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

Out[2]:

	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan
0	Male	No	0	Graduate	No	584900	0.0	
1	Male	Yes	1	Graduate	No	458300	150800.0	
2	Male	Yes	0	Graduate	Yes	300000	0.0	
3	Male	Yes	0	Not Graduate	No	258300	235800.0	
4	Male	No	0	Graduate	No	600000	0.0	

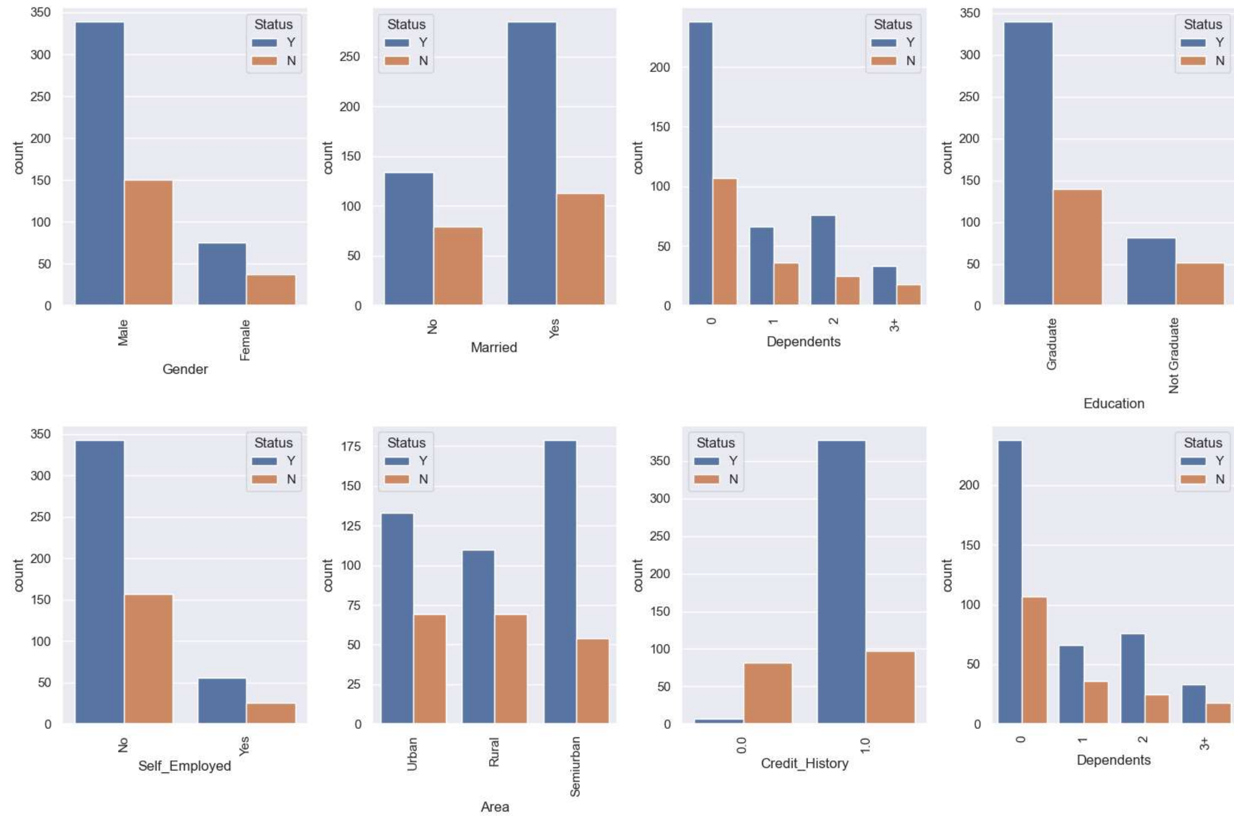
Data Preprocessing Part 1

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

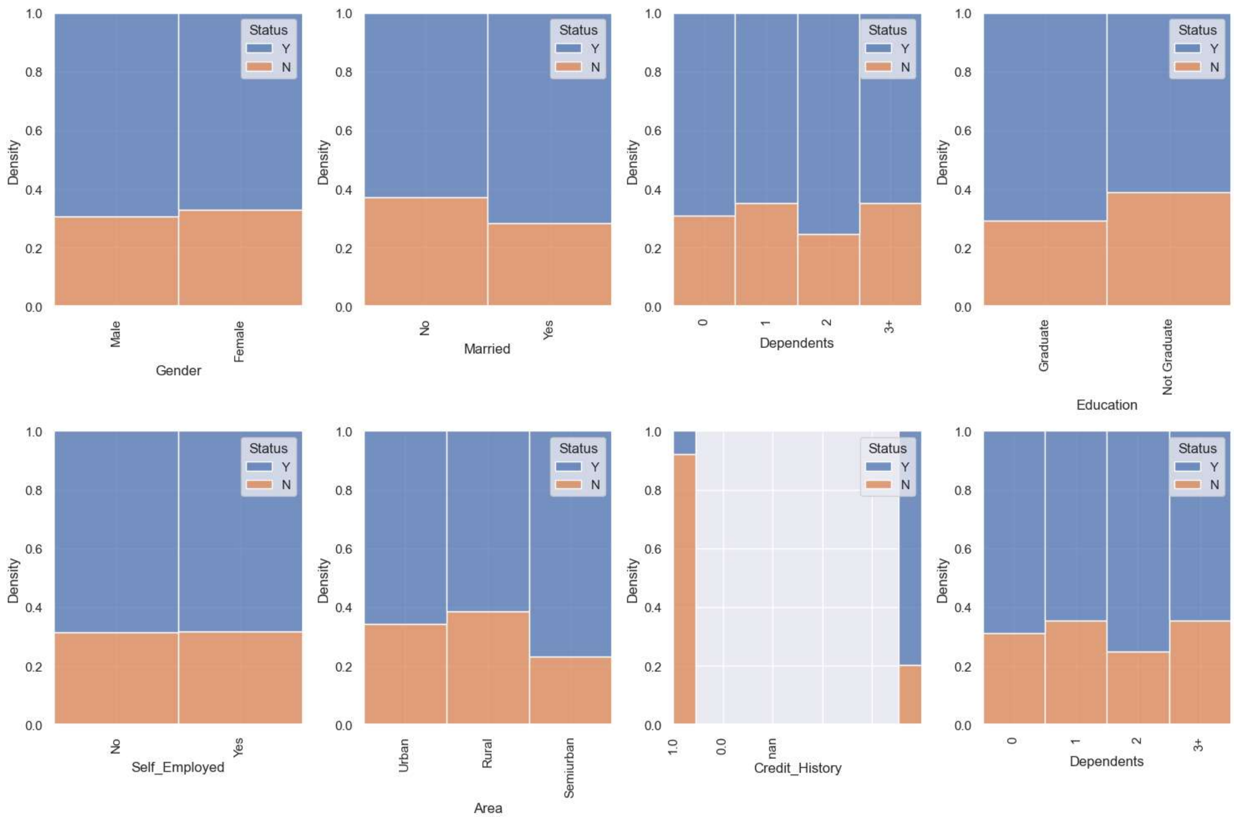
Out[3]:
```
Gender           2
Married          2
Dependents       4
Education        2
Self_Employed    2
Area             3
Status           2
dtype: int64
```

Exploratory Data Analysis

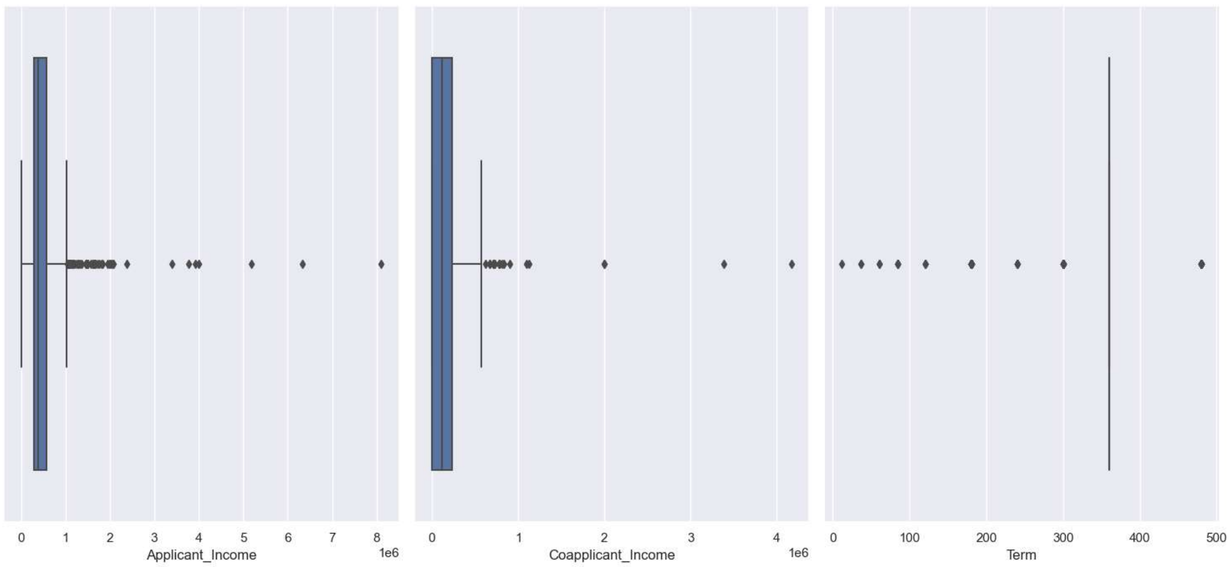
In [4]:
```python
# list of categorical variables to plot
cat_vars = ['Gender', 'Married', 'Dependents', 'Education',
            'Self_Employed', 'Area', 'Credit_History', 'Dependents']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(15, 10))
axs = axs.flatten()

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

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



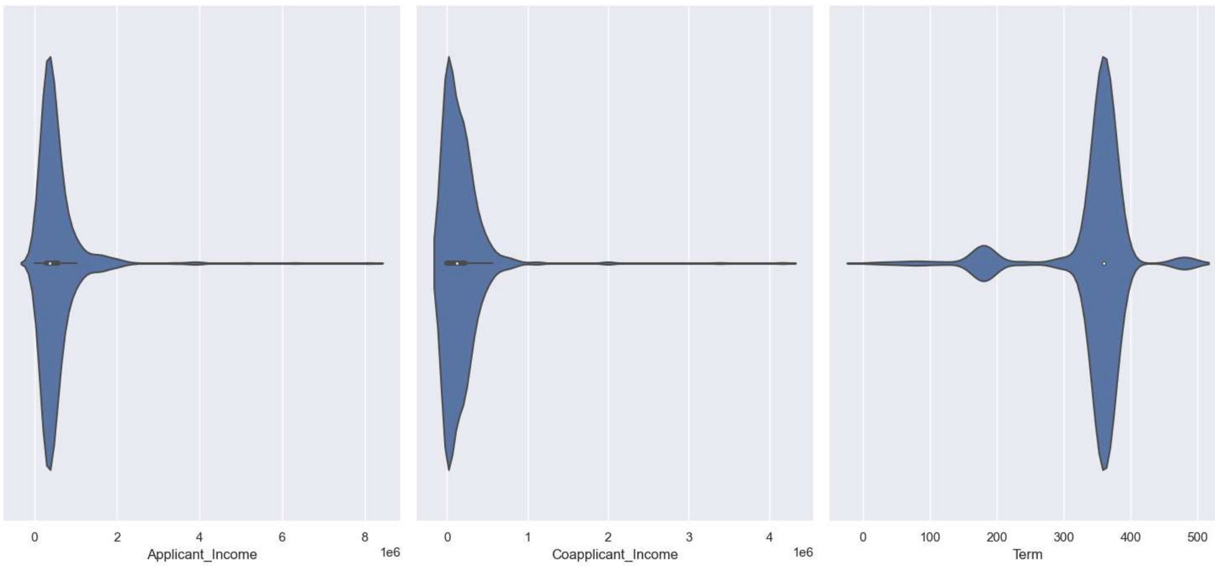
In [5]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Gender', 'Married', 'Dependents', 'Education',
            'Self_Employed', 'Area', 'Credit_History', 'Dependents']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(15, 10))
axs = axs.flatten()

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Status', 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()

# show plot
plt.show()
```



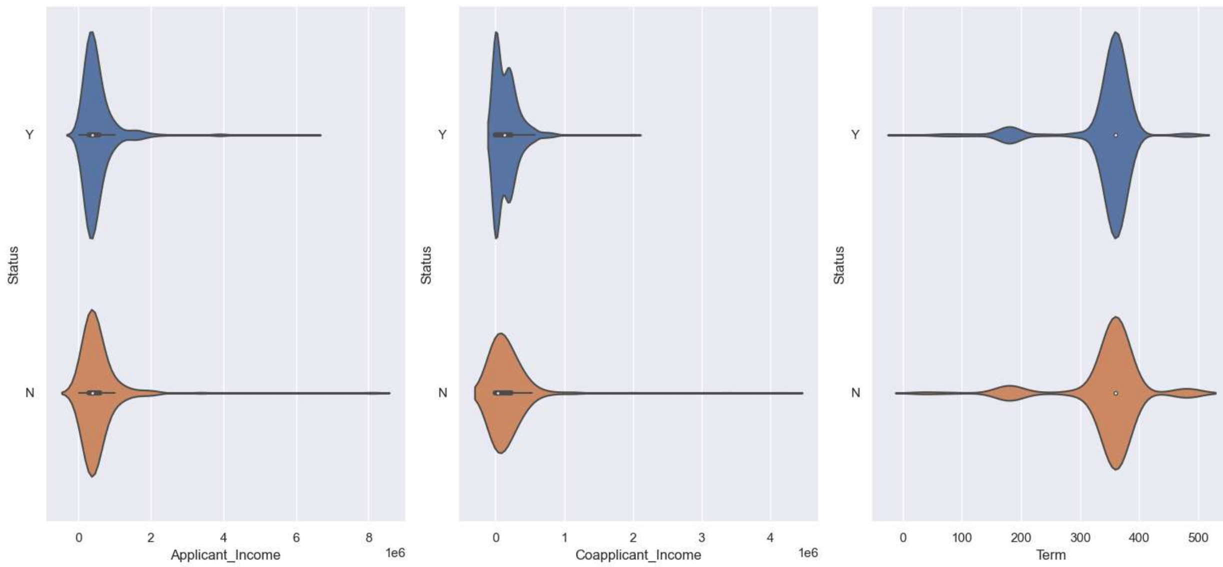
In [6]:
```
num_vars = ['Applicant_Income', 'Coapplicant_Income', 'Term']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 7))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



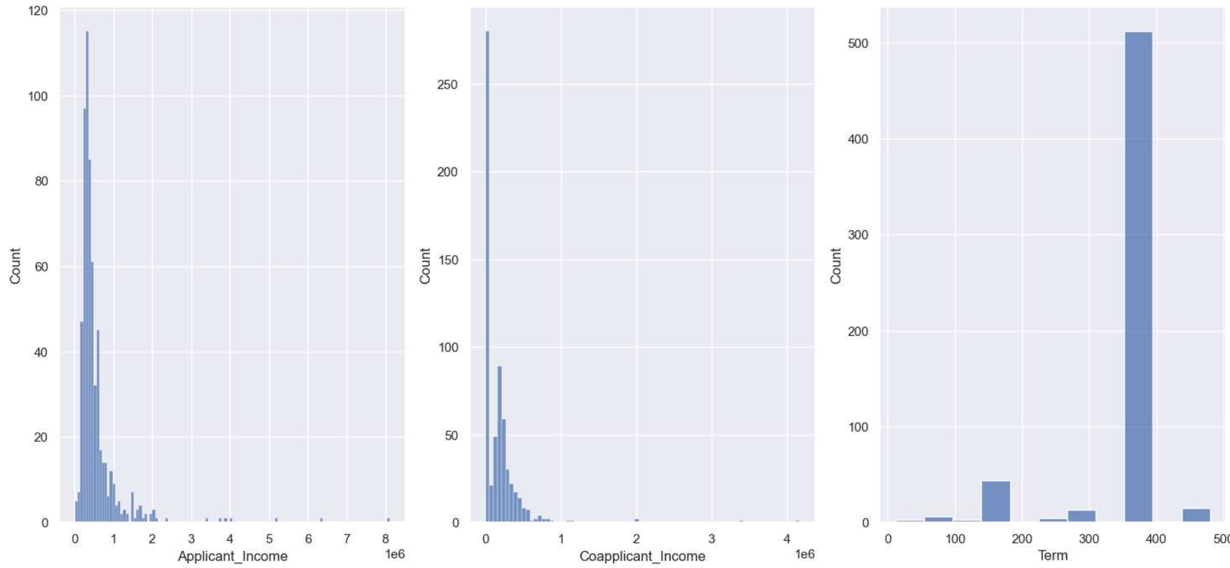
In [7]:
```
num_vars = ['Applicant_Income', 'Coapplicant_Income', 'Term']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 7))
axs = axs.flatten()

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

fig.tight_layout()
plt.show()
```



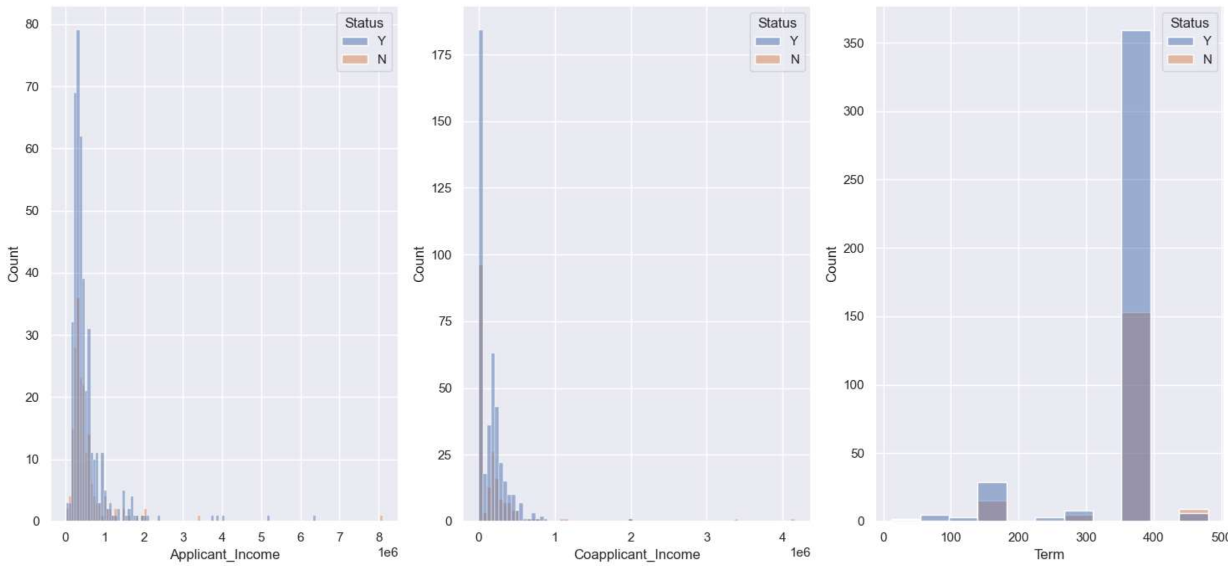
In [8]:
```
num_vars = ['Applicant_Income', 'Coapplicant_Income', 'Term']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 7))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



In [9]:
```
num_vars = ['Applicant_Income', 'Coapplicant_Income', 'Term']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 7))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



In [10]:
```
num_vars = ['Applicant_Income', 'Coapplicant_Income', 'Term']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(15, 7))
axs = axs.flatten()

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

fig.tight_layout()

plt.show()
```



Data Preprocessing Part 2

In [11]:
```
df.head()
```

Out[11]:

	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan
0	Male	No	0	Graduate	No	584900	0.0	
1	Male	Yes	1	Graduate	No	458300	150800.0	
2	Male	Yes	0	Graduate	Yes	300000	0.0	
3	Male	Yes	0	Not Graduate	No	258300	235800.0	
4	Male	No	0	Graduate	No	600000	0.0	

In [12]:
```
#Check the missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[12]:
```
Credit_History    8.143322
Self_Employed     5.211726
Dependents        2.442997
Term              2.280130
Gender            2.117264
Married           0.488599
dtype: float64
```

In [13]:
```
# Fill null values with 'Unknown'
df.fillna('Unknown', inplace=True)

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

Out[13]:
```
Series([], dtype: float64)
```

In [14]:
```
df.dtypes
```

Out[14]:
```
Gender                 object
Married                object
Dependents             object
Education              object
Self_Employed          object
Applicant_Income        int64
Coapplicant_Income    float64
Loan_Amount             int64
Term                   object
Credit_History         object
Area                   object
Status                 object
dtype: object
```

Label Encoding for Object datatype

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()}")
```

```
Gender: ['Male' 'Female' 'Unknown']
Married: ['No' 'Yes' 'Unknown']
Dependents: ['0' '1' '2' '3+' 'Unknown']
Education: ['Graduate' 'Not Graduate']
Self_Employed: ['No' 'Yes' 'Unknown']
Term: [360.0 120.0 240.0 'Unknown' 180.0 60.0 300.0 480.0 36.0 84.0 12.0]
Credit_History: [1.0 0.0 'Unknown']
Area: ['Urban' 'Rural' 'Semiurban']
Status: ['Y' 'N']
```

In [16]:
```
# Convert selected columns to string data type
df[['Term', 'Credit_History']] = df[['Term', 'Credit_History']].astype(str)
```

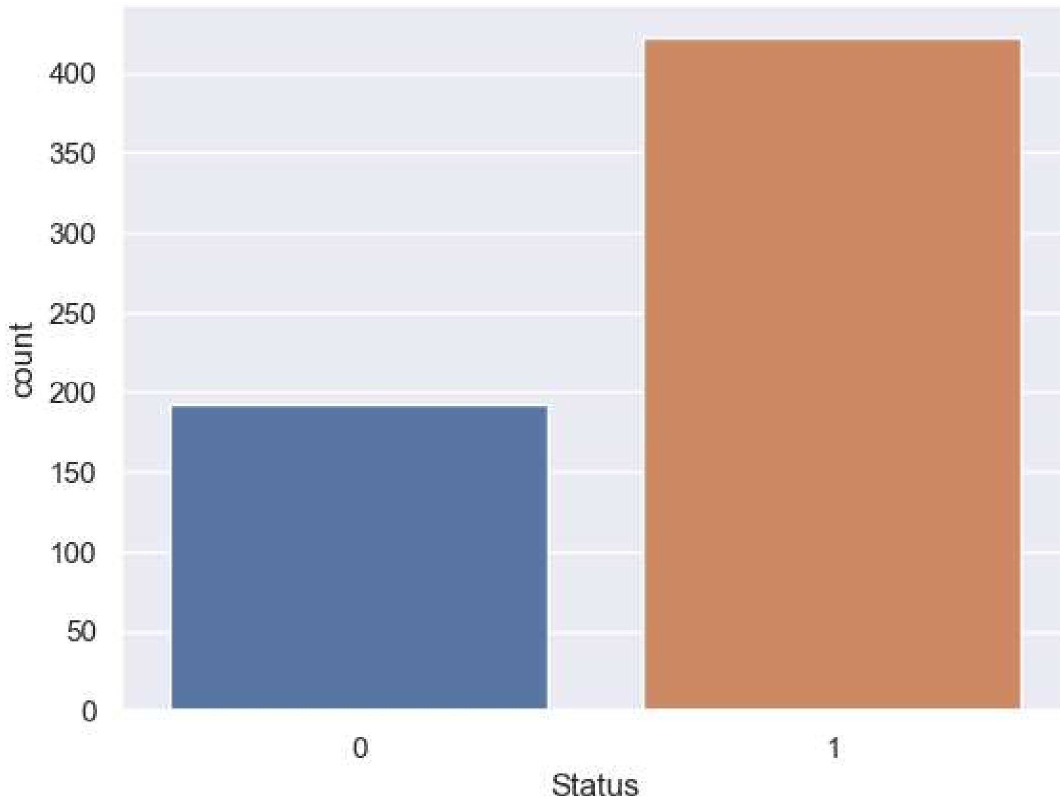
In [17]:
```
from sklearn import preprocessing

# 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()}")
```

```
Gender: [1 0 2]
Married: [0 2 1]
Dependents: [0 1 2 3 4]
Education: [0 1]
Self_Employed: [0 2 1]
Term: [ 6  1  3 10  2  8  4  7  5  9  0]
Credit_History: [1 0 2]
Area: [2 0 1]
Status: [1 0]
```

Check if the Label 'Status' is balanced or not

In [18]:
```
sns.countplot(df['Status'])
df['Status'].value_counts()
```

Out[18]:
```
1    422
0    192
Name: Status, dtype: int64
```

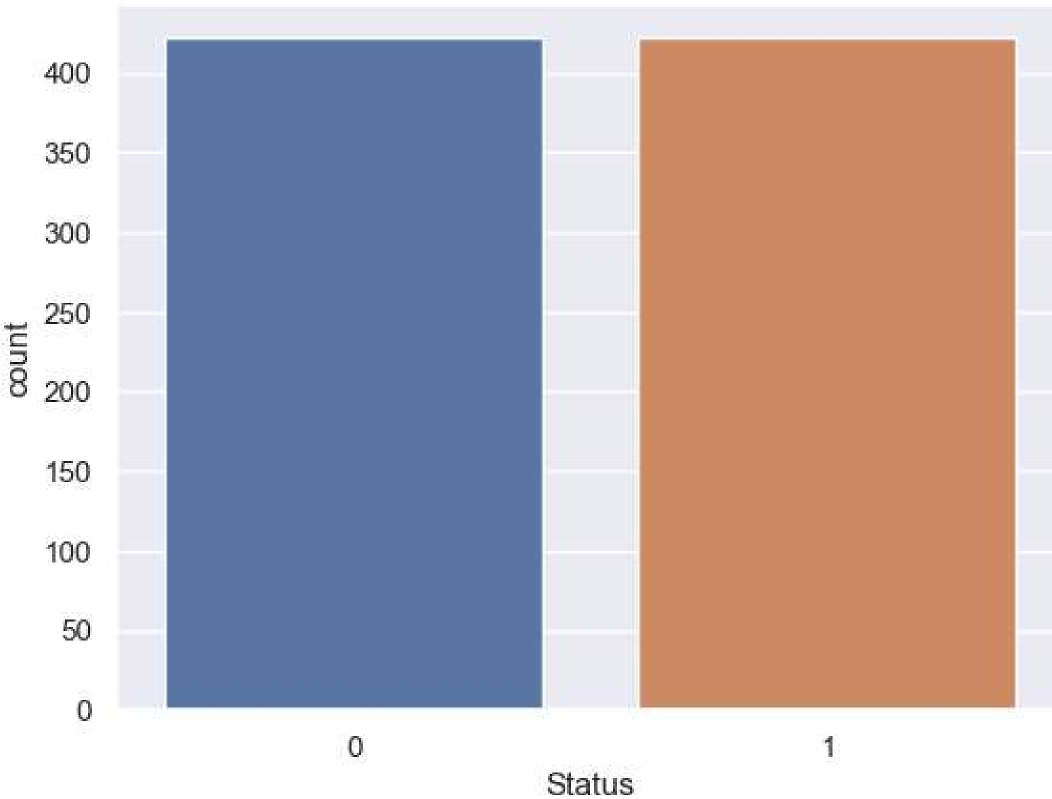


Oversampling Minority Class to balance the Label

In [19]:
```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df[(df['Status']==1)]
df_minority = df[(df['Status']==0)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,    # sample with replacement
                                 n_samples= 422, # to match majority class
                                 random_state=0)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

In [20]:
```
sns.countplot(df_upsampled['Status'])
df_upsampled['Status'].value_counts()
```

Out[20]:
```
0    422
1    422
Name: Status, dtype: int64
```



Remove Outlier using IQR because there are alot of extreme value

In [21]:
```
df_upsampled.shape
```

Out[21]:
```
(844, 12)
```

In [22]:
```
# specify the columns to remove outliers from dataframe
column_names = ['Applicant_Income', 'Coapplicant_Income', 'Term']

# remove outliers for each selected column using the IQR method
for column_name in column_names:
    Q1 = df_upsampled[column_name].quantile(0.25)
    Q3 = df_upsampled[column_name].quantile(0.75)
    IQR = Q3 - Q1
    df_upsampled = df_upsampled[~((df_upsampled[column_name] < (Q1 - 1.5 * IQR)) | (d

df_upsampled.head()
```

Out[22]:

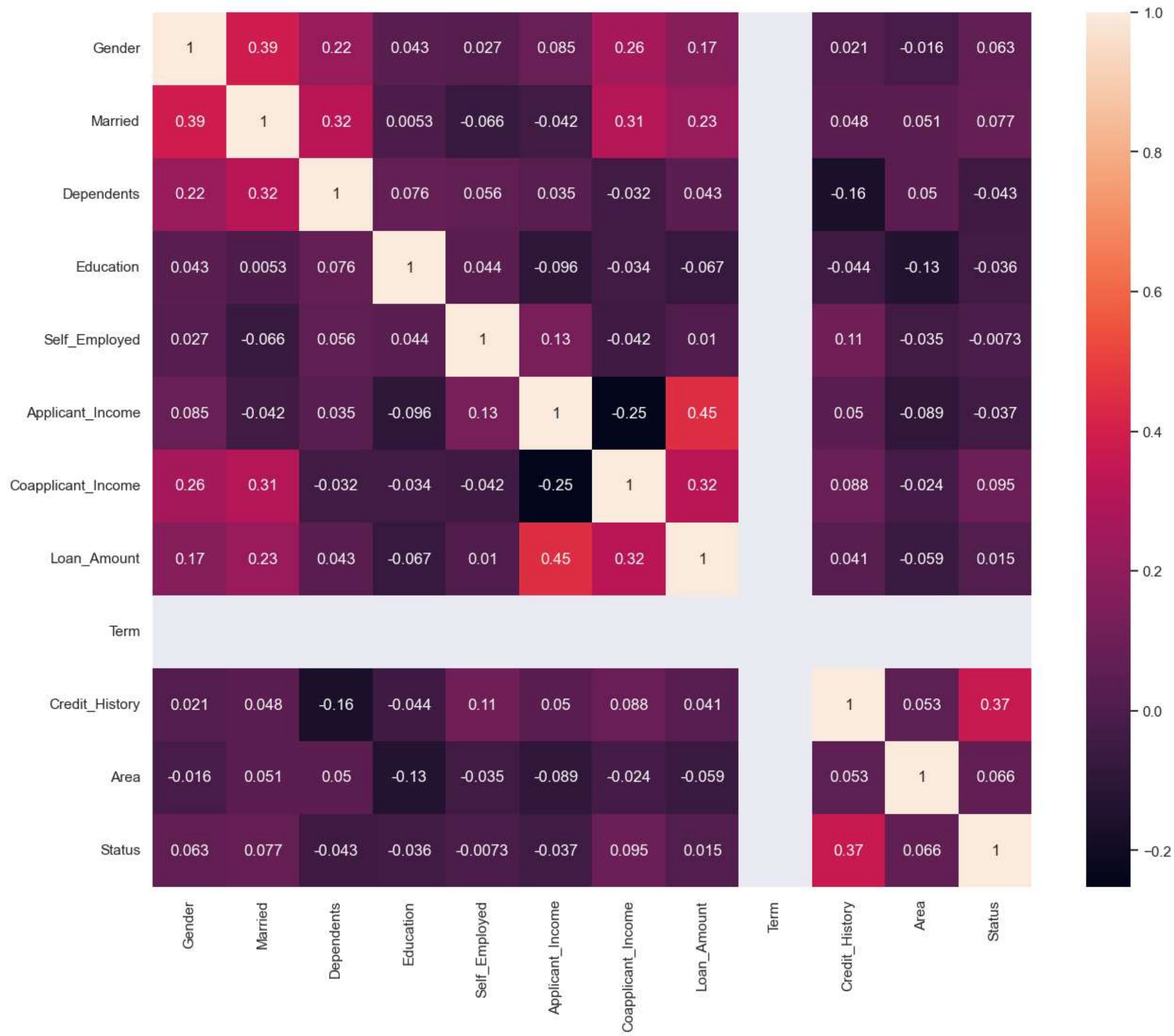
	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Lo
148	0	0	0	0	0	1000000	166600.0	
338	0	0	3	1	0	183000	0.0	
24	1	2	1	0	1	371700	292500.0	
57	1	2	0	0	0	336600	220000.0	
107	1	0	0	1	1	733300	0.0	

In [23]:
```
#Check the shape after outlier removal
df_upsampled.shape
```

Out[23]: (614, 12)

In [24]:
```
plt.figure(figsize=(15,12))
sns.heatmap(df_upsampled.corr(), fmt='.2g', annot=True)
```

Out[24]: <AxesSubplot:>



In [25]:
```
df_upsampled.drop(columns='Term', inplace=True)
```

Train Test Split

In [26]:
```
X = df_upsampled.drop('Status', axis=1)
y = df_upsampled['Status']
```

In [27]:
```
#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 [28]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier()
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4]
}

# 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': 2}
```

In [29]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=8, min_samples_leaf=1, min_s
dtree.fit(X_train, y_train)
```

Out[29]:
```
DecisionTreeClassifier(max_depth=8, random_state=0)
```

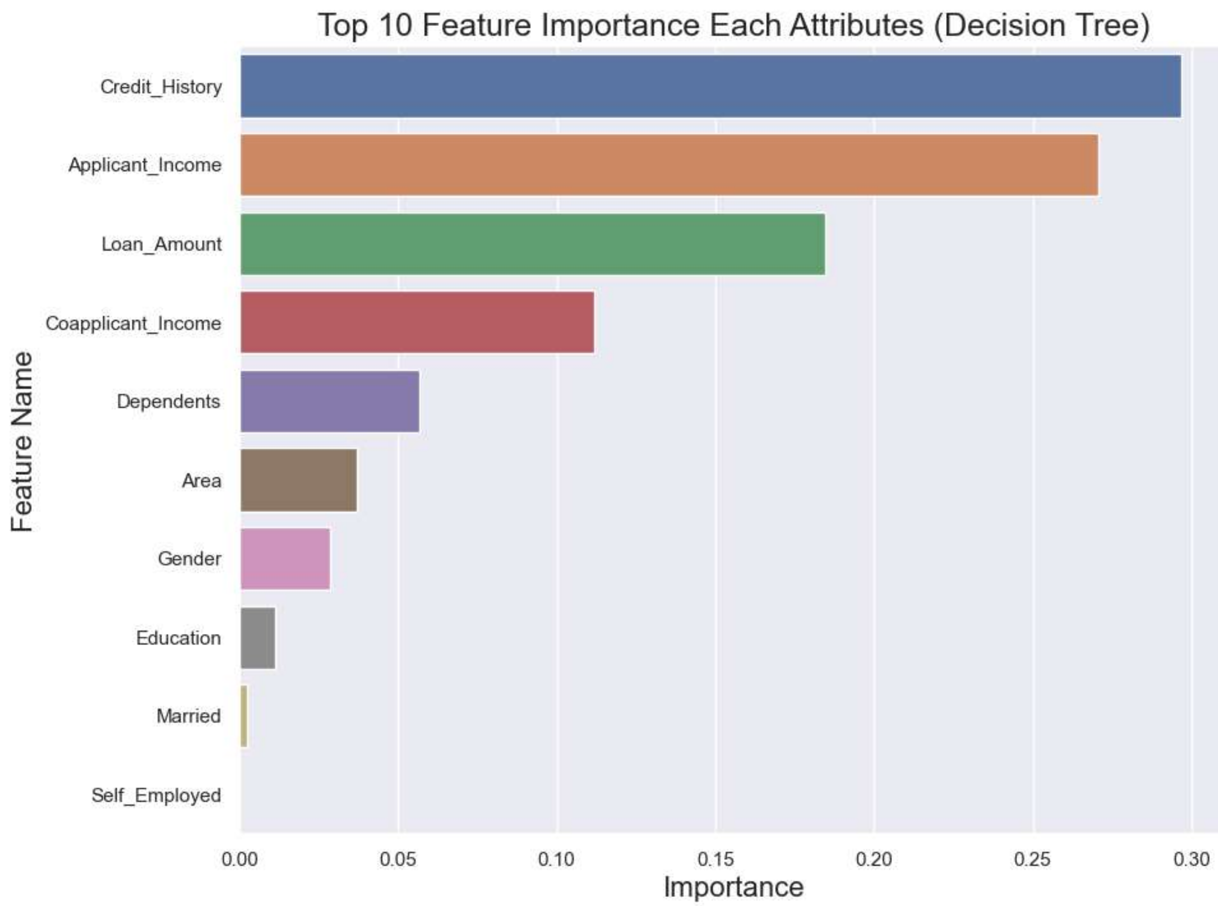
In [30]:
```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 86.18 %
```

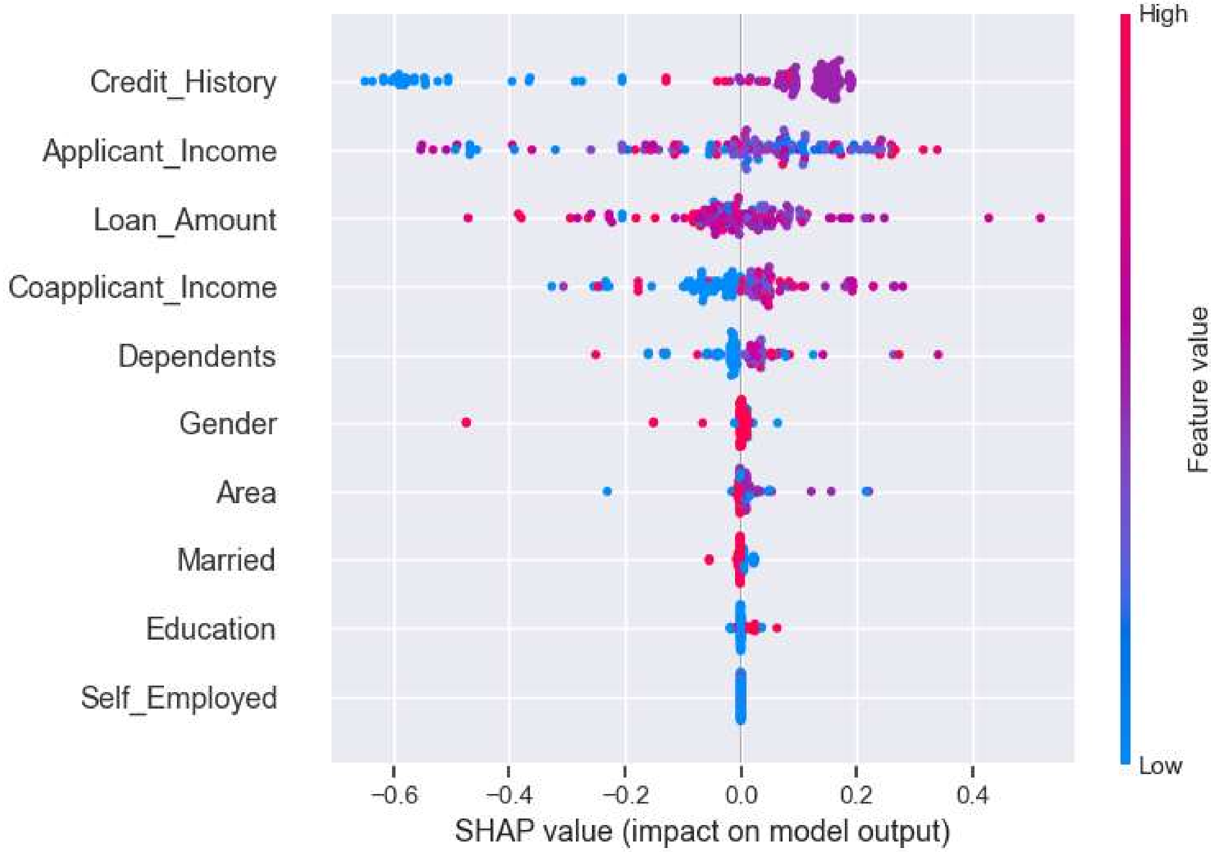
In [31]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, 
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.861788617886179
Precision Score :  0.8617886178861789
Recall Score :  0.8617886178861789
Jaccard Score :  0.7571428571428571
Log Loss :  4.773697527605633
```

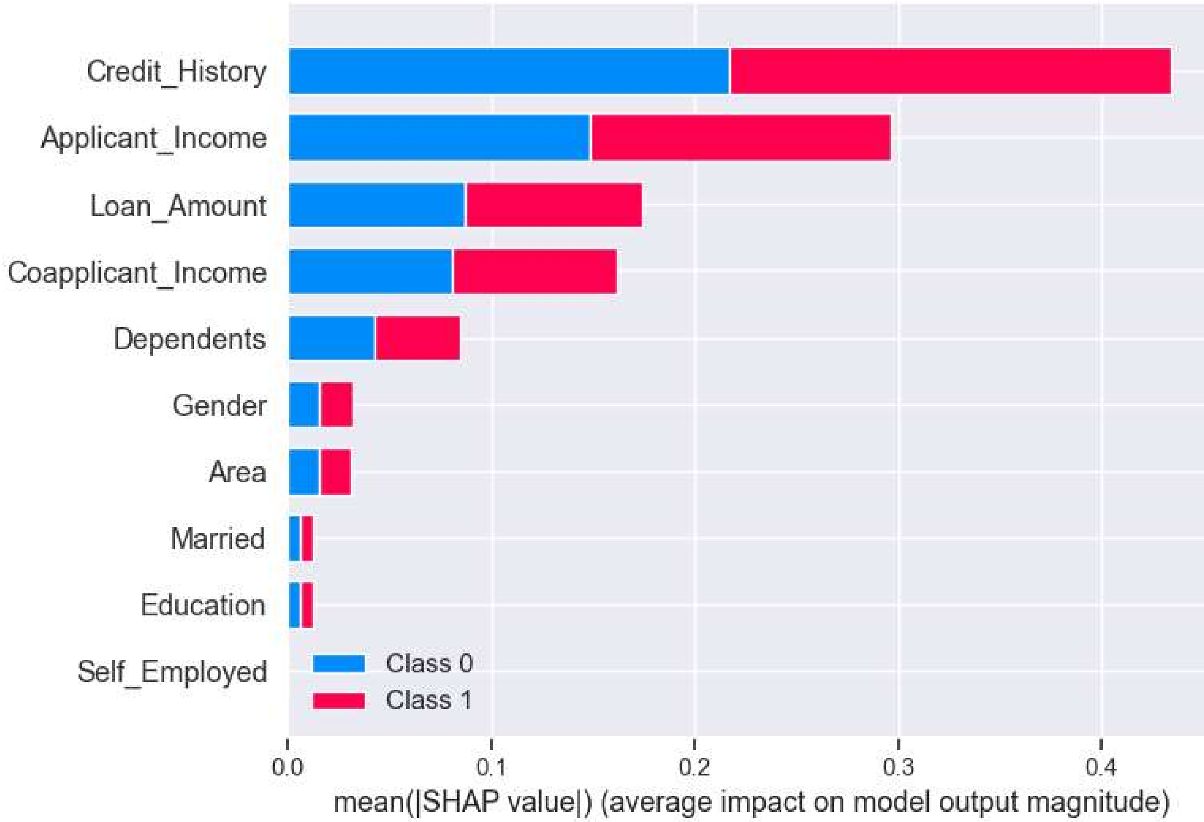
In [32]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [33]:
```
import shap
# 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)
```

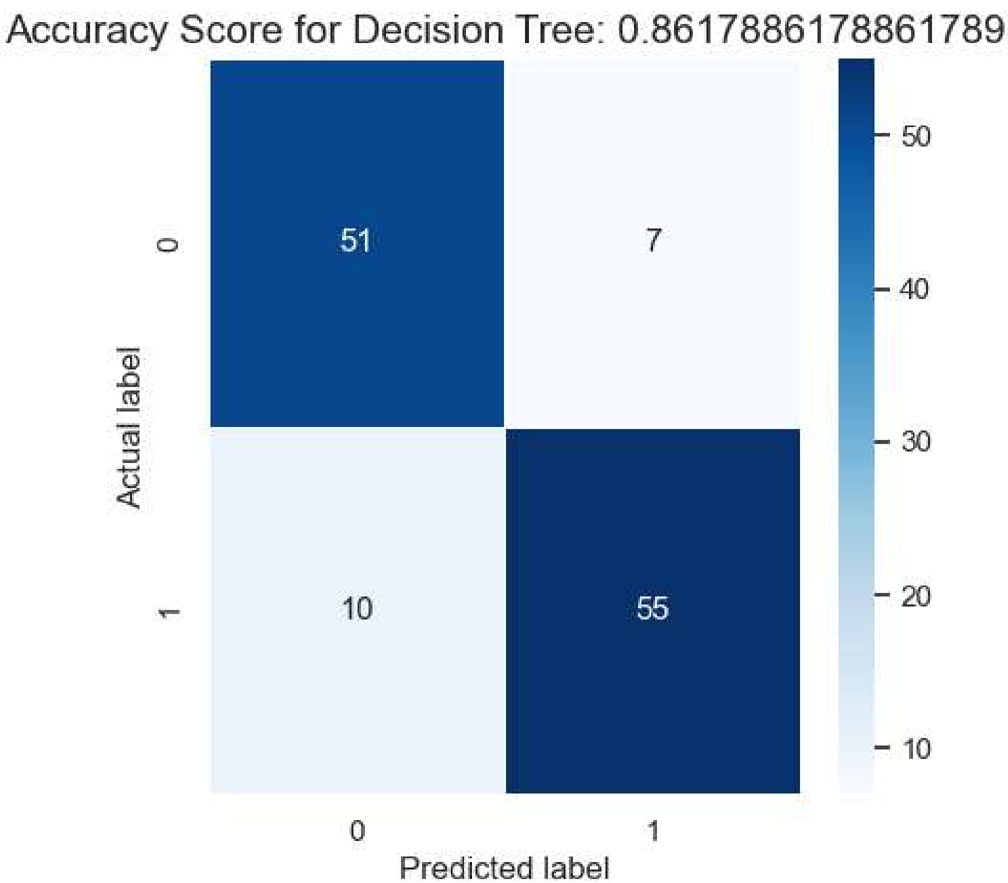


In [35]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



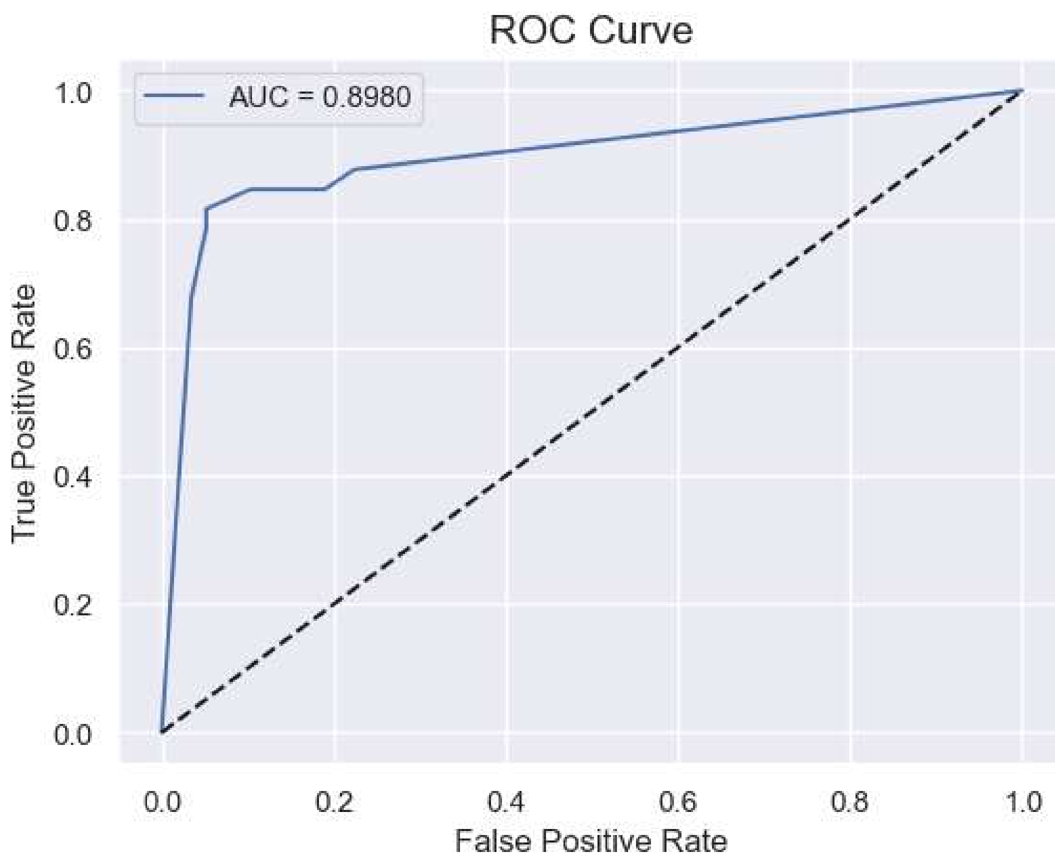
In [37]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test,
plt.title(all_sample_title, size = 15)
```

Out[37]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8617886178861789')
```



In [38]:
```
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'])
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_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[38]: <matplotlib.legend.Legend at 0x20f4afc3310>



Random Forest

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier()
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None]
}

# 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': None, 'max_features': 'log2', 'n_estimators': 200}
```

In [40]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='log2', n_estimators=200)
rfc.fit(X_train, y_train)
```

Out[40]:
```
RandomForestClassifier(max_features='log2', n_estimators=200, random_state=0)
```

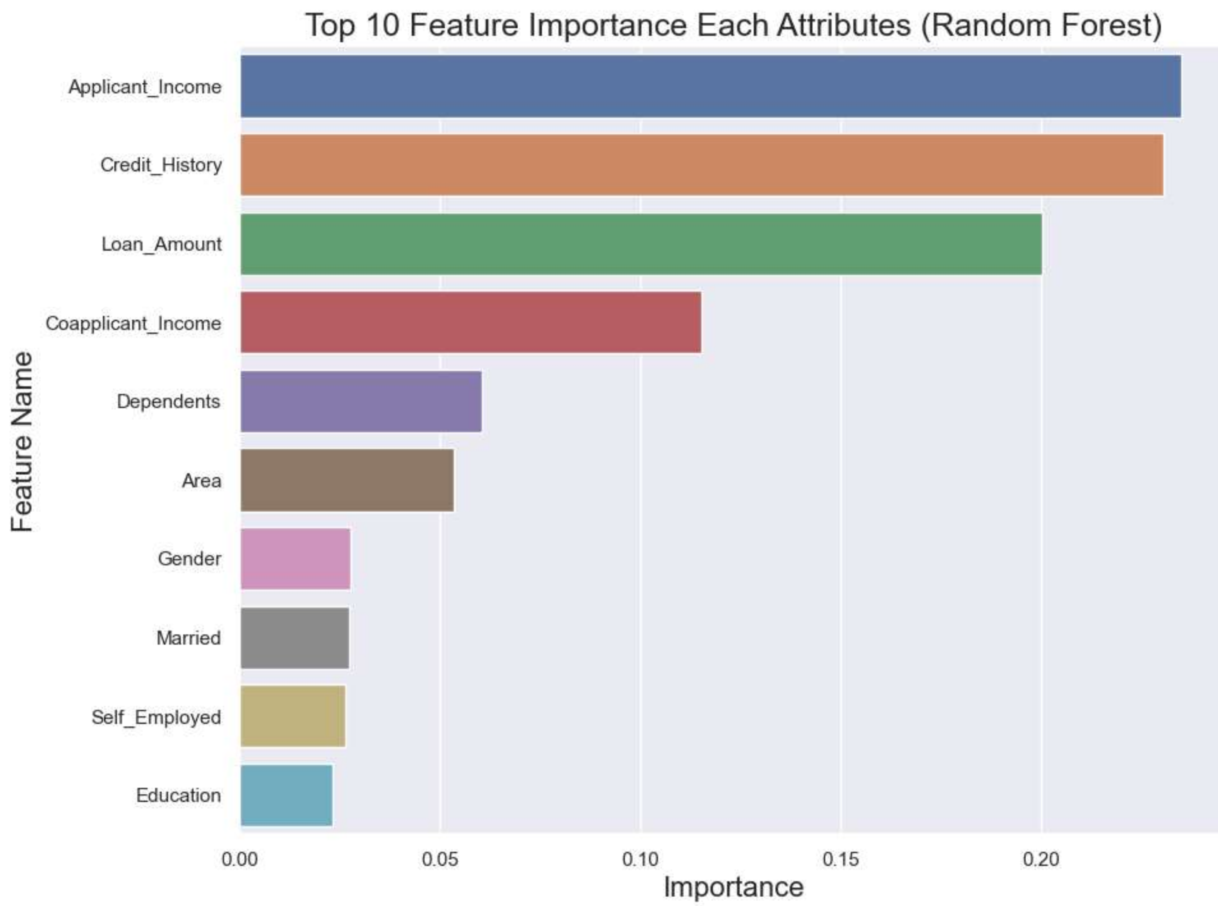
In [41]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 95.12 %
```

In [42]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, j
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)))
```

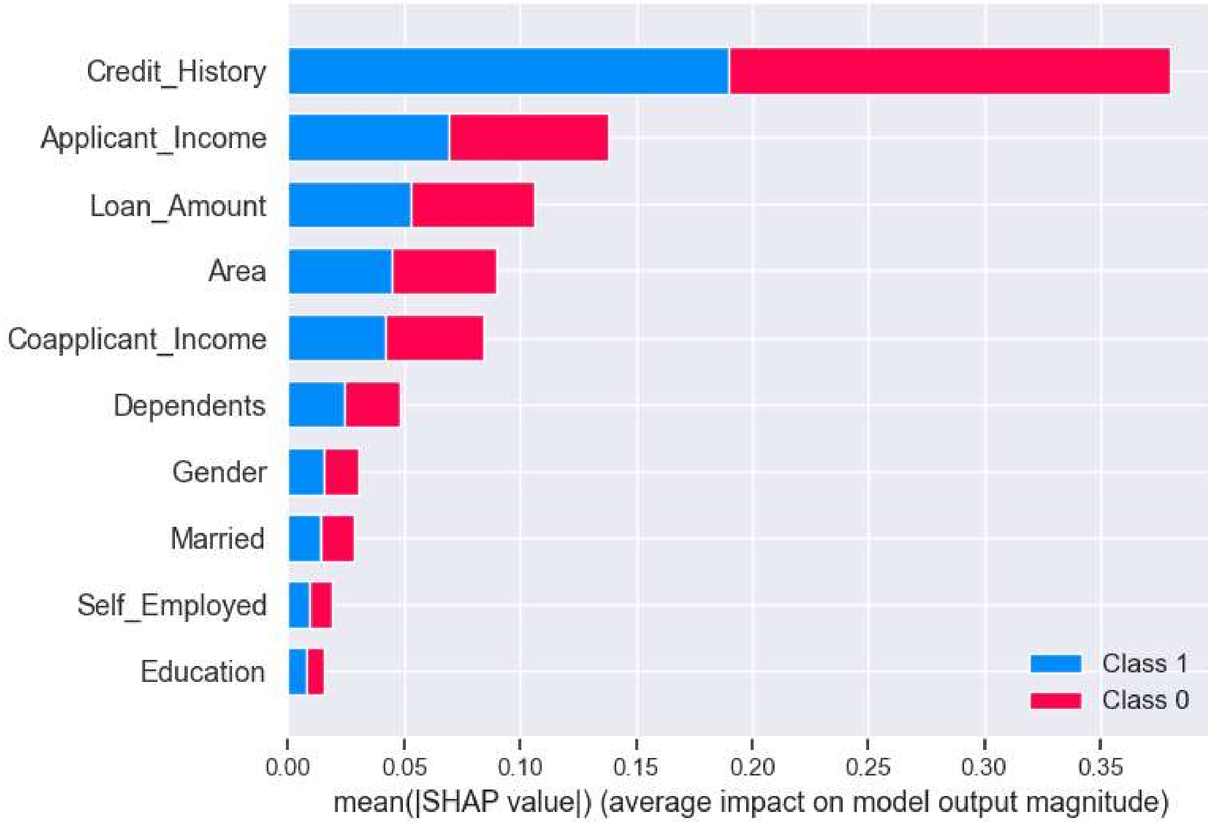
```
F-1 Score :  0.9512195121951219
Precision Score :  0.9512195121951219
Recall Score :  0.9512195121951219
Jaccard Score :  0.9069767441860465
Log Loss :  1.6848443638958128
```

In [43]:
```
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()
```



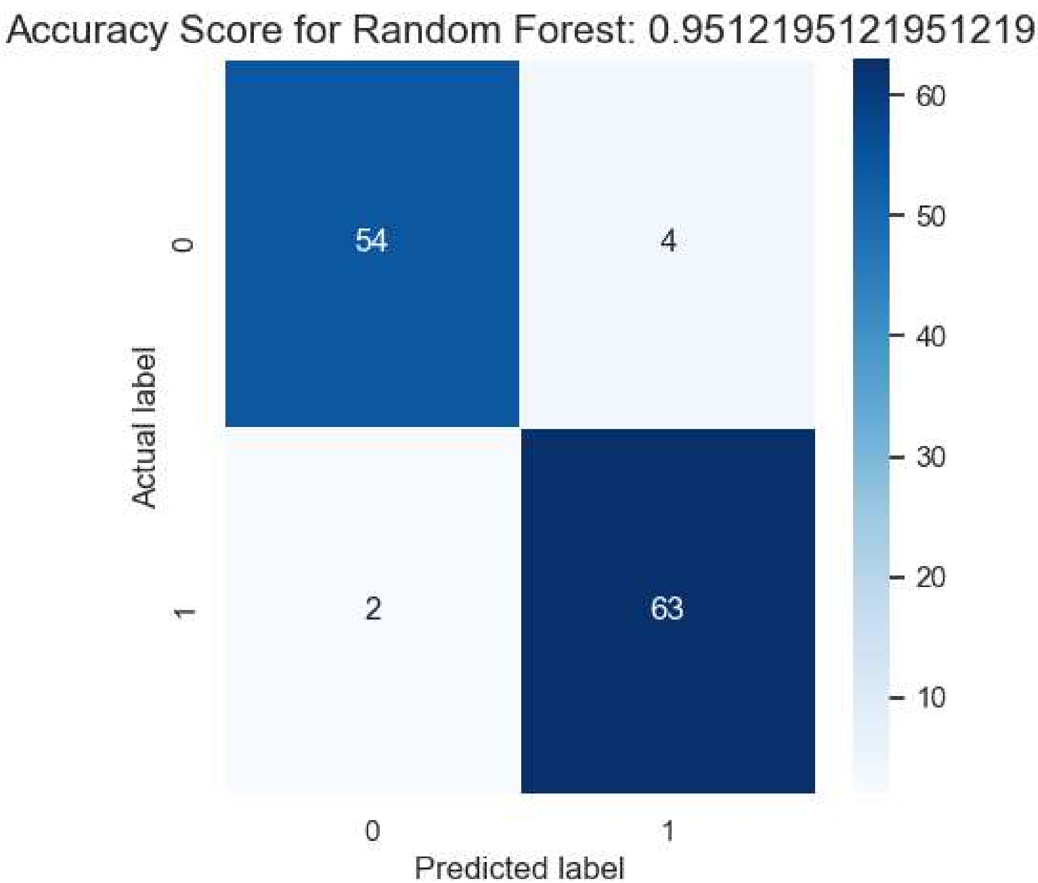
In [44]:
```
import shap
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```

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



In [46]:
```
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_
plt.title(all_sample_title, size = 15)
```

Out[46]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9512195121951219')
```



In [47]:
```
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'])
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_prob

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[47]: <matplotlib.legend.Legend at 0x20f4b644fd0>