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)
```

In [2]:
```
df = pd.read_csv('Employee.csv')
df.head()
```

Out[2]:

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	Lea
0	Bachelors	2017	Bangalore	3	34	Male	No	0	
1	Bachelors	2013	Pune	1	28	Female	No	3	
2	Bachelors	2014	New Delhi	3	38	Female	No	2	
3	Masters	2016	Bangalore	3	27	Male	No	5	
4	Masters	2017	Pune	3	24	Male	Yes	2	

Data Preprocessing Part 1

In [3]:
```
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
Education      3
City           3
Gender         2
EverBenched    2
dtype: int64
```

In [4]:
```
# Change the data type to string
df['LeaveOrNot'] = df['LeaveOrNot'].astype(str)
# Change 1 to yes and 0 to no for visualization
df['LeaveOrNot'] = df['LeaveOrNot'].map({'1': 'yes', '0': 'no'})
```

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

Out[5]:

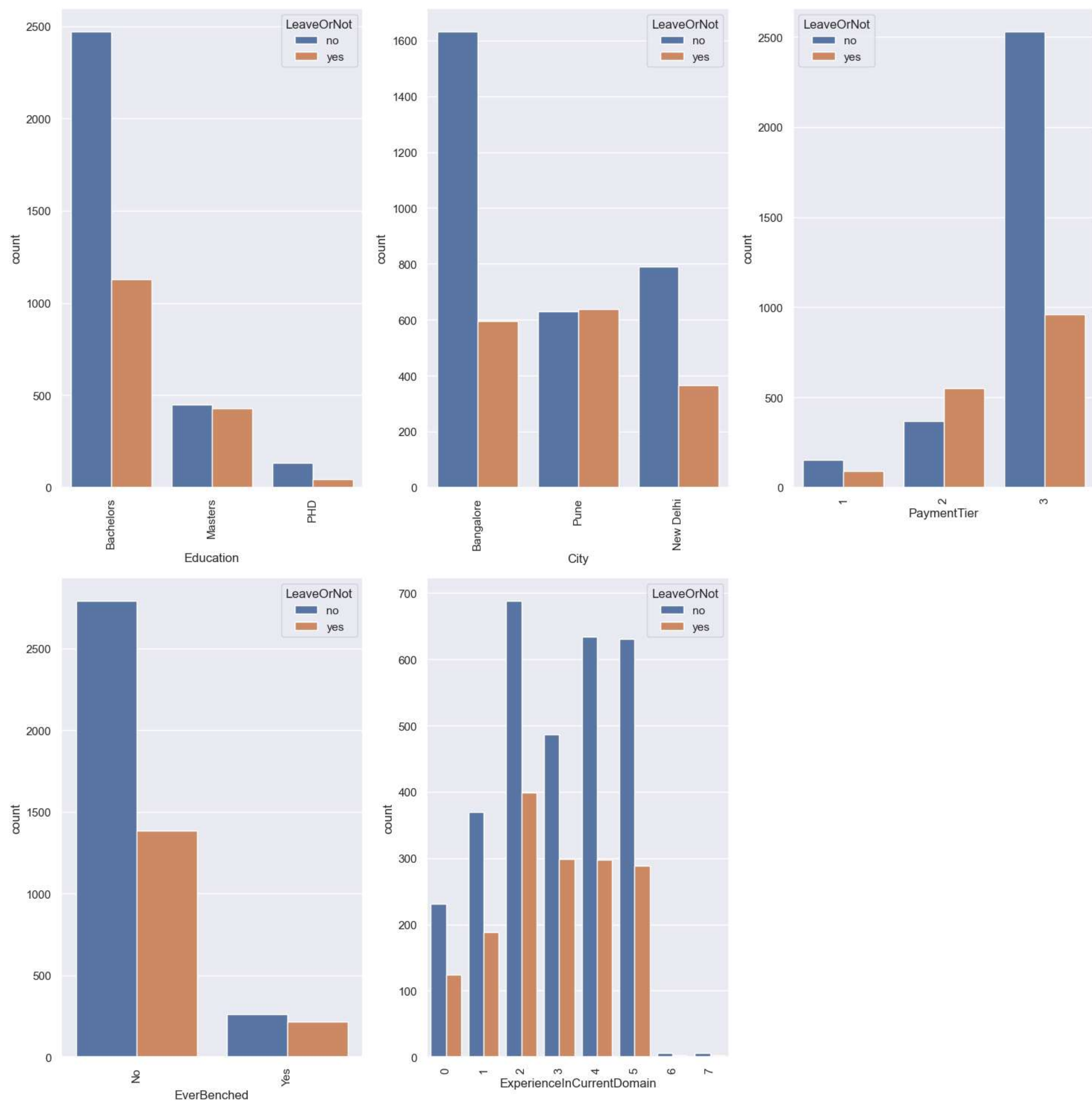
	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	Lea
0	Bachelors	2017	Bangalore	3	34	Male	No	0	
1	Bachelors	2013	Pune	1	28	Female	No	3	
2	Bachelors	2014	New Delhi	3	38	Female	No	2	
3	Masters	2016	Bangalore	3	27	Male	No	5	
4	Masters	2017	Pune	3	24	Male	Yes	2	

In [6]:
```
df.dtypes
```

Out[6]:
```
Education                    object
JoiningYear                   int64
City                         object
PaymentTier                   int64
Age                           int64
Gender                       object
EverBenched                  object
ExperienceInCurrentDomain     int64
LeaveOrNot                   object
dtype: object
```

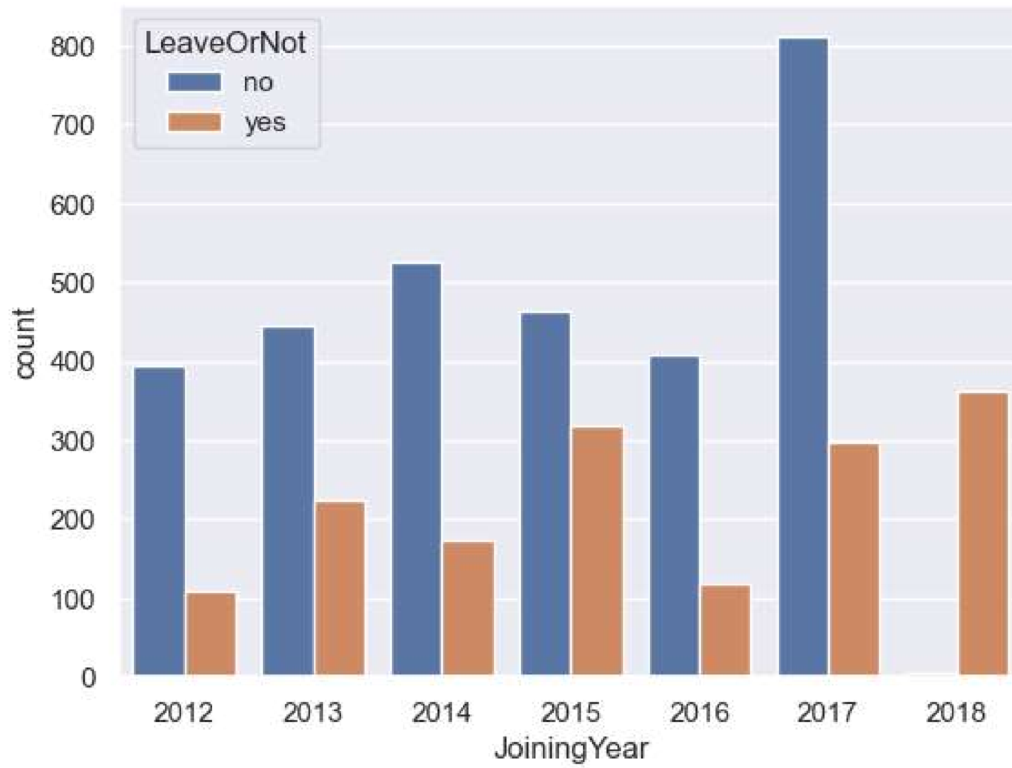
Exploratory Data Analysis

In [7]:
```
# list of categorical variables to plot
cat_vars = ['Education', 'City', 'PaymentTier', 'EverBenched', 'ExperienceInCurrentDomain']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='LeaveOrNot', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# remove the sixth subplot
fig.delaxes(axs[5])

# show plot
plt.show()
```

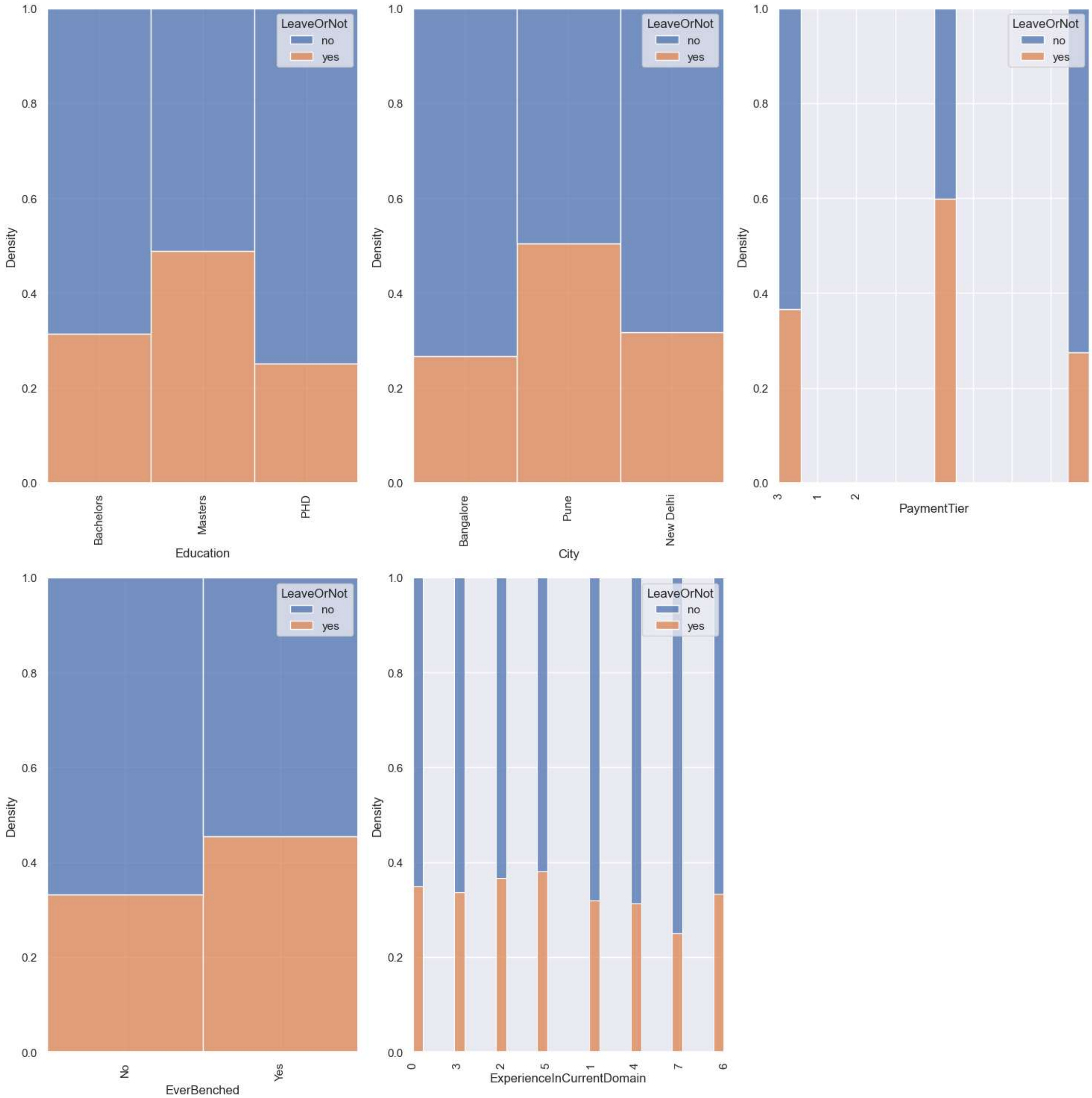


In [11]:
```
sns.countplot(x='JoiningYear', hue='LeaveOrNot', data=df)
```

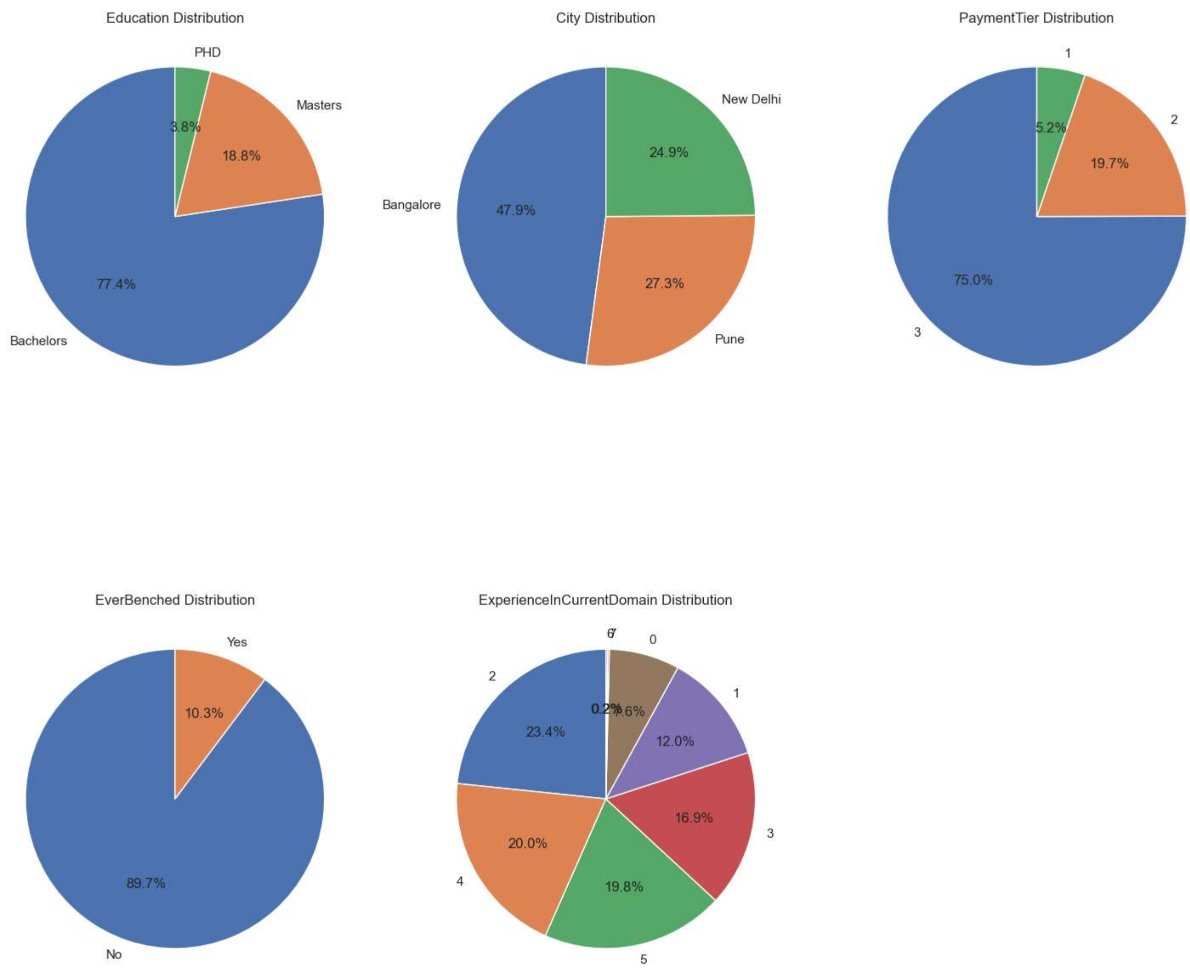
Out[11]: <AxesSubplot:xlabel='JoiningYear', ylabel='count'>



In [8]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Education', 'City', 'PaymentTier', 'EverBenched', 'ExperienceInCurrentDomain']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))
axs = axs.flatten()

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

# adjust spacing between subplots
fig.tight_layout()

# remove the sixth subplot
fig.delaxes(axs[5])

# show plot
plt.show()
```

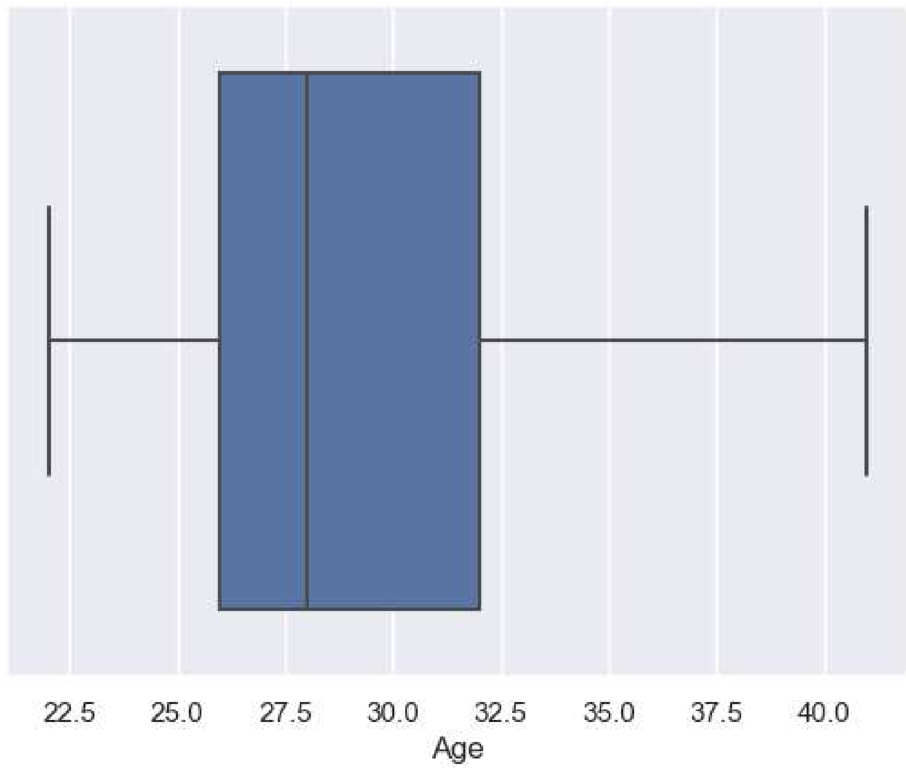

LeaveOrNot
no
yes
Density
Education
Bachelors
Masters
PHD
City
Bangalore
Pune
New Delhi
PaymentTier
3
1
2
EverBenched
No
Yes
ExperienceInCurrentDomain
0
3
2
5
1
4
7
6
0.0
0.2
0.4
0.6
0.8
1.0

In [9]:
```
cat_vars = ['Education', 'City', 'PaymentTier', 'EverBenched', 'ExperienceInCurrentDomain']

# create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))

# create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle

        # set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# adjust spacing between subplots
fig.tight_layout()
fig.delaxes(axs[1][2])
# show the plot
plt.show()
```



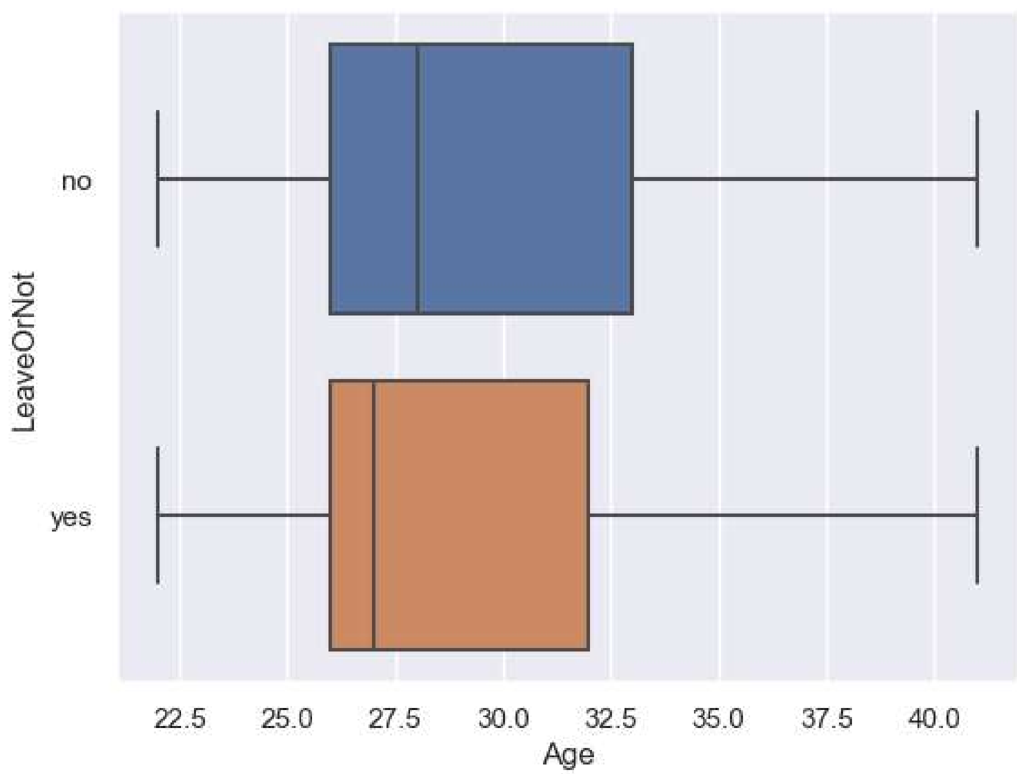
In [12]:
```
sns.boxplot(x='Age', data=df)
```

Out[12]: <AxesSubplot:xlabel='Age'>



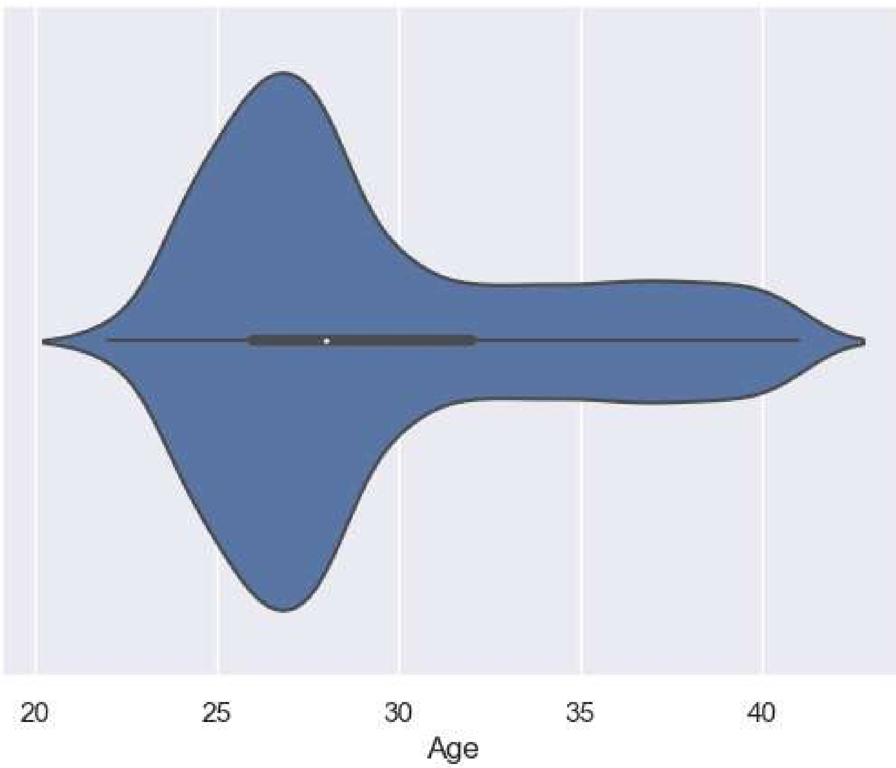
In [13]:
```
sns.boxplot(x='Age', data=df, y='LeaveOrNot')
```

Out[13]: <AxesSubplot:xlabel='Age', ylabel='LeaveOrNot'>



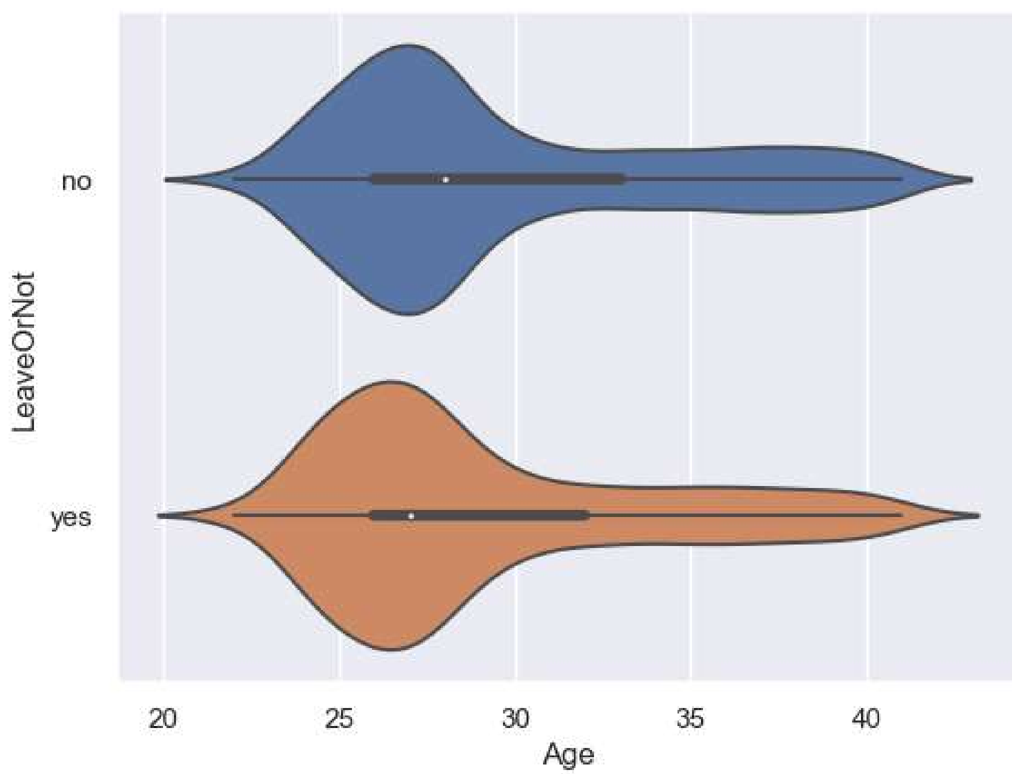
In [14]:
```
sns.violinplot(x='Age', data=df)
```

Out[14]: <AxesSubplot:xlabel='Age'>



In [15]:
```
sns.violinplot(x='Age', data=df, y='LeaveOrNot')
```

Out[15]: <AxesSubplot:xlabel='Age', ylabel='LeaveOrNot'>



Data Preprocessing Part 2

In [16]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[16]:
```
Series([], dtype: float64)
```

Label Encoding for Object datatypes

In [17]:
```
# 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()}")
```

```
Education: ['Bachelors' 'Masters' 'PHD']
City: ['Bangalore' 'Pune' 'New Delhi']
Gender: ['Male' 'Female']
EverBenched: ['No' 'Yes']
LeaveOrNot: ['no' 'yes']
```

In [18]:
```
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()}")
```

```
Education: [0 1 2]
City: [0 2 1]
Gender: [1 0]
EverBenched: [0 1]
LeaveOrNot: [0 1]
```

In [19]:
```
df.head()
```

Out[19]:

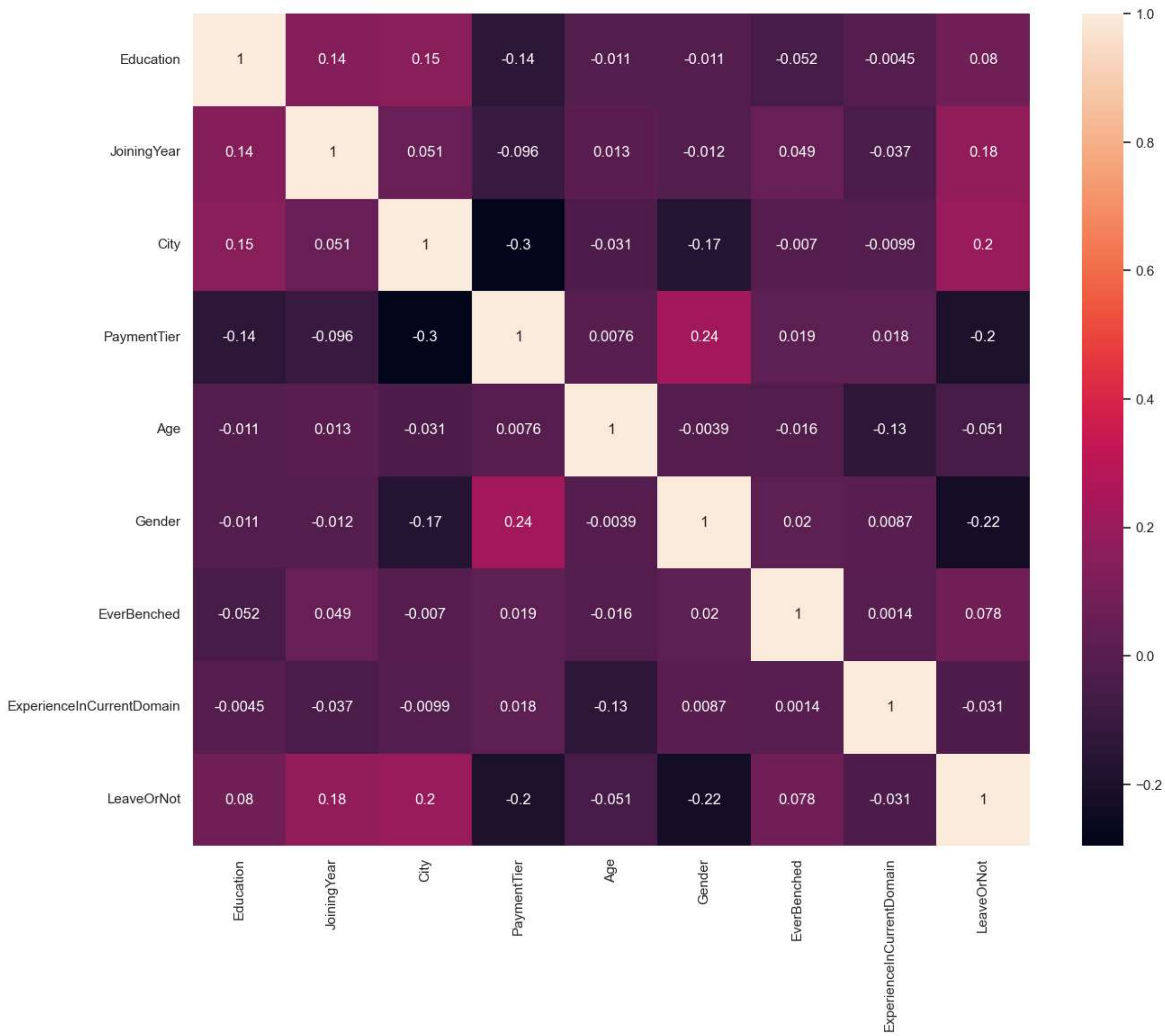
	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	LeaveOrI
0	0	2017	0	3	34	1	0	0	
1	0	2013	2	1	28	0	0	3	
2	0	2014	1	3	38	0	0	2	
3	1	2016	0	3	27	1	0	5	
4	1	2017	2	3	24	1	1	2	

In [ ]:
```
# There's no outlier in the dataset and we can balanced the class imbalanced using
# class_weight='balanced' in machine learning model
```

Correlation Heatmap

In [20]:
```
plt.figure(figsize=(15,12))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[20]: <AxesSubplot:>



Train Test Split

In [21]:
```
X = df.drop('LeaveOrNot', axis=1)
y = df['LeaveOrNot']
```

In [22]:
```
#test size 20% and train size 80%
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)
```

Decision Tree

In [23]:
```
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': 7, 'min_samples_leaf': 1, 'min_samples_split': 3, 'random_state': 0}
```

In [24]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=7, min_samples_leaf=1, min_sample
dtree.fit(X_train, y_train)
```

Out[24]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=7,
                       min_samples_split=3, random_state=0)
```

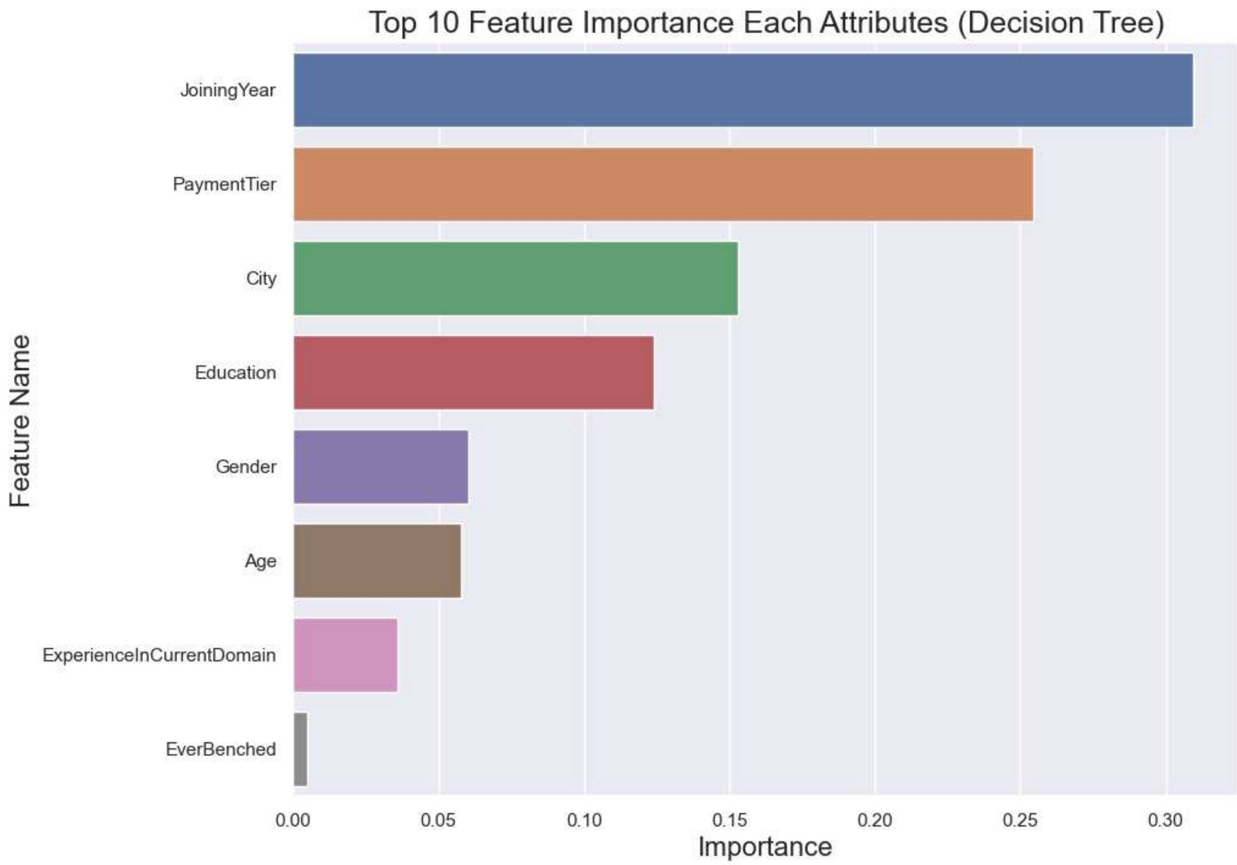
In [25]:
```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 82.92 %
```

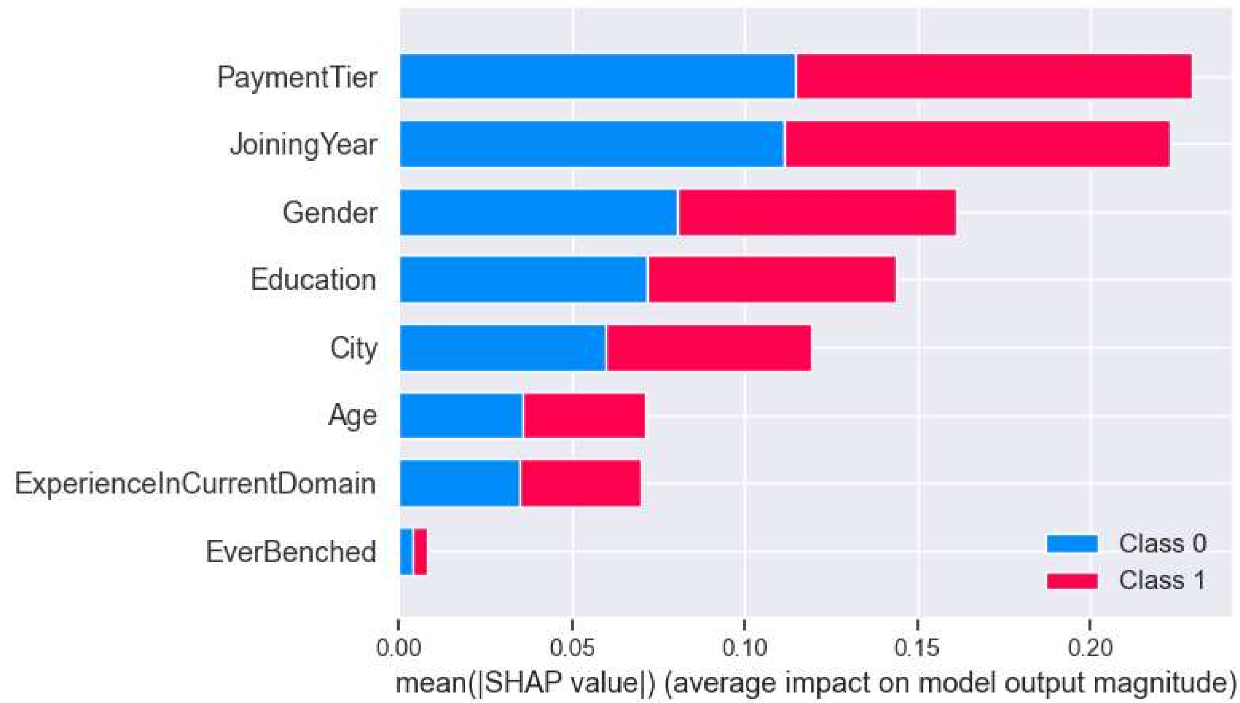
In [26]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jacca
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, 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 :  0.8292158968850698
Precision Score :  0.8292158968850698
Recall Score :  0.8292158968850698
Jaccard Score :  0.708256880733945
Log Loss :  5.89873063396476
```

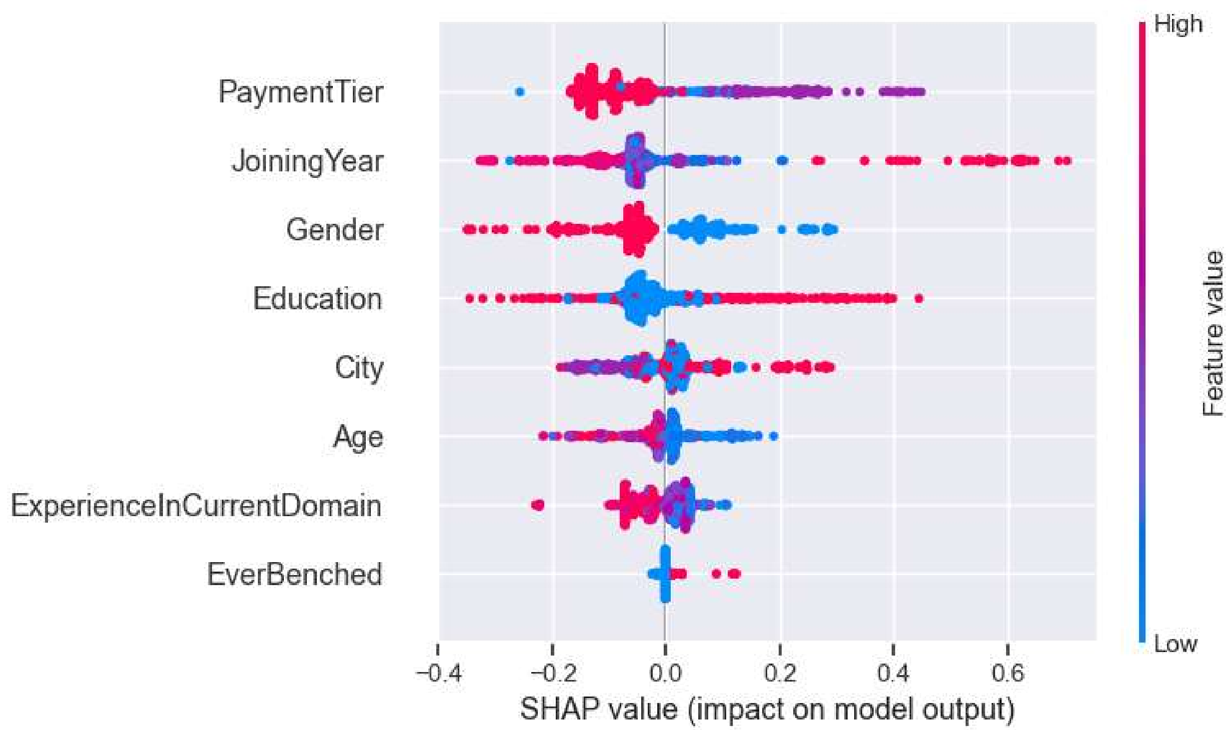
In [27]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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()
```



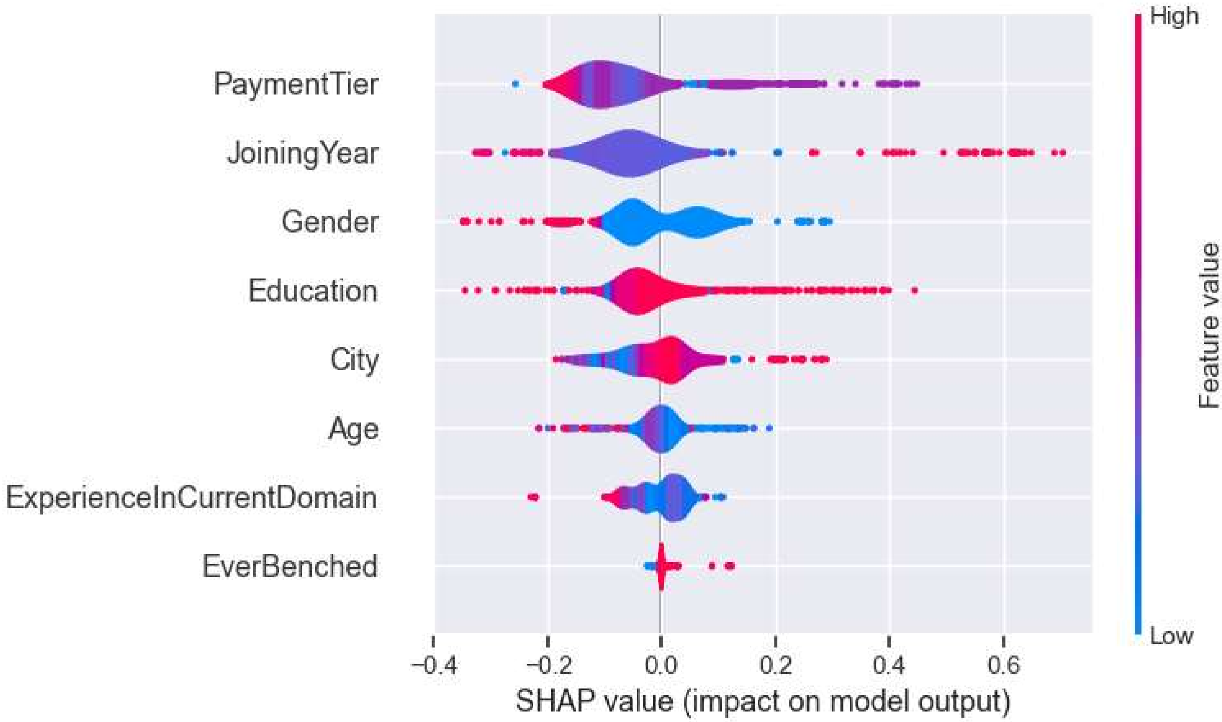
In [28]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [29]:
```
# 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)
```

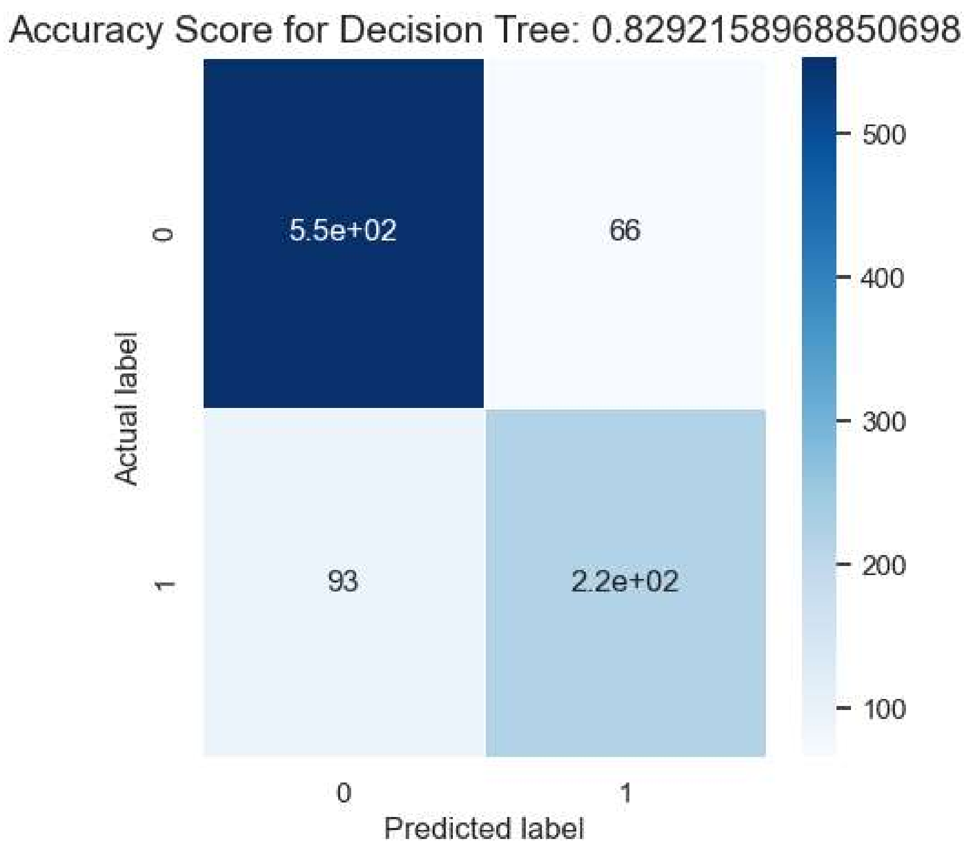


In [32]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type
```



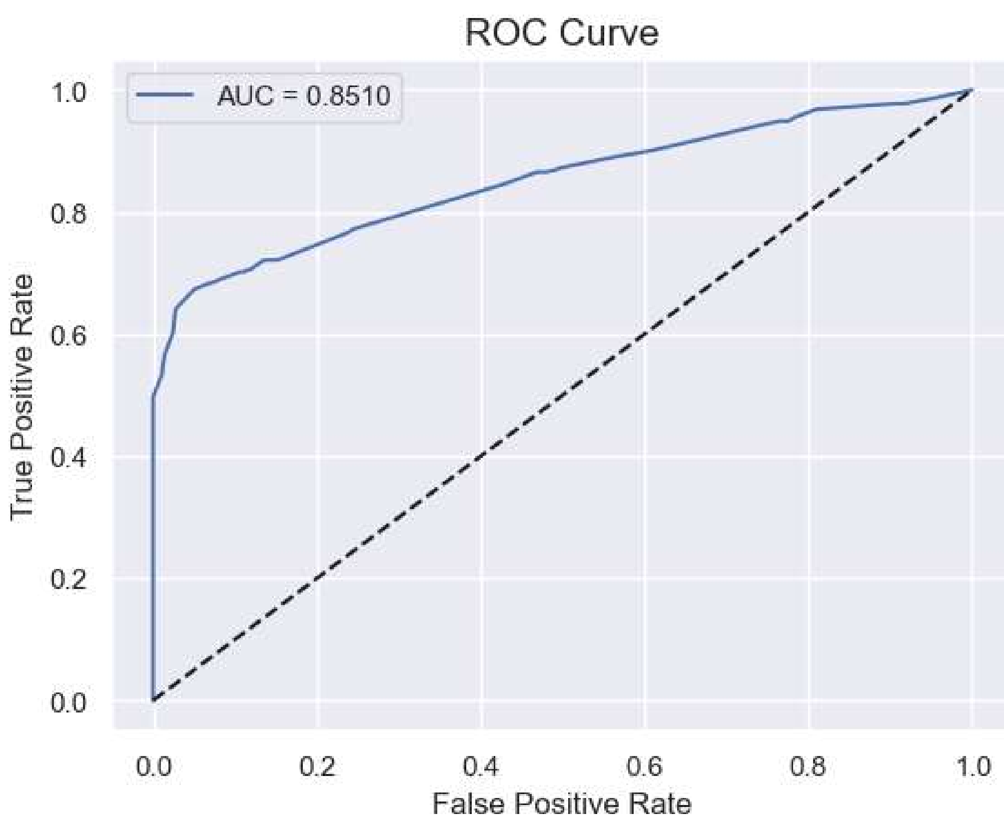
In [33]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_te
plt.title(all_sample_title, size = 15)
```

Out[33]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8292158968850698')
```



In [34]:
```
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.I
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[34]: `<matplotlib.legend.Legend at 0x2265509fd90>`



Random Forest

In [35]:
```
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': 10, 'max_features': 'sqrt', 'n_estimators': 100, 'random_state': 0}
```

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='sqrt', n_estimators=100, class_w
rfc.fit(X_train, y_train)
```

Out[36]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10,
                       max_features='sqrt', random_state=0)
```

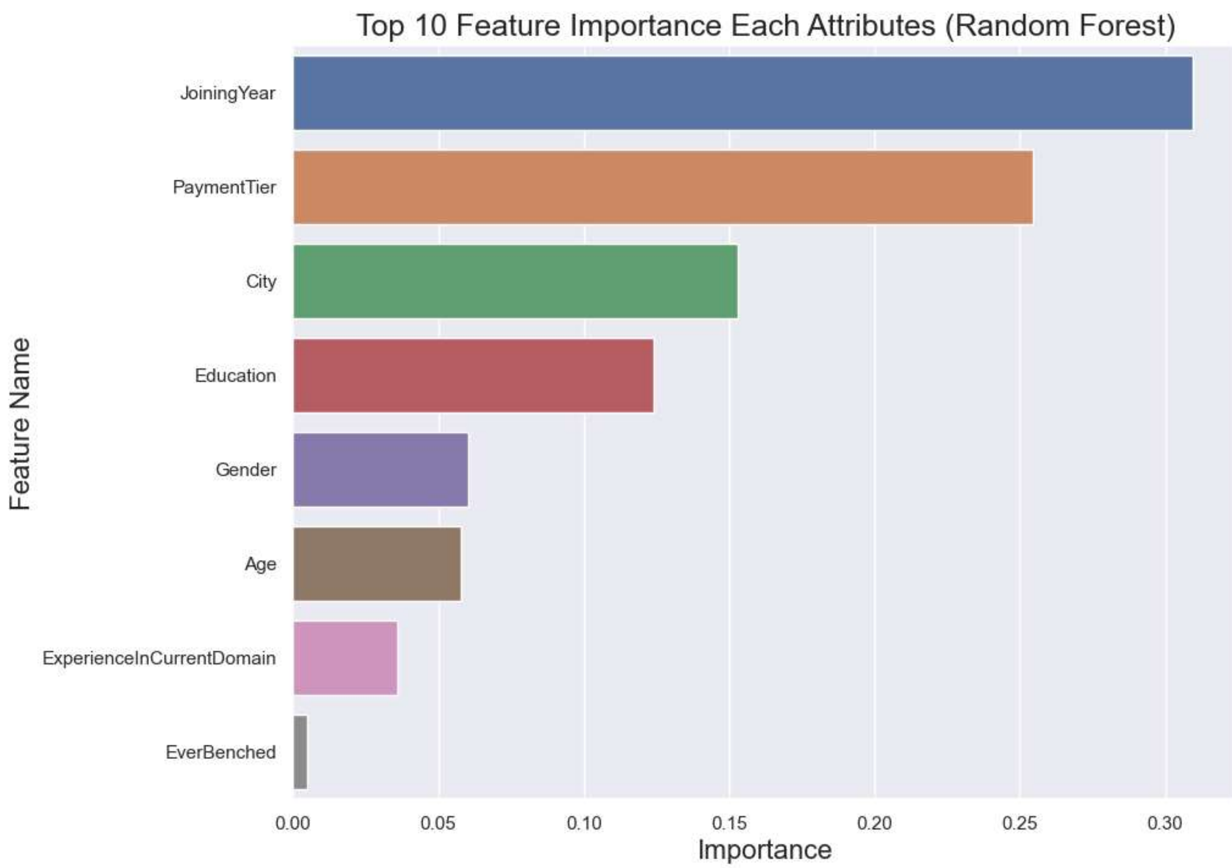
In [37]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 83.67 %
```

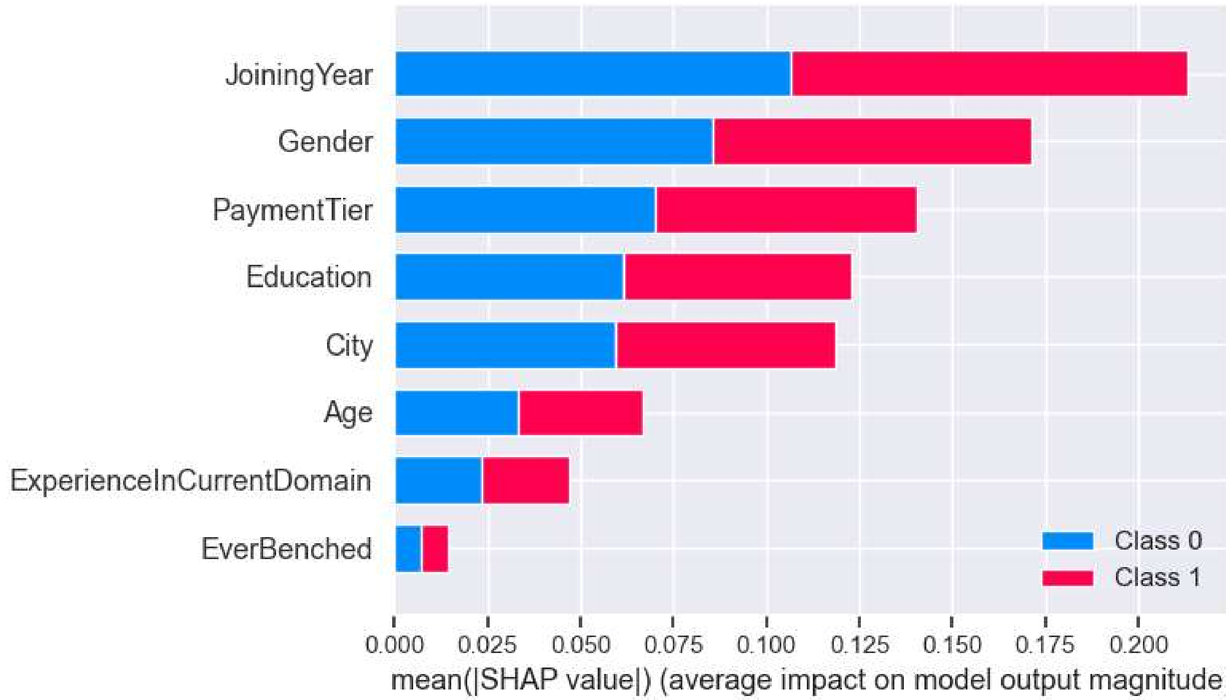
In [38]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccar
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, 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 :  0.8367346938775511
Precision Score :  0.8367346938775511
Recall Score :  0.8367346938775511
Jaccard Score :  0.7192982456140351
Log Loss :  5.639030279578576
```

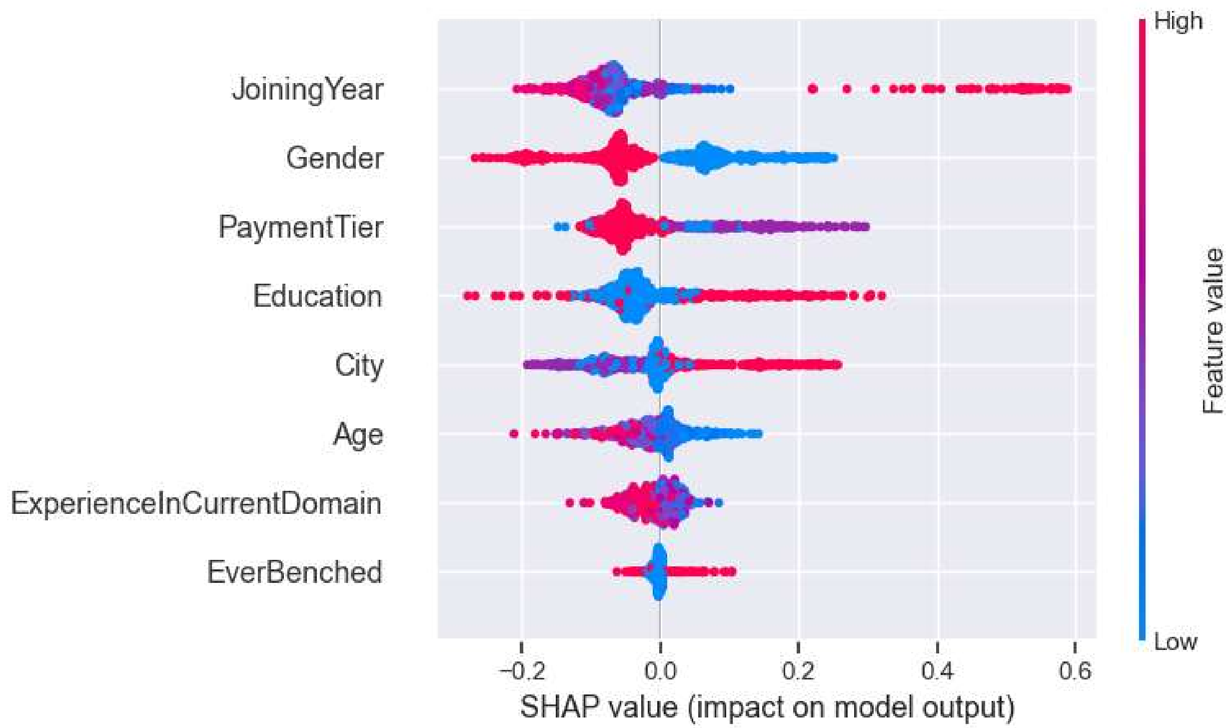
In [39]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



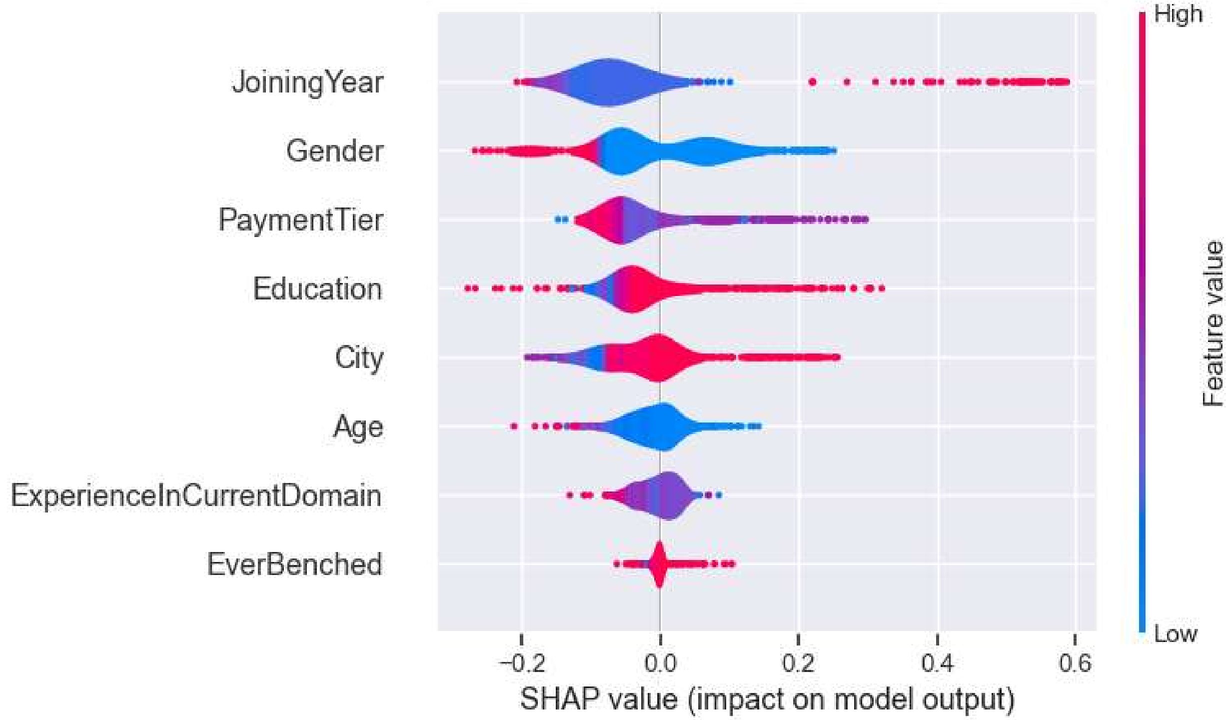
In [40]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [41]:
```
# 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)
```

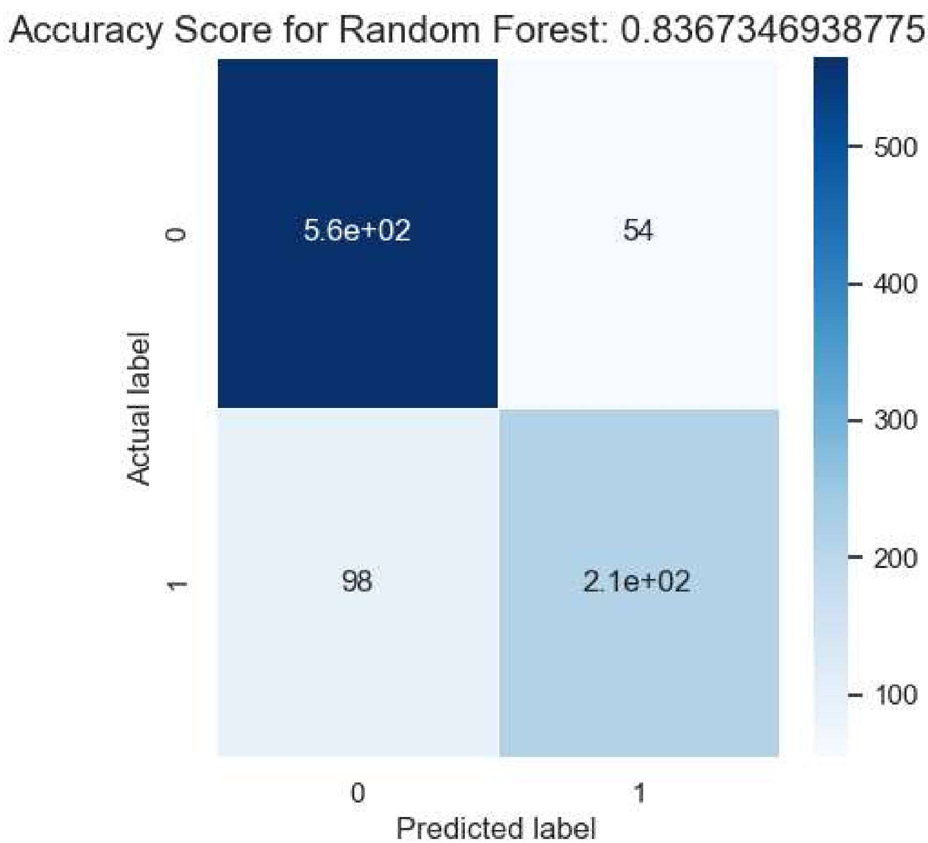


In [42]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type
```



In [43]:
```
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[43]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.8367346938775511')
```



In [44]:
```
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.I
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[44]: <matplotlib.legend.Legend at 0x22654d71d90>