Credit Risk Assessment

December 1, 2024

1 Credit Risk Assessment

[766]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import metrics
from sklearn.metrics import confusion_matrix, classification_report,␣
↪roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score
from sklearn.model_selection import GridSearchCV
```

"Credit risk refers to the possibility that a borrower or counterparty will fail to meet their obligations as agreed in a loan or credit arrangement, resulting in financial loss to the lender, investor, or creditor. In other words, credit risk is the risk that a borrower will not repay their debt, either by failing to make timely payments or defaulting altogether.

This risk is most commonly associated with loans, bonds, and other forms of credit. Credit risk is a critical consideration for financial institutions, banks, investors, and any entity that extends credit to another party."

For this project, I will assess *credit risk* by predicting *loan status* (0 = non-default, 1 = default).

This project utilizes data visualization, data preprocessing, feature engineering, machine learning techniques, and credit risk assessment features, which are essential for financial institutions to make informed lending decisions and effectively manage credit risk.

1.1 Data collection

[772]:
```
df = pd.read_csv('credit_risk_dataset.csv')
df.head()
```

[772]:
```
   person_age  person_income person_home_ownership  person_emp_length  \
0          22          59000                  RENT              123.0
1          21           9600                   OWN                5.0
2          25           9600              MORTGAGE                1.0
3          23          65500                  RENT                4.0
4          24          54400                  RENT                8.0

  loan_intent loan_grade  loan_amnt  loan_int_rate  loan_status  \
0    PERSONAL          D      35000          16.02            1
1   EDUCATION          B       1000          11.14            0
2     MEDICAL          C       5500          12.87            1
3     MEDICAL          C      35000          15.23            1
4     MEDICAL          C      35000          14.27            1

   loan_percent_income cb_person_default_on_file  cb_person_cred_hist_length
0                 0.59                         Y                           3
1                 0.10                         N                           2
2                 0.57                         N                           3
3                 0.53                         N                           2
4                 0.55                         Y                           4
```

1.2 Feature Understanding and Preprocessing

person_age [Age]: Older borrowers may have more financial stability, while younger borrowers may have less established credit history. However, age alone might not be a strong indicator of default risk. **person_income** [Annual Income]: Higher income borrowers are less likely to default since they are more likely to afford repayments. However, income alone is not enough; it's also important to consider income relative to loan amount and other factors. **person_home_ownership** [Home ownership]: Homeowners may have more financial stability (they may have equity in a property), but renters may be more likely to default due to less asset security. **person_emp_length** [Employment length (in years)]: Longer employment history could indicate more stable financial circumstances and greater ability to repay loans, reducing default risk. **loan_intent** [Loan intent]: This reflects the purpose of the loan (e.g., debt consolidation, home improvement, etc.), which can affect default risk. Certain purposes (e.g., debt consolidation) may indicate higher credit risk, while others (e.g., home improvement) may indicate more financially responsible behavior. **loan_grade** [Loan grade]: This is often a rating (A, B, C, etc.) assigned to loans based on factors like the borrower's creditworthiness. A lower grade typically indicates higher risk. **loan_amnt** [Loan amount]: A higher loan amount may be riskier, especially if the borrower's income or other factors don't support such a large loan. **loan_int_rate** [Interest rate]: A higher interest rate often indicates higher risk, and is commonly associated with borrowers who are considered higher risk by lenders. **loan_status** [Loan status (0 is non default 1 is default)]: This is the target variable for classification — 1 for default and 0 for no default. This is the outcome you're trying to predict. **loan_percent_income** [Percent income]: If a large portion of a borrower's income is going toward the loan, this may indicate a higher chance of default due to financial strain. **cb_person_default_on_file** [Historical default]: This is a strong indicator of future risk. A previous default is one of the strongest predictors of future default risk. **cb_preson_cred_hist_length** [Credit history length]: A longer credit history generally suggests that a borrower has more experience managing debt, and thus may be less likely to default.

1.3 Objective

The objective for this project is to predict whether a borrower will default on a loan (i.e., loan status = 1) or not default (loan status = 0) based on a set of attributes such as age, income, employment history, loan amount, and past credit history.

This project aims to develop a predictive model that classifies loan applicants as high or low risk for defaulting on their loans. By accurately predicting defaults, the goal is to help the lending institution make more informed decisions, reducing financial loss while preserving customer satisfaction.

1.4 Data Preprocessing

Handling Missing Values

[785]:
```
print(df.isna().any())
```

```
person_age                    False
person_income                 False
person_home_ownership         False
person_emp_length              True
loan_intent                   False
loan_grade                    False
loan_amnt                     False
loan_int_rate                  True
loan_status                   False
loan_percent_income           False
cb_person_default_on_file     False
cb_person_cred_hist_length    False
dtype: bool
```

[787]:
```
df.nunique()
```

[787]:
```
person_age                      58
person_income                 4295
person_home_ownership            4
person_emp_length               36
loan_intent                      6
loan_grade                       7
loan_amnt                      753
loan_int_rate                  348
loan_status                      2
loan_percent_income             77
cb_person_default_on_file        2
```

```
cb_person_cred_hist_length      29
dtype: int64
```

[789]:
```python
df.isnull().sum()
```

[789]:
```
person_age                       0
person_income                    0
person_home_ownership            0
person_emp_length              895
loan_intent                      0
loan_grade                       0
loan_amnt                        0
loan_int_rate                 3116
loan_status                      0
loan_percent_income              0
cb_person_default_on_file        0
cb_person_cred_hist_length       0
dtype: int64
```

[791]:
```python
df['person_emp_length'].skew(), df['loan_int_rate'].skew()
```

[791]:
```
(2.6144551214595424, 0.2085503016908977)
```

The above tells us that 'person_emp_length' is significantly right-skewed, 'loan_int_rate' is approximately symmetric.

[794]:
```python
df['loan_int_rate'] = df['loan_int_rate'].fillna(df['loan_int_rate'].mean())
df['person_emp_length'] = df['person_emp_length'].
 ↪fillna(df['person_emp_length'].median())
```

Replaced NaN values in 'person_emp_length' with the median, as the data is skewed, and filled NaN values in 'loan_int_rate' with the mean, as the distribution is relatively symmetric.

[797]:
```python
df.head()
```

[797]:
```
   person_age  person_income person_home_ownership  person_emp_length  \
0          22          59000                  RENT              123.0
1          21           9600                   OWN                5.0
2          25           9600              MORTGAGE                1.0
3          23          65500                  RENT                4.0
4          24          54400                  RENT                8.0

  loan_intent loan_grade  loan_amnt  loan_int_rate  loan_status  \
0    PERSONAL          D      35000          16.02            1
1   EDUCATION          B       1000          11.14            0
2     MEDICAL          C       5500          12.87            1
3     MEDICAL          C      35000          15.23            1
4     MEDICAL          C      35000          14.27            1
```

```
   loan_percent_income  cb_person_default_on_file  cb_person_cred_hist_length
0                 0.59                          Y                           3
1                 0.10                          N                           2
2                 0.57                          N                           3
3                 0.53                          N                           2
4                 0.55                          Y                           4
```

[799]:
```python
df.isnull().sum()
```

[799]:
```
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_status                   0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
dtype: int64
```

One-Hot Encoding Categorical Variables

Encode Categorical Variables ('person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file')

[803]:
```python
# Define feature columns (X) and target variable (y)
X = df.drop('loan_status', axis=1)
y = df['loan_status']
```

[805]:
```python
categorical_features = ['person_home_ownership', 'loan_intent', 'loan_grade',␣
 ↪'cb_person_default_on_file']
numerical_features = ['person_age', 'person_income', 'person_emp_length',␣
 ↪'loan_amnt', 'loan_int_rate', 'loan_percent_income',␣
 ↪'cb_person_cred_hist_length']

# Apply OneHotEncoding to categorical columns & pass through the numerical␣
 ↪columns
# Normalize numerical columns later
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(sparse_output=False), categorical_features),
        ('num', 'passthrough', numerical_features)  # for numerical columns:␣
 ↪normalize using MinMaxScaler() or StandardScaler() later
    ])
```

```python
# Fit and transform the data
encoded_X = preprocessor.fit_transform(X)

# After encoding, get feature names
encoded_columns = preprocessor.get_feature_names_out()

# X is now encoded using the transformed data
encoded_X_df = pd.DataFrame(encoded_X, columns=encoded_columns)

print(encoded_X_df)
```

```
       cat__person_home_ownership_MORTGAGE  cat__person_home_ownership_OTHER  \
0                                      0.0                               0.0
1                                      0.0                               0.0
2                                      1.0                               0.0
3                                      0.0                               0.0
4                                      0.0                               0.0
…                                    …                               …
32576                                  1.0                               0.0
32577                                  1.0                               0.0
32578                                  0.0                               0.0
32579                                  1.0                               0.0
32580                                  0.0                               0.0

       cat__person_home_ownership_OWN  cat__person_home_ownership_RENT  \
0                                 0.0                              1.0
1                                 1.0                              0.0
2                                 0.0                              0.0
3                                 0.0                              1.0
4                                 0.0                              1.0
…                               …                              …
32576                             0.0                              0.0
32577                             0.0                              0.0
32578                             0.0                              1.0
32579                             0.0                              0.0
32580                             0.0                              1.0

       cat__loan_intent_DEBTCONSOLIDATION  cat__loan_intent_EDUCATION  \
0                                     0.0                         0.0
1                                     0.0                         1.0
2                                     0.0                         0.0
3                                     0.0                         0.0
4                                     0.0                         0.0
…                                   …                         …
32576                                 0.0                         0.0
32577                                 0.0                         0.0
32578                                 0.0                         0.0
```

```
32579                                0.0                           0.0
32580                                0.0                           0.0

       cat__loan_intent_HOMEIMPROVEMENT  cat__loan_intent_MEDICAL  \
0                                   0.0                       0.0
1                                   0.0                       0.0
2                                   0.0                       1.0
3                                   0.0                       1.0
4                                   0.0                       1.0
…                                 …                       …
32576                               0.0                       0.0
32577                               0.0                       0.0
32578                               1.0                       0.0
32579                               0.0                       0.0
32580                               0.0                       1.0

       cat__loan_intent_PERSONAL  cat__loan_intent_VENTURE  …  \
0                            1.0                       0.0  …
1                            0.0                       0.0  …
2                            0.0                       0.0  …
3                            0.0                       0.0  …
4                            0.0                       0.0  …
…                          …                       …  …
32576                        1.0                       0.0  …
32577                        1.0                       0.0  …
32578                        0.0                       0.0  …
32579                        1.0                       0.0  …
32580                        0.0                       0.0  …

       cat__loan_grade_G  cat__cb_person_default_on_file_N  \
0                    0.0                               0.0
1                    0.0                               1.0
2                    0.0                               1.0
3                    0.0                               1.0
4                    0.0                               0.0
…                  …                               …
32576                0.0                               1.0
32577                0.0                               1.0
32578                0.0                               1.0
32579                0.0                               1.0
32580                0.0                               1.0

       cat__cb_person_default_on_file_Y  num__person_age  num__person_income  \
0                                   1.0             22.0             59000.0
1                                   0.0             21.0              9600.0
2                                   0.0             25.0              9600.0
3                                   0.0             23.0             65500.0
4                                   1.0             24.0             54400.0
```

```
…                                …              …                 …
32576                          0.0           57.0           53000.0
32577                          0.0           54.0          120000.0
32578                          0.0           65.0           76000.0
32579                          0.0           56.0          150000.0
32580                          0.0           66.0           42000.0

       num__person_emp_length  num__loan_amnt  num__loan_int_rate  \
0                       123.0         35000.0               16.02
1                         5.0          1000.0               11.14
2                         1.0          5500.0               12.87
3                         4.0         35000.0               15.23
4                         8.0         35000.0               14.27
…                         …             …                   …
32576                     1.0          5800.0               13.16
32577                     4.0         17625.0                7.49
32578                     3.0         35000.0               10.99
32579                     5.0         15000.0               11.48
32580                     2.0          6475.0                9.99

       num__loan_percent_income  num__cb_person_cred_hist_length
0                          0.59                              3.0
1                          0.10                              2.0
2                          0.57                              3.0
3                          0.53                              2.0
4                          0.55                              4.0
…                           …                               …
32576                      0.11                             30.0
32577                      0.15                             19.0
32578                      0.46                             28.0
32579                      0.10                             26.0
32580                      0.15                             30.0

[32581 rows x 26 columns]
```

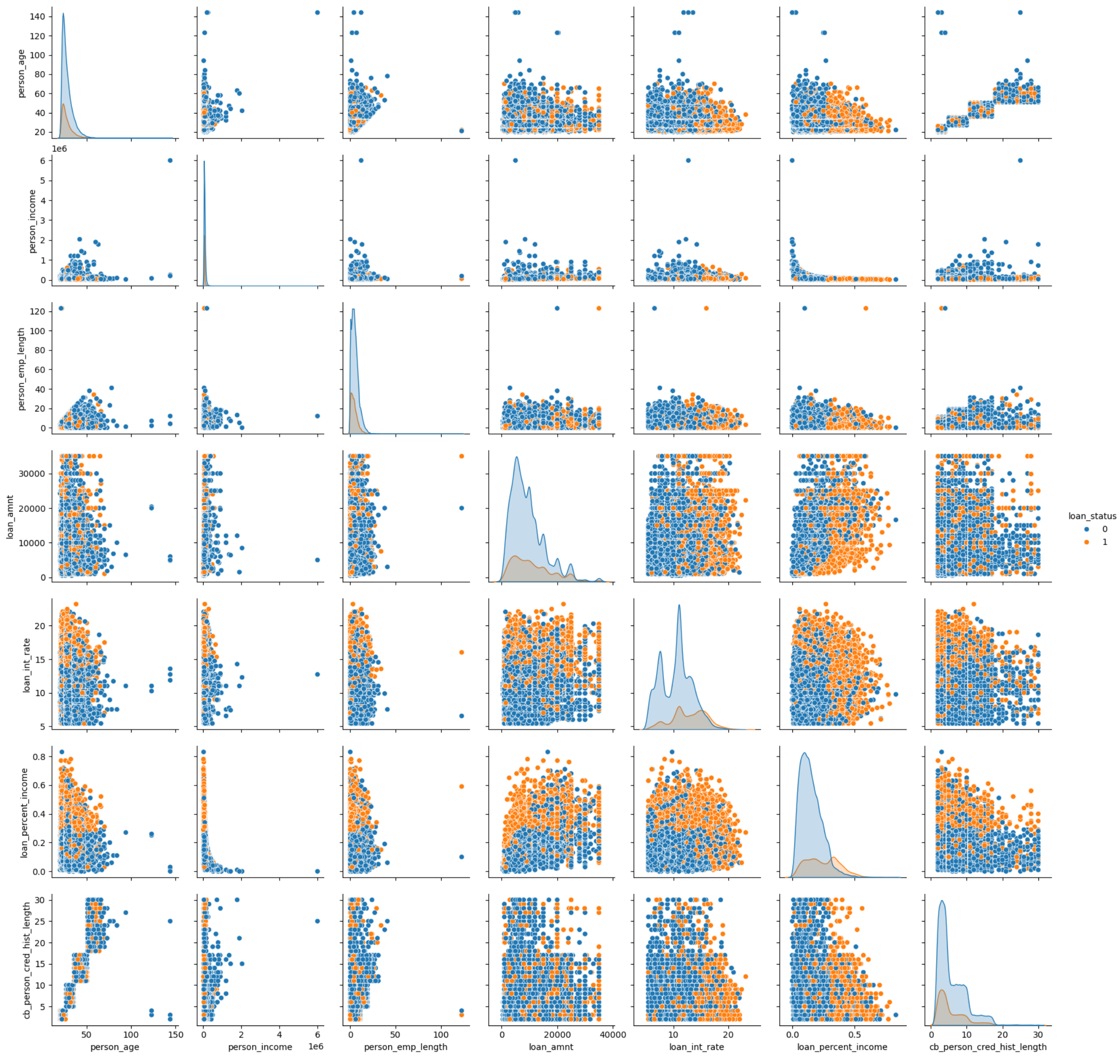
1.5 Exploratory Data Analysis (EDA)

Exploratory analysis to identify correlations and relationships between features and the target variable "loan_status"

Data Visualizations

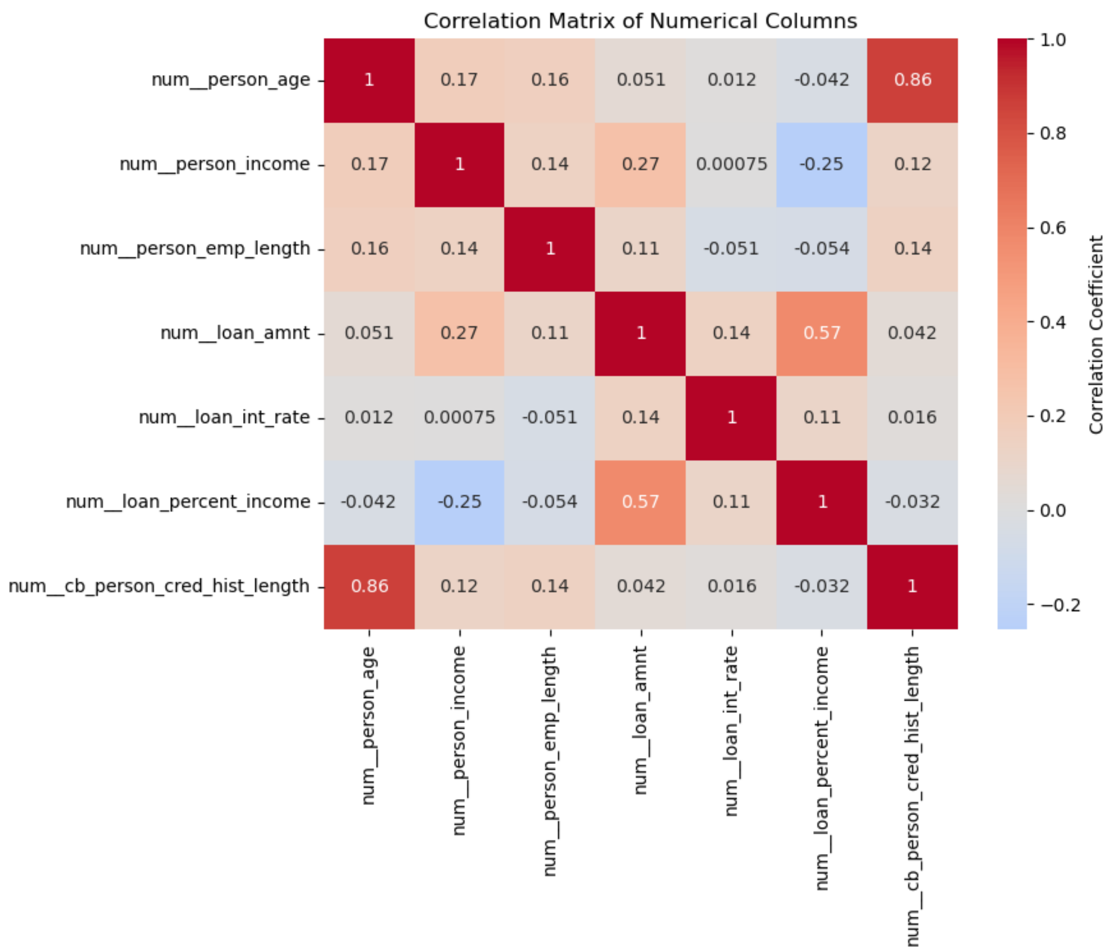
[810]:
```
sns.pairplot(df, hue="loan_status")
```

[810]:
```
<seaborn.axisgrid.PairGrid at 0x30eebde20>
```



Visualizing Numerical Columns with a Heatmap

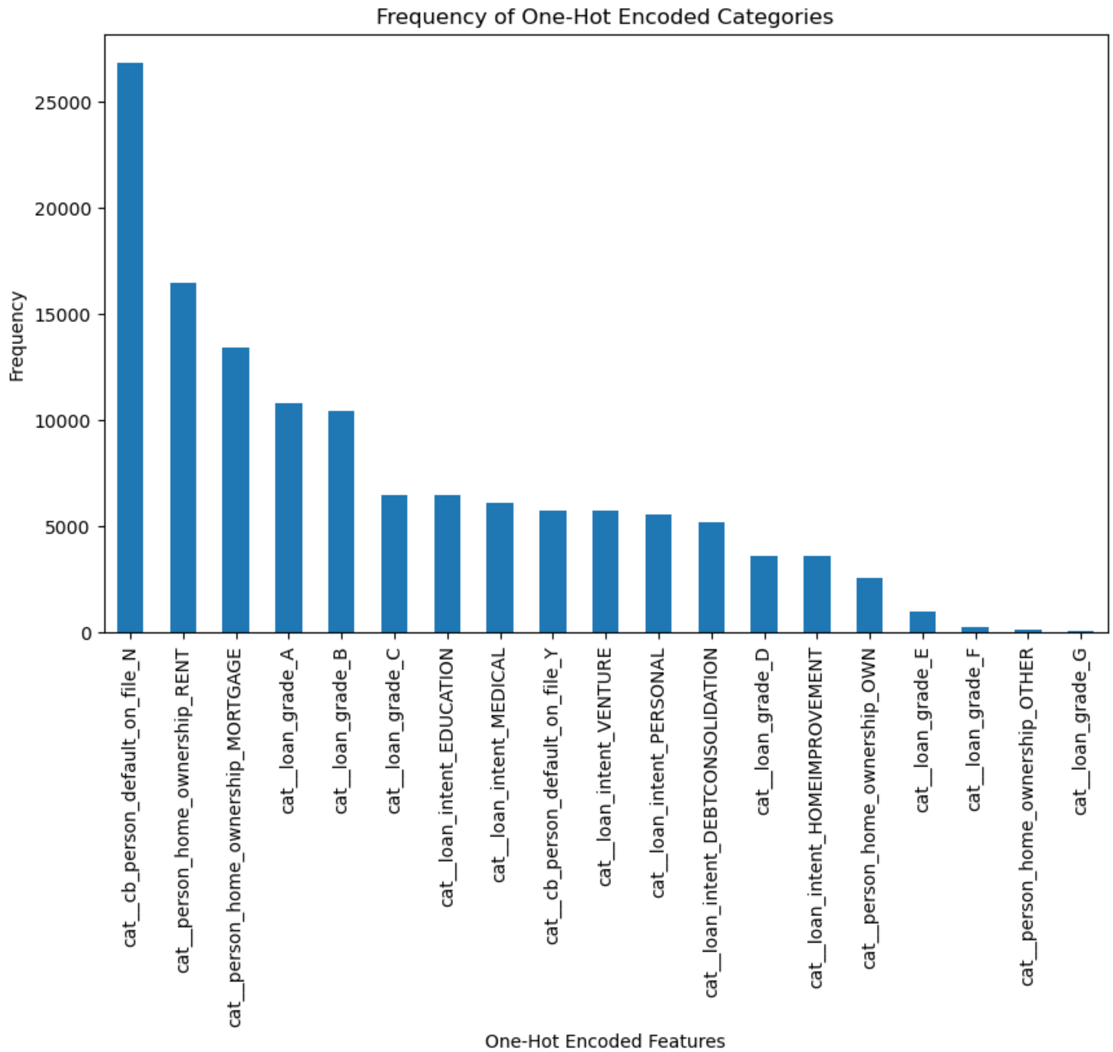
[813]:
```
num_columns = encoded_X_df.columns[encoded_X_df.columns.str.startswith('num')]
df_num = encoded_X_df[num_columns]

# Correlation matrix using numerical features
correlation_matrix = df_num.corr()

# Plotting the correlation matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
 ↪cbar_kws={'label': 'Correlation Coefficient'})
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
```



Two things to note: - 'num___cb_person_cred_hist_length' and 'num___person_age' are highly correlated - 'num___loan_amnt' and 'num___loan_percent_income' are moderately correlated

Visualizing One-Hot Encoded Categorical Columns With a Bar Plot

[817]:
```
cat_columns = encoded_X_df.columns[encoded_X_df.columns.str.startswith('cat')]
one_hot_encoded_columns = cat_columns

# Plot a graph
plt.figure(figsize=(10, 6))
encoded_X_df[one_hot_encoded_columns].sum().sort_values(ascending=False).
  ↪plot(kind='bar')
plt.title('Frequency of One-Hot Encoded Categories')
plt.ylabel('Frequency')
plt.xlabel('One-Hot Encoded Features')
plt.show()
```



A significantly large number of people in the dataset have no history of default.

1.6 Feature Engineering

[822]:
```
encoded_X_df.columns
```

[822]:
```
Index(['cat__person_home_ownership_MORTGAGE',
       'cat__person_home_ownership_OTHER', 'cat__person_home_ownership_OWN',
       'cat__person_home_ownership_RENT', 'cat__loan_intent_DEBTCONSOLIDATION',
       'cat__loan_intent_EDUCATION', 'cat__loan_intent_HOMEIMPROVEMENT',
       'cat__loan_intent_MEDICAL', 'cat__loan_intent_PERSONAL',
       'cat__loan_intent_VENTURE', 'cat__loan_grade_A', 'cat__loan_grade_B',
       'cat__loan_grade_C', 'cat__loan_grade_D', 'cat__loan_grade_E',
       'cat__loan_grade_F', 'cat__loan_grade_G',
       'cat__cb_person_default_on_file_N', 'cat__cb_person_default_on_file_Y',
```

```
 'num__person_age', 'num__person_income', 'num__person_emp_length',
 'num__loan_amnt', 'num__loan_int_rate', 'num__loan_percent_income',
 'num__cb_person_cred_hist_length'],
dtype='object')
```

Loan-to-Income Ratio indicates how financially strained a borrower might be. If someone has a very high loan amount relative to their income, they may be more likely to default.

[825]:
```
encoded_X_df['loan_income_ratio'] = encoded_X_df['num__loan_amnt'] /
 ↪encoded_X_df['num__person_income']
encoded_X_df['loan_income_ratio']
```

[825]:
```
0        0.593220
1        0.104167
2        0.572917
3        0.534351
4        0.643382
           …
32576    0.109434
32577    0.146875
32578    0.460526
32579    0.100000
32580    0.154167
Name: loan_income_ratio, Length: 32581, dtype: float64
```

Binning Personal Income using num___person_income

[828]:
```
cut = pd.qcut(encoded_X_df['num__person_income'], q=4, duplicates='drop')
print(cut)
```

```
0          (55000.0, 79200.0]
1         (3999.999, 38500.0]
2         (3999.999, 38500.0]
3          (55000.0, 79200.0]
4          (38500.0, 55000.0]
                 …
32576      (38500.0, 55000.0]
32577    (79200.0, 6000000.0]
32578      (55000.0, 79200.0]
32579    (79200.0, 6000000.0]
32580      (38500.0, 55000.0]
Name: num__person_income, Length: 32581, dtype: category
Categories (4, interval[float64, right]): [(3999.999, 38500.0] < (38500.0,
55000.0] < (55000.0, 79200.0] < (79200.0, 6000000.0]]
```

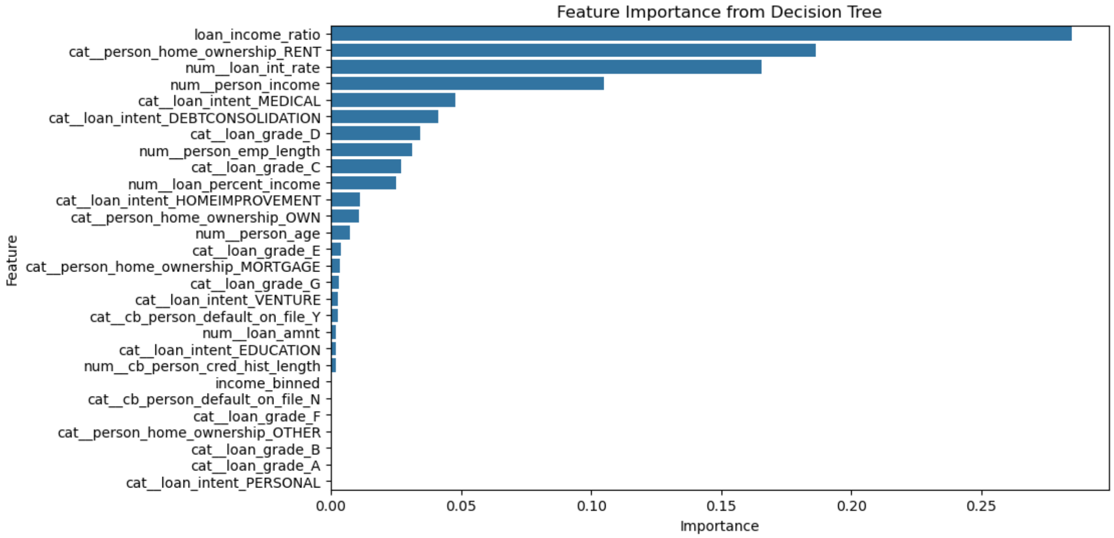
[830]:
```
bins = [3999.999, 38500, 55000, 79200, 6000000]
labels = [0, 1, 2, 3]
encoded_X_df['income_binned'] = pd.cut(encoded_X_df['num__person_income'],
 ↪bins=bins, labels=labels)
```

```
encoded_X_df['income_binned']
```

[830]:
```
0        2
1        0
2        0
3        2
4        1
        ..
32576    1
32577    3
32578    2
32579    3
32580    1
Name: income_binned, Length: 32581, dtype: category
Categories (4, int64): [0 < 1 < 2 < 3]
```

Feature Importance Using a Decision Tree

[833]:
```
model = DecisionTreeClassifier(random_state=42, max_depth=10,␣
 ↪min_samples_split=15, min_samples_leaf=10)
model.fit(encoded_X_df, y)

# Get the feature importances
feature_importances = model.feature_importances_

# Create a dataframe with feature importances
feature_importance_df = pd.DataFrame({
    'Feature': encoded_X_df.columns,
    'Importance': feature_importances
})

# Sort the features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance',␣
 ↪ascending=False)

# Plot the feature importances using feature_importance_df (sorted)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance from Decision Tree')
plt.show()

# Print the feature importances
print(feature_importance_df)
```



	Feature	Importance
26	loan_income_ratio	0.284861
3	cat__person_home_ownership_RENT	0.186398
23	num__loan_int_rate	0.165612
20	num__person_income	0.104810
7	cat__loan_intent_MEDICAL	0.047699
4	cat__loan_intent_DEBTCONSOLIDATION	0.041454
13	cat__loan_grade_D	0.034481
21	num__person_emp_length	0.031227
12	cat__loan_grade_C	0.027120
24	num__loan_percent_income	0.025053
6	cat__loan_intent_HOMEIMPROVEMENT	0.011141
2	cat__person_home_ownership_OWN	0.010663
19	num__person_age	0.007492
14	cat__loan_grade_E	0.003701
0	cat__person_home_ownership_MORTGAGE	0.003638
16	cat__loan_grade_G	0.003153
9	cat__loan_intent_VENTURE	0.002761
18	cat__cb_person_default_on_file_Y	0.002717
22	num__loan_amnt	0.001952
5	cat__loan_intent_EDUCATION	0.001890
25	num__cb_person_cred_hist_length	0.001779
27	income_binned	0.000328
17	cat__cb_person_default_on_file_N	0.000071
15	cat__loan_grade_F	0.000000
1	cat__person_home_ownership_OTHER	0.000000
11	cat__loan_grade_B	0.000000
10	cat__loan_grade_A	0.000000
8	cat__loan_intent_PERSONAL	0.000000

1.7 Building Credit Risk Models And Evaluating Model Performance

Now I'll build a model to predict credit risk using Neural Networks and Gradient Boosting.

Feature Selection and Normalization on Numerical Columns

[838]:
```
# Feature selection
# features = ['num__person_age', 'num__loan_percent_income', 'num__loan_amnt',␣
↪'loan_income_ratio', 'cat__person_home_ownership_RENT', 'num__loan_int_rate']
columns_to_drop = ['cat__loan_intent_PERSONAL', 'cat__loan_grade_A',␣
↪'cat__loan_grade_B', 'cat__person_home_ownership_OTHER',␣
↪'cat__loan_grade_F', 'cat__cb_person_default_on_file_N']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(encoded_X_df.
↪drop(columns=columns_to_drop, inplace=False), y, test_size=0.2,␣
↪random_state=42)

# Normalize the numerical columns using StandardScalar()
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

num_columns1 = X_train.columns[~X_train.columns.str.startswith('cat')]
X_train[num_columns1] = scaler.fit_transform(X_train[num_columns1])

num_columns2 = X_test.columns[~X_test.columns.str.startswith('cat')]
X_test[num_columns2] = scaler.fit_transform(X_test[num_columns2])
```

Initially chose 'num___person_age', 'num___loan_percent_income', 'num___loan_amnt', 'loan_income_ratio', 'cat___person_home_ownership_RENT', and 'num___loan_int_rate' as features. This is because - the correlation with 'num___person_age' and 'num___cb_person_cred_hist_length' was extremely high (removed 'num___cb_person_cred_hist_length' to avoid potential multicollinearity - 'num___loan_percent_income', 'num___loan_amnt' had moderate correlation - 'loan_income_ratio', 'cat___person_home_ownership_RENT', and 'num___loan_int_rate' had extremely high feature importance

However, the accuracy was lower than expected. So I tried dropping only 6 features that had very low feature importance instead.

[841]:
```
print(X_train.shape, X_test.shape)
```

```
(26064, 22) (6517, 22)
```

Neural Networks

[844]:
```
seq_model = Sequential()

# Input layer and the first hidden layer
```

```python
seq_model.add(Input(shape=(X_train.shape[1],)))
seq_model.add(Dense(units=64, activation='relu'))

# Another hidden layer
seq_model.add(Dense(units=32, activation='relu'))

# Output layer
seq_model.add(Dense(units=1, activation='sigmoid'))
```

[846]:

```python
# Compile the model
seq_model.compile(optimizer=Adam(learning_rate=0.00001, clipvalue=1.0),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[848]:

```python
# Train the model
history = seq_model.fit(X_train, y_train,
                    epochs=40,
                    batch_size=32,
                    validation_data=(X_test, y_test),
                    verbose=1)
```

```
Epoch 1/40
815/815              1s 588us/step -
accuracy: 0.5093 - loss: 0.7028 - val_accuracy: 0.7546 - val_loss: 0.6141
Epoch 2/40
815/815              0s 497us/step -
accuracy: 0.7912 - loss: 0.5840 - val_accuracy: 0.8024 - val_loss: 0.5362
Epoch 3/40
815/815              0s 502us/step -
accuracy: 0.8158 - loss: 0.5100 - val_accuracy: 0.8065 - val_loss: 0.4894
Epoch 4/40
815/815              0s 490us/step -
accuracy: 0.8180 - loss: 0.4674 - val_accuracy: 0.8151 - val_loss: 0.4593
Epoch 5/40
815/815              0s 496us/step -
accuracy: 0.8266 - loss: 0.4423 - val_accuracy: 0.8225 - val_loss: 0.4388
Epoch 6/40
815/815              0s 532us/step -
accuracy: 0.8369 - loss: 0.4170 - val_accuracy: 0.8289 - val_loss: 0.4236
Epoch 7/40
815/815              0s 518us/step -
accuracy: 0.8358 - loss: 0.4087 - val_accuracy: 0.8301 - val_loss: 0.4121
Epoch 8/40
815/815              0s 491us/step -
accuracy: 0.8430 - loss: 0.3946 - val_accuracy: 0.8329 - val_loss: 0.4030
Epoch 9/40
815/815              1s 682us/step -
accuracy: 0.8466 - loss: 0.3845 - val_accuracy: 0.8352 - val_loss: 0.3956
```

```
Epoch 10/40
815/815          0s 538us/step -
accuracy: 0.8431 - loss: 0.3867 - val_accuracy: 0.8389 - val_loss: 0.3896
Epoch 11/40
815/815          0s 563us/step -
accuracy: 0.8478 - loss: 0.3745 - val_accuracy: 0.8398 - val_loss: 0.3844
Epoch 12/40
815/815          1s 618us/step -
accuracy: 0.8484 - loss: 0.3723 - val_accuracy: 0.8412 - val_loss: 0.3800
Epoch 13/40
815/815          0s 540us/step -
accuracy: 0.8466 - loss: 0.3722 - val_accuracy: 0.8426 - val_loss: 0.3761
Epoch 14/40
815/815          0s 542us/step -
accuracy: 0.8520 - loss: 0.3638 - val_accuracy: 0.8450 - val_loss: 0.3726
Epoch 15/40
815/815          0s 553us/step -
accuracy: 0.8543 - loss: 0.3560 - val_accuracy: 0.8461 - val_loss: 0.3694
Epoch 16/40
815/815          0s 534us/step -
accuracy: 0.8515 - loss: 0.3615 - val_accuracy: 0.8485 - val_loss: 0.3665
Epoch 17/40
815/815          1s 609us/step -
accuracy: 0.8598 - loss: 0.3554 - val_accuracy: 0.8516 - val_loss: 0.3638
Epoch 18/40
815/815          0s 562us/step -
accuracy: 0.8593 - loss: 0.3502 - val_accuracy: 0.8532 - val_loss: 0.3613
Epoch 19/40
815/815          0s 590us/step -
accuracy: 0.8572 - loss: 0.3497 - val_accuracy: 0.8544 - val_loss: 0.3590
Epoch 20/40
815/815          0s 492us/step -
accuracy: 0.8599 - loss: 0.3457 - val_accuracy: 0.8562 - val_loss: 0.3569
Epoch 21/40
815/815          0s 494us/step -
accuracy: 0.8594 - loss: 0.3417 - val_accuracy: 0.8579 - val_loss: 0.3548
Epoch 22/40
815/815          0s 493us/step -
accuracy: 0.8611 - loss: 0.3452 - val_accuracy: 0.8584 - val_loss: 0.3529
Epoch 23/40
815/815          0s 493us/step -
accuracy: 0.8611 - loss: 0.3406 - val_accuracy: 0.8591 - val_loss: 0.3511
Epoch 24/40
815/815          0s 522us/step -
accuracy: 0.8607 - loss: 0.3454 - val_accuracy: 0.8604 - val_loss: 0.3493
Epoch 25/40
815/815          0s 572us/step -
accuracy: 0.8651 - loss: 0.3362 - val_accuracy: 0.8628 - val_loss: 0.3477
```

```
Epoch 26/40
815/815            1s 619us/step -
accuracy: 0.8655 - loss: 0.3393 - val_accuracy: 0.8659 - val_loss: 0.3461
Epoch 27/40
815/815            1s 608us/step -
accuracy: 0.8630 - loss: 0.3372 - val_accuracy: 0.8670 - val_loss: 0.3447
Epoch 28/40
815/815            0s 557us/step -
accuracy: 0.8705 - loss: 0.3297 - val_accuracy: 0.8671 - val_loss: 0.3432
Epoch 29/40
815/815            0s 600us/step -
accuracy: 0.8658 - loss: 0.3359 - val_accuracy: 0.8674 - val_loss: 0.3418
Epoch 30/40
815/815            0s 589us/step -
accuracy: 0.8677 - loss: 0.3316 - val_accuracy: 0.8676 - val_loss: 0.3405
Epoch 31/40
815/815            0s 592us/step -
accuracy: 0.8724 - loss: 0.3277 - val_accuracy: 0.8680 - val_loss: 0.3392
Epoch 32/40
815/815            0s 598us/step -
accuracy: 0.8687 - loss: 0.3300 - val_accuracy: 0.8700 - val_loss: 0.3379
Epoch 33/40
815/815            1s 740us/step -
accuracy: 0.8734 - loss: 0.3225 - val_accuracy: 0.8706 - val_loss: 0.3367
Epoch 34/40
815/815            0s 519us/step -
accuracy: 0.8707 - loss: 0.3293 - val_accuracy: 0.8717 - val_loss: 0.3356
Epoch 35/40
815/815            0s 485us/step -
accuracy: 0.8745 - loss: 0.3259 - val_accuracy: 0.8733 - val_loss: 0.3344
Epoch 36/40
815/815            0s 483us/step -
accuracy: 0.8760 - loss: 0.3210 - val_accuracy: 0.8748 - val_loss: 0.3333
Epoch 37/40
815/815            0s 563us/step -
accuracy: 0.8753 - loss: 0.3203 - val_accuracy: 0.8756 - val_loss: 0.3323
Epoch 38/40
815/815            1s 603us/step -
accuracy: 0.8756 - loss: 0.3250 - val_accuracy: 0.8757 - val_loss: 0.3312
Epoch 39/40
815/815            0s 568us/step -
accuracy: 0.8774 - loss: 0.3176 - val_accuracy: 0.8763 - val_loss: 0.3302
Epoch 40/40
815/815            1s 599us/step -
accuracy: 0.8814 - loss: 0.3170 - val_accuracy: 0.8766 - val_loss: 0.3292
```

[850]:

```python
# Evaluate the model on the test set
loss, accuracy = seq_model.evaluate(X_test, y_test)
print(f'Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}')
```

```
204/204              0s 370us/step -
accuracy: 0.8767 - loss: 0.3292
Test Loss: 0.3292, Test Accuracy: 0.8766
```

[852]:

```python
# Predict on the test data
y_pred_prob = seq_model.predict(X_test)

# Convert probabilities to binary predictions
y_pred = (y_pred_prob > 0.5).astype(int)

# Display the first 5 predictions
print("Predicted probabilities:", y_pred_prob[:5])
print("Predicted classes:", y_pred[:5])

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Classification report with precision, recall, and F1-score
cr = classification_report(y_test, y_pred)
print("Classification Report:")
print(cr)

# ROC-AUC score
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC Score: {roc_auc:.4f}")
```

```
204/204              0s 340us/step
Predicted probabilities: [[0.3346724 ]
 [0.2454338 ]
 [0.08979551]
 [0.8104967 ]
 [0.88028973]]
Predicted classes: [[0]
 [0]
 [0]
 [1]
 [1]]
Confusion Matrix:
[[4852  220]
 [ 584  861]]
Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.89      0.96      0.92      5072
              1       0.80      0.60      0.68      1445

       accuracy                           0.88      6517
      macro avg       0.84      0.78      0.80      6517
   weighted avg       0.87      0.88      0.87      6517

ROC-AUC Score: 0.8796
```

The overall accuracy of the Neural Network model is relatively high at 88%. However, the performance for class 1 (loan status = 1) is less satisfactory. The recall for class 1 is notably lower, with only 60% of class 1 instances being correctly identified. Given this, I will now build an alternative model using Gradient Boosting in an effort to improve performance.

Gradient Boosting

[857]:
```python
# Initialize and train the Gradient Boosting model
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
 ↪max_depth=3, random_state=42)
gbc.fit(X_train, y_train)
# Predict on the test set
y_pred = gbc.predict(X_test)
```

Evaluate the Model:

[860]:
```python
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

print(classification_report(y_test, y_pred))

y_pred_prob = gbc.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc:.4f}")
```

```
Accuracy: 0.9026
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      5072
           1       0.83      0.71      0.76      1445

    accuracy                           0.90      6517
   macro avg       0.87      0.83      0.85      6517
weighted avg       0.90      0.90      0.90      6517

ROC-AUC: 0.9106
```

To improve the model performance, I'll use GridSearchSV that optimizes hyperparameters.

[863]:

```python
# Hyperparameters to tune
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.001, 0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7, 9]
}

# Initialize GridSearchCV and fit the model on training data
grid_search = GridSearchCV(GradientBoostingClassifier(), param_grid, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best hyperparameters
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Use the above hyperparameters to build the model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print(classification_report(y_test, y_pred_best))
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators':
200}
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      5072
           1       0.84      0.74      0.78      1445

    accuracy                           0.91      6517
   macro avg       0.88      0.85      0.86      6517
weighted avg       0.91      0.91      0.91      6517
```

Now, the model achieves an improved accuracy of 91%, with a slightly higher recall for class 1, which has increased to 0.73.

As a result, by utilizing 22 features, including 'num___person_age', 'loan_income_ratio', 'income_binned', 'num___person_emp_length', and others, and building two models using Neural Networks and Gradient Boosting, I was able to accurately predict loan status with 88% and 91% accuracy, respectively.