Credit Card Approval

Import Libraries

In [85]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
```

Load Dataset

In [86]:
```
#load the dataset
df = pd.read_csv('credit_risk_dataset.csv')

# Display the first few rows of the dataset
df.head()
```

Out [86]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	RENT	123.0	PERSONAL	35000	16.02	1	0.59
1	21	9600	OWN	5.0	EDUCATION	1000	11.14	0	0.10
2	25	9600	MORTGAGE	1.0	MEDICAL	5500	12.87	1	0.57
3	23	65500	RENT	4.0	MEDICAL	35000	15.23	1	0.53
4	24	54400	RENT	8.0	MEDICAL	35000	14.27	1	0.55

Dataset Overview

In [87]:
```
# Check the size of the dataset
df.shape
```

Out [87]: (32581, 11)

In [88]:
```
# Display basic information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  32581 non-null  int64  
 1   person_income               32581 non-null  int64  
 2   person_home_ownership       32581 non-null  object 
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object 
 5   loan_amnt                   32581 non-null  int64  
 6   loan_int_rate               29465 non-null  float64
 7   loan_status                 32581 non-null  int64  
 8   loan_percent_income         32581 non-null  float64
 9   cb_person_default_on_file   32581 non-null  object 
 10  cb_person_cred_hist_length  32581 non-null  int64  
dtypes: float64(3), int64(5), object(3)
memory usage: 2.7+ MB
```

Check and Handle Missing Values

In [89]:
```
# Check for missing values in the dataset
df.isnull().sum()
```

Out [89]:
```
person_age                     0
person_income                  0
person_home_ownership          0
person_emp_length            895
loan_intent                    0
loan_amnt                      0
```

```
loan_int_rate                3116
loan_status                     0
loan_percent_income             0
cb_person_default_on_file       0
cb_person_cred_hist_length      0
dtype: int64
```

In [90]:
```
#check if missing values are indicated as nan
column = ['person_age', 'person_income', 'person_home_ownership', 'person_emp_length', 'loan_intent', 'loan_am

for missing_nan in column:
    if df[missing_nan].isna().any():
        print(f'Missing values are written as nan in column {missing_nan}')
```

```
Missing values are written as nan in column person_emp_length
Missing values are written as nan in column loan_int_rate
```

In [91]:
```
# Verify that missing values are handled
df.isnull().sum()
```

Out [91]:
```
person_age                     0
person_income                  0
person_home_ownership          0
person_emp_length            895
loan_intent                    0
loan_amnt                      0
loan_int_rate               3116
loan_status                    0
loan_percent_income            0
cb_person_default_on_file      0
cb_person_cred_hist_length     0
dtype: int64
```

In [92]:
```
# Replace NaN values in 'person_emp_length' with 0
df['person_emp_length'] = df['person_emp_length'].fillna(0)
```

In [93]:
```
from sklearn.impute import SimpleImputer
impute = SimpleImputer(strategy='median')
df['loan_int_rate'] = impute.fit_transform(df[['loan_int_rate']])
```

In [94]:
```
# Impute missing target values with the mean
#loan_int_rate_mean = df['loan_int_rate'].median()
#df['loan_int_rate'].fillna(loan_int_rate_mean, inplace=True)
```

Encoding Categorical Values

In [95]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   person_age                  32581 non-null  int64
 1   person_income               32581 non-null  int64
 2   person_home_ownership       32581 non-null  object
 3   person_emp_length           32581 non-null  float64
 4   loan_intent                 32581 non-null  object
 5   loan_amnt                   32581 non-null  int64
 6   loan_int_rate               32581 non-null  float64
 7   loan_status                 32581 non-null  int64
 8   loan_percent_income         32581 non-null  float64
 9   cb_person_default_on_file   32581 non-null  object
 10  cb_person_cred_hist_length  32581 non-null  int64
dtypes: float64(3), int64(5), object(3)
memory usage: 2.7+ MB
```

From the dataset info, person_home_ownership, loan_intent & cb_person_default_on_file colums are categorical values

In [96]:
```
#check for categorical unique values
df['person_home_ownership'].unique()
```

Out [96]:
```
array(['RENT', 'OWN', 'MORTGAGE', 'OTHER'], dtype=object)
```

In [97]:
```

#create a dictionary to map the categorical value to numeric value equivalent
ownership = {'RENT':1, 'OWN':2, 'MORTGAGE':3, 'OTHER':4}

## Replace the categorical values in 'person_home_ownership' with their numeric equivalents
df['person_home_ownership'] = df['person_home_ownership'].replace(ownership)
```

In [98]:
```
#check if the change has been effected

df.head()
```

Out [98]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	1	123.0	PERSONAL	35000	16.02	1	0.59
1	21	9600	2	5.0	EDUCATION	1000	11.14	0	0.10
2	25	9600	3	1.0	MEDICAL	5500	12.87	1	0.57
3	23	65500	1	4.0	MEDICAL	35000	15.23	1	0.53
4	24	54400	1	8.0	MEDICAL	35000	14.27	1	0.55

In [99]:
```
#check for loan_intent categorical column unique value
df['loan_intent'].unique()
```

Out [99]:
```
array(['PERSONAL', 'EDUCATION', 'MEDICAL', 'VENTURE', 'HOMEIMPROVEMENT',
       'DEBTCONSOLIDATION'], dtype=object)
```

In [100]:
```
#create a dictionary to map the categorical value to numeric value equivalent
loan_intent = {'PERSONAL':1, 'EDUCATION':2, 'MEDICAL':3, 'VENTURE':4, 'HOMEIMPROVEMENT':5,
       'DEBTCONSOLIDATION':6}

## Replace the categorical values in 'loan_intent' with their numeric equivalents
df['loan_intent'] = df['loan_intent'].replace(loan_intent)
```

In [101]:
```
#check if the change has been effected, display dataset
df.head()
```

Out [101]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	1	123.0	1	35000	16.02	1	0.59
1	21	9600	2	5.0	2	1000	11.14	0	0.10
2	25	9600	3	1.0	3	5500	12.87	1	0.57
3	23	65500	1	4.0	3	35000	15.23	1	0.53
4	24	54400	1	8.0	3	35000	14.27	1	0.55

In [102]:
```
#create a dictionary to map the categorical value to numeric value equivalent
cb ={'Y':1, 'N':0}

#Replace the categorical values in 'loan_intent' with their numeric equivalents
df['cb_person_default_on_file'] = df['cb_person_default_on_file'].replace(cb)
```

In [103]:
```
#check if the change has been effected, display dataset
df.head()
```

Out [103]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	1	123.0	1	35000	16.02	1	0.59
1	21	9600	2	5.0	2	1000	11.14	0	0.10
2	25	9600	3	1.0	3	5500	12.87	1	0.57
3	23	65500	1	4.0	3	35000	15.23	1	0.53
4	24	54400	1	8.0	3	35000	14.27	1	0.55

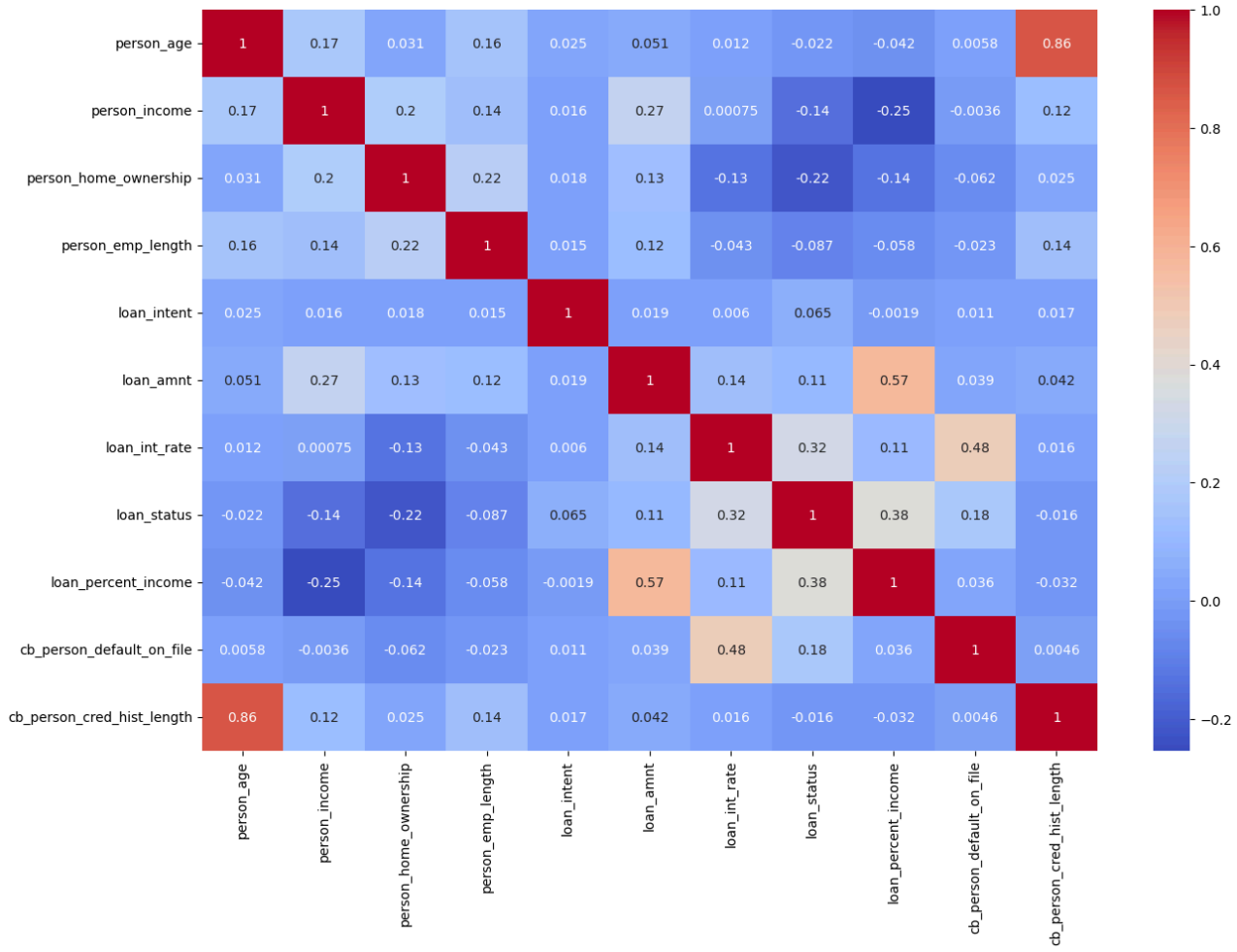
In [104]:
```
df.isnull().sum()
```

Out [104]:
```
person_age                  0
person_income               0
person_home_ownership       0
person_emp_length           0
loan_intent                 0
loan_amnt                   0
loan_int_rate               0
loan_status                 0
loan_percent_income         0
cb_person_default_on_file   0
cb_person_cred_hist_length  0
dtype: int64
```

Correlation Matrix

In [105]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Check for linear relationships using a correlation matrix
plt.figure(figsize=(15,10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```



Save and Reload Transformed Dataset

In [106]:
```
# Save the transformed file as a CSV file
df.to_csv('credit_risk.csv', index=False)
```

In [107]:
```
# Reload the transformed dataset
df = pd.read_csv('credit_risk.csv')
```

In [108]:
```
#Display few rows of transformed dataset
df.head()
```

Out [108]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	1	123.0	1	35000	16.02	1	0.59
1	21	9600	2	5.0	2	1000	11.14	0	0.10
2	25	9600	3	1.0	3	5500	12.87	1	0.57
3	23	65500	1	4.0	3	35000	15.23	1	0.53
4	24	54400	1	8.0	3	35000	14.27	1	0.55

Feature and Target Variables

In [109]:
```
#Define feature and target variable
X = df.drop('loan_status', axis=1).values
y = df['loan_status'].values
```

Train-Test Split

In [110]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=21)
```

Data Standardization

In [111]:
```
# Standardize the feature variables
scale = StandardScaler()
scale_train = scale.fit_transform(X_train)
scale_test = scale.transform(X_test)
```

Initialize Classifiers

In [112]:
```

# Initialize classifiers
from xgboost import XGBClassifier
lg = LogisticRegression()
knn = KNeighborsClassifier()
#xgb = XGBClassifier()
#rnn = RandomForestClassifier()

 #Create a list of classifiers for the Voting Classifier
classifiers = [('Logistic Regression', lg), ('KNeighbor Classifier', knn)]# ('XGBoost', xgb), ('Random Forest (
```

Voting Classifier

In [113]:
```
#instantite Voting Classifier, the algorithm that defines the final predictions from all the models

vc = VotingClassifier(estimators = classifiers, voting='soft')

#fit model
vc.fit(scale_train, y_train)

#make final predictions
vpred = vc.predict(scale_test)

# Calculate accuracy for the individual classifiers
accuracies = {}

for name, model in classifiers:
    # Fit each classifier on the training data
    model.fit(scale_train, y_train)

    # Predict using each individual model
    y_pred = model.predict(scale_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Store the accuracy of each classifier
    accuracies[name] = accuracy

# Calculate the accuracy for the Voting Classifier
voting_accuracy = accuracy_score(y_test, vpred)
accuracies['Voting Classifier'] = voting_accuracy

# Print out the accuracy of all models
print(accuracies)
```

```
{'Logistic Regression': 0.843486266687126, 'KNeighbor Classifier': 0.8635875402792696, 'Voting Classifier': 0.8683443302132883}
```

In [114]:
```
import pandas as pd

# Save to a DataFrame for easier handling
accuracies_df = pd.DataFrame(list(accuracies.items()), columns=['Model', 'Accuracy'])

# Save to a CSV file
accuracies_df.to_csv('model_accuracies.csv', index=False)
```

```
# Optionally, print the accuracies DataFrame
print(accuracies_df)
```

```
                 Model  Accuracy
0   Logistic Regression  0.843486
1  KNeighbor Classifier  0.863588
2     Voting Classifier  0.868344
```

Model Evaluation

In [115]:
```
# Evaluate Voting Classifier accuracy
vaccuracy = accuracy_score(vpred, y_test)

print("Voting Classifier Accuracy:", vaccuracy)
```

```
Voting Classifier Accuracy: 0.8683443302132883
```

In [116]:
```
print(len(y_test))
print(len(vpred))
# Print the classification report for the Voting Classifier
evaluation_report = classification_report(vpred, y_test, output_dict=True)
```

```
6517
6517
```

In [117]:
```
# Convert to DataFrame for saving
report_before_df = pd.DataFrame(evaluation_report).transpose()
report_before_df.to_csv("report.csv")
```

Checle For Class Imbalance

In [119]:
```
# Check for class imbalance
def check_class_imbalance(data, column):
    class_count = data[column].value_counts()
    class_frequencies = class_count / class_count.sum()
    print("Class Frequencies")
    print(class_frequencies)
    # Save the class frequencies to a DataFrame
    class_frequencies_df = pd.DataFrame(class_frequencies).reset_index()
    class_frequencies_df.columns = [column, 'Frequency']  # Rename columns

    # Save to CSV
    class_frequencies_df.to_csv("class_imbalance.csv", index=False)


check_class_imbalance(df, 'loan_status')
```

```
Class Frequencies
0    0.781836
1    0.218164
Name: loan_status, dtype: float64
```

Handle Class Imbalance

In [120]:
```

# Handle class imbalance using SMOTE
from imblearn.over_sampling import SMOTE

smote =SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X,y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=5)

# Standardize the resampled feature variables
X_train_transform = scale.fit_transform(X_train)
X_test_transform = scale.transform(X_test)

# Fit the Voting Classifier model on the resampled data
vc.fit(X_train_transform, y_train)
vpred_2 = vc.predict(X_test_transform)
```

In [121]:
```
print(len(y_test))
print(len(vpred))
# Print the classification report for the Voting Classifier
evaluation_report_after = classification_report(vpred_2, y_test, output_dict=True)
# Convert to DataFrame for saving
report_after_df = pd.DataFrame(evaluation_report_after).transpose()
report_after_df.to_csv("report_after.csv")
```

```
10190
6517
```

In [122]:
```
# Generate confusion matrix and save it
conf_matrix = confusion_matrix(y_test, vpred_2)
conf_matrix_df = pd.DataFrame(conf_matrix, columns=['Predicted 0', 'Predicted 1'], index=['Actual 0', 'Actual 

# Save confusion matrix to CSV
conf_matrix_df.to_csv('confusion_matrix.csv')
```

Cross Validation

In [ ]:
```

```

In [34]:
```

# Perform cross-validation
cv_scores = cross_val_score(vc, X_train_transform, y_train, cv=5)  # You can adjust the number of folds (cv) as

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV score:", cv_scores.mean())
```

```
Cross-validation scores: [0.83918057 0.83216783 0.83388541 0.83155441 0.8340081 ]
Mean CV score: 0.834159264370772
```

In [124]:
```
# Save the cross-validation scores to a DataFrame
cv_scores_df = pd.DataFrame(cv_scores, columns=['CV_Score'])

# Save to CSV
cv_scores_df.to_csv('cross_validation_scores.csv', index=False)
```

In [37]:
```
# Save accuracy result into a dictionary or DataFrame
accuracy_results = pd.DataFrame({
    'Model': ['Voting Classifier'],
    'Accuracy': [vaccuracy]
})

# Save it to CSV
accuracy_results.to_csv('model_accuracy.csv', index=False)
```

In [38]:
```
df = pd.read_csv("model_accuracy.csv")
```

In [39]:
```
df.head()
```

Out [39]:

	Model	Accuracy
0	Voting Classifier	0.868344

In [ ]:
```
import os
import joblib

# Saving the model
model_dir = r'C:\Users\VALERIE KELECHUKWU\Documents\Work_Projects\CProject\RiskApp\RiskAppModel'
joblib.dump(vc, os.path.join(model_dir, 'RiskApp2.6.pkl'))
```

In [ ]:
```
print(f"Model saved at: {os.path.join(model_dir, 'CRisk.pkl')}")
```

In [ ]:
```
import xgboost
print(xgboost.__version__)
```

In [ ]:
```

```

In [ ]:
```

```