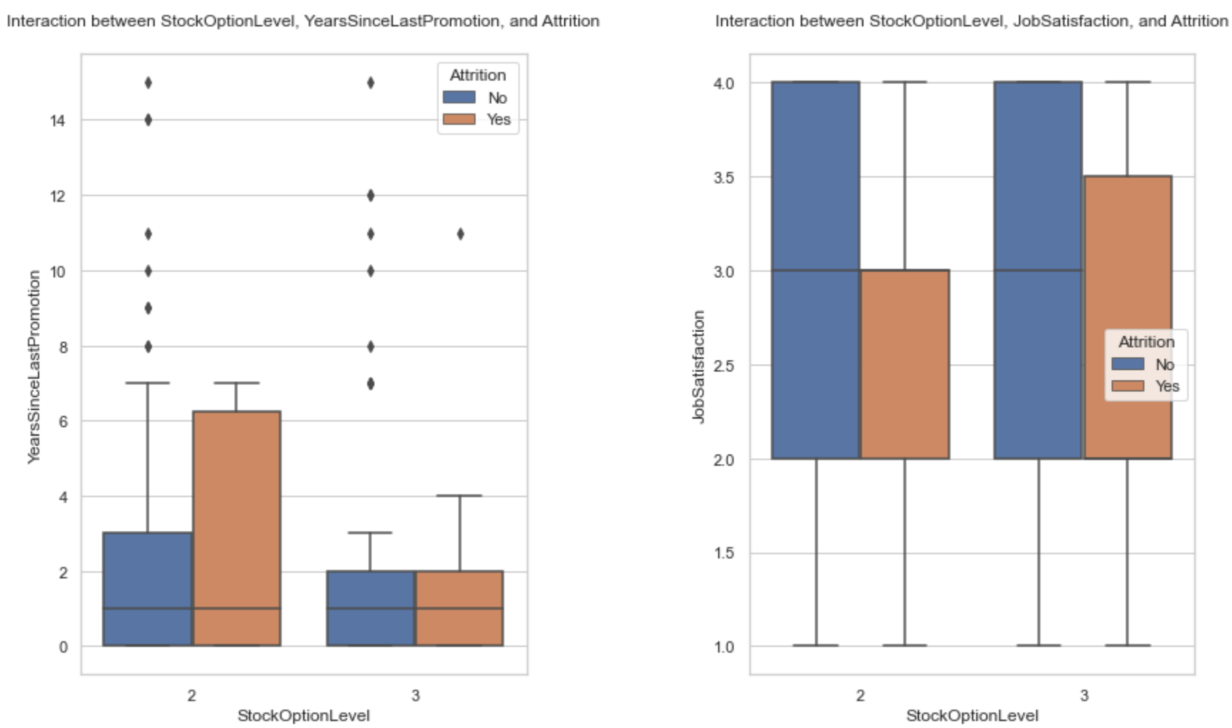


In [87]:
```python
# Example: Interaction between JobSatisfaction and StockOptionLevel, s
sns.boxplot(x='StockOptionLevel', y='MonthlyIncome', hue='Attrition',
plt.title('Interaction between StockOptionLevel, MonthlyIncome, and At
plt.show()
```



In [88]:

```python
# Filter the data for StockOptionLevel 2 & 3
filtered_df = df[df['StockOptionLevel'].isin([2, 3])]

# Set the features of interest
features = ['YearsSinceLastPromotion', 'JobSatisfaction']

# Create box plots for each feature
plt.figure(figsize=(14, 8))  # Increase figure size
for i, feature in enumerate(features, 1):
    ax = plt.subplot(1, 2, i)
    sns.boxplot(
        data=filtered_df,
        x='StockOptionLevel',
        y=feature,
        hue='Attrition',
        ax=ax
    )
    ax.set_title(f'Interaction between StockOptionLevel, {feature}, ar

plt.subplots_adjust(wspace=0.5, hspace=0.5)  # Adjust space between pl
plt.show()
```



In [ ]:

In [ ]:

In [56]:
```
# Creating Subset DF Features
df = pd.read_csv('/Users/tgisakia/Desktop/EmployeeAttrition.csv')
df_features = df[['JobLevel','MonthlyIncome','PercentSalaryHike','Stoc
df_features
```

Out[56]:

	JobLevel	MonthlyIncome	PercentSalaryHike	StockOptionLevel	TotalWorkingYears	Years
0	2	5993	11	0	8	
1	2	5130	23	1	10	
2	1	2090	15	0	7	
3	1	2909	11	0	8	
4	1	3468	12	1	6	
...	...	...	...	...	...	
1465	2	2571	17	1	17	
1466	3	9991	15	1	9	
1467	2	6142	20	1	6	
1468	2	5390	14	0	17	
1469	2	4404	12	0	6	

1470 rows × 9 columns

In [57]:
```
# Select the Features for the Model
df_features = ['JobLevel', 'MonthlyIncome', 'PercentSalaryHike', 'Stoc
               'TotalWorkingYears', 'YearsAtCompany', 'YearsSinceLastP
               'EnvironmentSatisfaction', 'JobSatisfaction']

# Create the Feature Matrix X Using the Selected Features
X = df[df_features]

# Create the Target Vector y Using the 'Attrition' Column
y = df['Attrition']
```

In [58]:

```python
# Create the Feature Matrix X Using the Selected Features
X = df[df_features]

# Create the Target Vector y Using the 'Attrition' Column
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.
```

In [59]:

```python
# Initialize the SMOTE (Synthetic Minority Over-sampling Technique) Ob
smote = SMOTE(random_state=42)

# Apply SMOTE to the Training Data to Balance the Classes
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [60]:

```python
# Initialize the StandardScaler for Feature Scaling
scaler = StandardScaler()

# Fit the Scaler on the SMOTE-Resampled Training Data and Transform It
X_train_smote_scaled = scaler.fit_transform(X_train_smote)

# Transform the Testing Data Using the Same Scaler
X_test_scaled = scaler.transform(X_test)
```

In [61]:
```python
# Initialize the Logistic Regression Model with a Specified Random Sta
logreg = LogisticRegression(random_state=42)

# Fit the Logistic Regression Model to the SMOTE-Resampled and Scaled
logreg.fit(X_train_smote_scaled, y_train_smote)
```

Out[61]:
```
▾ LogisticRegression
LogisticRegression(random_state=42)
```

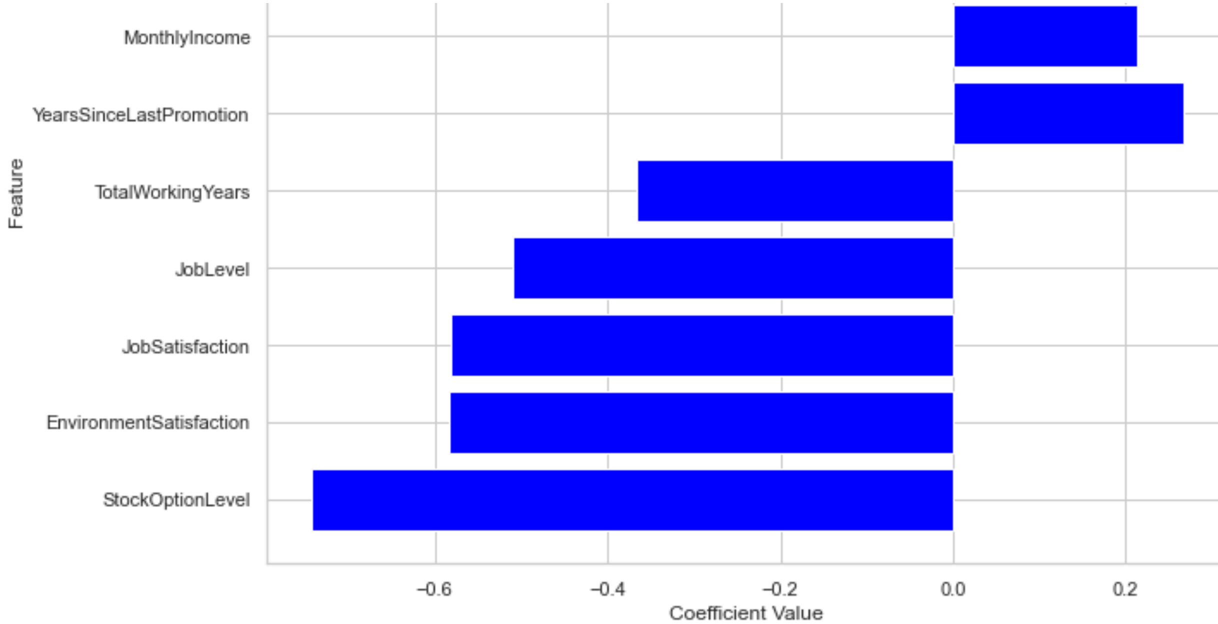
In [62]:
```python
# Make Predictions on the Scaled Testing Data Using the Trained Logist
y_pred = logreg.predict(X_test_scaled)

# Print the Classification Report to Evaluate the Model's Performance
print(classification_report(y_test, y_pred))

# Calculate and Print the AUC-ROC Score to Assess the Model's Discrimi
print("AUC-ROC:", roc_auc_score(y_test, logreg.predict_proba(X_test_sc
```

```
              precision    recall  f1-score   support

          No       0.93      0.71      0.81       255
         Yes       0.26      0.64      0.36        39

    accuracy                           0.70       294
   macro avg       0.59      0.68      0.59       294
weighted avg       0.84      0.70      0.75       294

AUC-ROC: 0.7098039215686275
```

In [65]:
```python
```

```python
# Create a DataFrame to Store the Features and Their Corresponding Coe
coefficients = pd.DataFrame({
    'Feature': df_features,
    'Coefficient': logreg.coef_[0]
}).sort_values(by='Coefficient', ascending=False)

# Print the Coefficients DataFrame Sorted by the Coefficient Values ir
print(coefficients)

# Calculate the Absolute Value of the Coefficients for Sorting Purpose
coefficients['abs_coefficient'] = coefficients['Coefficient'].abs()

# Sort the DataFrame by the Absolute Value of the Coefficients in Desc
coefficients = coefficients.sort_values(by='abs_coefficient', ascendir

# Set the Size of the Figure for the Bar Plot
plt.figure(figsize=(10, 8))

# Create a Horizontal Bar Plot of the Logistic Regression Coefficients
plt.barh(coefficients['Feature'], coefficients['Coefficient'], color='

# Label the X-Axis as Coefficient Value
plt.xlabel('Coefficient Value')

# Label the Y-Axis as Feature
plt.ylabel('Feature')

# Set the Title of the Plot
plt.title('Logistic Regression Coefficients')

# Display the Plot
plt.show()
```

```
                    Feature  Coefficient
6  YearsSinceLastPromotion     0.266410
1            MonthlyIncome     0.212718
5           YearsAtCompany    -0.207444

2        PercentSalaryHike    -0.208557
4        TotalWorkingYears    -0.367091
0                 JobLevel    -0.510052
8          JobSatisfaction    -0.582410
7  EnvironmentSatisfaction    -0.582701
3         StockOptionLevel    -0.743618
```



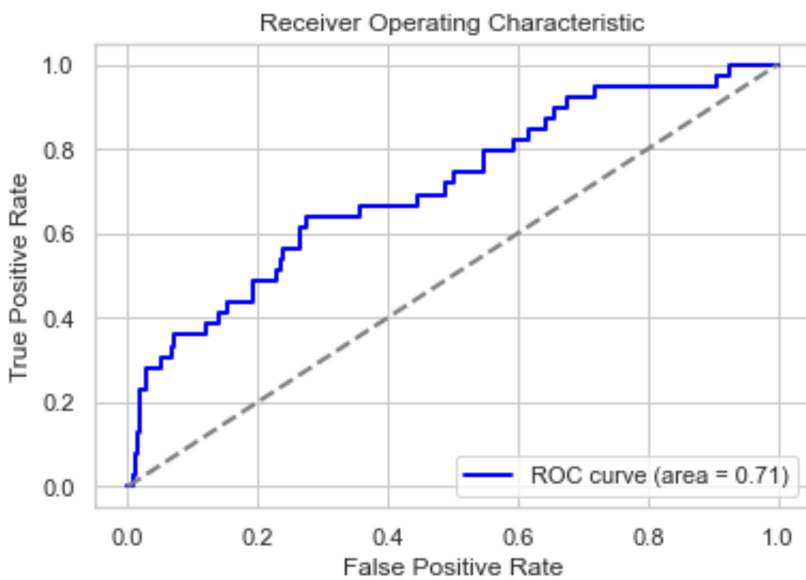
Model Evaluation

In [66]:
```python
# Convert 'Yes' to 1 and 'No' to 0 in y_test
y_test_binary = y_test.map({'No': 0, 'Yes': 1})
print(y_test_binary.unique())
# # Now calculate the ROC curve
y_prob = logreg.predict_proba(X_test_scaled)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test_binary, y_prob)
roc_auc = auc(fpr, tpr)
```

```
[0 1]
```

In [67]:
```python
# Calculate the ROC curve and specify the positive label
y_prob = logreg.predict_proba(X_test_scaled)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob, pos_label='Yes')
roc_auc = auc(fpr, tpr)
```

In [68]:
```
# Calculate the ROC curve using the binary y_test
y_prob = logreg.predict_proba(X_test_scaled)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test_binary, y_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve if needed
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



In [ ]:
```
Employees who haven’t been promoted recently, earn more, or have lower
slightly more likely to leave. Strong retention factors: stock options
    and a higher job level or seniority.
```

In [69]:
```
# Create a Pipeline with SMOTE, StandardScaler, and RandomForestClassi
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Define the Parameter Grid for GridSearchCV
param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [5, 10, 15]
}

# Initialize GridSearchCV with the Pipeline, Parameter Grid, and AUC-R
grid_search = GridSearchCV(pipeline, param_grid, scoring='roc_auc', cv

# Fit the GridSearchCV on the Training Data
grid_search.fit(X_train, y_train)

# Print the Best Parameters Found by GridSearchCV
print("Best parameters found: ", grid_search.best_params_)

# Print the Best AUC-ROC Score Achieved
print("Best AUC score: ", grid_search.best_score_)
```

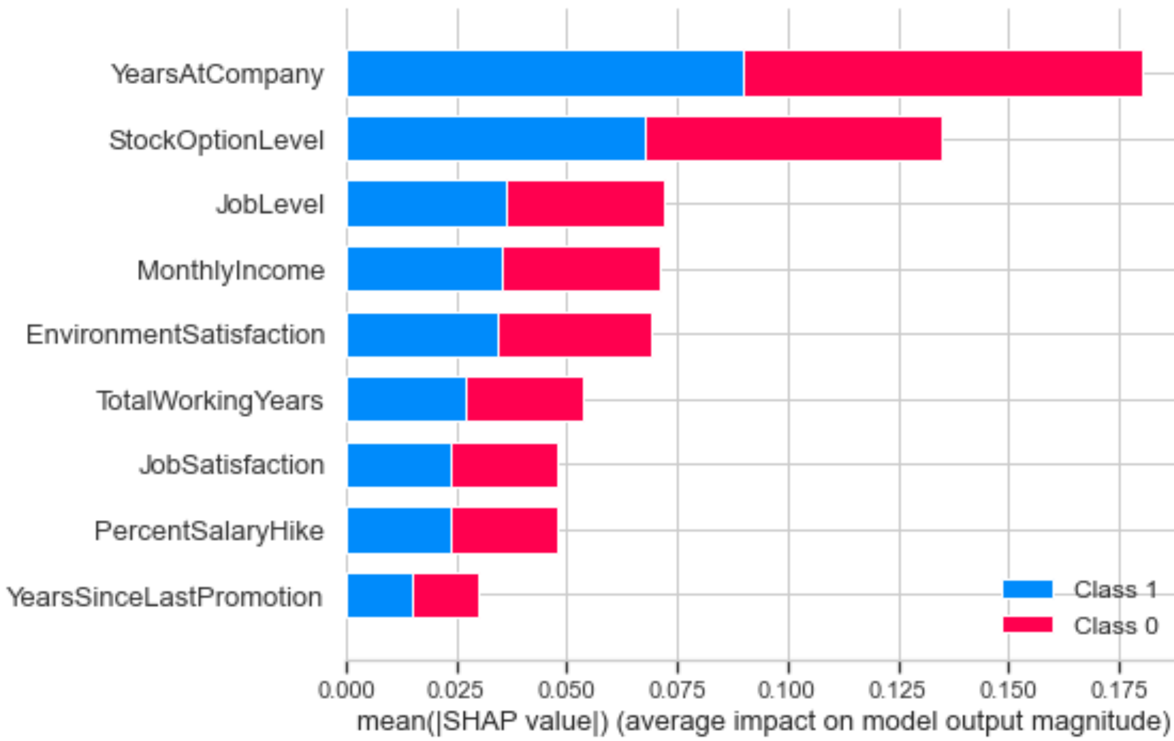
```
Best parameters found:  {'classifier__max_depth': 5, 'classifier__n_e
stimators': 200}
Best AUC score:  0.73790515436944
```

In [70]:
```
# Extract the Best Model from GridSearchCV
best_model = grid_search.best_estimator_

# Make Predictions on the Test Data Using the Best Model
y_pred = best_model.predict(X_test)

# Print the Classification Report to Evaluate the Model's Performance
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          No       0.92      0.79      0.85       255
         Yes       0.28      0.54      0.37        39

    accuracy                           0.76       294
   macro avg       0.60      0.66      0.61       294
weighted avg       0.83      0.76      0.78       294
```

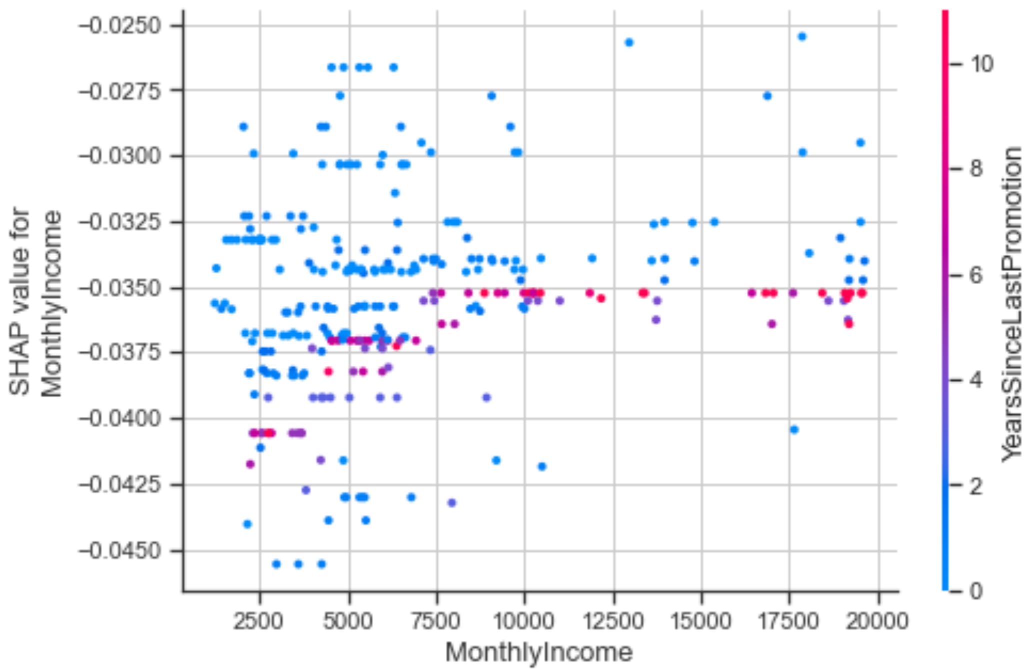
Model Interpretation

In [71]:
```
# Import the SHAP Library for Model Interpretation
import shap

# Initialize a SHAP TreeExplainer for the Best Model's Classifier
explainer = shap.TreeExplainer(best_model.named_steps['classifier'])

# Calculate the SHAP Values for the Test Data
shap_values = explainer.shap_values(X_test)

# Create a SHAP Summary Plot to Visualize the Impact of Features on th
shap.summary_plot(shap_values, X_test)
```



In [ ]:
```
#Notes
In order to reduce attrition: improving on aspects related to YearsAtC
Employees with high YearsAtCompany might be more likely to leave
```

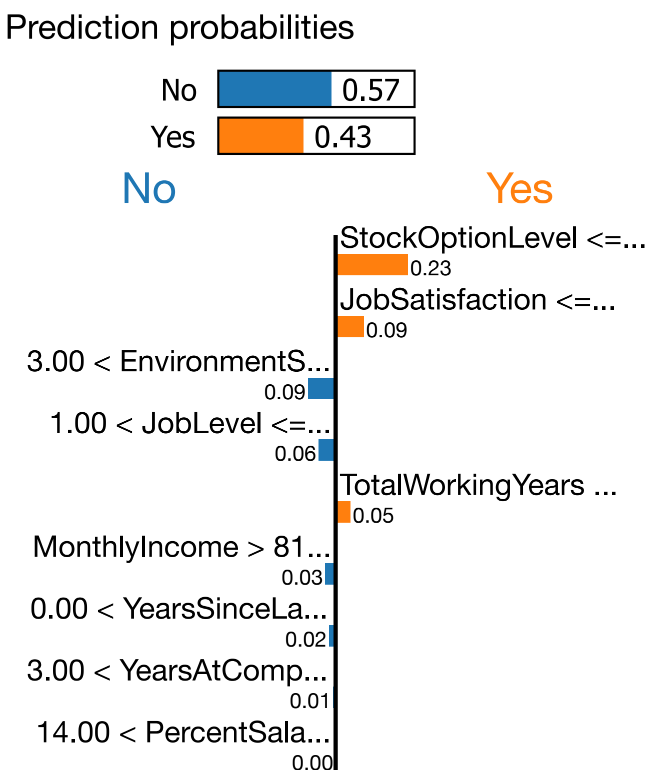
In [72]:
```
# Create a SHAP Dependence Plot for the 'MonthlyIncome' Feature
shap.dependence_plot("MonthlyIncome", shap_values[1], X_test, feature_
```



In [74]:
```
# Import the LIME Library for Model Interpretation
import lime
import lime.lime_tabular

# Initialize a LIME Tabular Explainer with the Training Data
explainer = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train.values,  # Use the training data for LIME's
    feature_names=df_features,     # Set the feature names for interpr
    class_names=['No', 'Yes'],     # Define the class names (No Attrit
    mode='classification'          # Set the mode to classification
)
```

In [75]:

```
# Explain a single prediction
i = 0  # Index of the test sample to explain
exp = explainer.explain_instance(X_test.iloc[i], best_model.predict_pr
exp.show_in_notebook(show_table=True)
```



Feature	Value
StockOptionLevel	0.00
JobSatisfaction	1.00
EnvironmentSatisfaction	4.00
JobLevel	2.00
TotalWorkingYears	6.00
MonthlyIncome	8463.00
YearsSinceLastPromotion	1.00
YearsAtCompany	5.00
PercentSalaryHike	18.00

Conclusion:

In this project, we set out to understand why some employees leave the company by looking at various factors that might influence their decision. We used data analysis and machine learning techniques to identify the key reasons behind employee attrition and to predict which employees are most at risk of leaving.

Key Insights:

Top Reasons for Leaving:

1. Job and Environment Satisfaction: Employees who are less satisfied with their jobs and work environment are much more likely to leave the company. Improving how employees feel about their jobs could significantly reduce turnover.
2. Lack of Promotion: Employees who haven't been promoted in a while are more likely to leave. This suggests that career growth opportunities are crucial for employee retention.
3. Pay, Stock Options, and Job Level: Employees with good monthly income, stock options, and recent promotions or salary hikes tend to have higher job and environmental satisfaction, which significantly lowers the likelihood of them leaving the company. On the other hand, lower-paid employees, especially those in lower job positions, are at a higher risk of leaving.

How Well the Model Predicted Attrition:

1. Random Forest Model: The model we used was effective at predicting which employees are likely to leave. It balanced accuracy and the ability to catch true positives well, meaning it was good at identifying those at risk.
2. Why Certain Factors Matter: Using a tool called SHAP, we confirmed that job satisfaction, career growth opportunities, and compensation factors like stock options and salary hikes are the most influential in predicting whether an employee will leave.

Trends Among Job Roles and Salaries:

1. Certain job roles, like Sales Representatives and those in Human Resources, have higher attrition rates, especially if their salaries are lower compared to others in similar positions. This indicates that focusing on these roles could help reduce overall turnover.

What This Means and What to Do Next:

1. Boost Job Satisfaction: Since job satisfaction is a key factor in whether employees stay or leave, the company should focus on making work more fulfilling and enjoyable. Regular feedback and addressing issues quickly could help keep employees happy and engaged.
2. Promote Career Growth: Employees who feel stuck in their current roles are more likely to leave. Providing clear paths for promotion and ensuring employees are recognized and rewarded for their contributions could help retain them.
3. Reevaluate Pay and Stock Options: For roles with high turnover, especially where pay is lower, consider revisiting the compensation packages, including stock options. Ensuring that employees feel they are fairly compensated for their work, with opportunities for salary hikes and promotions, is crucial for reducing attrition.

Focus on High-Risk Roles:

1. Targeted efforts to retain employees in high-risk roles like Sales Representatives could make a big difference. Tailored programs that address the specific needs of these groups could help lower attrition rates.

Final Thoughts:

This analysis has helped identify the main reasons why employees leave and provided actionable steps the company can take to reduce turnover. By improving job satisfaction, offering better career growth opportunities, and ensuring competitive pay and stock options, the company can keep more of its valuable employees. In the future, it might be worth exploring other factors, like employee engagement or economic conditions, to further refine these insights and improve the model's predictions.

In [ ]:
```

```