In [2]:
```
df = pd.read_csv('C:/Users/zizhe/Desktop/Leo Zhao/Employee Attrition for Healthcare prediction/watson_healthcare_modifi
df.head()
```

Out[2]:

	EmployeeID	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	Envi
0	1313919	41	No	Travel_Rarely	1102	Cardiology	1	2	Life Sciences	1	
1	1200302	49	No	Travel_Frequently	279	Maternity	8	1	Life Sciences	1	
2	1060315	37	Yes	Travel_Rarely	1373	Maternity	2	2	Other	1	
3	1272912	33	No	Travel_Frequently	1392	Maternity	3	4	Life Sciences	1	
4	1414939	27	No	Travel_Rarely	591	Maternity	2	1	Medical	1	

Data Preprocessing Part 1

In [3]:
```
# Drop identifier column
df.drop(columns = 'EmployeeID', inplace=True)
df.head()
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EnvironmentSatisf
0	41	No	Travel_Rarely	1102	Cardiology	1	2	Life Sciences	1	
1	49	No	Travel_Frequently	279	Maternity	8	1	Life Sciences	1	
2	37	Yes	Travel_Rarely	1373	Maternity	2	2	Other	1	
3	33	No	Travel_Frequently	1392	Maternity	3	4	Life Sciences	1	
4	27	No	Travel_Rarely	591	Maternity	2	1	Medical	1	

In [4]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[4]:
```
Attrition          2
BusinessTravel     3
Department         3
EducationField     6
Gender             2
JobRole            5
MaritalStatus      3
Over18             1
OverTime           2
dtype: int64
```

In [5]:
```
# Drop column with only 1 unique value
df.drop(columns = 'Over18', inplace=True)
df.head()
```

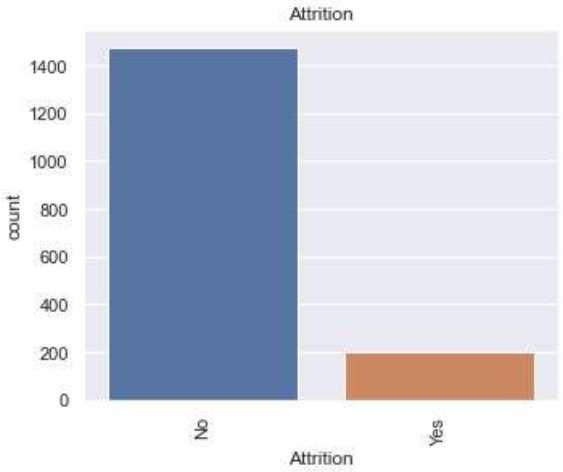
Out[5]:

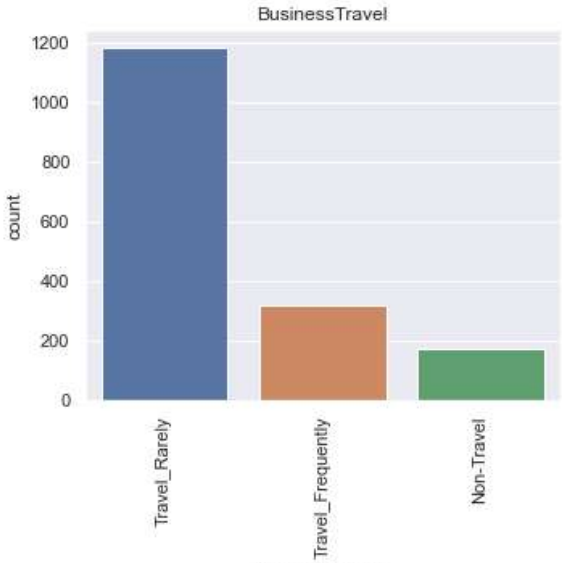
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EnvironmentSatisf
0	41	No	Travel_Rarely	1102	Cardiology	1	2	Life Sciences	1	
1	49	No	Travel_Frequently	279	Maternity	8	1	Life Sciences	1	
2	37	Yes	Travel_Rarely	1373	Maternity	2	2	Other	1	
3	33	No	Travel_Frequently	1392	Maternity	3	4	Life Sciences	1	
4	27	No	Travel_Rarely	591	Maternity	2	1	Medical	1	

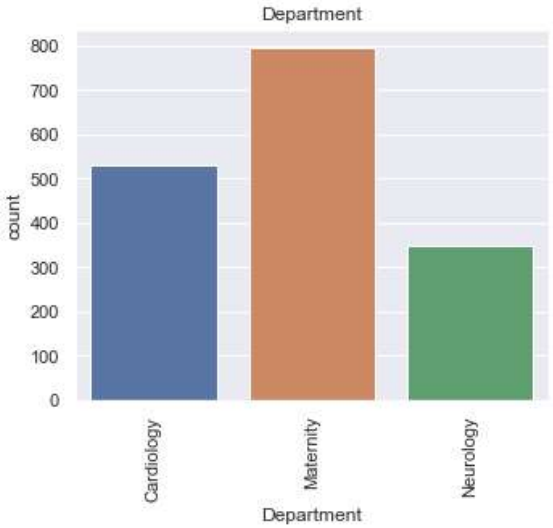
Exploratory Data Analysis

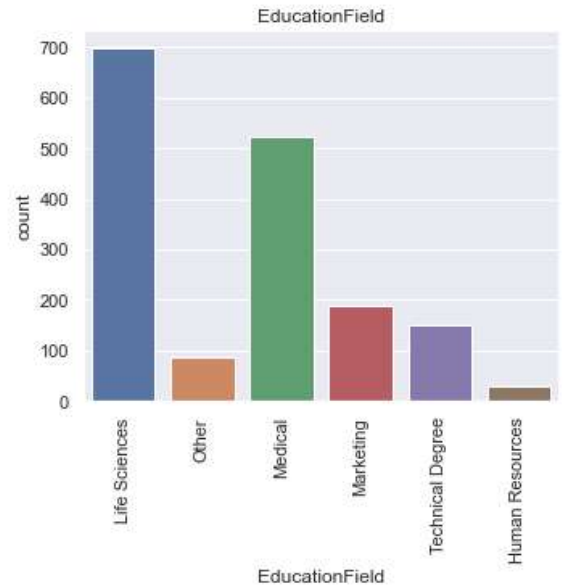
In [6]:
```
# Get the names of all columns with data type 'object' (categorical columns)
cat_vars = df.select_dtypes(include='object').columns.tolist()

# 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)
axs = axs.flatten()

# Create a countplot for the top 6 values of each categorical variable using Seaborn
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(6).index
    filtered_df = df[df[var].isin(top_values)]
```

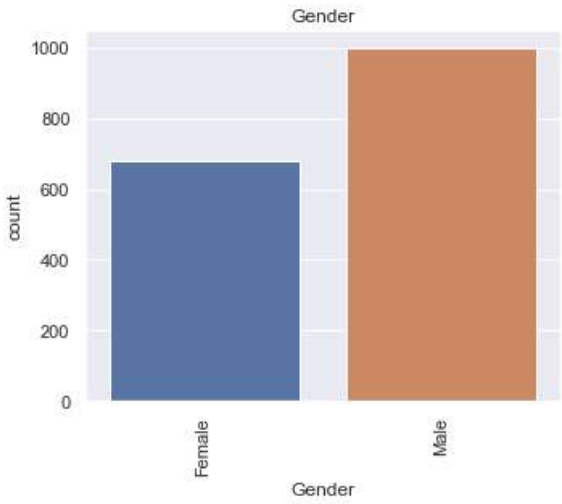
```
    sns.countplot(x=var, data=filtered_df, ax=axs[i])
    axs[i].set_title(var)
    axs[i].tick_params(axis='x', rotation=90)

# 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()
```

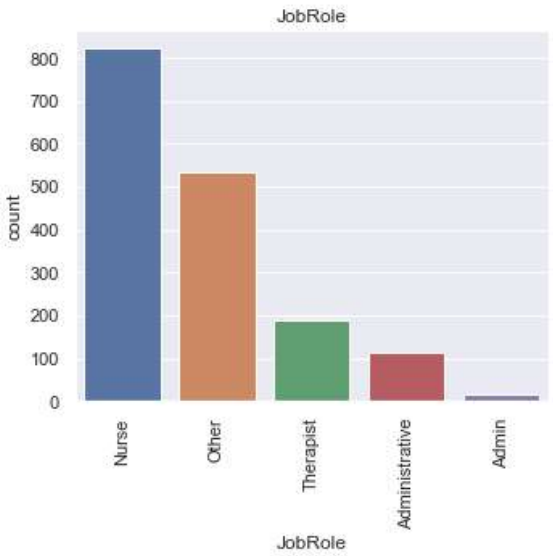

Attrition
count
Attrition
No
Yes

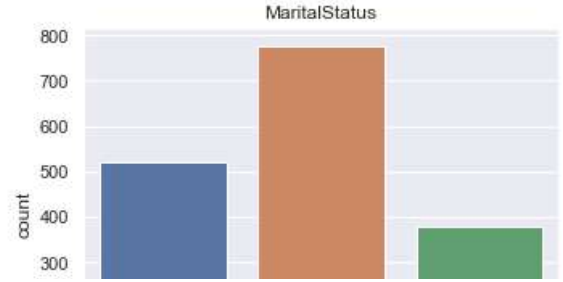

BusinessTravel
count
BusinessTravel
Travel_Rarely
Travel_Frequently
Non-Travel

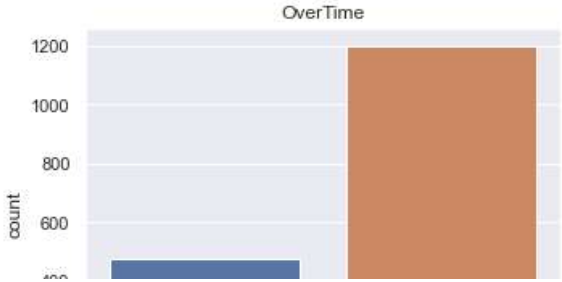

Department
count
Department
Cardiology
Maternity
Neurology

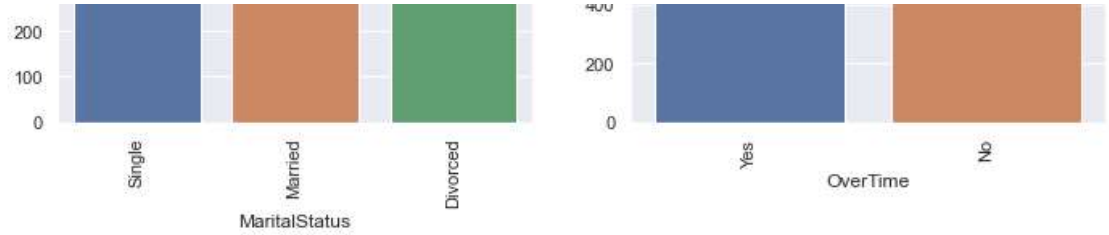

EducationField
count
EducationField
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources


Gender
count
Gender
Female
Male


JobRole
count
JobRole
Nurse
Other
Therapist
Administrative
Admin

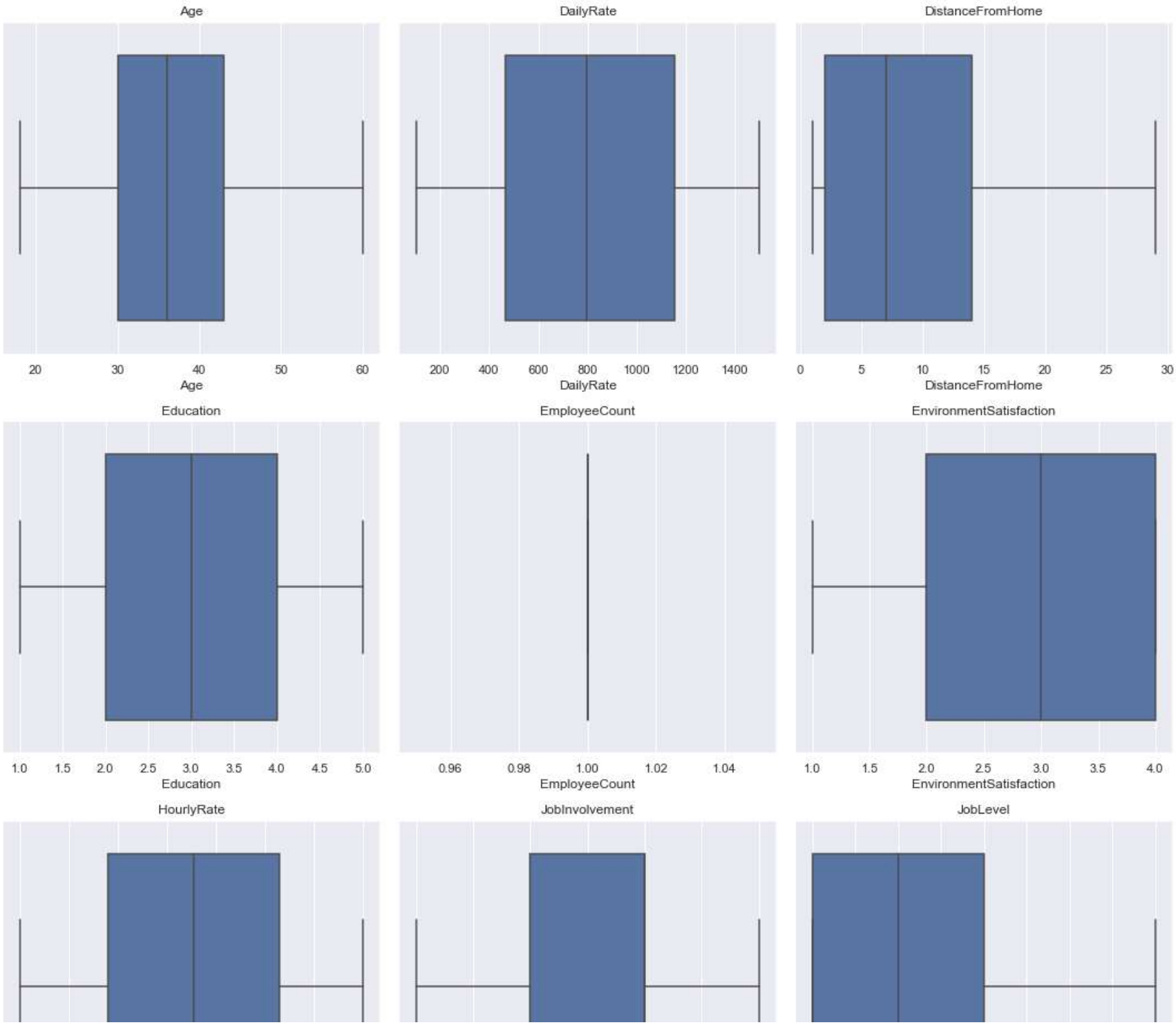

MaritalStatus
count

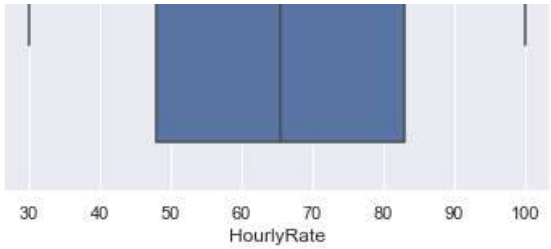

OverTime
count

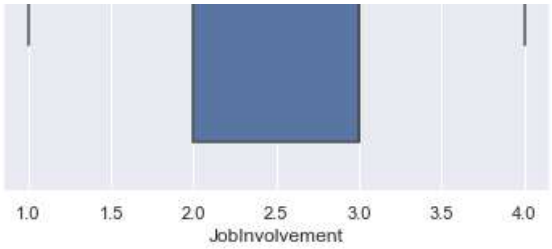


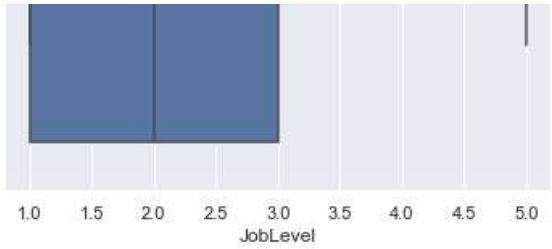
In [7]:

```
# 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))
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
EmployeeCount
0.96
0.98
1.00
1.02
1.04
EmployeeCount
EnvironmentSatisfaction
1.0
1.5
2.0
2.5
3.0
3.5
4.0
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel



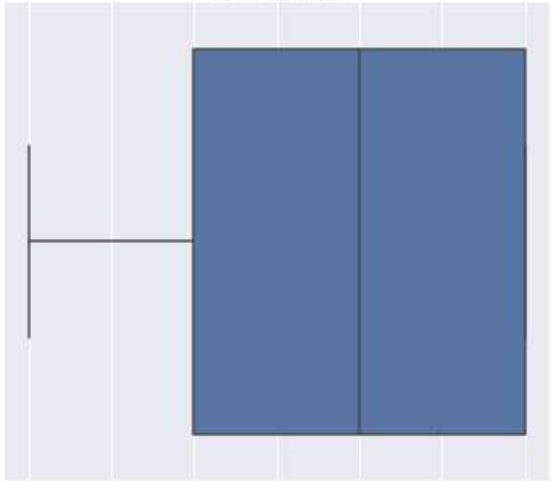


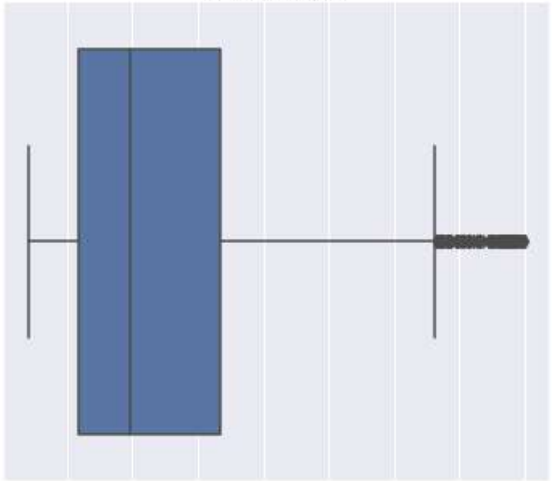


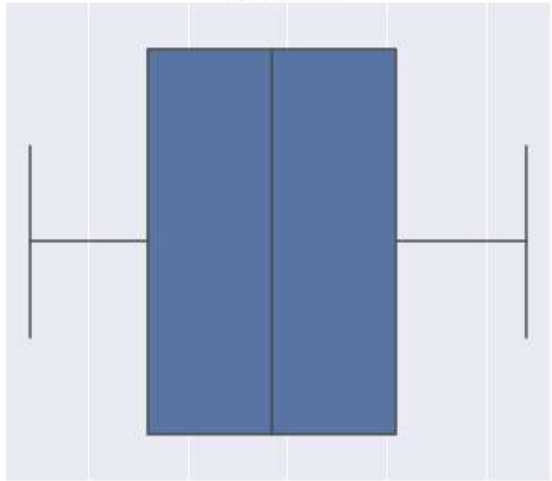
JobSatisfaction

MonthlyIncome

MonthlyRate



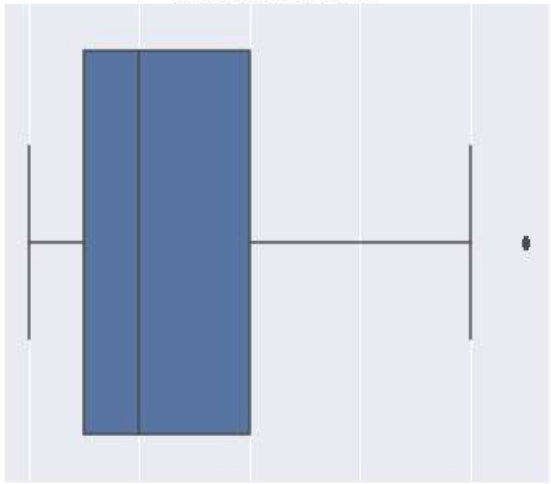


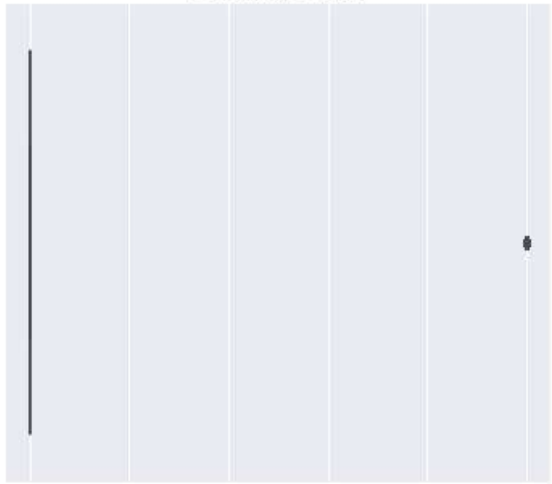


NumCompaniesWorked

PercentSalaryHike

PerformanceRating



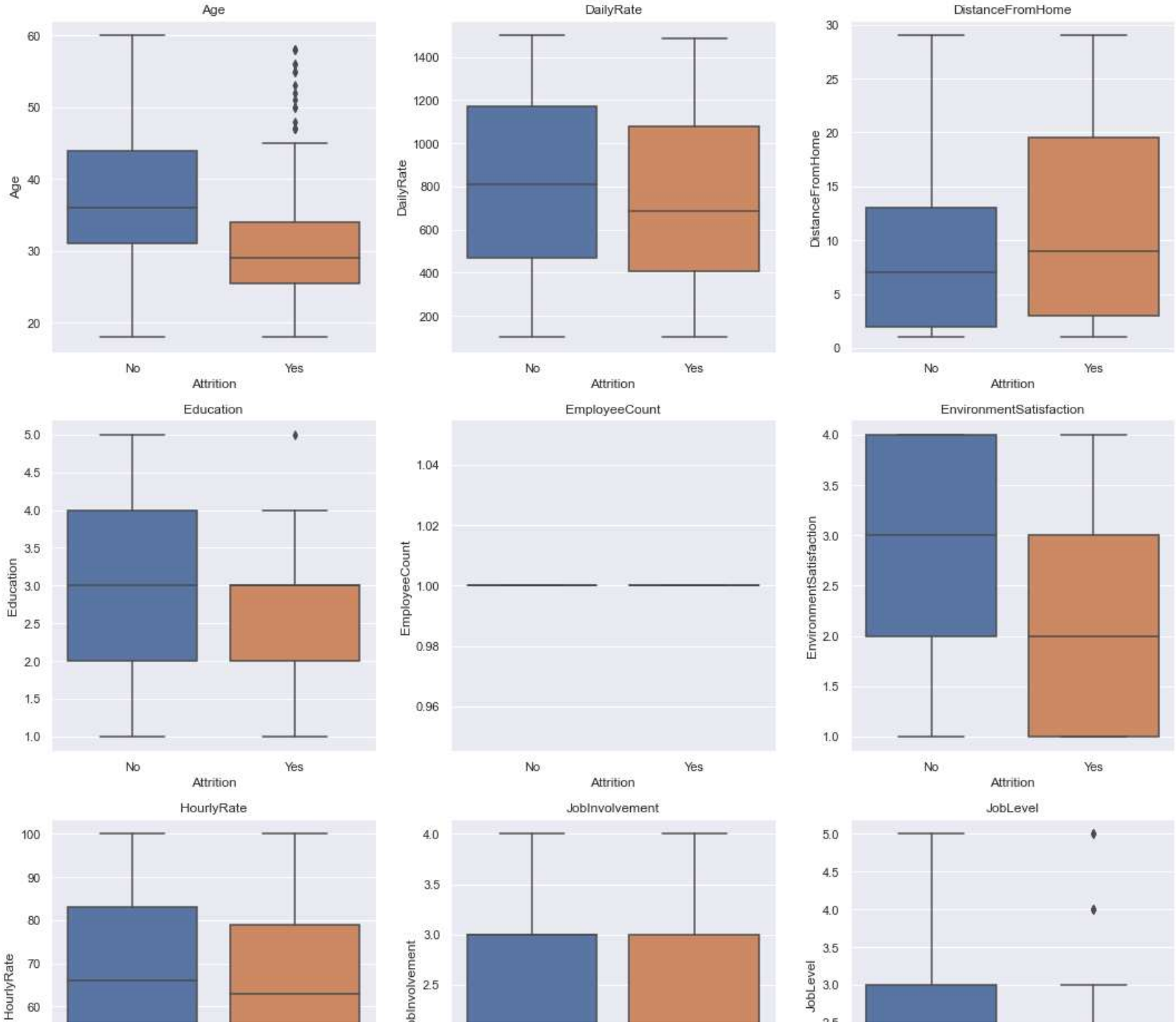


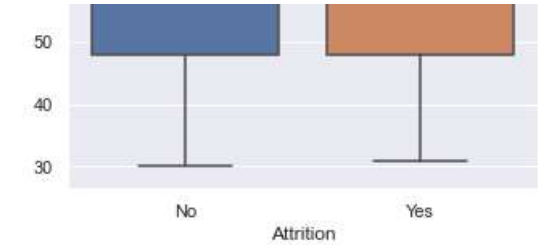
RelationshipSatisfaction

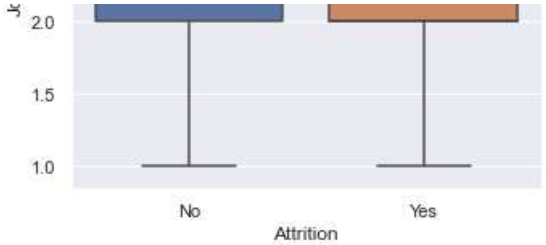
StandardHours

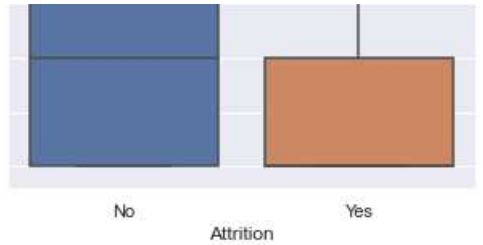
Shift

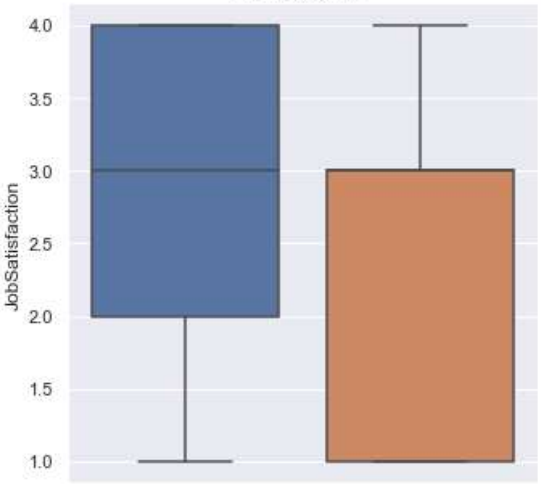
In [8]:
```
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) / 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5)
axs = axs.flatten()

# 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()
```

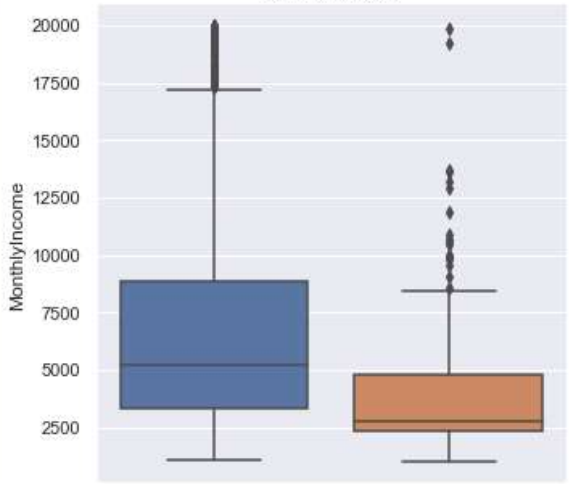

Age
Age
60
50
40
30
20
No
Yes
Attrition
DailyRate
DailyRate
1400
1200
1000
800
600
400
200
No
Yes
Attrition
DistanceFromHome
DistanceFromHome
30
25
20
15
10
5
0
No
Yes
Attrition
Education
Education
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
No
Yes
Attrition
EmployeeCount
EmployeeCount
1.04
1.02
1.00
0.98
0.96
No
Yes
Attrition
EnvironmentSatisfaction
EnvironmentSatisfaction
4.0
3.5
3.0
2.5
2.0
1.5
1.0
No
Yes
Attrition
HourlyRate
HourlyRate
100
90
80
70
60
JobInvolvement
4.0
3.5
3.0
2.5
JobLevel
JobLevel
5.0
4.5
4.0
3.5
3.0

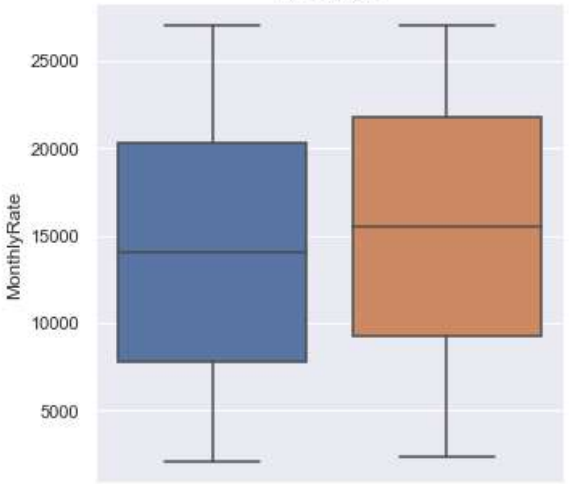

50
40
30
No
Yes
Attrition

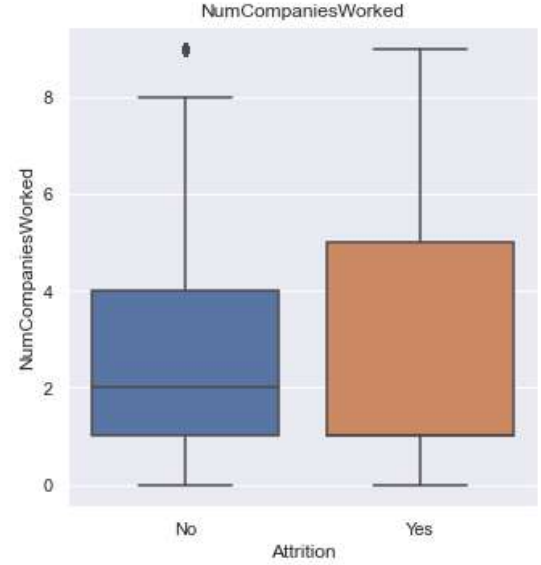

2.0
1.5
1.0
No
Yes
Attrition

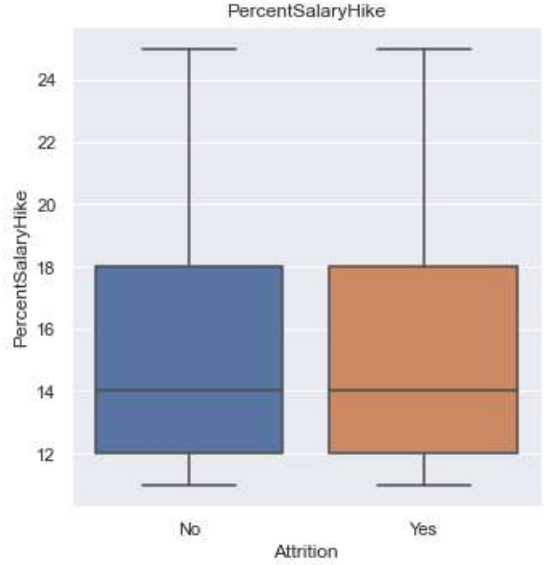

2.0
1.5
1.0
No
Yes
Attrition

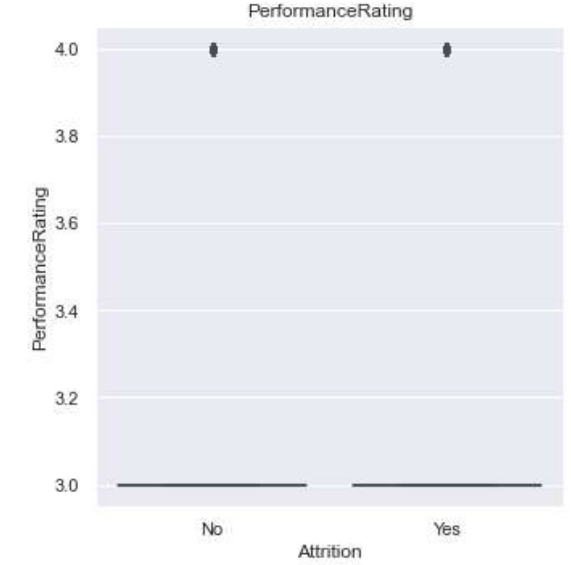

JobSatisfaction
4.0
3.5
3.0
2.5
2.0
1.5
1.0
No
Yes
Attrition


MonthlyIncome
20000
17500
15000
12500
10000
7500
5000
2500
No
Yes
Attrition


MonthlyRate
25000
20000
15000
10000
5000
No
Yes
Attrition

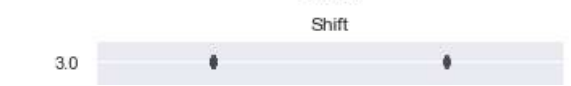

NumCompaniesWorked
8
6
4
2
0
No
Yes
Attrition


PercentSalaryHike
24
22
20
18
16
14
12
No
Yes
Attrition

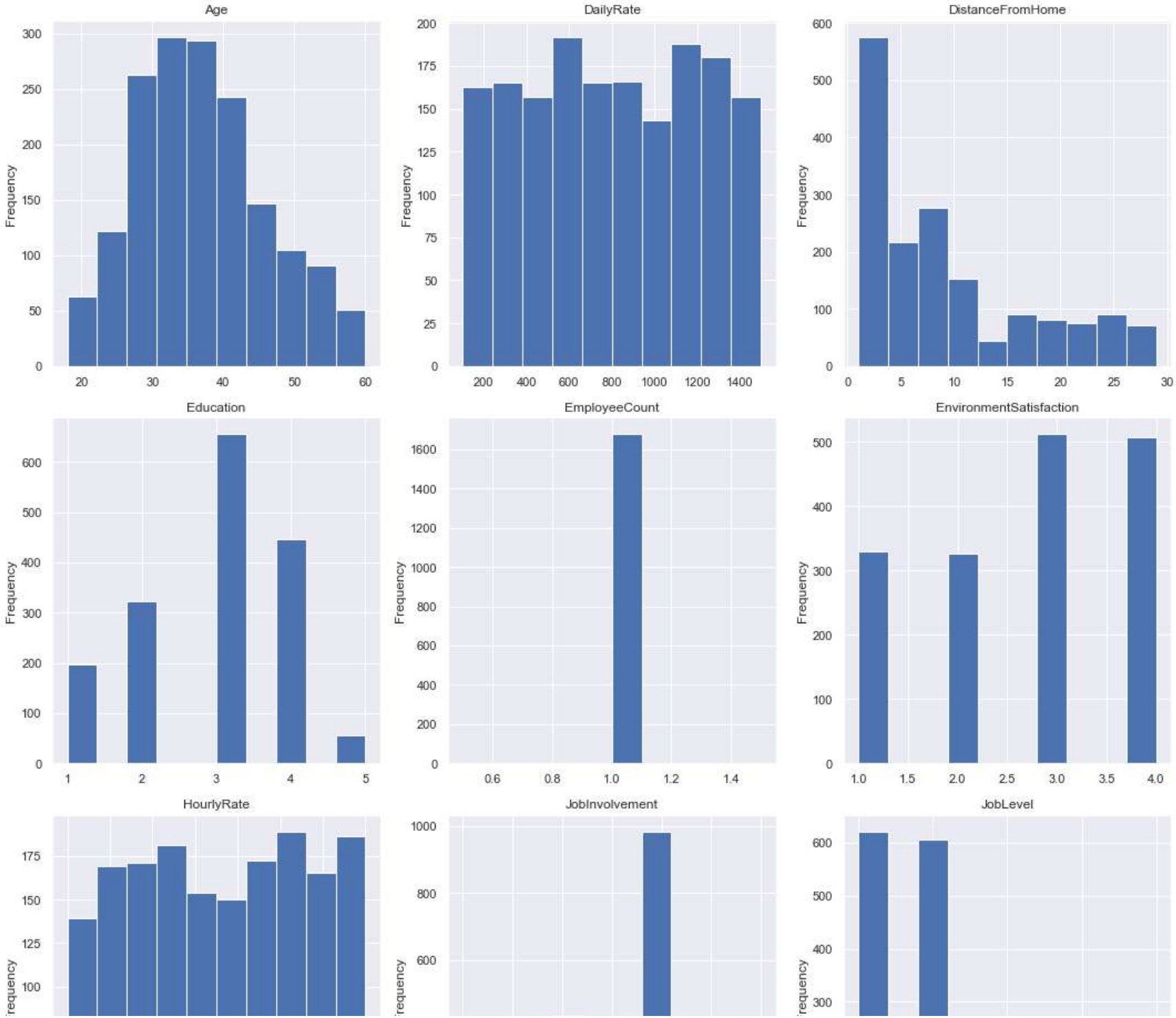

PerformanceRating
4.0
3.8
3.6
3.4
3.2
3.0
No
Yes
Attrition

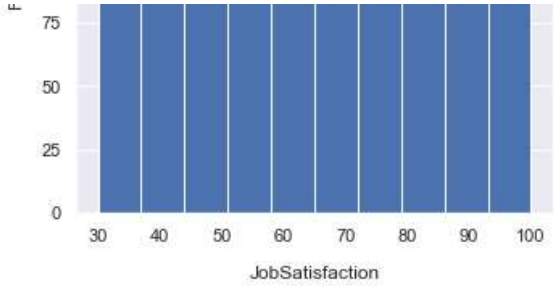

RelationshipSatisfaction
4.0


StandardHours
84


Shift
3.0

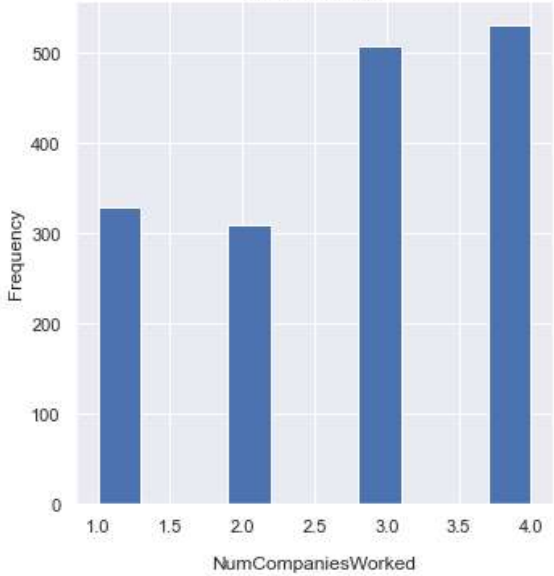
In [9]:
```
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) / 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5)
axs = axs.flatten()

# 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()
```

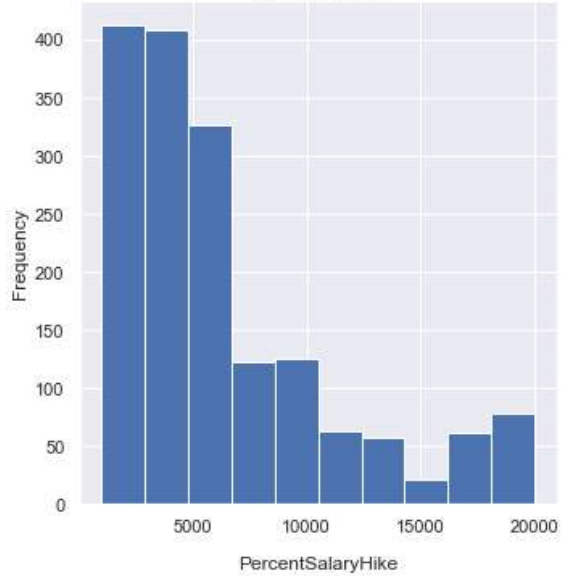

Age
Frequency
DailyRate
Frequency
DistanceFromHome
Frequency
Education
Frequency
EmployeeCount
Frequency
EnvironmentSatisfaction
Frequency
HourlyRate
Frequency
JobInvolvement
Frequency
JobLevel
Frequency

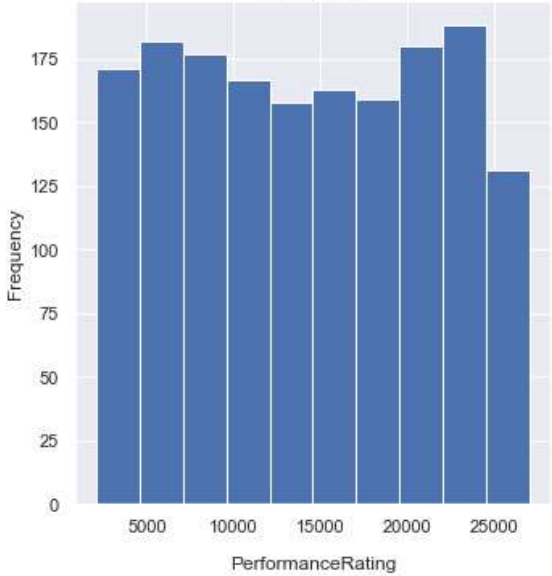

0
25
50
75
30
40
50
60
70
80
90
100
JobSatisfaction

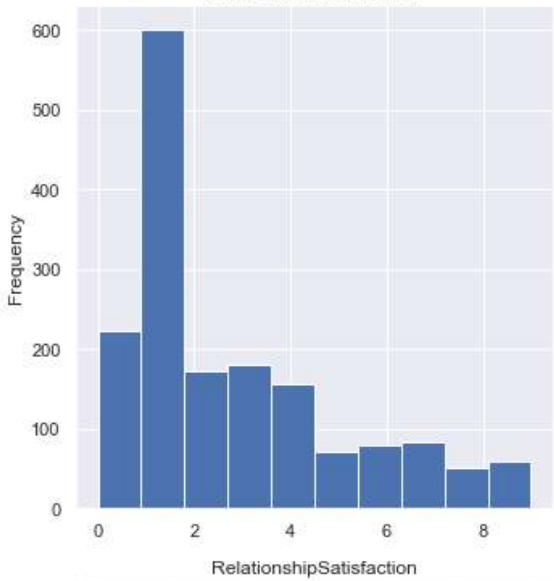

0
200
400
1.0
1.5
2.0
2.5
3.0
3.5
4.0
MonthlyIncome

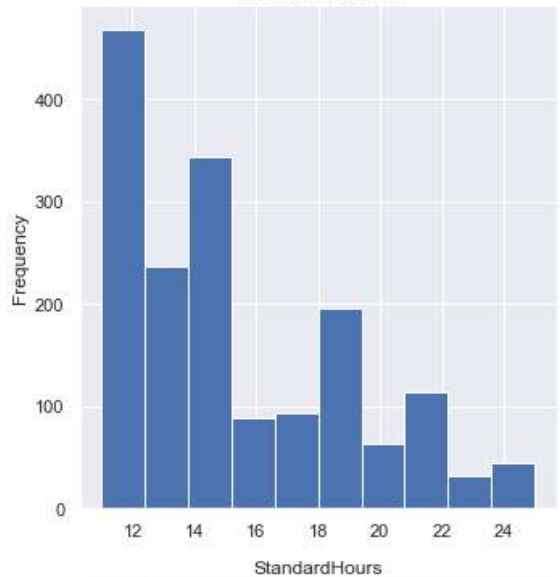

0
100
200
1
2
3
4
5
MonthlyRate

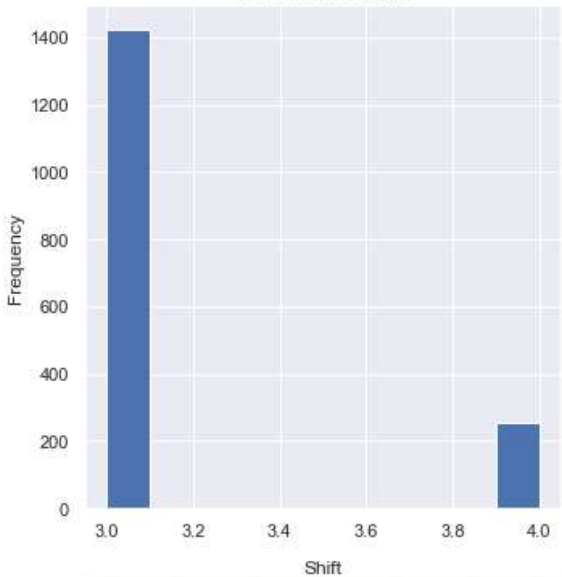

Frequency
0
100
200
300
400
500
1.0
1.5
2.0
2.5
3.0
3.5
4.0
NumCompaniesWorked


Frequency
0
50
100
150
200
250
300
350
400
5000
10000
15000
20000
PercentSalaryHike


Frequency
0
25
50
75
100
125
150
175
5000
10000
15000
20000
25000
PerformanceRating

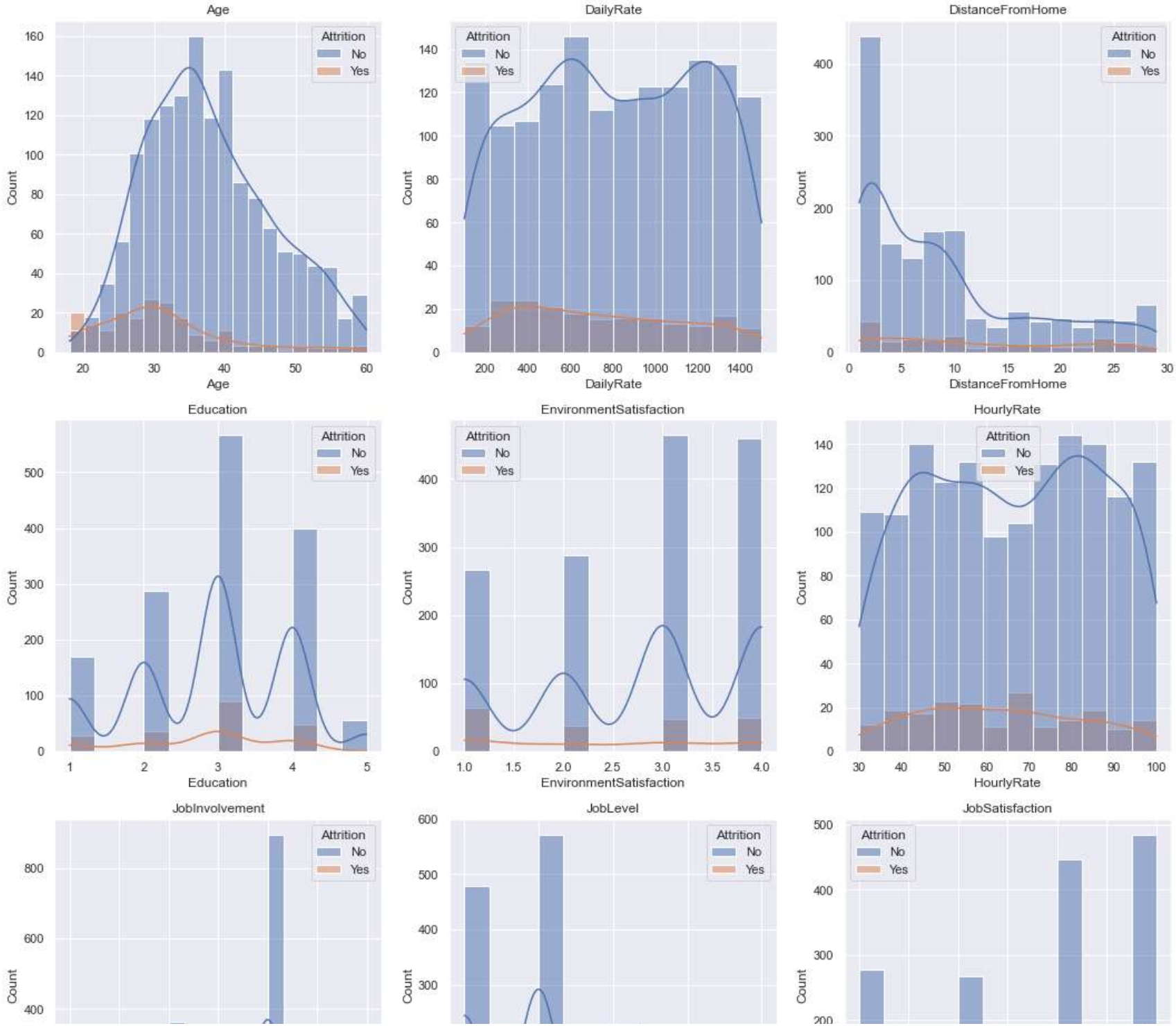

Frequency
0
100
200
300
400
500
600
0
2
4
6
8
RelationshipSatisfaction

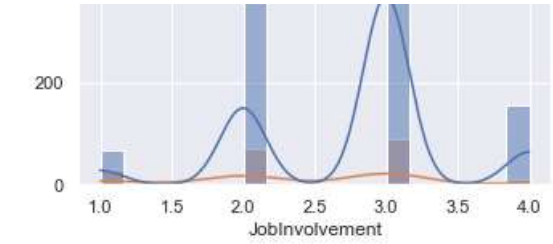

Frequency
0
100
200
300
400
12
14
16
18
20
22
24
StandardHours

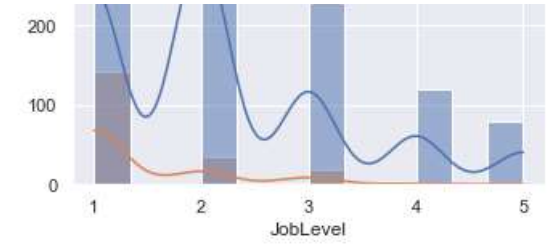

Frequency
0
200
400
600
800
1000
1200
1400
3.0
3.2
3.4
3.6
3.8
4.0
Shift


700

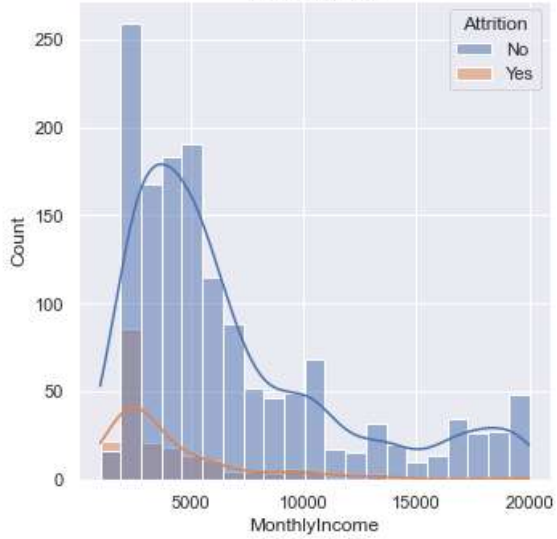
In [10]:
```
# Calculate the number of rows and columns for subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) / 3  # 3 subplots per row, rounding up

# Create a figure with subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5)
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    if i < num_cols:  # To avoid accessing non-existent subplots
        sns.histplot(data=df, x=var, hue='Attrition', kde=True, ax=axs[i])
        axs[i].set_title(var)

# Adjust spacing between subplots
fig.tight_layout()

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

# Show plot
plt.show()
```


Age
Attrition
No
Yes
Count
Age
DailyRate
DailyRate
DistanceFromHome
DistanceFromHome
Education
Education
EnvironmentSatisfaction
EnvironmentSatisfaction
HourlyRate
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction

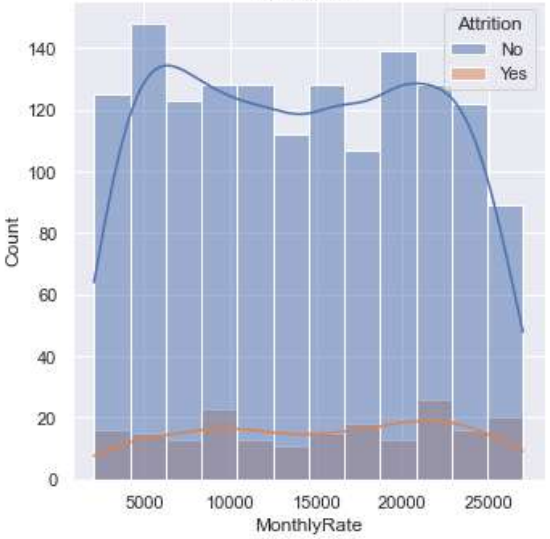




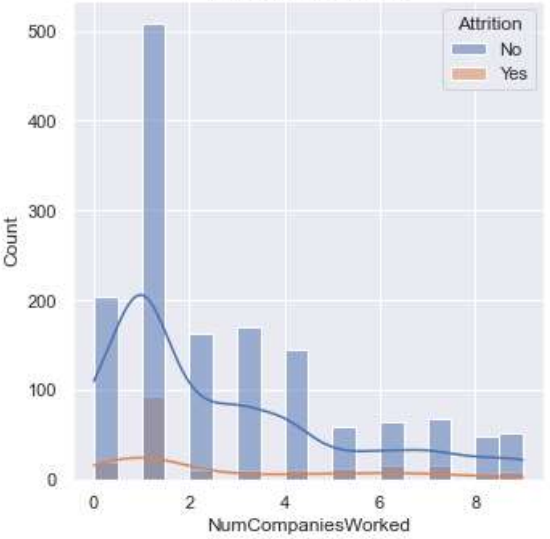
MonthlyIncome



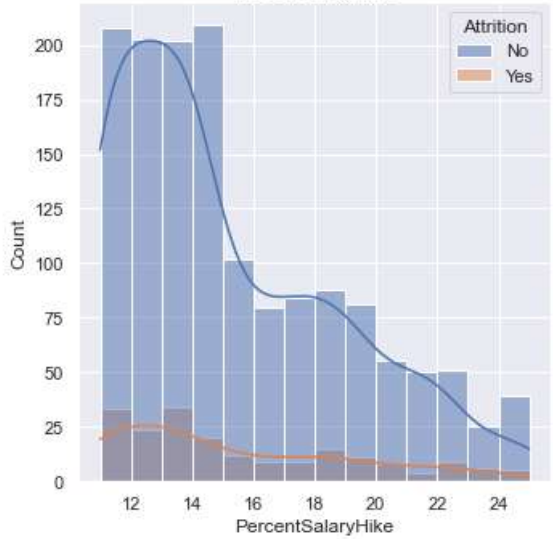
MonthlyRate



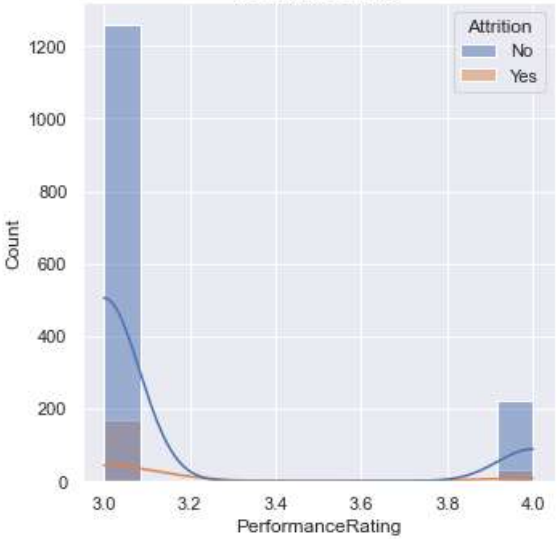
NumCompaniesWorked



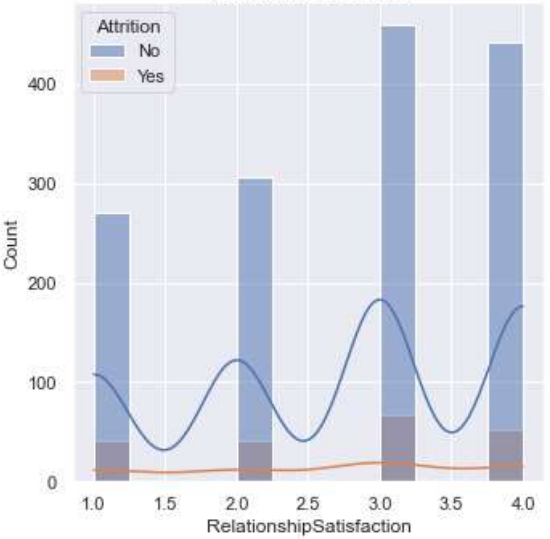
PercentSalaryHike



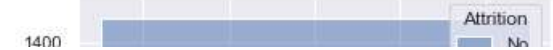
PerformanceRating



RelationshipSatisfaction



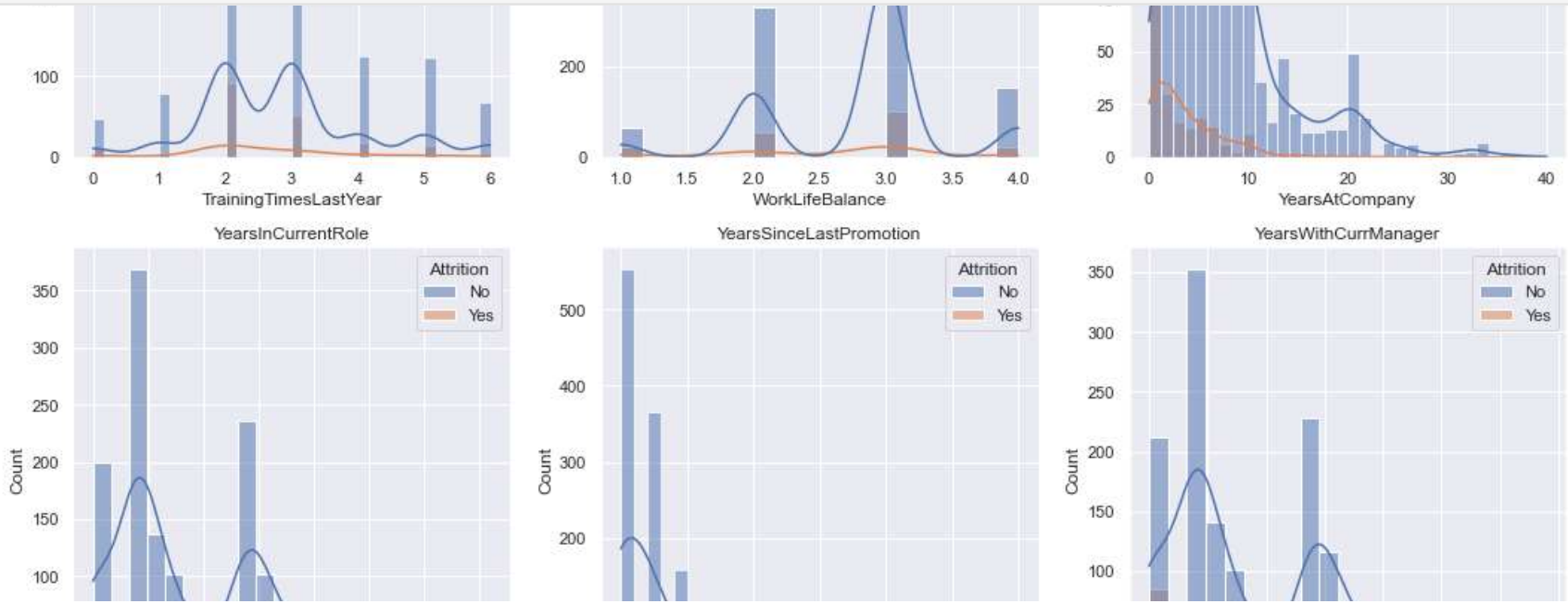
StandardHours

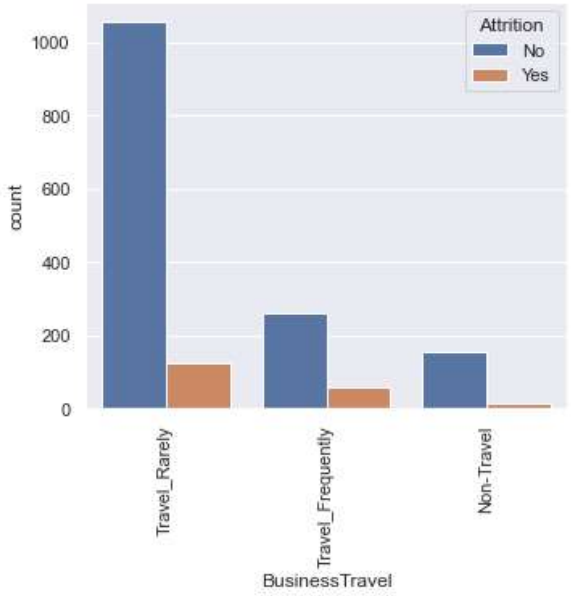


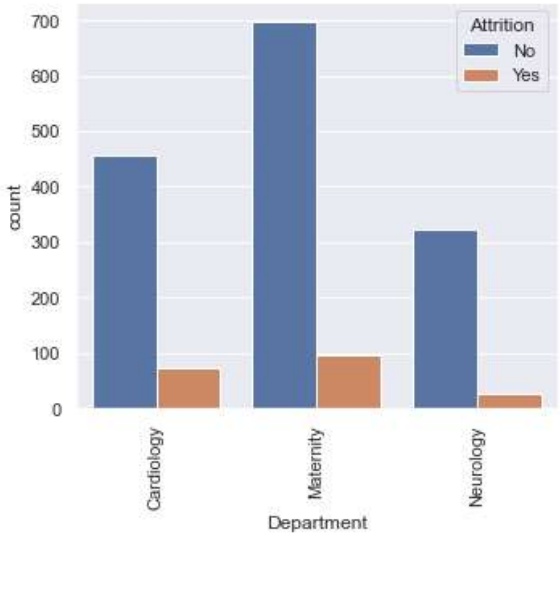
Shift

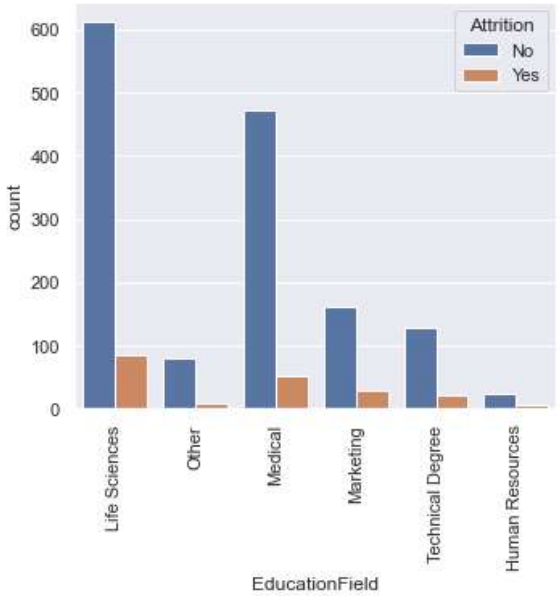
TotalWorkingYears

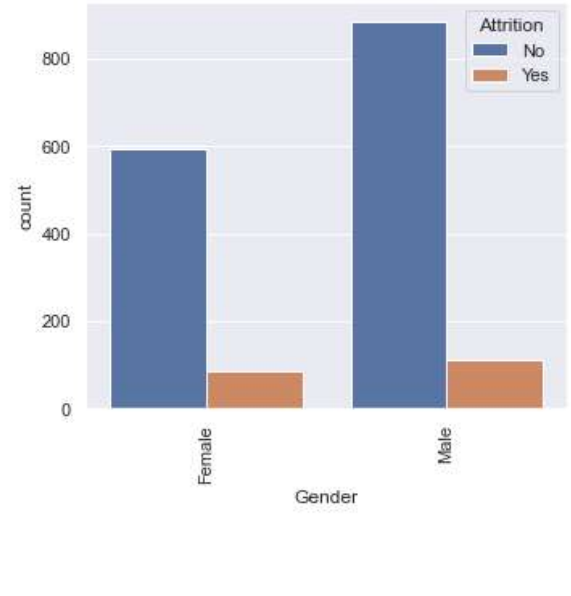
In [11]:
```
# Exclude 'Attrition' from the list if it exists in cat_vars
if 'Attrition' in cat_vars:
    cat_vars.remove('Attrition')

# 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)
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()
```

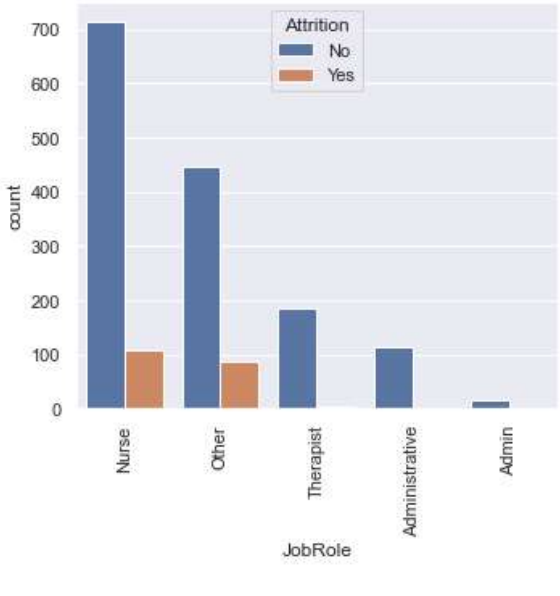


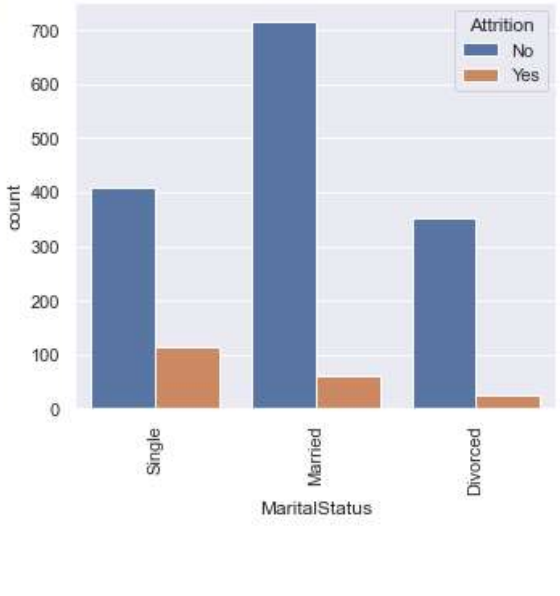

Attrition
No
Yes
count
1000
800
600
400
200
0
Travel_Rarely
Travel_Frequently
Non-Travel
BusinessTravel


Attrition
No
Yes
count
700
600
500
400
300
200
100
0
Cardiology
Maternity
Neurology
Department

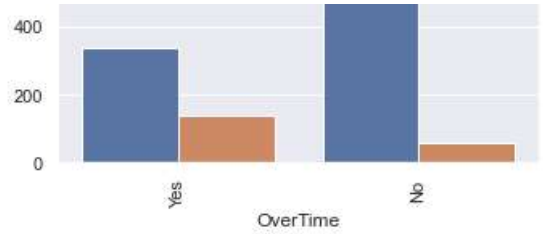

Attrition
No
Yes
count
600
500
400
300
200
100
0
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources
EducationField


Attrition
No
Yes
count
800
600
400
200
0
Female
Male
Gender


Attrition
No
Yes
count
700
600
500
400
300
200
100
0
Nurse
Other
Therapist
Administrative
Admin
JobRole


Attrition
No
Yes
count
700
600
500
400
300
200
100
0
Single
Married
Divorced
MaritalStatus

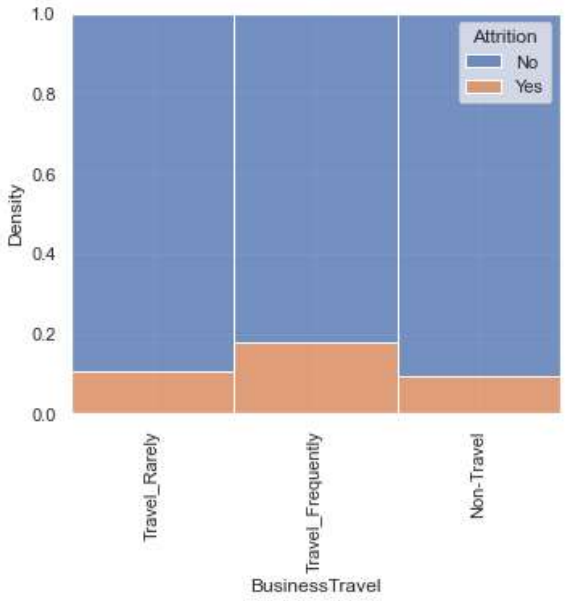

Attrition
No
Yes
count
1000
800
600

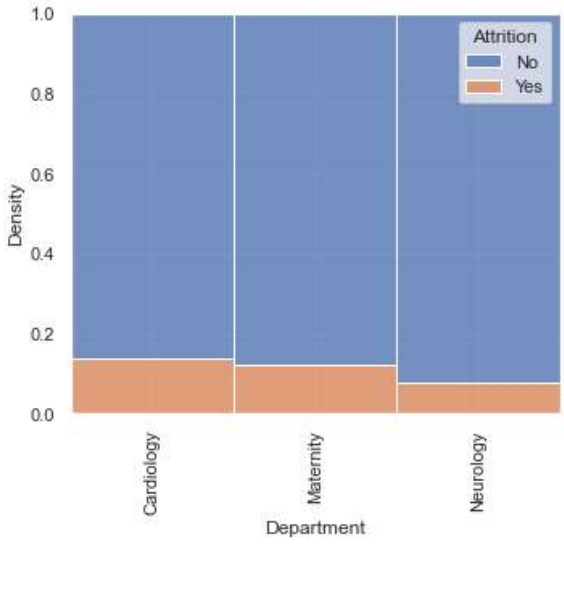


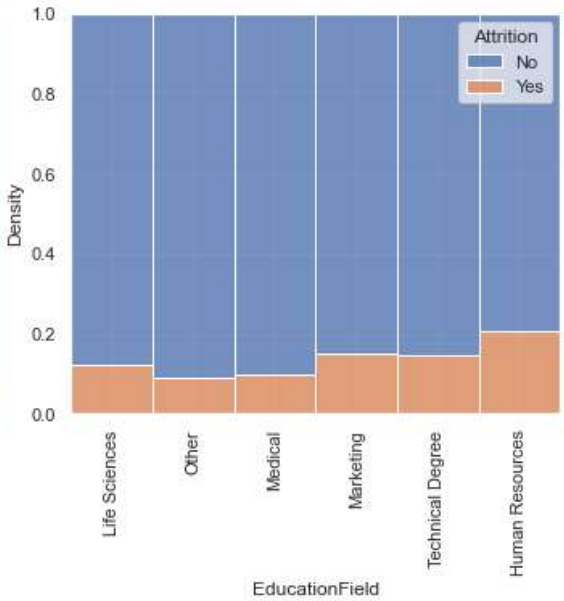
In [12]:

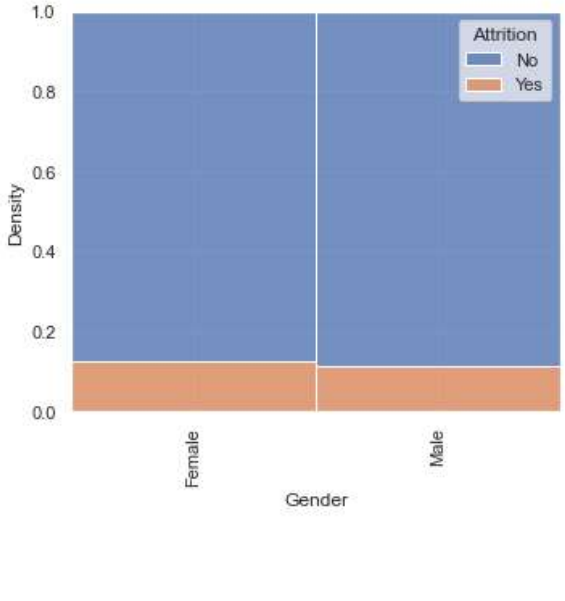
```
# Exclude 'Attrition' from the list if it exists in cat_vars
if 'Attrition' in cat_vars:
    cat_vars.remove('Attrition')

# 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)
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, element="bars", fill=True, sta
    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()
```

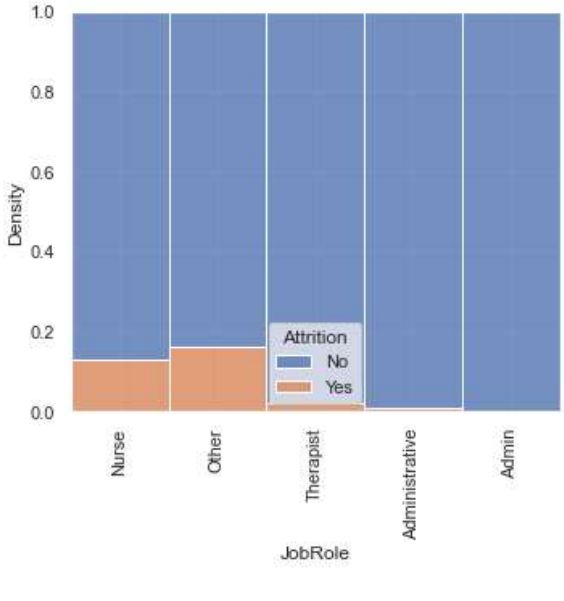
```
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\1288156870.py:17: UserWarning: FixedFormatter should only be used tog
ether with FixedLocator
  axs[i].set_xticklabels(df[var].unique(), rotation=90)
```

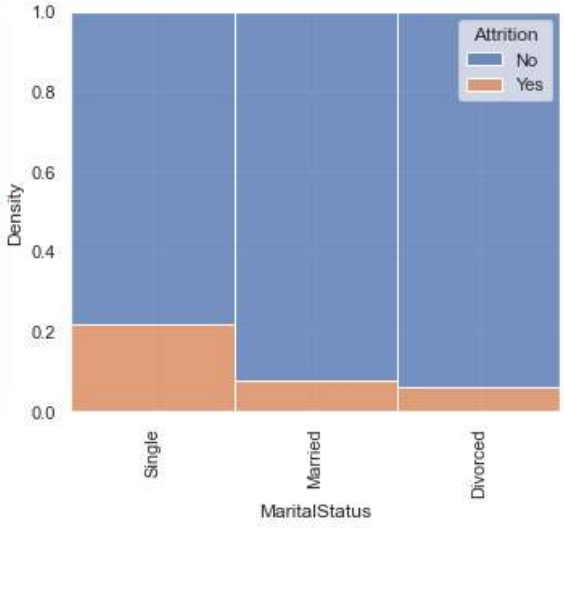

Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Travel_Rarely
Travel_Frequently
Non-Travel
BusinessTravel

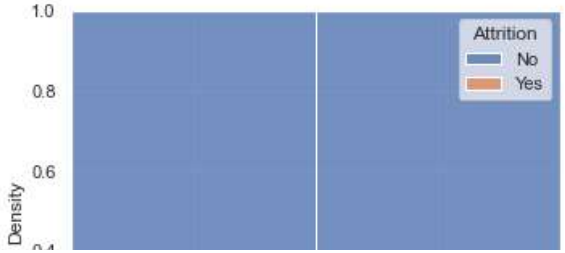

Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Cardiology
Maternity
Neurology
Department

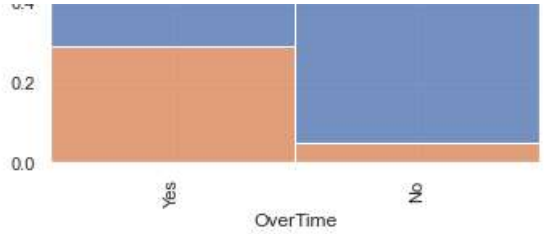

Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources
EducationField


Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Female
Male
Gender


Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Nurse
Other
Therapist
Administrative
Admin
JobRole


Attrition
No
Yes
Density
1.0
0.8
0.6
0.4
0.2
0.0
Single
Married
Divorced
MaritalStatus


Attrition
No
Yes
Density
1.0
0.8
0.6

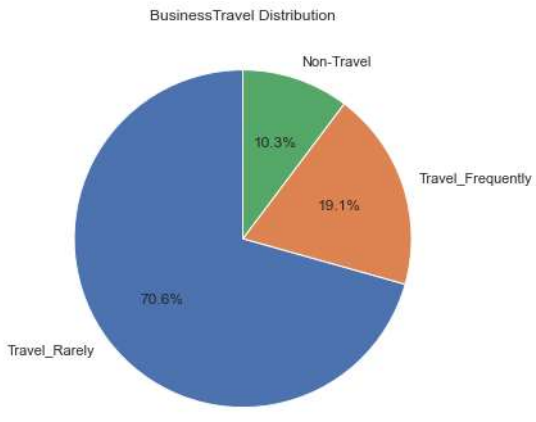


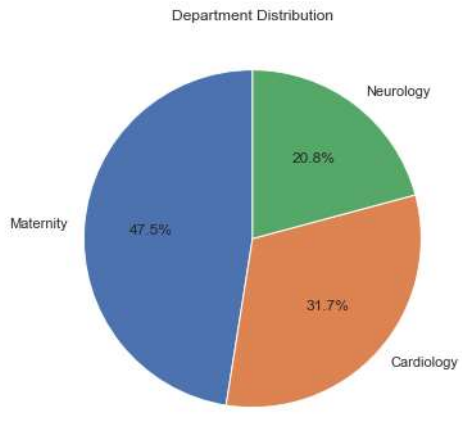
In [13]:
```python
# Specify the maximum number of categories to show individually
max_categories = 5

# 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)

# 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,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
```

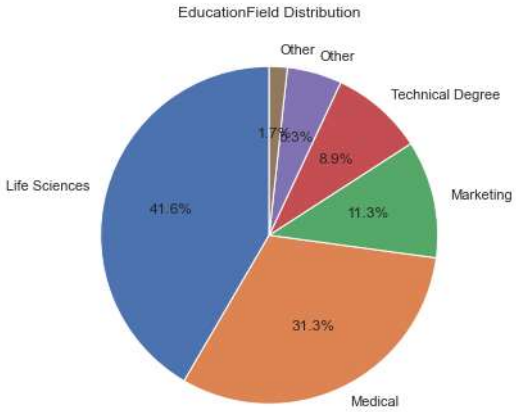
```
fig.tight_layout()

# Show plot
plt.show()
```

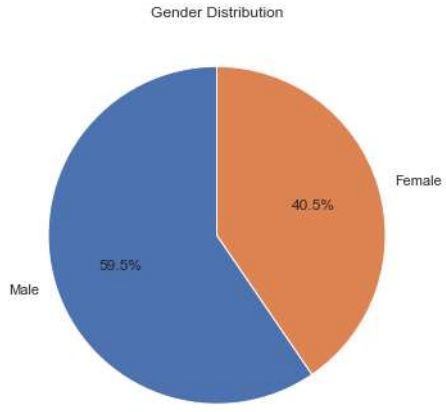
```
C:\Users\zizhe\AppData\Local\Temp\ipykernel_11744\943223290.py:25: FutureWarning: The series.append method is deprecate
d and will be removed from pandas in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```

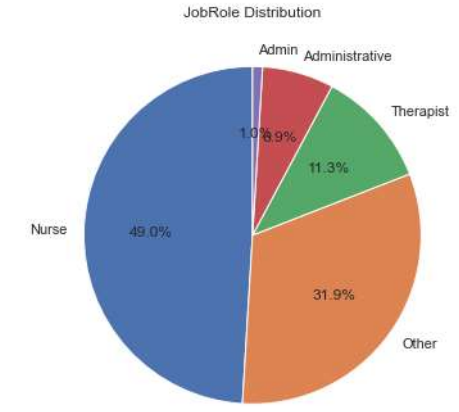

Attrition Distribution
Yes
11.9%
88.1%
No

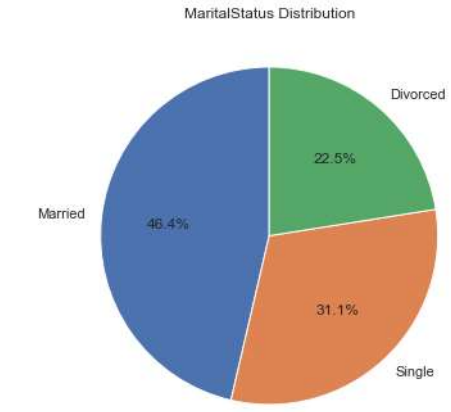

BusinessTravel Distribution
Non-Travel
10.3%
Travel_Frequently
19.1%
70.6%
Travel_Rarely

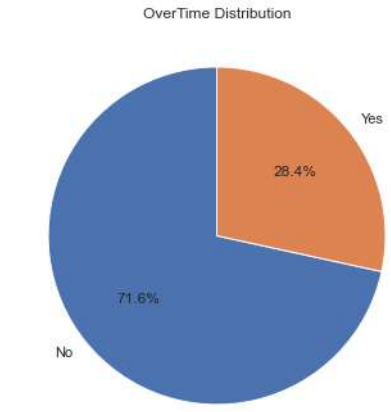

Department Distribution
Neurology
20.8%
Maternity
47.5%
31.7%
Cardiology


EducationField Distribution
Other
Other
Technical Degree
1.7%
3.3%
8.9%
Life Sciences
41.6%
11.3%
Marketing
31.3%
Medical


Gender Distribution
40.5%
Female
59.5%
Male


JobRole Distribution
Admin
Administrative
Therapist
1.0%
8.9%
11.3%
Nurse
49.0%
31.9%
Other


MaritalStatus Distribution
Divorced
22.5%
Married
46.4%
31.1%
Single


OverTime Distribution
Yes
28.4%
71.6%
No

Data Preprocessing Part 2

In [14]:
```
# Check the amounnt of missing value
check_missing = df.isnull().sum()/ df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[14]:
```
Series([], dtype: float64)
```

Label Encoding for Object Datatypes

In [15]:
```
# 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: ['No' 'Yes']
BusinessTravel: ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
Department: ['Cardiology' 'Maternity' 'Neurology']
EducationField: ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']
Gender: ['Female' 'Male']
JobRole: ['Nurse' 'Other' 'Therapist' 'Administrative' 'Admin']
MaritalStatus: ['Single' 'Married' 'Divorced']
OverTime: ['Yes' 'No']
```

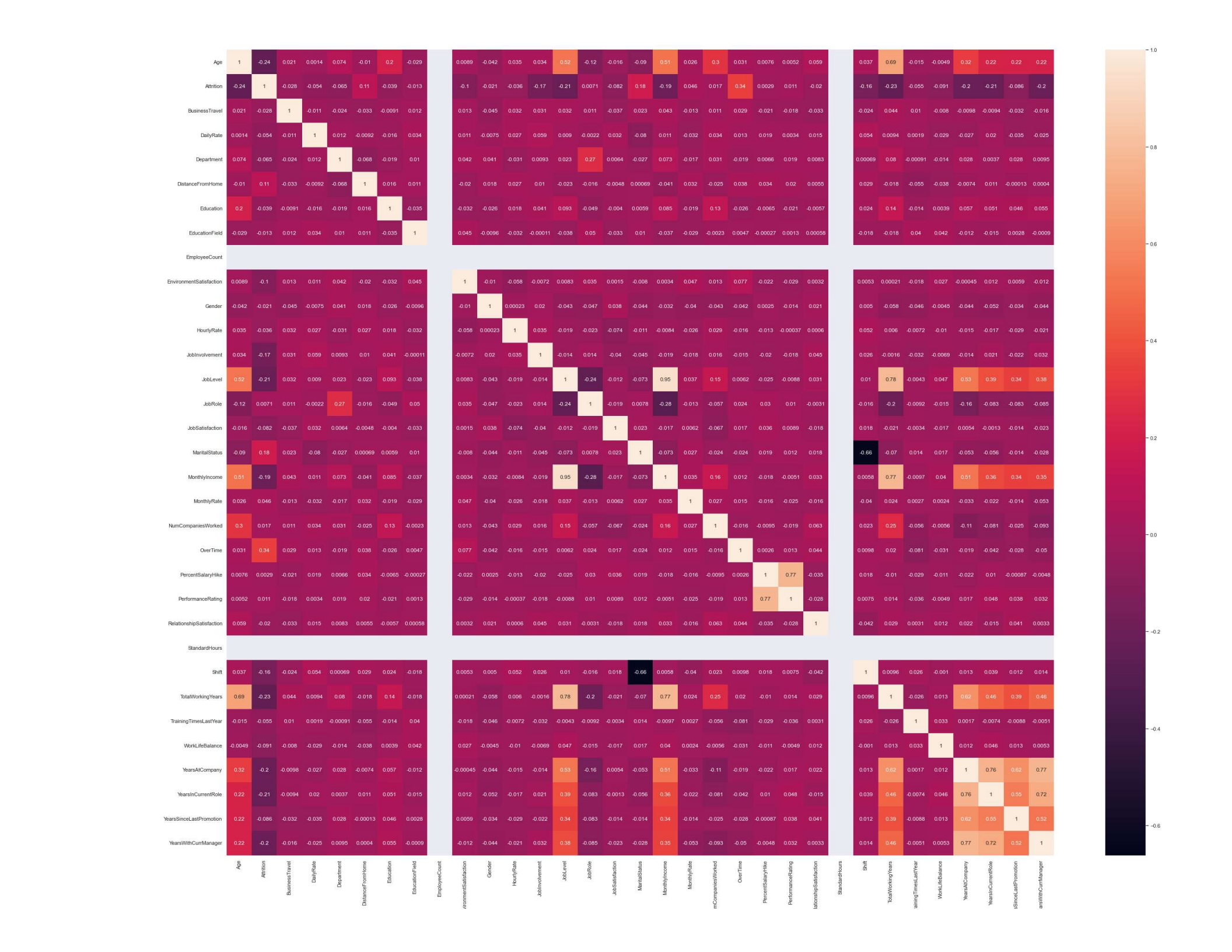
In [16]:
```
from sklearn import preprocessing

# 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: [0 1]
BusinessTravel: [2 1 0]
Department: [0 1 2]
EducationField: [1 4 3 2 5 0]
Gender: [0 1]
JobRole: [2 3 4 1 0]
MaritalStatus: [2 1 0]
OverTime: [1 0]
```

In [17]:
```
# Correlation Heatmap
plt.figure(figsize=(40, 32))
sns.heatmap(df.corr(), fmt='2g', annot=True)
```

Out[17]:
```
<AxesSubplot:>
```


Age
Attrition
BusinessTravel
DailyRate
Department
DistanceFromHome
Education
EducationField
EmployeeCount
EnvironmentSatisfaction
Gender
HourlyRate
JobInvolvement
JobLevel
JobRole
JobSatisfaction
MaritalStatus
MonthlyIncome
MonthlyRate
NumCompaniesWorked
OverTime
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StandardHours
Shift
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6

In [18]:
```
# Remove EmployeeCount and StandardHours column because they have 0 correlation
df.drop(columns = ['EmployeeCount', 'StandardHours'], inplace=True)
```

Train Test Split

In [19]:
```
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 [20]:
```
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]))

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[2]

# 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 [21]:
```
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': 8, 'min_samples_leaf': 1, 'min_samples_split': 3, 'random_state': 0}
```

In [22]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=6, min_samples_leaf=1, min_samples_split=5, class_weight='bala
dtree.fit(X_train, y_train)
```

Out[22]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=8,
                       min_samples_split=3, random_state=0)
```

In [23]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred),2), "%")
```

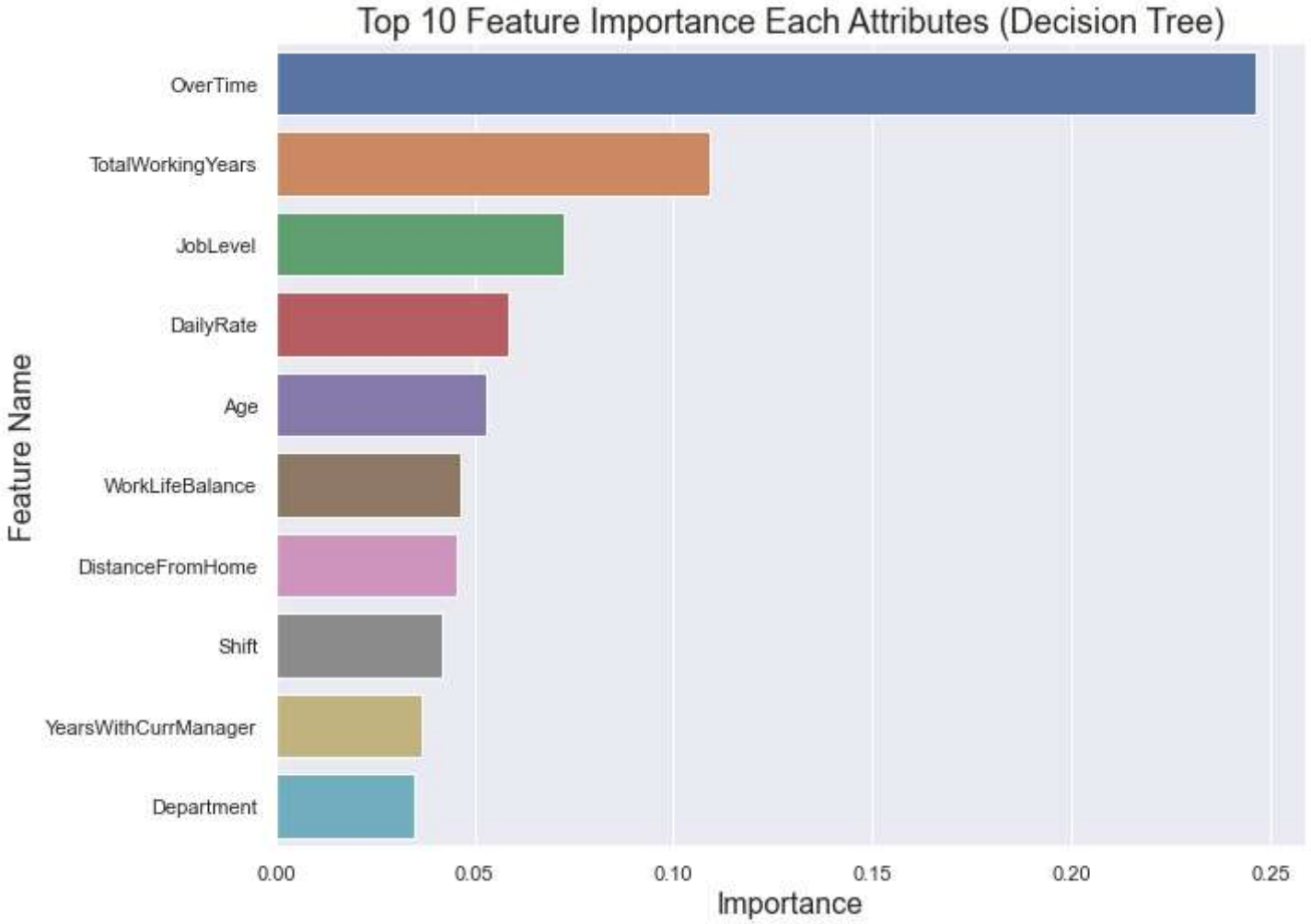
```
Accuracy Score : 87.8 %
```

In [24]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
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 :  0.8779761904761906
Precision Score :  0.8779761904761905
Recall Score :  0.8779761904761905
Jaccard Score :  0.7824933687002652
Log Loss :  4.214610186100187
```

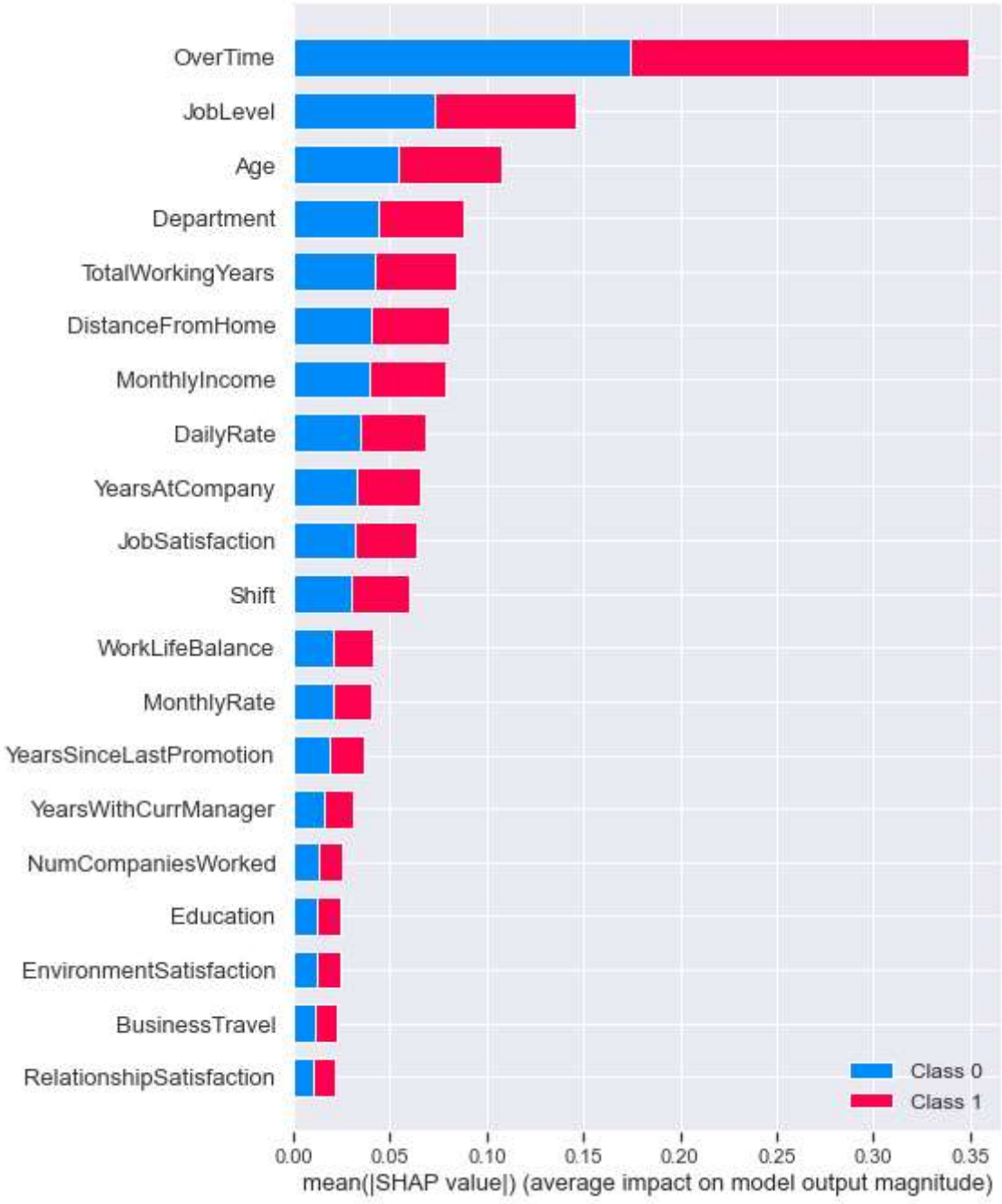
In [25]:
```
imp_df = DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance")
```

```
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')
plt.ylabel ('Feature Name')
plt.show()
```



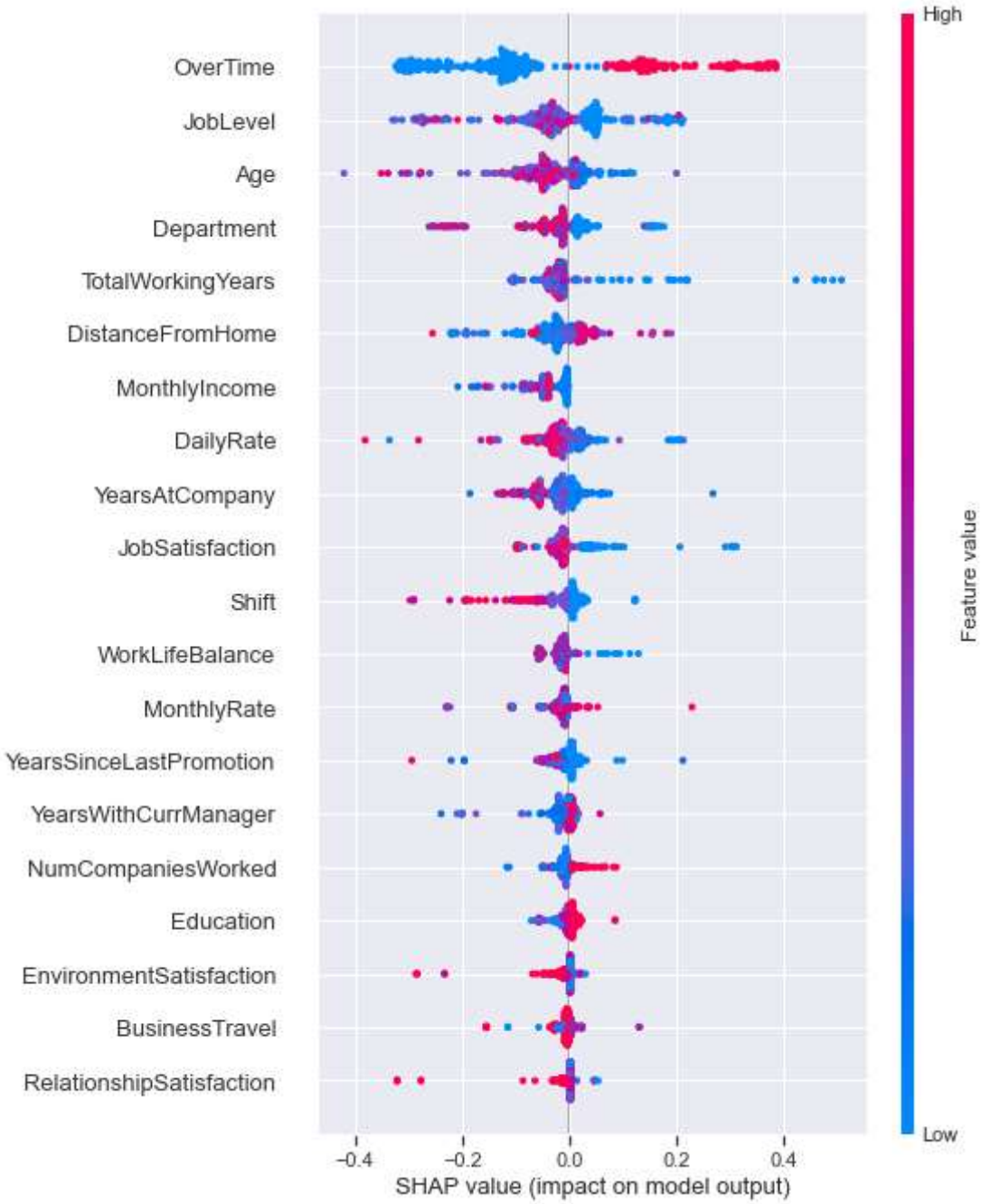
In [26]:
```
import shap
explainer = shap.TreeExplainer(tree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter consol
e)
```



In [27]:

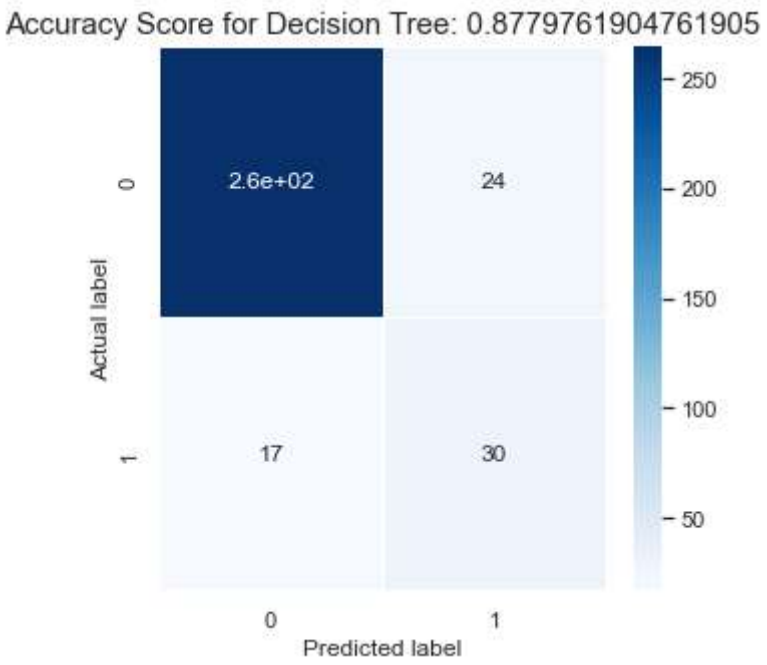
```
# compute SHAP values
explainer = shap.TreeExplainer(tree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



In [28]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

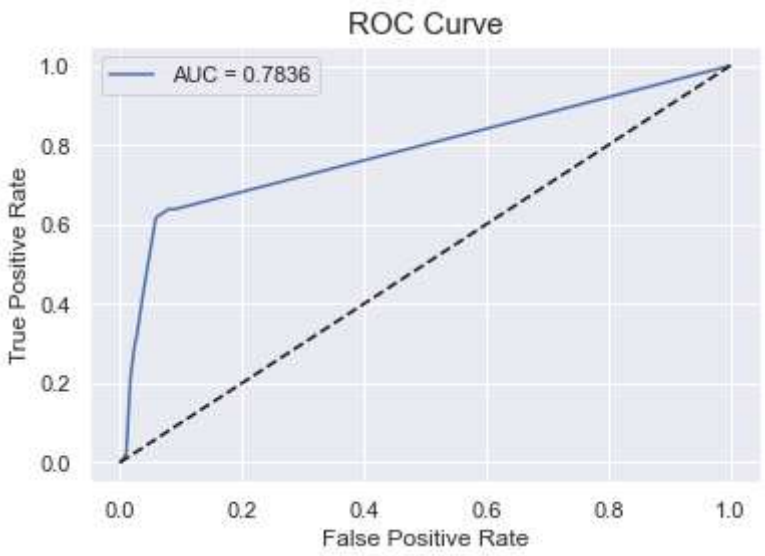
```
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=5, annot=True, map = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[28]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8779761904761905')



In [29]:
```
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(array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba, columns=
df_actual_predicted = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_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[29]: <matplotlib.legend.Legend at 0x21cafdafdf0>



Random Forest Classifier

In [30]:
```
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': 10, 'max_features': None, 'n_estimators': 100, 'random_state': 42}
```

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_depth=10, max_features=None, n_estimators=100, class_weight='balanced
rfc.fit(X_train, y_train)
```

Out[31]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10, max_features=None,
                       random_state=42)
```

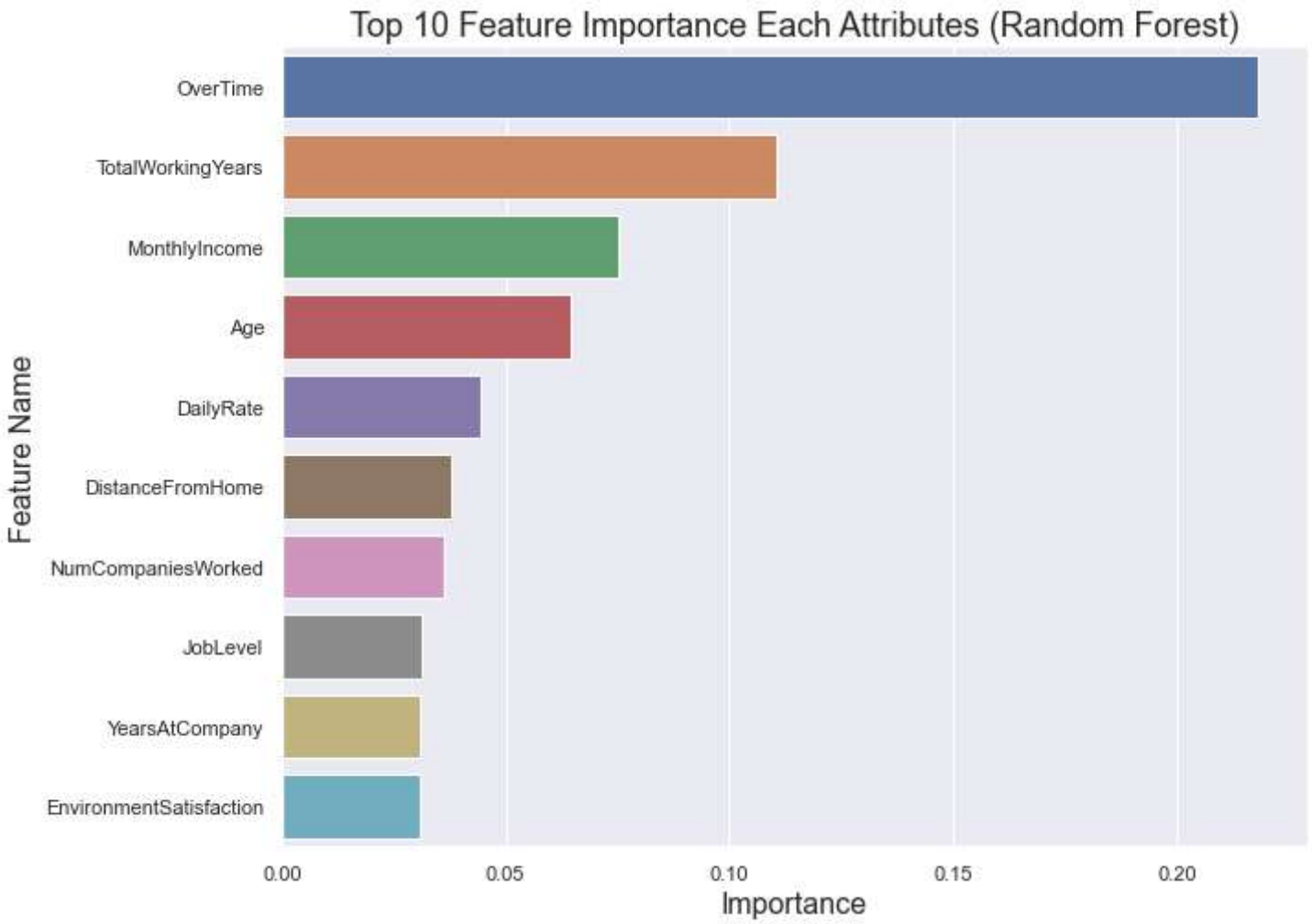
In [32]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)100 ,2) "%")
```

```
Accuracy Score : 91.07 %
```

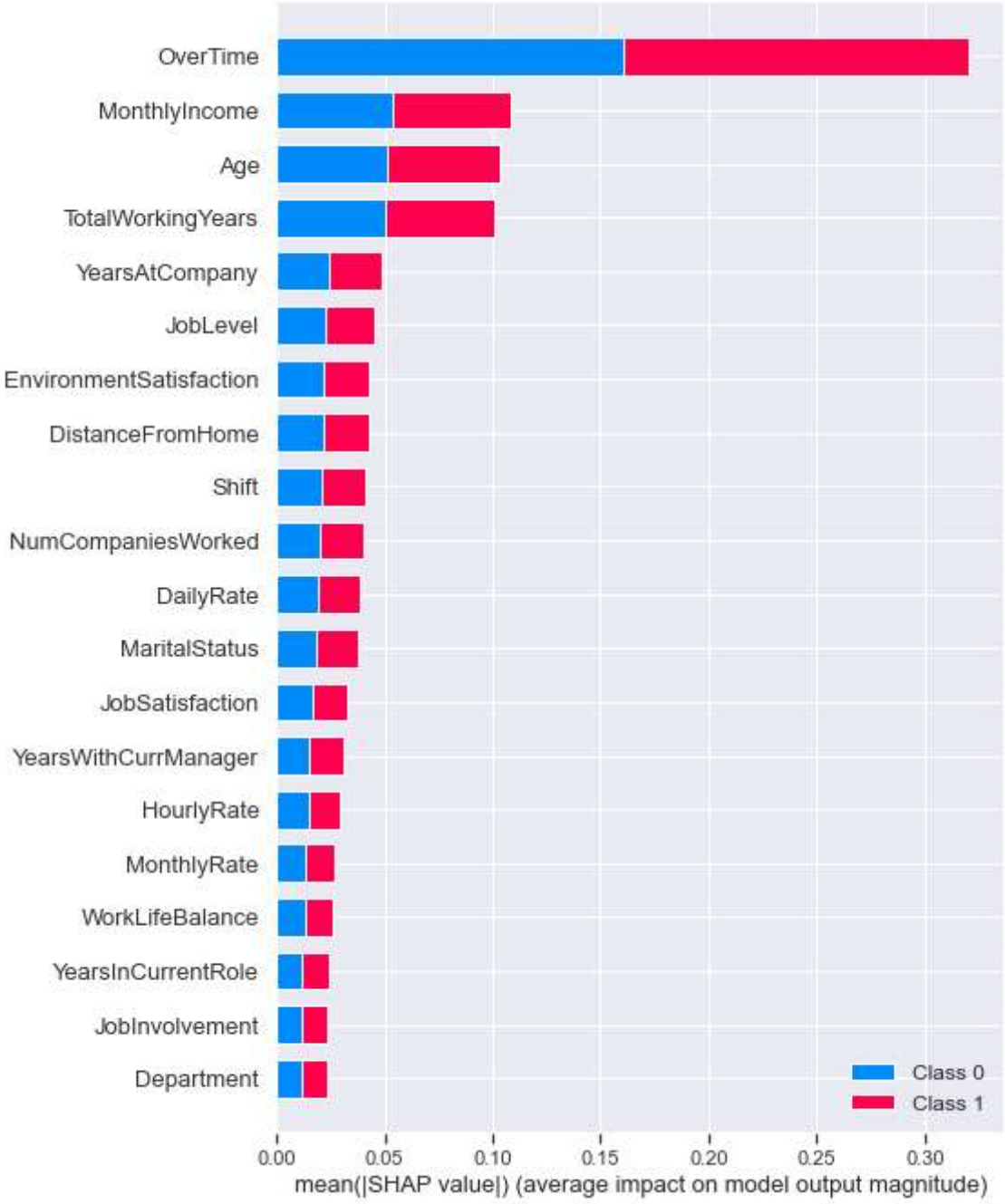
In [33]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.9107142857142857
Precision Score :  0.9107142857142857
Recall Score :  0.9107142857142857
Jaccard Score :  0.8360655737704918
Log Loss :  3.0838335994997084
```

In [34]:
```
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.plot(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()
```

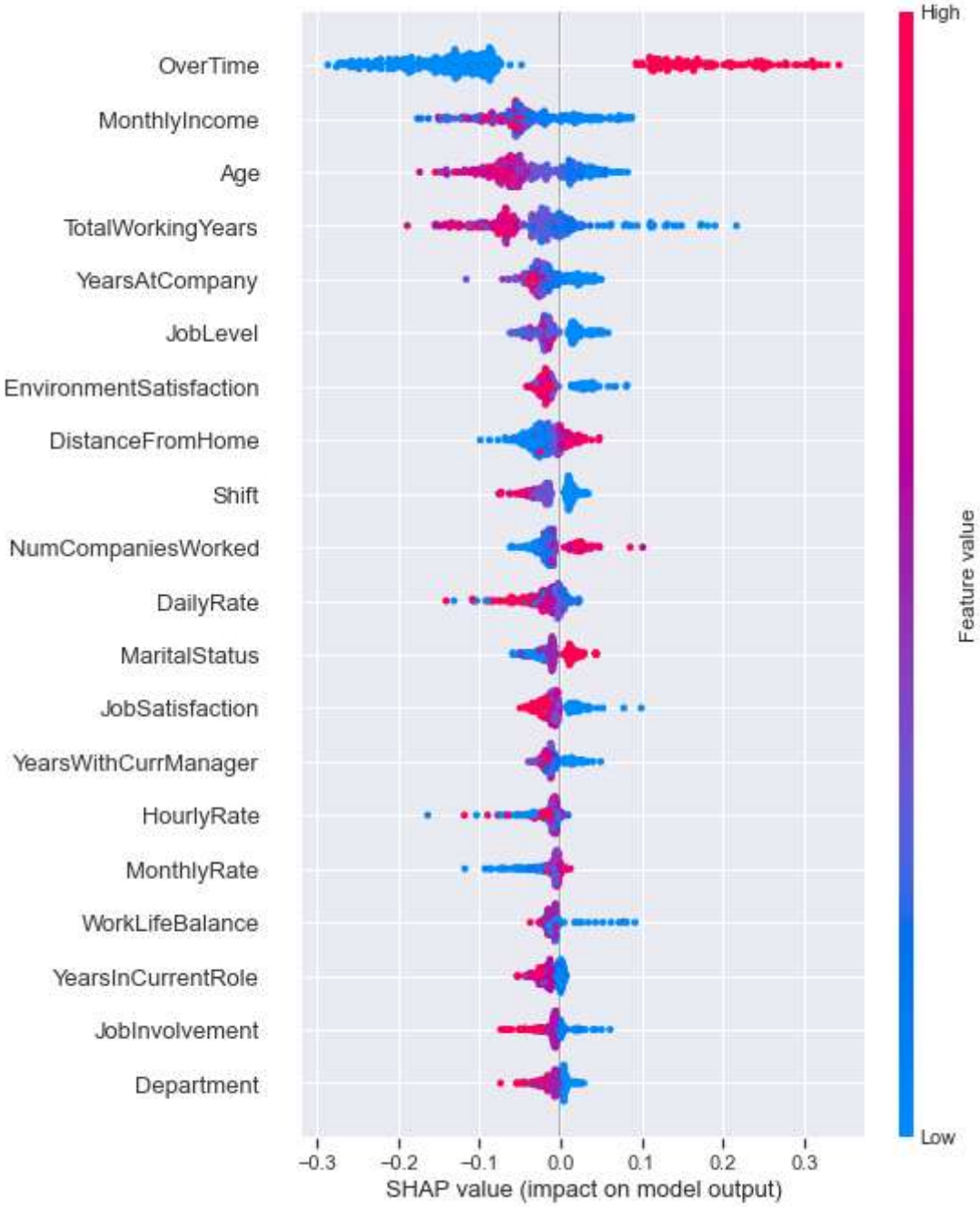


In [35]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



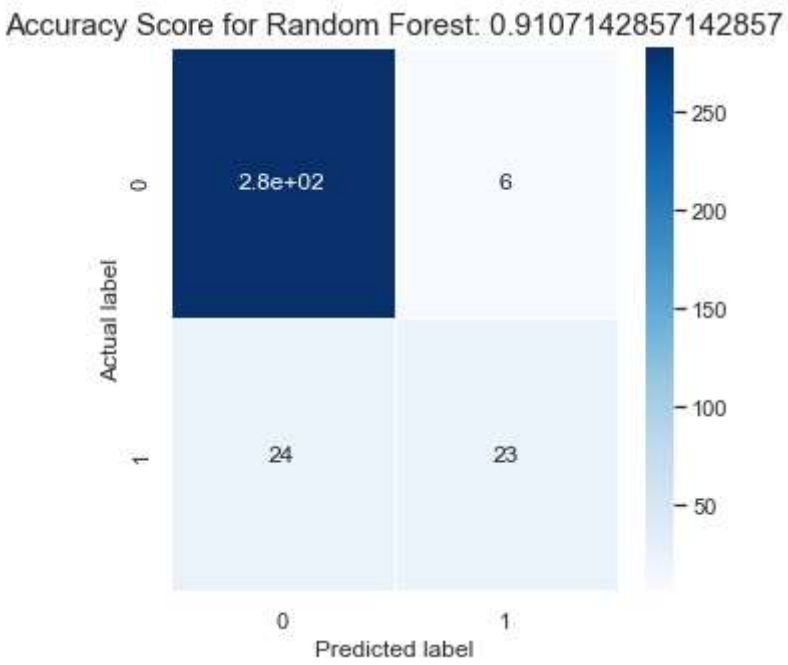
In [36]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
```

```
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



In [37]:
```
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,  map = '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)
```

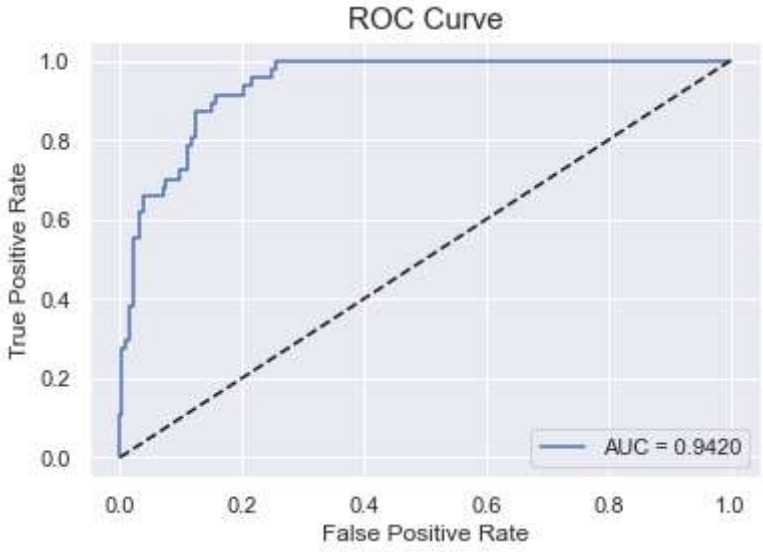
Out[37]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9107142857142857')
```



In [38]:
```
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']), DataFrame(y_pred_proba, columns=
df_actual_predicted.index = y_test.index

fpr, tpr = 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 = %04f' %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[38]: <matplotlib.legend.Legend at 0x21cafdfb5b0>



XGBoost Classifier

In [39]:
```
# Create an XGBoost classifier
xgb = XGBClassifier()

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

# 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': 3, 'n_estimators': 200}
```

In [40]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0, learning_rate=0.1, max_depth=3, n_estimators=200)
xgb.fit(X_train, y_train)
```

Out[40]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0, 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=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

In [41]:
```
from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

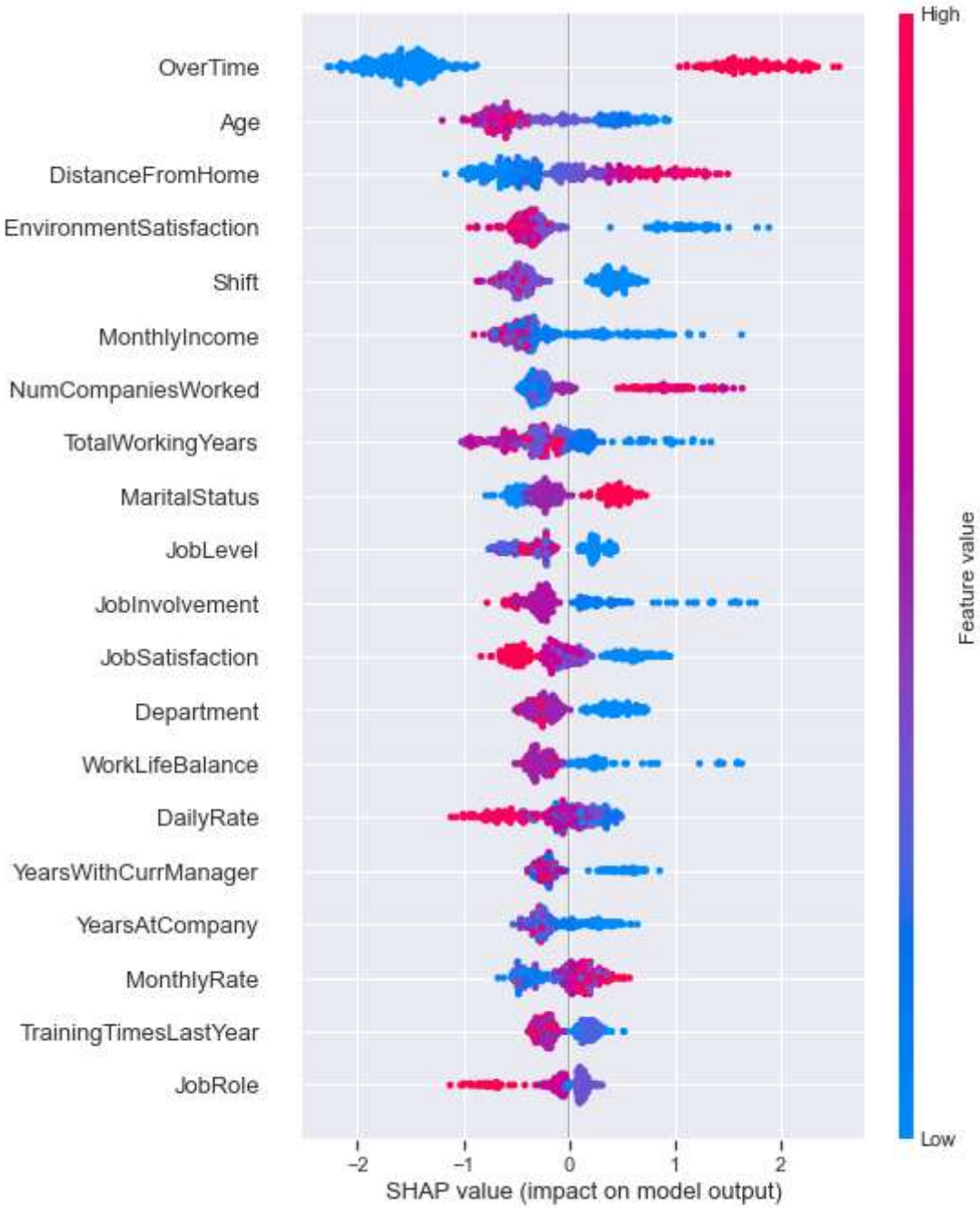
```
Accuracy Score : 91.96 %
```

In [42]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
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 :  0.9196428571428571
Precision Score :  0.9196428571428571
Recall Score :  0.9196428571428571
Jaccard Score :  0.8512396694214877
Log Loss :  2.7754540471565474
```

In [43]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
[19:13:18] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0cec3277c4d9d0165-1\xgboost\xgboost-ci-windows\src\c_api\c_api.cc:1240: Saving into deprecated binary model format, please consider using `json` or `ubj`. Model format will default to JSON in XGBoost 2.2 if not specified.
```

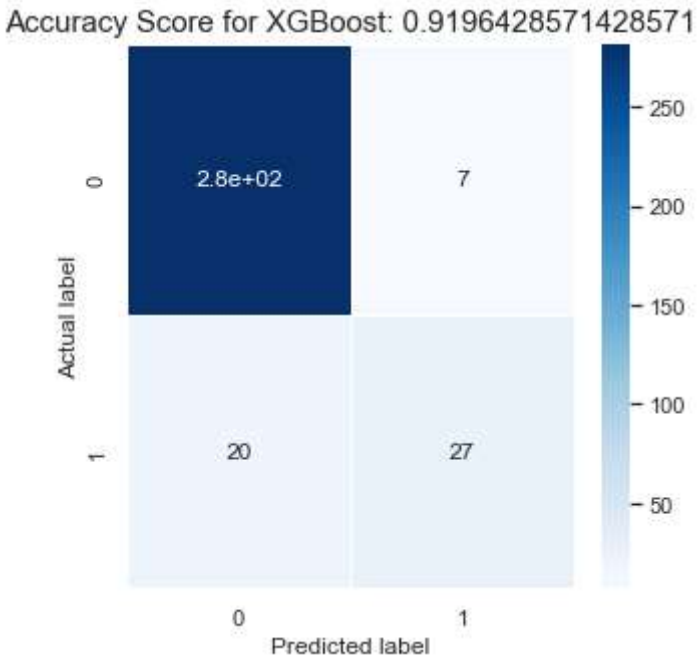


In [44]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
```

```
sns.heatmap(data=cm,linewidths=5, annot=True,  map = '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[44]:

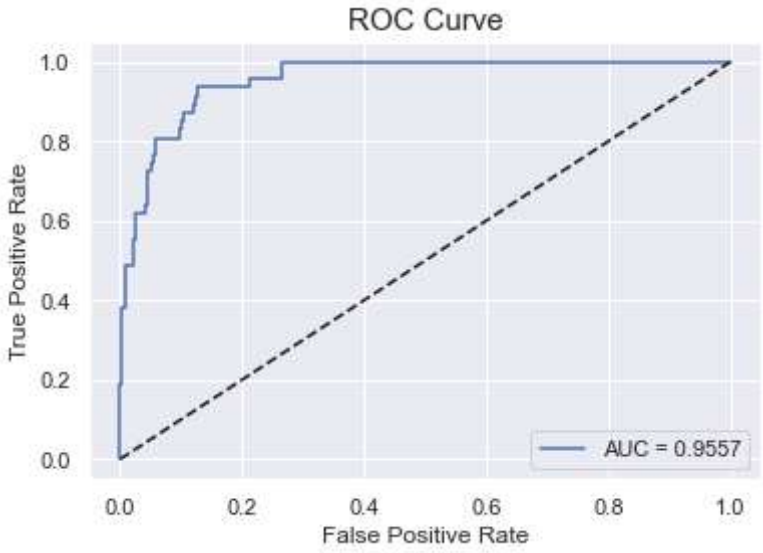
```
Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.9196428571428571')
```



In [45]:

```
y_pred_proba = xgb.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba, colum
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 = %04f' %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[45]: <matplotlib.legend.Legend at 0x21cb0ead520>



In [ ]: