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('income_evaluation.csv')
df.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hour pe we
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	

Data Preprocessing Part 1

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

Out[3]:
```
workclass          9
education         16
marital-status     7
occupation        15
relationship       6
race               5
sex                2
native-country    42
income             2
dtype: int64
```

Segment the Education column into smaller number

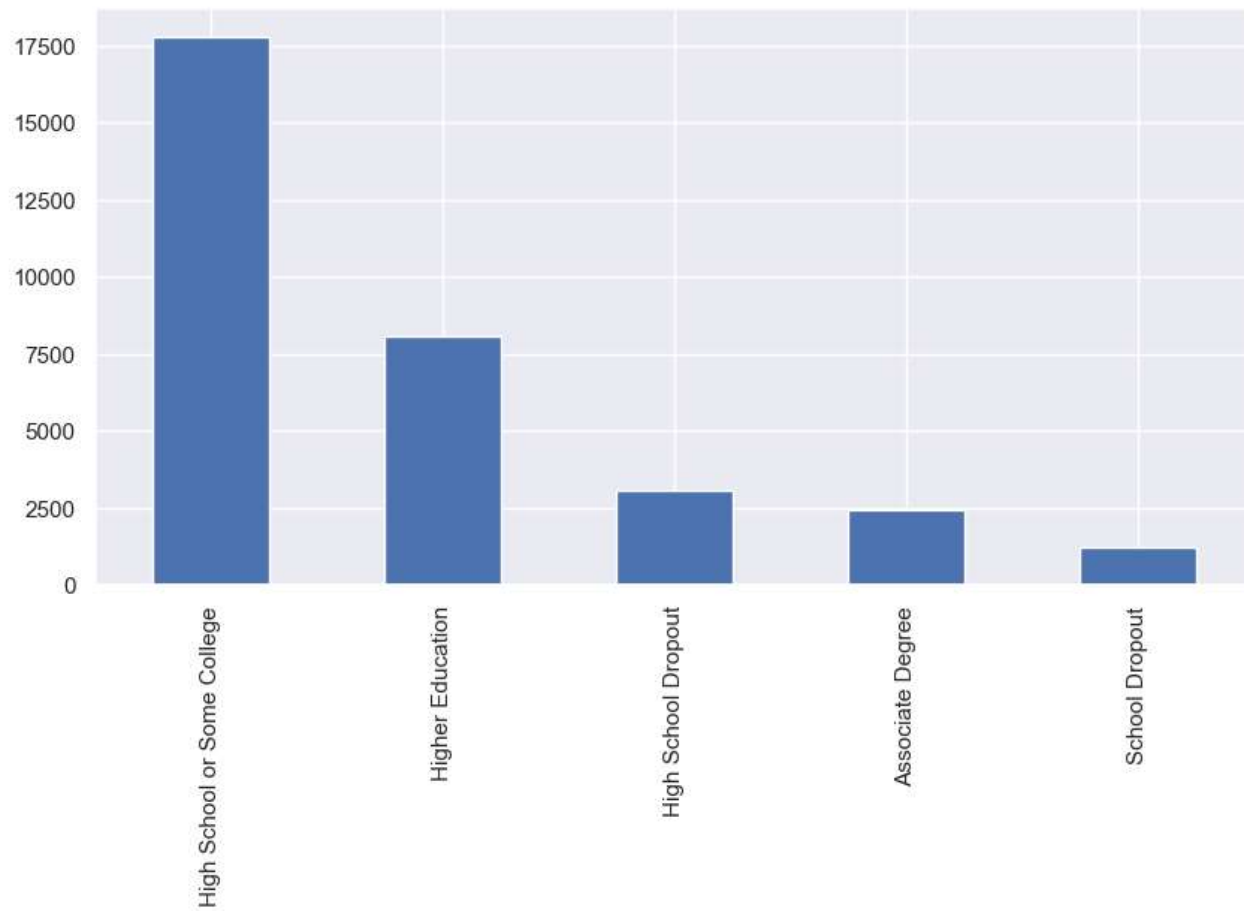
In [9]:
```
df[' education'].unique()
```

Out[9]:
```
array([' Bachelors', ' HS-grad', ' 11th', ' Masters', ' 9th',
       ' Some-college', ' Assoc-acdm', ' Assoc-voc', ' 7th-8th',
       ' Doctorate', ' Prof-school', ' 5th-6th', ' 10th', ' 1st-4th',
       ' Preschool', ' 12th'], dtype=object)
```

In [10]:
```
def segment_education(education):
    if education.strip() == 'Bachelors' or education.strip() == 'Masters' or education.strip() == '
        return 'Higher Education'
    elif education.strip() == 'HS-grad' or education.strip() == 'Some-college':
        return 'High School or Some College'
    elif '9th' in education or '10th' in education or '11th' in education or '12th' in education:
        return 'High School Dropout'
    elif 'Assoc' in education:
        return 'Associate Degree'
    elif '7th-8th' in education or '5th-6th' in education or '1st-4th' in education or 'Preschool' 
        return 'School Dropout'
    else:
        return 'Other'

df[' education'] = df[' education'].apply(segment_education)
```

In [11]:
```
plt.figure(figsize=(10,5))
df[' education'].value_counts().plot(kind='bar')
```

Out[11]: <AxesSubplot:>



Segment the Occupation column into smaller number

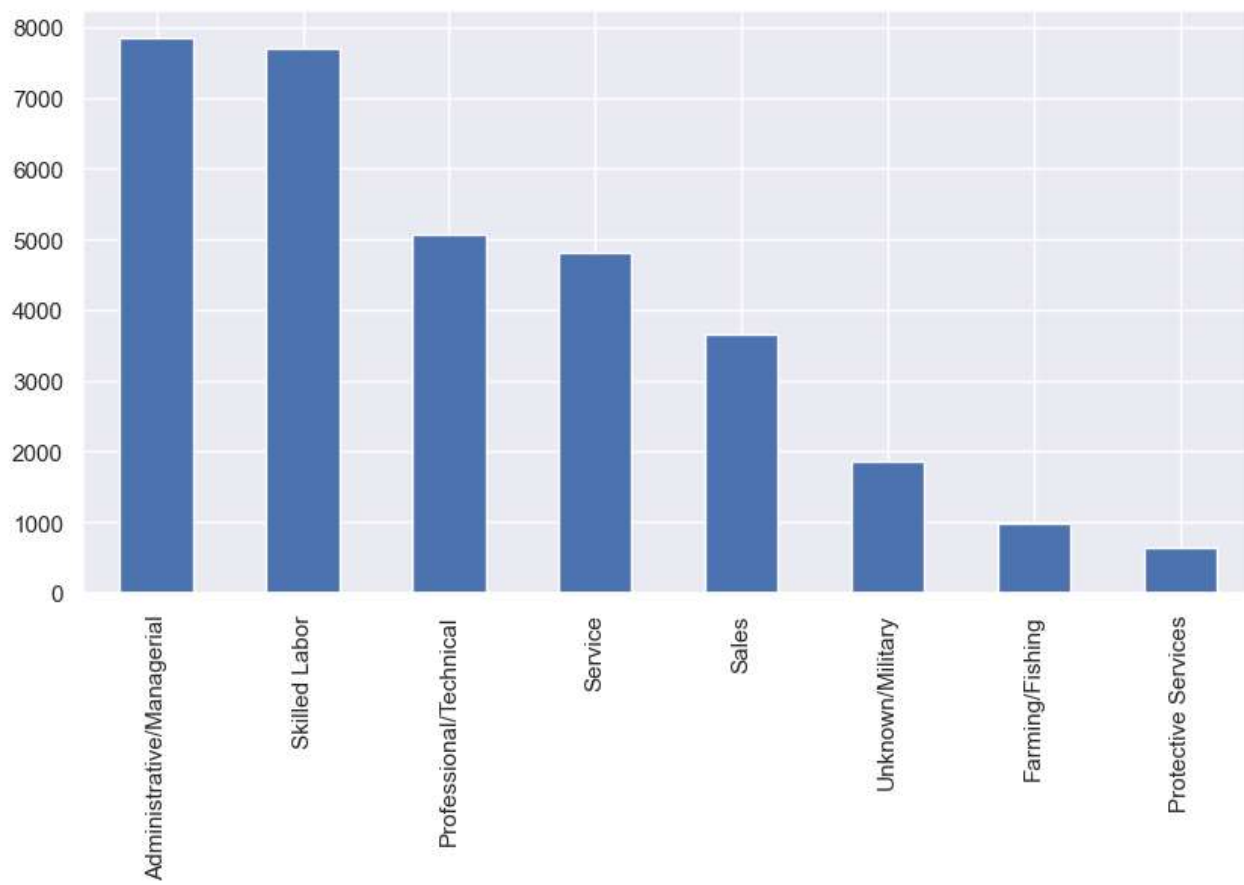
In [12]:
```
df[' occupation'].unique()
```

Out[12]:
```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

In [13]:
```
def segment_occupation(occupation):
    if occupation.strip() in ['Adm-clerical', 'Exec-managerial']:
        return 'Administrative/Managerial'
    elif occupation.strip() in ['Handlers-cleaners', 'Other-service', 'Priv-house-serv']:
        return 'Service'
    elif occupation.strip() in ['Prof-specialty', 'Tech-support']:
        return 'Professional/Technical'
    elif occupation.strip() in ['Sales']:
        return 'Sales'
    elif occupation.strip() in ['Craft-repair', 'Machine-op-inspct', 'Transport-moving']:
        return 'Skilled Labor'
    elif occupation.strip() in ['Farming-fishing']:
        return 'Farming/Fishing'
    elif occupation.strip() in ['?', 'Armed-Forces']:
        return 'Unknown/Military'
    elif occupation.strip() in ['Protective-serv']:
        return 'Protective Services'
    else:
        return 'Other'

df[' occupation'] = df[' occupation'].apply(segment_occupation)
```

In [14]:
```
plt.figure(figsize=(10,5))
df[' occupation'].value_counts().plot(kind='bar')
```

Out[14]: <AxesSubplot:>



Segment the Native Country column into smaller number

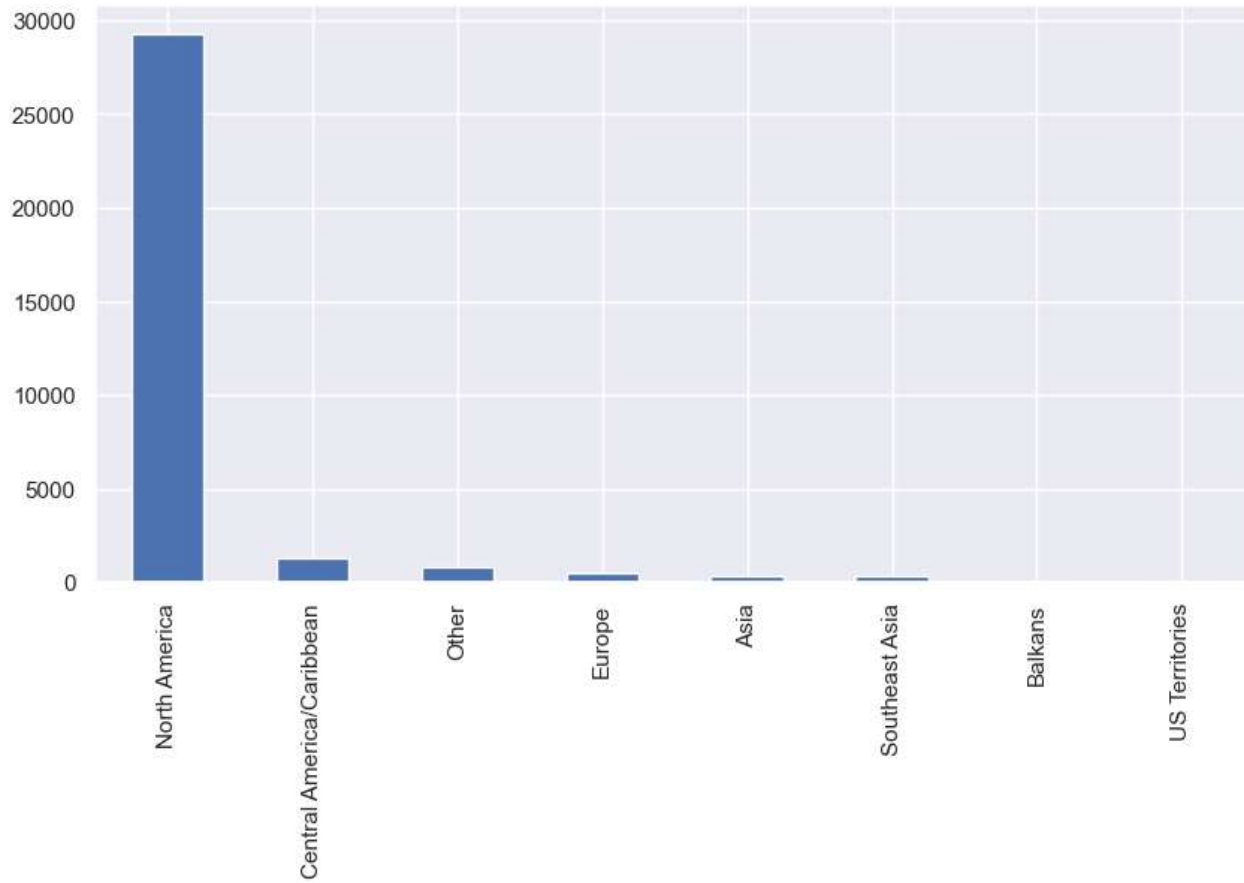
In [15]:
```
df[' native-country'].unique()
```

Out[15]:
```
array([' United-States', ' Cuba', ' Jamaica', ' India', ' ?', ' Mexico',
       ' South', ' Puerto-Rico', ' Honduras', ' England', ' Canada',
       ' Germany', ' Iran', ' Philippines', ' Italy', ' Poland',
       ' Columbia', ' Cambodia', ' Thailand', ' Ecuador', ' Laos',
       ' Taiwan', ' Haiti', ' Portugal', ' Dominican-Republic',
       ' El-Salvador', ' France', ' Guatemala', ' China', ' Japan',
       ' Yugoslavia', ' Peru', ' Outlying-US(Guam-USVI-etc)', ' Scotland',
       ' Trinadad&Tobago', ' Greece', ' Nicaragua', ' Vietnam', ' Hong',
       ' Ireland', ' Hungary', ' Holand-Netherlands'], dtype=object)
```

In [17]:
```
def segment_native_country(country):
    if country.strip() in ['United-States', 'Canada']:
        return 'North America'
    elif country.strip() in ['Mexico', 'Cuba', 'Jamaica', 'Dominican-Republic', 'Puerto-Rico', 'Hon
        return 'Central America/Caribbean'
    elif country.strip() in ['Germany', 'England', 'France', 'Italy', 'Poland', 'Scotland', 'Greece
        return 'Europe'
    elif country.strip() in ['India', 'China', 'Japan', 'Taiwan', 'Iran']:
        return 'Asia'
    elif country.strip() in ['Cambodia', 'Thailand', 'Laos', 'Philippines', 'Vietnam']:
        return 'Southeast Asia'
    elif country.strip() in ['Yugoslavia', 'Greece']:
        return 'Balkans'
    elif country.strip() in ['Outlying-US(Guam-USVI-etc)']:
        return 'US Territories'
    else:
        return 'Other'

df[' native-country'] = df[' native-country'].apply(segment_native_country)
```

In [18]:
```
plt.figure(figsize=(10,5))
df[' native-country'].value_counts().plot(kind='bar')
```

Out[18]:
```
<AxesSubplot:>
```

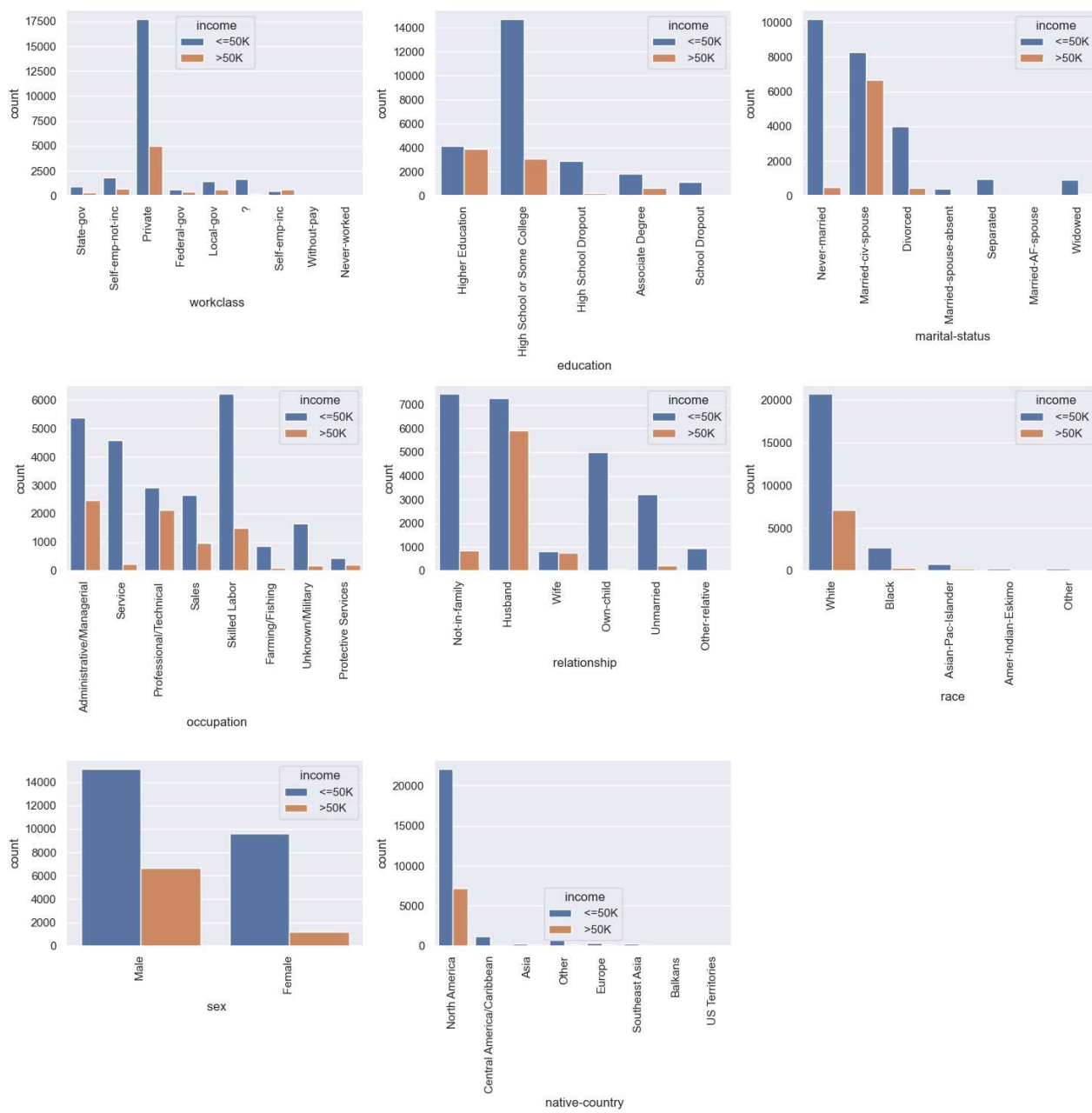


In [19]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

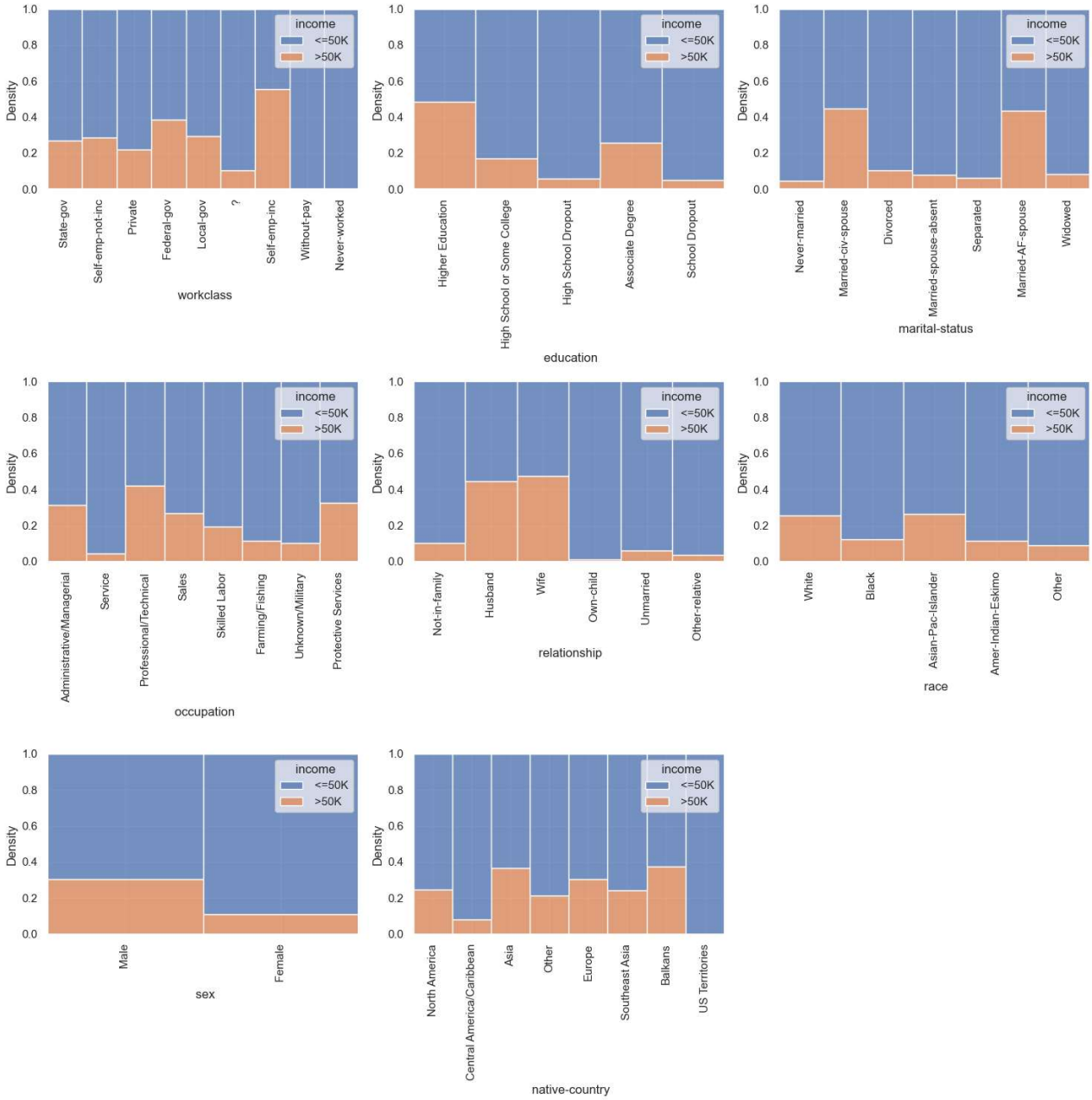
Out[19]:
```
workclass         9
education         5
marital-status    7
occupation        8
relationship      6
race              5
sex               2
native-country    8
income            2
dtype: int64
```

Exploratory Data Analysis

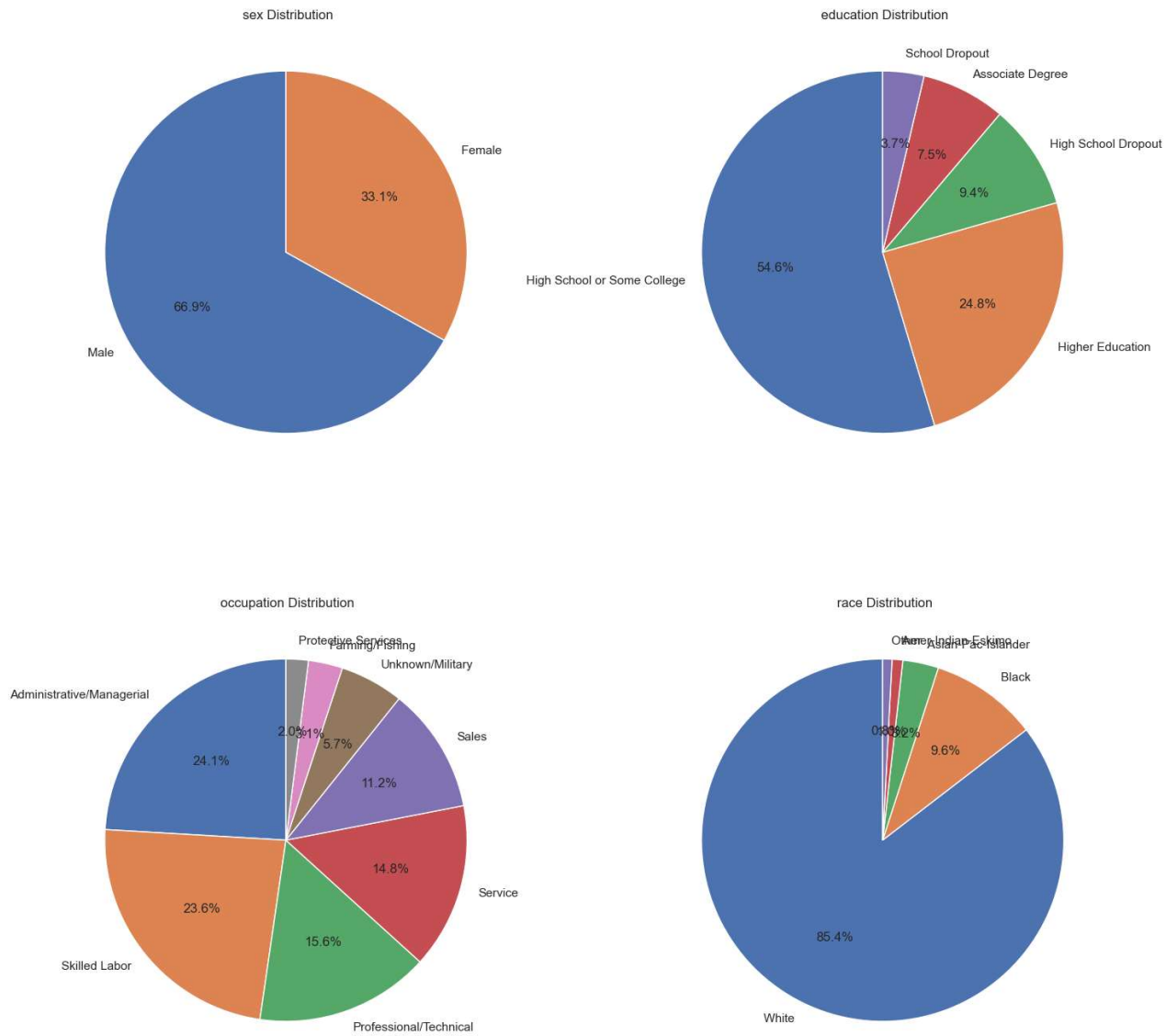
In [23]:
```
# list of categorical variables to plot
cat_vars = [' workclass', ' education', ' marital-status', ' occupation', ' relationship',
            ' race', ' sex', ' native-country']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, 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=' income', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# remove the ninth subplot
fig.delaxes(axs[8])

# show plot
plt.show()
```



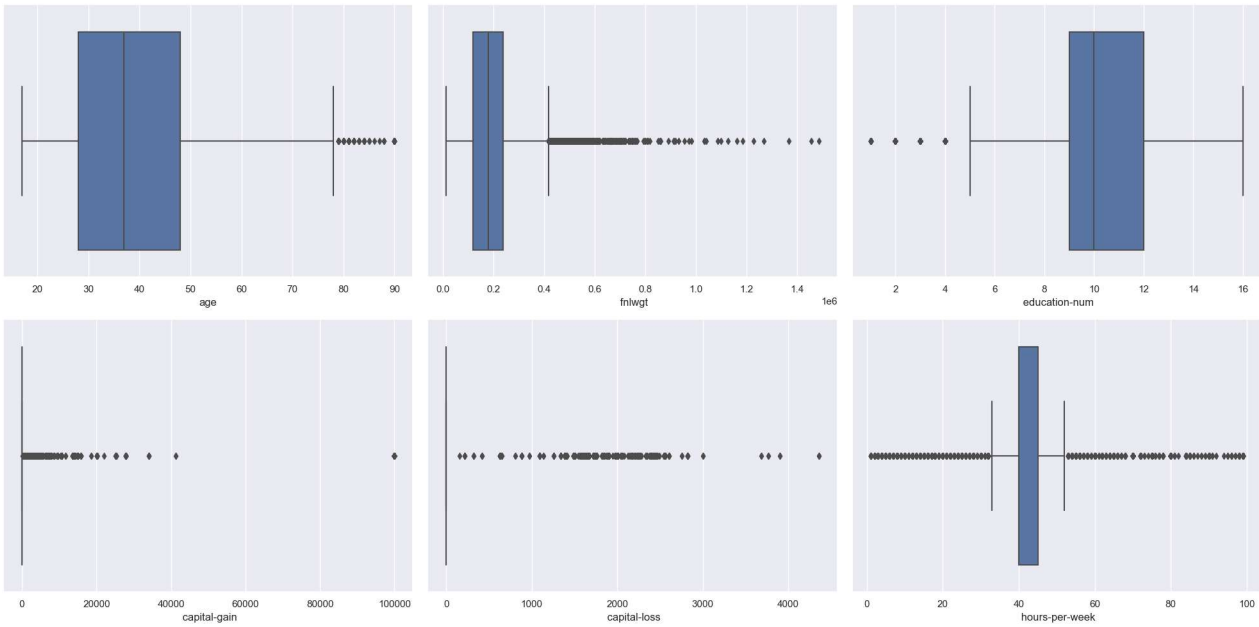
In [25]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = [' workclass', ' education', ' marital-status', ' occupation', ' relationship',
            ' race', ' sex', ' native-country']

# create figure with subplots
fig, axs = plt.subplots(nrows=3, 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=' income', data=df, ax=axs[i], multiple="fill", kde=False, element="bar
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# remove the ninth subplot
fig.delaxes(axs[8])

# show plot
plt.show()
```


income
<=50K
>50K
Density
workclass
State-gov
Self-emp-not-inc
Private
Federal-gov
Local-gov
?
Self-emp-inc
Without-pay
Never-worked
education
Higher Education
High School or Some College
High School Dropout
Associate Degree
School Dropout
marital-status
Never-married
Married-civ-spouse
Divorced
Married-spouse-absent
Separated
Married-AF-spouse
Widowed
occupation
Administrative/Managerial
Service
Professional/Technical
Sales
Skilled Labor
Farming/Fishing
Unknown/Military
Protective Services
relationship
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
race
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other
sex
Male
Female
native-country
North America
Central America/Caribbean
Asia
Other
Europe
Southeast Asia
Balkans
US Territories

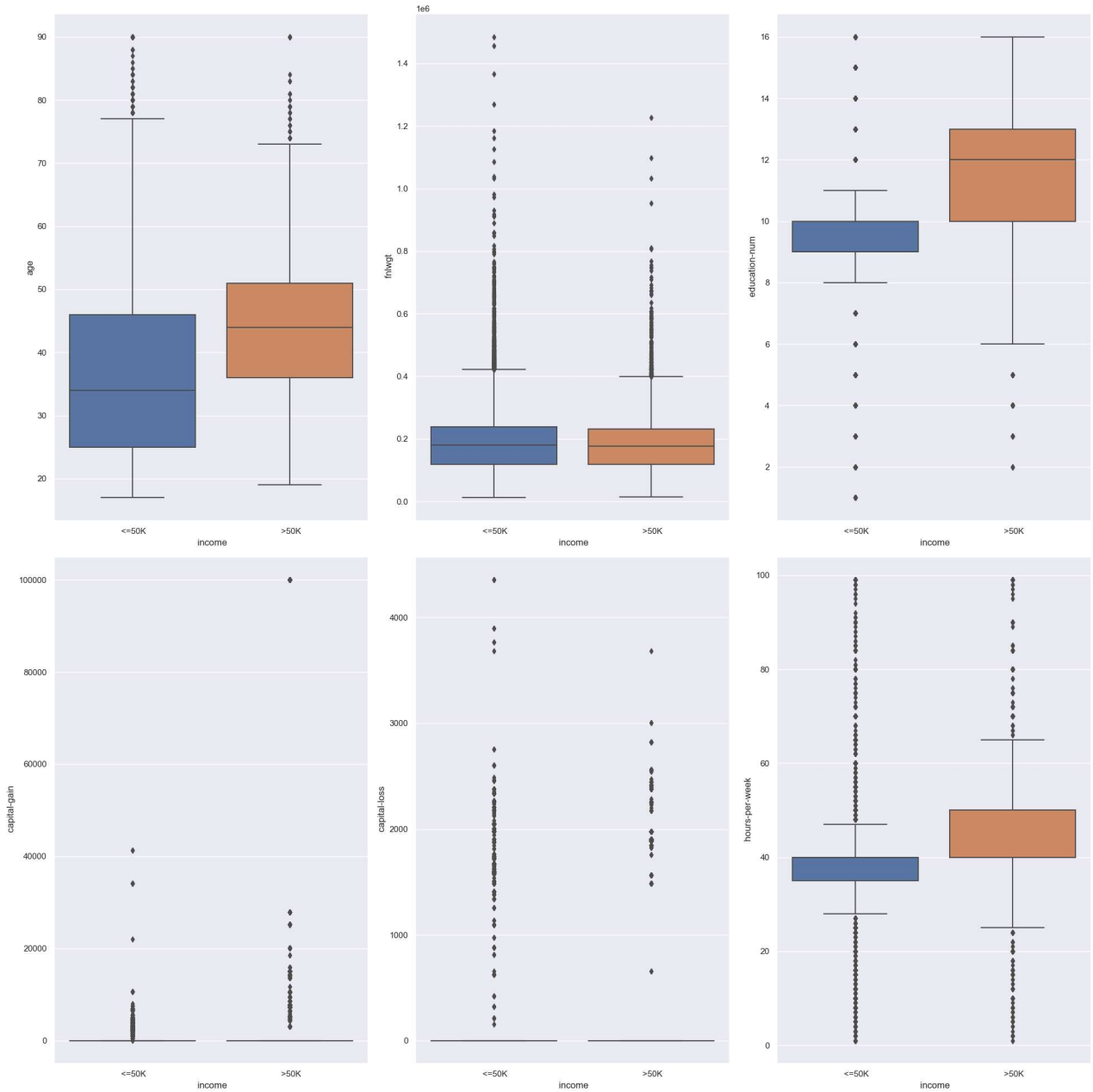
In [26]:
```
cat_vars = [' sex', ' education', ' occupation', ' race']

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

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



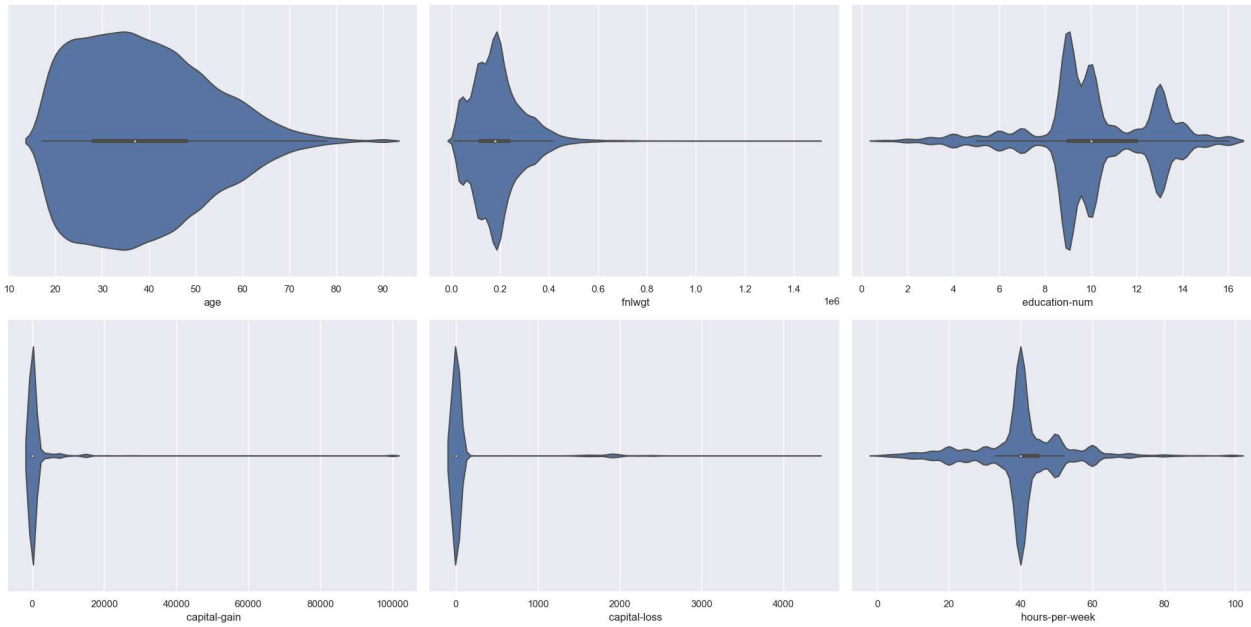
In [29]:
```
num_vars = ['age', ' fnlwgt', ' education-num', ' capital-gain',
            ' capital-loss', ' hours-per-week']

fig, axs = plt.subplots(nrows=2, 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()
```



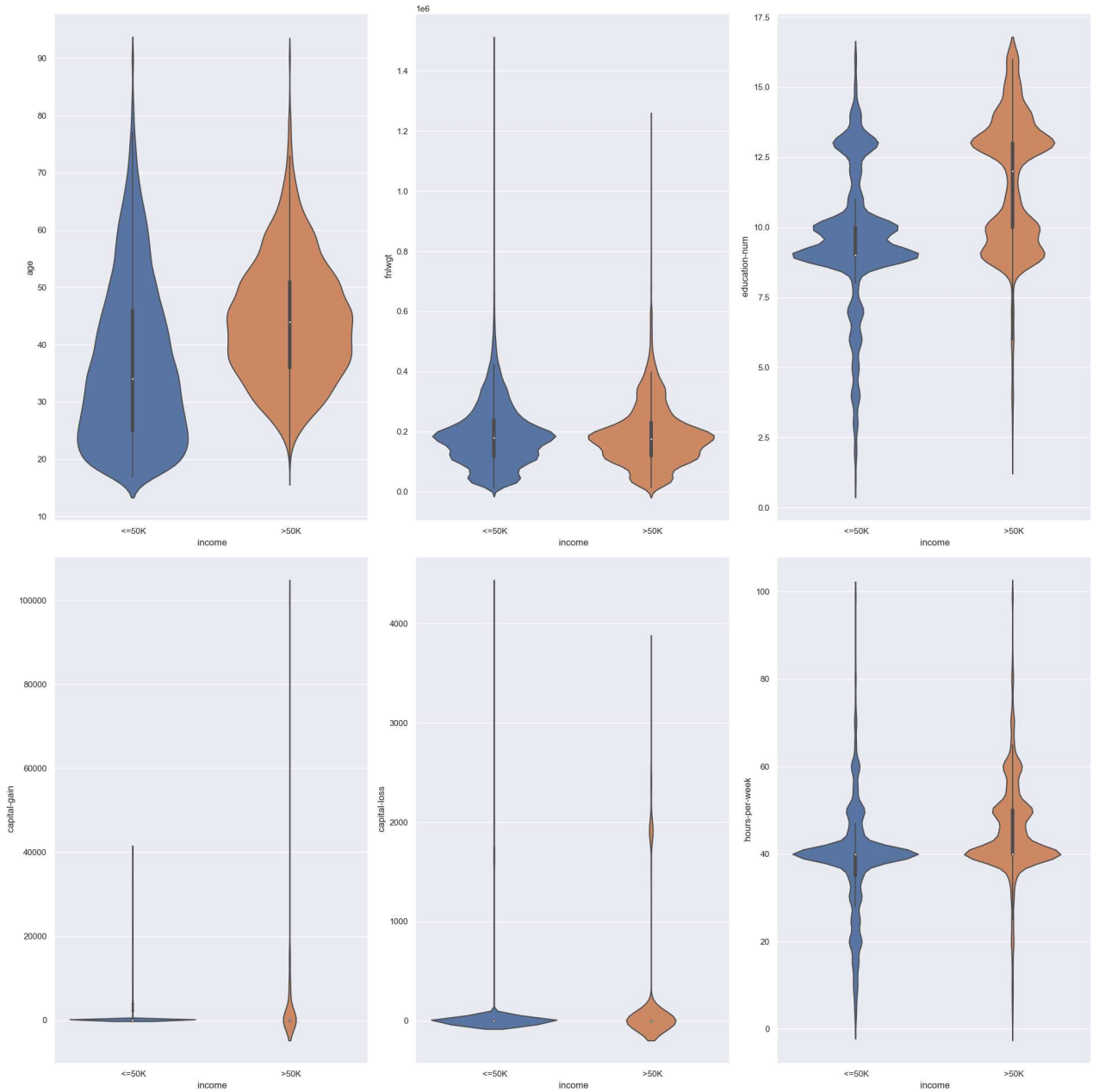
In [31]:
```
num_vars = ['age', ' fnlwgt', ' education-num', ' capital-gain',
            ' capital-loss', ' hours-per-week']

fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 20))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



In [32]:
```
num_vars = ['age', ' fnlwgt', ' education-num', ' capital-gain',
            ' capital-loss', ' hours-per-week']

fig, axs = plt.subplots(nrows=2, 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 [34]:
```
num_vars = ['age', ' fnlwgt', ' education-num', ' capital-gain',
            ' capital-loss', ' hours-per-week']

fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 20))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [35]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[35]:
```
Series([], dtype: float64)
```

Label encoding for object datatypes

In [36]:
```
# 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()}")
```

```
workclass: [' State-gov' ' Self-emp-not-inc' ' Private' ' Federal-gov' ' Local-gov'
' ?' ' Self-emp-inc' ' Without-pay' ' Never-worked']
education: ['Higher Education' 'High School or Some College' 'High School Dropout'
'Associate Degree' 'School Dropout']
marital-status: [' Never-married' ' Married-civ-spouse' ' Divorced'
' Married-spouse-absent' ' Separated' ' Married-AF-spouse' ' Widowed']
occupation: ['Administrative/Managerial' 'Service' 'Professional/Technical' 'Sales'
'Skilled Labor' 'Farming/Fishing' 'Unknown/Military'
'Protective Services']
relationship: [' Not-in-family' ' Husband' ' Wife' ' Own-child' ' Unmarried'
' Other-relative']
race: [' White' ' Black' ' Asian-Pac-Islander' ' Amer-Indian-Eskimo' ' Other']
sex: [' Male' ' Female']
native-country: ['North America' 'Central America/Caribbean' 'Asia' 'Other' 'Europe'
'Southeast Asia' 'Balkans' 'US Territories']
income: [' <=50K' ' >50K']
```

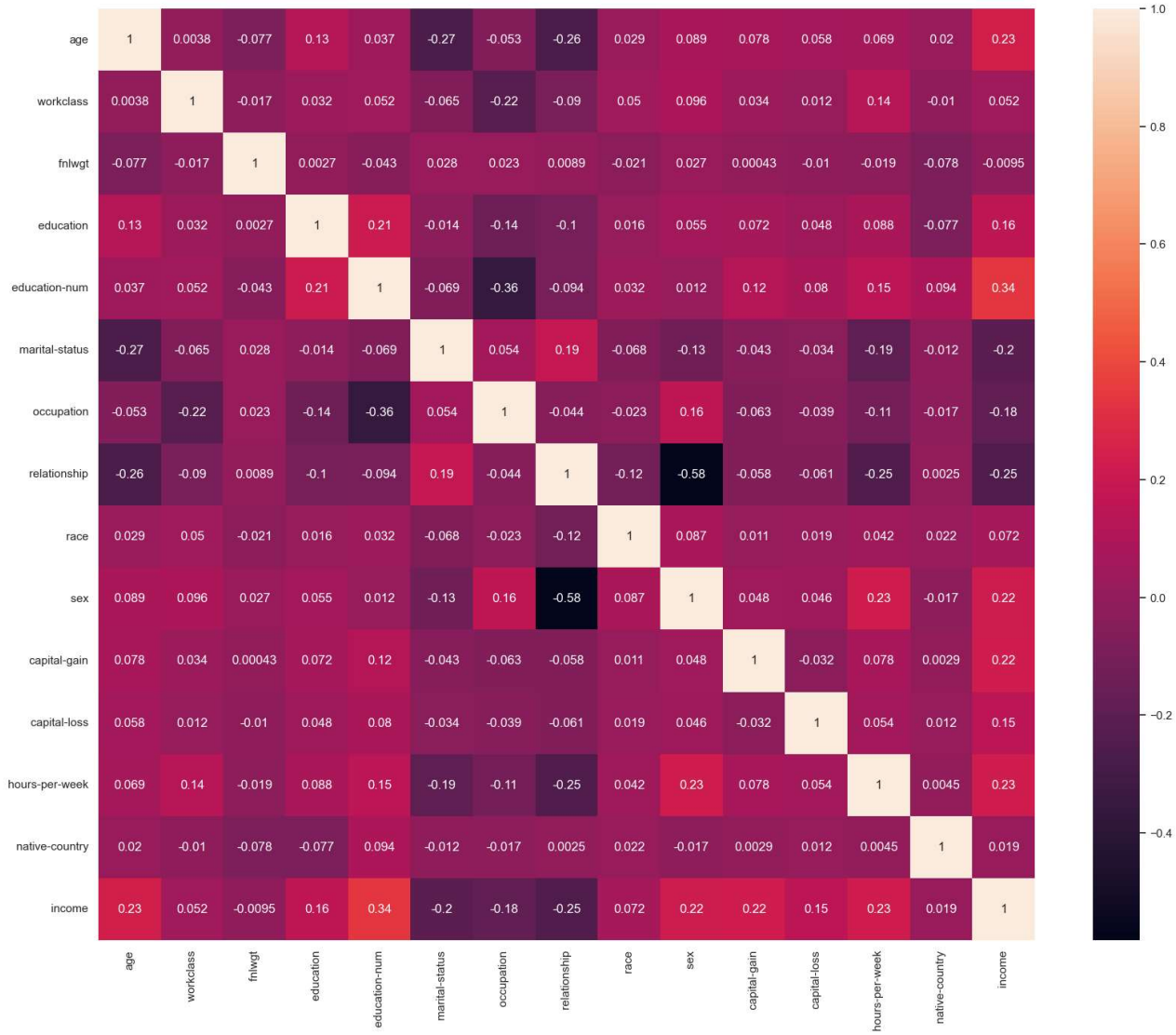
In [37]:
```
from sklearn import preprocessing

# 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()}")
```

```
workclass: [7 6 4 1 2 0 5 8 3]
education: [3 2 1 0 4]
marital-status: [4 2 0 3 5 1 6]
occupation: [0 5 2 4 6 1 7 3]
relationship: [1 0 5 3 4 2]
race: [4 2 1 0 3]
sex: [1 0]
native-country: [4 2 0 5 3 6 1 7]
income: [0 1]
```

Correlation Heatmap

In [38]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[38]: <AxesSubplot:>



Train Test Split

In [39]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop(' income', axis=1)
y = df[' income']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Remove Outlier in Train Data using Z-Score

In [41]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = [' fnlwgt', ' capital-gain', ' capital-loss', ' hours-per-week']

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

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

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

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

Decision Tree

In [42]:
```
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': 2, 'min_samples_split': 2, 'random_state': 42}
```

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=42, max_depth=8, min_samples_leaf=2, min_samples_split=
dtree.fit(X_train, y_train)
```

Out[43]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=8, min_samples_leaf=2,
                       random_state=42)
```

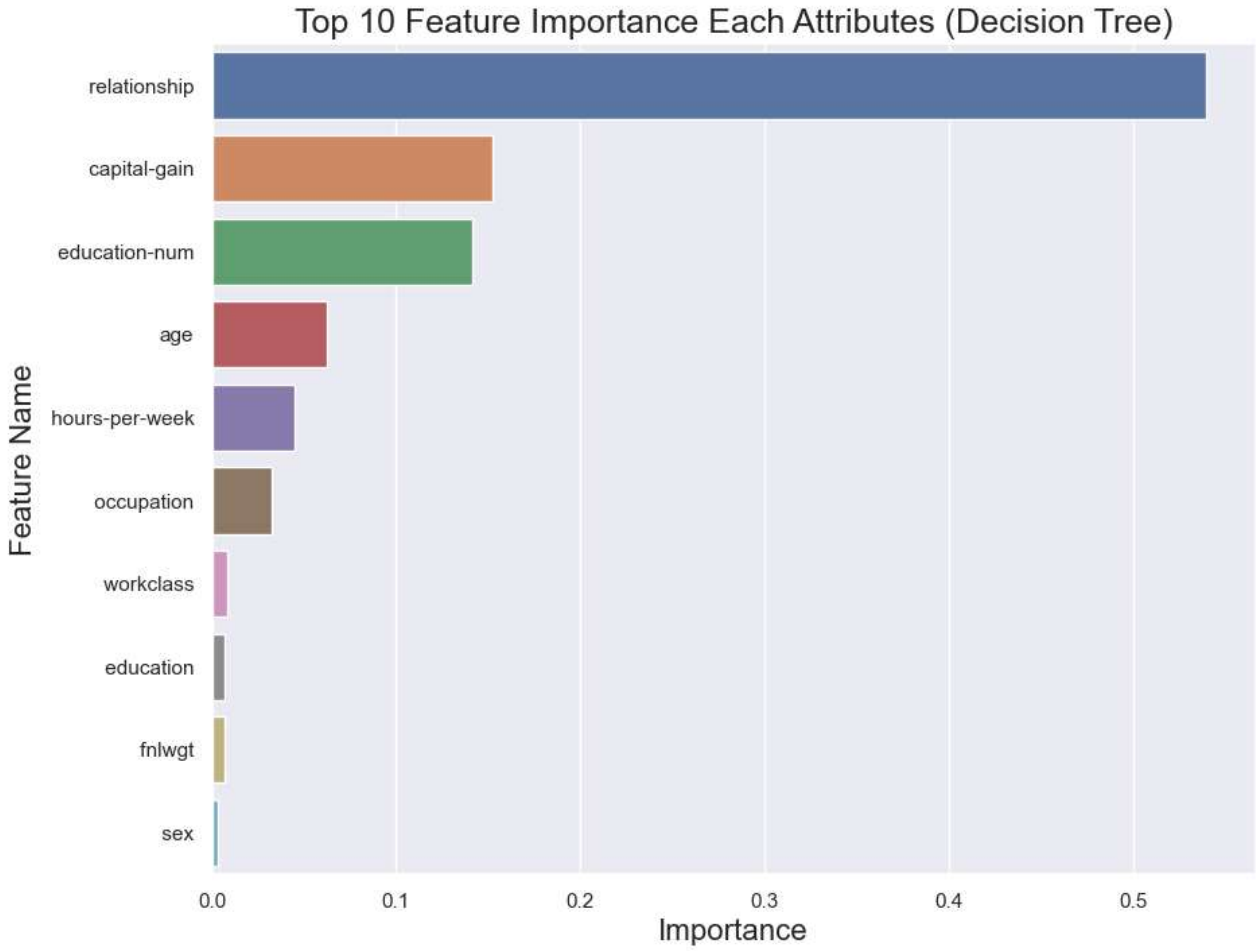
In [45]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 79.13 %
```

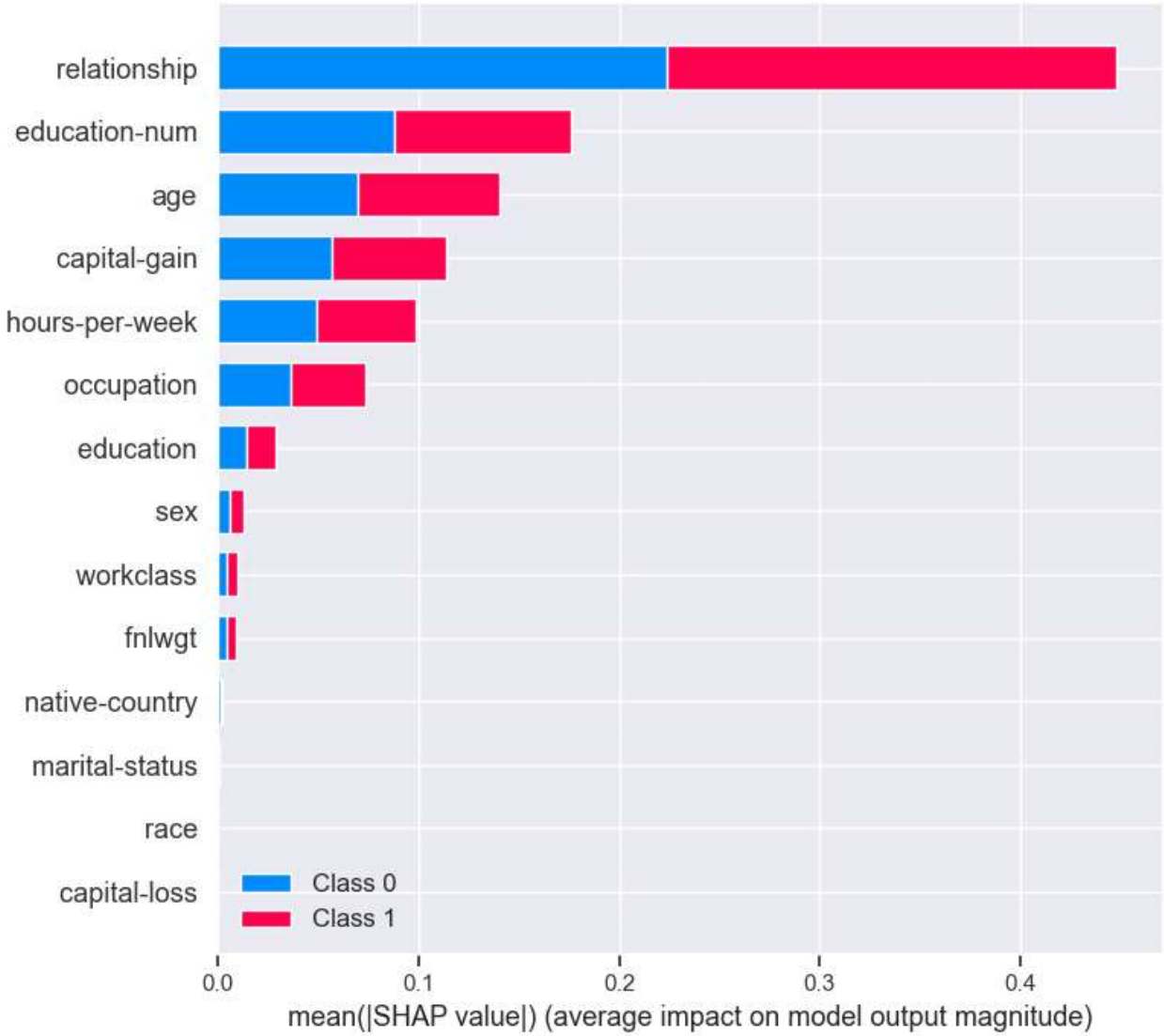
In [46]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score,
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, 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 :  0.7913403961308154
Precision Score :  0.7913403961308153
Recall Score :  0.7913403961308153
Jaccard Score :  0.6547256097560976
Log Loss :  7.206989936020279
```

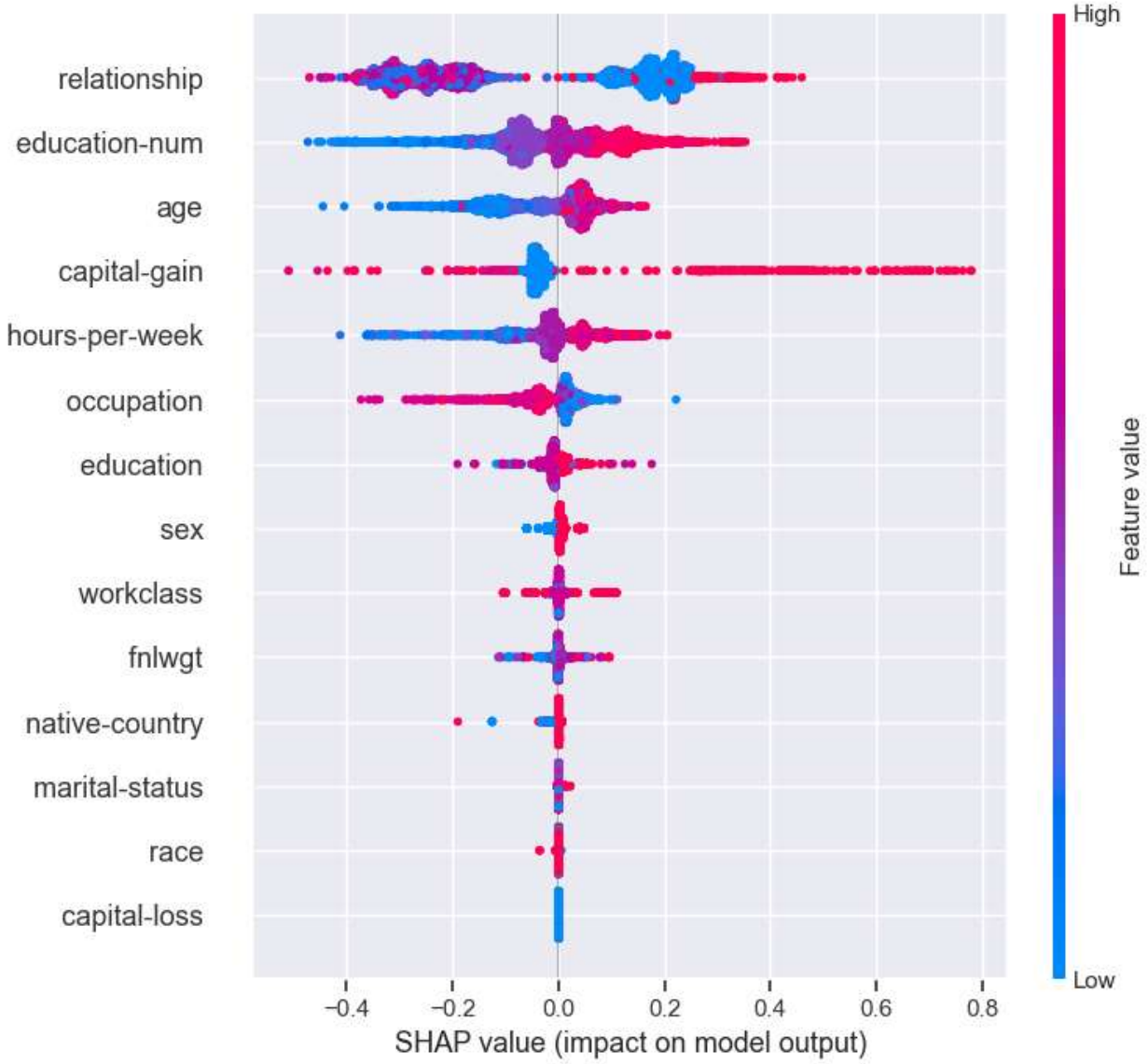
In [47]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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()
```



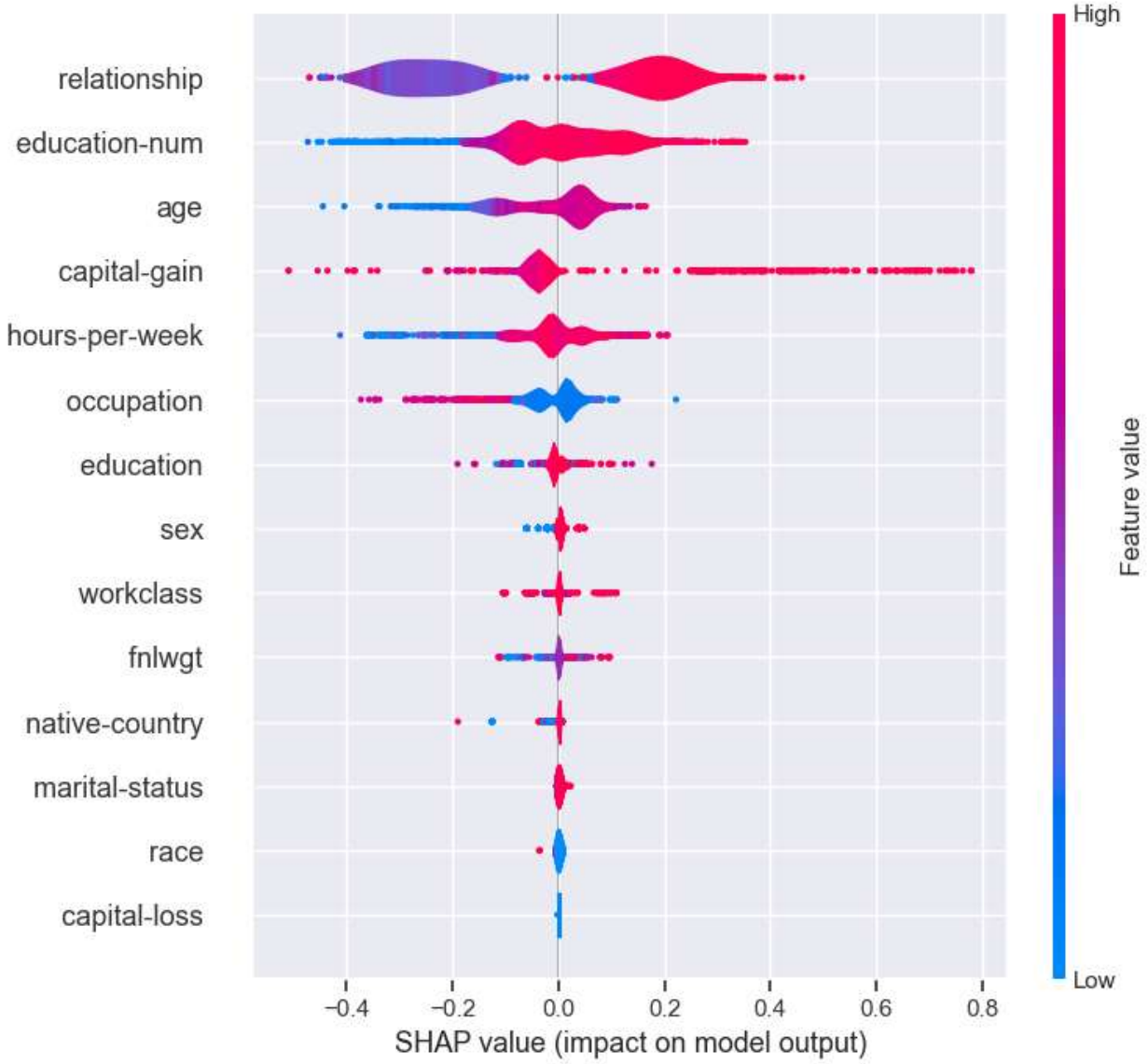
In [48]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [49]:
```
# 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)
```

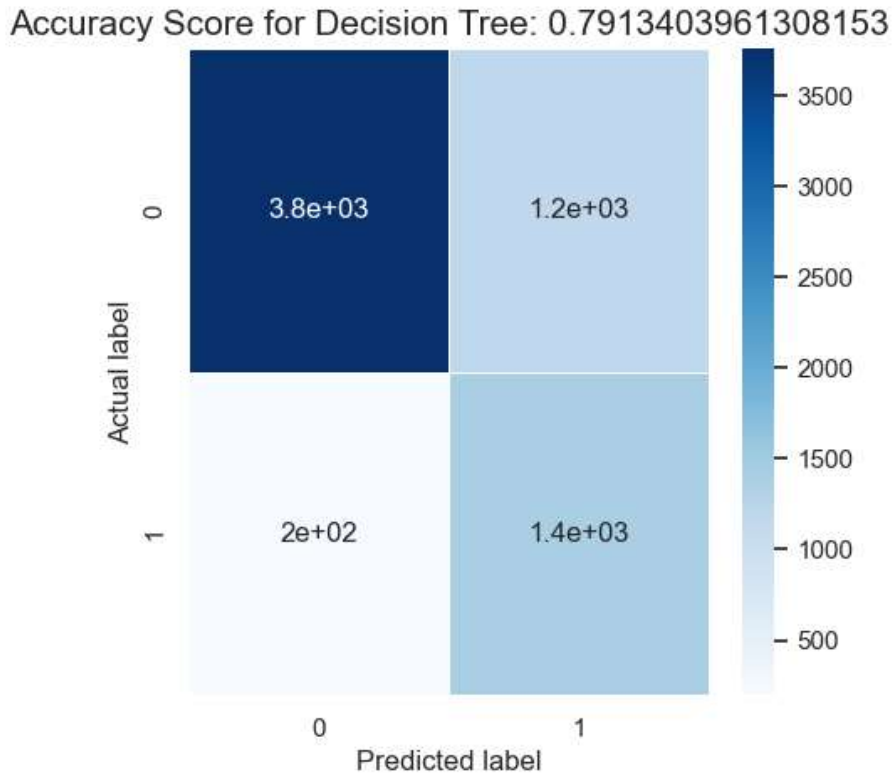


In [50]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



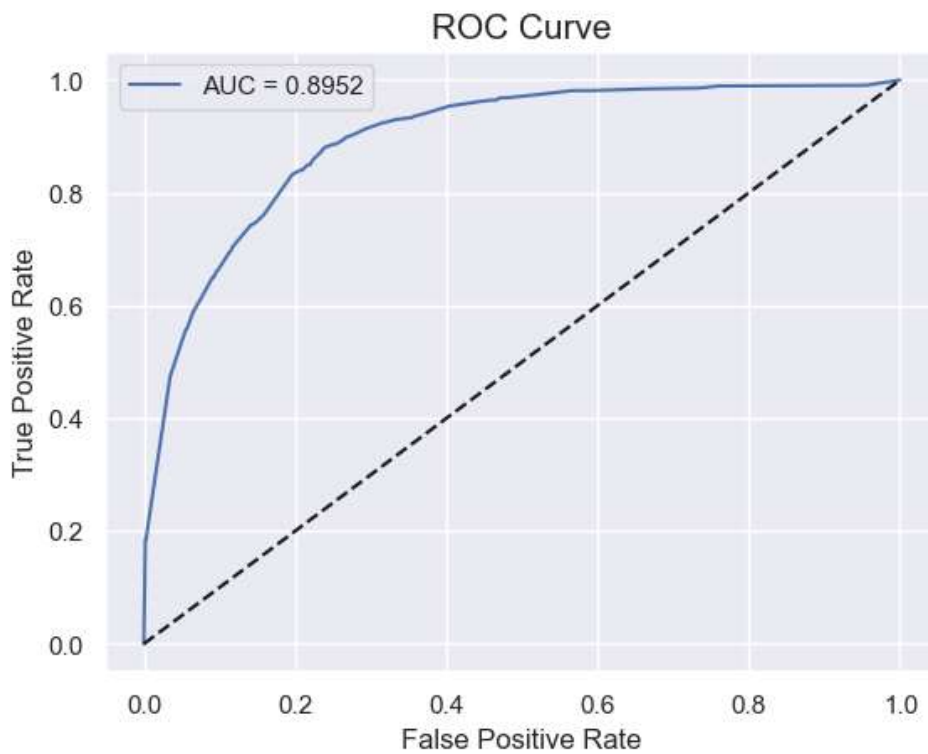
In [51]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[51]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.7913403961308153')
```



In [52]:
```
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
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[52]:
```
<matplotlib.legend.Legend at 0x1f5615eda00>
```



XGBoost

In [54]:
```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Create an XGBoost classifier
xgb = XGBClassifier()

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

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

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

```
{'gamma': 0.2, 'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
```

In [55]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0.2, learning_rate=0.1, max_depth=5, n_estimators=200)
xgb.fit(X_train, y_train)
```

Out[55]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

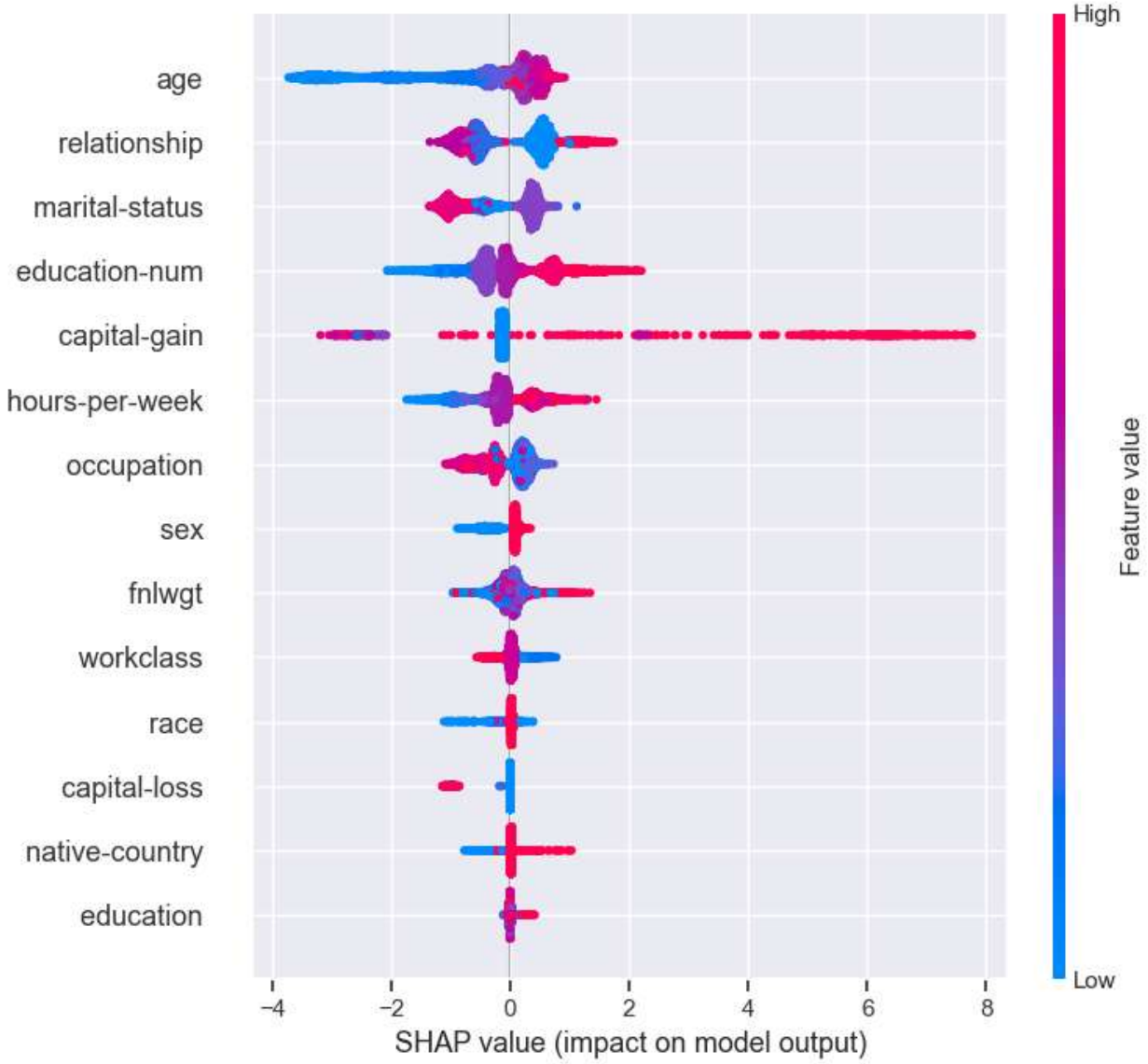
In [56]:
```
from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 85.12 %
```

In [57]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score,
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)))
```

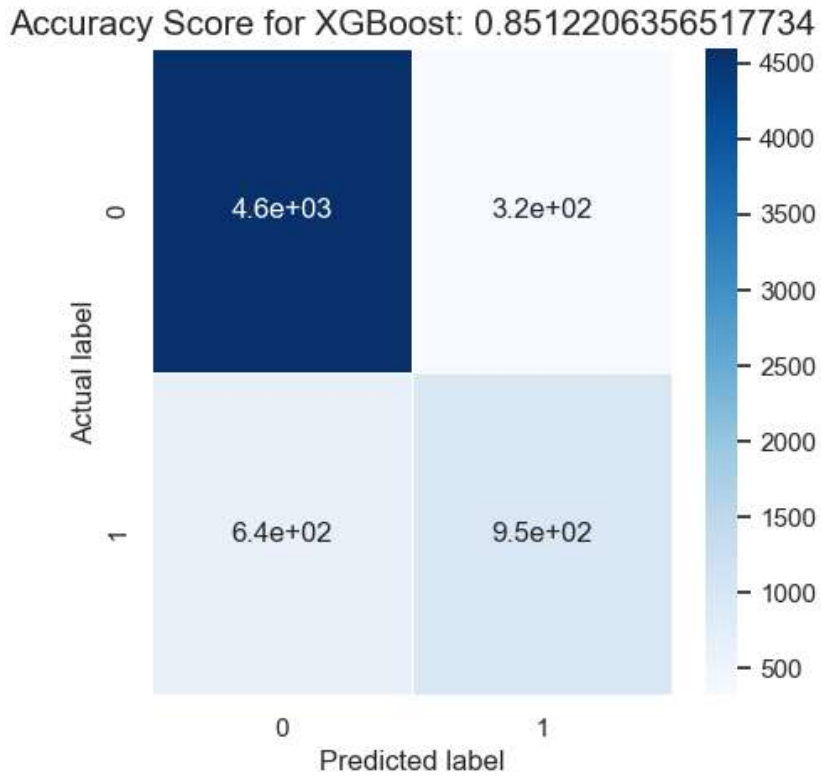
```
F-1 Score :  0.8512206356517734
Precision Score :  0.8512206356517734
Recall Score :  0.8512206356517734
Jaccard Score :  0.7409783480352847
Log Loss :  5.138697097471713
```

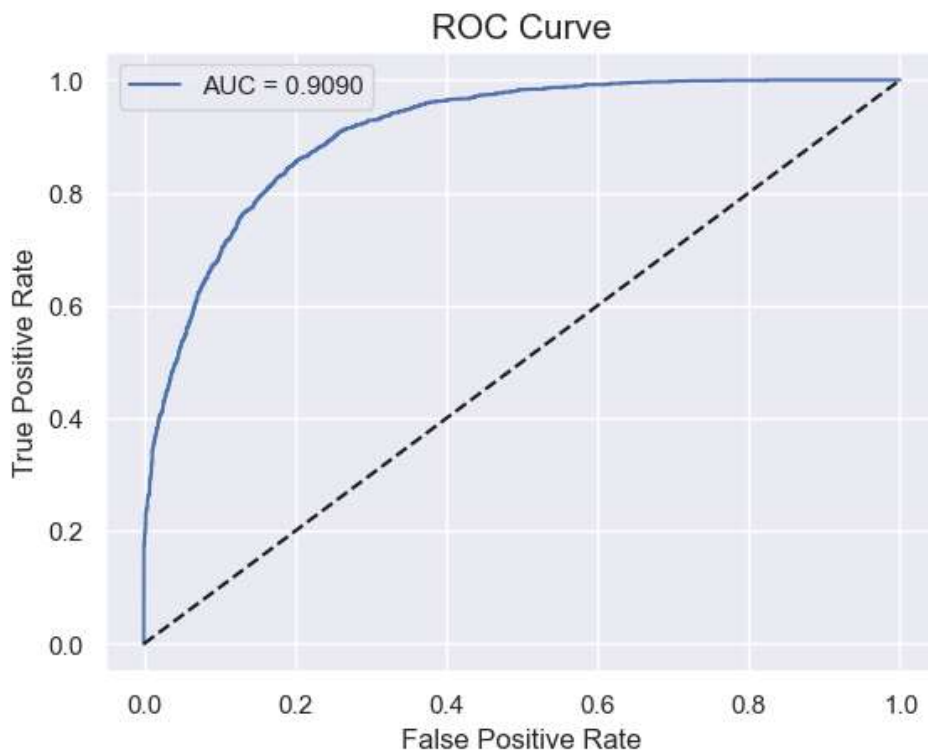
In [58]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [59]:
```
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[59]:
```
Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.8512206356517734')
```



In [60]:
```
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
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[60]:
```
<matplotlib.legend.Legend at 0x1f55d5b5670>
```