SuvoGanguli_Assignment_2.1

May 20, 2024

1 Assignment 2.1: Home Credit Default Risk

The Kaggle dataset on Home Credit Default Risk provides information about the loan applicants' credit bureau data, previous loan records, and other attributes that could influence their ability to repay a loan. The goal is to use this information to predict whether or not an applicant will be able to repay a loan, which is a critical issue for financial services.

[1]:
```
# import libraries

import numpy as np
import pandas as pd
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt


np.random.seed(42)
```

2 Read Dataset

[2]:
```
df = pd.read_csv('data/train_data.csv')

df.columns
```

[2]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       …
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

3 Exploratory Data Analysis

[3]:
```
df.shape
```

[3]:
```
(153755, 122)
```

[4]:
```
df.head()
```

[4]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0      410704       0         Cash loans           F            N
1      381230       0         Cash loans           F            N
2      450177       0         Cash loans           F            Y
3      332445       0         Cash loans           M            Y
4      357429       0         Cash loans           F            Y

  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
0               Y             1          157500.0    900000.0      26446.5
1               Y             1           90000.0    733176.0      21438.0
2               Y             0          189000.0   1795500.0      62541.0
3               N             0          175500.0    494550.0      45490.5
4               Y             0          270000.0   1724688.0      54283.5

   …  FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20 FLAG_DOCUMENT_21  \
0  …                 0                0                0                0
1  …                 0                0                0                0
2  …                 0                0                0                0
3  …                 0                0                0                0
4  …                 0                0                0                0

  AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY  \
0                        0.0                       0.0
1                        0.0                       0.0
2                        0.0                       0.0
3                        0.0                       0.0
4                        0.0                       0.0

   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
2                         0.0                        0.0
3                         0.0                        0.0
4                         0.0                        0.0

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                         0.0
1                        2.0                         1.0
2                        0.0                         0.0
```

```
3                    0.0                    1.0
4                    0.0                    0.0

[5 rows x 122 columns]
```

[5]:
```python
df.describe()
```

[5]:
```
           SK_ID_CURR         TARGET   CNT_CHILDREN  AMT_INCOME_TOTAL  \
count  153755.000000  153755.000000  153755.000000      1.537550e+05
mean   277867.616930       0.080726       0.417398      1.692611e+05
std    102831.742645       0.272414       0.722523      3.180805e+05
min    100004.000000       0.000000       0.000000      2.565000e+04
25%    188542.000000       0.000000       0.000000      1.125000e+05
50%    277749.000000       0.000000       0.000000      1.462500e+05
75%    366718.000000       0.000000       1.000000      2.025000e+05
max    456255.000000       1.000000      19.000000      1.170000e+08

         AMT_CREDIT    AMT_ANNUITY  AMT_GOODS_PRICE  \
count  1.537550e+05  153750.000000     1.536060e+05
mean   5.988824e+05   27083.127015     5.383057e+05
std    4.023748e+05   14468.883776     3.693544e+05
min    4.500000e+04    1615.500000     4.500000e+04
25%    2.700000e+05   16506.000000     2.385000e+05
50%    5.135310e+05   24903.000000     4.500000e+05
75%    8.086500e+05   34587.000000     6.795000e+05
max    4.050000e+06  230161.500000     4.050000e+06

       REGION_POPULATION_RELATIVE     DAYS_BIRTH  DAYS_EMPLOYED  …  \
count               153755.000000  153755.000000  153755.000000  …
mean                     0.020813  -16025.981438   63742.602751  …
std                      0.013796    4363.552861  141204.275368  …
min                      0.000290  -25201.000000  -17583.000000  …
25%                      0.010006  -19662.000000   -2746.000000  …
50%                      0.018850  -15725.000000   -1211.000000  …
75%                      0.028663  -12399.000000    -290.000000  …
max                      0.072508   -7678.000000  365243.000000  …

       FLAG_DOCUMENT_18  FLAG_DOCUMENT_19  FLAG_DOCUMENT_20  FLAG_DOCUMENT_21  \
count     153755.000000     153755.000000     153755.000000     153755.000000
mean           0.007909          0.000650          0.000501          0.000416
std            0.088579          0.025494          0.022373          0.020398
min            0.000000          0.000000          0.000000          0.000000
25%            0.000000          0.000000          0.000000          0.000000
50%            0.000000          0.000000          0.000000          0.000000
75%            0.000000          0.000000          0.000000          0.000000
max            1.000000          1.000000          1.000000          1.000000
```

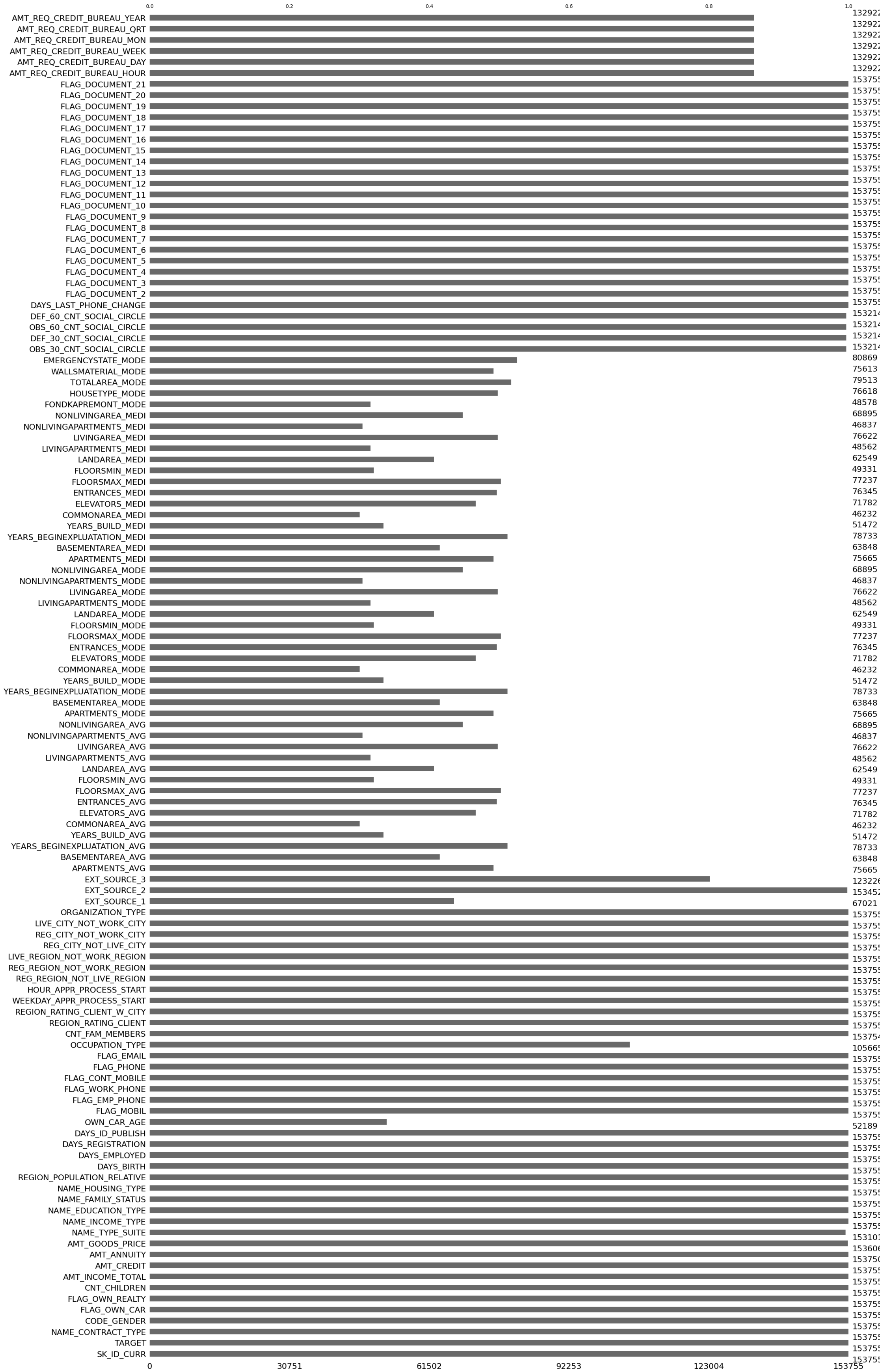
```
       AMT_REQ_CREDIT_BUREAU_HOUR  AMT_REQ_CREDIT_BUREAU_DAY  \
count               132922.000000              132922.000000
mean                     0.006417                   0.006854
std                      0.084608                   0.110151
min                      0.000000                   0.000000
25%                      0.000000                   0.000000
50%                      0.000000                   0.000000
75%                      0.000000                   0.000000
max                      4.000000                   9.000000

       AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
count               132922.000000              132922.000000
mean                     0.034012                   0.265547
std                      0.201581                   0.907185
min                      0.000000                   0.000000
25%                      0.000000                   0.000000
50%                      0.000000                   0.000000
75%                      0.000000                   0.000000
max                      8.000000                  27.000000

       AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
count              132922.000000               132922.000000
mean                    0.267555                    1.901777
std                     0.941286                    1.873638
min                     0.000000                    0.000000
25%                     0.000000                    0.000000
50%                     0.000000                    1.000000
75%                     0.000000                    3.000000
max                   261.000000                   25.000000

[8 rows x 106 columns]
```

3.1 Inspect dataset for missing data

Drop columns and rows with missing data

[ ]:
```
msno.bar(df)
```

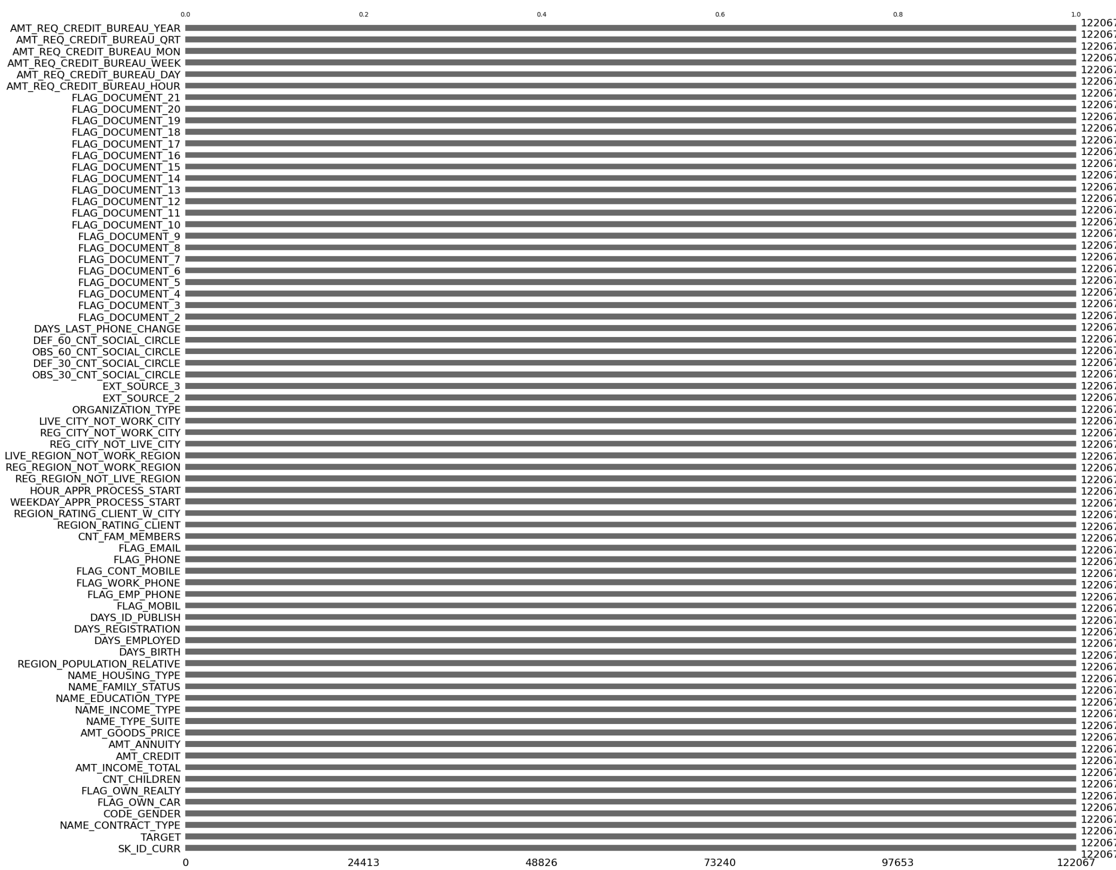
[ ]: <Axes: >



AMT_REQ_CREDIT_BUREAU_YEAR
AMT_REQ_CREDIT_BUREAU_QRT
AMT_REQ_CREDIT_BUREAU_MON
AMT_REQ_CREDIT_BUREAU_WEEK
AMT_REQ_CREDIT_BUREAU_DAY
AMT_REQ_CREDIT_BUREAU_HOUR
FLAG_DOCUMENT_21
FLAG_DOCUMENT_20
FLAG_DOCUMENT_19
FLAG_DOCUMENT_18
FLAG_DOCUMENT_17
FLAG_DOCUMENT_16
FLAG_DOCUMENT_15
FLAG_DOCUMENT_14
FLAG_DOCUMENT_13
FLAG_DOCUMENT_12
FLAG_DOCUMENT_11
FLAG_DOCUMENT_10
FLAG_DOCUMENT_9
FLAG_DOCUMENT_8
FLAG_DOCUMENT_7
FLAG_DOCUMENT_6
FLAG_DOCUMENT_5
FLAG_DOCUMENT_4
FLAG_DOCUMENT_3
FLAG_DOCUMENT_2
DAYS_LAST_PHONE_CHANGE
DEF_60_CNT_SOCIAL_CIRCLE
OBS_60_CNT_SOCIAL_CIRCLE
DEF_30_CNT_SOCIAL_CIRCLE
OBS_30_CNT_SOCIAL_CIRCLE
EMERGENCYSTATE_MODE
WALLSMATERIAL_MODE
TOTALAREA_MODE
HOUSETYPE_MODE
FONDKAPREMONT_MODE
NONLIVINGAREA_MEDI
NONLIVINGAPARTMENTS_MEDI
LIVINGAREA_MEDI
LIVINGAPARTMENTS_MEDI
LANDAREA_MEDI
FLOORSMIN_MEDI
FLOORSMAX_MEDI
ENTRANCES_MEDI
ELEVATORS_MEDI
COMMONAREA_MEDI
YEARS_BUILD_MEDI
YEARS_BEGINEXPLUATATION_MEDI
BASEMENTAREA_MEDI
APARTMENTS_MEDI
NONLIVINGAREA_MODE
NONLIVINGAPARTMENTS_MODE
LIVINGAREA_MODE
LIVINGAPARTMENTS_MODE
LANDAREA_MODE
FLOORSMIN_MODE
FLOORSMAX_MODE
ENTRANCES_MODE
ELEVATORS_MODE
COMMONAREA_MODE
YEARS_BUILD_MODE
YEARS_BEGINEXPLUATATION_MODE
BASEMENTAREA_MODE
APARTMENTS_MODE
NONLIVINGAREA_AVG
NONLIVINGAPARTMENTS_AVG
LIVINGAREA_AVG
LIVINGAPARTMENTS_AVG
LANDAREA_AVG
FLOORSMIN_AVG
FLOORSMAX_AVG
ENTRANCES_AVG
ELEVATORS_AVG
COMMONAREA_AVG
YEARS_BUILD_AVG
YEARS_BEGINEXPLUATATION_AVG
BASEMENTAREA_AVG
APARTMENTS_AVG
EXT_SOURCE_3
EXT_SOURCE_2
EXT_SOURCE_1
ORGANIZATION_TYPE
LIVE_CITY_NOT_WORK_CITY
REG_CITY_NOT_WORK_CITY
REG_CITY_NOT_LIVE_CITY
LIVE_REGION_NOT_WORK_REGION
REG_REGION_NOT_WORK_REGION
REG_REGION_NOT_LIVE_REGION
HOUR_APPR_PROCESS_START
WEEKDAY_APPR_PROCESS_START
REGION_RATING_CLIENT_W_CITY
REGION_RATING_CLIENT
CNT_FAM_MEMBERS
OCCUPATION_TYPE
FLAG_EMAIL
FLAG_PHONE
FLAG_CONT_MOBILE
FLAG_WORK_PHONE
FLAG_EMP_PHONE
FLAG_MOBIL
OWN_CAR_AGE
DAYS_ID_PUBLISH
DAYS_REGISTRATION
DAYS_EMPLOYED
DAYS_BIRTH
REGION_POPULATION_RELATIVE
NAME_HOUSING_TYPE
NAME_FAMILY_STATUS
NAME_EDUCATION_TYPE
NAME_INCOME_TYPE
NAME_TYPE_SUITE
AMT_GOODS_PRICE
AMT_ANNUITY
AMT_CREDIT
AMT_INCOME_TOTAL
CNT_CHILDREN
FLAG_OWN_REALTY
FLAG_OWN_CAR
CODE_GENDER
NAME_CONTRACT_TYPE
TARGET
SK_ID_CURR
132922
132922
132922
132922
132922
132922
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153214
153214
153214
153214
80869
75613
79513
76618
48578
68895
46837
76622
48562
62549
49331
77237
76345
71782
46232
51472
78733
63848
75665
68895
46837
76622
48562
62549
49331
77237
76345
71782
46232
51472
78733
63848
75665
68895
46837
76622
48562
62549
49331
77237
76345
71782
46232
51472
78733
63848
75665
123226
153452
67021
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153755
153754
105665
153755
153755
153755
153755
153755
153755
52189
153755
153755
153755
153755
153755
153755
153755
153755
153755
153101
153606
153750
153755
153755
153755
153755
153755
153755
153755
153755
153755
0.0
0.2
0.4
0.6
0.8
1.0
0
30751
61502
92253
123004
153755

```
[ ]: # Calculate the percentage of missing values for each column
missing_percent = df.isnull().mean() * 100

# Filter out columns with more than 50% missing values
columns_to_drop = missing_percent[missing_percent > 25].index
df.drop(columns=columns_to_drop, inplace=True)

df = df.dropna()

# Display the resulting DataFrame
msno.bar(df)
```

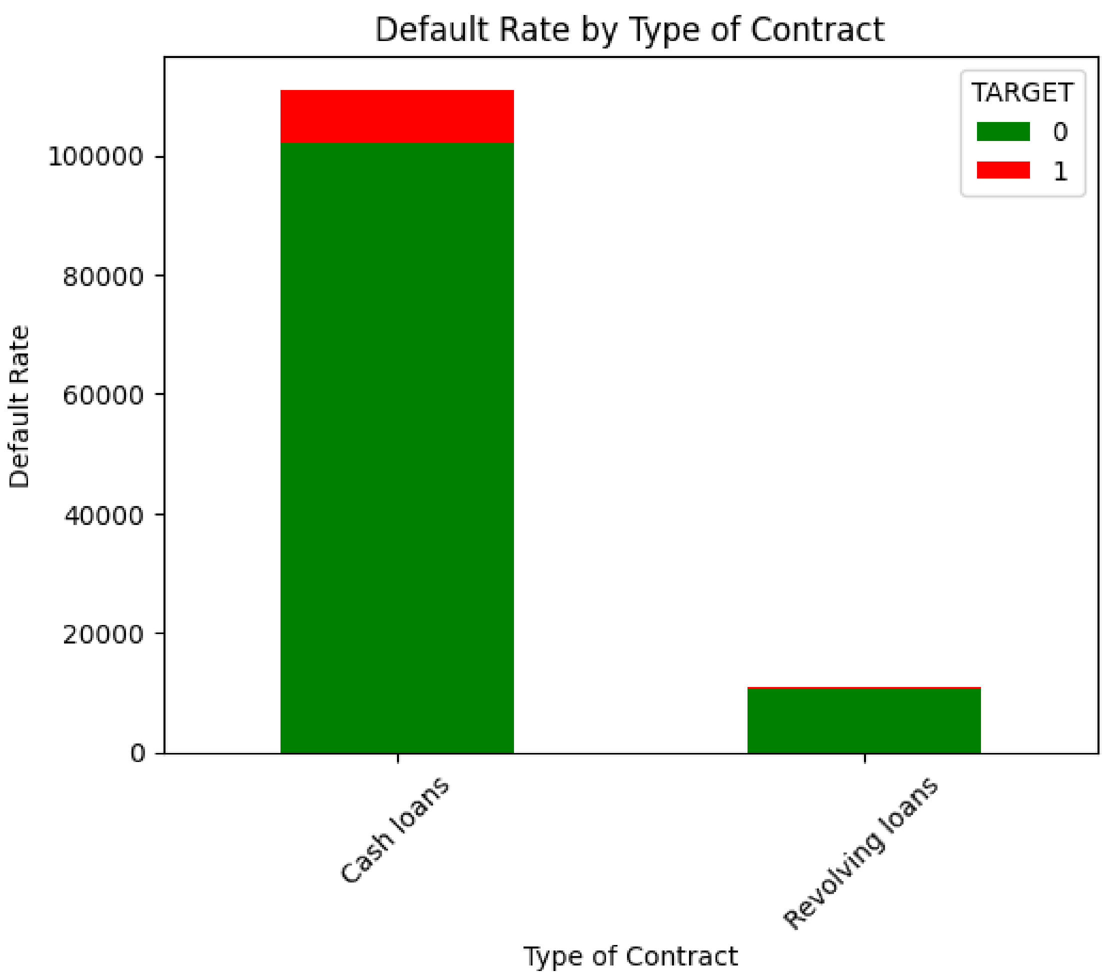
```
[ ]: <Axes: >
```



4 Exploratory Data Plots

The bar plot shows that only Cash Loans for defaulted

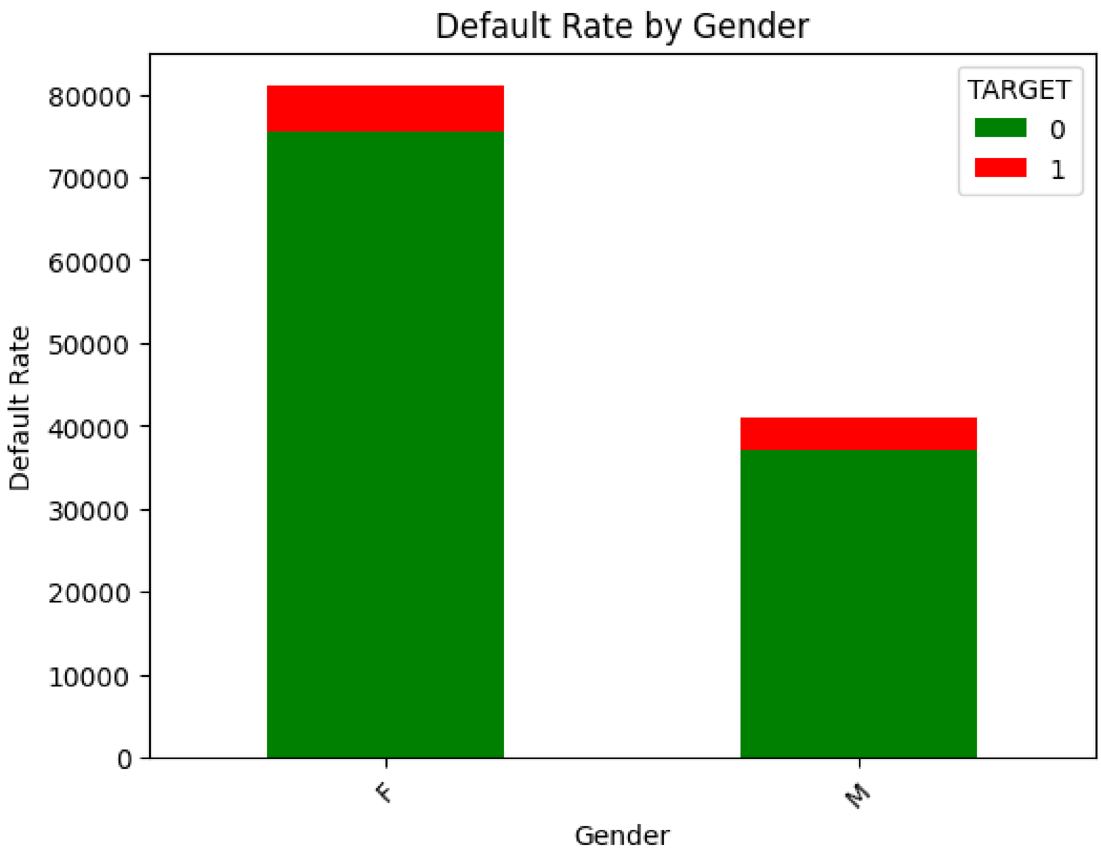
[8]:
```
count_data = df.groupby(['NAME_CONTRACT_TYPE', 'TARGET']).size().
    ↪unstack(fill_value=0)

count_data.plot(kind='bar', stacked=True, color=['green', 'red'])
plt.xlabel('Type of Contract')
plt.ylabel('Default Rate')
plt.title('Default Rate by Type of Contract')
plt.xticks(rotation=45)
plt.show()
```



The barplot below shows that more females have applied for loan, and the percentage of loan default for both gender is or the same order

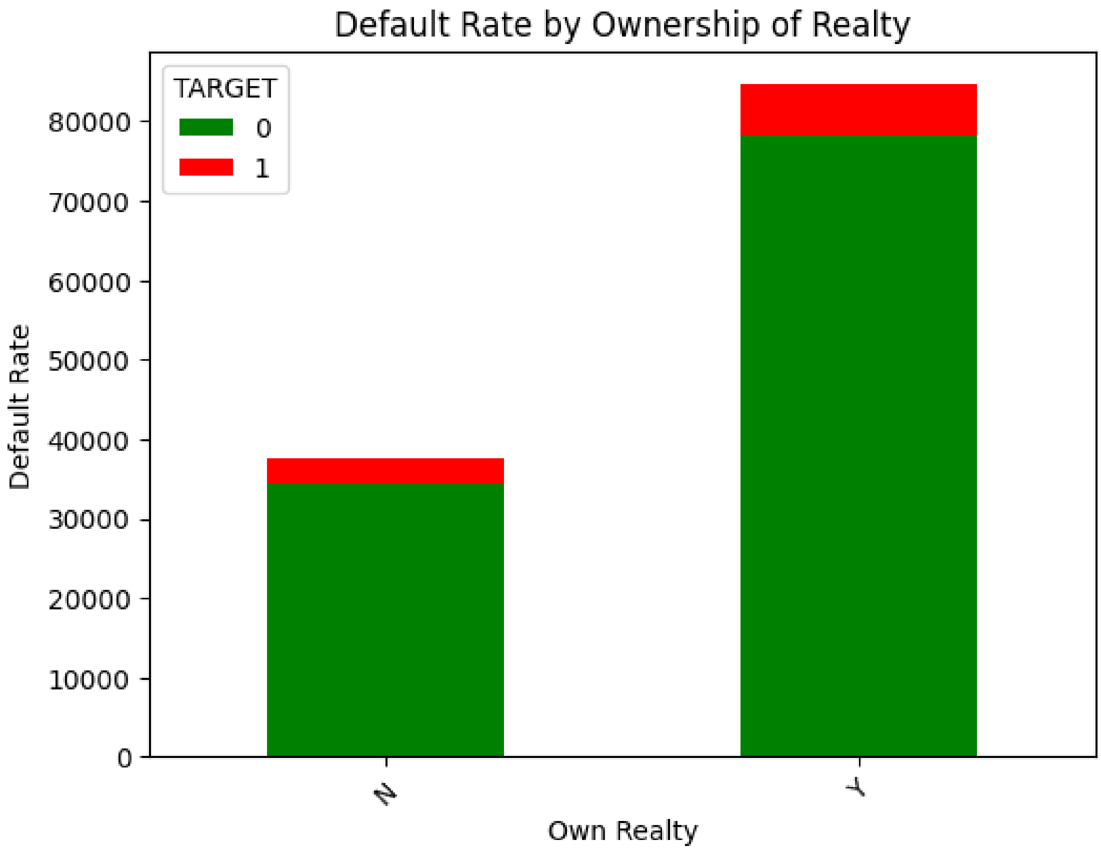
[9]:
```
count_data = df.groupby(['CODE_GENDER', 'TARGET']).size().unstack(fill_value=0)

count_data.plot(kind='bar', stacked=True, color=['green', 'red'])
plt.xlabel('Gender')
```

```
plt.ylabel('Default Rate')
plt.title('Default Rate by Gender')
plt.xticks(rotation=45)
plt.show()
```



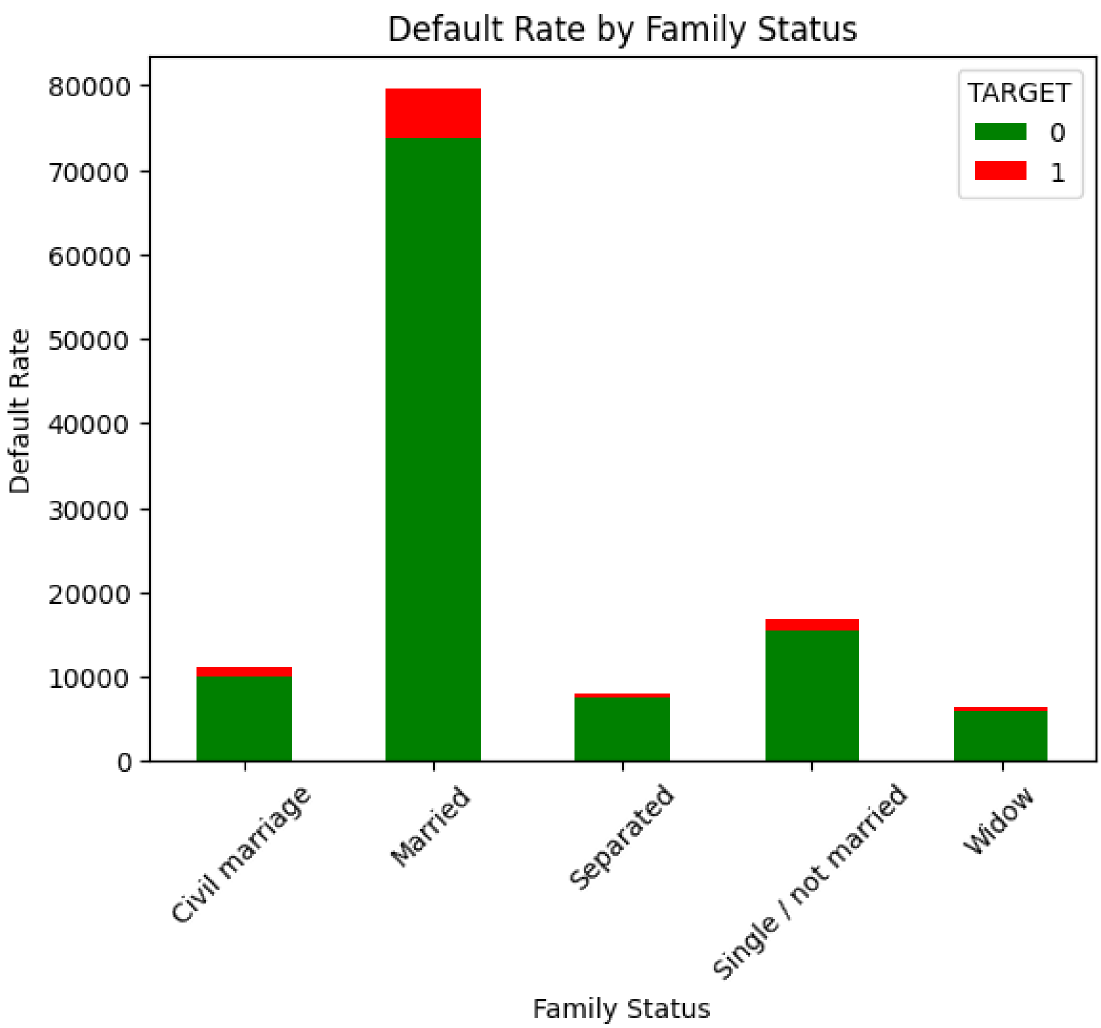
The barplot shows that the percentage of loan default is similar whether the person had previous ownership of realty or not.

[10]:
```
count_data = df.groupby(['FLAG_OWN_REALTY', 'TARGET']).size().
 ↪unstack(fill_value=0)

count_data.plot(kind='bar', stacked=True, color=['green', 'red'])
plt.xlabel('Own Realty')
plt.ylabel('Default Rate')
plt.title('Default Rate by Ownership of Realty')
plt.xticks(rotation=45)
plt.show()
```



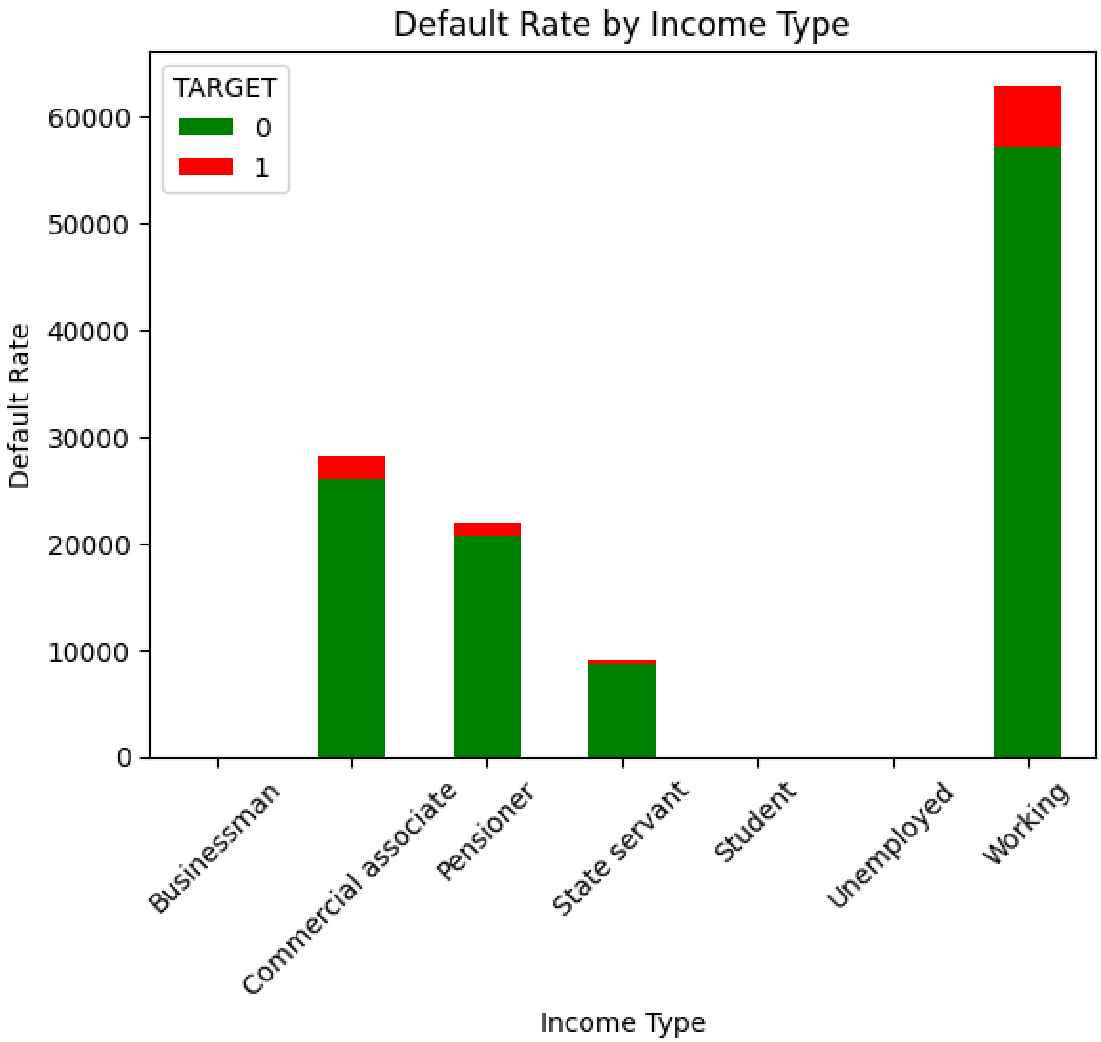
The barplot below shows the relationship between loan default status and family status

[11]:
```
count_data = df.groupby(['NAME_FAMILY_STATUS', 'TARGET']).size().
  ↪unstack(fill_value=0)

count_data.plot(kind='bar', stacked=True, color=['green', 'red'])
plt.xlabel('Family Status')
plt.ylabel('Default Rate')
plt.title('Default Rate by Family Status')
plt.xticks(rotation=45)
plt.show()
```



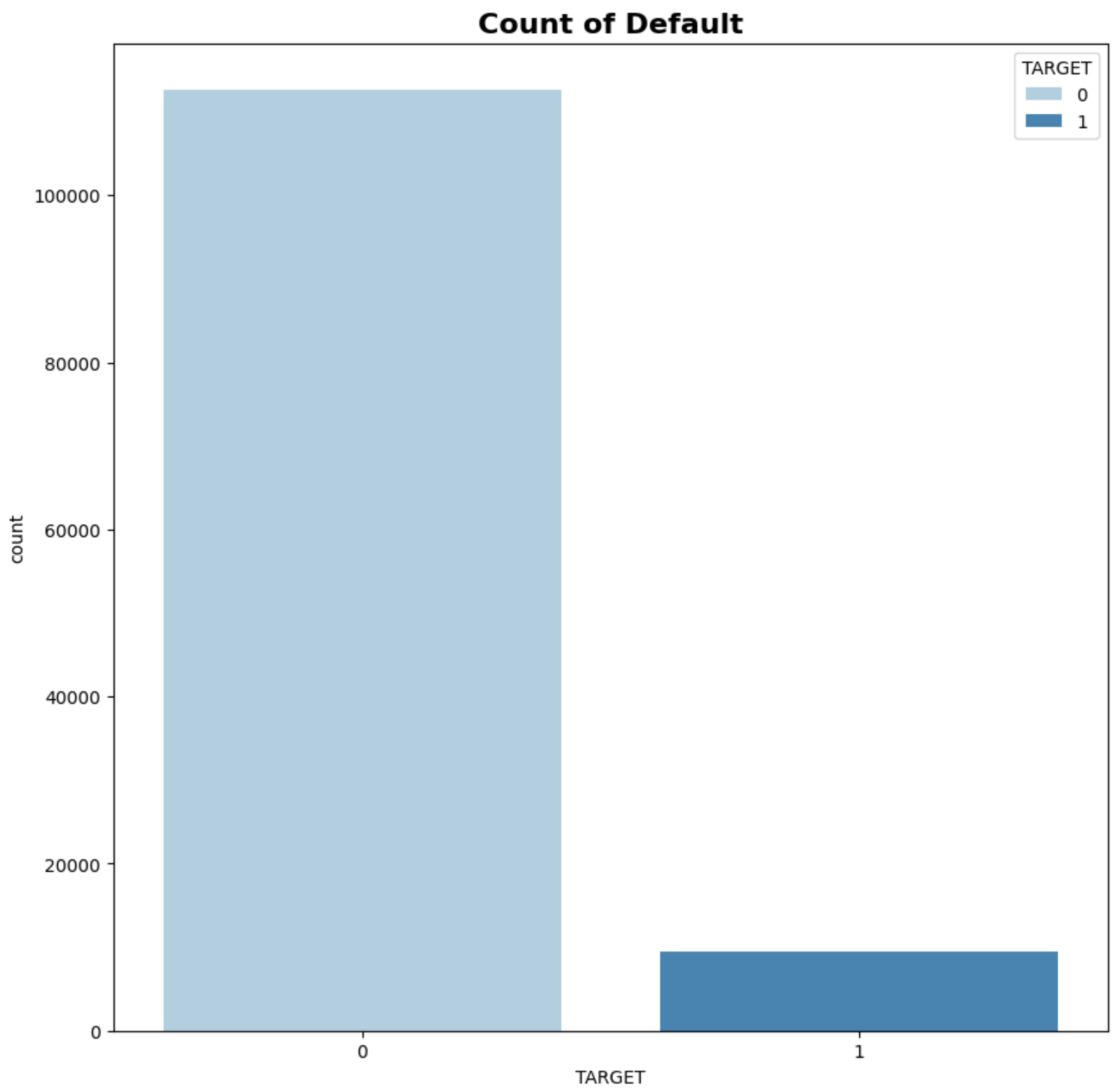
The barplot below shows the relationship between load default status and income type

[12]:
```
count_data = df.groupby(['NAME_INCOME_TYPE', 'TARGET']).size().
  ↪unstack(fill_value=0)

count_data.plot(kind='bar', stacked=True, color=['green', 'red'])
plt.xlabel('Income Type')
plt.ylabel('Default Rate')
plt.title('Default Rate by Income Type')
plt.xticks(rotation=45)
plt.show()
```



[57]:
```
plt.figure(figsize=(10,10))
plt.title("Count of Default", fontweight = 'bold', fontsize = 16)
sns.countplot(x ='TARGET',data=df, hue='TARGET',palette="Blues")
```

[57]: <Axes: title={'center': 'Count of Default'}, xlabel='TARGET', ylabel='count'>



5 Prediction

5.1 Decision Tree

[13]:
```
from sklearn.model_selection import train_test_split

# Identify non-numeric columns
non_numeric_columns = df.select_dtypes(exclude=[np.number]).columns

# Convert non-numeric columns to one-hot encoding
df_encoded = pd.get_dummies(df, columns=non_numeric_columns)

X = df_encoded.drop('TARGET', axis=1)
```

```
y = df_encoded['TARGET']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Further split the training set into training and validation sets (80% train,␣
↪20% validation of the training data)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪25, random_state=42)
```

[55]:
```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import SMOTE

# Apply SMOTE to address class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Initialize the model
model = XGBClassifier(random_state=42, use_label_encoder=False,␣
↪eval_metric='logloss')

print("Performing cross-validation with all features...")
cv_scores_all = cross_val_score(model, X_train_resampled_scaled,␣
↪y_train_resampled, cv=5, scoring='f1_weighted', verbose=1, n_jobs=-1)
print(f'Cross-validation F1 scores with all features: {cv_scores_all}')
print(f'Average cross-validation F1 score with all features: {cv_scores_all.
↪mean()}')

# Train the model on the training set
model.fit(X_train_resampled, y_train_resampled)

# Predict on the validation set
print('')
print("Predicting on the validation set...")
y_val_pred = model.predict(X_val)

# Evaluate the predictions
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred)

print(f'Validation Accuracy: {val_accuracy}')
print('Validation Classification Report:')
print(val_report)

# Predict on the test set
```

```python
print("Predicting on the test set...")
y_test_pred = model.predict(X_test)

# Evaluate the predictions
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred, zero_division=1)

print(f'Test Accuracy: {test_accuracy}')
print('Test Classification Report:')
print(test_report)
```

```
Performing cross-validation with all features…

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    7.6s remaining:   11.4s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    7.7s finished

Cross-validation F1 scores with all features: [0.78080702 0.99607853 0.9953016
0.9953756  0.99637449]
Average cross-validation F1 score with all features: 0.9527874496368529

Predicting on the validation set…
Validation Accuracy: 0.9189809125911362
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     22469
           1       0.40      0.03      0.06      1945

    accuracy                           0.92     24414
   macro avg       0.66      0.51      0.51     24414
weighted avg       0.88      0.92      0.89     24414

Predicting on the test set…
Test Accuracy: 0.9211108380437454
Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     22527
           1       0.38      0.03      0.06      1887

    accuracy                           0.92     24414
   macro avg       0.65      0.51      0.51     24414
weighted avg       0.88      0.92      0.89     24414
```

Note: with all the features selected, the cross-validation score is 0.952.

6 Feature Selection

6.1 Approach 1: Linear Discriminant Analysis

[18]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
```

[60]:
```
# Apply SMOTE to address class imbalance
print("Applying SMOTE...")
smote = SMOTE(random_state=42, n_jobs=-1)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Verify the shapes of the resampled data
print(f'Resampled training set shape: {X_train_resampled.shape}')
print(f'Resampled training labels shape: {y_train_resampled.shape}')

# Scale the data before applying LDA
print("Scaling data...")
scaler = StandardScaler()
X_train_resampled_scaled = scaler.fit_transform(X_train_resampled)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Apply LDA for feature selection
print("Applying LDA for feature selection...")
lda = LDA()
X_train_lda = lda.fit_transform(X_train_resampled_scaled, y_train_resampled)
X_val_lda = lda.transform(X_val_scaled)
X_test_lda = lda.transform(X_test_scaled)

# Verify the shapes of the LDA-transformed data
print(f'LDA-transformed training set shape: {X_train_lda.shape}')
print(f'LDA-transformed validation set shape: {X_val_lda.shape}')
print(f'LDA-transformed test set shape: {X_test_lda.shape}')
```

```
Applying SMOTE…
```

```
/opt/anaconda3/envs/tf2/lib/python3.8/site-
packages/imblearn/over_sampling/_smote/base.py:363: FutureWarning: The parameter
`n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass
an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

```
Resampled training set shape: (135156, 163)
Resampled training labels shape: (135156,)
Scaling data…
Applying LDA for feature selection…
```

```
LDA-transformed training set shape: (135156, 1)
LDA-transformed validation set shape: (24414, 1)
LDA-transformed test set shape: (24414, 1)
```

[42]:

```
# Initialize the model
model = XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='logloss', verbosity=1)

# Perform cross-validation on the training data
print("Performing cross-validation...")
cv_scores = cross_val_score(model, X_train_lda, y_train_resampled, cv=5,␣
 ↪verbose=1, n_jobs=-1)
print(f'Cross-validation scores: {cv_scores}')
print(f'Average cross-validation score: {cv_scores.mean()}')

# Train the model on the resampled and transformed training set
print("Training the model...")
model.fit(X_train_lda, y_train_resampled, verbose=True)

# Predict on the validation set
print("Predicting on the validation set...")
y_val_pred = model.predict(X_val_lda)

# Evaluate the predictions
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred, zero_division=1)

print(f'Validation Accuracy: {val_accuracy}')
print('Validation Classification Report:')
print(val_report)

# Predict on the test set
print("Predicting on the test set...")
y_test_pred = model.predict(X_test_lda)

# Evaluate the predictions
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred, zero_division=1)

print(f'Test Accuracy: {test_accuracy}')
print('Test Classification Report:')
print(test_report)
```

```
Performing cross-validation…
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    3.3s remaining:    4.9s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    3.3s finished
```

```
Cross-validation scores: [0.79202427 0.99652251 0.99711442 0.99696645
0.99711442]
Average cross-validation score: 0.9559484146034736
Training the model…
Predicting on the validation set…
Validation Accuracy: 0.919062832800852
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     22469
           1       0.22      0.01      0.01      1945

    accuracy                           0.92     24414
   macro avg       0.57      0.50      0.48     24414
weighted avg       0.86      0.92      0.88     24414

Predicting on the test set…
Test Accuracy: 0.9219710002457606
Test Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     22527
           1       0.30      0.01      0.01      1887

    accuracy                           0.92     24414
   macro avg       0.61      0.50      0.49     24414
weighted avg       0.87      0.92      0.89     24414
```

6.2 Analysis of performance with LDA

Note that with LDA, the accuracy has increased from 0.918 to 0.919. However, the f1-score decreaed from 0.06 to 0.01. The cross-validation score has improved to 0.956 compared to the case with all the features selected.

Hence, we can conclude that the LDA has improved the performance.

6.3 Possible reasons for reduction in f1-score

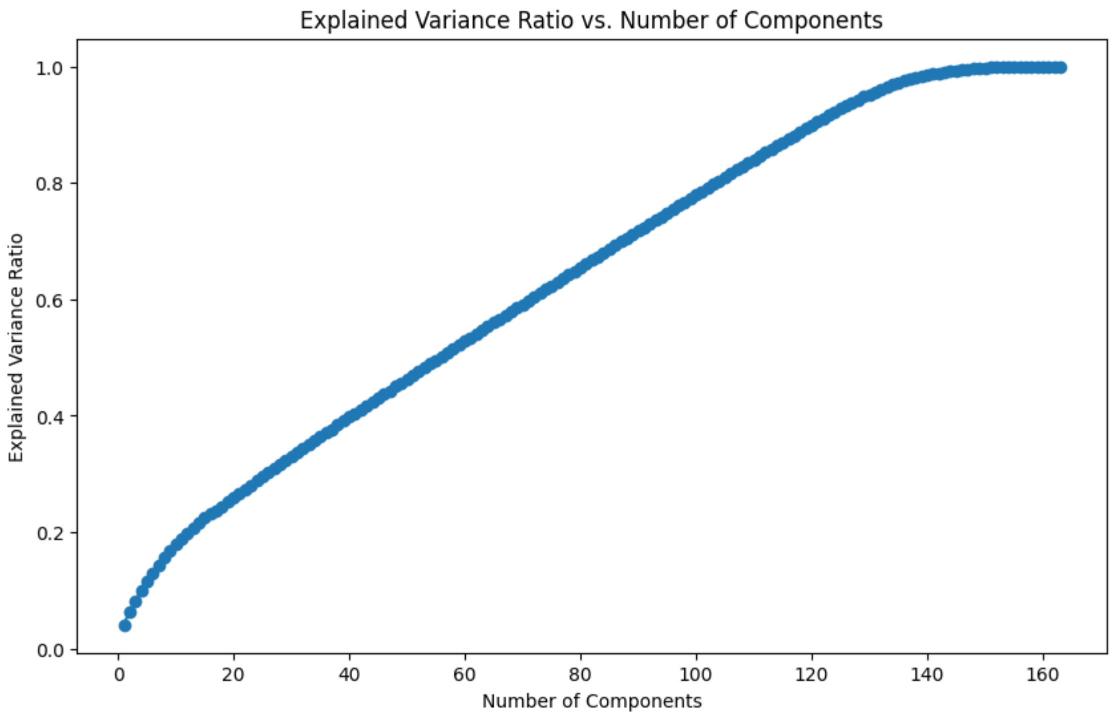
- LDA assumes that the data for each class is normally distributed. It also assumes that all classes have the same covariance matrix. If these assumptions do not hold, the LDA projection might not be optimal, and can lead to reduction in f1-score.
- While LDA reduces the dimensionality of the data, this lead to loss of information, which can in turn lead to reduced f1-score

6.4 Approach 2: Principal Component Analysis

[52]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Scale the data before applying PCA
scaler = StandardScaler()
X_train_resampled_scaled = scaler.fit_transform(X_train_resampled)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Find the optimal number of components for PCA
explained_variance_ratios = []
components_range = range(1, X_train_resampled_scaled.shape[1] + 1)

for n_components in components_range:
    if (n_components % 20) == 0:
        print(n_components)
    pca = PCA(n_components=n_components)
    pca.fit(X_train_resampled_scaled)
    explained_variance_ratios.append(np.sum(pca.explained_variance_ratio_))

# Plot the explained variance ratios to find the elbow point
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(components_range, explained_variance_ratios, marker='o',␣
 ↪linestyle='--')
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Components')
plt.show()

# Choose the number of components that capture the desired amount of variance␣
 ↪(e.g., 95%)
optimal_n_components = next(x[0] for x in enumerate(explained_variance_ratios)␣
 ↪if x[1] >= 0.95)
print(f'Optimal number of components: {optimal_n_components}')

# Apply PCA with the optimal number of components
pca = PCA(n_components=optimal_n_components)
X_train_pca = pca.fit_transform(X_train_resampled_scaled)
X_val_pca = pca.transform(X_val_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Verify the shapes of the PCA-transformed data
print(f'PCA-transformed training set shape: {X_train_pca.shape}')
```

```python
print(f'PCA-transformed validation set shape: {X_val_pca.shape}')
print(f'PCA-transformed test set shape: {X_test_pca.shape}')
```

```
20
40
60
80
100
120
140
160
```




```
Optimal number of components: 129
PCA-transformed training set shape: (135156, 129)
PCA-transformed validation set shape: (24414, 129)
PCA-transformed test set shape: (24414, 129)
```

[56]:
```python
from sklearn.model_selection import cross_val_score

# Initialize the model
model = XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='logloss')

# Perform cross-validation on the training data
```

```
cv_scores = cross_val_score(model, X_train_pca, y_train_resampled, cv=5,␣
 ↪verbose=1, n_jobs=-1)
print(f'Cross-validation scores: {cv_scores}')
print(f'Average cross-validation score: {np.mean(cv_scores)}')

# Train the model on the resampled and transformed training set
model.fit(X_train_pca, y_train_resampled)

# Predict on the validation set
print('')
print("Predicting on the validation set...")
y_val_pred = model.predict(X_val_pca)

# Evaluate the predictions
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report = classification_report(y_val, y_val_pred, zero_division=1)

print(f'Validation Accuracy: {val_accuracy}')
print('Validation Classification Report:')
print(val_report)

# Predict on the test set
print("Predicting on the test set...")
y_test_pred = model.predict(X_test_pca)

# Evaluate the predictions
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred, zero_division=1)

print(f'Test Accuracy: {test_accuracy}')
print('Test Classification Report:')
print(test_report)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:   18.2s remaining:   27.3s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   20.1s finished
```

```
Cross-validation scores: [0.79061853 0.97787725 0.97732233 0.97532463
0.97839518]
Average cross-validation score: 0.939907583008204

Predicting on the validation set…
Validation Accuracy: 0.8982960596379127
Validation Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     22469
           1       0.20      0.09      0.13      1945
```

```
    accuracy                           0.90     24414
   macro avg       0.56      0.53      0.54     24414
weighted avg       0.87      0.90      0.88     24414

Predicting on the test set…
Test Accuracy: 0.9034160727451462
Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     22527
           1       0.24      0.11      0.15      1887

    accuracy                           0.90     24414
   macro avg       0.58      0.54      0.55     24414
weighted avg       0.88      0.90      0.89     24414
```

6.5 Analysis of performance with PCA

With PCA, the accuracy slightly decreased to 0.90, but the f1-score for '1' increased from 0.06 to 0.14. The cross-validation score has reduced to 0.939 from 0.952 with all the features.

Hence, it is not advisable to use PCA.

6.6 Possible reasons for decreaed accuracy:

- While PCA reduces the dimensionality of the data by projecting it into a smaller number of components, it does lead to loss of imformation. This may lead to reduction in accuracy
- PCA in based on using linear transformation and may not capture the nonlinear relationships between features. This can also lead to reduced accuracy.

6.7 Approach 3: Factor Analysis

[45]:
```
from sklearn.decomposition import FactorAnalysis

# Apply SMOTE to address class imbalance
print("Applying SMOTE...")
smote = SMOTE(random_state=42, n_jobs=-1)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Verify the shapes of the resampled data
print(f'Resampled training set shape: {X_train_resampled.shape}')
print(f'Resampled training labels shape: {y_train_resampled.shape}')

# Scale the data before applying Factor Analysis
print("Scaling data...")
scaler = StandardScaler()
```

```
X_train_resampled_scaled = scaler.fit_transform(X_train_resampled)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Apply Factor Analysis for feature reduction
print("Applying Factor Analysis for feature reduction...")
fa = FactorAnalysis(n_components=None)
X_train_fa = fa.fit_transform(X_train_resampled_scaled)
X_val_fa = fa.transform(X_val_scaled)
X_test_fa = fa.transform(X_test_scaled)

# Verify the shapes of the FA-transformed data
print(f'FA-transformed training set shape: {X_train_fa.shape}')
print(f'FA-transformed validation set shape: {X_val_fa.shape}')
print(f'FA-transformed test set shape: {X_test_fa.shape}')
```

```
Applying SMOTE…
```

```
/opt/anaconda3/envs/tf2/lib/python3.8/site-
packages/imblearn/over_sampling/_smote/base.py:363: FutureWarning: The parameter
`n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass
an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

```
Resampled training set shape: (135156, 163)
Resampled training labels shape: (135156,)
Scaling data…
Applying Factor Analysis for feature reduction…
FA-transformed training set shape: (135156, 163)
FA-transformed validation set shape: (24414, 163)
FA-transformed test set shape: (24414, 163)
```

[53]:
```
# Initialize the model
model = XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='logloss', verbosity=1)

# Perform cross-validation on the training data
print("Performing cross-validation with Factor Analysis...")
cv_scores_fa = cross_val_score(model, X_train_fa, y_train_resampled, cv=5,␣
 ↪scoring='f1_weighted', verbose=1, n_jobs=-1)
print(f'Average cross-validation F1 score with Factor Analysis: {cv_scores_fa.
 ↪mean()}')

# Train the model on the FA-transformed training set
print("Training the model with Factor Analysis...")
model.fit(X_train_fa, y_train_resampled, verbose=True)

# Predict on the validation set with Factor Analysis
```

```python
print("Predicting on the validation set with Factor Analysis...")
y_val_pred_fa = model.predict(X_val_fa)

# Evaluate the predictions with Factor Analysis
val_accuracy = accuracy_score(y_val, y_val_pred)
val_report_fa = classification_report(y_val, y_val_pred_fa, zero_division=1)

print(f'Validation Accuracy: {val_accuracy}')
print('Validation Classification Report with Factor Analysis:')
print(val_report_fa)

# Predict on the test set
print("Predicting on the test set...")
y_test_pred = model.predict(X_test_fa)

# Evaluate the predictions
test_accuracy = accuracy_score(y_test, y_test_pred)
test_report = classification_report(y_test, y_test_pred, zero_division=1)

print(f'Test Accuracy: {test_accuracy}')
print('Test Classification Report:')
print(test_report)
```

```
Performing cross-validation with Factor Analysis…

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:   27.1s remaining:   40.6s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   27.5s finished

Average cross-validation F1 score with Factor Analysis: 0.9480905261980699
Training the model with Factor Analysis…
Predicting on the validation set with Factor Analysis…
Validation Accuracy: 0.8995658228885066
Validation Classification Report with Factor Analysis:
              precision    recall  f1-score   support

           0       0.92      0.99      0.96     22469
           1       0.30      0.06      0.09      1945

    accuracy                           0.91     24414
   macro avg       0.61      0.52      0.52     24414
weighted avg       0.87      0.91      0.89     24414

Predicting on the test set…
Test Accuracy: 0.9148849021053493
Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96     22527
```

```
           1       0.28      0.06      0.10      1887

    accuracy                           0.91     24414
   macro avg       0.60      0.52      0.53     24414
weighted avg       0.88      0.91      0.89     24414
```

6.8 Analysis of performance with Factor Analysis

With Factor Analysis, the accuracy is almost the same (0.919) as that of the original prediction model (without feature reduction). However, the f1-score for '1' has increaed from 0.06 to 0.09. The cross-validation score has slightly reduced to 0.948.

However, the number of features selected has not reduced.

6.9 Conclusion

Given the three feature selection techniques:

With LDA, the number of features selected is 1 and the cross-validation score has improved.

With PCA, the number of features selected is 129 and the cross-validation score has reduced.

With Factor Analysis, the cross-validation score has reduced. Also, there no reduction in the number of features - which remains at 163. Hence, we reject this method.

Given above, the LDA method is best suited for feature reduction.

6.9.1 Reason why LDA performs better than PCA

- LDA: Maximizes class separability. PCA: Maximizes data variance.
- LDA: Supervised (uses class labels). PCA: Unsupervised (does not use class labels).

[ ]:

```

```