In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_theme(color_codes=True)
```

In [9]:
```
df = pd.read_csv('TravelInsurancePrediction.csv')
df.head()
```

Out[9]:

	Unnamed: 0	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	0	31	Government Sector	Yes	400000	6	1	No	No	0
1	1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0
2	2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1
3	3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0
4	4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0

Data Preprocessing Part 1

In [10]:
```
df.select_dtypes(include='object').nunique()
```

Out[10]:
```
Employment Type        2
GraduateOrNot          2
FrequentFlyer          2
EverTravelledAbroad    2
dtype: int64
```

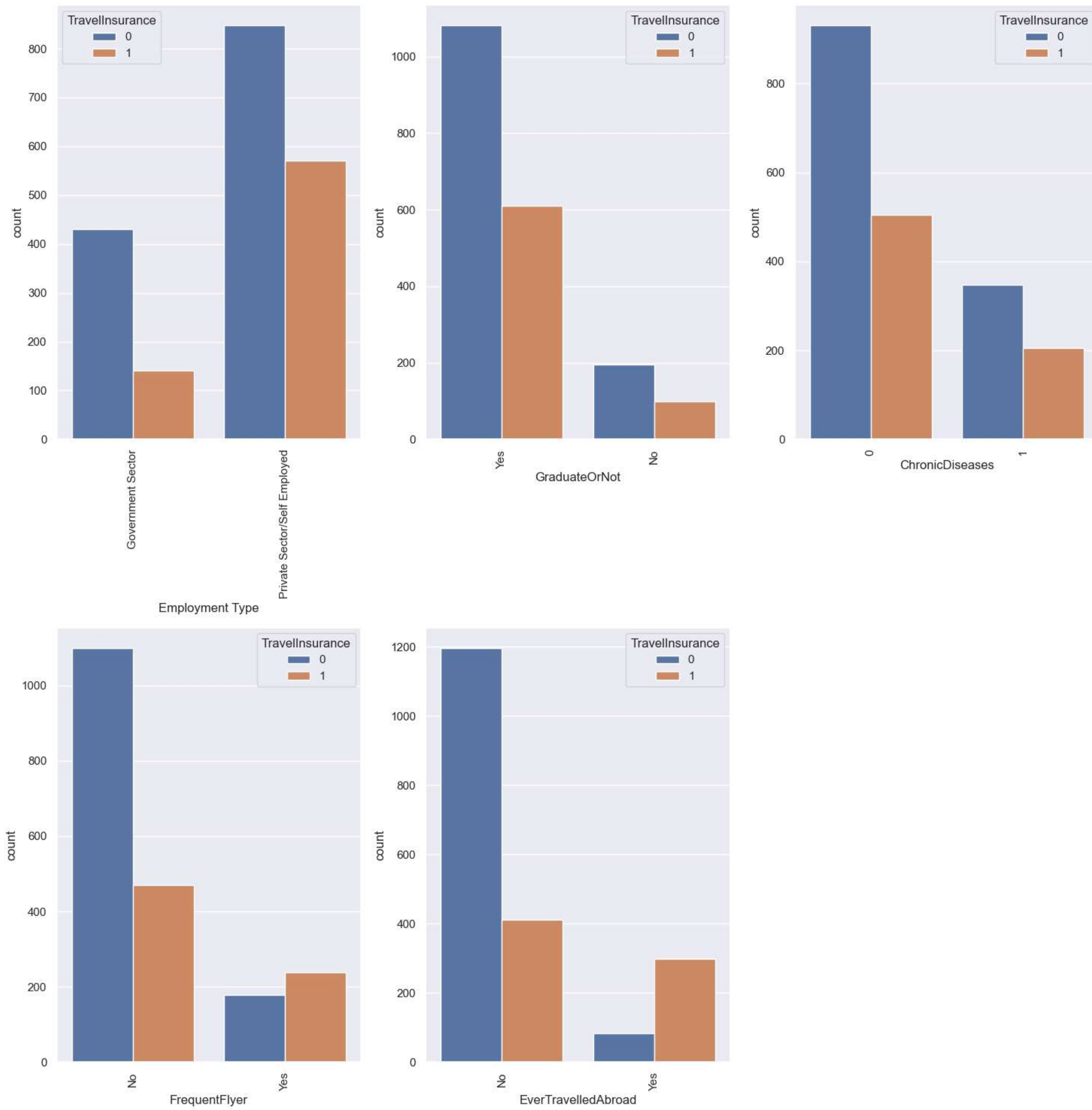
In [11]:
```
# Remove Unnamed: 0 attributes because its unnecessary for prediction
df.drop(columns='Unnamed: 0', inplace=True)
df.head()
```

Out[11]:

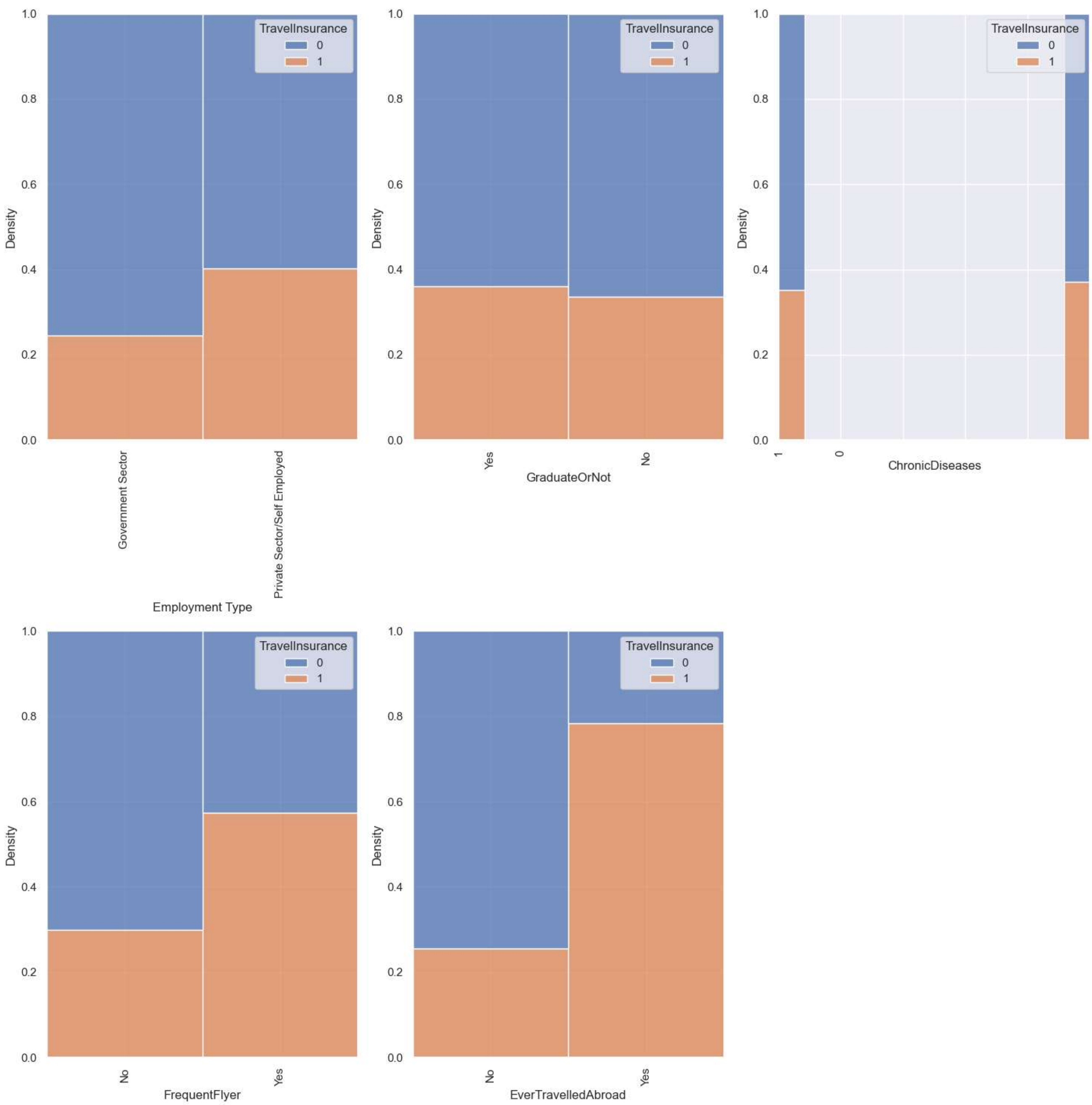
	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	31	Government Sector	Yes	400000	6	1	No	No	0
1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0
2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1
3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0
4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0

Exploratory Data Analysis

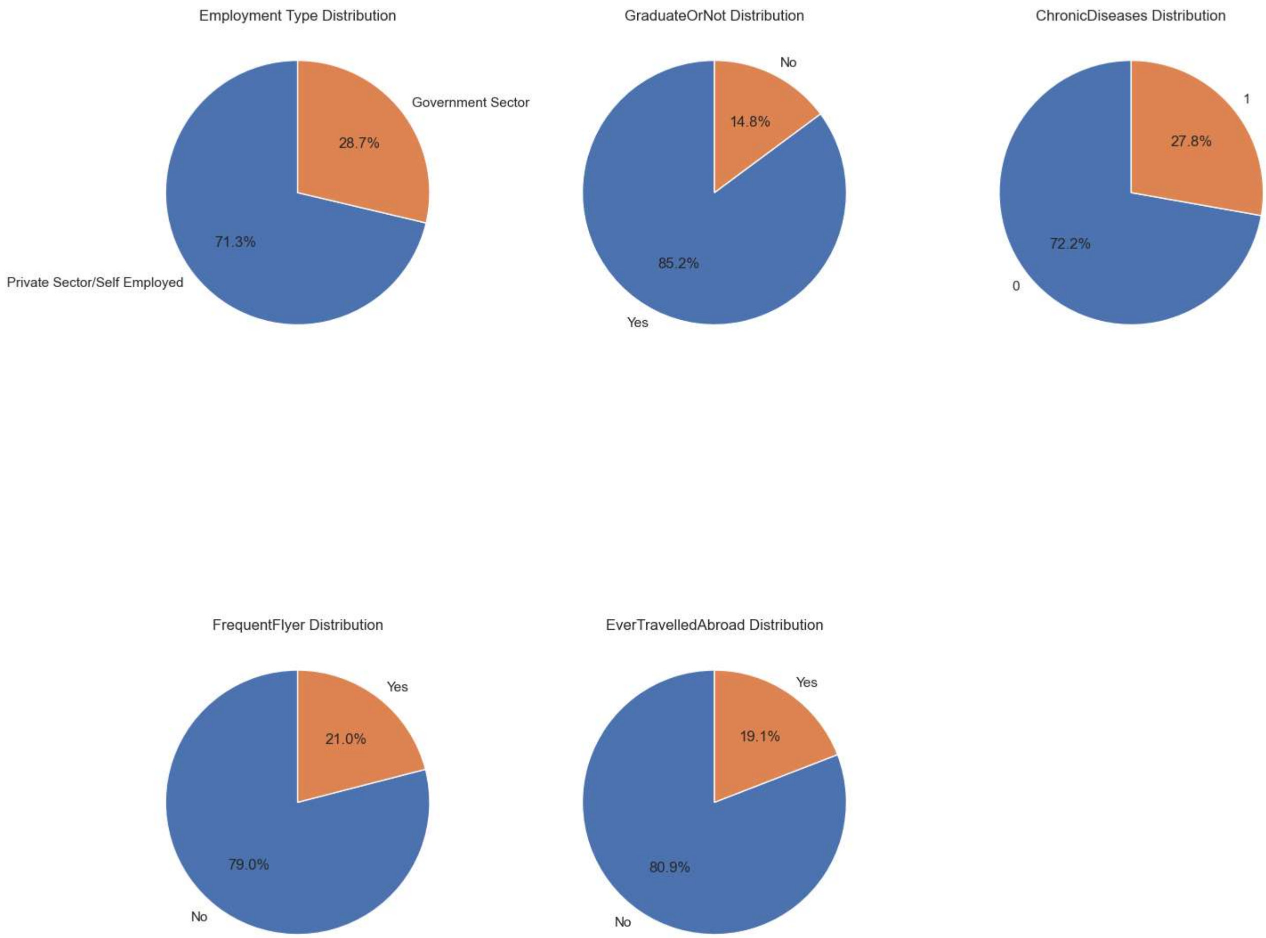
In [12]:
```
# list of categorical variables to plot
cat_vars = ['Employment Type', 'GraduateOrNot', 'ChronicDiseases', 'FrequentFlyer', 'EverTravelledAbroad']

# 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='TravelInsurance', 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()
```



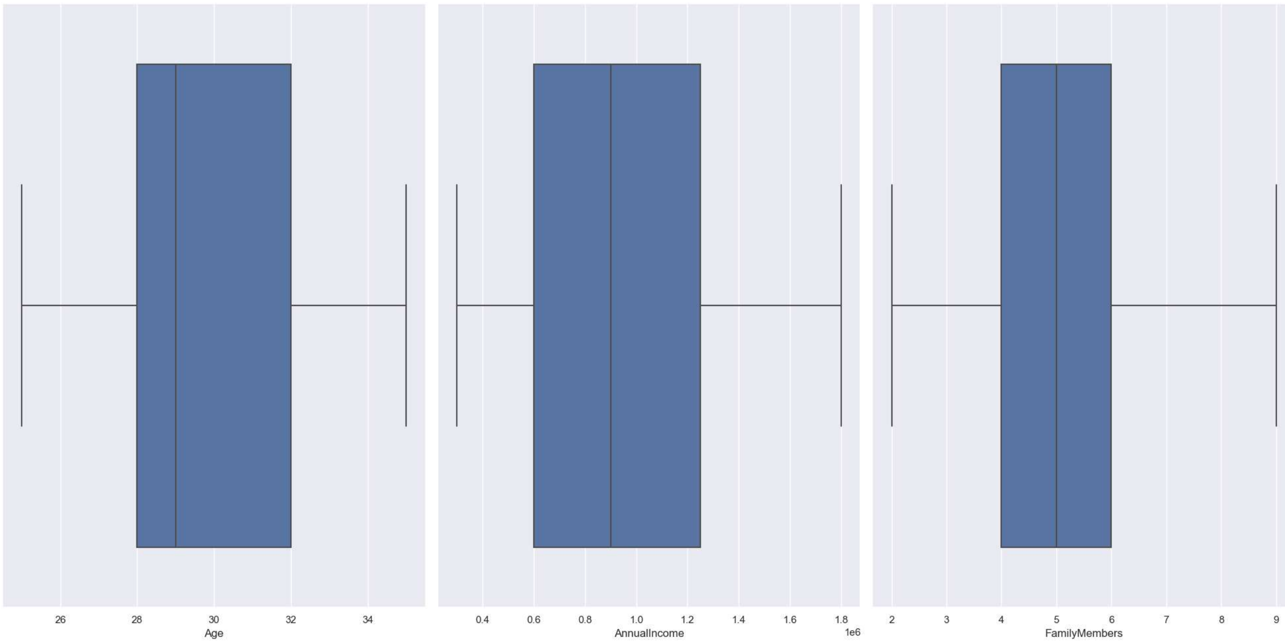
In [13]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Employment Type', 'GraduateOrNot', 'ChronicDiseases', 'FrequentFlyer', 'EverTravelledAbroad']

# 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='TravelInsurance', data=df, ax=axs[i], multiple="fill", kde=False, element="bars", fill=True, 
    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()
```



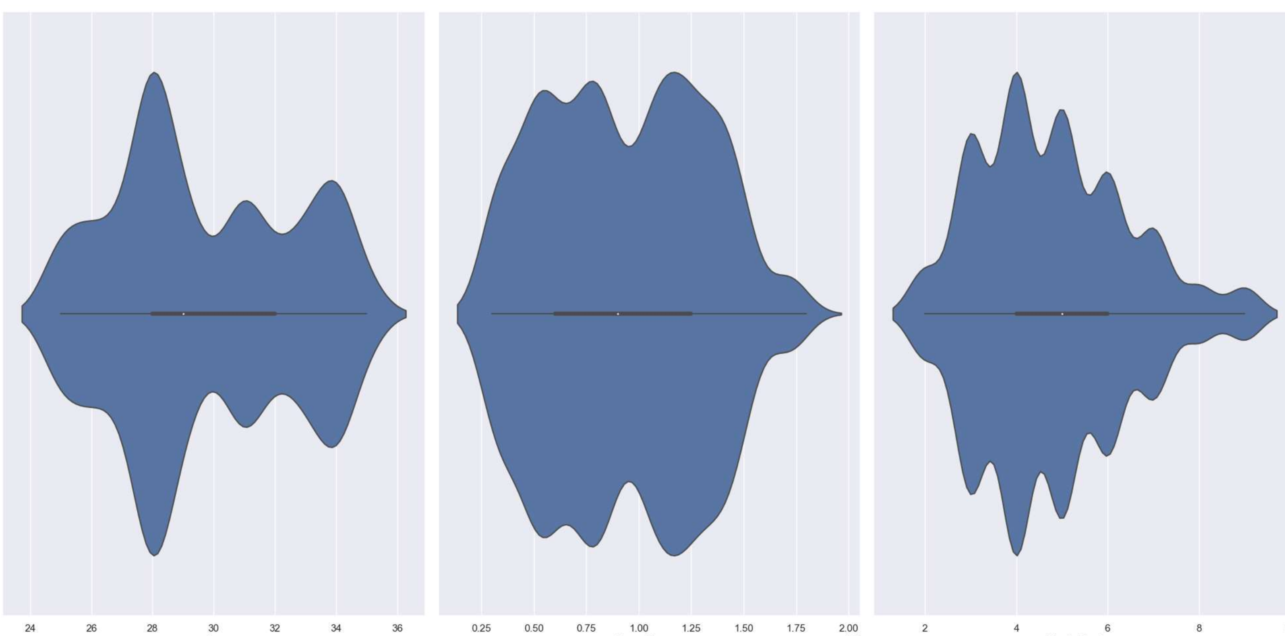
In [14]:
```python
cat_vars = ['Employment Type', 'GraduateOrNot', 'ChronicDiseases', 'FrequentFlyer', 'EverTravelledAbroad']

# 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=90)

        # 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()
```



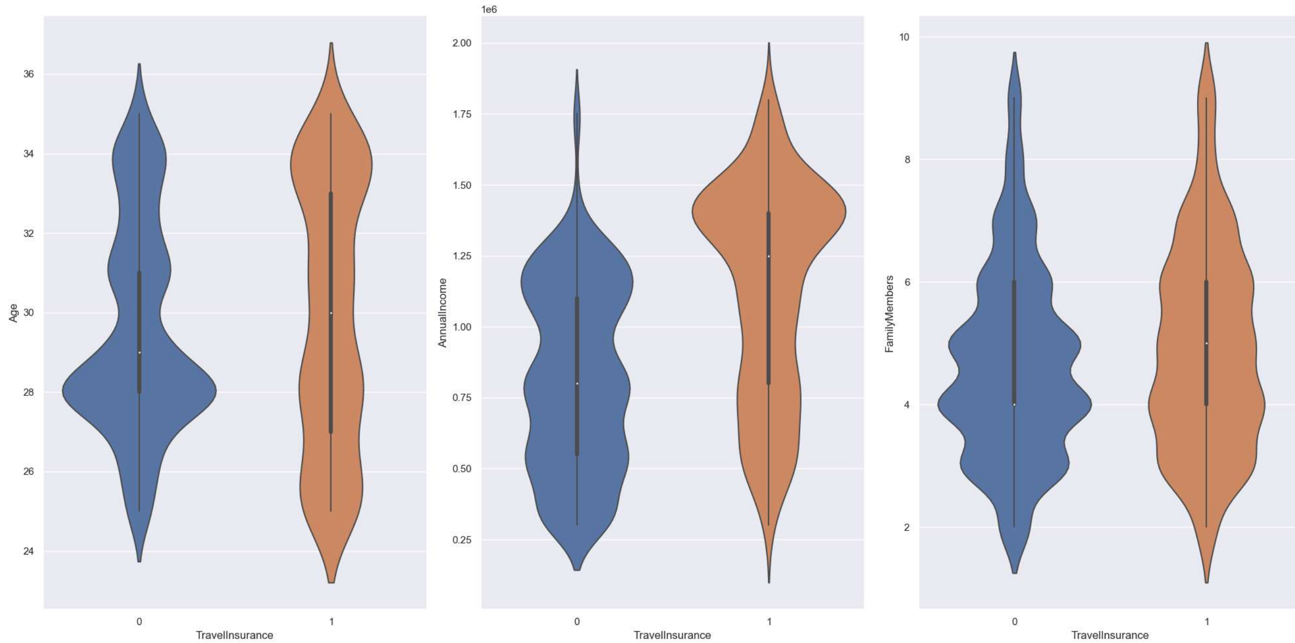
In [15]:
```
num_vars = ['Age', 'AnnualIncome', 'FamilyMembers']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



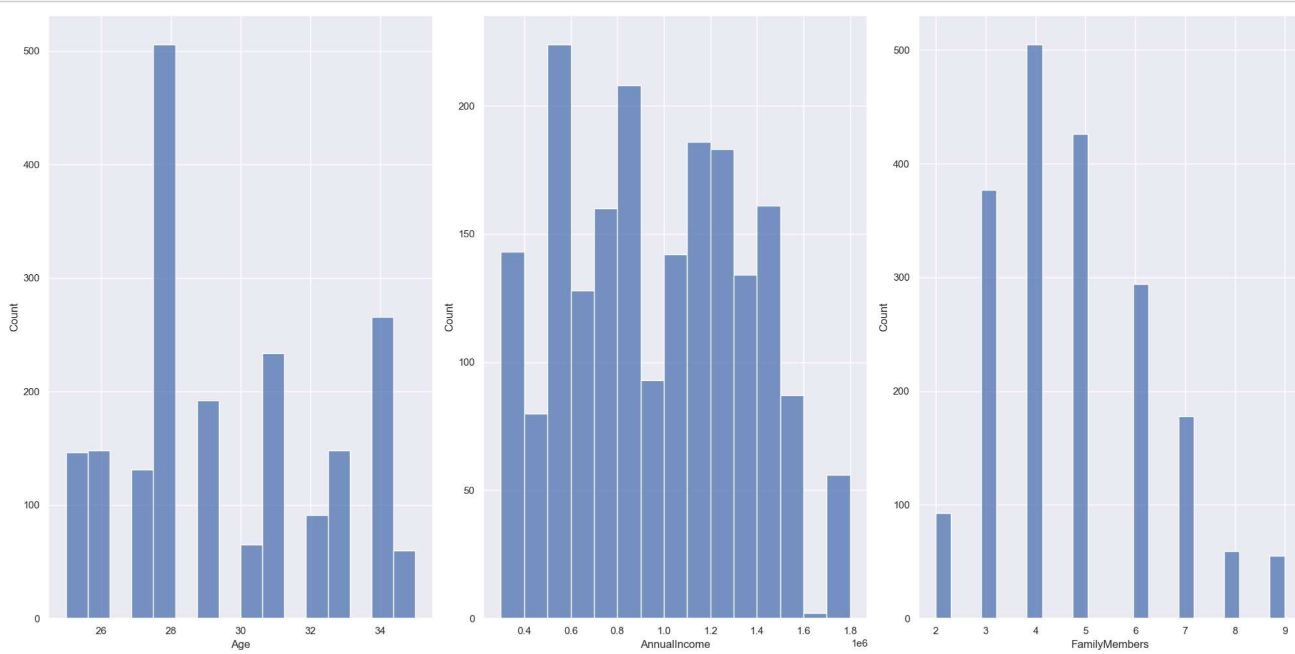
In [16]:
```
num_vars = ['Age', 'AnnualIncome', 'FamilyMembers']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [18]:
```
num_vars = ['Age', 'AnnualIncome', 'FamilyMembers']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x='TravelInsurance', y=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [19]:
```
num_vars = ['Age', 'AnnualIncome', 'FamilyMembers']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [20]:
```
df.head()
```

Out[20]:

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	31	Government Sector	Yes	400000	6	1	No	No	0
1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	0
2	34	Private Sector/Self Employed	Yes	500000	4	1	No	No	1
3	28	Private Sector/Self Employed	Yes	700000	3	1	No	No	0
4	28	Private Sector/Self Employed	Yes	700000	8	1	Yes	No	0

In [21]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[21]:
```
Series([], dtype: float64)
```

Label Encoding for Object datatypes

In [22]:
```
# 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()}")
```

```
Employment Type: ['Government Sector' 'Private Sector/Self Employed']
GraduateOrNot: ['Yes' 'No']
FrequentFlyer: ['No' 'Yes']
EverTravelledAbroad: ['No' 'Yes']
```

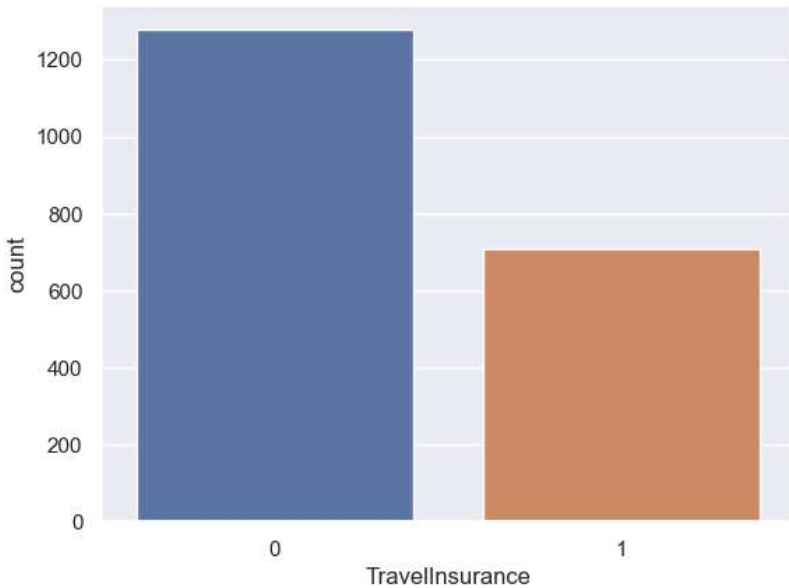
In [23]:
```
from sklearn import preprocessing

# 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()}")
```

```
Employment Type: [0 1]
GraduateOrNot: [1 0]
FrequentFlyer: [0 1]
EverTravelledAbroad: [0 1]
```

Check the Class imbalance

In [24]:
```
sns.countplot(df['TravelInsurance'])
df['TravelInsurance'].value_counts()
```

Out[24]:
```
0    1277
1     710
Name: TravelInsurance, dtype: int64
```

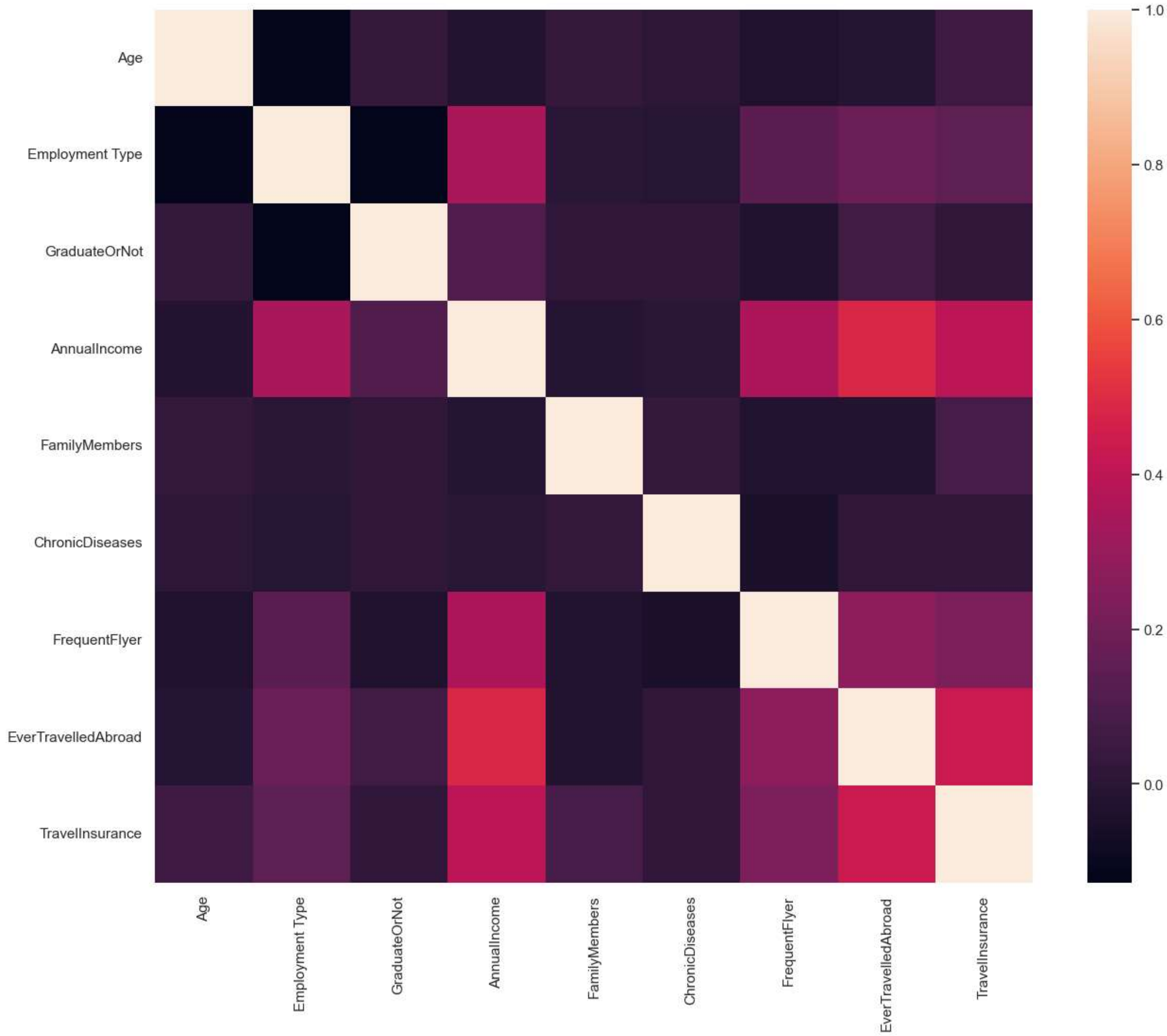


The class value is pretty imbalanced, we can balance it in our machine learning model using parameter, class_weight='balanced'

No need to remove Outlier because there is no outlier

In [26]:
```
#Correlation heatmap
plt.figure(figsize=(15,12))
sns.heatmap(df.corr(), fmt='.2g')
```

Out[26]: <AxesSubplot:>



Train Test Split

In [27]:
```
X = df.drop('TravelInsurance', axis=1)
y = df['TravelInsurance']
```

In [28]:
```
#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 Classifier

In [30]:
```
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': 0}
```

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=3, min_samples_leaf=1, min_samples_split=2, class_weight='balance
dtree.fit(X_train, y_train)
```

Out[31]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=3, random_state=0)
```

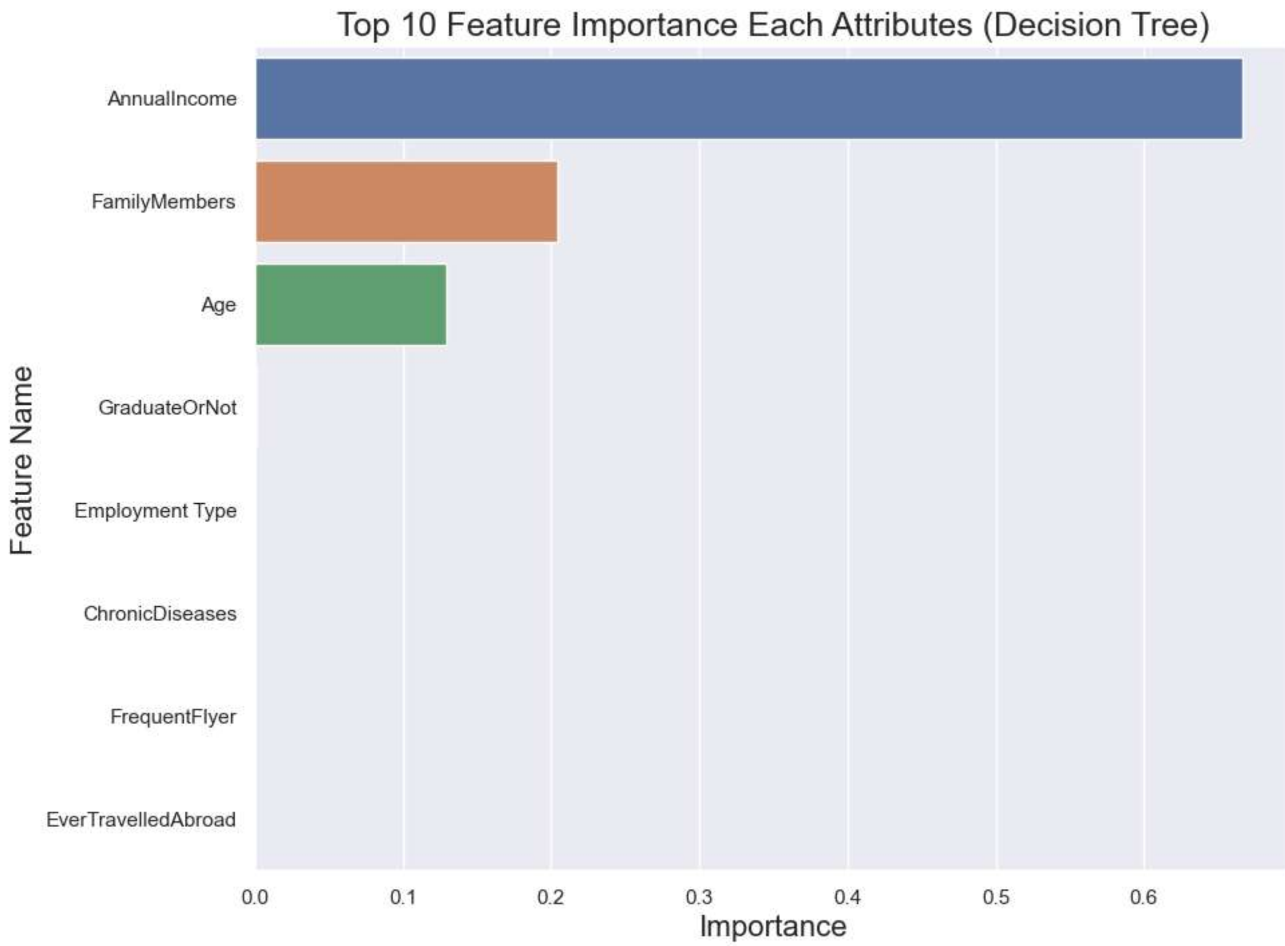
In [32]:
```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 82.66 %
```

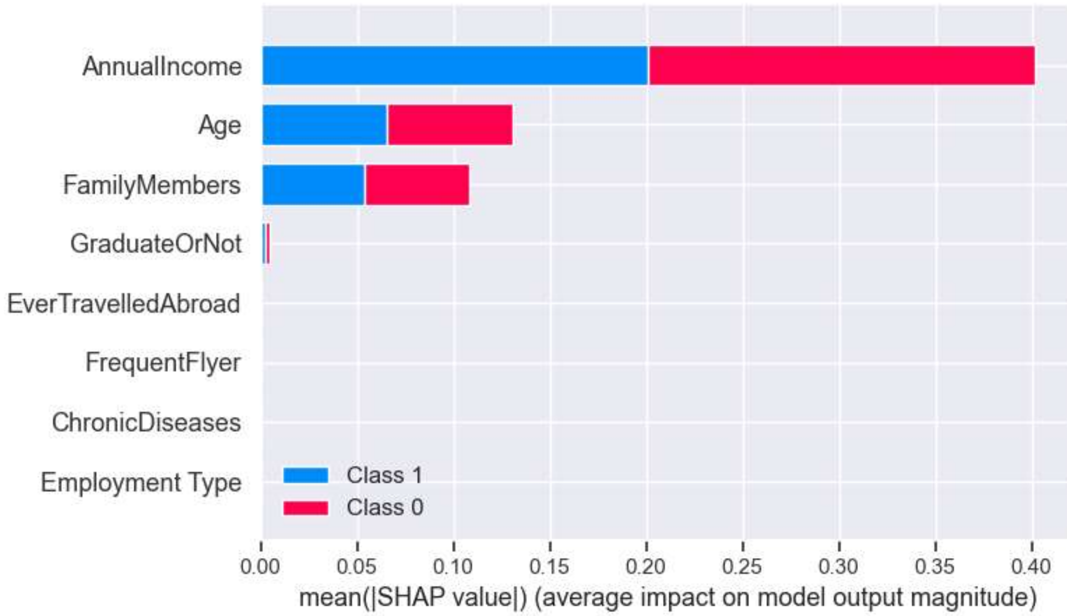
In [33]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.8266331658291457
Precision Score :  0.8266331658291457
Recall Score :  0.8266331658291457
Jaccard Score :  0.7044967880085653
Log Loss :  5.987898410108389
```

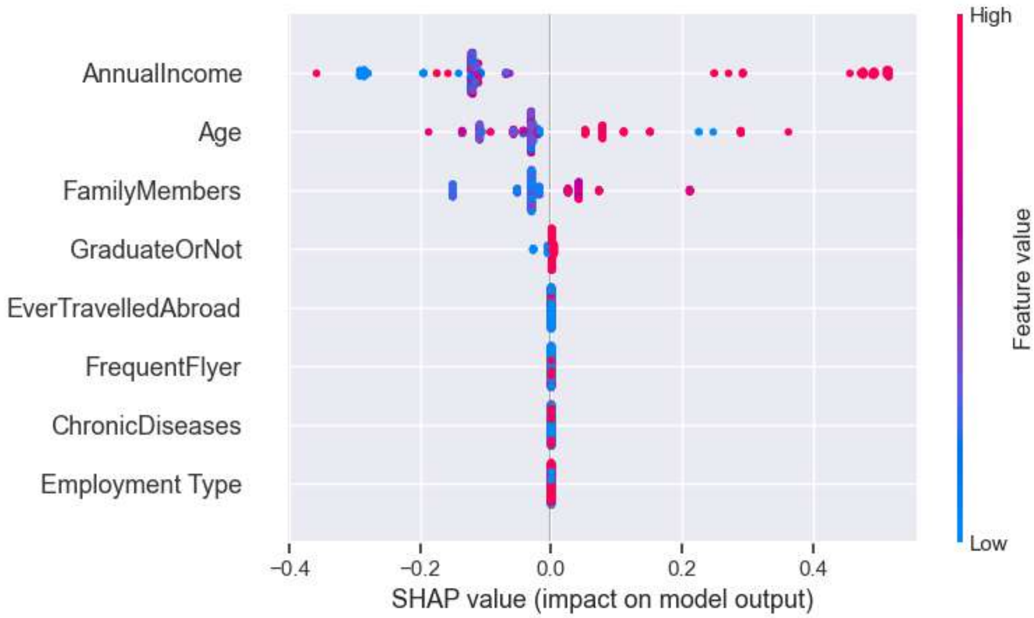
In [35]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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)
```

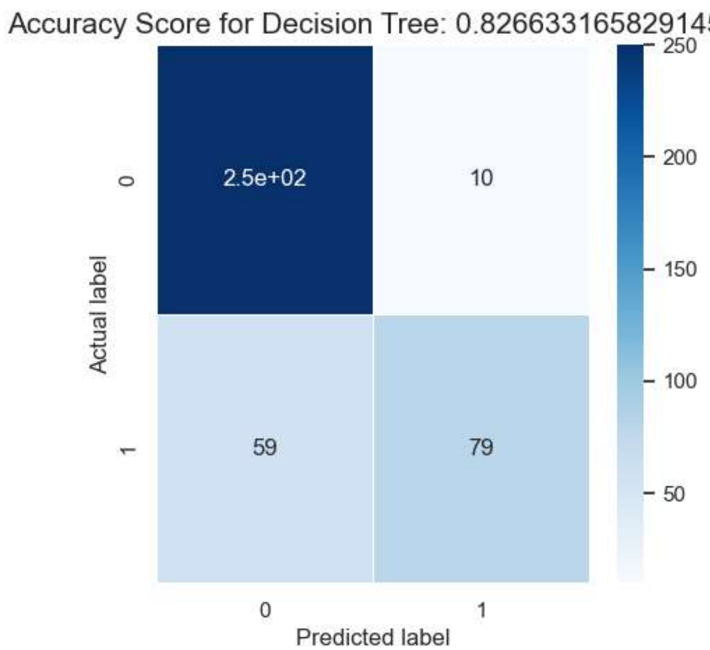


In [37]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



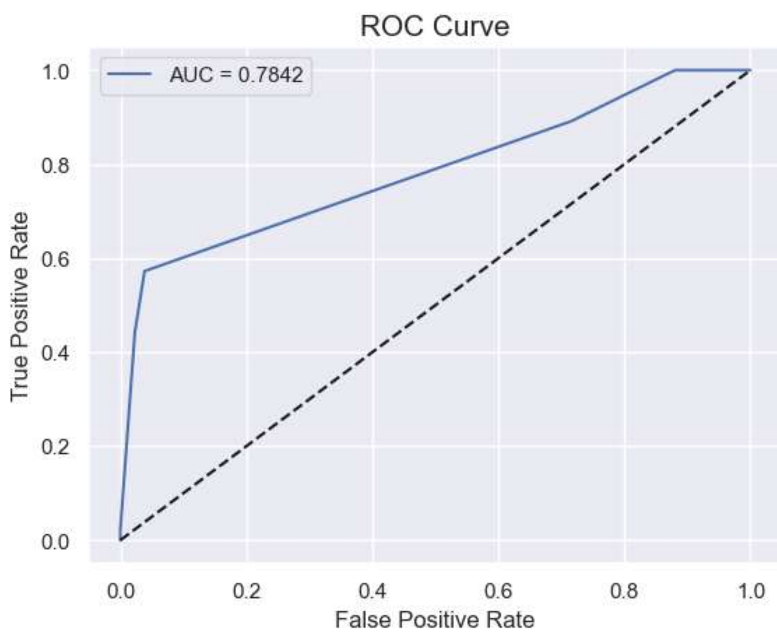
In [38]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test,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

Out[38]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8266331658291457')
```



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.DataFrame(y_pred_proba, columns=
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 0x231ac748d90>
```



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': 'log2', 'n_estimators': 200, 'random_state': 0}
```

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='log2', n_estimators=200, max_depth=5)
rfc.fit(X_train, y_train)
```

Out[41]:
```
RandomForestClassifier(max_depth=5, max_features='log2', n_estimators=200,
                       random_state=0)
```

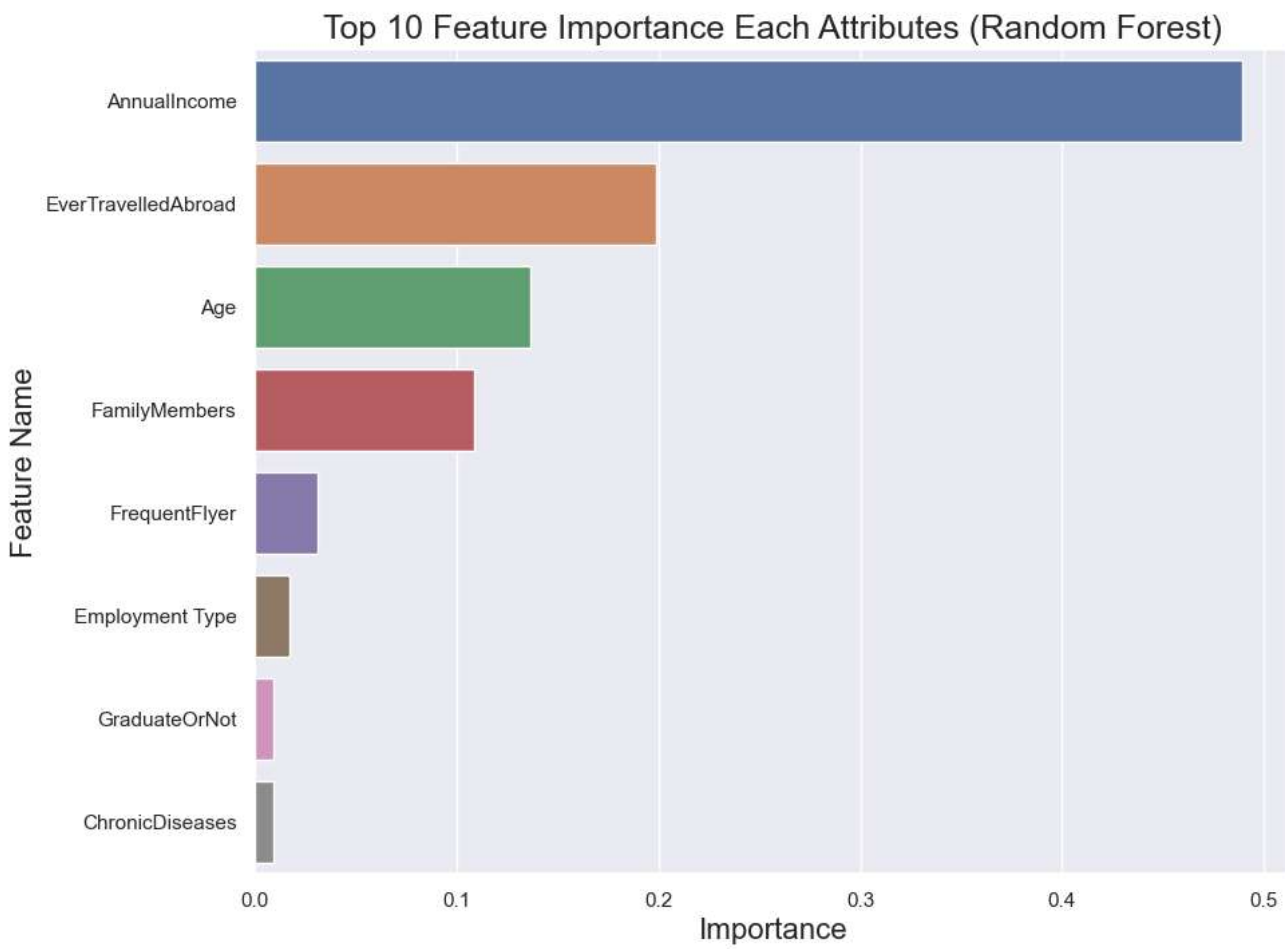
In [42]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 82.91 %
```

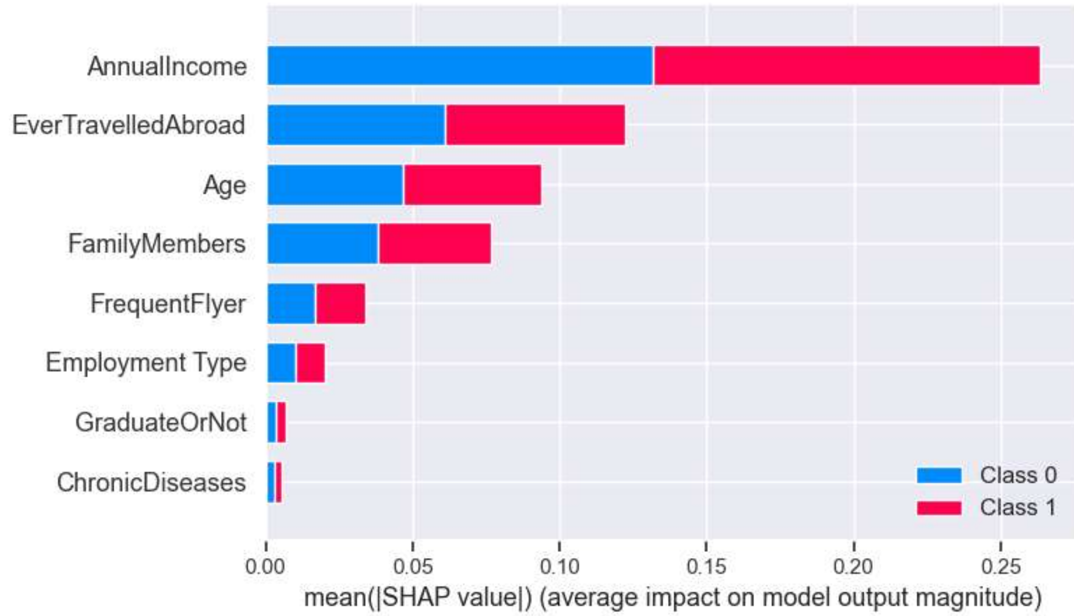
In [43]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.8291457286432161
Precision Score :  0.8291457286432161
Recall Score :  0.8291457286432161
Jaccard Score :  0.7081545064377682
Log Loss :  5.901117564895047
```

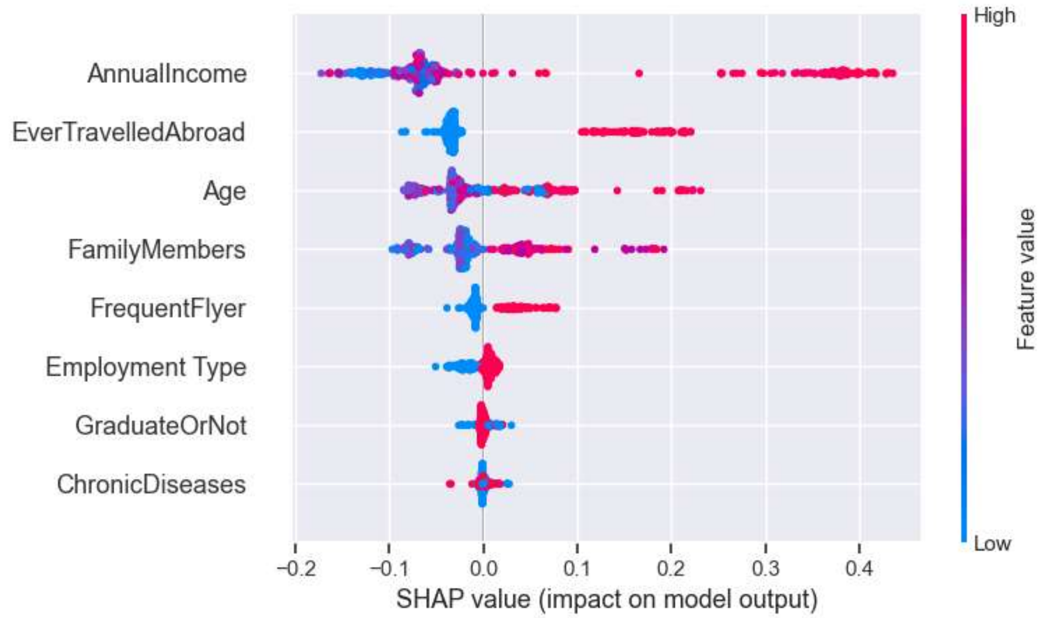
In [44]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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)
```

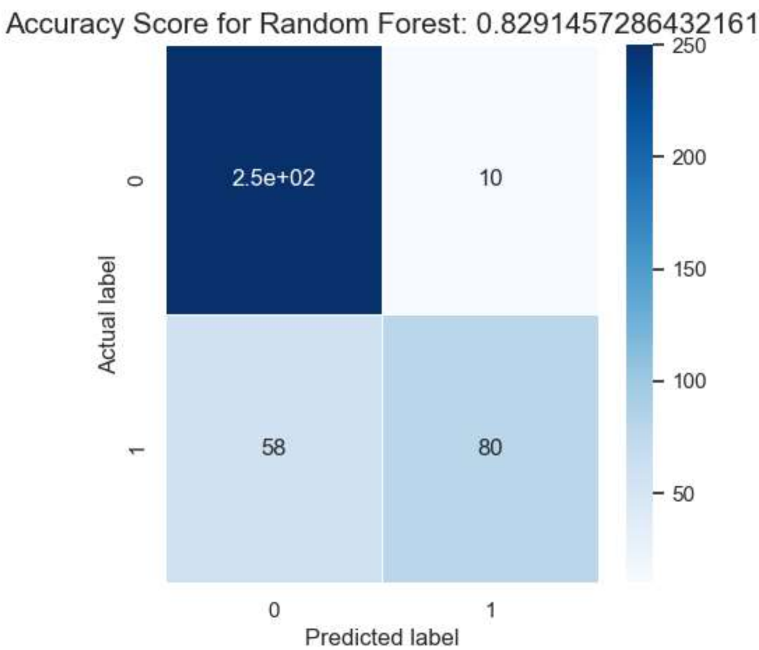


In [46]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



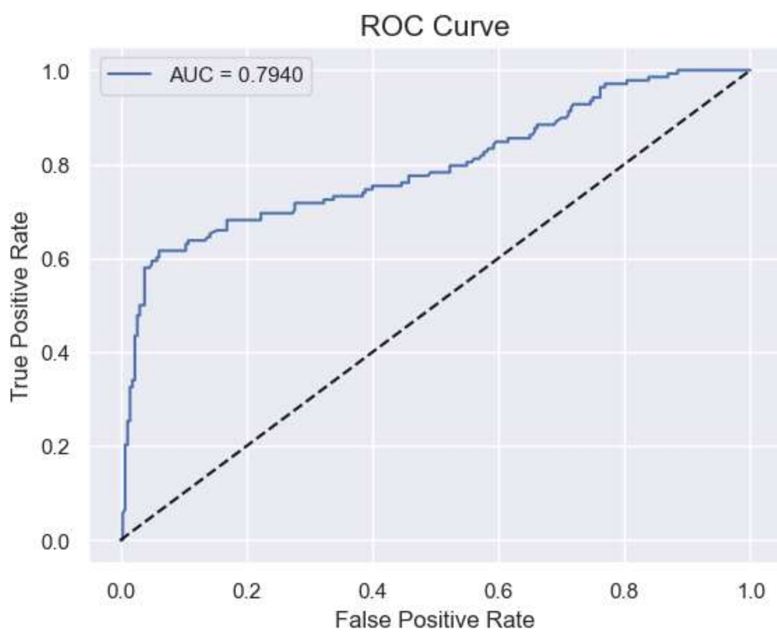
In [47]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test,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

Out[47]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.8291457286432161')
```



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.DataFrame(y_pred_proba, columns=
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 0x231b413b4c0>



XGBoost

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

# Create an instance of the XGBoost classifier
xgb = XGBClassifier()

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 10],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'random_state': [0, 42]
}

# 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_)
```

```
{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100, 'random_state': 0, 'subsample': 0.
8}
```

In [50]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.01, subsample=0.8, colsample_bytree=0.8)
xgb.fit(X_train, y_train)
```

Out[50]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, 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,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

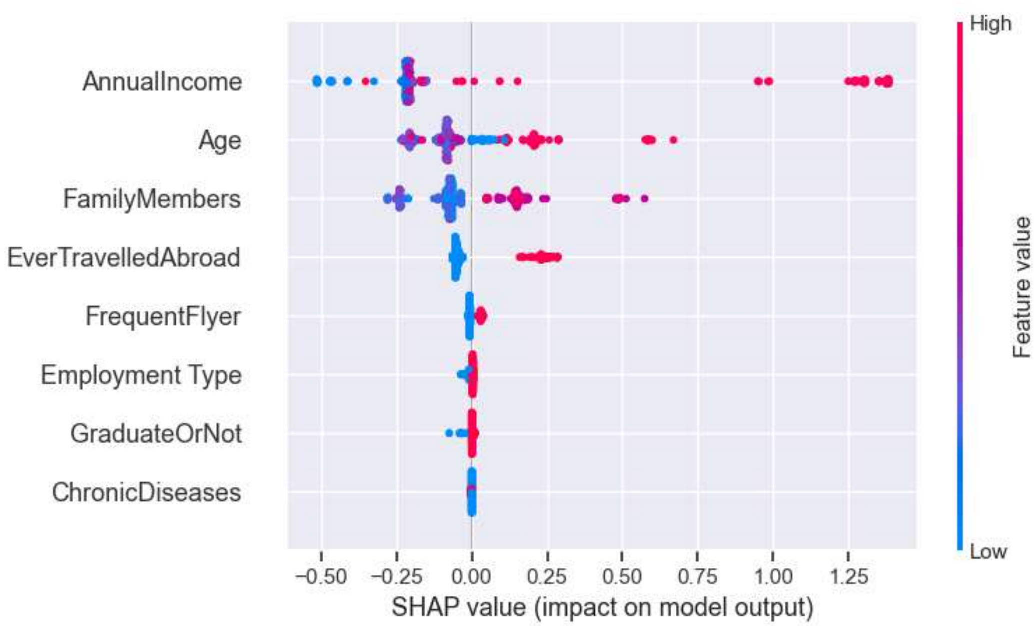
In [51]:
```
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 82.16 %
```

In [52]:
```
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)))
```

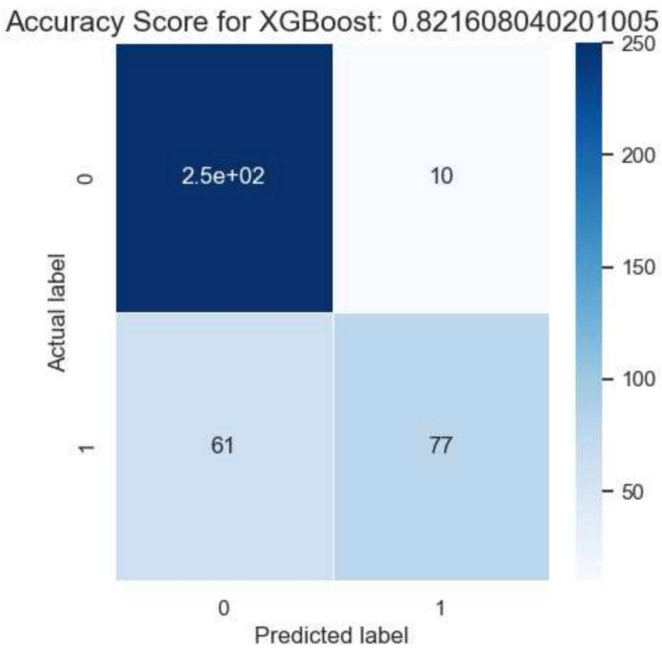
```
F-1 Score :  0.821608040201005
Precision Score :  0.821608040201005
Recall Score :  0.821608040201005
Jaccard Score :  0.697228144989339
Log Loss :  6.161460100535076
```

In [54]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [56]:
```
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[56]:
```
Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.821608040201005')
```



In [57]:
```
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.DataFrame(y_pred_proba, columns=
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[57]:
```
<matplotlib.legend.Legend at 0x231b3c8b4f0>
```