In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

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

Out[2]:

kOptionLevel	TotalWorkingYears	TrainingTimesLastYear	WorkLifeBalance	YearsAtCompany	YearsInCurrentRole	Yea
0	8	0	1	6	4	
1	10	3	3	10	7	
0	7	3	3	0	0	
0	8	3	3	8	7	
1	6	3	3	2	2	

Data Preprocessing Part 1

In [3]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
Attrition         2
BusinessTravel    3
Department        3
EducationField    6
Gender            2
JobRole           9
MaritalStatus     3
Over18            1
OverTime          2
dtype: int64
```

In [4]:
```
# Drop column with only 1 unique value
df.drop(columns = 'Over18', inplace=True)
```

In [5]:
```
df.nunique()
```

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

In [6]:
```
# Delete other columns with only 1 unique value
df.drop(columns = ['EmployeeCount', 'StandardHours'], inplace=True)
```

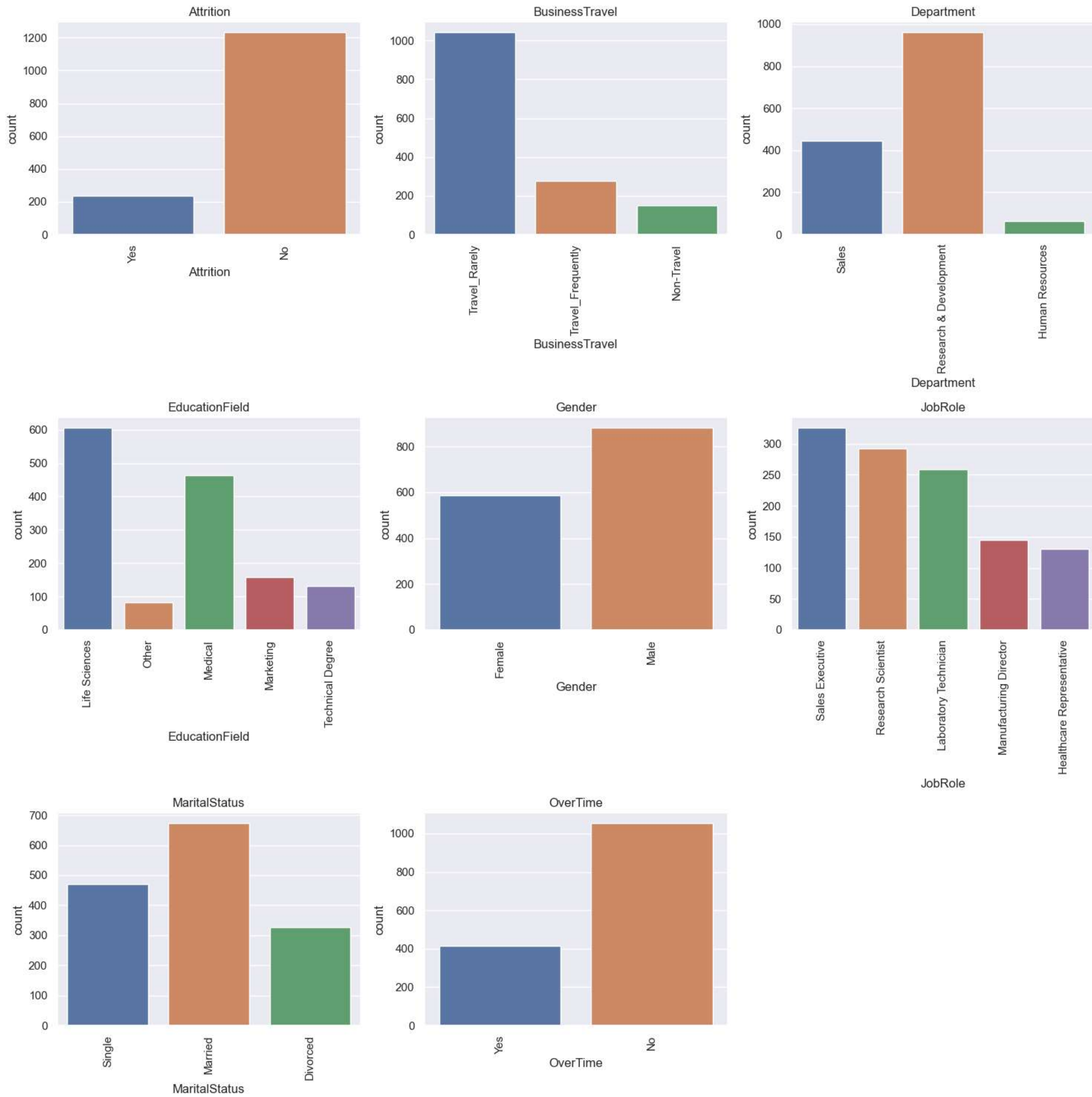
In [7]:
```
df.head()
```

Out[7]:

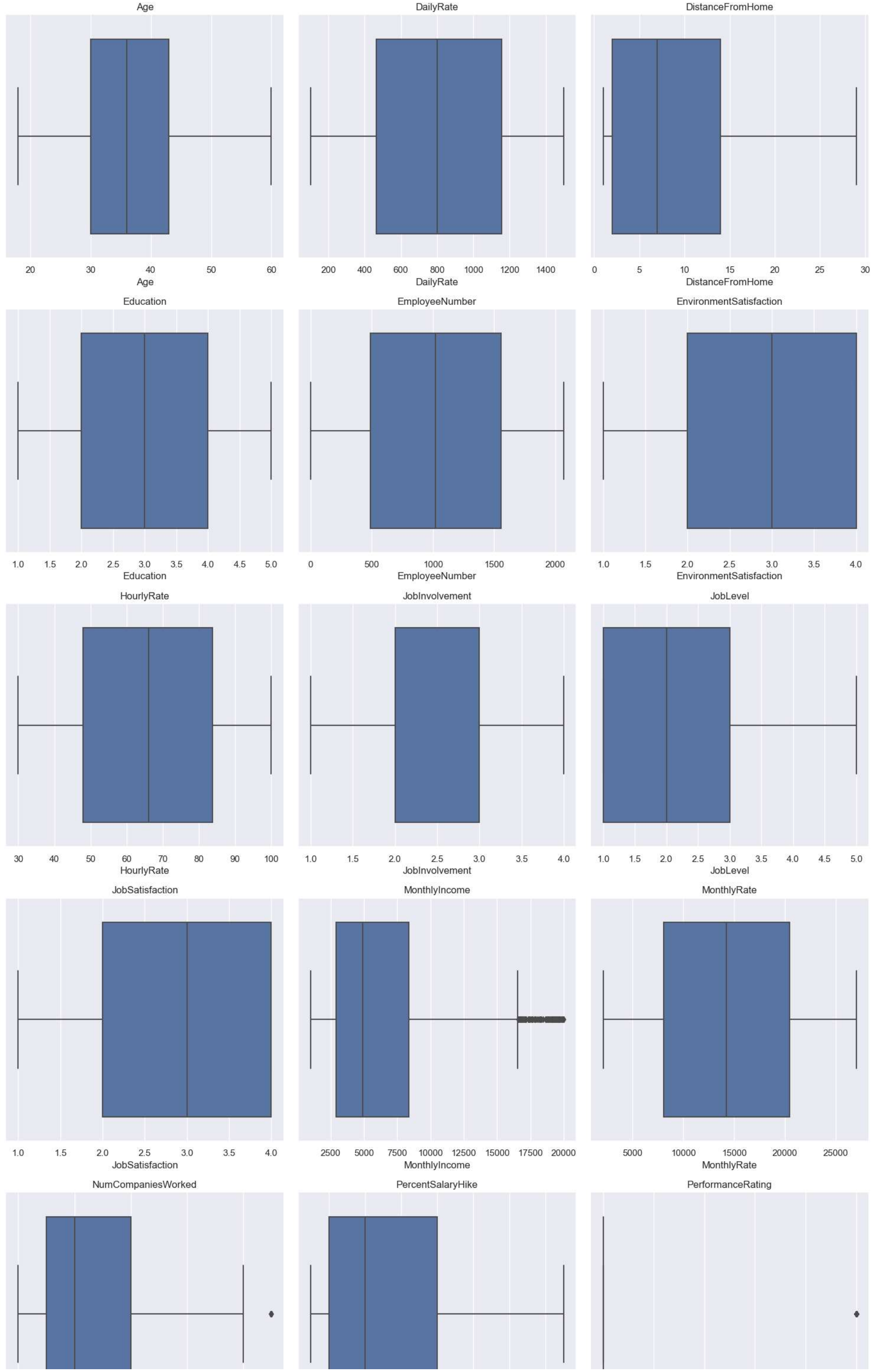
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Employ
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	
3	33	No	Travel_Frequently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

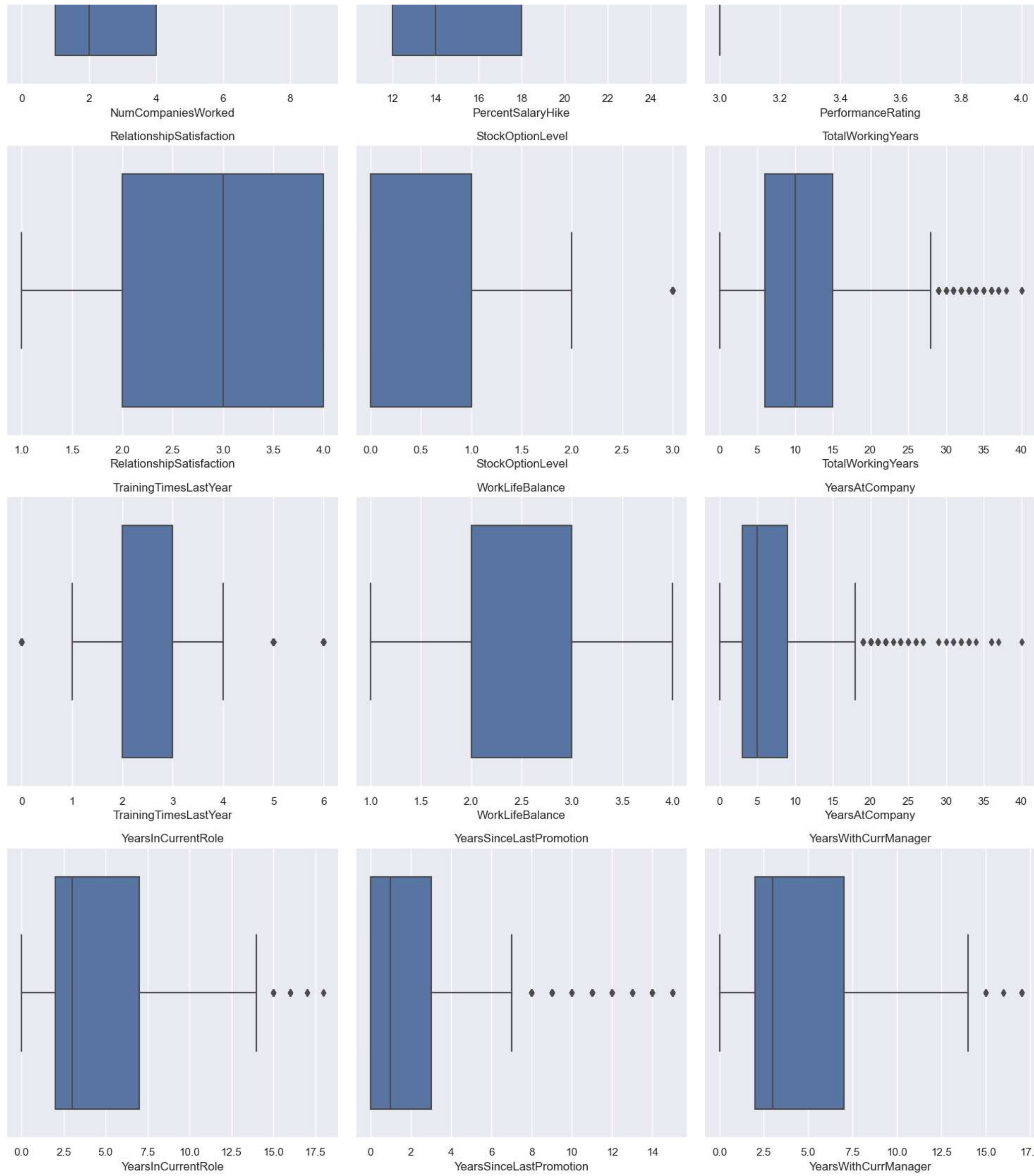
Exploratory Data Analysis

In [8]:
```
# Get the names of all columns with data type 'object' (categorical columns)
cat_vars = df.select_dtypes(include='object').columns.tolist()

# Create a figure with subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a countplot for the top 5 values of each categorical variable using Seaborn
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(5).index
    filtered_df = df[df[var].isin(top_values)]
    sns.countplot(x=var, data=filtered_df, ax=axs[i])
    axs[i].set_title(var)
    axs[i].tick_params(axis='x', rotation=90)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

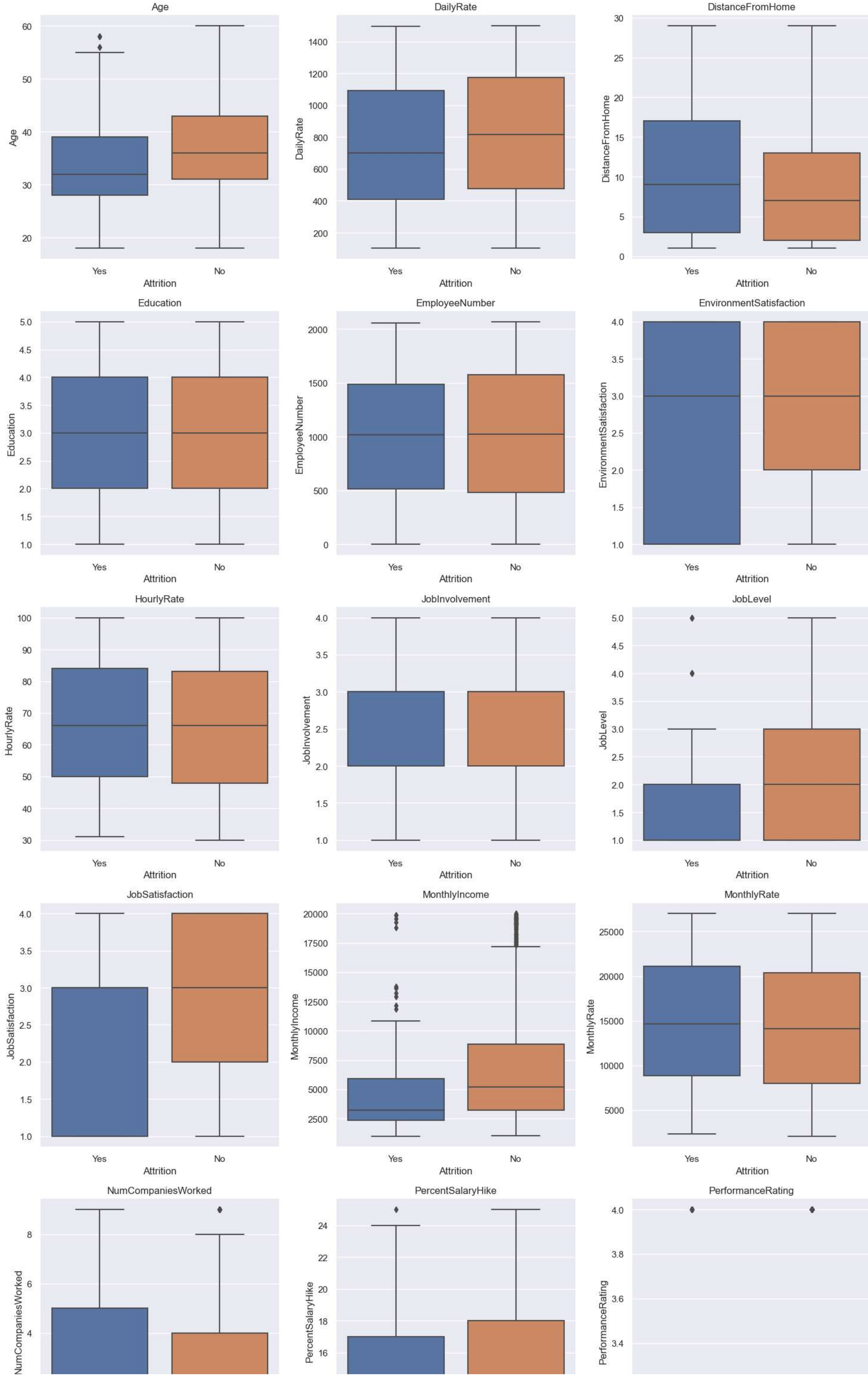

Attrition
count
Yes
No
Attrition
BusinessTravel
Travel_Rarely
Travel_Frequently
Non-Travel
Department
Sales
Research & Development
Human Resources
EducationField
Life Sciences
Other
Medical
Marketing
Technical Degree
Gender
Female
Male
JobRole
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
MaritalStatus
Single
Married
Divorced
OverTime

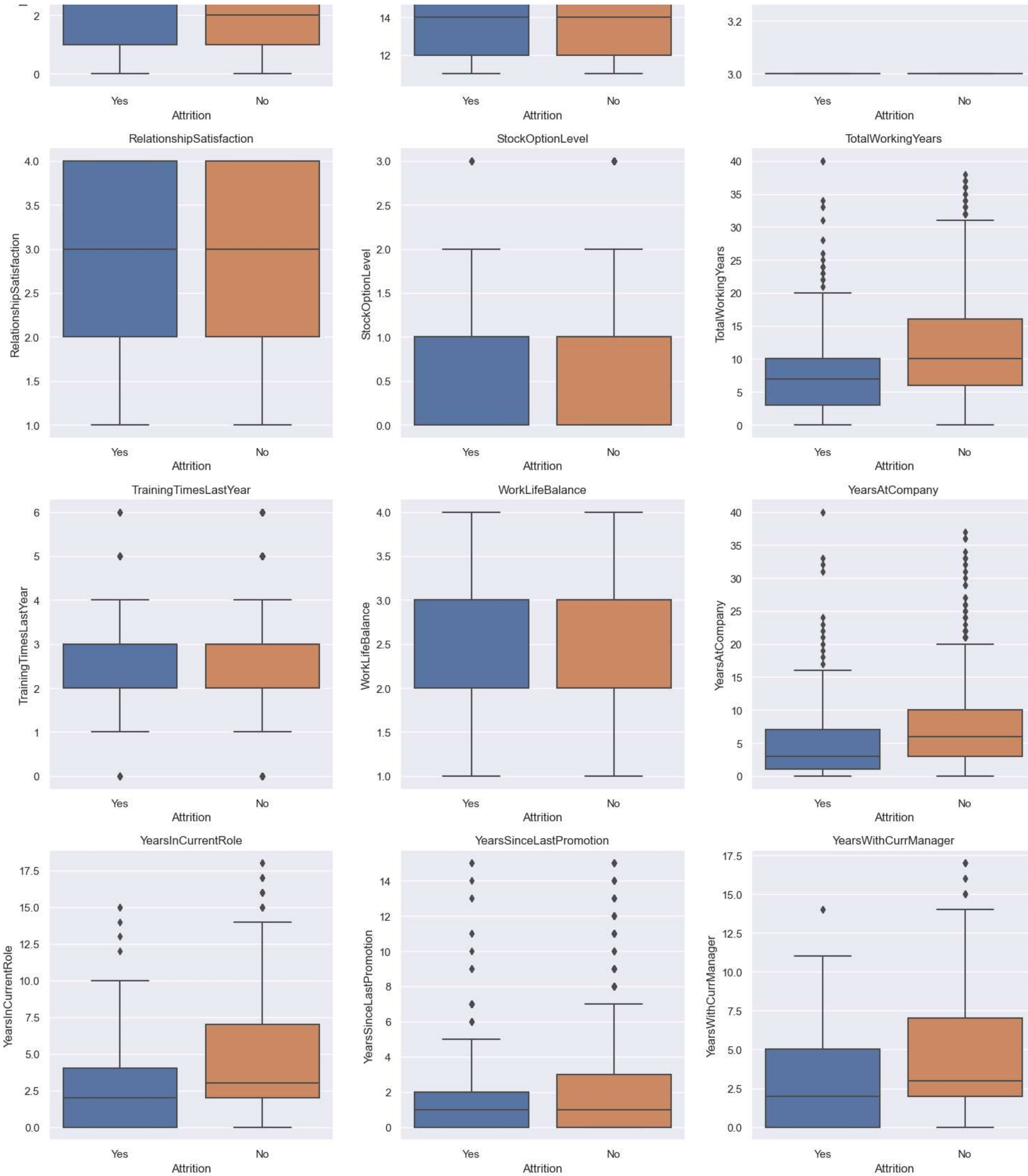
In [9]:
```
# Get the names of all columns with data type 'int' or 'float'
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(num_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
for i, var in enumerate(num_vars):
    sns.boxplot(x=df[var], ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Age
20 30 40 50 60
Age
DailyRate
200 400 600 800 1000 1200 1400
DailyRate
DistanceFromHome
0 5 10 15 20 25 30
DistanceFromHome
Education
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
Education
EmployeeNumber
0 500 1000 1500 2000
EmployeeNumber
EnvironmentSatisfaction
1.0 1.5 2.0 2.5 3.0 3.5 4.0
EnvironmentSatisfaction
HourlyRate
30 40 50 60 70 80 90 100
HourlyRate
JobInvolvement
1.0 1.5 2.0 2.5 3.0 3.5 4.0
JobInvolvement
JobLevel
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
JobLevel
JobSatisfaction
1.0 1.5 2.0 2.5 3.0 3.5 4.0
JobSatisfaction
MonthlyIncome
2500 5000 7500 10000 12500 15000 17500 20000
MonthlyIncome
MonthlyRate
5000 10000 15000 20000 25000
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating

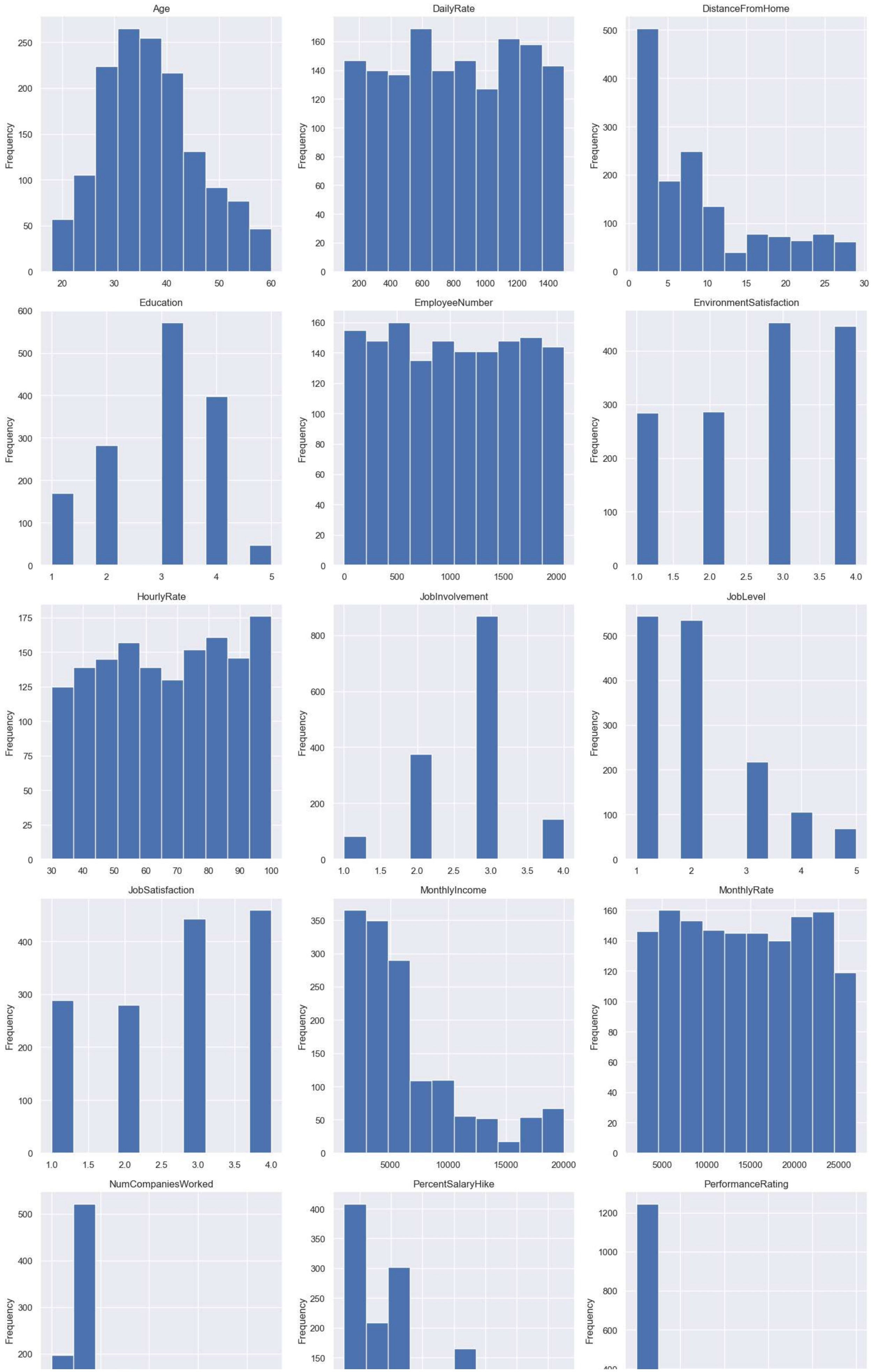

NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager

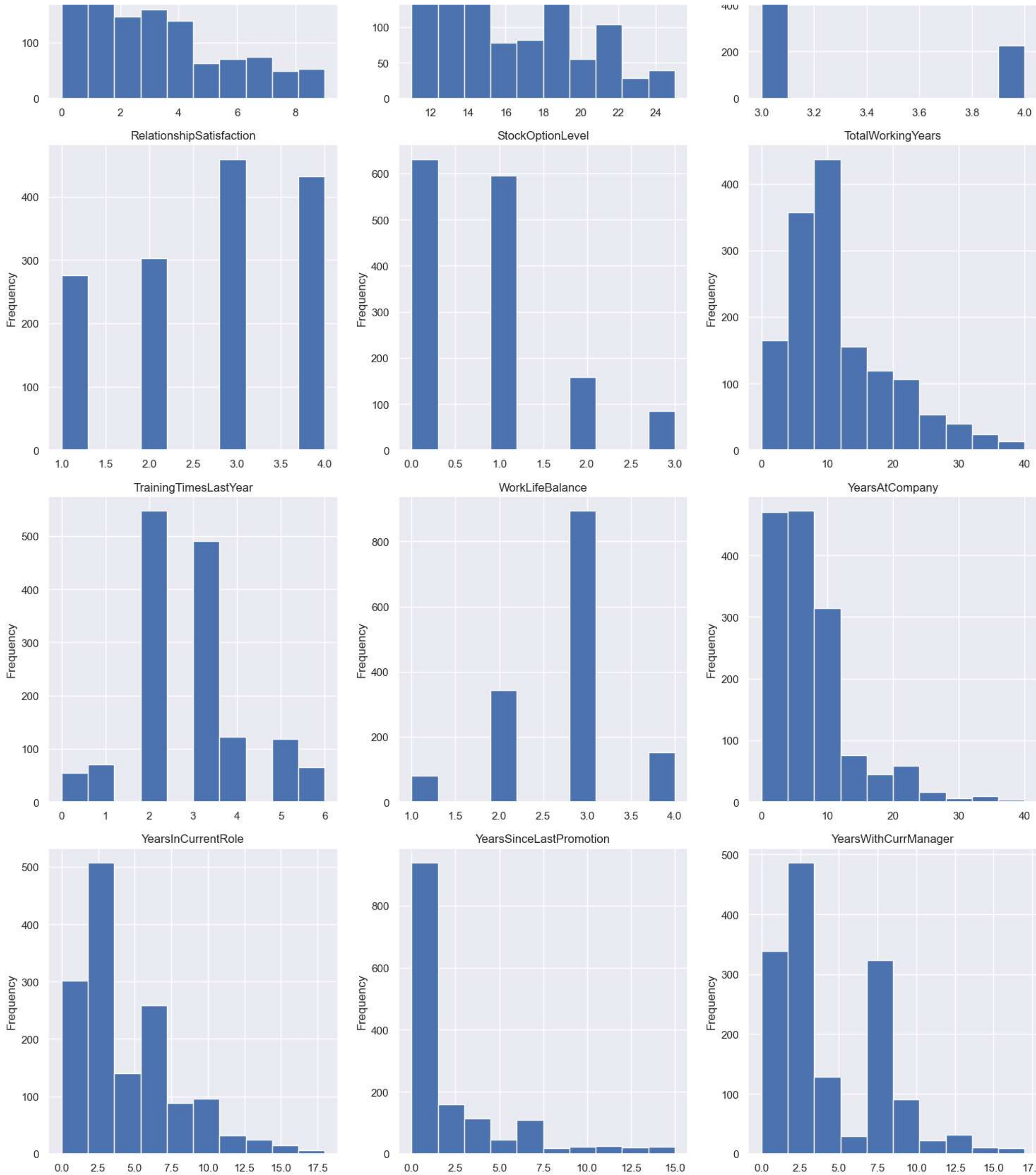
In [11]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each integer variable using Seaborn with hue='attrition'
for i, var in enumerate(int_vars):
    sns.boxplot(y=var, x='Attrition', data=df, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Age
Age
Attrition
Yes
No
DailyRate
DailyRate
Attrition
DistanceFromHome
DistanceFromHome
Attrition
Education
Education
Attrition
EmployeeNumber
EmployeeNumber
Attrition
EnvironmentSatisfaction
EnvironmentSatisfaction
Attrition
HourlyRate
HourlyRate
Attrition
JobInvolvement
JobInvolvement
Attrition
JobLevel
JobLevel
Attrition
JobSatisfaction
JobSatisfaction
Attrition
MonthlyIncome
MonthlyIncome
Attrition
MonthlyRate
MonthlyRate
Attrition
NumCompaniesWorked
NumCompaniesWorked
PercentSalaryHike
PercentSalaryHike
PerformanceRating
PerformanceRating

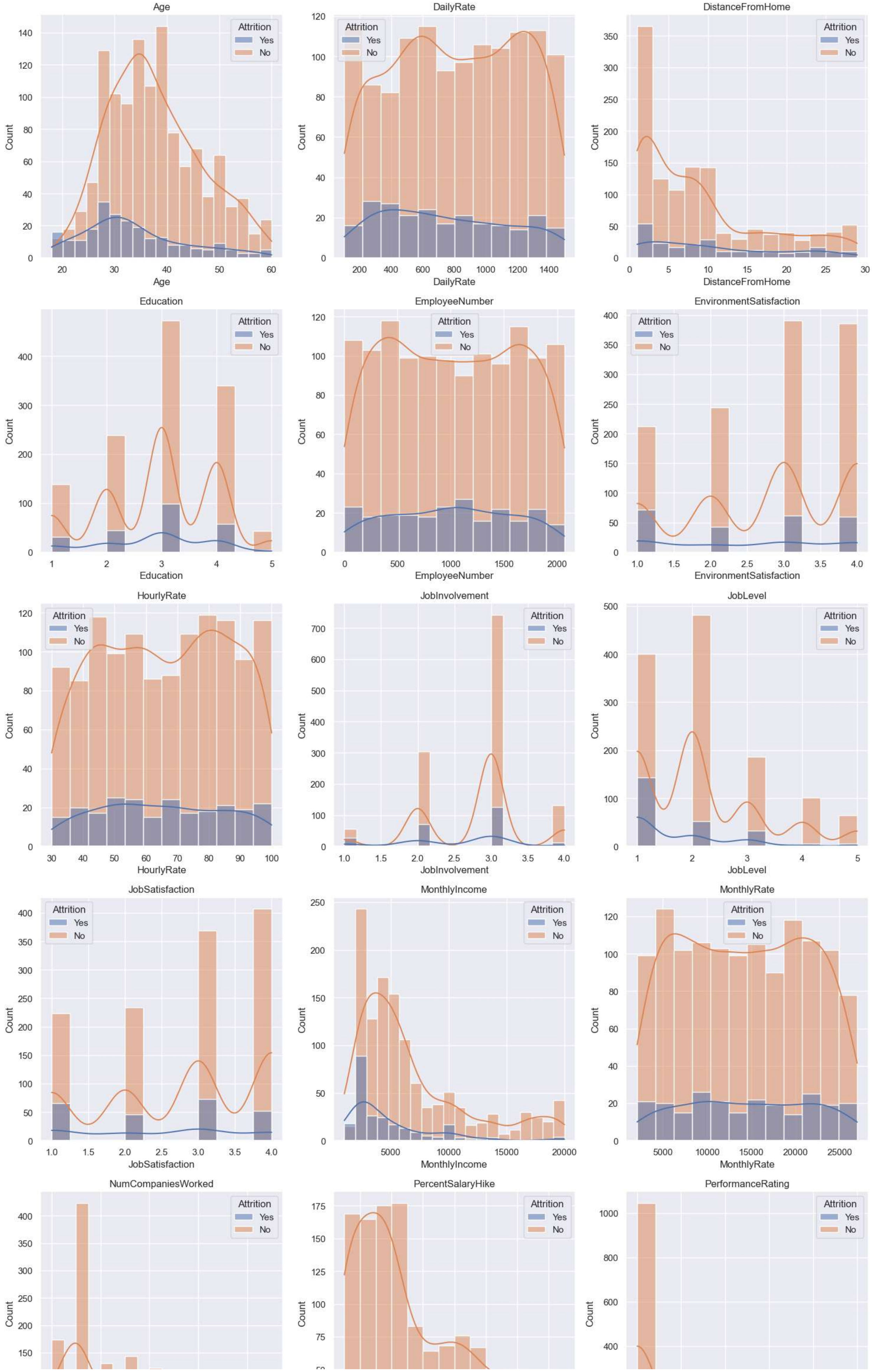

Attrition
Yes
No
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager

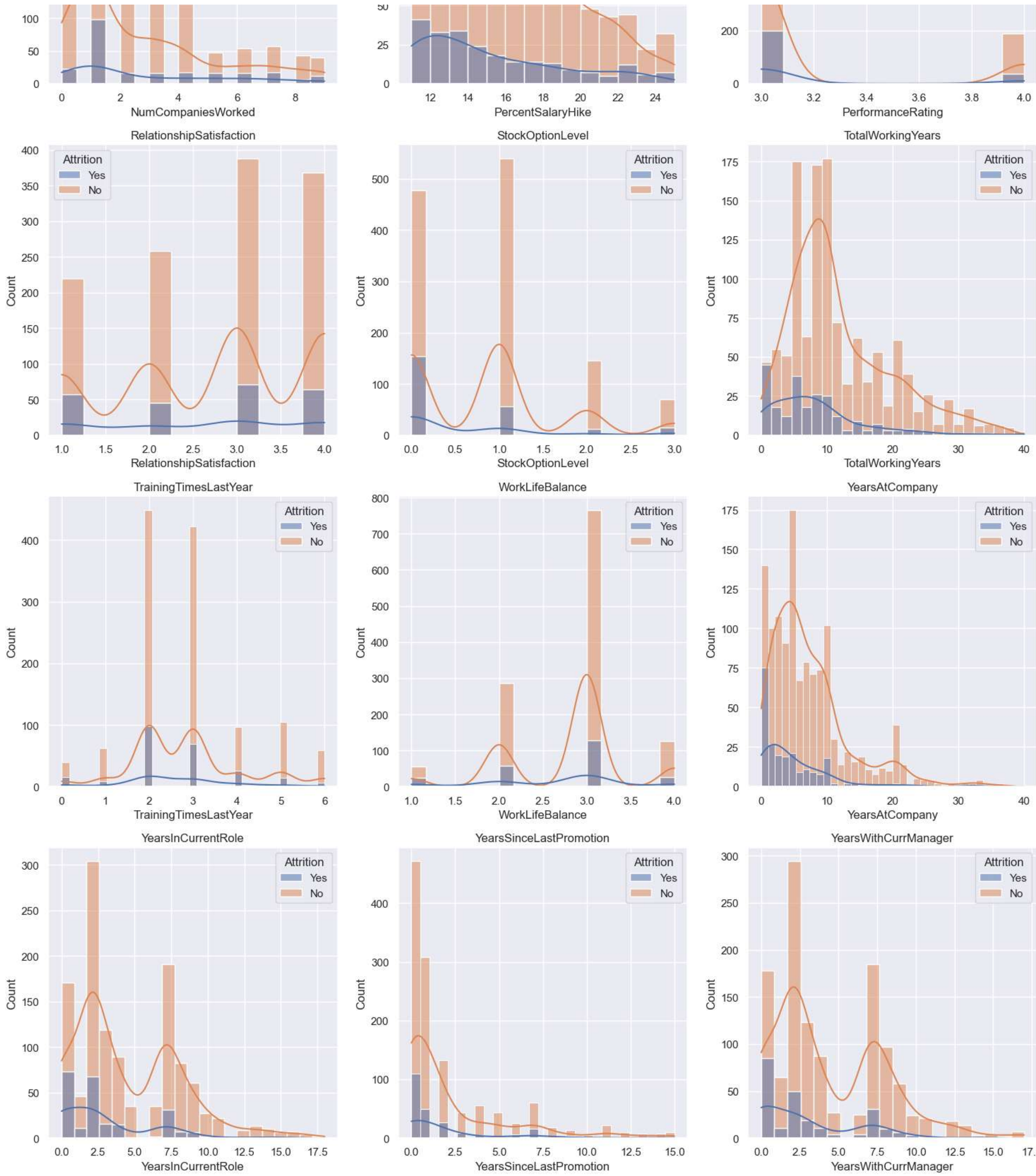
In [12]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable
for i, var in enumerate(int_vars):
    df[var].plot.hist(ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Age
DailyRate
DistanceFromHome
Education
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating

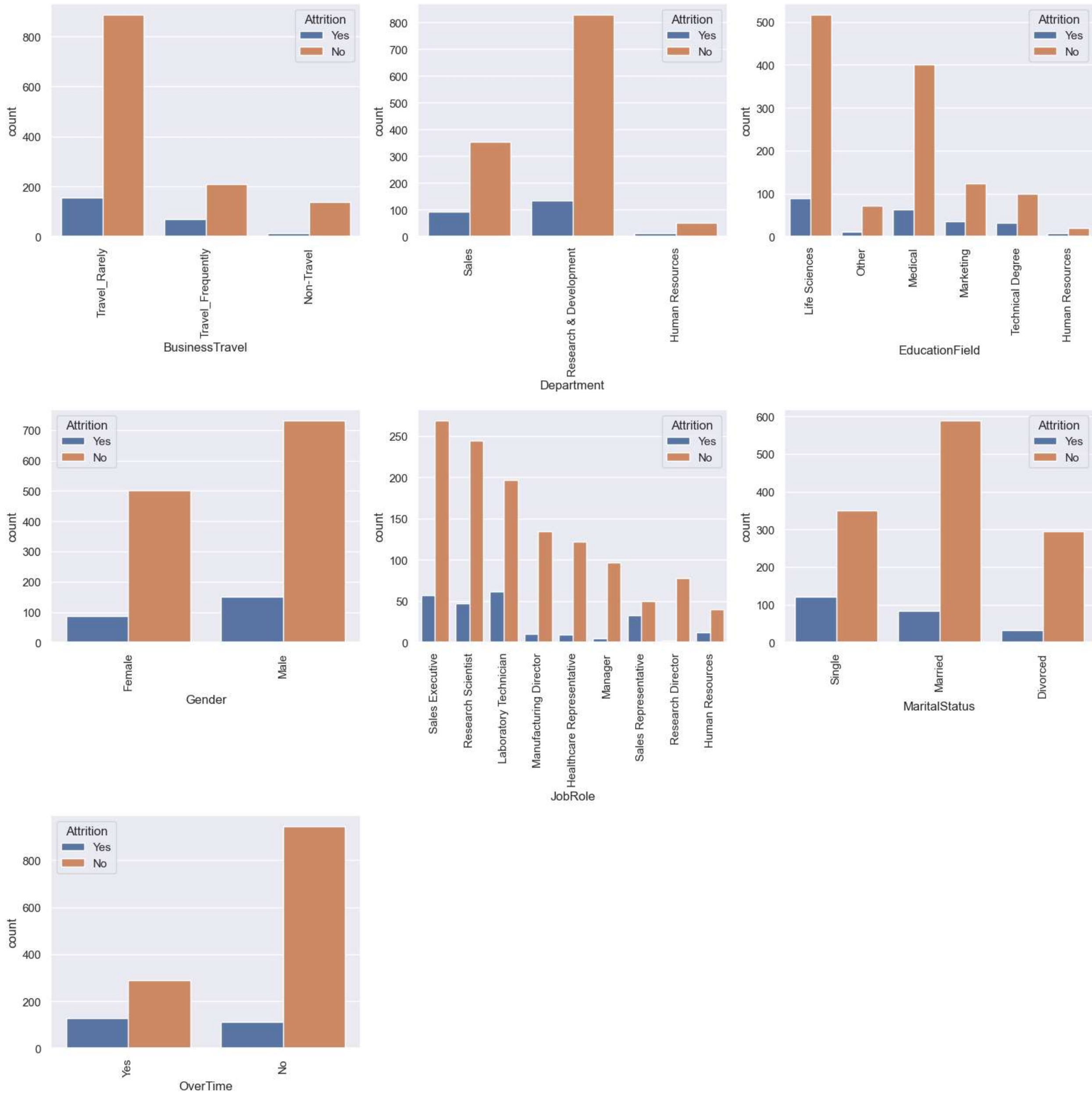

RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
Frequency

In [13]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    sns.histplot(data=df, x=var, hue='Attrition', kde=True, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Age
DailyRate
DistanceFromHome
Education
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating
Count
Attrition
Yes
No

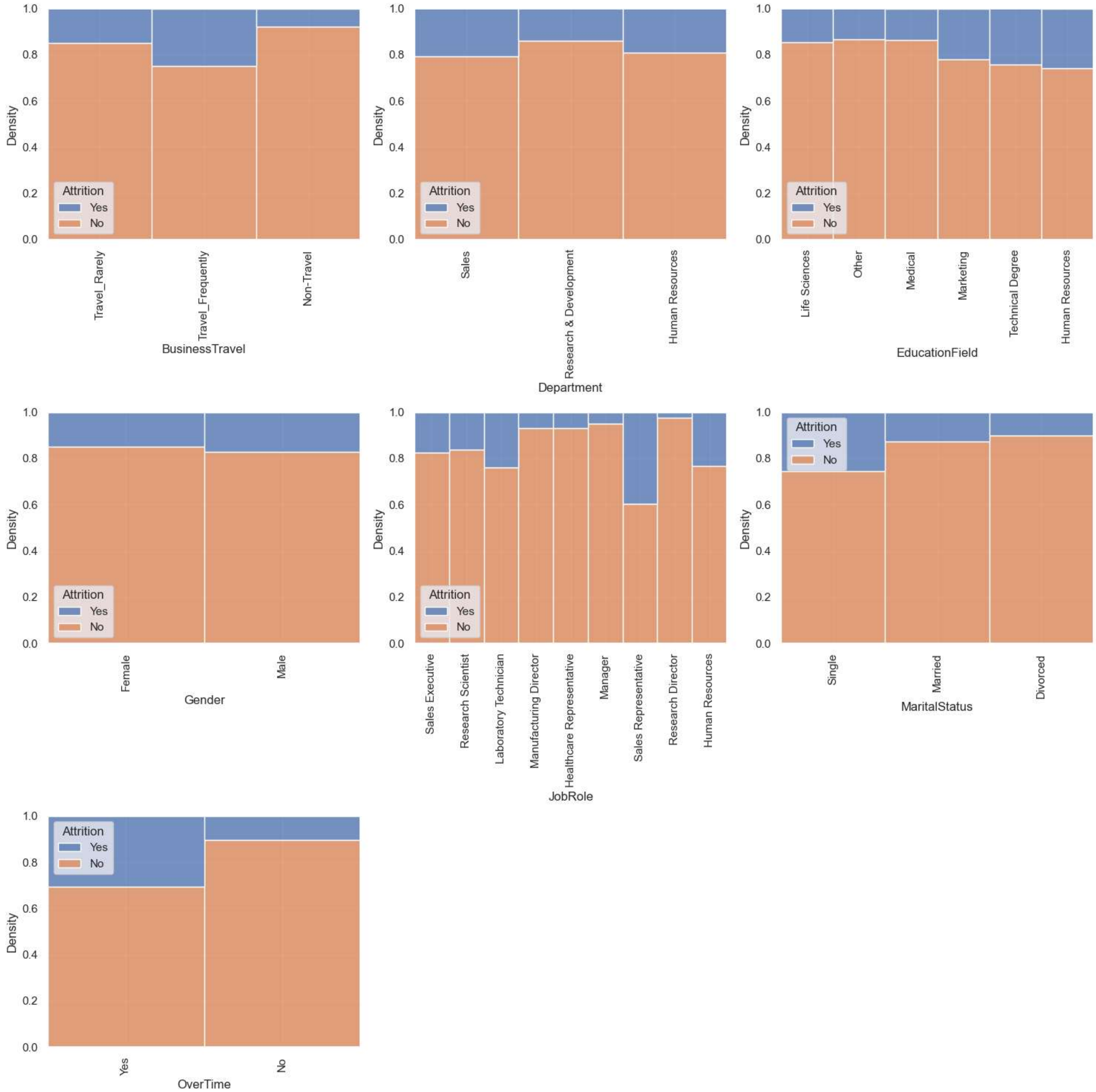

NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
Count
Attrition
Yes
No

In [16]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()

# Exclude 'Attrition' from the list if it exists in cat_vars
if 'Attrition' in cat_vars:
    cat_vars.remove('Attrition')

# Create a figure with subplots, but only include the required number of subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='Attrition', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show the plot
plt.show()
```


Attrition
Yes
No
count
Travel_Rarely
Travel_Frequently
Non-Travel
BusinessTravel
Sales
Research & Development
Human Resources
Department
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources
EducationField
Female
Male
Gender
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
Manager
Sales Representative
Research Director
Human Resources
JobRole
Single
Married
Divorced
MaritalStatus
Yes
No
OverTime

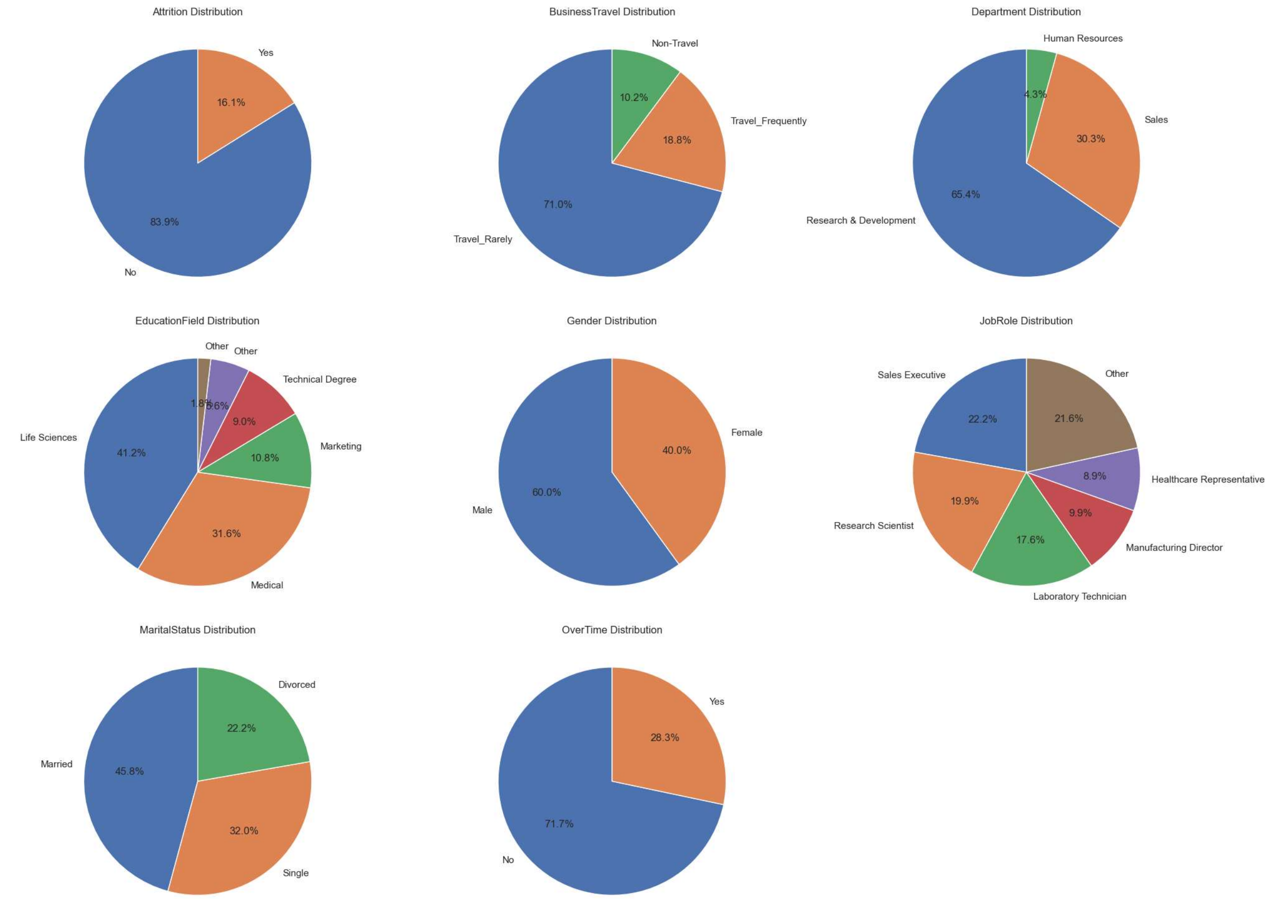
In [17]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()

# Exclude 'Attrition' from the list if it exists in cat_vars
if 'Attrition' in cat_vars:
    cat_vars.remove('Attrition')

# Create a figure with subplots, but only include the required number of subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Attrition', data=df, ax=axs[i], multiple="fill", kde=False, ele
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show the plot
plt.show()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\1288156870.py:17: UserWarning: FixedFor
matter should only be used together with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
```


Density
Attrition
Yes
No
Travel_Rarely
Travel_Frequently
Non-Travel
BusinessTravel
Sales
Research & Development
Human Resources
Department
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources
EducationField
Female
Male
Gender
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
Manager
Sales Representative
Research Director
Human Resources
JobRole
Single
Married
Divorced
MaritalStatus
Yes
No
OverTime

In [21]:
```python
# Specify the maximum number of categories to show individually
max_categories = 5

# Filter categorical columns with 'object' data type
cat_cols = [col for col in df.columns if col != 'y' and df[col].dtype == 'object']

# Create a figure with subplots
num_cols = len(cat_cols)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(20, 5*num_rows))

# Flatten the axs array for easier indexing
axs = axs.flatten()

# Create a pie chart for each categorical column
for i, col in enumerate(cat_cols):
    if i < len(axs):  # Ensure we don't exceed the number of subplots
        # Count the number of occurrences for each category
        cat_counts = df[col].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other'])
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)
        axs[i].set_title(f'{col} Distribution')

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\943223290.py:25: FutureWarning: The ser
ies.append method is deprecated and will be removed from pandas in a future version. Use pa
ndas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_24776\943223290.py:25: FutureWarning: The ser
ies.append method is deprecated and will be removed from pandas in a future version. Use pa
ndas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```



Data Preprocessing Part 2

In [22]:
```
# Check the amounnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[22]:
```
Series([], dtype: float64)
```

In [23]:
```
df.shape
```

Out[23]:
```
(1470, 32)
```

In [24]:
```
df.head()
```

Out[24]:

Education	EducationField	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	JobInvolvement	JobLevel
2	Life Sciences	1	2	Female	94	3	2
1	Life Sciences	2	3	Male	61	2	2
2	Other	4	4	Male	92	2	1
4	Life Sciences	5	4	Female	56	3	1
1	Medical	7	1	Male	40	3	1

Label Encoding for Object Datatypes

In [25]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

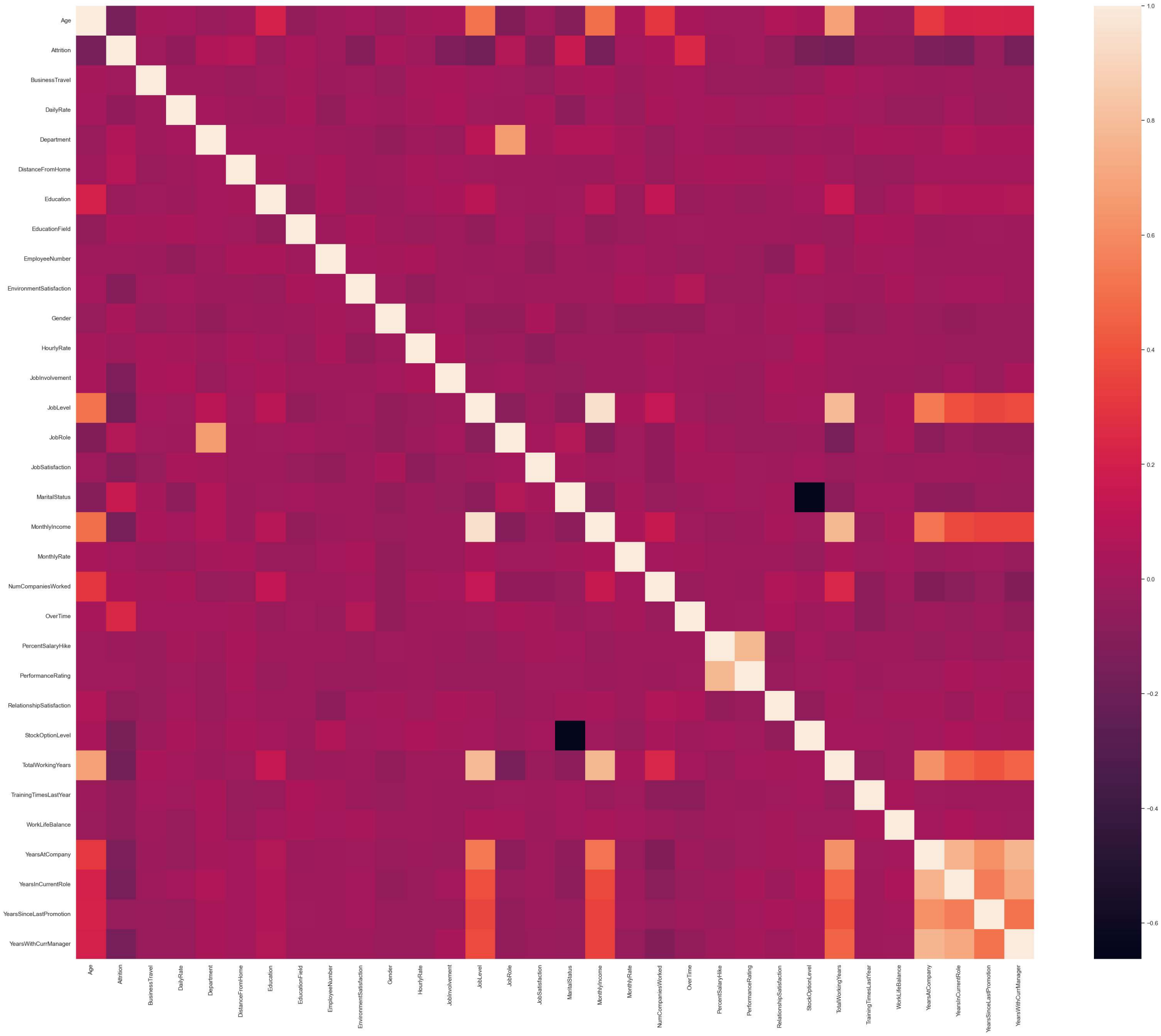
```
Attrition: ['Yes' 'No']
BusinessTravel: ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
Department: ['Sales' 'Research & Development' 'Human Resources']
EducationField: ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']
Gender: ['Female' 'Male']
JobRole: ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
MaritalStatus: ['Single' 'Married' 'Divorced']
OverTime: ['Yes' 'No']
```

In [26]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Attrition: [1 0]
BusinessTravel: [2 1 0]
Department: [2 1 0]
EducationField: [1 4 3 2 5 0]
Gender: [0 1]
JobRole: [7 6 2 4 0 3 8 5 1]
MaritalStatus: [2 1 0]
OverTime: [1 0]
```

In [27]:
```
# Correlation Heatmap
plt.figure(figsize=(40, 32))
sns.heatmap(df.corr(), fmt='.2g', annot=False)
```

Out[27]:
```
<AxesSubplot:>
```



In [28]:
```
# Remove JobLevel column because it have 100% correlation with monthly income
df.drop(columns = 'JobLevel', inplace=True)
```

Train Test Split

In [29]:
```
X = df.drop('Attrition', axis=1)
y = df['Attrition']
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Remove Outlier from Train Data using Z-Score

In [30]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['MonthlyIncome', 'TotalWorkingYears', 'YearsAtCompany',
                    'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Classifier

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 42}
```

In [32]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=42, max_depth=3, min_samples_leaf=1, min_samples_
dtree.fit(X_train, y_train)
```

Out[32]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=3, random_state=42)
```

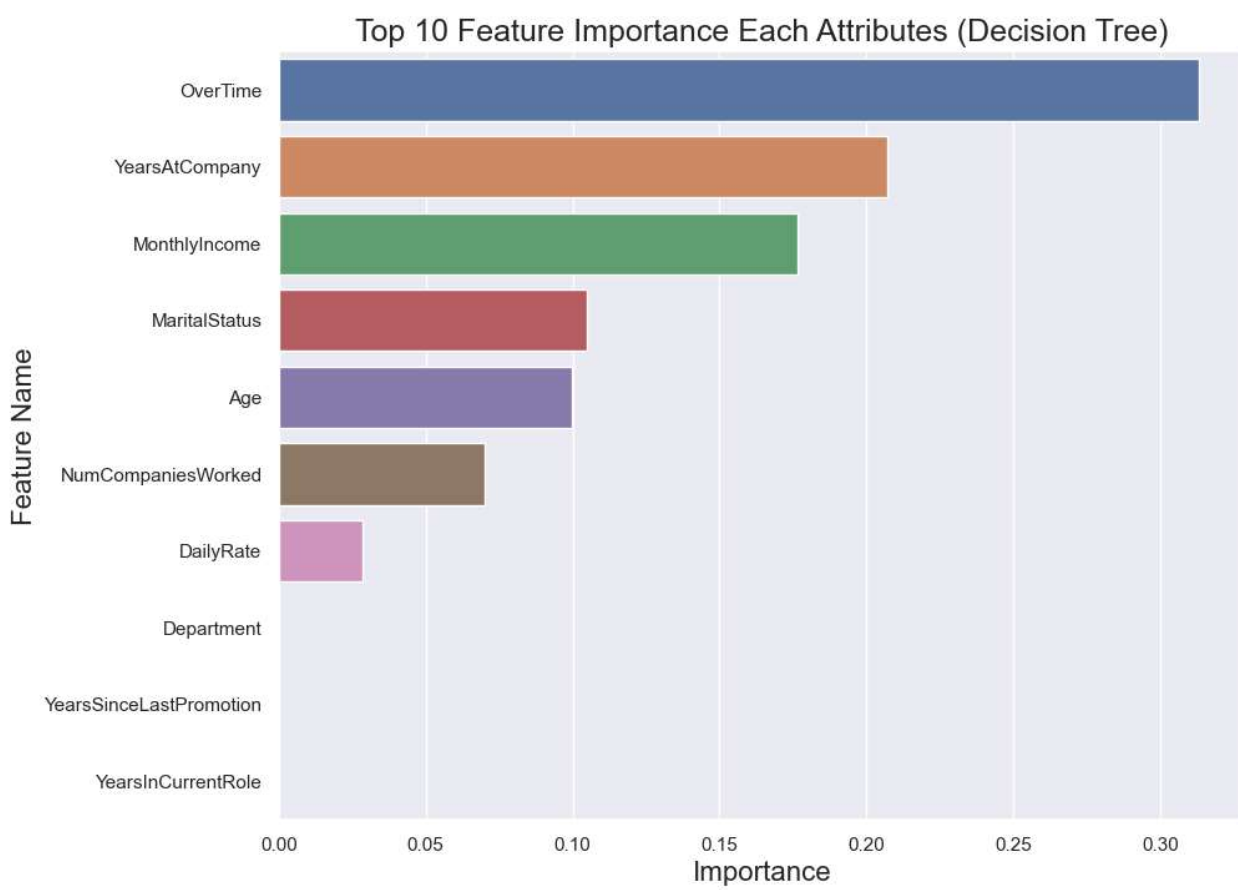
In [33]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 78.57 %
```

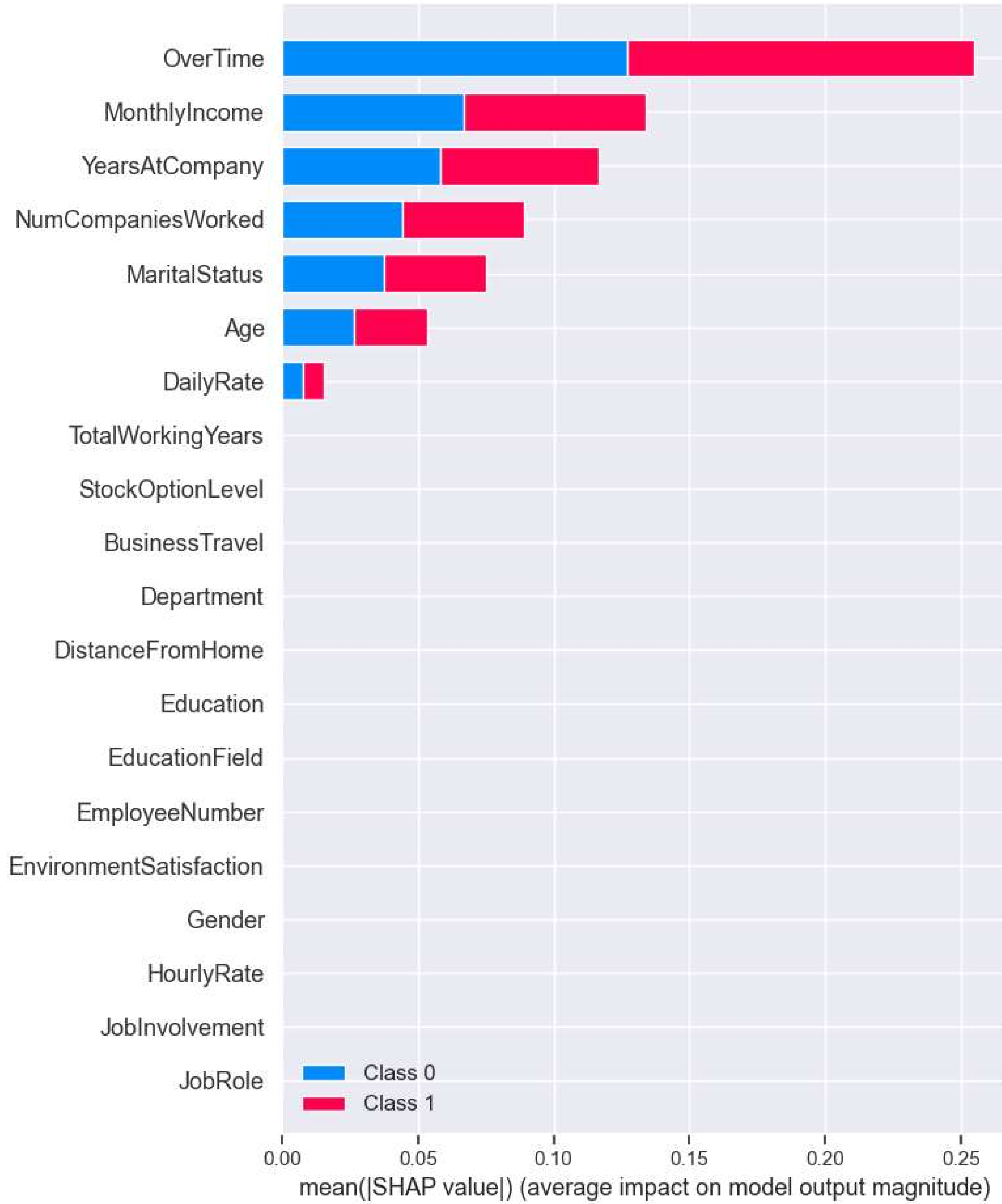
In [34]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.7857142857142857
Precision Score :  0.7857142857142857
Recall Score :  0.7857142857142857
Jaccard Score :  0.6470588235294118
Log Loss :  7.401264280227416
```

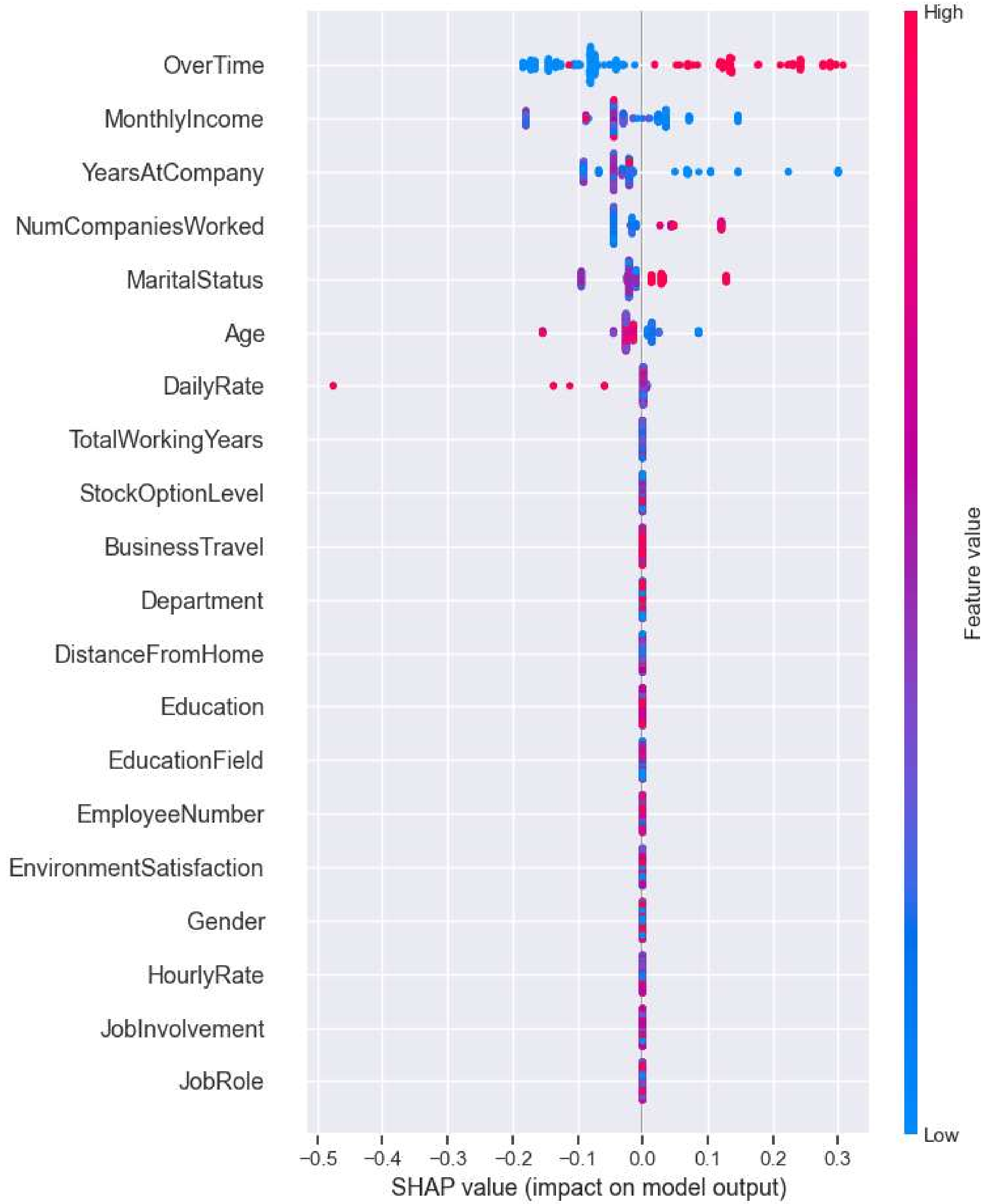
In [35]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [36]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

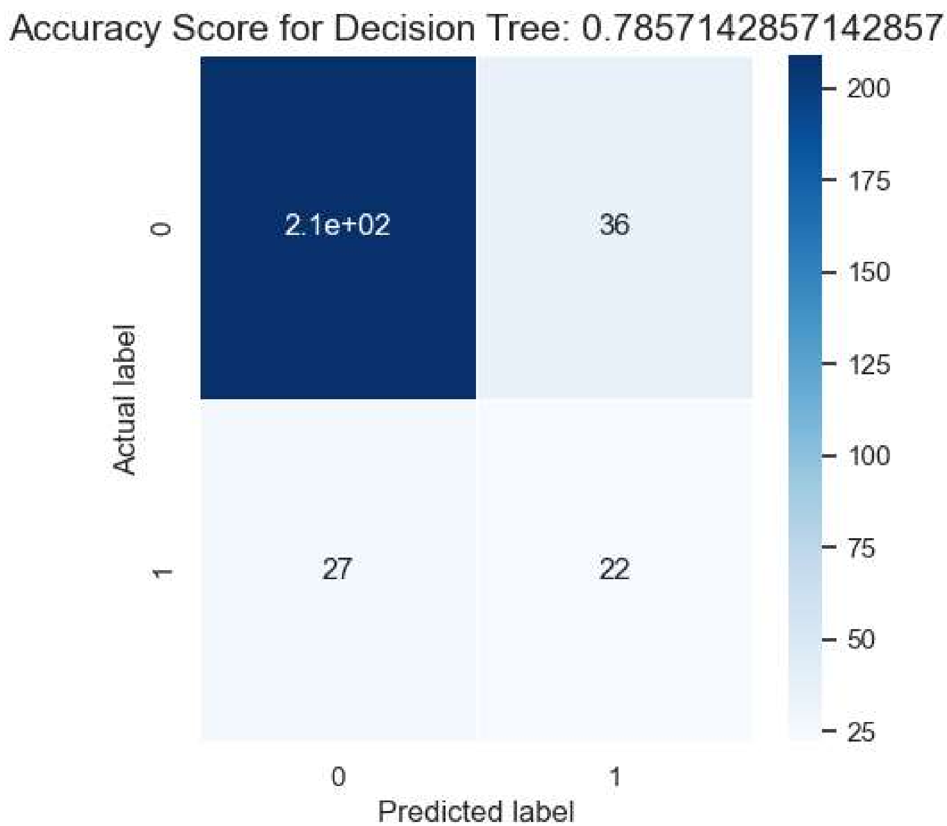


In [37]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



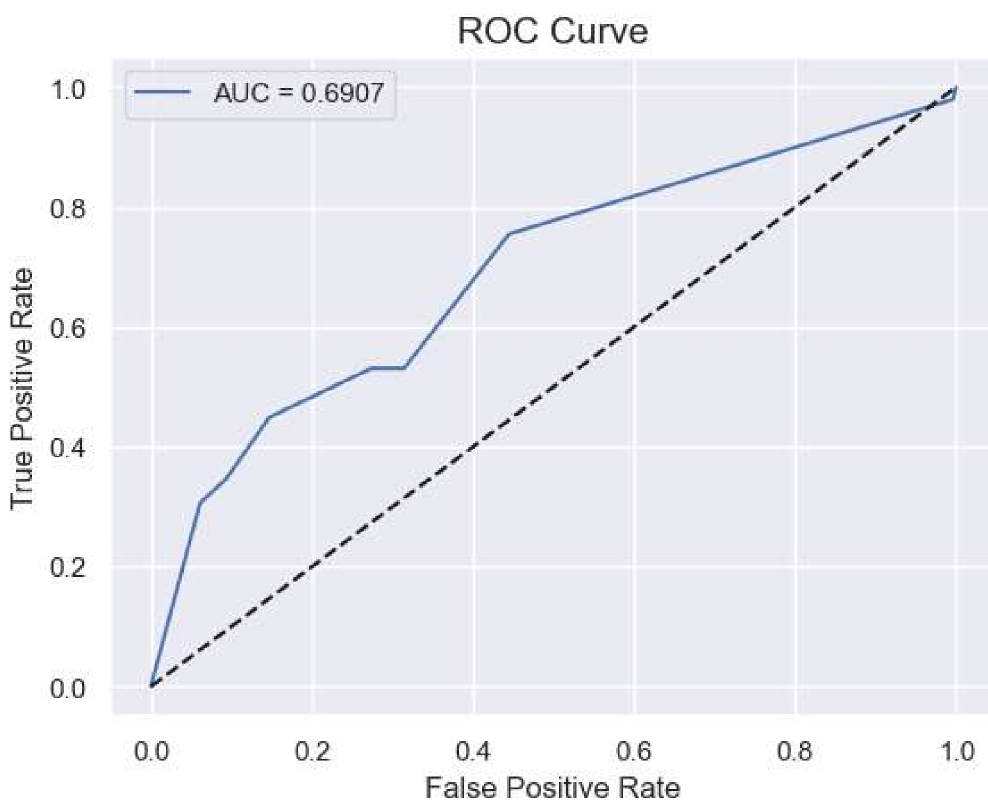
In [38]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test
plt.title(all_sample_title, size = 15)
```

Out[38]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.7857142857142857')
```



In [39]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.Da
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[39]: <matplotlib.legend.Legend at 0x1f2afe522b0>



Random Forest Classifier

In [40]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 5, 'max_features': 'sqrt', 'n_estimators': 200, 'random_state': 0}
```

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_depth=5, max_features='sqrt', n_estimators=
rfc.fit(X_train, y_train)
```

Out[41]:
```
RandomForestClassifier(class_weight='balanced', max_depth=5,
                       max_features='sqrt', n_estimators=200, random_state=0)
```

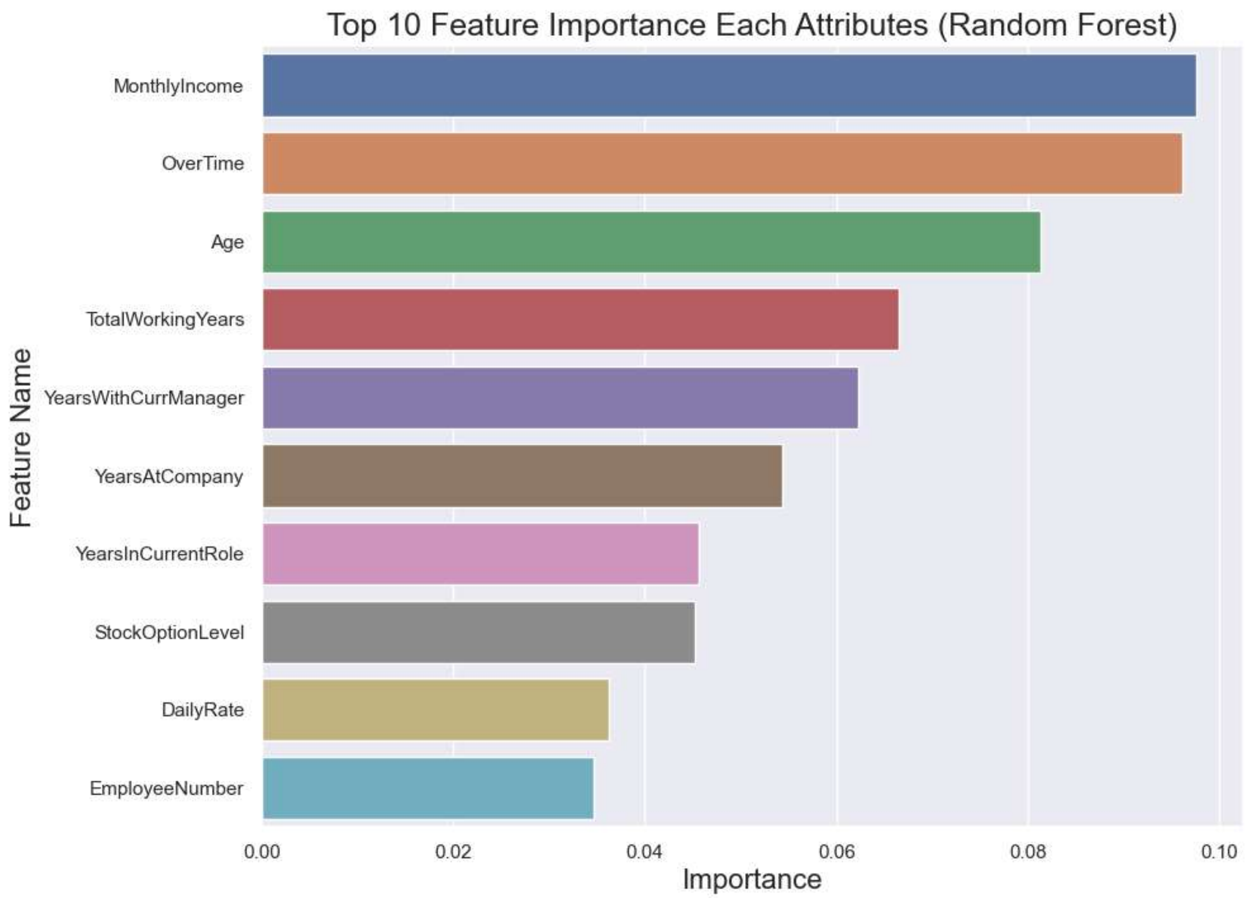
In [42]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 82.65 %
```

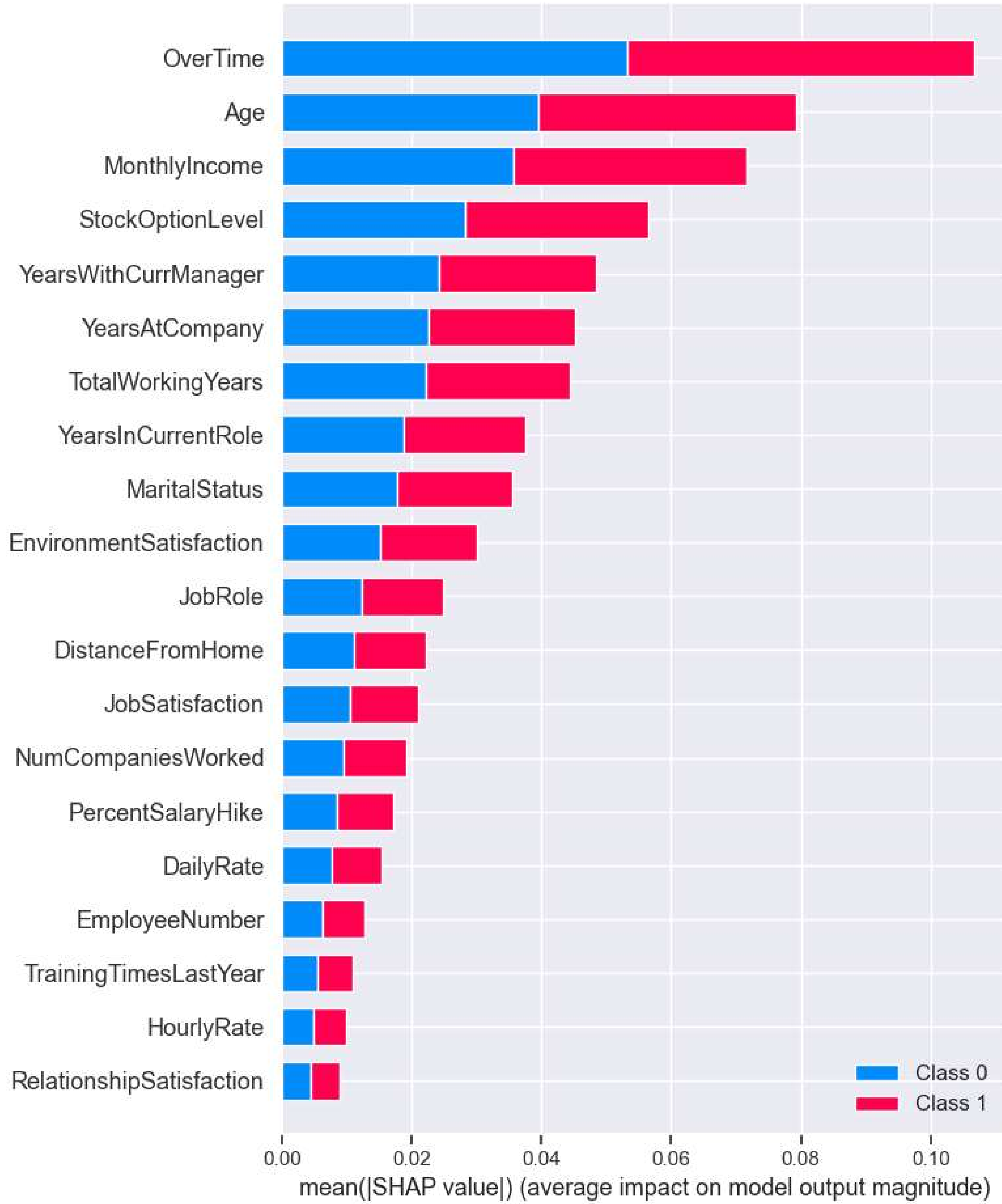
In [43]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.826530612244898
Precision Score :  0.826530612244898
Recall Score :  0.826530612244898
Jaccard Score :  0.7043478260869566
Log Loss :  5.991472069699383
```

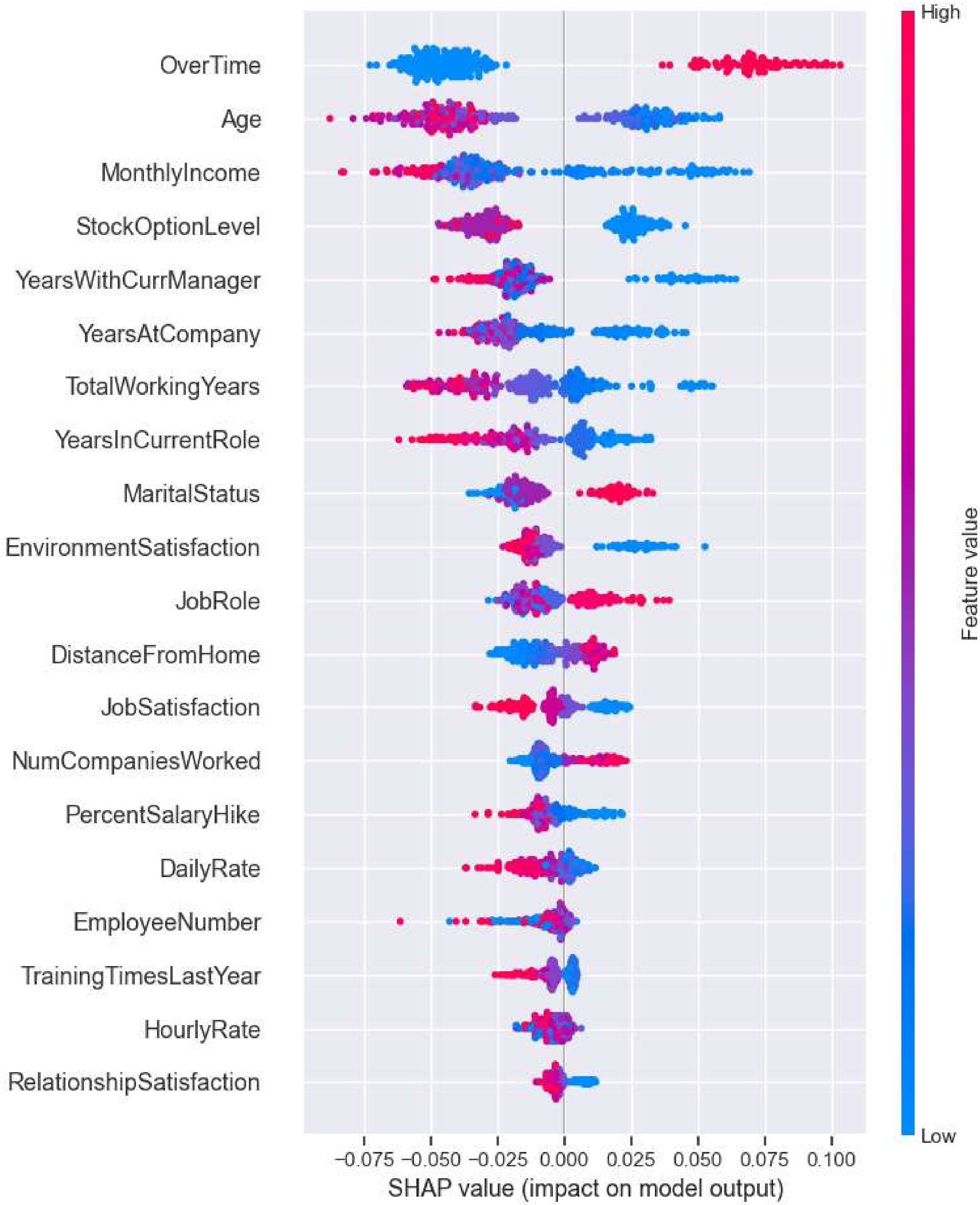
In [44]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [45]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

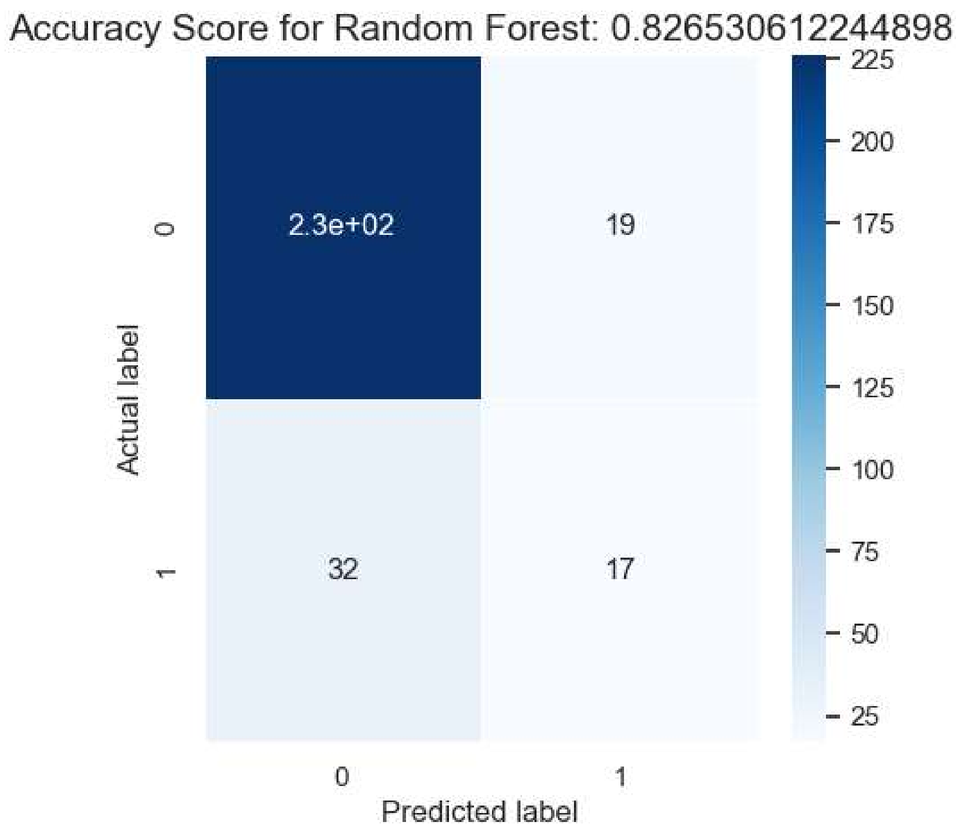


In [46]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



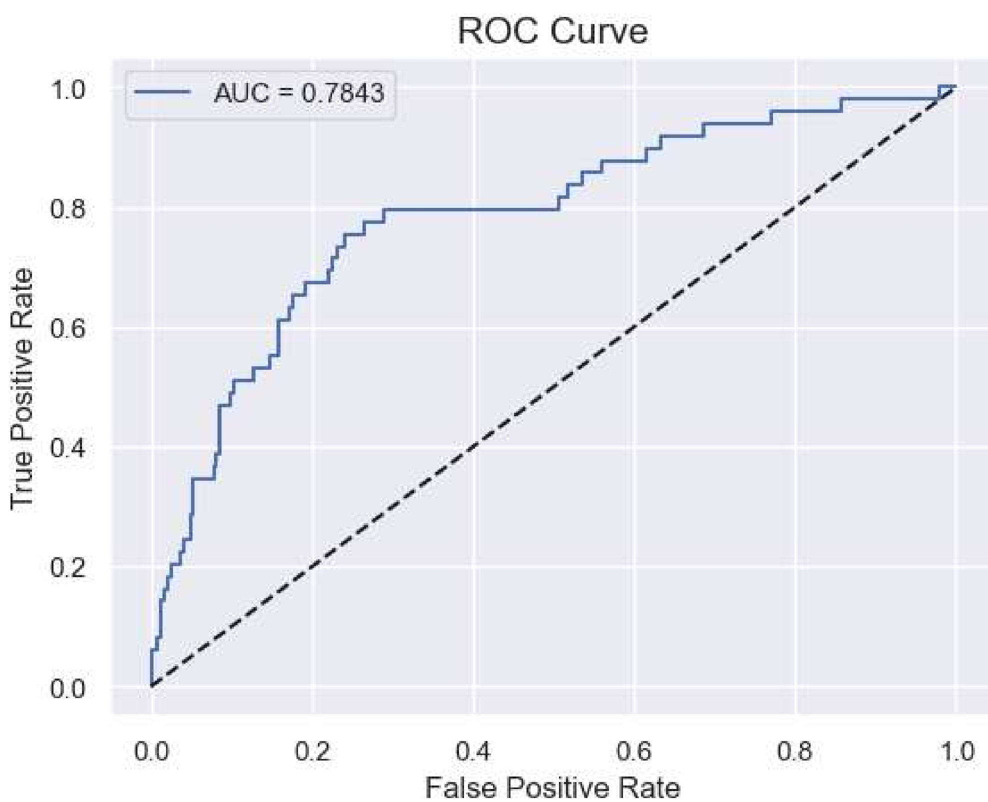
In [47]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[47]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.826530612244898')
```



In [48]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.Da
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x1f2b254b550>



XGBoost Classifier

In [49]:
```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Create an XGBoost classifier
xgb = XGBClassifier()

# Define the parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 0.1, 0.2]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(xgb, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200}
```

In [50]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0.1, learning_rate=0.1, max_depth=7, n_estimators=200)
xgb.fit(X_train, y_train)
```

Out[50]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.1, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=7, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [51]:
```
from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

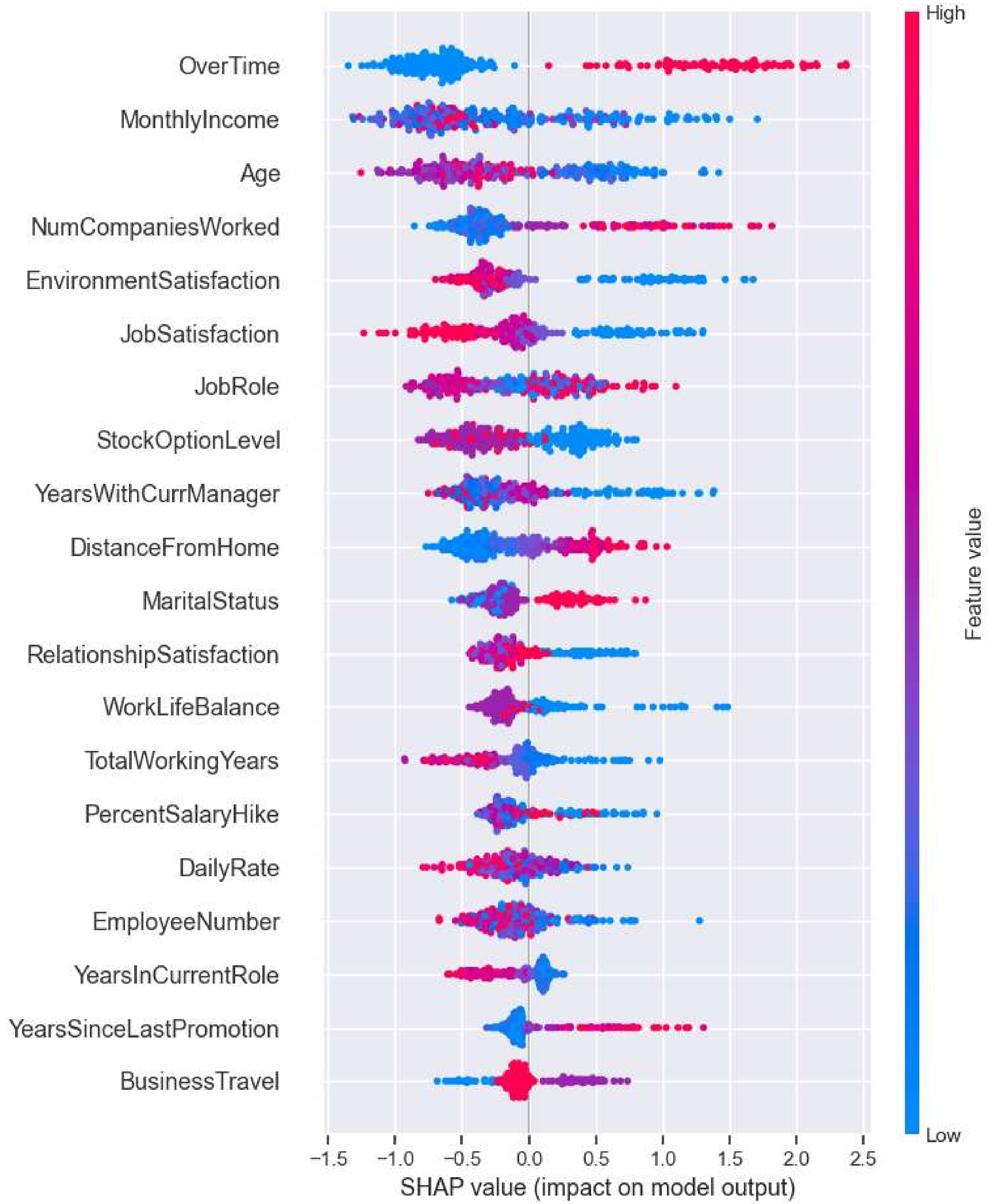
```
Accuracy Score : 86.05 %
```

In [52]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8605442176870749
Precision Score :  0.8605442176870748
Recall Score :  0.8605442176870748
Jaccard Score :  0.755223880597015
Log Loss :  4.816648400598365
```

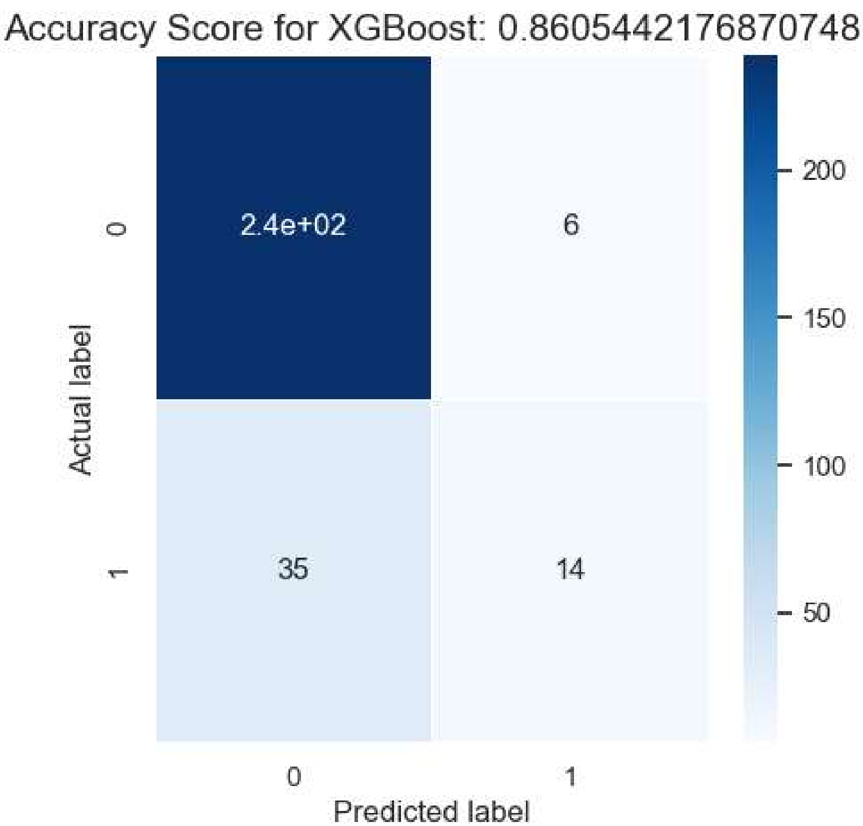
In [53]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```



In [54]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for XGBoost: {0}'.format(xgb.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[54]:
```
Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.8605442176870748')
```



In [55]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = xgb.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.Da
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[55]: <matplotlib.legend.Legend at 0x1f2b2503be0>