DataExploration

April 27, 2024

1 Einleitung

Das vorliegende Projekt zielt darauf ab, ein binäres Klassifikationsproblem im Bereich des Kreditwesens zu lösen. Dabei wird mittels Machine Learning Techniken versucht, anhand der Informationen von Kreditnehmern präzise Vorhersagen über die Zusage oder Absage von Krediten zu treffen, um Risiken der Kreditvergbern zu reduzieren.

Der Bericht wird den gesamten Prozess des Projekts detailliert darlegen, beginnend mit der Vorbereitung und Erkundung der Daten bis hin zur Entwicklung und Auswertung des Modells. Es werden die angewandten Methoden, die erzielten Ergebnisse und die gezogenen Schlussfolgerungen ausführlich beschrieben, um ein umfassendes Verständnis für den Ansatz und die erreichten Ergebnisse zu vermitteln.

Link für den Gitlab-Ordner: https://git.dhbw-stuttgart.de/wi22077/dataexploration

2 Installation

Um die erforderlichen Bibliotheken zu installieren, bitte den folgenden Befehl ausführen:

```
pip install -r requirements.txt
```

[98]:
```
import pandas as pd
import seaborn as sns
import random
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪roc_auc_score,roc_curve, auc, f1_score, make_scorer, confusion_matrix
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from sklearn.model_selection import train_test_split,learning_curve,GridSearchCV
from sklearn.dummy import DummyClassifier
```

3 Charakterisierung des Datensatzes

Der vorliegende Datensatz enthält Informationen über potenzielle Kreditnehmer und dient als Grundlage für die Vorhersage von Kreditzusagen oder -ablehnungen mittels eines binären Klas-

sifikationsmodells. Der Datensatz besteht aus insgesamt 614 Zeilen und umfasst 13 Features, von denen 8 kategorischer und 4 kontinuierlicher Features sind. Zusätzlich gibt es einen Feature zur Identifizierung des Kredits (Loan_ID).

Die kategorischen Features umfassen: - Geschlecht (Gender): Die Geschlechtsidentität des Kreditnehmers. - Verheiratet (Married): Der eheliche Status des Kreditnehmers. - Abhängige (Dependents): Die Anzahl der Familienmitglieder, die finanziell vom Kreditnehmer abhängig sind. - Bildung (Education): Erfasst, ob der Kreditnehmer eine höhere Abschlussausbildung abgeschlossen hat oder nicht. - Selbstständig (Self_Employed): Der Beschäftigungsstatus des Kreditnehmers. - Kreditgeschichte (Credit_History): Aufzeichnungen über die vorherige Kredithistorie des Kreditnehmers (0: schlechte Kredithistorie, 1: gute Kredithistorie). - Immobilienbereich (Property_Area): Die Lage des Eigentums des Kreditnehmers (ländlich, halb-urban, urban). - Kreditstatus (Loan_Status): Der Status des beantragten Kredits (Y: akzeptiert, N: nicht akzeptiert).

Die kontinuierlichen Features umfassen: - ApplicantIncome: Das monatliche Einkommen des Antragstellers. - CoapplicantIncome: Das zusätzliche monatliche Einkommen des Mitbewerbers. - LoanAmount: Die Höhe des beantragten Kredits in Tausenden. - Loan_Amount_Term: Die Laufzeit des Kredits in Monaten.

Die Kombination dieser Variablen bietet einen detailierten Einblick in die finanzielle Situation der Kreditnehmer sowie in ihre persönlichen und beruflichen Hintergründe. Dies ermöglicht es, potenzielle Muster und Zusammenhänge zu identifizieren, die die Kreditentscheidung beeinflussen könnten.

[99]:
```
# Load CSV data into DataFrame and display summary
df = pd.read_csv("data/loan_dataset.csv")
df.info()  # Display DataFrame info
df.head()  # Display first few rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[99]:
```
     Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

Die Datenqualität des vorliegenden Datensatzes für das Prjekt weist einige wichtige Aspekte auf, die berücksichtigt werden müssen.

Zunächst einmal gibt es einige NaN-Werte pro Zeile, insbesondere in den Spalten 'Gender', 'Married', 'Dependents', 'Self_Employed', 'LoanAmount', 'Loan_Amount_Term' und 'Credit_History'. Für Spalten mit kategorischen Werten im Objektformat werden die NaN-Werte durch den Modus der jeweiligen Spalte ersetzt, während für Spalten mit numerischen kontinuierlichen Werten der Median verwendet wird. Ein weiterer wichtiger Punkt betrifft die Spalte 'CoapplicantIncome', die auffällig viele 0-Werte aufweist. Dies deutet darauf hin, dass in vielen Fällen der Kreditnehmer der alleinige Verdienstträger ist, was etwa 44% der Fälle ausmacht. Dieser Merkmal könnte potenziell wichtige Informationen über die finanzielle Situation der Kreditnehmer liefern und sollte bei der Modellierung berücksichtigt werden. Desweiteren weist die Spalte 'Credit_History' 0-Werte auf, was auf einen negativen Kreditverlauf hinweist. Es ist wichtig zu beachten, dass diese Werte eigentlich kategorial sind und entweder einen guten Kreditverlauf (1) oder einen schlechten Kreditverlauf (0) repräsentieren. Daher müssen diese Werte entsprechend umgewandelt werden, um eine korrekte Interpretation zu ermöglichen.

[100]:
```python
# Iterating through each column in the DataFrame
for column in df.columns:
    # Counting the number of NaN (missing) values in the current column
    null_count = (df[column].isna()).sum()

    # Counting the number of 0 values in the current column
    zero_count = (df[column] == 0).sum()

    # Printing the counts of zero and NaN values for the current column
    print(f"Column '{column}': 0: {zero_count}, NaN: {null_count}")
```

```
Column 'Loan_ID': 0: 0, NaN: 0
Column 'Gender': 0: 0, NaN: 13
Column 'Married': 0: 0, NaN: 3
Column 'Dependents': 0: 0, NaN: 15
Column 'Education': 0: 0, NaN: 0
Column 'Self_Employed': 0: 0, NaN: 32
Column 'ApplicantIncome': 0: 0, NaN: 0
Column 'CoapplicantIncome': 0: 273, NaN: 0
Column 'LoanAmount': 0: 0, NaN: 22
Column 'Loan_Amount_Term': 0: 0, NaN: 14
Column 'Credit_History': 0: 89, NaN: 50
Column 'Property_Area': 0: 0, NaN: 0
Column 'Loan_Status': 0: 0, NaN: 0
```

Die Analyse der Werte in jeweiligen Spalten in unserem Datensatz zeigt eine gute Variation in den Kategorien, jedoch mit einigen fehlenden Daten in Schlüsselspalten wie 'Gender', 'Married', 'Dependents','Self_Employed', und 'Credit_History'. Diese Spalten umfassen sowohl kategorische als auch numerische Daten, die den Familienstand, das Geschlecht, die Abhängigen, den Beschäftigungsstatus und die Kredithistorie des Kreditnehmers darstellen. Die 'Credit_History'-Spalte, obwohl sie numerische Werte wie 0 und 1 enthält, wird als kategorische Variable behandelt, da sie lediglich zwei Zustände (gute oder schlechte Kredithistorie) repräsentiert. Während Spalten wie 'Education', 'Property_Area', und 'Loan_Status' vollständig sind, bedürfen andere einer sorgfältigen Behandlung der fehlenden Werte, um die Modellgenauigkeit zu verbessern. Zusätzlich wurde die erste Spalte 'Loan_ID' entfernt, da sie keine Relevanz für weitere Analysen oder Modellierungen hat und lediglich als Identifikator für einzelne Kredite diente.

[101]:
```python
# Drops the first column 'Loan ID' since it's deemed unnecessary
df = df.drop(df.columns[0], axis=1)

# Converts the 'Credit_History' column from float to object datatype since it's␣
 ↪categorical
df['Credit_History'] = df['Credit_History'].astype(str)

# Iterates through columns with object datatype and prints their unique values
for column in df.select_dtypes(include='object').columns:
    # Extracts unique values of the current column
    unique_values = df[column].unique()

    # Prints the unique values of the current column
    print(f"Unique values of column '{column}': {unique_values}")
```

```
Unique values of column 'Gender': ['Male' 'Female' nan]
Unique values of column 'Married': ['No' 'Yes' nan]
Unique values of column 'Dependents': ['0' '1' '2' '3+' nan]
Unique values of column 'Education': ['Graduate' 'Not Graduate']
Unique values of column 'Self_Employed': ['No' 'Yes' nan]
Unique values of column 'Credit_History': ['1.0' '0.0' 'nan']
Unique values of column 'Property_Area': ['Urban' 'Rural' 'Semiurban']
```

```
Unique values of column 'Loan_Status': ['Y' 'N']
```

Die Analyse der Klassenverteilung zeigt, dass der Datensatz eine gewisse Ungleichheit aufweist, wobei 422 Kredite als akzeptiert ('Y') und nur 192 Kredite als nicht akzeptiert ('N') gekennzeichnet sind. Diese Ungleichheit könnte potenziell Auswirkungen auf die Modellleistung haben, da das Modell möglicherweise dazu neigt, die überrepräsentierte Klasse besser zu lernen und die unterrepräsentierte Klasse zu vernachlässigen. Um dieses Problem anzugehen, könnten verschiedene Ansätze wie das Sampling von Daten (z. B. Oversampling der unterrepräsentierten Klasse oder Undersampling der überrepräsentierten Klasse) oder die Verwendung von Techniken wie dem Gewichten von Klassen in Betracht gezogen werden.

[102]:
```
print("Verteilung der Klassen:", Counter(df['Loan_Status']))
```

```
Verteilung der Klassen: Counter({'Y': 422, 'N': 192})
```

Die Spalten werden visualisiert, indem sie jeweils nach einzigartigen Werten gruppiert und anschließend nach dem Kreditstatus, also ob eine Person einen Kredit erhalten hat oder nicht, aufgeschlüsselt wurden.

Die Analyse der Datenvisualisierung zeigt interessante Muster in Bezug auf die Genehmigung von Krediten basierend auf verschiedenen Merkmalen: - Die Mehrheit der Kreditnehmer sind männlich, und Männer haben eine höhere Genehmigungsrate für Kredite im Vergleich zu Frauen. - Verheiratete Personen stellen den größten Anteil der Kreditnehmer dar, und sie haben eine höhere Genehmigungsrate im Vergleich zu unverheirateten Personen. - Personen ohne Angehörige haben eine höhere Genehmigungsrate für Kredite. - Absolventen haben eine höhere Genehmigungsrate als Personen ohne Hochschulabschluss. - Selbstständige haben eine niedrigere Genehmigungsrate im Vergleich zu Angestellten. - Städtische Bewohner haben tendenziell eine leicht höhere Genehmigungsrate für Kredite im Vergleich zu ländlichen oder semiurbanen Bewohnern. - Personen mit einer positiven Kredithistorie haben eine deutlich höhere Genehmigungsrate im Vergleich zu Personen mit negativer oder unbekannter Kredithistorie.

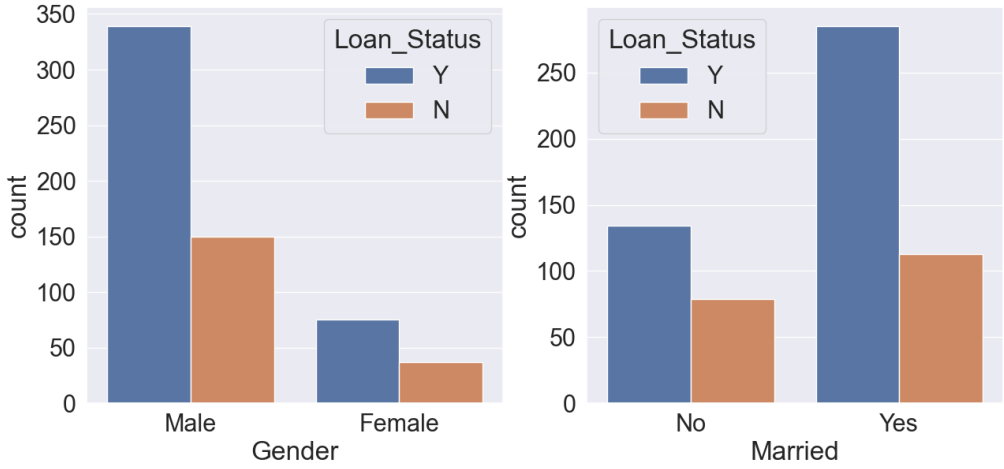
Die Histogramme und Boxplots der numerischen Spalten zeigen eine breite Streuung der Daten und das Vorhandensein von Ausreißern, was auf eine dringende Notwendigkeit der Bereinigung hinweist, um die Qualität der Modelle zu verbessern. Des Weiteren offenbart die Korrelationsmatrix starke Zusammenhänge zwischen bestimmten Merkmalen, insbesondere zwischen der Kreditsumme und dem Einkommen des Kreditnehmers sowie zwischen Kreditzusage und Kredithistorie.

Der Datensatz enthält reichhaltige Informationen, aber weist auch Herausforderungen in Bezug auf die Datenqualität und Ausreißer auf, die behoben werden müssen, um genaue Vorhersagen zu treffen und aussagekräftige Einsichten zu gewinnen.

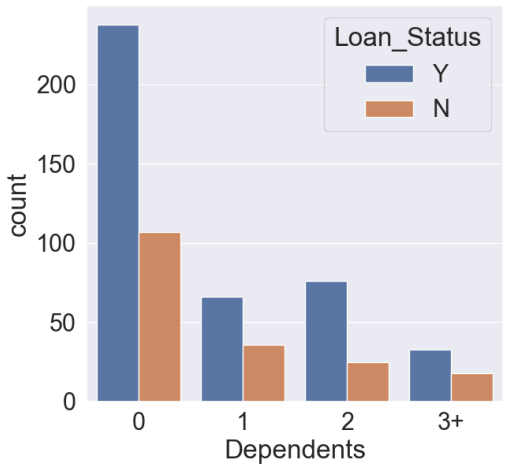
[103]:
```
# adjust fig and font size
sns.set(font_scale=2)

# Distribution of categorical variables
plt.figure(figsize=(15, 30))  # Set the figure size for better visualization

# Subplots for each categorical variable
plt.subplot(4, 2, 1)
sns.countplot(data=df, x='Gender', hue='Loan_Status')
```

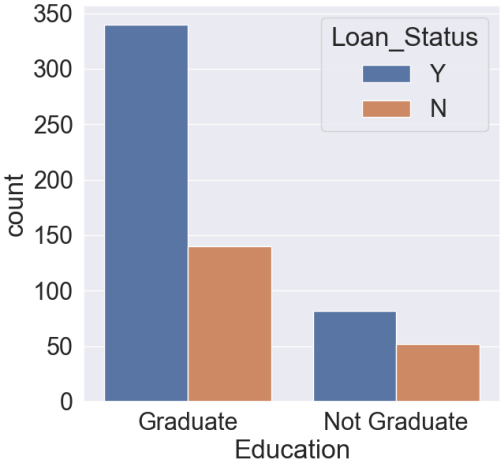
```
plt.subplot(4, 2, 2)
sns.countplot(data=df, x='Married', hue='Loan_Status')

plt.subplot(4, 2, 3)
sns.countplot(data=df, x='Dependents', hue='Loan_Status')

plt.subplot(4, 2, 4)
sns.countplot(data=df, x='Education', hue='Loan_Status')

plt.subplot(4, 2, 5)
sns.countplot(data=df, x='Self_Employed', hue='Loan_Status')

plt.subplot(4, 2, 6)
sns.countplot(data=df, x='Property_Area', hue='Loan_Status')

plt.subplot(4, 2, 7)
sns.countplot(data=df, x='Credit_History', hue='Loan_Status')

plt.subplot(4, 2, 8)
sns.countplot(data=df, x='Loan_Status')  # Distribution of loan status itself

plt.show()  # Show the plots

# Histograms for numerical data
plt.figure(figsize=(12, 15))  # Set the figure size

# Subplots for each numerical variable
plt.subplot(2, 2, 1)
sns.histplot(df['ApplicantIncome'], bins=20, kde=True)

plt.subplot(2, 2, 2)
sns.histplot(df['CoapplicantIncome'], bins=20, kde=True)

plt.subplot(2, 2, 3)
sns.histplot(df['LoanAmount'], bins=20, kde=True)

plt.subplot(2, 2, 4)
sns.histplot(df['Loan_Amount_Term'], bins=20, kde=True)

plt.show()  # Show the plots

# Boxplots for outlier detection
plt.figure(figsize=(12, 15))  # Set the figure size

# Subplots for each numerical variable
plt.subplot(2, 2, 1)
```

```
sns.boxplot(data=df, x='Loan_Status', y='ApplicantIncome')

plt.subplot(2, 2, 2)
sns.boxplot(data=df, x='Loan_Status', y='CoapplicantIncome')

plt.subplot(2, 2, 3)
sns.boxplot(data=df, x='Loan_Status', y='LoanAmount')

plt.subplot(2, 2, 4)
sns.boxplot(data=df, x='Loan_Status', y='Loan_Amount_Term')

plt.show()  # Show the plots

# Correlation matrix
# Encoding categorical variables for correlation analysis
df_encoded = df.copy()
label_encoder = LabelEncoder()
for column in df_encoded.select_dtypes(include=['object']).columns:
    df_encoded[column] = label_encoder.fit_transform(df_encoded[column])

# Creating a heatmap for the correlation matrix of all columns
plt.figure(figsize=(20, 20))
correlation_matrix = df_encoded.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap for All Columns (After Label Encoding)')
plt.show()  # Show the heatmap
```

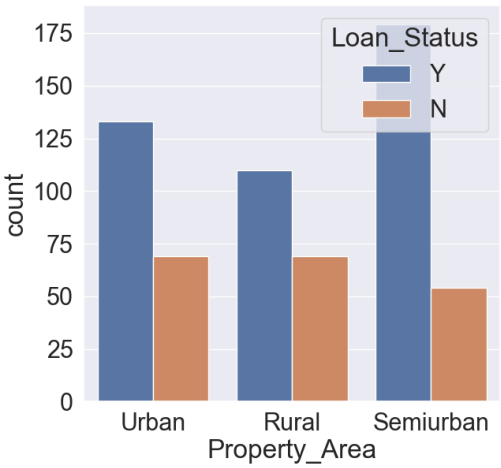

Loan_Status
Y
N
count
Male
Female
Gender

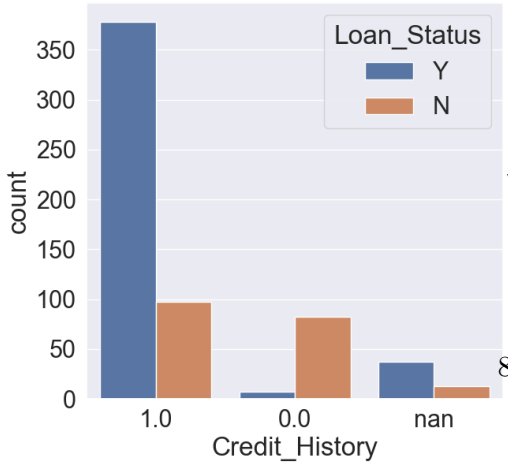
Loan_Status
Y
N
count
No
Yes
Married

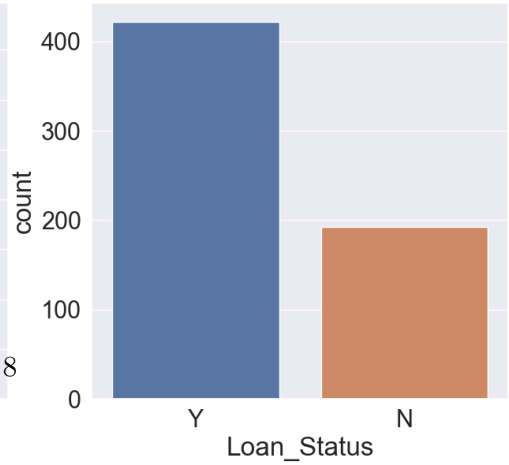

Loan_Status
Y
N
count
0
1
2
3+
Dependents

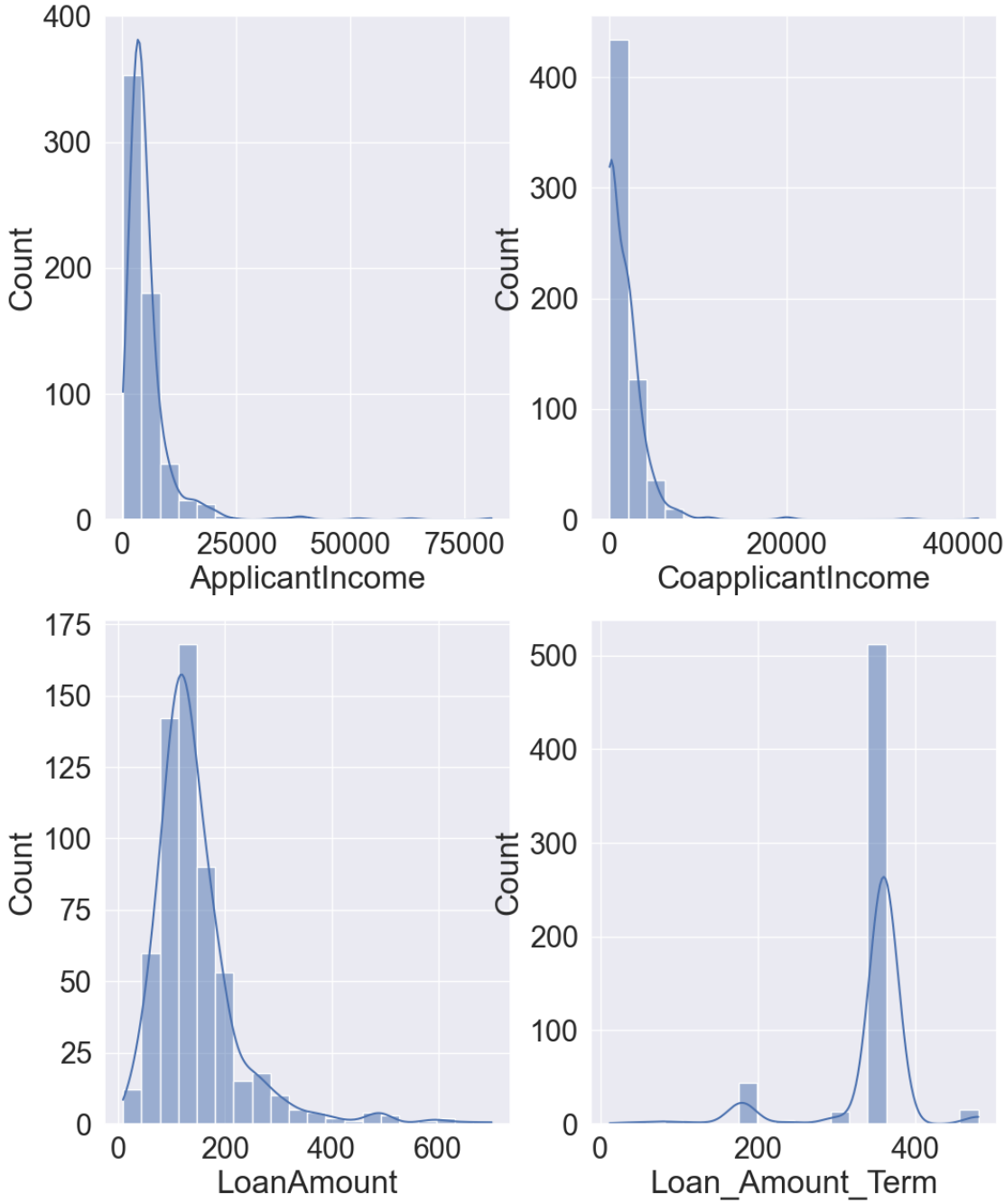

Loan_Status
Y
N
count
Graduate
Not Graduate
Education

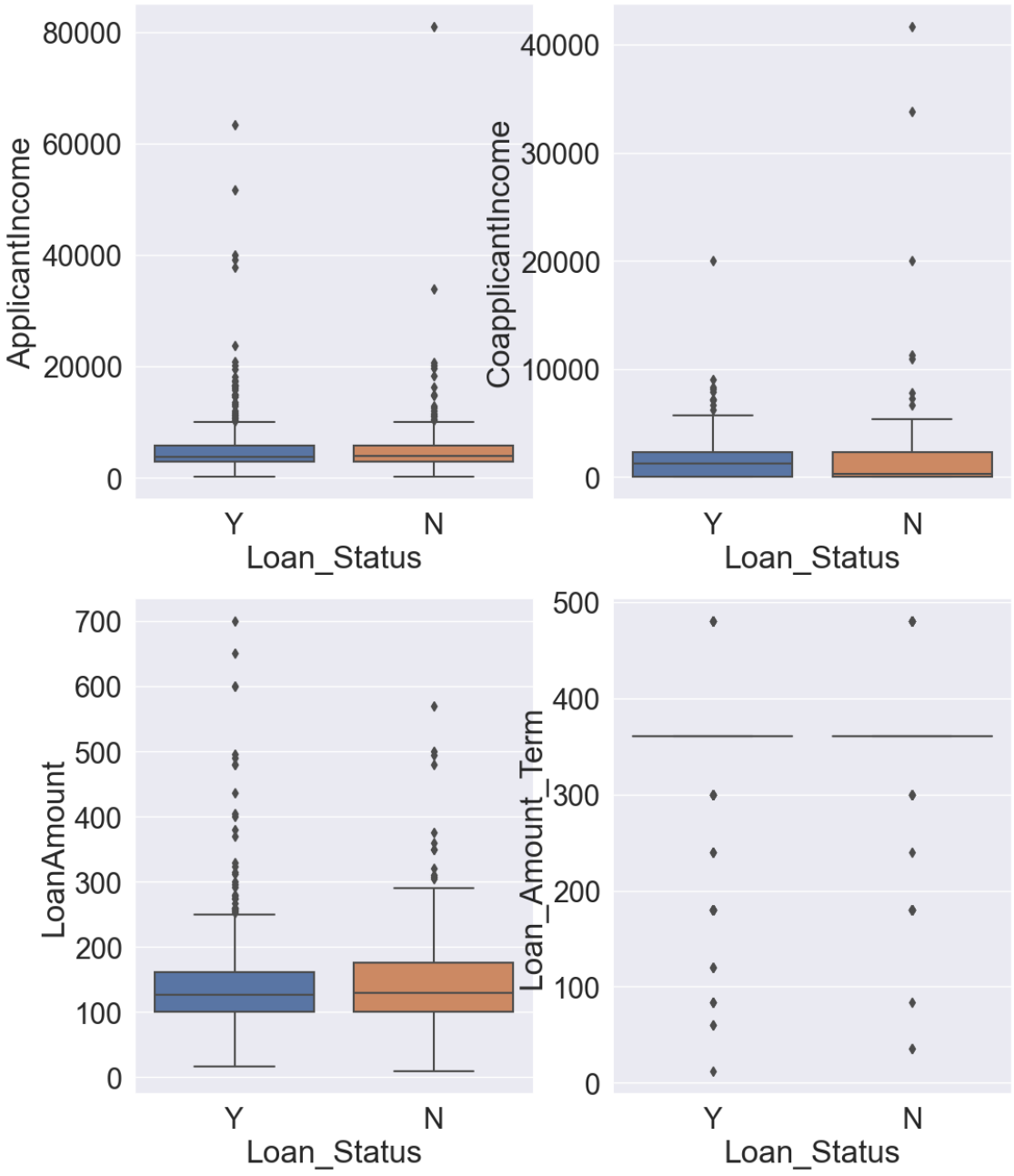

Loan_Status
Y
N
count
No
Yes
Self_Employed

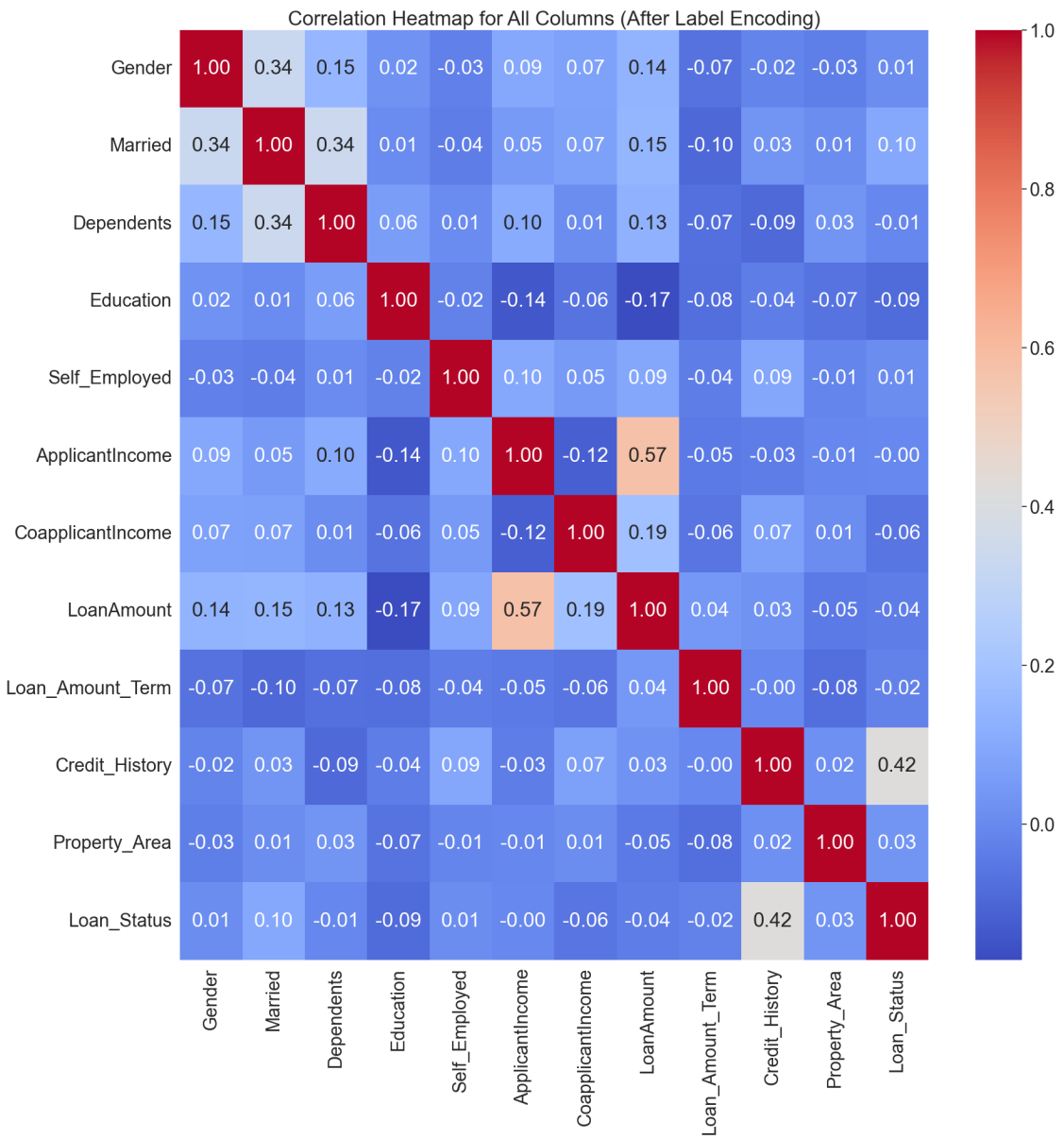

Loan_Status
Y
N
count
Urban
Rural
Semiurban
Property_Area


Loan_Status
Y
N
count
1.0
0.0
nan
Credit_History


count
Y
N
Loan_Status


Count
ApplicantIncome
0
25000
50000
75000
100
200
300
400
Count
CoapplicantIncome
0
20000
40000
100
200
300
400
Count
LoanAmount
0
200
400
600
0
25
50
75
100
125
150
175
Count
Loan_Amount_Term
0
200
400
0
100
200
300
400
500


ApplicantIncome
80000
60000
40000
20000
0
Y
N
Loan_Status
CoapplicantIncome
40000
30000
20000
10000
0
Y
N
Loan_Status
LoanAmount
700
600
500
400
300
200
100
0
Y
N
Loan_Status
Loan_Amount_Term
500
400
300
200
100
0
Y
N
Loan_Status



Der Datensatz bietet relevante Informationen für das Machine-Learning-Projekt, indem er das Problem der Kreditvergabe anspricht. Obwohl einige Datenbereinigungen und -transformationen erforderlich sind, um die Vorhersagegenauigkeit des Modells zu verbessern und Auswirkungen wie reduzierte Modellleistung sowie verzerrte Modellschätzungen und Ergebnisse zu vermeiden, ist die Qualität der Daten insgesamt solide. Trotz des Vorhandenseins von Ausreißern und fehlenden Werten verfügt der Datensatz über ausreichend Datenpunkte, die potenziell bearbeitet werden können, um die Modellrobustheit zu steigern. Die Analyse liefert wichtige Erkenntnisse über die Einflussfaktoren auf die Kreditgenehmigung und legt den Grundstein für weitere Modellierungs- und Optimierungsschritte.

4 Feature Engineering

Im Feature Engineering-Abschnitt wird zunächst die Datenbereinigung durchgeführt. Dieser Schritt ist wichtig, um sicherzustellen, dass der Datensatz frei von fehlenden Werten ist und eine konsistente Grundlage für die Modellbildung bietet. Für kategoriale Spalten werden fehlende Werte durch den am häufigsten vorkommenden Wert, den sogenannten Modus, ersetzt. Bei numerischen Spalten hingegen werden die fehlenden Werte durch den Median ersetzt, um die Auswirkungen von Ausreißern zu minimieren und eine robuste Schätzung zu gewährleisten. Diese Bereinigungsmethoden tragen dazu bei, dass der Datensatz homogen und für die folgenden Schritte des Feature Engineerings und der Modellierung geeignet ist.

[104]:
```
# Replace NaN values with mode values for categorical columns
for col in df.columns:
    # Check if the data type of the column is 'object'
    if df[col].dtype == 'object':
        # For columns with 'object' data type: Convert to string data type and␣
 ↪fill NaN values with mode
        df[col] = df[col].fillna(df[col].mode()[0])

# Replace NaN values with median for numerical values
for column in df.select_dtypes(include='number').columns:
    # Calculate the median value of the column
    median_value = df[column].median()
    # Fill NaN values with the calculated median value
    df[column].fillna(median_value, inplace=True)

# Check if there are any remaining NaN values
print(df.isna().sum())
```

```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Anschließend werden basierend auf den vorhandenen Spalten neue Features generiert, wie zum Beispiel TotalIncome und die LoanAmount-to-TotalIncome-Ratio. Diese neuen Features können die Leistung der Modelle verbessern, indem sie zusätzliche Einblicke in die finanzielle Situation der Kreditnehmer liefern und möglicherweise relevante Muster hervorheben, die bei der Kreditentscheidung eine Rolle spielen könnten.

[105]:
```
# Create a new column 'TotalIncome' by summing 'ApplicantIncome' and␣
 ↪'CoapplicantIncome'
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']

# Create a new column 'LA2TI-Ratio' by dividing 'LoanAmount' by 'TotalIncome'
df['LA2TI-Ratio'] = df['LoanAmount'] / df['TotalIncome']
```

Im nächsten Schritt werden Ausreißer in den numerischen Spalten durch den Median ersetzt. Dieser Schritt ist wichtig, um sicherzustellen, dass extreme Werte das Modell nicht übermäßig beeinflussen und die Modellleistung verzerrt wird. Durch die Ersetzung von Ausreißern mit dem Median wird die Robustheit des Modells verbessert und eine zuverlässige Schätzung der Daten ermöglicht.

[106]:
```
# Iterate through numerical columns
for column in df.select_dtypes(include='number').columns:
    # Calculate the first quartile (Q1)
    q1 = df[column].quantile(0.25)

    # Calculate the third quartile (Q3)
    q3 = df[column].quantile(0.75)

    # Calculate the interquartile range (IQR)
    iqr = q3 - q1

    # Calculate the lower bound
    lower_bound = q1 - 1.5 * iqr

    # Calculate the upper bound
    upper_bound = q3 + 1.5 * iqr

    # Replace outliers with median value
    df[column] = df[column].apply(lambda x: x if lower_bound <= x <=␣
 ↪upper_bound else df[column].median())
```

Im nächsten Schritt werden die kategorialen Daten in ein maschinenlesbares Format mit der One-Hot-Encoding-Technik umgewandelt. Bei One-Hot-Encoding wird jede Kategorie in einen Binärvektor transformiert, wobei für jede Kategorie eine separate Dimension vorhanden ist. Diese Dimensionen werden mit Nullen initialisiert, außer derjenigen, die der Position der Kategorie im ursprünglichen Kategorie-Array entspricht, die den Wert Eins erhält.

Ein einfaches Beispiel dafür ist die Umwandlung der “Gender”-Spalte. Durch One-Hot-Encoding werden zwei neue Spalten erstellt: “gender_male” und “gender_female”. Wenn in der Spalte “gender_male” der Wert “True” ist, ist der Wert in der Spalte “gender_female” automatisch “False”. Dieses Prinzip wird auf alle kategorialen Spalten angewendet. Es ist zu beachten, dass jeweils eine Spalte entfernt wird, um Redundanz zu vermeiden. Wenn beispielsweise eine Spalte zwei verschiedene Werte aufnehmen kann (z. B. “Ja” und “Nein”), reicht eine einzelne Binärspalte aus, um die Information zu repräsentieren.

Die Umwandlung von Objektspalten in boolsche Spalten mittels One-Hot-Encoding ist wichtig, damit das Modell die kategorialen Daten korrekt interpretieren und nutzen kann. Maschinelle Ler-

nalgorithmen, insbesondere lineare Modelle und Modelle, die auf numerischen Eingaben basieren, benötigen numerische Werte, um die Beziehungen zwischen den Merkmalen zu verstehen und Vorhersagen zu treffen. Kategoriale Daten in Form von Texten oder Wörtern können von den meisten Algorithmen nicht direkt verarbeitet werden. Daher ist es erforderlich, diese Daten in eine numerische Darstellung umzuwandeln, die das Modell verarbeiten kann.

[107]:
```
# Get the names of categorical columns
categorical_columns = df.select_dtypes(include='object').columns

# Perform One Hot Encoding on categorical columns
# pd.get_dummies() function is used to convert categorical variables into dummy/
 ↪indicator variables
# 'drop_first=True' is used to drop the first category of each categorical␣
 ↪variable to avoid multicollinearity issues
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Output information about the DataFrame after encoding
df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ApplicantIncome          614 non-null    float64
 1   CoapplicantIncome        614 non-null    float64
 2   LoanAmount               614 non-null    float64
 3   Loan_Amount_Term         614 non-null    float64
 4   TotalIncome              614 non-null    float64
 5   LA2TI-Ratio              614 non-null    float64
 6   Gender_Male              614 non-null    uint8
 7   Married_Yes              614 non-null    uint8
 8   Dependents_1             614 non-null    uint8
 9   Dependents_2             614 non-null    uint8
 10  Dependents_3+            614 non-null    uint8
 11  Education_Not Graduate   614 non-null    uint8
 12  Self_Employed_Yes        614 non-null    uint8
 13  Credit_History_1.0       614 non-null    uint8
 14  Credit_History_nan       614 non-null    uint8
 15  Property_Area_Semiurban  614 non-null    uint8
 16  Property_Area_Urban      614 non-null    uint8
 17  Loan_Status_Y            614 non-null    uint8
dtypes: float64(6), uint8(12)
memory usage: 36.1 KB
```

In diesem Schritt werden die Werte der numerischen Spalten normalisiert, sodass sie im Bereich zwischen 0 und 1 liegen. Dies dient dazu, die Merkmale auf vergleichbare Skalen anzupassen und Verzerrungen aufgrund unterschiedlicher Skalen zu vermeiden. Durch diese Maßnahme wird die

potenzielle Probleme bei der Modellierung reduziert.

[108]:
```
# Define the list of numerical columns to be normalized
numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'TotalIncome', 'LA2TI-Ratio']

# Instantiate MinMaxScaler for normalization
scaler = MinMaxScaler()

# Apply Min-Max scaling to the numerical columns using fit_transform()
df_encoded[numerical_columns] = scaler.
 ↪fit_transform(df_encoded[numerical_columns])
```

Nach dem Encoding und der Normalisierung der Daten wurden nun zusätzliche Spalten mit einheitlicher Skalierung hinzugefügt, um die Informationen maschinenlesbar zu machen und potenzielle Probleme bei der Modellierung zu minimieren.

[109]:
```
# Output information about the DataFrame after encoding and normalization
df_encoded.info()

# Output descriptive statistics of the DataFrame after normalization
# This provides summary statistics such as mean, min, max, etc. for each␣
 ↪numerical column
df_encoded.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ApplicantIncome          614 non-null    float64
 1   CoapplicantIncome        614 non-null    float64
 2   LoanAmount               614 non-null    float64
 3   Loan_Amount_Term         614 non-null    float64
 4   TotalIncome              614 non-null    float64
 5   LA2TI-Ratio              614 non-null    float64
 6   Gender_Male              614 non-null    uint8
 7   Married_Yes              614 non-null    uint8
 8   Dependents_1             614 non-null    uint8
 9   Dependents_2             614 non-null    uint8
 10  Dependents_3+            614 non-null    uint8
 11  Education_Not Graduate   614 non-null    uint8
 12  Self_Employed_Yes        614 non-null    uint8
 13  Credit_History_1.0       614 non-null    uint8
 14  Credit_History_nan       614 non-null    uint8
 15  Property_Area_Semiurban  614 non-null    uint8
 16  Property_Area_Urban      614 non-null    uint8
 17  Loan_Status_Y            614 non-null    uint8
```

```
dtypes: float64(6), uint8(12)
memory usage: 36.1 KB
```

[109]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	\
count	614.000000	614.000000	614.000000	614.0	
mean	0.395365	0.225606	0.475901	0.0	
std	0.185074	0.252623	0.178485	0.0	
min	0.000000	0.000000	0.000000	0.0	
25%	0.273050	0.000000	0.363546	0.0	
50%	0.366628	0.206587	0.474104	0.0	
75%	0.473521	0.378749	0.569721	0.0	
max	1.000000	1.000000	1.000000	0.0	

	TotalIncome	LA2TI-Ratio	Gender_Male	Married_Yes	Dependents_1	\
count	614.000000	614.000000	614.000000	614.000000	614.000000	
mean	0.375766	0.493203	0.817590	0.653094	0.166124	
std	0.194900	0.185420	0.386497	0.476373	0.372495	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.245383	0.383670	1.000000	0.000000	0.000000	
50%	0.358008	0.510410	1.000000	1.000000	0.000000	
75%	0.457459	0.615335	1.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	

	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	\
count	614.000000	614.000000	614.000000	614.000000	
mean	0.164495	0.083062	0.218241	0.133550	
std	0.371027	0.276201	0.413389	0.340446	
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	

	Credit_History_1.0	Credit_History_nan	Property_Area_Semiurban	\
count	614.000000	614.000000	614.000000	
mean	0.773616	0.081433	0.379479	
std	0.418832	0.273722	0.485653	
min	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	
50%	1.000000	0.000000	0.000000	
75%	1.000000	0.000000	1.000000	
max	1.000000	1.000000	1.000000	

	Property_Area_Urban	Loan_Status_Y
count	614.000000	614.000000
mean	0.328990	0.687296
std	0.470229	0.463973

```
min             0.000000        0.000000
25%             0.000000        0.000000
50%             0.000000        1.000000
75%             1.000000        1.000000
max             1.000000        1.000000
```

5 Split des Datensatzes

Nun erfolgt die Aufteilung des Datensatzes in Trainings-, Test- und Validierungssets. Zuerst wird der Datensatz in Trainings- und Testsets aufgeteilt, wobei 70% der Daten dem Trainingsset und 30% dem Testset zugewiesen werden. Anschließend wird das Testset erneut in Test- und Validierungssets aufgeteilt, wobei 60% des ursprünglichen Testsets dem neuen Testset und 40% dem Validierungsset zugewiesen werden.

Nach der Durchführung des Splits ergibt sich folgende Größenverteilung der einzelnen Datensätze: - Trainingsset: 429 Einträgen - Validierungsset: 74 Einträgen - Testset: 111 Einträgen

Diese Aufteilung garantiert, dass das Modell ausreichend Daten zum Training erhält, während separate Test- und Validierungssets zur Bewertung der Modellleistung genutzt werden können.

Des Weiteren ist es wichtig zu berücksichtigen, dass Ungleichgewichte im Datensatz bei einer zufälligen Aufteilung zu einer verzerrten Stichprobe führen können, insbesondere wenn die Klassen unterschiedlich häufig vertreten sind. In solchen Situationen wird eine stratifizierte Aufteilung des Datensatzes durchgeführt, um sicherzustellen, dass die Proportionen jeder Klasse sowohl im Trainings- als auch im Testset gleich bleiben.

[110]:
```
# Separate the data into features (X) and target variable (y)
X = df_encoded.drop('Loan_Status_Y', axis=1)  # Features
y = df_encoded['Loan_Status_Y']  # Target variable

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=7,stratify=y)

# Further split the test set into test and validation sets
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.4,␣
 ↪random_state=7)

# Print the sizes of the individual sets
print("Größe des Trainingssets:", X_train.shape[0])
print("Größe des Validierungssets:", X_val.shape[0])
print("Größe des Testsets:", X_test.shape[0])
```

```
Größe des Trainingssets: 429
Größe des Validierungssets: 74
Größe des Testsets: 111
```

Hier wird das Ungleichgewicht in den Klassifikationsdatensätzen mit der SMOTE-Technik (Synthetic Minority Over-sampling Technique) behoben. Diese Methode vergrößert die Minderheit-

sklasse im Trainingsdatensatz durch die Generierung synthetischer Beispiele, was drei wesentliche Vorteile bietet:

Erstens verhindert SMOTE Verzerrungen, indem sichergestellt wird, dass das Modell die Minderheitsklasse angemessen lernt und sich nicht ausschließlich auf die Mehrheitsklasse konzentriert. Dies ist besonders wichtig, wenn das Hauptinteresse auf der Minderheitsklasse liegt. Zweitens trägt SMOTE dazu bei, Overfitting zu reduzieren, da durch die Erweiterung des Trainingssets das Risiko gemindert wird, dass das Modell zu spezifisch auf die Mehrheitsklasse zugeschnitten wird. Drittens verbessert die SMOTE-Technik die Gesamtleistung des Modells, indem sie der Minderheitsklasse mehr Gewicht verleiht und dem Modell mehr Trainingsbeispiele dieser Klasse zur Verfügung stellt. Dies führt zu einem ausgewogeneren Verhältnis zwischen den Klassen und ermöglicht genauere Vorhersagen bei seltenen Ereignissen.

Vor der Anwendung von SMOTE war die Verteilung der Klassen im Trainingsset unausgewogen: {1:295, 0:134}. Nach dem Einsatz von SMOTE zeigt die Verteilung ein ausgeglichenes Verhältnis: {1: 295, 0: 295}.

[111]:
```
# Print the distribution of classes before applying SMOTE
print("Verteilung der Klassen in training set vor Oversampling:",␣
 ↪Counter(y_train))

# Initialize the SMOTE oversampler
smote = SMOTE(random_state=7)

# Apply SMOTE to the data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Print the distribution of classes after applying SMOTE
print("Verteilung der Klassen in training set nach SMOTE:",␣
 ↪Counter(y_train_resampled))
```

```
Verteilung der Klassen in training set vor Oversampling: Counter({1: 295, 0:
134})
Verteilung der Klassen in training set nach SMOTE: Counter({1: 295, 0: 295})
```

Der Trainingsdatensatz wird durch das SMOTE-Verfahren ausbalanciert, wobei Validierungs- und Testdatensätze unverändert bleiben. Dies geschieht aus dem Grund, dass diese Datensätze die natürliche Verteilung der Klassen widerspiegeln sollen, um realistisch zu bewerten, wie gut das Modell mit neuen, unbekannten Daten umgehen kann. Das Anwenden von SMOTE nur auf den Trainingsdatensatz verhindert eine künstliche Verzerrung der Modellbewertung, da das Ziel darin besteht, die Leistung des Modells unter realen Bedingungen zu testen.

[112]:
```
# Count the number of instances of each class in each set
train_classes, train_counts = np.unique(y_train_resampled, return_counts=True)
test_classes, test_counts = np.unique(y_test, return_counts=True)
val_classes, val_counts = np.unique(y_val, return_counts=True)

# Calculate the percentage distribution of classes in each set
```

```
train_class_percentages = {cls: count / len(y_train_resampled) * 100 for cls,␣
 ↪count in zip(train_classes, train_counts)}
test_class_percentages = {cls: count / len(y_test) * 100 for cls, count in␣
 ↪zip(test_classes, test_counts)}
val_class_percentages = {cls: count / len(y_val) * 100 for cls, count in␣
 ↪zip(val_classes, val_counts)}

# Display the percentage distribution of classes for each set
print("Percentage class distribution in the training set:",␣
 ↪train_class_percentages)
print("Percentage class distribution in the validation set:",␣
 ↪val_class_percentages)
print("Percentage class distribution in the test set:", test_class_percentages)
```

```
Percentage class distribution in the training set: {0: 50.0, 1: 50.0}
Percentage class distribution in the validation set: {0: 31.08108108108108, 1:
68.91891891891892}
Percentage class distribution in the test set: {0: 31.53153153153153, 1:
68.46846846846847}
```

6 Auswahl der Metriken

Für eine Bank, die über die Vergabe von Krediten entscheidet, ist die Bewertung der Kreditvergabe ein sensibler Prozess, der sorgfältige Überlegungen erfordert. In einem solchen Kontext sind Metriken wie Precision, Recall und F1-Score von besonderer Relevanz.

Precision misst das Verhältnis der korrekt als kreditwürdig identifizierten Kreditnehmer zur Gesamtanzahl der als kreditwürdig identifizierten Kreditnehmer. Ein hoher Precision-Wert deutet darauf hin, dass das Modell eine genauere Auswahl potenziell kreditwürdiger Kunden trifft, was das Risiko von Kreditausfällen verringert, indem es nur Kunden auswählt, von denen angenommen wird, dass sie ihre Kredite zurückzahlen können. Ein hoher Recall-Wert bedeutet, dass das Modell dazu neigt, die meisten potenziell kreditwürdigen Kunden zu identifizieren. Dies könnte der Bank helfen, potenzielle Kunden nicht zu verlieren und ihr Kreditgeschäft zu erweitern. Die Wahl zwischen Recall und Precision hängt davon ab, welche Art von Fehlern die Bank eher vermeiden möchte. Wenn die Bank eher sicherstellen möchte, dass sie keinen potenziell kreditwürdigen Kunden verpasst, selbst wenn dies bedeutet, einige nicht kreditwürdige Kunden zu akzeptieren (hoher Recall, geringe Precision), wäre Recall die bevorzugte Metrik. In diesem Fall ist es der Bank wichtiger, möglichst viele kreditwürdige Kunden zu identifizieren, auch wenn dadurch einige nicht kreditwürdige Kunden fälschlicherweise akzeptiert werden. Wenn die Bank jedoch das Risiko von Kreditausfällen minimieren möchte, selbst wenn dies bedeutet, potenziell kreditwürdige Kunden abzulehnen (hohe Precision, geringer Recall), wäre Precision die bevorzugte Metrik. Hier liegt der Fokus darauf, nur Kunden zu akzeptieren, bei denen das Modell sehr sicher ist, dass sie kreditwürdig sind, auch wenn dadurch einige tatsächlich kreditwürdige Kunden abgelehnt werden.

Im vorliegenden Datensatz wurden jedoch deutlich mehr Kredite genehmigt als abgelehnt. Dies könnte darauf hinweisen, dass die Kosten für das Ablehnen von kreditwürdigen Kunden höher sind als die Kosten von Kreditausfällen oder dass die Bank ein breiteres Risiko eingehen möchte.

In unserem Fall könnte man sich jedoch dafür entscheiden, auf den F1-Score zu optimieren, denn die Kosten der jeweiligen Entscheidungen und die wirtschaftliche Situation sind hier unklar. Der F1-Score wird maximiert, wenn Recall und Precision gleich sind, was bedeutet, dass das Modell eine ausgewogene Bewertung zwischen beiden Metriken bietet. Der F1-Score berücksichtigt sowohl False Positives als auch False Negatives und wäre daher eine sinnvolle Entscheidung für die Modellbewertung, wenn die Bank ein ausgewogenes Verhältnis zwischen der Identifizierung kreditwürdiger Kunden und der Minimierung von Kreditausfällen anstrebt.

Es sind zwar weitere Metriken wie Accuracy oder ROC AUC verfügbar, berücksichtigen diese nicht spezifisch die Kosten von Fehlklassifikationen. In einem Szenario, in dem False Positives und False Negatives unterschiedliche Auswirkungen haben können, sind Metriken wie Precision und Recall besser geeignet, um die Leistung des Modells zu bewerten. Wenn die Kosten für die Entscheidungen der Bank, ob sie einem Kreditnehmer einen Kredit gewährt oder nicht, nicht bekannt sind oder nicht berücksichtigt werden können, ist es schwierig, eine präzise Bewertung der Modelleffizienz vorzunehmen. In solchen Fällen könnte es angemessen sein, auf Metriken wie Accuracy oder ROC AUC zu verzichten.

7 Auswahl und Beschreibung der ML-Methode

Für die Vorhersage wurde das Random Forest-Modell ausgewählt, da es sich als gute Wahl für die binäre Klassifikation erweist, insbesondere wenn ein Großteil der Features kategorial ist. Die Robustheit gegen Overfitting macht es zu einer attraktiven Option, da es eine zuverlässige Generalisierung auf neue Daten ermöglicht. Zudem ist Random Forest in der Lage, sowohl mit numerischen als auch mit kategorialen Variablen effektiv umzugehen, was die Flexibilität des Modells erhöht. Im Vergleich zu einigen anderen Modellen erfordert Random Forest weniger Feinabstimmung der Hyperparameter, was die Modellentwicklung und -optimierung erleichtert.

Random Forest funktioniert, indem es eine Vielzahl von Entscheidungsbäumen erstellt, die jeweils auf einem zufälligen Teil des Trainingsdatensatzes trainiert werden. Jeder Baum gibt eine Vorhersage ab, und die Vorhersage des Random Forest-Modells wird durch Durchschnitt oder Mehrheitsabstimmung der Vorhersagen der einzelnen Bäume bestimmt. Dies ermöglicht eine robuste und stabile Vorhersageleistung, da das Modell durch die Kombination mehrerer Bäume weniger anfällig für Rauschen und Overfitting ist.

[113]:
```
# Initialize a Random Forest Classifier model with a fixed random state for␣
↪reproducibility
rf_model = RandomForestClassifier(random_state=7)
```

8 Implementierung Training

Der Code rf_model.fit(X_train_resampled, y_train_resampled) trainiert das Random Forest-Modell. Dabei werden die durch SMOTE-Verfahren modifizierte Trainingsdaten X_train_resampled verwendet, die die Merkmale (Features) enthalten, und die entsprechenden Zielvariablen y_train_resampled, die die Labels oder Klassen darstellen. Während des Trainings passt das Modell eine Vielzahl von Entscheidungsbäumen an die Trainingsdaten an. Jeder Baum wird auf einem zufälligen Teil des Trainingsdatensatzes trainiert, und während des Trainings optimiert das Modell die Entscheidungsregeln in jedem Baum, um die Vorhersagegenauigkeit zu maximieren.

[114]:
```
# Train the Random Forest Classifier model using the training data
rf_model.fit(X_train_resampled, y_train_resampled)
```

[114]:
```
RandomForestClassifier(random_state=7)
```

Basierend auf den präsentierten Ergebnissen kann festgestellt werden, dass die Leistung des Modells bei der Bewertung der Kreditvergabe als mäßig einzustufen ist. Der ROC-AUC-Wert von 0,6368286445012787 deutet darauf hin, dass die Trennschärfe zwischen kreditwürdigen und nicht kreditwürdigen Kunden noch Verbesserungspotenzial aufweist.

Im Klassifikationsbericht wird ersichtlich, dass das Modell eine höhere Precision von 0,76 für die Klasse der kreditwürdigen Kunden aufweist, was bedeutet, dass 76% der als kreditwürdig eingestuften Kunden tatsächlich kreditwürdig sind. Allerdings ist der Recall-Wert für diese Klasse mit 0,88 deutlich höher, was darauf hindeutet, dass das Modell dazu neigt, die meisten potenziell kreditwürdigen Kunden zu identifizieren, selbst wenn dies mit dem Risiko verbunden ist, einige nicht kreditwürdige Kunden als kreditwürdig einzustufen. Für die Klasse der nicht kreditwürdigen Kunden zeigt sich ein umgekehrtes Bild: Hier ist der Precision-Wert mit 0,60 niedriger, während der Recall-Wert mit 0,39 relativ gering ausfällt. Dies könnte darauf hinweisen, dass das Modell Schwierigkeiten hat, nicht kreditwürdige Kunden zuverlässig zu identifizieren. Berücksichtigt man die ungleiche Verteilung der Klassen im Validierungsdatensatz (31,08% Klasse 0, 68,92% Klasse 1), könnte dies eine mögliche Erklärung für die niedrigeren Werte bei der Erkennung von nicht kreditwürdigen Kunden sein. Das Modell könnte dazu tendieren, Kunden eher als kreditwürdig einzustufen, um mögliche Geschäftsverluste zu vermeiden.

Der gesamte F1-Score von 0,8181818181818181 für die Validierungsdaten kann als akzeptable Leistung des Modells interpretiert werden. Allerdings zeigt sich, dass je nach Risikobereitschaft und strategischen Prioritäten der Bank, entweder der Precision-Wert (um das Kreditausfallrisiko zu senken) oder der Recall-Wert (um mehr potenzielle Kunden zu akquirieren) optimiert werden könnte.

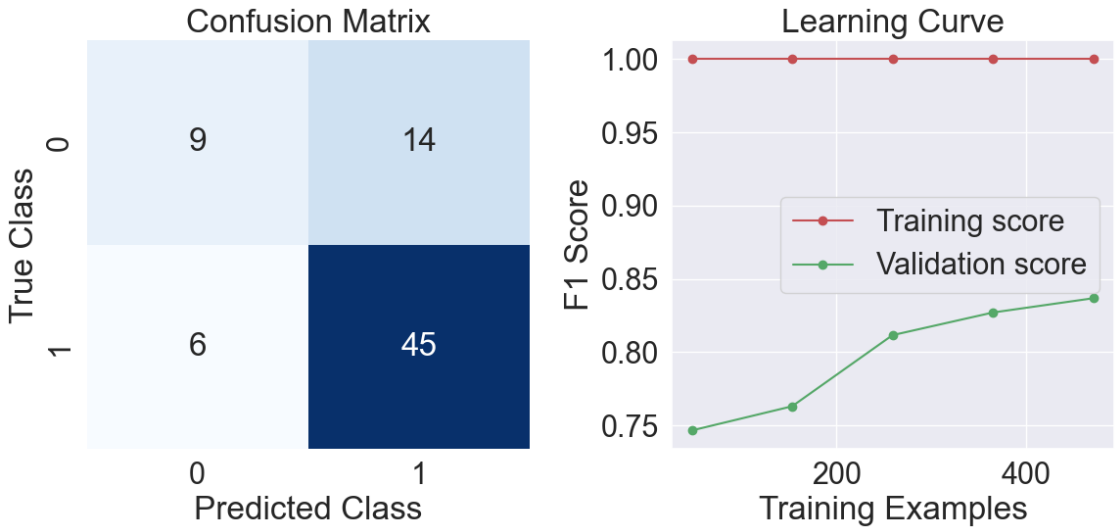
[115]:
```
# Evaluation of the model on validation data
# Predictions on validation data using the trained Random Forest Classifier␣
 ↪model
y_val_pred = rf_model.predict(X_val)

print("RF-Classifier:")
# Output ROC-AUC for validation data
print("ROC-AUC for validation data:", roc_auc_score(y_val, y_val_pred))

# Output classification report for validation data
print("Classification Report for validation data:")
print(classification_report(y_val, y_val_pred))

# Calculate and output the F1-score for validation data average=binary(None)
f1 = f1_score(y_val, y_val_pred)
print("F1-Score for validation data:", f1)
```

```
RF-Classifier:
ROC-AUC for validation data: 0.6368286445012787
```

```
Classification Report for validation data:
              precision    recall  f1-score   support

           0       0.60      0.39      0.47        23
           1       0.76      0.88      0.82        51

    accuracy                           0.73        74
   macro avg       0.68      0.64      0.65        74
weighted avg       0.71      0.73      0.71        74

F1-Score for validation data: 0.8181818181818181
```

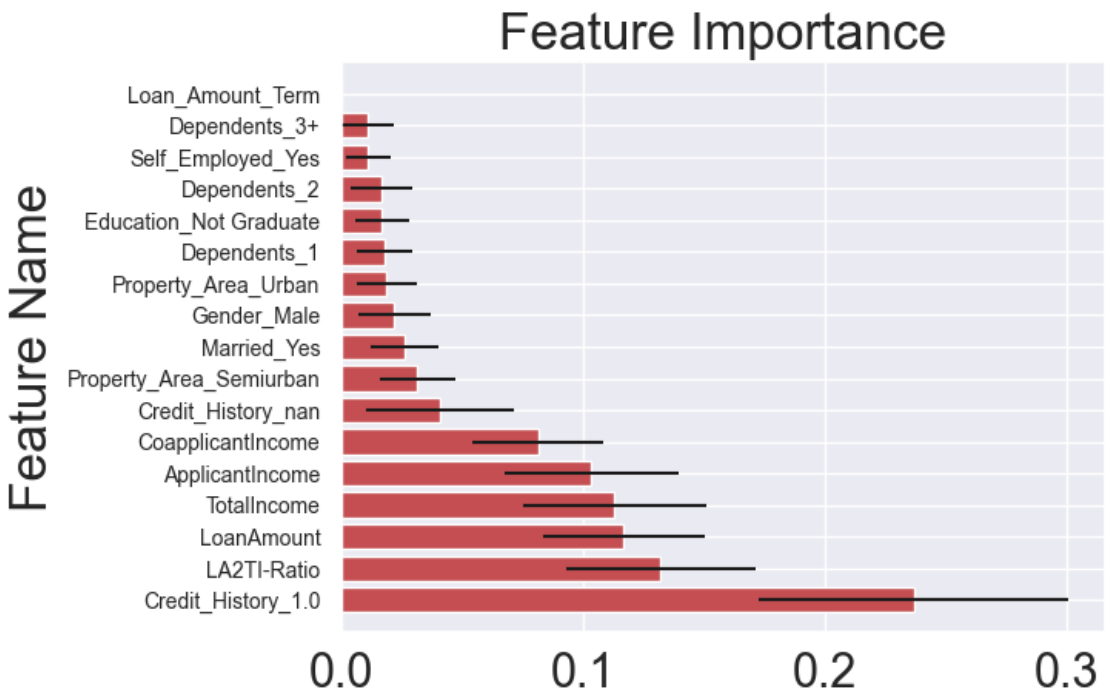
[116]:
```
# Visualization
# Set the figure size for the visualization
plt.figure(figsize=(12, 6))

# Confusion Matrix
# Calculate the confusion matrix
cm = confusion_matrix(y_val, y_val_pred)

# Plot the confusion matrix
plt.subplot(1, 2, 1)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix')

# Define F1-Score as the scoring function for the learning curve
scorer = make_scorer(f1_score)

# Learning Curve
# Generate the learning curve
train_sizes, train_scores, test_scores = learning_curve(rf_model,␣
 ↪X_train_resampled, y_train_resampled, cv=5, scoring=scorer)

# Plot the learning curve
plt.subplot(1, 2, 2)
plt.plot(train_sizes, np.mean(train_scores, axis=1), 'o-', color="r",␣
 ↪label="Training score")
plt.plot(train_sizes, np.mean(test_scores, axis=1), 'o-', color="g",␣
 ↪label="Validation score")
plt.xlabel('Training Examples')
plt.ylabel('F1 Score')
plt.title('Learning Curve')
plt.legend(loc="best")

# Adjust layout for better visualization
```

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



Die Grafik zur Feature-Wichtigkeit zeigt, dass das Feature CreditHistory das entscheidendste für die Vorhersage ist, gefolgt von den fünf numerischen Features.

[125]:
```
# extract Feature Importance
feature_names = X_train_resampled.columns.tolist()
importances = rf_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_model.estimators_],␣
 ↪axis=0)
indices = np.argsort(importances)[::-1]

# visualize Feature Importance
plt.figure()
plt.title("Feature Importance")
plt.barh(range(X_train_resampled.shape[1]), importances[indices], color="r",␣
 ↪xerr=std[indices], align="center")
plt.yticks(range(X_train_resampled.shape[1]), [feature_names[i] for i in␣
 ↪indices], fontsize= 10)
plt.ylim([-1, X_train_resampled.shape[1]])
plt.ylabel('Feature Name')
plt.show()
```



9 Hyperparametertuning

Dieser Schritt führt ein Hyperparameter-Tuning für das Random Forest-Modell durch, indem GridSearch verwendet wird. GridSearchCV ist eine Methode zur Durchführung von Hyperparameter-Tuning, bei der verschiedene Kombinationen von Hyperparametern durchlaufen und bewertet werden, um die besten Hyperparameter für das Modell zu finden und die Leistung zu optimieren.

In diesem Fall wird GridSearchCV mit dem Random Forest Classifier-Modell, dem definierten Hyperparameter-Grid, einer 3-fachen Kreuzvalidierung, dem F1-Score als Bewertungsmetrik und paralleler Verarbeitung instanziiert. Der ursprüngliche Parameter-Grid wurde auskommentiert, um die Laufzeit zu verkürzen. Daher wird der zweite Grid mit den besten Parametern verwendet, die im vorherigen Schritt gefunden wurden.

[118]:
```
# Hyperparameter Grid for GridSearchCV
# Define a grid of hyperparameters to search over
# param_grid = {
#     'bootstrap': [True,False],
#     'class_weight': [{0: 1, 1: 1}, {0: 1.61, 1: 1}, 'balanced'],
#     'max_depth': [5, 25 , 75, None],
#     'max_features': ['auto', 'sqrt'],
#     'min_samples_leaf': [1, 2, 5, 25],
#     'min_samples_split': [2, 5, 10,25],
#     'n_estimators': [50, 100, 150, 300, 500,1000],
#     'random_state': [1,4,7,16,64]
```

```
# }

param_grid = {
    'bootstrap': [True],
    'class_weight': [{0: 1, 1: 1}],
    'max_depth': [25],
    'max_features': ['auto'],
    'min_samples_leaf': [1],
    'min_samples_split': [2],
    'n_estimators': [300],
    'random_state': [7],
}

# GridSearchCV for Hyperparameter Tuning
# Instantiate GridSearchCV with the Random Forest Classifier model, the␣
 ↪hyperparameter grid, 3-fold cross-validation, F1 scoring, verbose mode, and␣
 ↪parallel processing
grid_search = GridSearchCV(rf_model, param_grid, cv=3, scoring='f1', verbose=3,␣
 ↪n_jobs=-1)

# Perform GridSearchCV to find the best hyperparameters
grid_search.fit(X_train_resampled, y_train_resampled)

# Display the best hyperparameters found by GridSearchCV
print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Best Parameters: {'bootstrap': True, 'class_weight': {0: 1, 1: 1}, 'max_depth':
25, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 300, 'random_state': 7}
```

10 Evaluation und Ergebnisdarstellung

Das optimierte Random Forest-Modell zur Bewertung der Kreditwürdigkeit zeigt insgesamt eine mäßige Verbesserung der Leistungsfähigkeit im Vergleich zum ursprünglichen Modell.

Der ROC-AUC-Wert des Testdatensatzes liegt nun bei 0,684. Dieser Wert deutet darauf hin, dass die Trennschärfe des Modells zwischen kreditwürdigen und nicht kreditwürdigen Kunden als mäßig einzustufen ist. Es gibt also noch Potenzial für weitere Verbesserungen. Im Klassifikationsbericht für den Testdatensatz wird ersichtlich, dass das Modell eine Precision von 0,79 für die Klasse der kreditwürdigen Kunden aufweist. Das bedeutet, dass 79% der als kreditwürdig eingestuften Kunden tatsächlich kreditwürdig sind. Der Recall-Wert für diese Klasse liegt bei 0,88, was darauf hindeutet, dass das Modell weiterhin dazu neigt, die meisten potenziell kreditwürdigen Kunden zu identifizieren. Für die Klasse der nicht kreditwürdigen Kunden zeigt sich ein Precision-Wert von 0,65 und ein Recall-Wert von 0,49. Dies bedeutet, dass das Modell Schwierigkeiten hat, nicht kreditwürdige Kunden zuverlässig zu erkennen. Hier besteht weiterhin Optimierungspotenzial. Der F1-Score für die Klasse der kreditwürdigen Kunden liegt bei sehr guten 0,83. Für die Klasse der nicht kreditwürdigen Kunden beträgt der F1-Score 0,56, was auf Verbesserungspotenzial hindeutet.

Bemerkenswert ist, dass der F1-Score über beide Klassen hinweg bei 0,832 liegt. Dies zeigt, dass das optimierte Modell insgesamt eine sehr gute Leistung erbringt. Insgesamt lässt sich festhalten, dass das optimierte Random Forest-Modell eine deutliche Verbesserung der Leistungsfähigkeit im Vergleich zum vorherigen Modell aufweist. Zwar besteht weiterhin Potenzial, um die Vorhersagegenauigkeit, insbesondere für die Klasse der nicht kreditwürdigen Kunden, zu erhöhen, aber das Modell bietet bereits eine solide Grundlage für die Unterstützung der Kreditvergabeentscheidungen. Je nach Risikobereitschaft und strategischen Prioritäten der Bank könnte eine weitere Optimierung des Modells sinnvoll sein, um ein noch ausgewogeneres Verhältnis zwischen Precision und Recall zu erreichen.

Um die Leistung des Modells weiter zu verbessern, könnten folgende Ansätze in Betracht gezogen werden: 1. Verbesserung der Datenqualität 2. Verwendung fortgeschrittener Techniken wie Ensemble-Methoden (z.B. Bagging, Boosting): 3. Ausprobieren anderer Modelltypen (z.B. logistische Regression, SVM)

[119]:
```
# Set the best Random Forest model from GridSearchCV
best_rf_model = grid_search.best_estimator_

# Evaluation of the model on test data
# Predictions on test data using the best Random Forest model
y_test_pred = best_rf_model.predict(X_test)

print("tuned RF-Classifier:")
# Output ROC-AUC for test data
print("ROC-AUC for test data:", roc_auc_score(y_test, y_test_pred))

# Output classification report for test data
print("Classification Report for test data:")
print(classification_report(y_test, y_test_pred))

# Calculate and output the F1-score for test data
f1 = f1_score(y_test, y_test_pred)
print("F1-Score for test data:", f1)
```

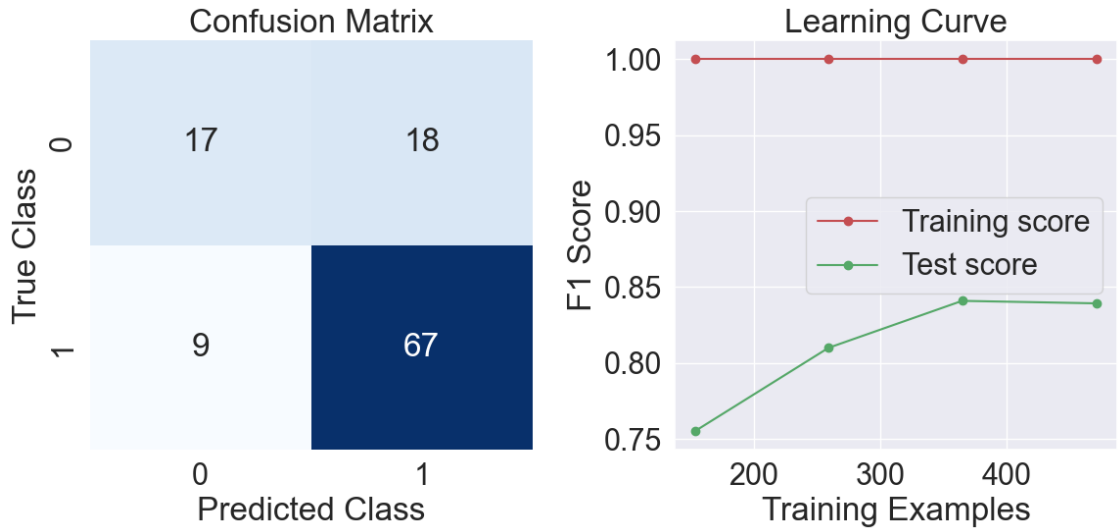
```
tuned RF-Classifier:
ROC-AUC for test data: 0.6836466165413534
Classification Report for test data:
              precision    recall  f1-score   support

           0       0.65      0.49      0.56        35
           1       0.79      0.88      0.83        76

    accuracy                           0.76       111
   macro avg       0.72      0.68      0.69       111
weighted avg       0.75      0.76      0.75       111

F1-Score for test data: 0.8322981366459626
```

Hier wurde die Vorhersage erneut mit einer Confusion Matrix dargestellt. Insgesamt hat das Modell

gute Vorhersagen getroffen.

Auch die Lernkurve wird angezeigt. Es ist zu erkennen, dass die Steigung tendenziell steigend ist. Mit einem größeren Datensatz könnte daher auch eine verbesserte Leistung des Modells erwartet werden.

[120]:
```
# Visualization
# Set the figure size for the visualization
plt.figure(figsize=(12, 6))

# Confusion Matrix for y_test and y_test_pred
cm = confusion_matrix(y_test, y_test_pred)

# Plot the confusion matrix
plt.subplot(1, 2, 1)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.title('Confusion Matrix')

# Define F1-Score as the scoring function for the learning curve
scorer = make_scorer(f1_score)

# Learning Curve
# Generate the learning curve
train_sizes, train_scores, test_scores = learning_curve(best_rf_model,␣
 ↪X_train_resampled, y_train_resampled, cv=5, scoring=scorer)

# Plot the learning curve
plt.subplot(1, 2, 2)
plt.plot(train_sizes, np.mean(train_scores, axis=1), 'o-', color="r",␣
 ↪label="Training score")
plt.plot(train_sizes, np.mean(test_scores, axis=1), 'o-', color="g",␣
 ↪label="Test score")
plt.xlabel('Training Examples')
plt.ylabel('F1 Score')
plt.title('Learning Curve')
plt.legend(loc="best")

# Adjust layout for better visualization
plt.tight_layout()
# show plot
plt.show()
```



Der Dummy-Classifier wird als einfaches Klassifikationsmodell verwendet, um als Basislinie zur Bewertung der Leistung anderer Modelle zu dienen. Es werden Vorhersagen basierend auf einfachen Regeln oder einer vorgegebenen Strategie generiert, ohne dass tatsächlich Muster oder Beziehungen in den Daten erlernt werden. Die Strategie 'stratified' des Dummy-Classifiers wird angewendet, um die Klassenverteilung im Trainingsdatensatz nachzuahmen. Die Leistung des Dummy-Classifiers zeigt eine relativ gleichmäßige Verteilung zwischen den Klassen, was zu einer niedrigen Genauigkeit von 51,35% und einem F1-Score von 0,597 führt. Die ROC-AUC liegt ebenfalls nahe bei 0,51, was darauf hinweist, dass der Classifier nicht in der Lage ist, zwischen den Klassen zu unterscheiden und zufällige Vorhersagen macht. Im Vergleich dazu zeigt das zuvor beschriebene, optimierte Modell eine deutlich verbesserte Leistung, wie durch eine höhere Genauigkeit, einen höheren F1-Score und eine höhere ROC-AUC gekennzeichnet ist. Dies deutet darauf hin, dass das optimierte Modell über die zufälligen Vorhersagen des Dummy-Classifiers hinausgeht und tatsächlich Muster und Zusammenhänge in den Daten lernt.

[121]:
```
# Dummy Classifier
# Initialize a Dummy Classifier with 'stratified' strategy and a fixed random␣
 ↪state
dummy_clf = DummyClassifier(strategy='stratified', random_state=7)

# Train the Dummy Classifier on the training data
dummy_clf.fit(X_train_resampled, y_train_resampled)

# Predictions on test data using the Dummy Classifier
y_test_dummy = dummy_clf.predict(X_test)

print("Dummy Classifier (stratified):")
# Output ROC-AUC for test data
print("ROC-AUC for test data:", roc_auc_score(y_test, y_test_dummy))
```

```
# Output classification report for test data
print("Classification Report for test data:")
print(classification_report(y_test, y_test_dummy))

# Calculate and output the F1-score for test data
f1 = f1_score(y_test, y_test_dummy)
print("F1-Score for test data:", f1)
```

```
Dummy Classifier (stratified):
ROC-AUC for test data: 0.5060150375939849
Classification Report for test data:
              precision    recall  f1-score   support

           0       0.32      0.49      0.39        35
           1       0.69      0.53      0.60        76

    accuracy                           0.51       111
   macro avg       0.51      0.51      0.49       111
weighted avg       0.57      0.51      0.53       111

F1-Score for test data: 0.5970149253731344
```

11 Vorhersage-Demo

Für die Vorhersage wurde eine Demonstration mit dem getunten Random Forest-Modell durchgeführt, das durch Gridsearch optimiert wurde. Die Vorhersage wurde mit dem Testdatensatz durchgeführt. Das Ergebnis zeigt, dass das Modell in 4 von 5 Fällen richtig lag, was darauf hindeutet, dass es in der Lage ist, eine gute Schätzung abzugeben.

[122]:
```
# Choose five random indices from the test dataset
random.seed(9)  # Set the random seed for reproducibility
indices = random.sample(range(len(X_test)), 5)

# Iterate over the selected indices and make predictions for each index
for index in indices:
    # Select the data for the current index
    X_sample = X_test.iloc[[index]]

    # Make the prediction
    prediction = best_rf_model.predict(X_sample)

    # Display the prediction and the actual value
    print("Index:", X_sample.index[0])
    print("Prediction:", prediction[0])
    print("Actual Value:", y_test.iloc[index])
    print("-------------------------")
```

```
Index: 584
Prediction: 0
Actual Value: 0
-------------------------
Index: 82
Prediction: 1
Actual Value: 0
-------------------------
Index: 311
Prediction: 1
Actual Value: 1
-------------------------
Index: 432
Prediction: 1
Actual Value: 1
-------------------------
Index: 565
Prediction: 1
Actual Value: 1
-------------------------
```