HR ANALYTICS OUTLINE

About Dataset: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset (https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset)

Importing Libraries.

In [64]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score
sns.set_context("notebook")

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")

pd.set_option('display.max_columns', None)

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, classification_r
eport, roc_auc_score, precision_recall_curve, roc_curve
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
```

Loading Dataset

In [65]:
```
employee_data = pd.read_csv('/content/WA_Fn-UseC_-HR-Employee-Attrition.csv')
```

In [66]:
```
employee_data.head(5)
```

Out[66]:

	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 [67]:
```
employee_data.tail(5)
```

Out[67]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Ec
1465	36	No	Travel_Frequently	884	Research & Development	23	2	
1466	39	No	Travel_Rarely	613	Research & Development	6	1	
1467	27	No	Travel_Rarely	155	Research & Development	4	3	
1468	49	No	Travel_Frequently	1023	Sales	2	3	
1469	34	No	Travel_Rarely	628	Research & Development	8	3	

Summary Statistics of numeric variables:

In [68]:
```
employee_data.describe().transpose().round(2)
```

Out[68]:

	count	mean	std	min	25%	50%	75%	max
Age	1470.0	36.92	9.14	18.0	30.00	36.0	43.00	60.0
DailyRate	1470.0	802.49	403.51	102.0	465.00	802.0	1157.00	1499.0
DistanceFromHome	1470.0	9.19	8.11	1.0	2.00	7.0	14.00	29.0
Education	1470.0	2.91	1.02	1.0	2.00	3.0	4.00	5.0
EmployeeCount	1470.0	1.00	0.00	1.0	1.00	1.0	1.00	1.0
EmployeeNumber	1470.0	1024.87	602.02	1.0	491.25	1020.5	1555.75	2068.0
EnvironmentSatisfaction	1470.0	2.72	1.09	1.0	2.00	3.0	4.00	4.0
HourlyRate	1470.0	65.89	20.33	30.0	48.00	66.0	83.75	100.0
JobInvolvement	1470.0	2.73	0.71	1.0	2.00	3.0	3.00	4.0
JobLevel	1470.0	2.06	1.11	1.0	1.00	2.0	3.00	5.0
JobSatisfaction	1470.0	2.73	1.10	1.0	2.00	3.0	4.00	4.0
MonthlyIncome	1470.0	6502.93	4707.96	1009.0	2911.00	4919.0	8379.00	19999.0
MonthlyRate	1470.0	14313.10	7117.79	2094.0	8047.00	14235.5	20461.50	26999.0
NumCompaniesWorked	1470.0	2.69	2.50	0.0	1.00	2.0	4.00	9.0
PercentSalaryHike	1470.0	15.21	3.66	11.0	12.00	14.0	18.00	25.0
PerformanceRating	1470.0	3.15	0.36	3.0	3.00	3.0	3.00	4.0
RelationshipSatisfaction	1470.0	2.71	1.08	1.0	2.00	3.0	4.00	4.0
StandardHours	1470.0	80.00	0.00	80.0	80.00	80.0	80.00	80.0
StockOptionLevel	1470.0	0.79	0.85	0.0	0.00	1.0	1.00	3.0
TotalWorkingYears	1470.0	11.28	7.78	0.0	6.00	10.0	15.00	40.0
TrainingTimesLastYear	1470.0	2.80	1.29	0.0	2.00	3.0	3.00	6.0
WorkLifeBalance	1470.0	2.76	0.71	1.0	2.00	3.0	3.00	4.0
YearsAtCompany	1470.0	7.01	6.13	0.0	3.00	5.0	9.00	40.0
YearsInCurrentRole	1470.0	4.23	3.62	0.0	2.00	3.0	7.00	18.0
YearsSinceLastPromotion	1470.0	2.19	3.22	0.0	0.00	1.0	3.00	15.0
YearsWithCurrManager	1470.0	4.12	3.57	0.0	2.00	3.0	7.00	17.0

In [69]:
```
employee_data.describe(include="all")
```

Out[69]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	I
count	1470.000000	1470	1470	1470.000000	1470	1470.000000	14
unique	NaN	2	3	NaN	3	NaN	
top	NaN	No	Travel_Rarely	NaN	Research & Development	NaN	
freq	NaN	1233	1043	NaN	961	NaN	
mean	36.923810	NaN	NaN	802.485714	NaN	9.192517	
std	9.135373	NaN	NaN	403.509100	NaN	8.106864	
min	18.000000	NaN	NaN	102.000000	NaN	1.000000	
25%	30.000000	NaN	NaN	465.000000	NaN	2.000000	
50%	36.000000	NaN	NaN	802.000000	NaN	7.000000	
75%	43.000000	NaN	NaN	1157.000000	NaN	14.000000	
max	60.000000	NaN	NaN	1499.000000	NaN	29.000000	

Summary Statistics of categorical variables:

In [70]:
```
# Get the list of categorical columns
cat_cols = employee_data.select_dtypes(include='object').columns.tolist()

# Create a DataFrame containing counts of unique values for each categorical c
olumn
cat_employee_data = pd.DataFrame(employee_data[cat_cols].melt(var_name='colum
n', value_name='value')
                        .value_counts()).rename(columns={0: 'count'}).sort_value
s(by=['column', 'count'])

# Display summary statistics of categorical variables
display(employee_data[cat_cols].describe())

# Display counts of unique values for each categorical column
display(cat_employee_data)
```

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus
count	1470	1470	1470	1470	1470	1470	1470
unique	2	3	3	6	2	9	3
top	No	Travel_Rarely	Research & Development	Life Sciences	Male	Sales Executive	Married
freq	1233	1043	961	606	882	326	673

		count
column	**value**	
Attrition	**Yes**	237
	No	1233
BusinessTravel	**Non-Travel**	150
	Travel_Frequently	277
	Travel_Rarely	1043
Department	**Human Resources**	63
	Sales	446
	Research & Development	961
EducationField	**Human Resources**	27
	Other	82
	Technical Degree	132
	Marketing	159
	Medical	464
	Life Sciences	606
Gender	**Female**	588
	Male	882
JobRole	**Human Resources**	52
	Research Director	80
	Sales Representative	83
	Manager	102
	Healthcare Representative	131
	Manufacturing Director	145
	Laboratory Technician	259
	Research Scientist	292
	Sales Executive	326
MaritalStatus	**Divorced**	327
	Single	470
	Married	673
Over18	**Y**	1470
OverTime	**Yes**	416
	No	1054

In [71]:
```
employee_data.describe(include='O')
```

Out[71]:

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus
count	1470	1470	1470	1470	1470	1470	1470
unique	2	3	3	6	2	9	3
top	No	Travel_Rarely	Research & Development	Life Sciences	Male	Sales Executive	Married
freq	1233	1043	961	606	882	326	673

Dataset Size:

In [72]:
```
employee_data.shape
```

Out[72]:
```
(1470, 35)
```

List of Columns:

In [73]:
```
employee_data.columns
```

Out[73]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

In [74]:
```
num_rows, num_cols = employee_data.shape
print(f'Number of rows: {num_rows}\nNumber of columns: {num_cols}')
```

```
Number of rows: 1470
Number of columns: 35
```

Data Information:

In [75]:
```
employee_data.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 [76]:
```
numerical_vars = employee_data.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_vars = employee_data.select_dtypes(include=['object']).columns.tolist()
print('Numerical variables:', numerical_vars)
print('Categorical variables:', categorical_vars)
```

```
Numerical variables: ['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount', 'EmployeeNumber', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']
Categorical variables: ['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus', 'Over18', 'OverTime']
```

In [77]:
```
# Count the number of categorical and numerical variables
categorical_count = employee_data.select_dtypes(include='object').shape[1]
numerical_count = employee_data.select_dtypes(exclude='object').shape[1]

print(f"Number of categorical variables: {categorical_count}")
print(f"Number of numerical variables: {numerical_count}")
```

```
Number of categorical variables: 9
Number of numerical variables: 26
```

In [78]:
```
# Counts of unique values for each categorical column
cat_unique_counts = employee_data[cat_cols].nunique().reset_index().rename(columns={'index': 'column', 0: 'unique_count'})

# Counts of non-unique values for each categorical column
cat_non_unique_counts = employee_data[cat_cols].apply(lambda x: x.shape[0] - x.nunique()).reset_index().rename(columns={'index': 'column', 0: 'non_unique_count'})

# Merging unique and non-unique counts
cat_counts = pd.merge(cat_unique_counts, cat_non_unique_counts, on='column')
print("\nCounts of Unique and Non-Unique Values for Categorical Variables:")
print(cat_counts)
```

```
Counts of Unique and Non-Unique Values for Categorical Variables:
          column  unique_count  non_unique_count
0      Attrition             2              1468
1  BusinessTravel            3              1467
2     Department             3              1467
3  EducationField             6              1464
4         Gender              2              1468
5        JobRole              9              1461
6  MaritalStatus              3              1467
7         Over18              1              1469
8       OverTime              2              1468
```

In [79]:
```
# Inspect useless features
employee_data.nunique().sort_values()
```

Out[79]:
```
Over18                         1
StandardHours                  1
EmployeeCount                  1
Gender                         2
Attrition                      2
PerformanceRating              2
OverTime                       2
MaritalStatus                  3
Department                     3
BusinessTravel                 3
StockOptionLevel               4
EnvironmentSatisfaction        4
JobInvolvement                 4
JobSatisfaction                4
RelationshipSatisfaction       4
WorkLifeBalance                4
Education                      5
JobLevel                       5
EducationField                 6
TrainingTimesLastYear          7
JobRole                        9
NumCompaniesWorked            10
PercentSalaryHike             15
YearsSinceLastPromotion       16
YearsWithCurrManager          18
YearsInCurrentRole            19
DistanceFromHome              29
YearsAtCompany                37
TotalWorkingYears             40
Age                           43
HourlyRate                    71
DailyRate                    886
MonthlyIncome               1349
MonthlyRate                 1427
EmployeeNumber              1470
dtype: int64
```

Duplicate Records:

In [80]:
```
employee_data[employee_data.duplicated(keep=False)]
```

Out[80]:

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatio

No Duplicate Records present in the Dataset.

In [81]:
```
employee_data=employee_data.drop_duplicates()
employee_data.shape
```

Out[81]:
```
(1470, 35)
```

Missing Values:

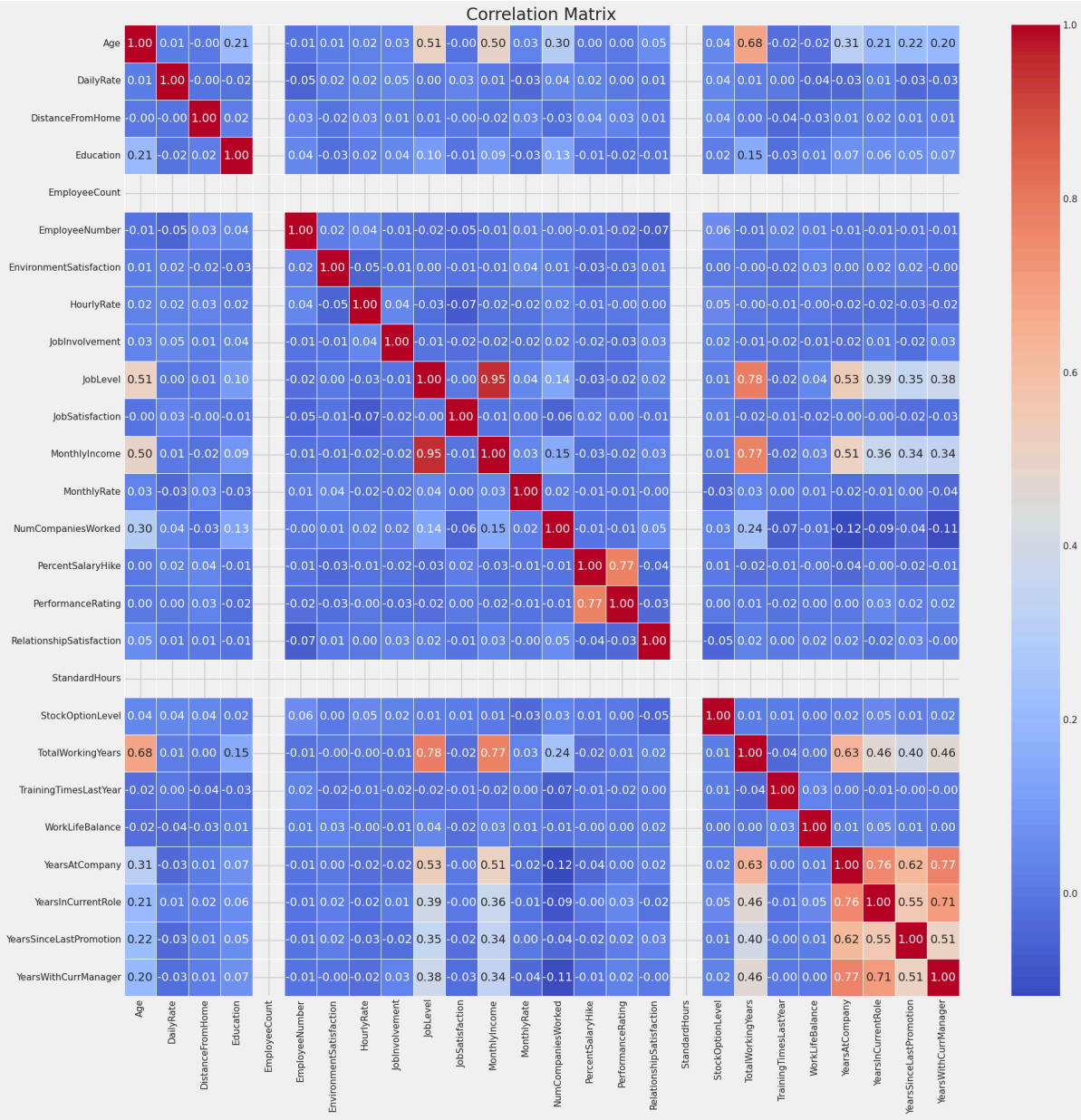
In [82]:
```
missing_employee_data = employee_data.isnull().sum().to_frame().rename(columns
={0:"Total No. of Missing Values"})
missing_employee_data["% of Missing Values"] = round((missing_employee_data["T
otal No. of Missing Values"]/len(employee_data))*100,2)
missing_employee_data
```

Out[82]:

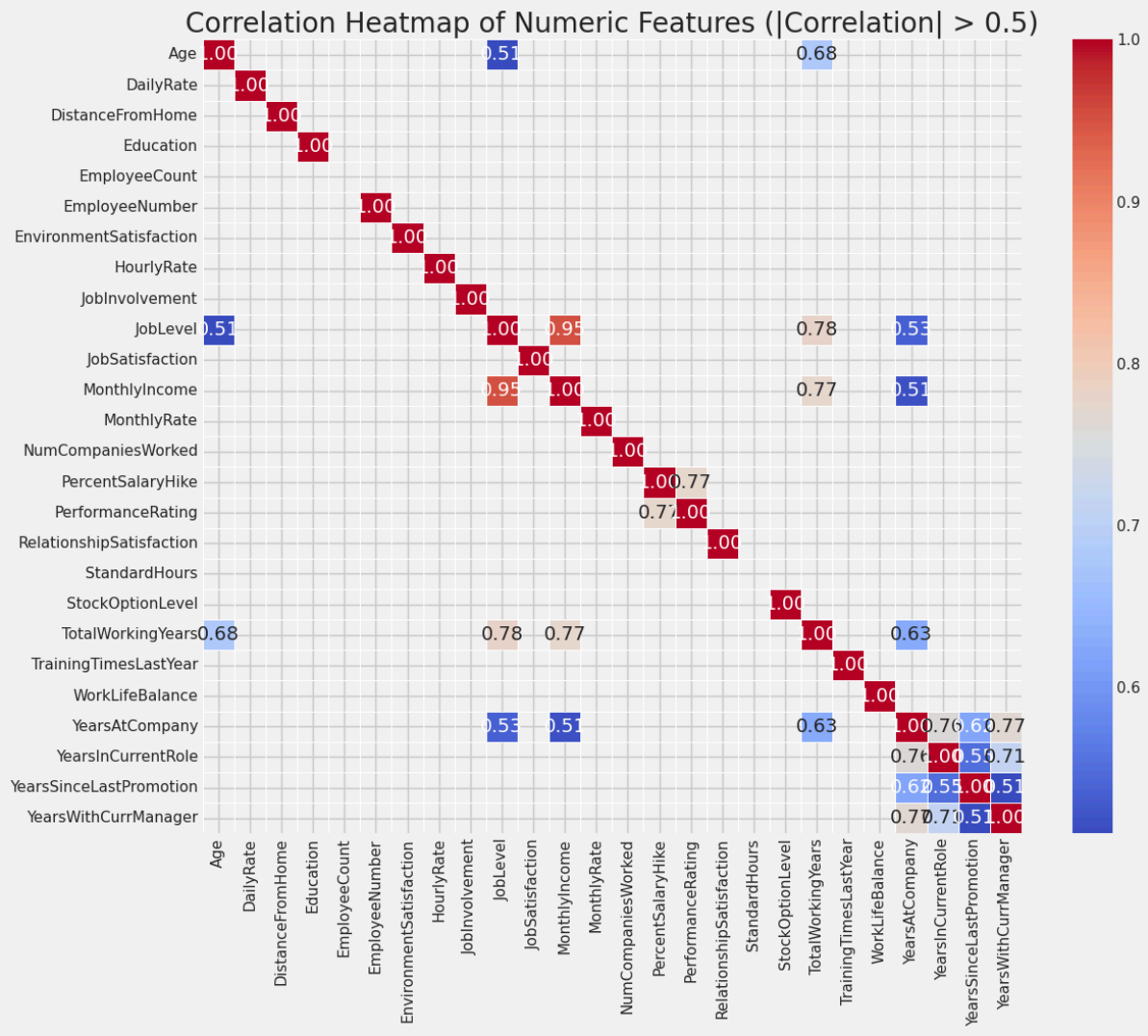
	Total No. of Missing Values	% of Missing Values
Age	0	0.0
Attrition	0	0.0
BusinessTravel	0	0.0
DailyRate	0	0.0
Department	0	0.0
DistanceFromHome	0	0.0
Education	0	0.0
EducationField	0	0.0
EmployeeCount	0	0.0
EmployeeNumber	0	0.0
EnvironmentSatisfaction	0	0.0
Gender	0	0.0
HourlyRate	0	0.0
JobInvolvement	0	0.0
JobLevel	0	0.0
JobRole	0	0.0
JobSatisfaction	0	0.0
MaritalStatus	0	0.0
MonthlyIncome	0	0.0
MonthlyRate	0	0.0
NumCompaniesWorked	0	0.0
Over18	0	0.0
OverTime	0	0.0
PercentSalaryHike	0	0.0
PerformanceRating	0	0.0
RelationshipSatisfaction	0	0.0
StandardHours	0	0.0
StockOptionLevel	0	0.0
TotalWorkingYears	0	0.0
TrainingTimesLastYear	0	0.0
WorkLifeBalance	0	0.0
YearsAtCompany	0	0.0
YearsInCurrentRole	0	0.0
YearsSinceLastPromotion	0	0.0
YearsWithCurrManager	0	0.0

Exploratory Data Analysis and Data Visualization

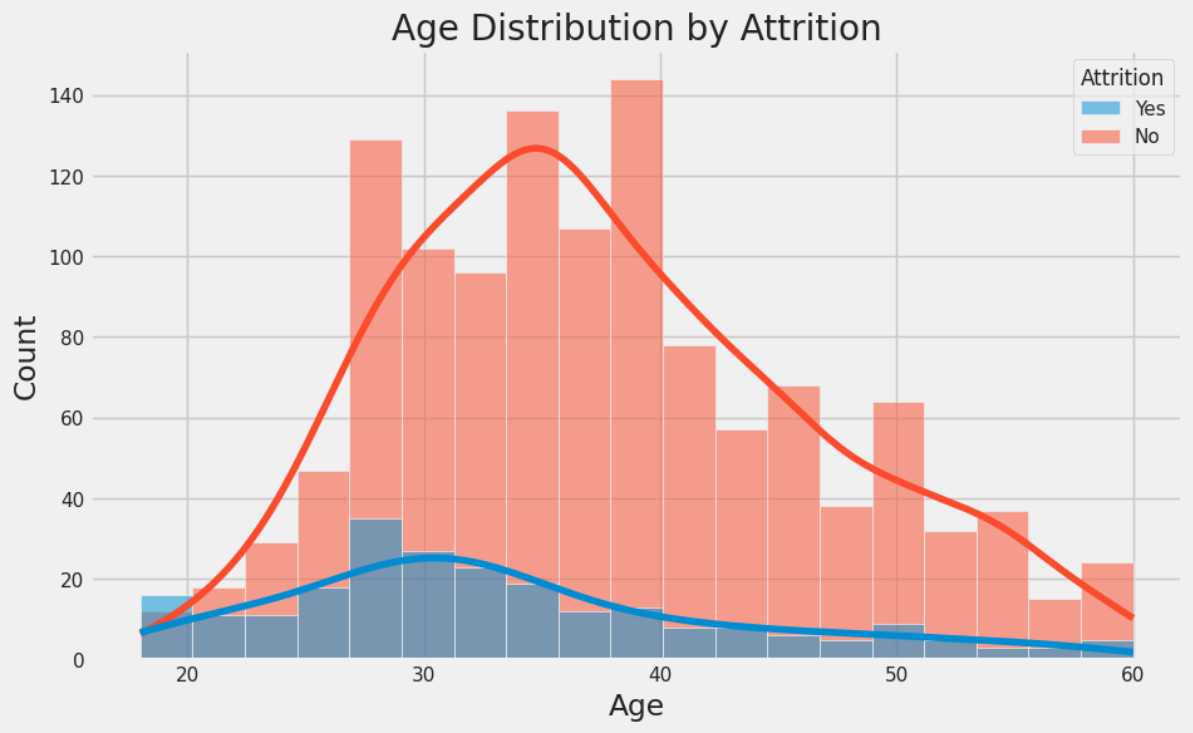
In [83]:
```
# Correlation matrix

# Select only the numeric columns from the DataFrame
numeric_columns = employee_data.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_columns.corr()

# Create a heatmap to visualize the correlations
plt.figure(figsize=(20, 20))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewi
dths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



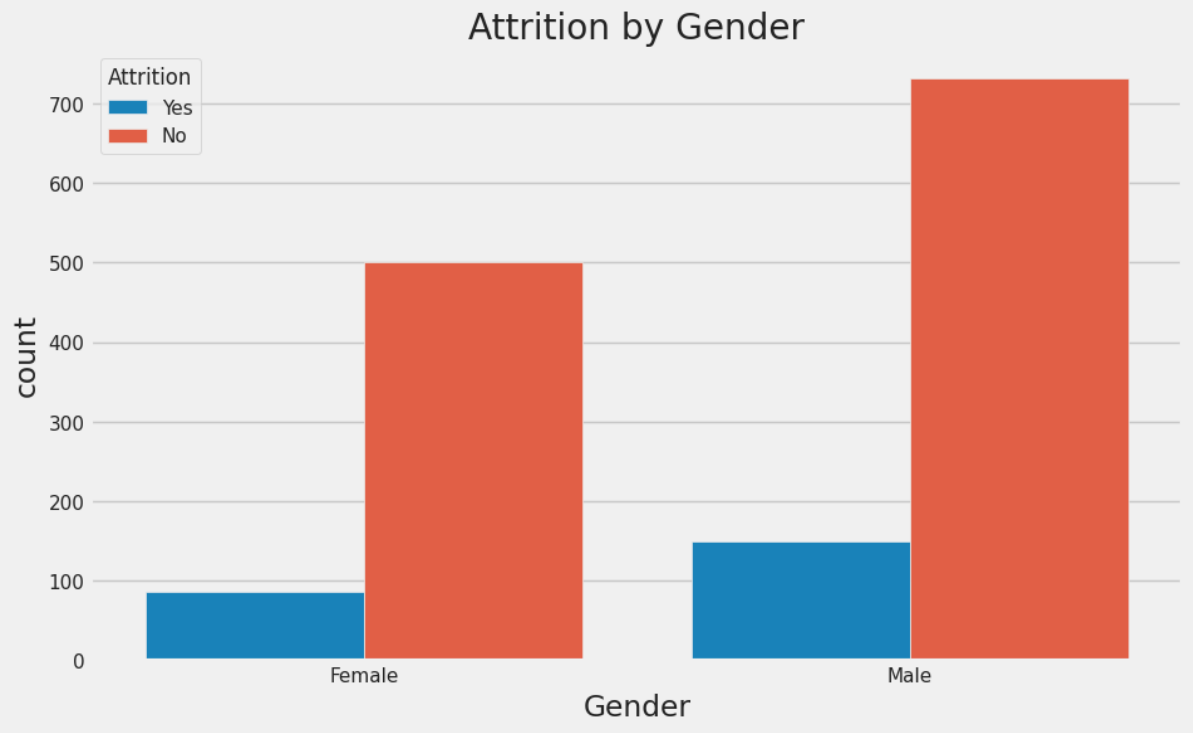
In [84]:
```
# Heatmap Plotting
# Select only numeric columns
numeric_columns = employee_data.select_dtypes(include=['int64', 'float64'])

# Calculate the correlation matrix
corr_matrix = numeric_columns.corr()

# Filter correlation matrix to include values greater than 0.5 or less than -0.5
corr_matrix_filtered = corr_matrix[(corr_matrix > 0.5) | (corr_matrix < -0.5)]

# Plot the heatmap with filtered correlation values
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix_filtered, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of Numeric Features (|Correlation| > 0.5)')
plt.show()
```



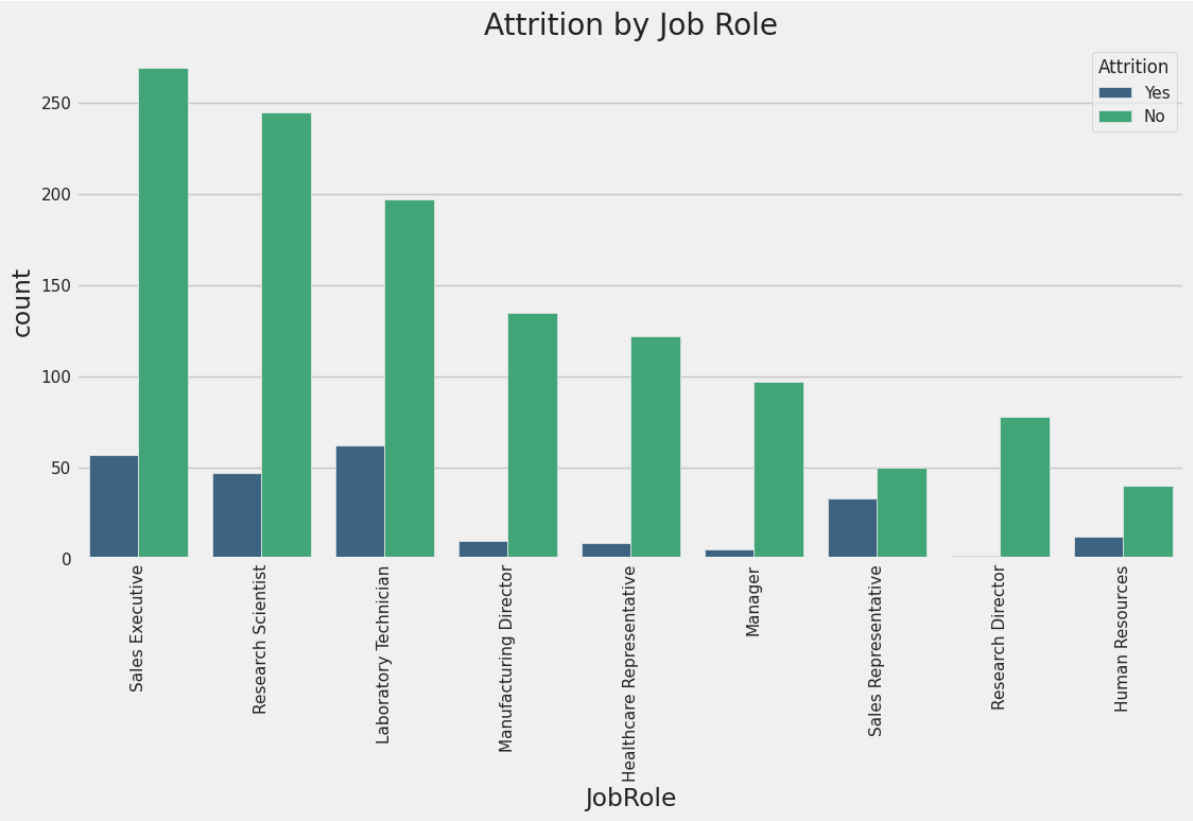
In [85]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=employee_data, x='Age', hue='Attrition', kde=True)
plt.title('Age Distribution by Attrition')
plt.show()
```



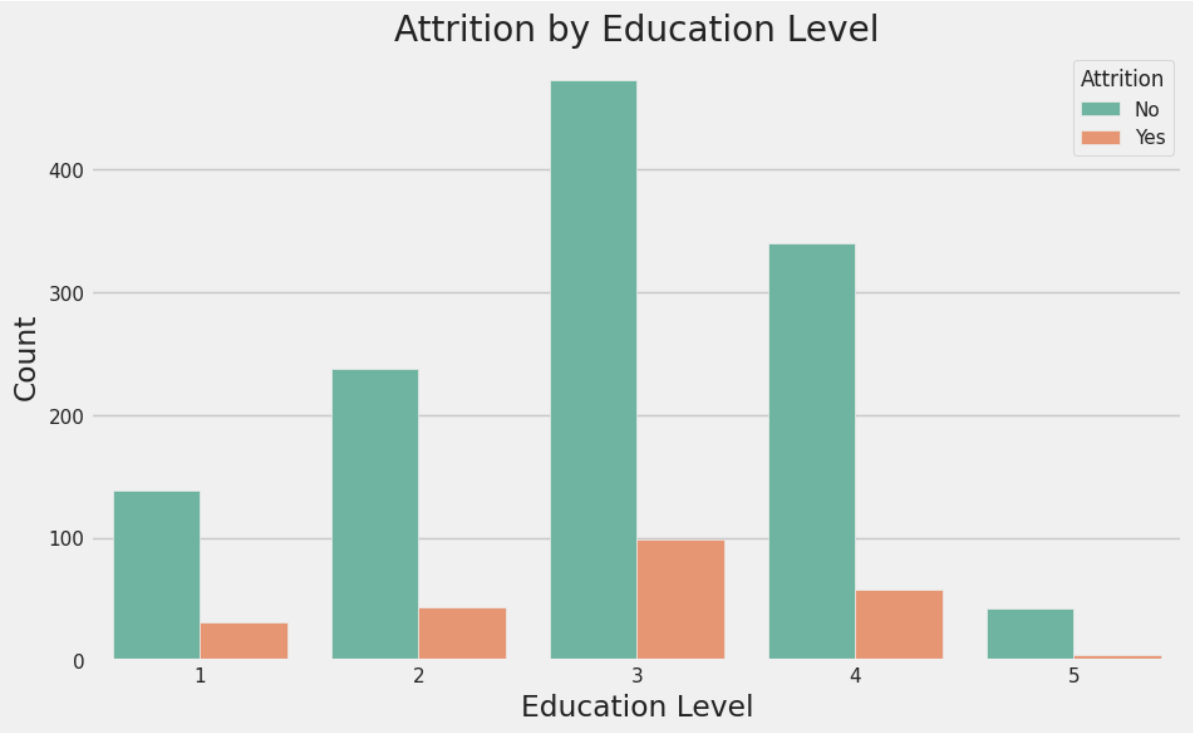
In [86]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=employee_data, x='Gender', hue='Attrition')
plt.title('Attrition by Gender')
plt.show()
```



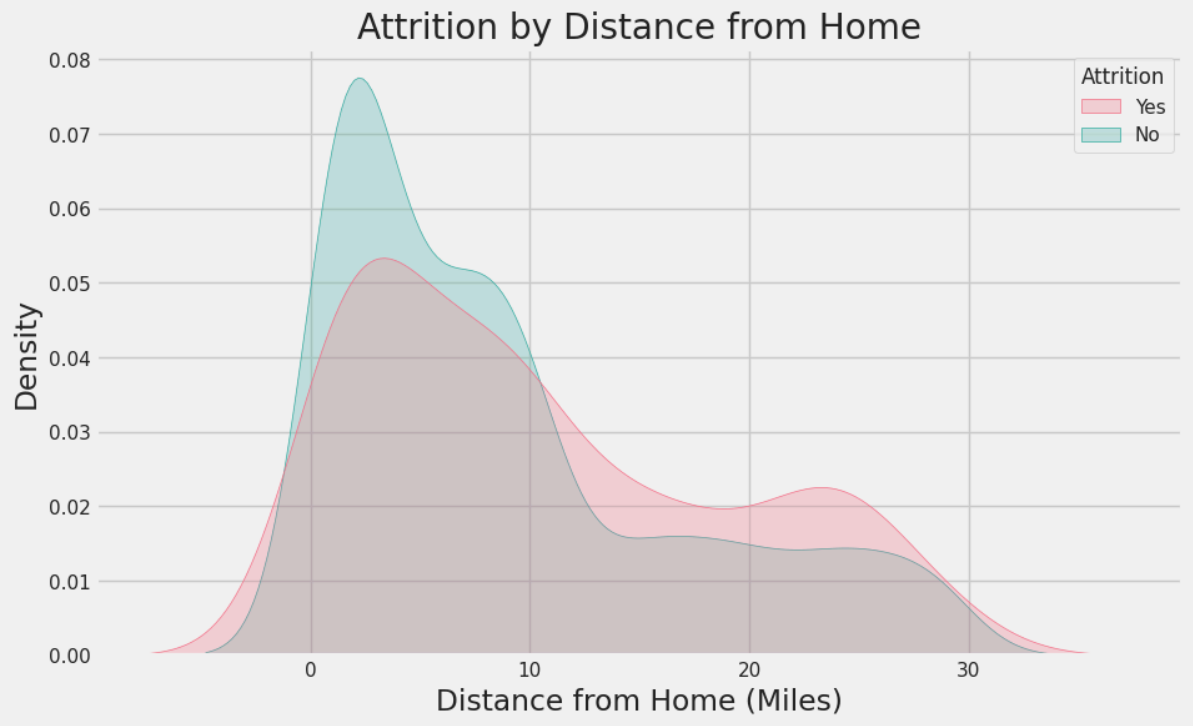
In [87]:
```
plt.figure(figsize=(12, 6))
sns.countplot(data=employee_data, x='JobRole', hue='Attrition', palette='viridis')
plt.title('Attrition by Job Role')
plt.xticks(rotation=90)
plt.show()
```



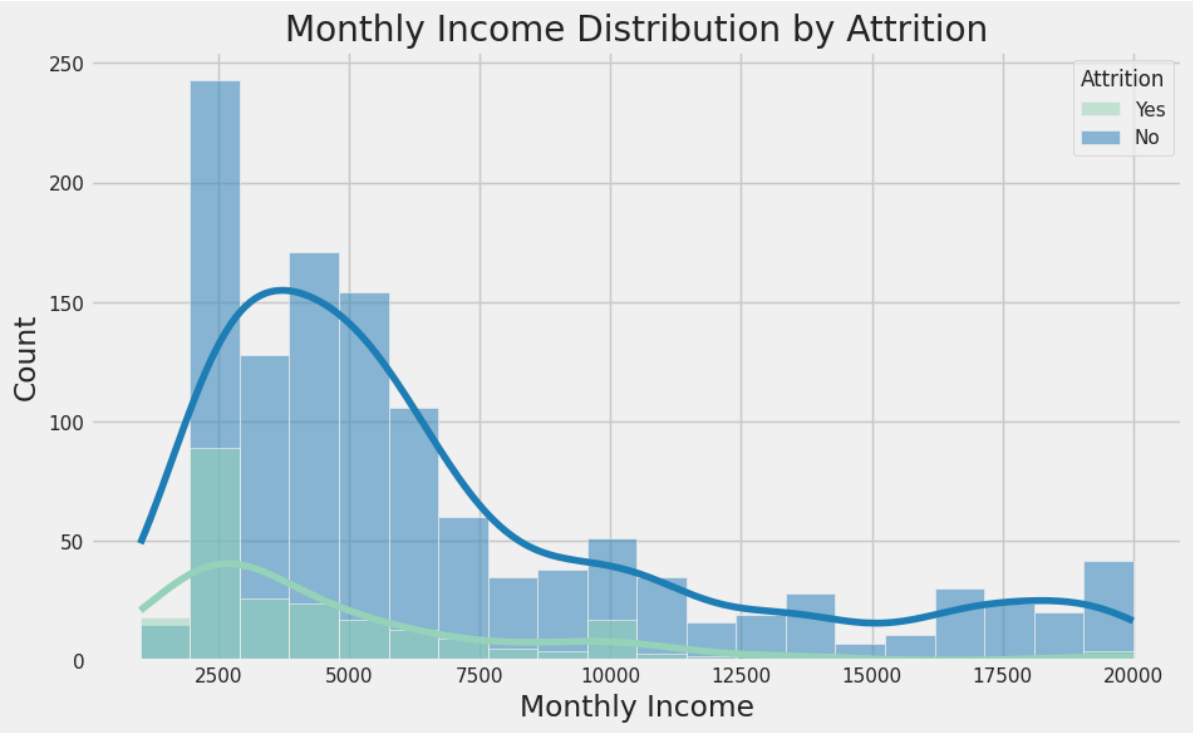
In [88]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=employee_data, x='Education', hue='Attrition', palette='Set2')
plt.title('Attrition by Education Level')
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.show()
```



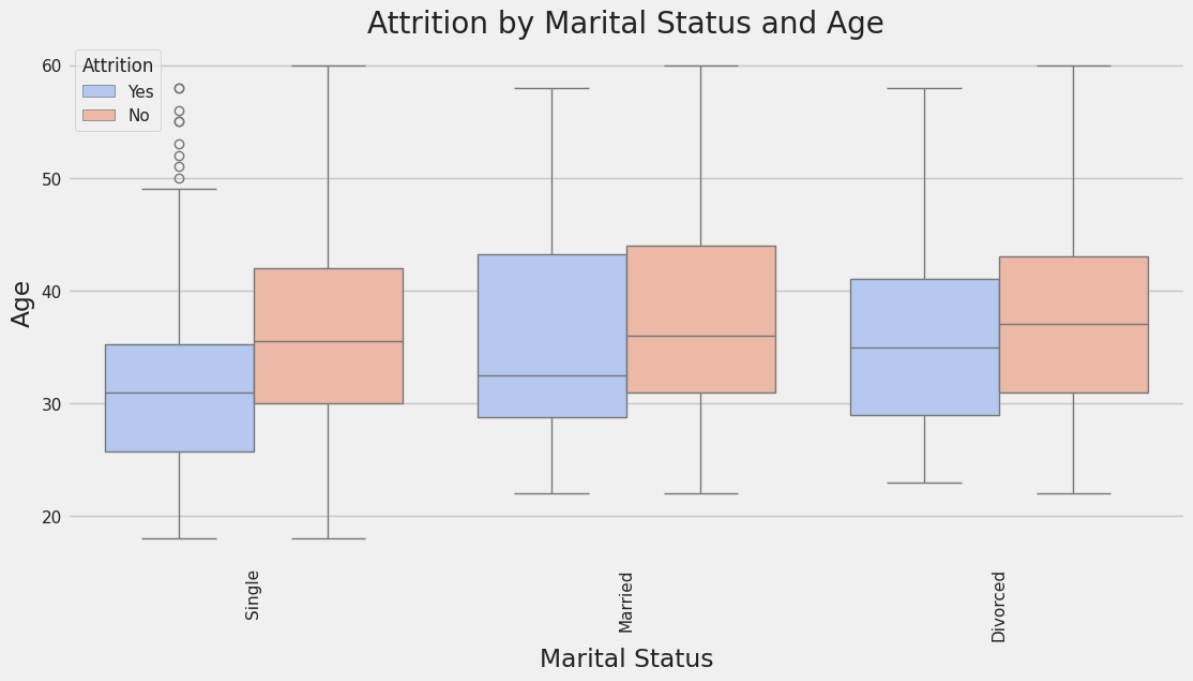
In [89]:
```
plt.figure(figsize=(10, 6))
sns.kdeplot(data=employee_data, x='DistanceFromHome', hue='Attrition', fill=True, common_norm=False, palette='husl')
plt.title('Attrition by Distance from Home')
plt.xlabel('Distance from Home (Miles)')
plt.ylabel('Density')
plt.show()
```



In [90]:
```
plt.figure(figsize=(10, 6))
sns.histplot(data=employee_data, x='MonthlyIncome', hue='Attrition', kde=True, palette='YlGnBu')
plt.title('Monthly Income Distribution by Attrition')
plt.xlabel('Monthly Income')
plt.ylabel('Count')
plt.show()
```



In [91]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=employee_data, x='MaritalStatus', y='Age', hue='Attrition', p
alette='coolwarm')
plt.title('Attrition by Marital Status and Age')
plt.xticks(rotation=90)
plt.xlabel('Marital Status')
plt.ylabel('Age')
plt.show()
```



Checking Unique Value Of Categorical Attributes

In [92]:
```
# Calculate the number of unique values in each column
for column in employee_data.columns:
    print(f"{column} - Number of unique values : {employee_data[column].nunique()}")
    print("-----------------------------------------------------------------------")
```

```
Age - Number of unique values : 43
--------------------------------------------------------------------------
Attrition - Number of unique values : 2
--------------------------------------------------------------------------
BusinessTravel - Number of unique values : 3
--------------------------------------------------------------------------
DailyRate - Number of unique values : 886
--------------------------------------------------------------------------
Department - Number of unique values : 3
--------------------------------------------------------------------------
DistanceFromHome - Number of unique values : 29
--------------------------------------------------------------------------
Education - Number of unique values : 5
--------------------------------------------------------------------------
EducationField - Number of unique values : 6
--------------------------------------------------------------------------
EmployeeCount - Number of unique values : 1
--------------------------------------------------------------------------
EmployeeNumber - Number of unique values : 1470
--------------------------------------------------------------------------
EnvironmentSatisfaction - Number of unique values : 4
--------------------------------------------------------------------------
Gender - Number of unique values : 2
--------------------------------------------------------------------------
HourlyRate - Number of unique values : 71
--------------------------------------------------------------------------
JobInvolvement - Number of unique values : 4
--------------------------------------------------------------------------
JobLevel - Number of unique values : 5
--------------------------------------------------------------------------
JobRole - Number of unique values : 9
--------------------------------------------------------------------------
JobSatisfaction - Number of unique values : 4
--------------------------------------------------------------------------
MaritalStatus - Number of unique values : 3
--------------------------------------------------------------------------
MonthlyIncome - Number of unique values : 1349
--------------------------------------------------------------------------
MonthlyRate - Number of unique values : 1427
--------------------------------------------------------------------------
NumCompaniesWorked - Number of unique values : 10
--------------------------------------------------------------------------
Over18 - Number of unique values : 1
--------------------------------------------------------------------------
OverTime - Number of unique values : 2
--------------------------------------------------------------------------
PercentSalaryHike - Number of unique values : 15
--------------------------------------------------------------------------
PerformanceRating - Number of unique values : 2
--------------------------------------------------------------------------
RelationshipSatisfaction - Number of unique values : 4
--------------------------------------------------------------------------
StandardHours - Number of unique values : 1
--------------------------------------------------------------------------
StockOptionLevel - Number of unique values : 4
--------------------------------------------------------------------------
TotalWorkingYears - Number of unique values : 40
--------------------------------------------------------------------------
TrainingTimesLastYear - Number of unique values : 7
--------------------------------------------------------------------------
WorkLifeBalance - Number of unique values : 4
--------------------------------------------------------------------------
YearsAtCompany - Number of unique values : 37
```

```
------------------------------------------------------------------------
YearsInCurrentRole - Number of unique values : 19
------------------------------------------------------------------------
YearsSinceLastPromotion - Number of unique values : 16
------------------------------------------------------------------------
YearsWithCurrManager - Number of unique values : 18
------------------------------------------------------------------------
```

In [93]:

```
categorical_features = []
for column in employee_data.columns:
    if employee_data[column].dtype == object and len(employee_data[column].uni
que()) <= 30:
        categorical_features.append(column)
        print(f"{column} : {employee_data[column].unique()}")
        print(employee_data[column].value_counts())
        print("==========================================================
====================")
categorical_features.remove('Attrition')
```

```
Attrition : ['Yes' 'No']
Attrition
No     1233
Yes     237
Name: count, dtype: int64
================================================================================
=======
BusinessTravel : ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
BusinessTravel
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: count, dtype: int64
================================================================================
=======
Department : ['Sales' 'Research & Development' 'Human Resources']
Department
Research & Development    961
Sales                     446
Human Resources            63
Name: count, dtype: int64
================================================================================
=======
EducationField : ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical De
gree'
 'Human Resources']
EducationField
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: count, dtype: int64
================================================================================
=======
Gender : ['Female' 'Male']
Gender
Male      882
Female    588
Name: count, dtype: int64
================================================================================
=======
JobRole : ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
JobRole
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: count, dtype: int64
================================================================================
=======
MaritalStatus : ['Single' 'Married' 'Divorced']
MaritalStatus
Married     673
Single      470
```

```
Divorced    327
Name: count, dtype: int64
================================================================================
=======
Over18 : ['Y']
Over18
Y    1470
Name: count, dtype: int64
================================================================================
=======
OverTime : ['Yes' 'No']
OverTime
No     1054
Yes     416
Name: count, dtype: int64
================================================================================
=======
```

In [94]:

```
# Get the value counts for the 'Attrition' column
value_counts = employee_data['Attrition'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
       Count  Percentage
No      1233       83.88
Yes      237       16.12
Total   1470      100.00
```

In [95]:
```
# Get the value counts for the 'BusinessTravel' column
value_counts = employee_data['BusinessTravel'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
                   Count  Percentage
Travel_Rarely       1043       70.95
Travel_Frequently    277       18.84
Non-Travel           150       10.20
Total               1470      100.00
```

In [96]:
```
# Get the value counts for the 'Department' column
value_counts = employee_data['Department'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
                        Count  Percentage
Research & Development    961       65.37
Sales                     446       30.34
Human Resources            63        4.29
Total                    1470      100.00
```

In [97]:
```
# Get the value counts for the 'EducationField' column
value_counts = employee_data['EducationField'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
                  Count  Percentage
Life Sciences       606       41.22
Medical             464       31.56
Marketing           159       10.82
Technical Degree    132        8.98
Other                82        5.58
Human Resources      27        1.84
Total              1470      100.00
```

In [98]:
```
# Get the value counts for the 'Gender' column
value_counts = employee_data['Gender'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
        Count  Percentage
Male      882        60.0
Female    588        40.0
Total    1470       100.0
```

In [99]:
```
# Get the value counts for the 'JobRole' column
value_counts = employee_data['JobRole'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
                          Count  Percentage
Sales Executive             326       22.18
Research Scientist          292       19.86
Laboratory Technician       259       17.62
Manufacturing Director      145        9.86
Healthcare Representative   131        8.91
Manager                     102        6.94
Sales Representative         83        5.65
Research Director            80        5.44
Human Resources              52        3.54
Total                      1470      100.00
```

In [100]:
```
# Get the value counts for the 'MaritalStatus' column
value_counts = employee_data['MaritalStatus'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
          Count  Percentage
Married     673       45.78
Single      470       31.97
Divorced    327       22.24
Total      1470      100.00
```

In [101]:
```
# Get the value counts for the 'OverTime' column
value_counts = employee_data['OverTime'].value_counts()

# Calculate the percentage
percentage = (value_counts / len(employee_data)) * 100

# Concatenate the count and percentage into a DataFrame
count_and_percentage = pd.concat([value_counts, percentage], axis=1)
count_and_percentage.columns = ['Count', 'Percentage']

# Round the values in the DataFrame to two decimal places
count_and_percentage_rounded = count_and_percentage.round(2)

# Calculate the total count
total_count = len(employee_data)

# Create a DataFrame for total count
total_employee_data = pd.DataFrame({'Count': [total_count], 'Percentage': [10
0]}, index=['Total'])

# Concatenate the total count DataFrame with the counts and percentages DataFr
ame
result_employee_data = pd.concat([count_and_percentage_rounded, total_employee
_data])

# Display the result DataFrame
print(result_employee_data)
```

```
       Count  Percentage
No      1054        71.7
Yes      416        28.3
Total   1470       100.0
```

In [102]:
```
# Print the shape of the DataFrame
print("The shape of data frame:", employee_data.shape)
# Print the length (number of rows) of the DataFrame
print("Number of Rows in the dataframe:", len(employee_data))
# Print the number of columns in the DataFrame
print("Number of Columns in the dataframe:", len(employee_data.columns))
```

```
The shape of data frame: (1470, 35)
Number of Rows in the dataframe: 1470
Number of Columns in the dataframe: 35
```

Encoding of Dataset

In [103]:
```
# Convert categorical variables into numerical form.
label = LabelEncoder()
employee_data["Attrition"] = label.fit_transform(employee_data.Attrition)
```

In [104]:
```
employee_data.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   int64 
 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(27), object(8)
memory usage: 402.1+ KB
```

Data Preprocessing

In [105]:
```
# Transform categorical data into dummies
dummy_col = [column for column in employee_data.drop('Attrition', axis=1).colu
mns if employee_data[column].nunique() < 20]
data = pd.get_dummies(employee_data, columns=dummy_col, drop_first=True, dtype
='uint8')
data.info()
```

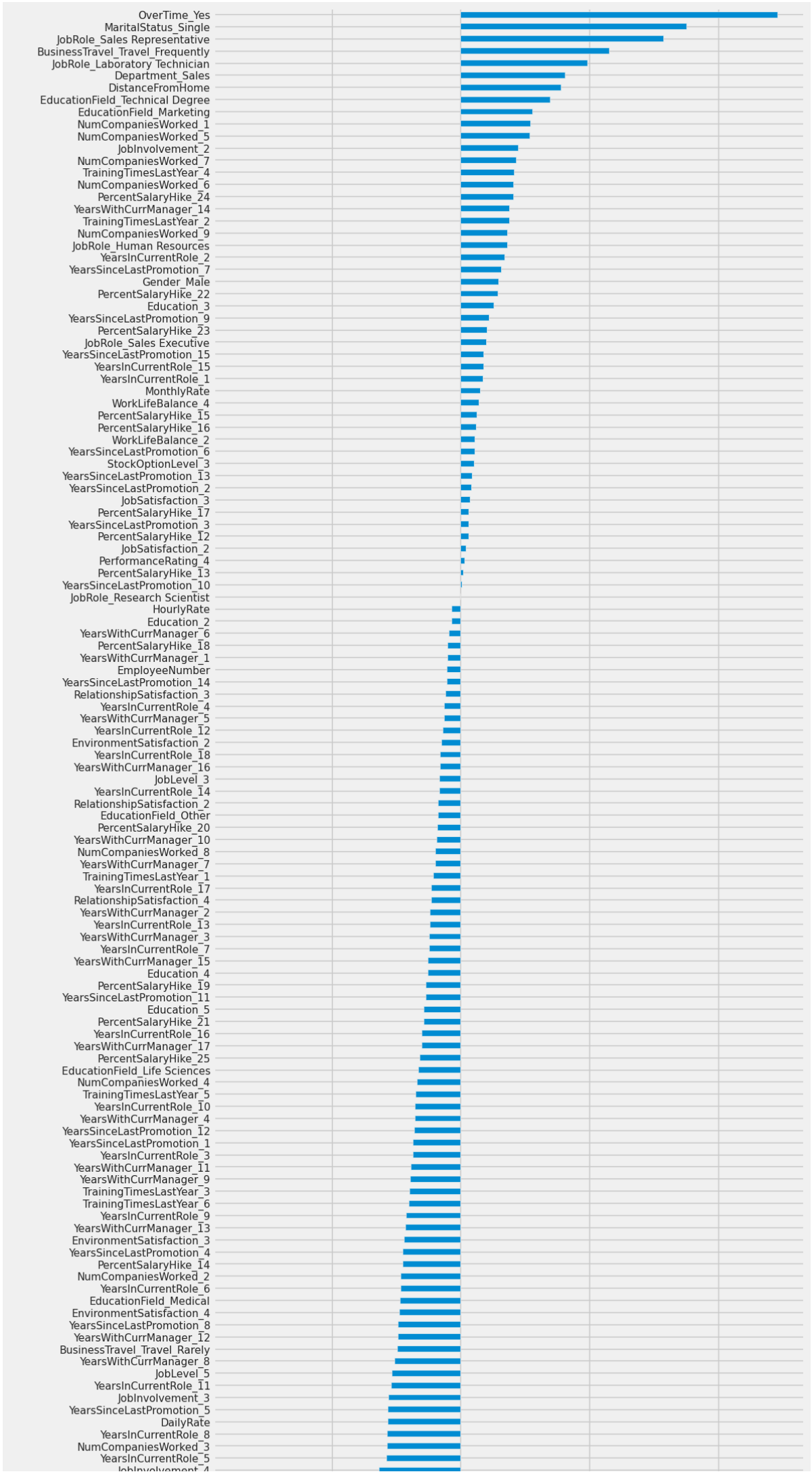
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Columns: 137 entries, Age to YearsWithCurrManager_17
dtypes: int64(10), uint8(127)
memory usage: 297.3 KB
```

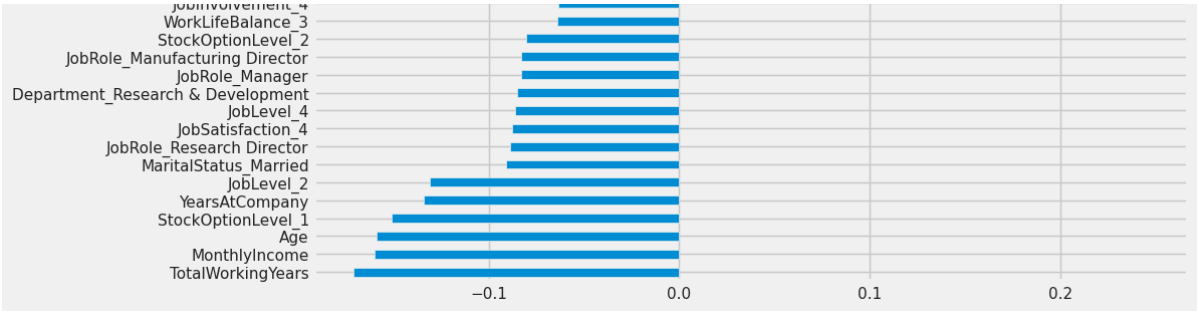
In [106]:
```
print(data.shape)

# Remove duplicate Features
data = data.T.drop_duplicates()
data = data.T

# Remove Duplicate Rows
data.drop_duplicates(inplace=True)

print(data.shape)
```

```
(1470, 137)
(1470, 137)
```

In [107]:
```
data.drop('Attrition', axis=1).corrwith(data.Attrition).sort_values().plot(kin
d='barh', figsize=(10, 30))
```

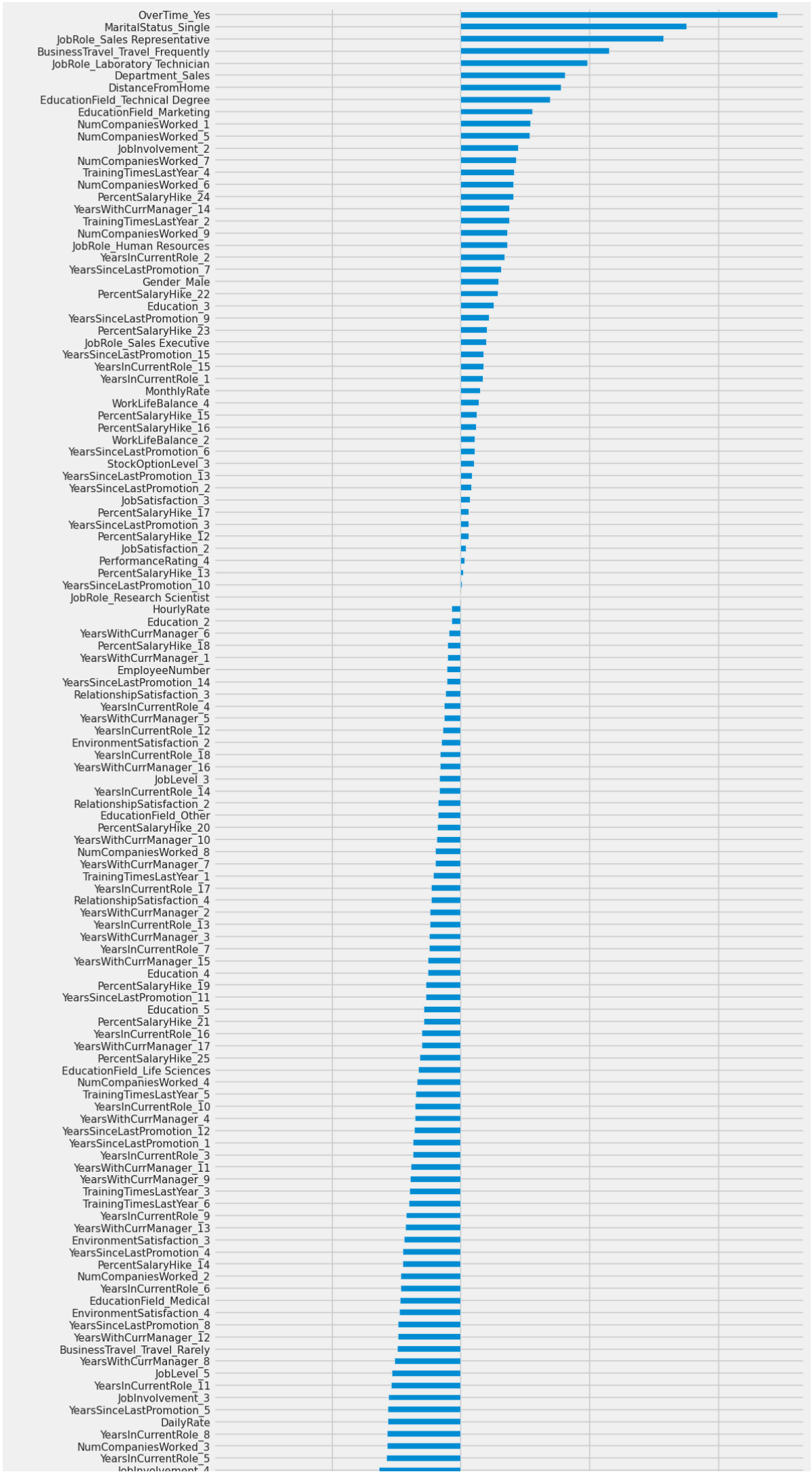
Out[107]: <Axes: >

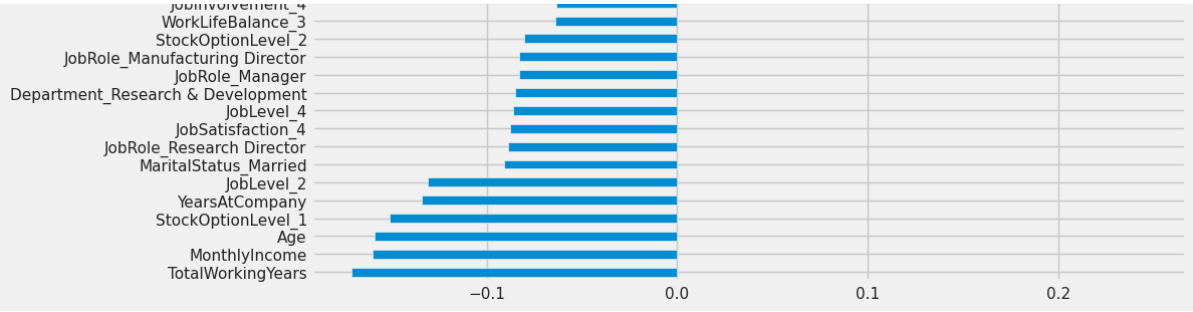

OverTime_Yes
MaritalStatus_Single
JobRole_Sales Representative
BusinessTravel_Travel_Frequently
JobRole_Laboratory Technician
Department_Sales
DistanceFromHome
EducationField_Technical Degree
EducationField_Marketing
NumCompaniesWorked_1
NumCompaniesWorked_5
JobInvolvement_2
NumCompaniesWorked_7
TrainingTimesLastYear_4
NumCompaniesWorked_6
PercentSalaryHike_24
YearsWithCurrManager_14
TrainingTimesLastYear_2
NumCompaniesWorked_9
JobRole_Human Resources
YearsInCurrentRole_2
YearsSinceLastPromotion_7
Gender_Male
PercentSalaryHike_22
Education_3
YearsSinceLastPromotion_9
PercentSalaryHike_23
JobRole_Sales Executive
YearsSinceLastPromotion_15
YearsInCurrentRole_15
YearsInCurrentRole_1
MonthlyRate
WorkLifeBalance_4
PercentSalaryHike_15
PercentSalaryHike_16
WorkLifeBalance_2
YearsSinceLastPromotion_6
StockOptionLevel_3
YearsSinceLastPromotion_13
YearsSinceLastPromotion_2
JobSatisfaction_3
PercentSalaryHike_17
YearsSinceLastPromotion_3
PercentSalaryHike_12
JobSatisfaction_2
PerformanceRating_4
PercentSalaryHike_13
YearsSinceLastPromotion_10
JobRole_Research Scientist
HourlyRate
Education_2
YearsWithCurrManager_6
PercentSalaryHike_18
YearsWithCurrManager_1
EmployeeNumber
YearsSinceLastPromotion_14
RelationshipSatisfaction_3
YearsInCurrentRole_4
YearsWithCurrManager_5
YearsInCurrentRole_12
EnvironmentSatisfaction_2
YearsInCurrentRole_18
YearsWithCurrManager_16
JobLevel_3
YearsInCurrentRole_14
RelationshipSatisfaction_2
EducationField_Other
PercentSalaryHike_20
YearsWithCurrManager_10
NumCompaniesWorked_8
YearsWithCurrManager_7
TrainingTimesLastYear_1
YearsInCurrentRole_17
RelationshipSatisfaction_4
YearsWithCurrManager_2
YearsInCurrentRole_13
YearsWithCurrManager_3
YearsInCurrentRole_7
YearsWithCurrManager_15
Education_4
PercentSalaryHike_19
YearsSinceLastPromotion_11
Education_5
PercentSalaryHike_21
YearsInCurrentRole_16
YearsWithCurrManager_17
PercentSalaryHike_25
EducationField_Life Sciences
NumCompaniesWorked_4
TrainingTimesLastYear_5
YearsInCurrentRole_10
YearsWithCurrManager_4
YearsSinceLastPromotion_12
YearsSinceLastPromotion_1
YearsInCurrentRole_3
YearsWithCurrManager_11
YearsWithCurrManager_9
TrainingTimesLastYear_3
TrainingTimesLastYear_6
YearsInCurrentRole_9
YearsWithCurrManager_13
EnvironmentSatisfaction_3
YearsSinceLastPromotion_4
PercentSalaryHike_14
NumCompaniesWorked_2
YearsInCurrentRole_6
EducationField_Medical
EnvironmentSatisfaction_4
YearsSinceLastPromotion_8
YearsWithCurrManager_12
BusinessTravel_Travel_Rarely
YearsWithCurrManager_8
JobLevel_5
YearsInCurrentRole_11
JobInvolvement_3
YearsSinceLastPromotion_5
DailyRate
YearsInCurrentRole_8
NumCompaniesWorked_3
YearsInCurrentRole_5
JobInvolvement_4


JobInvolvement_4
WorkLifeBalance_3
StockOptionLevel_2
JobRole_Manufacturing Director
JobRole_Manager
Department_Research & Development
JobLevel_4
JobSatisfaction_4
JobRole_Research Director
MaritalStatus_Married
JobLevel_2
YearsAtCompany
StockOptionLevel_1
Age
MonthlyIncome
TotalWorkingYears
−0.1
0.0
0.1
0.2

In [108]:
```
data.drop('Attrition', axis=1).corrwith(data.Attrition).sort_values().plot(kin
d='barh', figsize=(10, 30))
```

Out[108]: <Axes: >


OverTime_Yes
MaritalStatus_Single
JobRole_Sales Representative
BusinessTravel_Travel_Frequently
JobRole_Laboratory Technician
Department_Sales
DistanceFromHome
EducationField_Technical Degree
EducationField_Marketing
NumCompaniesWorked_1
NumCompaniesWorked_5
JobInvolvement_2
NumCompaniesWorked_7
TrainingTimesLastYear_4
NumCompaniesWorked_6
PercentSalaryHike_24
YearsWithCurrManager_14
TrainingTimesLastYear_2
NumCompaniesWorked_9
JobRole_Human Resources
YearsInCurrentRole_2
YearsSinceLastPromotion_7
Gender_Male
PercentSalaryHike_22
Education_3
YearsSinceLastPromotion_9
PercentSalaryHike_23
JobRole_Sales Executive
YearsSinceLastPromotion_15
YearsInCurrentRole_15
YearsInCurrentRole_1
MonthlyRate
WorkLifeBalance_4
PercentSalaryHike_15
PercentSalaryHike_16
WorkLifeBalance_2
YearsSinceLastPromotion_6
StockOptionLevel_3
YearsSinceLastPromotion_13
YearsSinceLastPromotion_2
JobSatisfaction_3
PercentSalaryHike_17
YearsSinceLastPromotion_3
PercentSalaryHike_12
JobSatisfaction_2
PerformanceRating_4
PercentSalaryHike_13
YearsSinceLastPromotion_10
JobRole_Research Scientist
HourlyRate
Education_2
YearsWithCurrManager_6
PercentSalaryHike_18
YearsWithCurrManager_1
EmployeeNumber
YearsSinceLastPromotion_14
RelationshipSatisfaction_3
YearsInCurrentRole_4
YearsWithCurrManager_5
YearsInCurrentRole_12
EnvironmentSatisfaction_2
YearsInCurrentRole_18
YearsWithCurrManager_16
JobLevel_3
YearsInCurrentRole_14
RelationshipSatisfaction_2
EducationField_Other
PercentSalaryHike_20
YearsWithCurrManager_10
NumCompaniesWorked_8
YearsWithCurrManager_7
TrainingTimesLastYear_1
YearsInCurrentRole_17
RelationshipSatisfaction_4
YearsWithCurrManager_2
YearsInCurrentRole_13
YearsWithCurrManager_3
YearsInCurrentRole_7
YearsWithCurrManager_15
Education_4
PercentSalaryHike_19
YearsSinceLastPromotion_11
Education_5
PercentSalaryHike_21
YearsInCurrentRole_16
YearsWithCurrManager_17
PercentSalaryHike_25
EducationField_Life Sciences
NumCompaniesWorked_4
TrainingTimesLastYear_5
YearsInCurrentRole_10
YearsWithCurrManager_4
YearsSinceLastPromotion_12
YearsSinceLastPromotion_1
YearsInCurrentRole_3
YearsWithCurrManager_11
YearsWithCurrManager_9
TrainingTimesLastYear_3
TrainingTimesLastYear_6
YearsInCurrentRole_9
YearsWithCurrManager_13
EnvironmentSatisfaction_3
YearsSinceLastPromotion_4
PercentSalaryHike_14
NumCompaniesWorked_2
YearsInCurrentRole_6
EducationField_Medical
EnvironmentSatisfaction_4
YearsSinceLastPromotion_8
YearsWithCurrManager_12
BusinessTravel_Travel_Rarely
YearsWithCurrManager_8
JobLevel_5
YearsInCurrentRole_11
JobInvolvement_3
YearsSinceLastPromotion_5
DailyRate
YearsInCurrentRole_8
NumCompaniesWorked_3
YearsInCurrentRole_5
JobInvolvement_4



In [47]:
```
feature_correlation = data.drop('Attrition', axis=1).corrwith(data.Attrition).
sort_values()
model_col = feature_correlation[np.abs(feature_correlation) > 0.02].index
len(model_col)
```

Out[47]: 92

Splitting the Dataset

In [48]:
```
X = data.drop('Attrition', axis=1)
y = data.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=42,
                                                    stratify=y)

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
X_std = scaler.transform(X)
```

In [49]:
```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

In [50]:
```
y_test.value_counts()[0] / y_test.shape[0]
```

Out[50]: 0.8390022675736961

Showing the distribution of Attrition After splitting

In [51]:
```
stay = (y_train.value_counts()[0] / y_train.shape)[0]
leave = (y_train.value_counts()[1] / y_train.shape)[0]

print("===============TRAIN=================")
print(f"Staying Rate: {stay * 100:.2f}%")
print(f"Leaving Rate: {leave * 100 :.2f}%")

stay = (y_test.value_counts()[0] / y_test.shape)[0]
leave = (y_test.value_counts()[1] / y_test.shape)[0]

print("===============TEST=================")
print(f"Staying Rate: {stay * 100:.2f}%")
print(f"Leaving Rate: {leave * 100 :.2f}%")
```

```
===============TRAIN=================
Staying Rate: 83.87%
Leaving Rate: 16.13%
===============TEST=================
Staying Rate: 83.90%
Leaving Rate: 16.10%
```

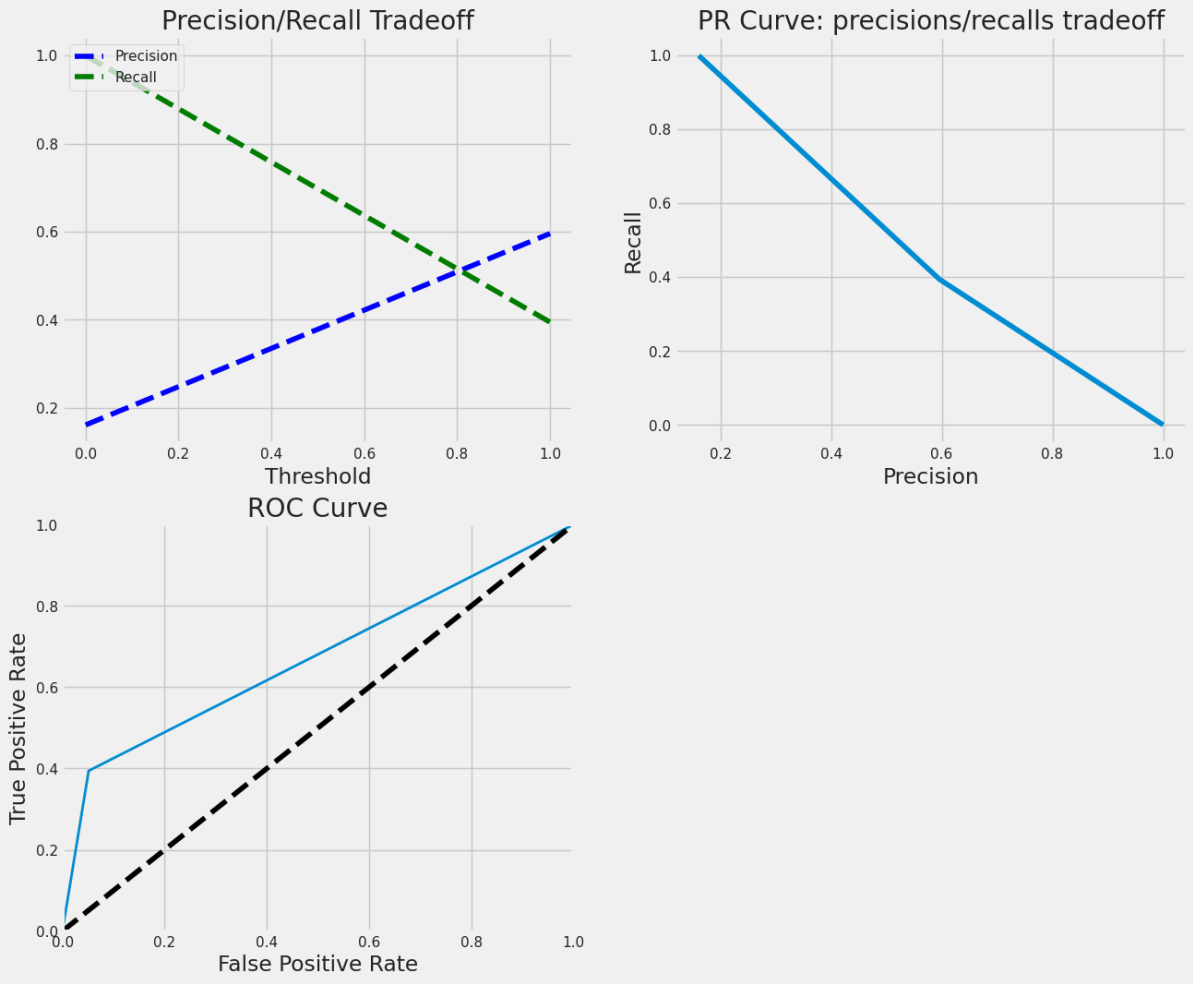
In [52]:
```
def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, out
put_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, outpu
t_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

Logistic Regression

In [62]:
```
lr_clf = LogisticRegression(solver='liblinear', penalty='l1')
lr_clf.fit(X_train_std, y_train)

evaluate(lr_clf, X_train_std, X_test_std, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[848  15]
 [ 59 107]]
ACCURACY SCORE:
0.9281
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.934950    0.877049  0.928086     0.906000      0.925610
recall       0.982619    0.644578  0.928086     0.813599      0.928086
f1-score     0.958192    0.743056  0.928086     0.850624      0.923486
support    863.000000  166.000000  0.928086  1029.000000   1029.000000
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[351  19]
 [ 43  28]]
ACCURACY SCORE:
0.8594
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.890863   0.595745   0.85941    0.743304      0.843350
recall       0.948649   0.394366   0.85941    0.671507      0.859410
f1-score     0.918848   0.474576   0.85941    0.696712      0.847321
support    370.000000  71.000000   0.85941  441.000000    441.000000
```

In [63]:
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.title("Precision/Recall Tradeoff")


def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')


precisions, recalls, thresholds = precision_recall_curve(y_test, lr_clf.predic
t(X_test_std))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, lr_clf.predict(X_test_std))
plot_roc_curve(fpr, tpr)
```

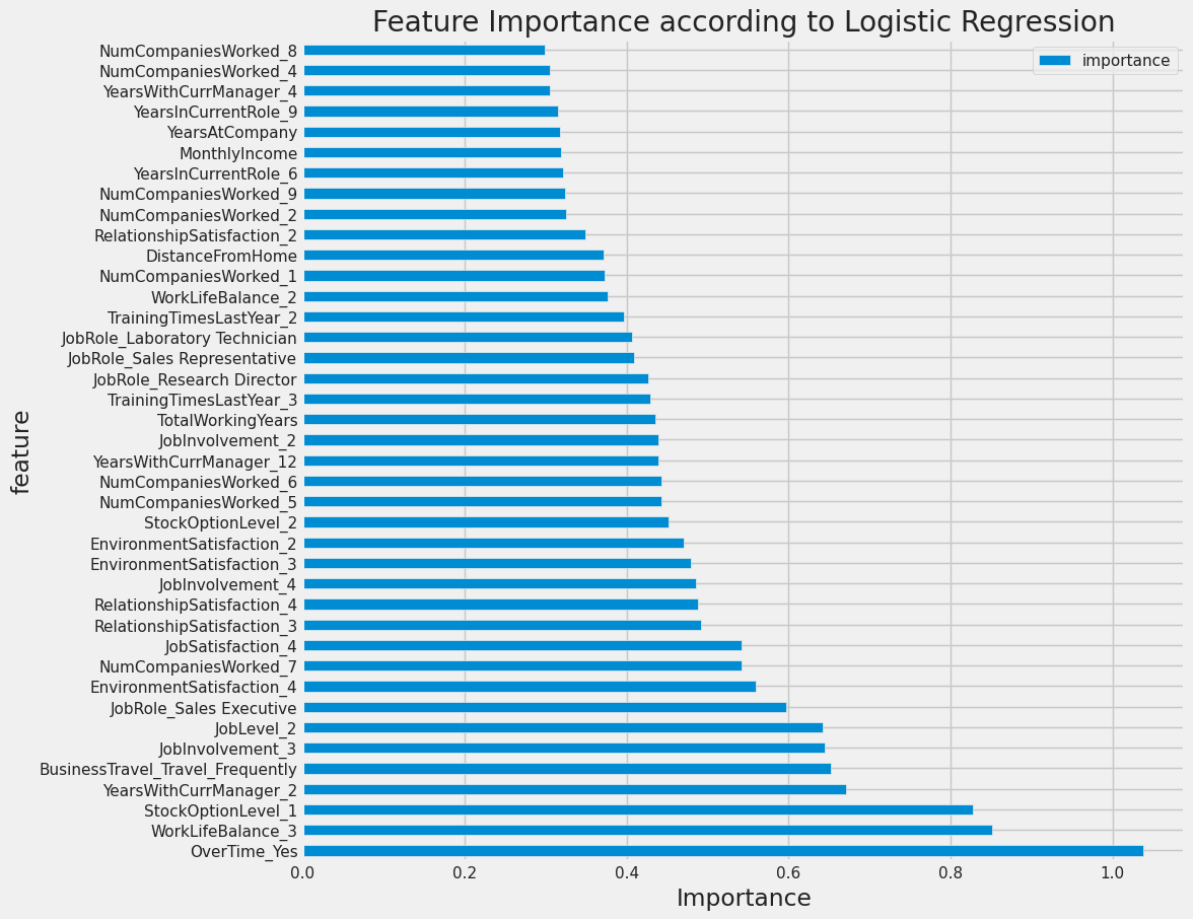


In [109]:
```
scores_dict = {
    'Logistic Regression': {
        'Train': roc_auc_score(y_train, lr_clf.predict(X_train)),
        'Test': roc_auc_score(y_test, lr_clf.predict(X_test)),
    },
}
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but LogisticRegression was fitted without feature names

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but LogisticRegression was fitted without feature names
```

In [110]:
```
# Get feature coefficients from the logistic regression model
feature_importance = np.abs(lr_clf.coef_[0])

# Create DataFrame for feature importance
df = pd.DataFrame({'feature': X.columns, 'importance': feature_importance})

# Sort DataFrame by importance in descending order
df = df.sort_values(by='importance', ascending=False)

# Select top 40 features
df = df[:40]

# Plotting top 40 feature importance
plt.figure(figsize=(10, 10))
df.set_index('feature').plot(kind='barh', figsize=(10, 10))
plt.title('Feature Importance according to Logistic Regression')
plt.xlabel('Importance')
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



Random Forest Clasifier

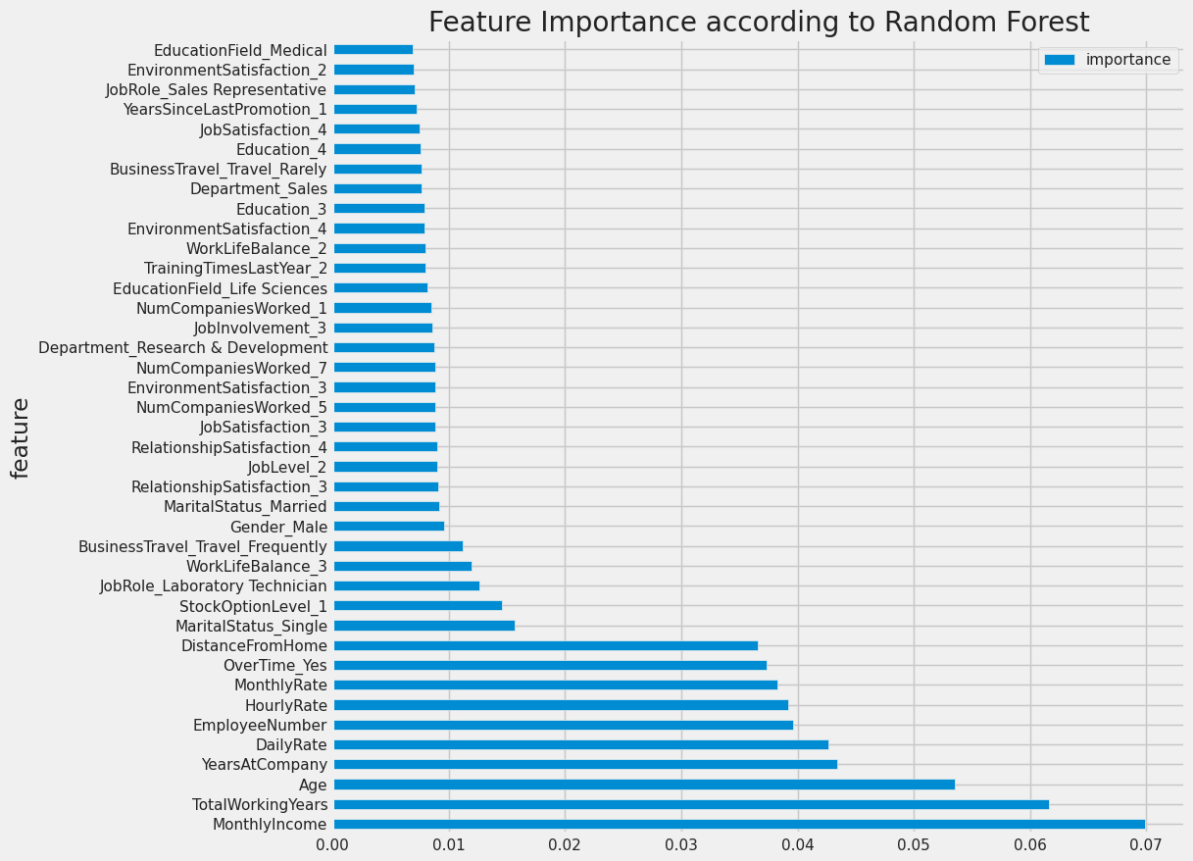
In [124]:
```
rf_clf = RandomForestClassifier(n_estimators=100, bootstrap=False,
#                                     class_weight={0:stay, 1:leave}
                                    )
rf_clf.fit(X_train, y_train)
evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
================================
CONFUSION MATRIX:
[[863   0]
 [  0 166]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
                0      1  accuracy  macro avg  weighted avg
precision     1.0    1.0       1.0        1.0           1.0
recall        1.0    1.0       1.0        1.0           1.0
f1-score      1.0    1.0       1.0        1.0           1.0
support     863.0  166.0       1.0     1029.0        1029.0
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[361   9]
 [ 62   9]]
ACCURACY SCORE:
0.8390
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.853428    0.500000  0.839002    0.676714      0.796527
recall       0.975676    0.126761  0.839002    0.551218      0.839002
f1-score     0.910467    0.202247  0.839002    0.556357      0.796445
support    370.000000   71.000000  0.839002  441.000000    441.000000
```

In [125]:
```
scores_dict['Random Forest'] = {
        'Train': roc_auc_score(y_train, rf_clf.predict(X_train)),
        'Test': roc_auc_score(y_test, rf_clf.predict(X_test)),
    }
```

In [126]:
```
df = feature_imp(X, rf_clf)[:40]
df.set_index('feature', inplace=True)
df.plot(kind='barh', figsize=(10, 10))
plt.title('Feature Importance according to Random Forest')
```

Out[126]:
```
Text(0.5, 1.0, 'Feature Importance according to Random Forest')
```



Support Vector Machine

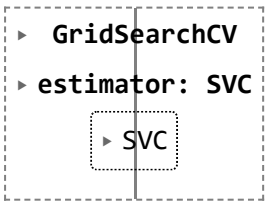
In [111]:
```
svm_clf = SVC(kernel='linear')
svm_clf.fit(X_train_std, y_train)

evaluate(svm_clf, X_train_std, X_test_std, y_train, y_test)
```

```
TRAINIG RESULTS:
================================
CONFUSION MATRIX:
[[857   6]
 [ 48 118]]
ACCURACY SCORE:
0.9475
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.946961    0.951613  0.947522     0.949287      0.947712
recall       0.993048    0.710843  0.947522     0.851945      0.947522
f1-score     0.969457    0.813793  0.947522     0.891625      0.944345
support    863.000000  166.000000  0.947522  1029.000000   1029.000000
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[345  25]
 [ 46  25]]
ACCURACY SCORE:
0.8390
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.882353    0.500000  0.839002    0.691176      0.820795
recall       0.932432    0.352113  0.839002    0.642273      0.839002
f1-score     0.906702    0.413223  0.839002    0.659962      0.827253
support    370.000000   71.000000  0.839002  441.000000    441.000000
```

In [112]:
```
svm_clf = SVC(random_state=42)

param_grid = [
    {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
    {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
]

search = GridSearchCV(svm_clf, param_grid=param_grid, scoring='roc_auc', cv=3,
refit=True, verbose=1)
search.fit(X_train_std, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

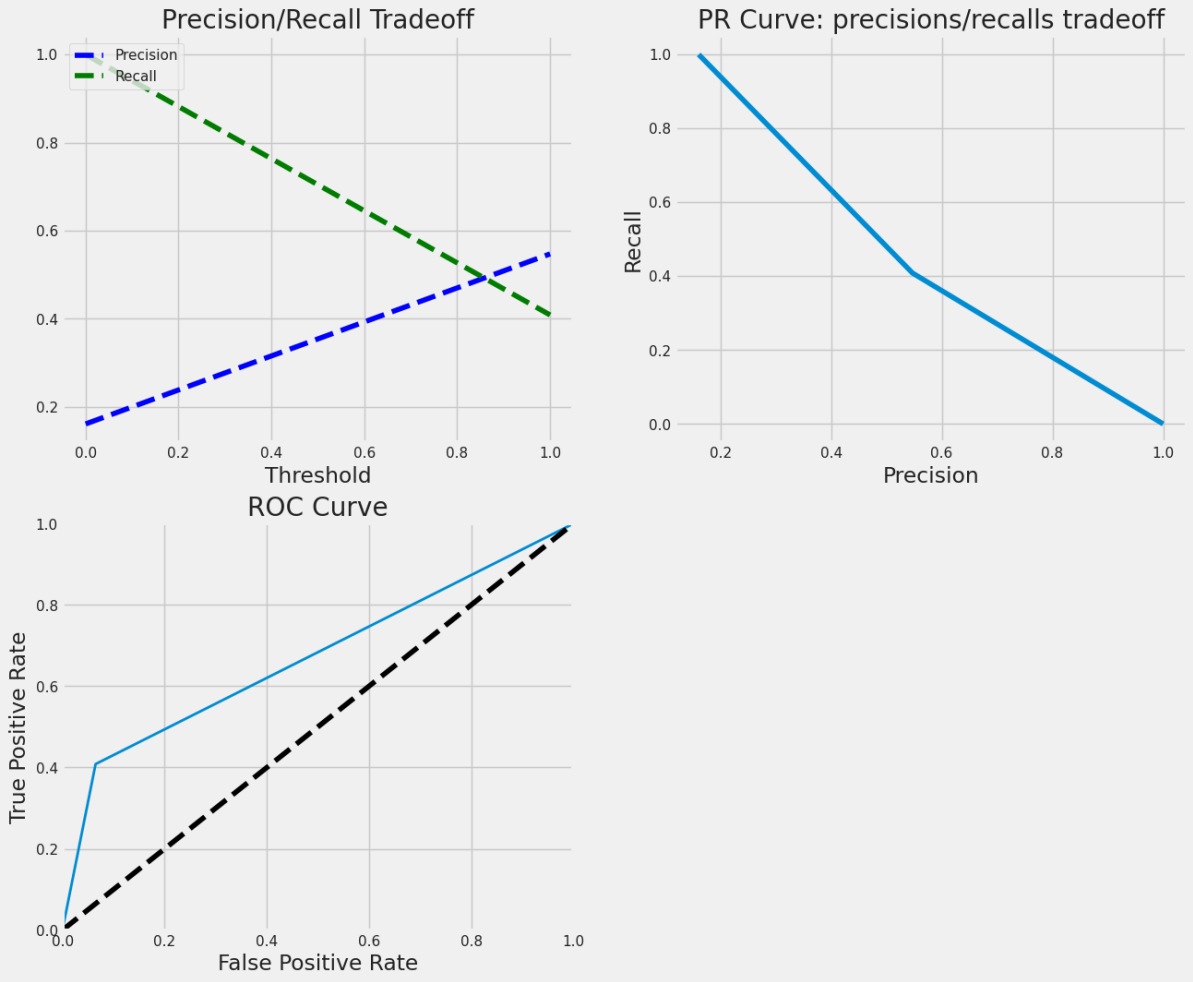
Out[112]:



In [113]:
```
svm_clf = SVC(**search.best_params_)
svm_clf.fit(X_train_std, y_train)

evaluate(svm_clf, X_train_std, X_test_std, y_train, y_test)
```

```
TRAINIG RESULTS:
================================
CONFUSION MATRIX:
[[862   1]
 [  6 160]]
ACCURACY SCORE:
0.9932
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.993088    0.993789  0.993197     0.993438      0.993201
recall       0.998841    0.963855  0.993197     0.981348      0.993197
f1-score     0.995956    0.978593  0.993197     0.987275      0.993155
support    863.000000  166.000000  0.993197  1029.000000   1029.000000
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[346  24]
 [ 42  29]]
ACCURACY SCORE:
0.8503
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.891753    0.547170   0.85034    0.719461      0.836276
recall       0.935135    0.408451   0.85034    0.671793      0.850340
f1-score     0.912929    0.467742   0.85034    0.690335      0.841255
support    370.000000   71.000000   0.85034  441.000000    441.000000
```

In [114]:
```
precisions, recalls, thresholds = precision_recall_curve(y_test, svm_clf.predi
ct(X_test_std))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, svm_clf.predict(X_test_std))
plot_roc_curve(fpr, tpr)
```



In [115]:
```
scores_dict['Support Vector Machine'] = {
        'Train': roc_auc_score(y_train, svm_clf.predict(X_train_std)),
        'Test': roc_auc_score(y_test, svm_clf.predict(X_test_std)),
    }
```

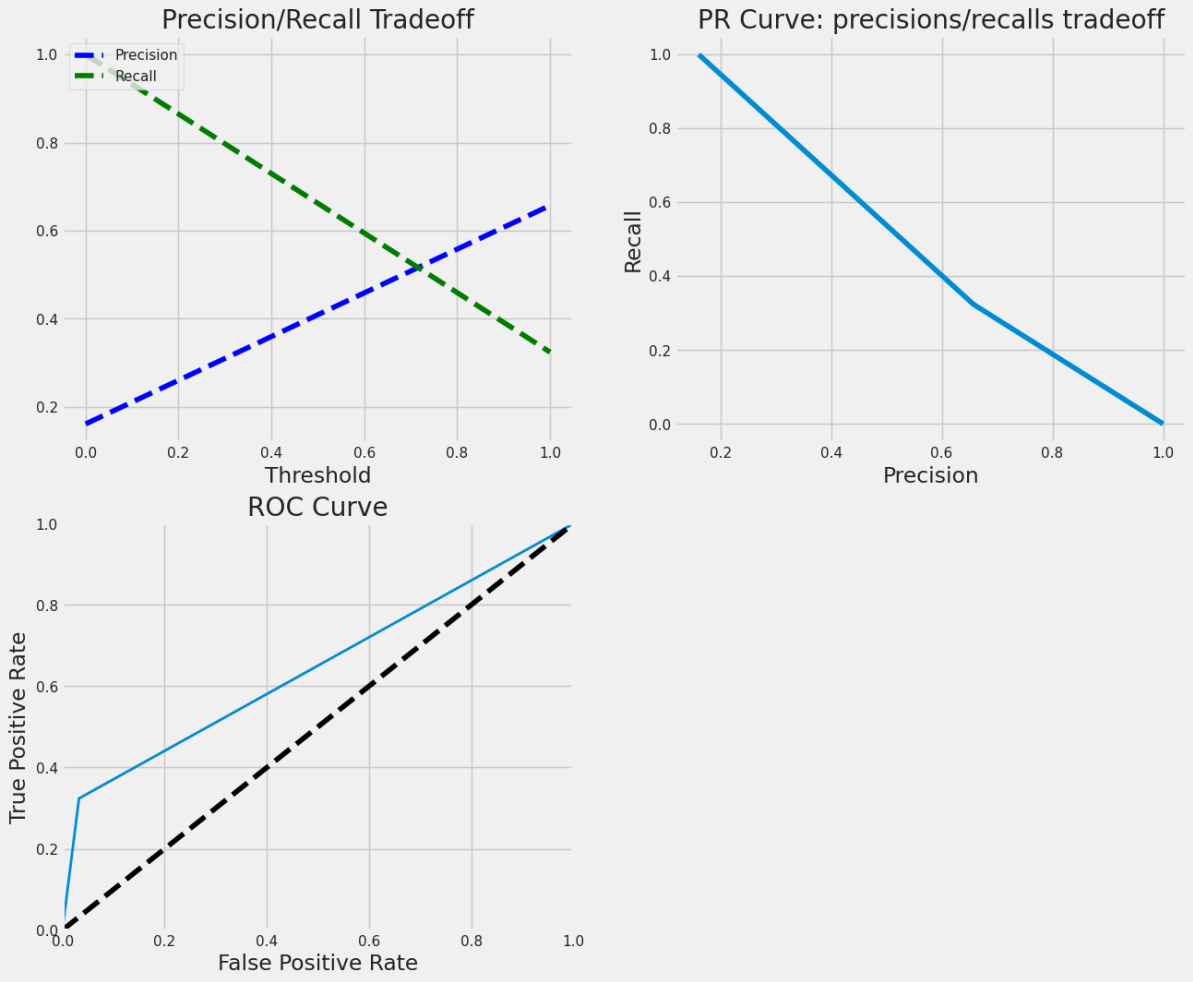
XGBoost Classifier.

In [117]:
```
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)

evaluate(xgb_clf, X_train, X_test, y_train, y_test)
```

```
TRAINIG RESULTS:
===============================
CONFUSION MATRIX:
[[863   0]
 [  0 166]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
                0      1  accuracy  macro avg  weighted avg
precision     1.0    1.0       1.0        1.0           1.0
recall        1.0    1.0       1.0        1.0           1.0
f1-score      1.0    1.0       1.0        1.0           1.0
support     863.0  166.0       1.0     1029.0        1029.0
TESTING RESULTS:
===============================
CONFUSION MATRIX:
[[358  12]
 [ 48  23]]
ACCURACY SCORE:
0.8639
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.881773    0.657143  0.863946    0.769458      0.845608
recall       0.967568    0.323944  0.863946    0.645756      0.863946
f1-score     0.922680    0.433962  0.863946    0.678321      0.843998
support    370.000000   71.000000  0.863946  441.000000    441.000000
```

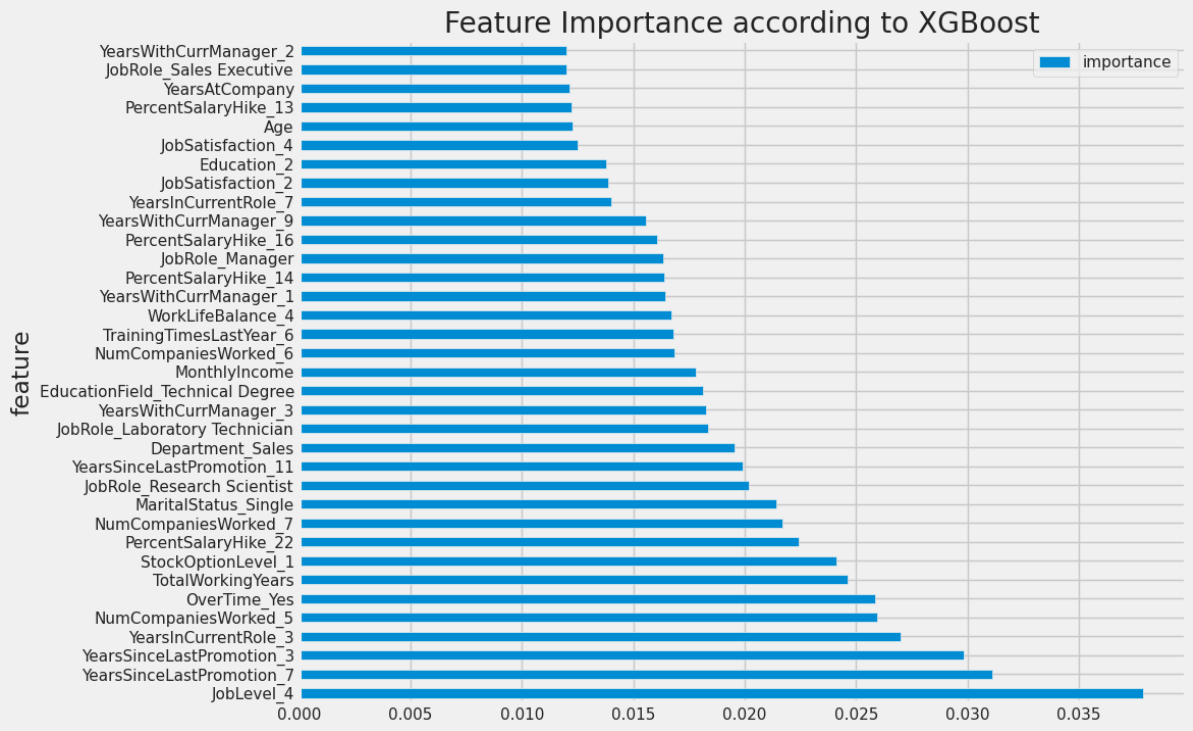
In [118]:
```
scores_dict['XGBoost'] = {
        'Train': roc_auc_score(y_train, xgb_clf.predict(X_train)),
        'Test': roc_auc_score(y_test, xgb_clf.predict(X_test)),
    }
```

In [119]:
```
precisions, recalls, thresholds = precision_recall_curve(y_test, xgb_clf.predi
ct(X_test))
plt.figure(figsize=(14, 25))
plt.subplot(4, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)

plt.subplot(4, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

plt.subplot(4, 2, 3)
fpr, tpr, thresholds = roc_curve(y_test, xgb_clf.predict(X_test))
plot_roc_curve(fpr, tpr)
```



In [120]:
```
df = feature_imp(X, xgb_clf)[:35]
df.set_index('feature', inplace=True)
df.plot(kind='barh', figsize=(10, 8))
plt.title('Feature Importance according to XGBoost')
```

Out[120]:
```
Text(0.5, 1.0, 'Feature Importance according to XGBoost')
```



COMPARING MODEL PERFORMANCE

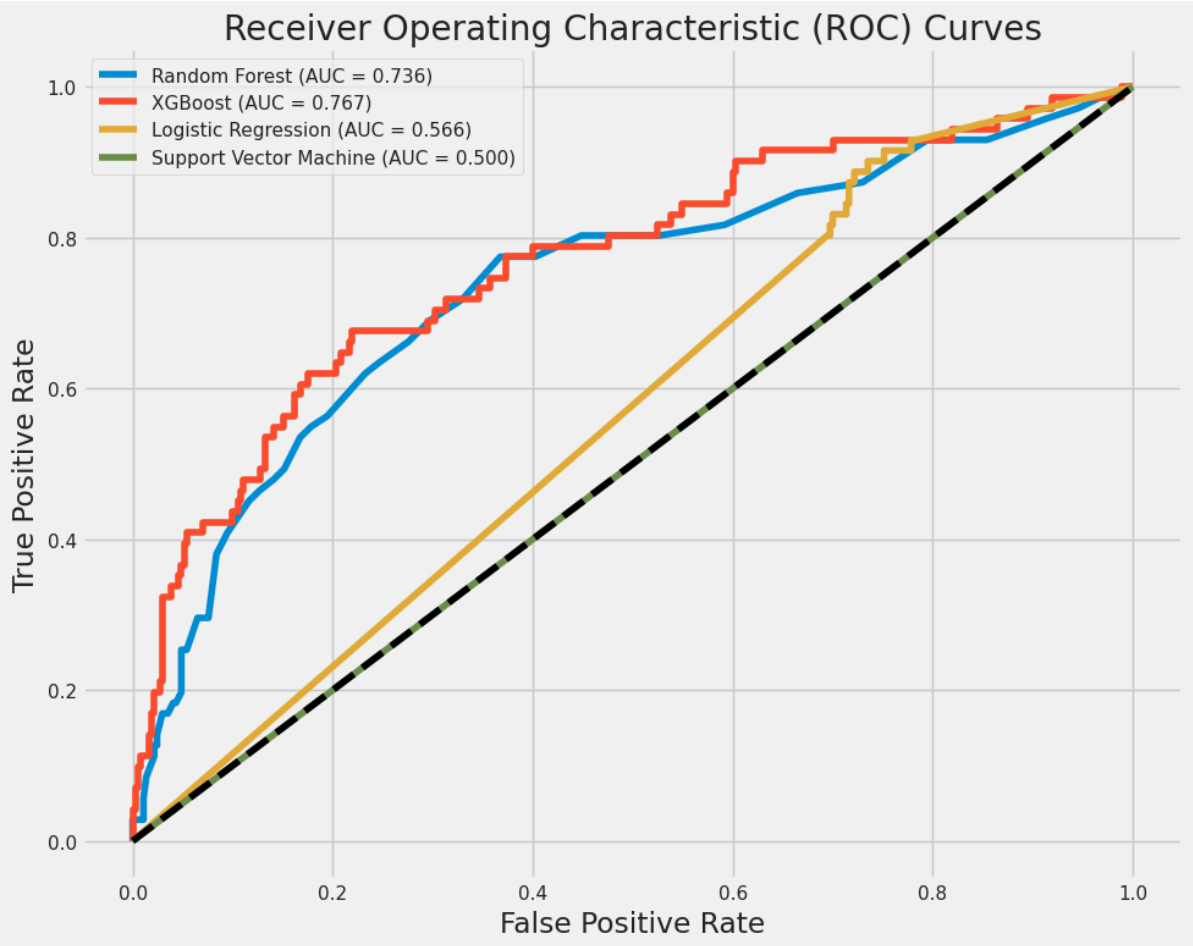
In [128]:
```
ml_models = {
    'Random Forest': rf_clf,
    'XGBoost': xgb_clf,
    'Logistic Regression': lr_clf,
    'Support Vector Machine': svm_clf,
}

for model in ml_models:
    print(f"{model.upper():{30}} roc_auc_score: {roc_auc_score(y_test, ml_mode
ls[model].predict(X_test)):.3f}")
```

```
RANDOM FOREST                  roc_auc_score: 0.551
XGBOOST                        roc_auc_score: 0.646
LOGISTIC REGRESSION            roc_auc_score: 0.560
SUPPORT VECTOR MACHINE         roc_auc_score: 0.500

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but LogisticRegression was fitted without feature names

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but SVC was fitted without feature names
```

In [129]:

```
plt.figure(figsize=(10, 8))

# Iterate through each model
for model_name, model in ml_models.items():
    if hasattr(model, "predict_proba"):  # Check if the model has predict_prob
a method
        # Predict probabilities for positive class
        y_pred_prob = model.predict_proba(X_test)[:, 1]
    else:
        # For models without predict_proba, use decision_function or predict
        y_pred_prob = model.decision_function(X_test) if hasattr(model, "decis
ion_function") else model.predict(X_test)

    # Compute ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

    # Compute AUC score
    auc_score = roc_auc_score(y_test, y_pred_prob)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {auc_score:.3f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend()
plt.grid(True)
plt.tight_layout()

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but LogisticRegression was fitted without feature names

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning:

X has feature names, but SVC was fitted without feature names
```



To determine which model is better, we typically look at the Area Under the Receiver Operating Characteristic Curve (AUC-ROC). A higher AUC value indicates better performance of the model in distinguishing between classes.

Based on the provided AUC values:

XGBoost (AUC=0.767)

Random Forest (AUC=0.736)

Logistic Regression (AUC=0.566)

Support Vector Machine (AUC=0.500)

From these values, we can see that AdaBoost has the highest AUC, followed by XGBoost and Random Forest. Therefore, based solely on the AUC values, AdaBoost would be considered the best-performing model among those listed. AdaBoost,XGBoost,and Random Forest models have higher AUC scores, indicating better performance in distinguishing between positive and negative instances. Higher AUC scores suggest that the model has a better ability to rank positive instances higher than negative instances across different threshold values.

In [ ]: