CSE587A Data Intensive Computing - Project Phase 1 Report

Title: Data-Driven Loan Default Prediction and Financial Analysis

Team members: Juseung Lee, Venkata Amballa

1. Problem statement

Loan defaults raise significant issues in the financial sector that affect lenders, borrowers, and the overall economic environment. It results in financial losses for lenders and impedes the financial stability of borrowers. Moreover, loan defaults can also lead to a loss of confidence in the lending ecosystem and can disrupt credit access for individuals and businesses. The main objective of this project is to develop predictive models that can address these challenges by accurately predicting key aspects of the risk of loan issuance.

1.1. Background

The problem of loan default is significant because of not only its far-reaching impact on both lenders and borrowers but also its broader impact on the economy as a whole. In addition, loan default could have wide-ranging economic impacts, for example, instability in financial markets and reduced consumer spending. Considering these broad economic impacts and consequences for several financial sectors, it is significantly critical to effectively manage default risks in order to maintain the stability and sustainability of the financial system.

1.2. Potential contribution

This project aims to develop an accurate predictive model for loan default prediction using machine learning models, providing several potential contributions. It can provide valuable insights into patterns that can inform risk assessment for financial institutions, making it easier for financial institutions to make decisions about approval of loan applications. In addition, it can be used in customer segmentation in financial institutions as it can offer valuable insights into patterns and behaviors that can identify customer segments with different risk profiles. This enables financial institutions to manage risk effectively. Furthermore, it can offer insights into wide-ranging economic trends, enabling economists to forecast economic performance. Therefore, this project's contribution holds significant importance as it has a direct impact on the risk assessment strategies of financial institutions and provides crucial insights into broader economic trends, ultimately improving decision-making processes and strengthening financial stability.

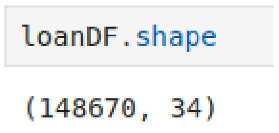
2. Data Source

2.1. Loan Default Dataset

The loan default dataset used in this project is available on the Kaggle [1].

2.2. Data Dimension

The loan default dataset has a total of 148670 rows and 34 columns.



```
loanDF.shape
```

```
(148670, 34)
```

2.3. Description

The loan default dataset provides a comprehensive overview of loan repayment patterns across various regions throughout the year 2019. This dataset is a significant resource for analyzing and understanding loan repayment behaviors. It includes several deterministic factors such as loan_purpose, loan_amount, property_value, and Credit_Score. In this dataset, the 'Status' feature indicates the loan status that value 1 identifies loan granted and value 0 identifies loan not granted. Using the data from the 'Status' feature, we can make assumptions about whether a loan applicant will default on the loan. For example, ‘Status=0’ indicates the loan is not sanctioned and the loan applicant will default on the loan.

2.4. Features and description

Here are features and their description from the loan default dataset.

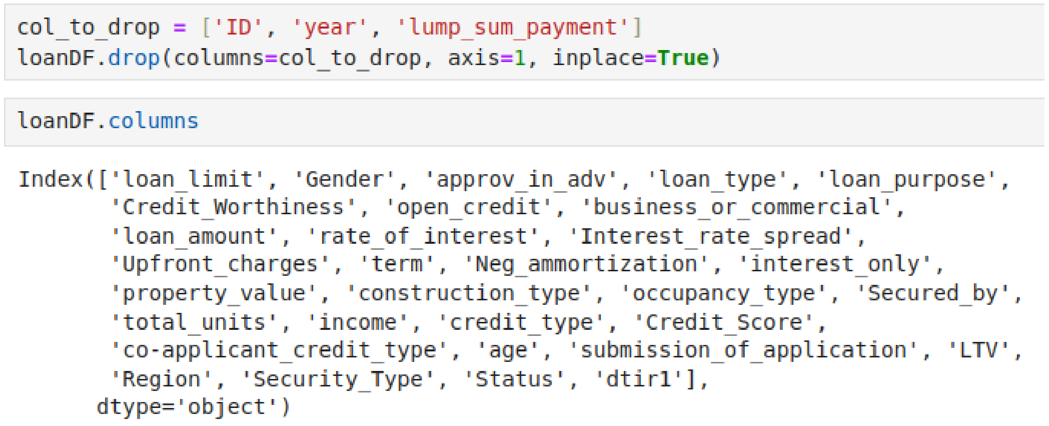
- ID : Unique identifier for each loan record.
- year : Year of loan issuance.
- loan_limit : Loan limit category.
- Gender : Loan applicant gender.
- approv_in_adv : Indicates whether the loan has been pre-approved.
- loan_type : Type of loan.
- loan_purpose : Purpose of the loan.
- Credit_Worthiness : The extent to which a person or company is considered suitable for receiving financial credit is often based on its reliability in repaying money in the past.
- open_credit : Indicates whether the applicant has other open credits.
- business_or_commercial : Whether the loan is for business or commercial.
- loan_amount : Total loan amount.
- rate_of_interest : Loan interest rate.
- Interest_rate_spread : Interest rate differential relative to a benchmark index.
- Upfront_charges : Initial charges for the loan.
- term : Loan duration in months.
- Neg_ammortization : Indicates whether there is negative amortization. Negative value indicates that the loan keeps accumulating when the paid amount is less than the Installment. [2]
- interest_only : Whether the loan allows for interest-only payments.

- lump_sum_payment : Indicates whether there is an option to pay in a single installment.
- property_value : Value of the property associated with the loan.
- construction_type : Type of construction of the property.
- occupancy_type : Type of property occupancy.
- Secured_by : Type of loan guarantee.
- total_units : Number of units related to the loan.
- income : Loan applicant's income.
- credit_type : Type of credit check used.
- Credit_Score : Applicant's credit score.
- co-applicant_credit_type : Type of credit check for co-applicants.
- age : Age range of the applicant.
- submission_of_application : Loan application submission method.
- LTV (Loan to Value): Ratio of loan amount to property value.
- Region : Geographic region of the loan.
- Security_Type : Type of loan security.
- Status : Loan Status (1 loan granted, 0 not granted)
- dtir1 (Debt to Income Ratio): Debt/income ratio of the applicant.

3. Data Cleaning/Processing

3.1. Dropping unwanted columns

In the dataset, there are three unnecessary columns: ‘ID’, ‘year’, and ‘lump_sum_payment’. The ‘ID’ column indicates each loan record and it is for a unique identifier, and this dataset includes only the year 2019. In addition, the ‘lump_sum_payment’ column indicates whether there is an option to pay in a single installment. We dropped these three unwanted columns.



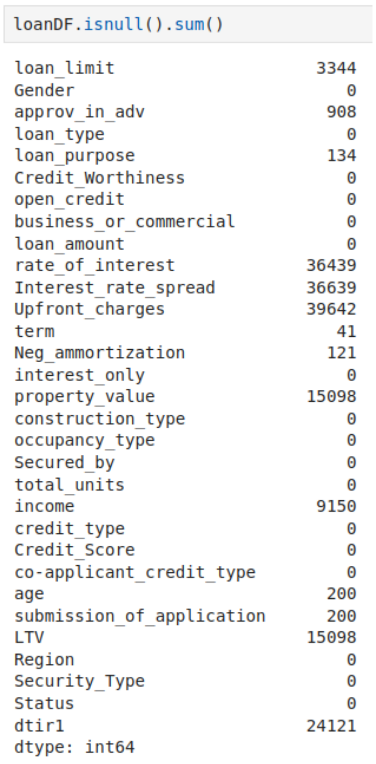
```
col_to_drop = ['ID', 'year', 'lump_sum_payment']
loanDF.drop(columns=col_to_drop, axis=1, inplace=True)
```

```
loanDF.columns
```

```
Index(['loan_limit', 'Gender', 'approv_in_adv', 'loan_type', 'loan_purpose',
       'Credit_Worthiness', 'open_credit', 'business_or_commercial',
       'loan_amount', 'rate_of_interest', 'Interest_rate_spread',
       'Upfront_charges', 'term', 'Neg_ammortization', 'interest_only',
       'property_value', 'construction_type', 'occupancy_type', 'Secured_by',
       'total_units', 'income', 'credit_type', 'Credit_Score',
       'co-applicant_credit_type', 'age', 'submission_of_application', 'LTV',
       'Region', 'Security_Type', 'Status', 'dtir1'],
      dtype='object')
```

3.2. Handling missing values

We identified the number of missing values in each column by using ‘loanDF.isnull().sum()’, and there are a total of 14 columns having missing values.



```
loanDF.isnull().sum()
```

```
loan_limit                    3344
Gender                           0
approv_in_adv                  908
loan_type                        0
loan_purpose                   134
Credit_Worthiness                0
open_credit                      0
business_or_commercial           0
loan_amount                      0
rate_of_interest             36439
Interest_rate_spread         36639
Upfront_charges              39642
term                            41
Neg_ammortization              121
interest_only                    0
property_value               15098
construction_type                0
occupancy_type                   0
Secured_by                       0
total_units                      0
income                        9150
credit_type                      0
Credit_Score                     0
co-applicant_credit_type         0
age                            200
submission_of_application      200
LTV                          15098
Region                           0
Security_Type                    0
Status                           0
dtir1                        24121
dtype: int64
```

We handled missing values both in the numerical columns and categorical columns. For the numerical columns, we filled the missing values with the mean of each corresponding column to three decimal places. For the categorical columns, we filled the missing values with the mode of each corresponding column.

```
# handling missing values in the numerical columns
loanDF[col_missingval_numeric] = loanDF[col_missingval_numeric].fillna(value=round(loanDF[col_missingval_numeric].mean(), 3))
```

```
# handling missing values in the categorical columns
loanDF[col_missingval_ctgy] = loanDF[col_missingval_ctgy].fillna(value=loanDF[col_missingval_ctgy].mode().iloc[0])
```

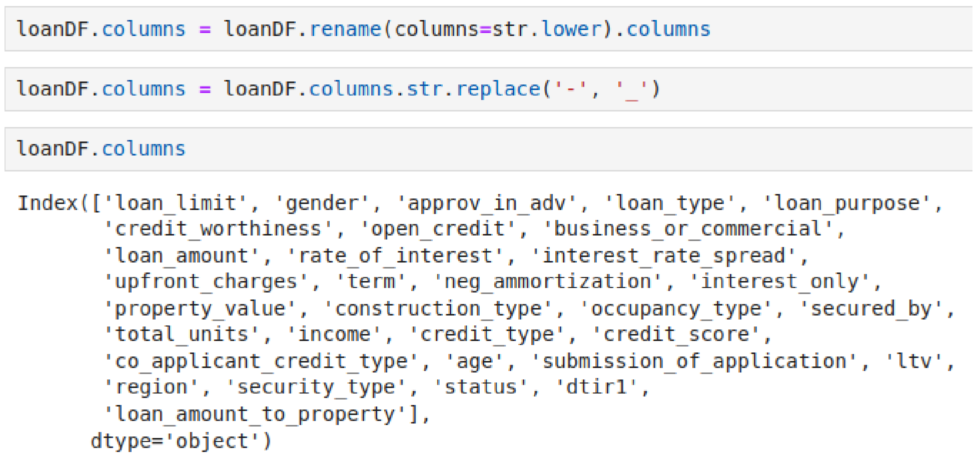
3.3. Feature Engineering

In this step, we divided ‘loan_amount’ by ‘property_value’ to create a new feature named ‘loan_amount_to_property’. It represents the ratio of ‘loan_amount’ to ‘property_value’. A possible reason for creating this feature is to inform the model that lower values of this ratio, have a higher likelihood of loan repayment. Also, in the event of a loan default, lower ratios suggest that the property can be used more effectively to cover the outstanding loan amount.

```
loanDF['loan_amount_to_property'] = loanDF['loan_amount']/loanDF['property_value']
```

3.4. Rename columns

To enhance the usability and consistency of the dataset, we renamed columns in two ways: (1) converted them to lowercase, and (2) replaced the hyphen with an underscore. This directly benefits our problem statement in several ways. First of all, it maintains consistency throughout the dataset if all the column names are in lowercase and follow the same format, making it easier for analysts to navigate and work with them. Moreover, renaming columns enhances compatibility and prevents potential issues during analysis since some programming languages or data analysis tools may have case sensitivity. Furthermore, it enhances the readability and interpretability of the dataset.



```
loanDF.columns = loanDF.rename(columns=str.lower).columns
```

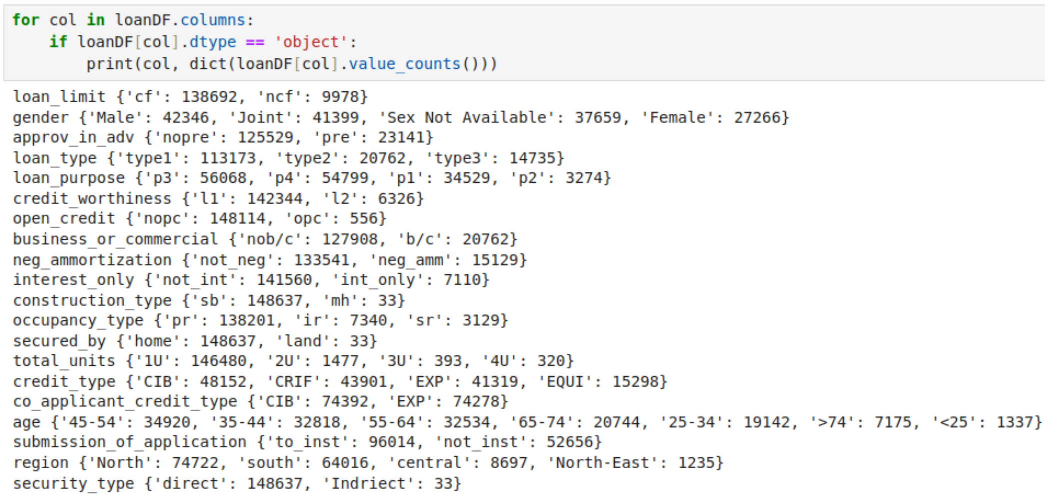
```
loanDF.columns = loanDF.columns.str.replace('-', '_')
```

```
loanDF.columns
```

```
Index(['loan_limit', 'gender', 'approv_in_adv', 'loan_type', 'loan_purpose',
       'credit_worthiness', 'open_credit', 'business_or_commercial',
       'loan_amount', 'rate_of_interest', 'interest_rate_spread',
       'upfront_charges', 'term', 'neg_ammortization', 'interest_only',
       'property_value', 'construction_type', 'occupancy_type', 'secured_by',
       'total_units', 'income', 'credit_type', 'credit_score',
       'co_applicant_credit_type', 'age', 'submission_of_application', 'ltv',
       'region', 'security_type', 'status', 'dtir1',
       'loan_amount_to_property'],
      dtype='object')
```

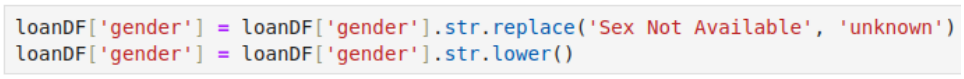
3.5. Formatting values in categorical columns

We identified values in categorical columns to ensure that the data format was consistent. In this step, we formatted values in several categorical columns in order to ensure consistency and compatibility. It prevents potential errors during analysis because since some programming languages or data analysis tools may have case sensitivity. In addition, it prevents potential issues while applying one-hot encoding techniques. We formatted values in five categorical columns: ‘gender’, ‘total_units’, ‘age’, ‘region’, and ‘security_type’. In ‘gender’, we replaced ‘Sex Not Available’ with ‘unknown’ and converted it to lowercase. In ‘total_units’, we removed ‘U’. In ‘age’, we replaced inequality signs with ‘over’ and ‘under’. In ‘region’, we replaced the hyphen with an underscore and converted it to lowercase. In ‘security_type’, we converted it to lowercase.

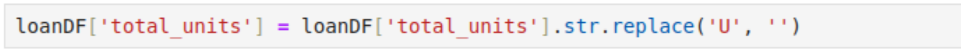


```
for col in loanDF.columns:
    if loanDF[col].dtype == 'object':
        print(col, dict(loanDF[col].value_counts()))
```

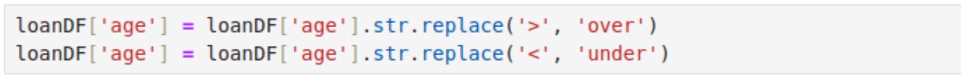
```
loan_limit {'cf': 138692, 'ncf': 9978}
gender {'Male': 42346, 'Joint': 41399, 'Sex Not Available': 37659, 'Female': 27266}
approv_in_adv {'nopre': 125529, 'pre': 23141}
loan_type {'type1': 113173, 'type2': 20762, 'type3': 14735}
loan_purpose {'p3': 56068, 'p4': 54799, 'p1': 34529, 'p2': 3274}
credit_worthiness {'l1': 142344, 'l2': 6326}
open_credit {'nopc': 148114, 'opc': 556}
business_or_commercial {'nob/c': 127908, 'b/c': 20762}
neg_ammortization {'not_neg': 133541, 'neg_amm': 15129}
interest_only {'not_int': 141560, 'int_only': 7110}
construction_type {'sb': 148637, 'mh': 33}
occupancy_type {'pr': 138201, 'ir': 7340, 'sr': 3129}
secured_by {'home': 148637, 'land': 33}
total_units {'1U': 146480, '2U': 1477, '3U': 393, '4U': 320}
credit_type {'CIB': 48152, 'CRIF': 43901, 'EXP': 41319, 'EQUI': 15298}
co_applicant_credit_type {'CIB': 74392, 'EXP': 74278}
age {'45-54': 34920, '35-44': 32818, '55-64': 32534, '65-74': 20744, '25-34': 19142, '>74': 7175, '<25': 1337}
submission_of_application {'to_inst': 96014, 'not_inst': 52656}
region {'North': 74722, 'south': 64016, 'central': 8697, 'North-East': 1235}
security_type {'direct': 148637, 'Indriect': 33}
```



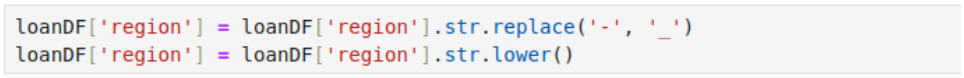
```
loanDF['gender'] = loanDF['gender'].str.replace('Sex Not Available', 'unknown')
loanDF['gender'] = loanDF['gender'].str.lower()
```



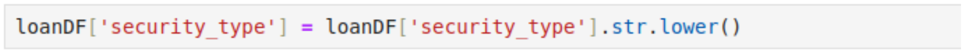
```
loanDF['total_units'] = loanDF['total_units'].str.replace('U', '')
```



```
loanDF['age'] = loanDF['age'].str.replace('>', 'over')
loanDF['age'] = loanDF['age'].str.replace('<', 'under')
```



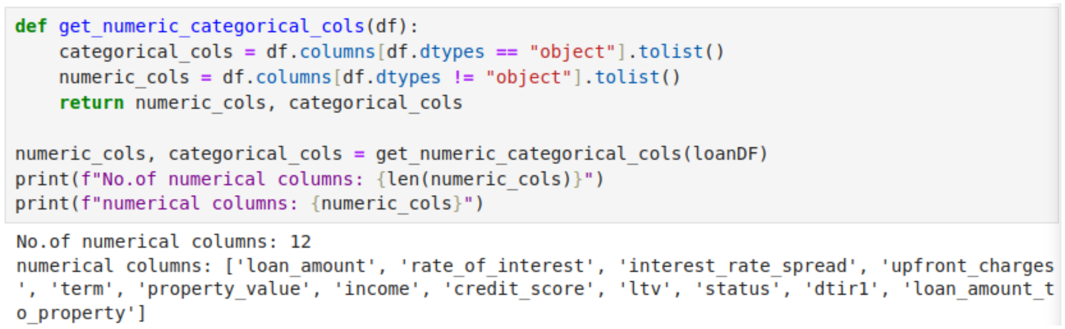
```
loanDF['region'] = loanDF['region'].str.replace('-', '_')
loanDF['region'] = loanDF['region'].str.lower()
```



```
loanDF['security_type'] = loanDF['security_type'].str.lower()
```

3.6. Handling outliers in numerical columns

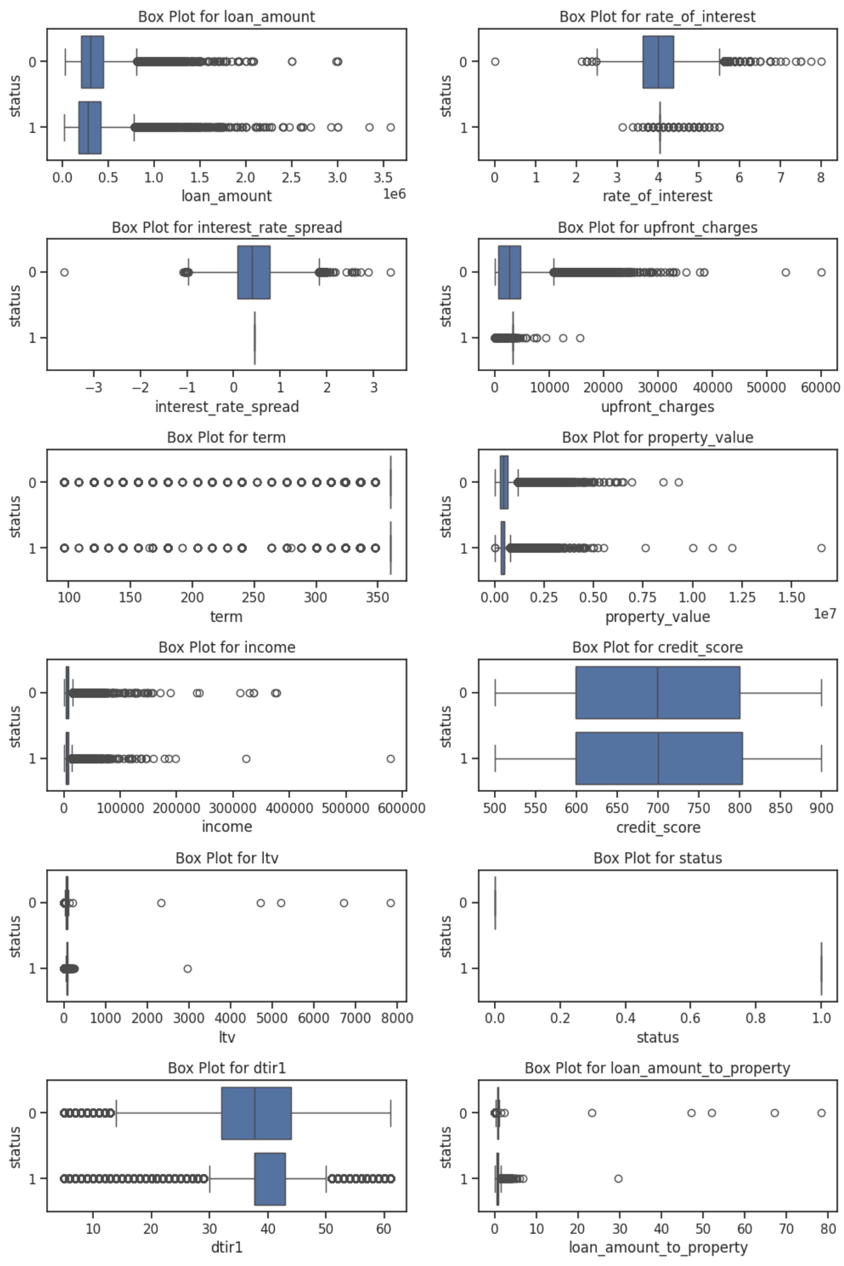
In our dataset, there is a total of eleven numerical columns, and we first plotted the numerical columns by using boxplots to identify outliers in them.



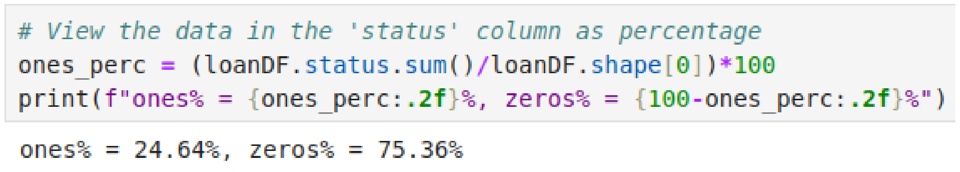
```
def get_numeric_categorical_cols(df):
    categorical_cols = df.columns[df.dtypes == "object"].tolist()
    numeric_cols = df.columns[df.dtypes != "object"].tolist()
    return numeric_cols, categorical_cols

numeric_cols, categorical_cols = get_numeric_categorical_cols(loanDF)
print(f"No.of numerical columns: {len(numeric_cols)}")
print(f"numerical columns: {numeric_cols}")
```

```
No.of numerical columns: 12
numerical columns: ['loan_amount', 'rate_of_interest', 'interest_rate_spread', 'upfront_charges
', 'term', 'property_value', 'income', 'credit_score', 'ltv', 'status', 'dtir1', 'loan_amount_t
o_property']
```

```
#Box Plot to analyse variability in the numerical columns

# Create subplots
fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(12, 18))
fig.subplots_adjust(hspace=0.6)  # Adjust vertical spacing

for ax, col in zip(np.ravel(axes), numeric_cols):
    sns.boxplot(x=col, y='status', data=loanDF, orient='h', ax=ax)
    ax.set_title(f'Box Plot for {col}')

# Show the plots
plt.show()
```

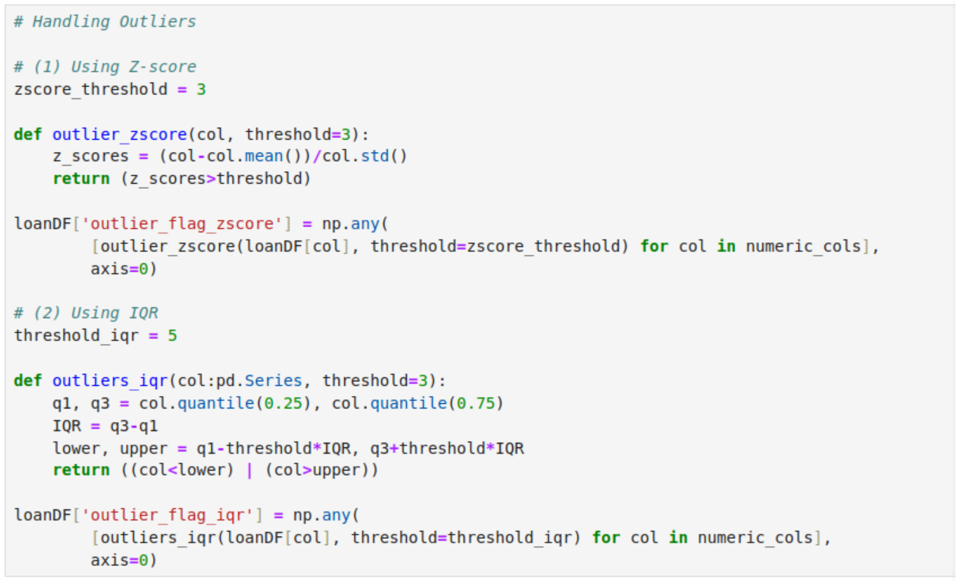


According to the box plots from the numerical columns, we were able to identify outliers in them, and there were several insights from them. First of all, the box plot for the ‘upfront_charges’ column is left-skewed, and there are several box plots with a single line without a box, which indicates that there are too many numbers with the same values. In addition, there are a lot of outliers in the ‘rate_of_interest’ column. We applied two approaches for handling outliers: (1) using Z-score, and (2) using IQR.

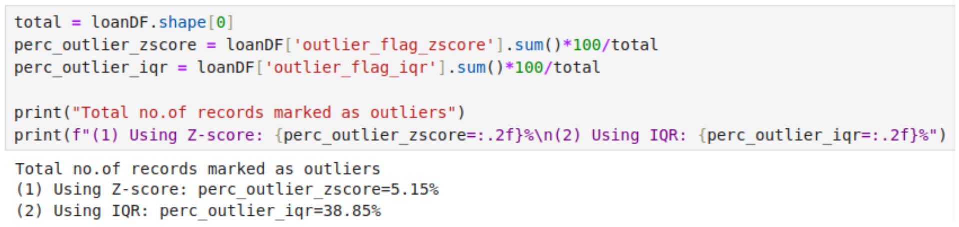


```python
# View the data in the 'status' column as percentage
ones_perc = (loanDF.status.sum()/loanDF.shape[0])*100
print(f"ones% = {ones_perc:.2f}%, zeros% = {100-ones_perc:.2f}%")
```

```
ones% = 24.64%, zeros% = 75.36%
```



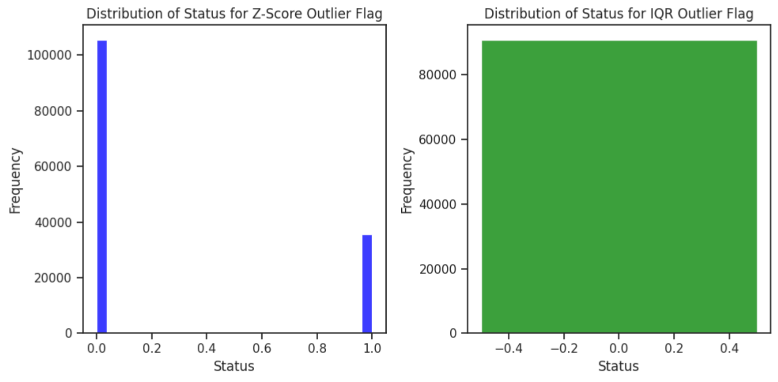
```python
# Handling Outliers

# (1) Using Z-score
zscore_threshold = 3

def outlier_zscore(col, threshold=3):
    z_scores = (col-col.mean())/col.std()
    return (z_scores>threshold)

loanDF['outlier_flag_zscore'] = np.any(
        [outlier_zscore(loanDF[col], threshold=zscore_threshold) for col in numeric_cols],
        axis=0)

# (2) Using IQR
threshold_iqr = 5

def outliers_iqr(col:pd.Series, threshold=3):
    q1, q3 = col.quantile(0.25), col.quantile(0.75)
    IQR = q3-q1
    lower, upper = q1-threshold*IQR, q3+threshold*IQR
    return ((col<lower) | (col>upper))

loanDF['outlier_flag_iqr'] = np.any(
        [outliers_iqr(loanDF[col], threshold=threshold_iqr) for col in numeric_cols],
        axis=0)
```



```python
total = loanDF.shape[0]
perc_outlier_zscore = loanDF['outlier_flag_zscore'].sum()*100/total
perc_outlier_iqr = loanDF['outlier_flag_iqr'].sum()*100/total

print("Total no.of records marked as outliers")
print(f"(1) Using Z-score: {perc_outlier_zscore=:.2f}%\n(2) Using IQR: {perc_outlier_iqr=:.2f}%")
```

```
Total no.of records marked as outliers
(1) Using Z-score: perc_outlier_zscore=5.15%
(2) Using IQR: perc_outlier_iqr=38.85%
```

Z-score marked records as outliers around 5% from the dataset, while IQR marked around 38%. Since the dataset has imbalanced data in the ‘status’ column, we viewed the distribution of ‘status’ for both of them by using histograms. It turns out that using IQR treats all the ‘status=1’ as outliers, so we used Z-score instead of IQR.



```
loanDF_cleaned = loanDF[loanDF['outlier_flag_zscore'] == 0].copy()
```

```
loanDF_cleaned.drop(['outlier_flag_iqr', 'outlier_flag_zscore'], axis=1, inplace=True)
```

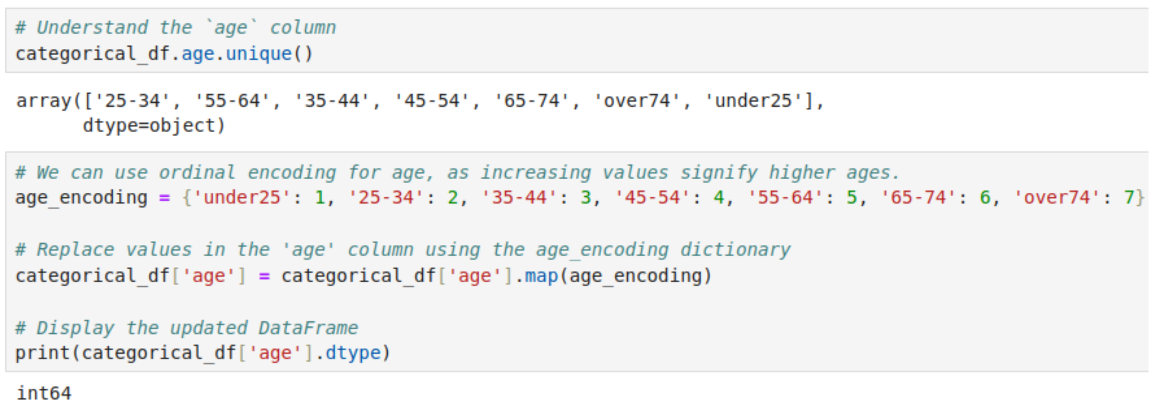
3.7. Normalizing numerical features

We applied robust scaling by using ‘RobustScaler()’ to the numerical features in the dataset in order to maintain all the features in a consistent range. Since we applied Z-score because of the imbalance ‘status’ column and the dataset still contains outliers, we applied robust scaling to make it robust to outliers and preserve the distribution shape of the data while scaling. In contrast, other scalers such as ‘StandardScaler()’ and ‘MinMaxScaler()’ are more suitable for datasets without outliers.

```
# Apply robust scaling to the features
rob_scaler = RobustScaler()
numerical_df_scaled = pd.DataFrame(rob_scaler.fit_transform(numerical_df), columns=numerical_df.columns)
```

3.8. Converting Datatype of Age

In this step, we converted the datatype of the ‘age’ column to an integer. We first used ordinal encoding for age as increasing values signify higher ages, and then replaced values in the ‘age’ column using the age_encoding dictionary.



```
# Understand the `age` column
categorical_df.age.unique()
```

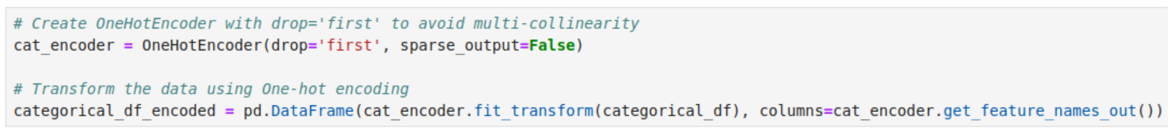
```
array(['25-34', '55-64', '35-44', '45-54', '65-74', 'over74', 'under25'],
      dtype=object)
```

```
# We can use ordinal encoding for age, as increasing values signify higher ages.
age_encoding = {'under25': 1, '25-34': 2, '35-44': 3, '45-54': 4, '55-64': 5, '65-74': 6, 'over74': 7}

# Replace values in the 'age' column using the age_encoding dictionary
categorical_df['age'] = categorical_df['age'].map(age_encoding)

# Display the updated DataFrame
print(categorical_df['age'].dtype)
```

```
int64
```

3.9. Categorical Encoding

In this step, we encoded categorical columns by using OneHotEncoder. We first created a OneHotEncoder with drop=‘first’ to avoid multi-collinearity, and then we transformed the data using one-hot encoding.



```
# Create OneHotEncoder with drop='first' to avoid multi-collinearity
cat_encoder = OneHotEncoder(drop='first', sparse_output=False)

# Transform the data using One-hot encoding
categorical_df_encoded = pd.DataFrame(cat_encoder.fit_transform(categorical_df), columns=cat_encoder.get_feature_names_out())
```

3.10. Concat the numerical and categorical dataframes

After normalizing numerical columns and encoding categorical columns, we concatenated the numerical and categorical dataframes.

```
loanDF_preprocessed = pd.concat([numerical_df_scaled, categorical_df_encoded], axis=1)
```

```
loanDF_preprocessed.columns
```

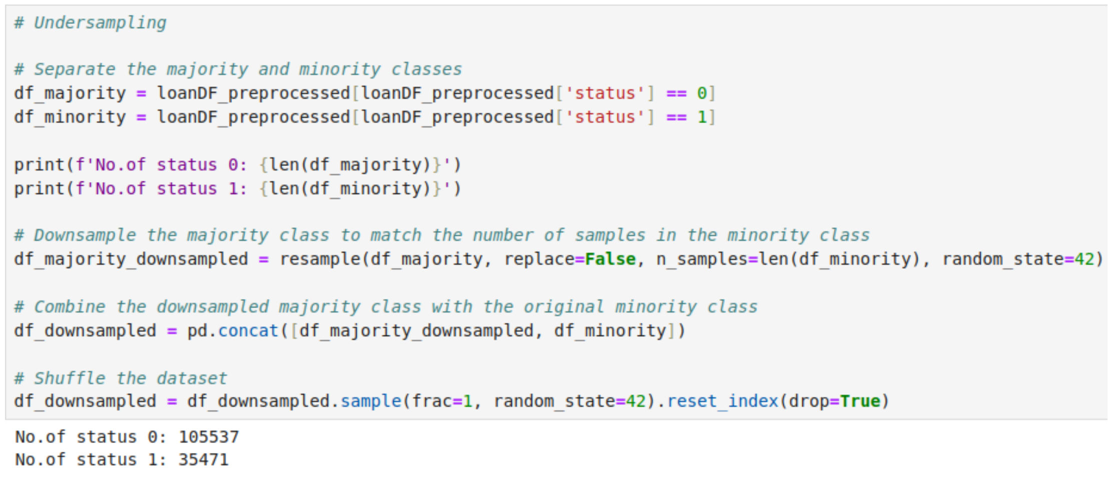
```
Index(['loan_amount', 'rate_of_interest', 'interest_rate_spread',
       'upfront_charges', 'term', 'property_value', 'income', 'credit_score',
       'ltv', 'status', 'dtir1', 'loan_amount_to_property', 'loan_limit_ncf',
       'gender_joint', 'gender_male', 'gender_unknown', 'approv_in_adv_pre',
       'loan_type_type2', 'loan_type_type3', 'loan_purpose_p2',
       'loan_purpose_p3', 'loan_purpose_p4', 'credit_worthiness_l2',
       'open_credit_opc', 'business_or_commercial_nob/c',
       'neg_ammortization_not_neg', 'interest_only_not_int',
       'construction_type_sb', 'occupancy_type_pr', 'occupancy_type_sr',
       'secured_by_land', 'total_units_2', 'total_units_3', 'total_units_4',
       'credit_type_CRIF', 'credit_type_EQUI', 'credit_type_EXP',
       'co_applicant_credit_type_EXP', 'age_2', 'age_3', 'age_4', 'age_5',
       'age_6', 'age_7', 'submission_of_application_to_inst', 'region_north',
       'region_north_east', 'region_south', 'security_type_indriect'],
      dtype='object')
```

```
loanDF_preprocessed.shape
```

```
(141008, 49)
```

3.11. Handling Imbalanced Dataset

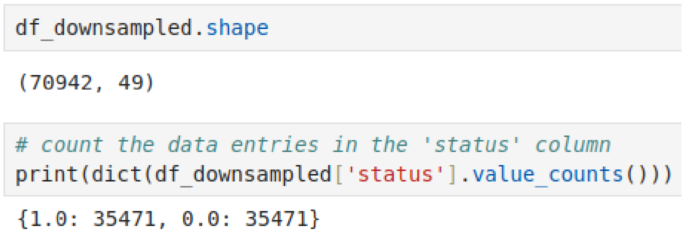
In order to balance the dataset, we can employ different approaches: (1) oversampling, and (2) undersampling. Alternatively, if N represents the total number of records in the given dataset and the total number of records in the minority class $N_{minority}$, we can divide the dataset into $\frac{N}{N_{minority}}$ groups (G). Each group can be paired with the minority(resulting in equal target distributions of size "$N_{minority} + G$") can be used to train separate classifiers as part of a meta-model. Since identifying the best approach is impossible at this stage, we will adapt this step and choose an approach accordingly during the modeling stage. In this step, we employed undersampling to handle the imbalanced dataset.



```python
# Undersampling

# Separate the majority and minority classes
df_majority = loanDF_preprocessed[loanDF_preprocessed['status'] == 0]
df_minority = loanDF_preprocessed[loanDF_preprocessed['status'] == 1]

print(f'No.of status 0: {len(df_majority)}')
print(f'No.of status 1: {len(df_minority)}')

# Downsample the majority class to match the number of samples in the minority class
df_majority_downsampled = resample(df_majority, replace=False, n_samples=len(df_minority), random_state=42)

# Combine the downsampled majority class with the original minority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])

# Shuffle the dataset
df_downsampled = df_downsampled.sample(frac=1, random_state=42).reset_index(drop=True)
```

```
No.of status 0: 105537
No.of status 1: 35471
```

We separated the majority and minority classes based on the ‘status’ value (zero and one): the majority class is for the value zero and the minority class is for the value one in the ‘status’ column. Then, we downsampled the majority class to match the number of samples in the minority class and combined the downsampled majority class with the original minority class [3]. Therefore, there are balanced data in the ‘status’ column as we have the same number of data entries in the ‘status’ column.



```python
df_downsampled.shape
```

```
(70942, 49)
```

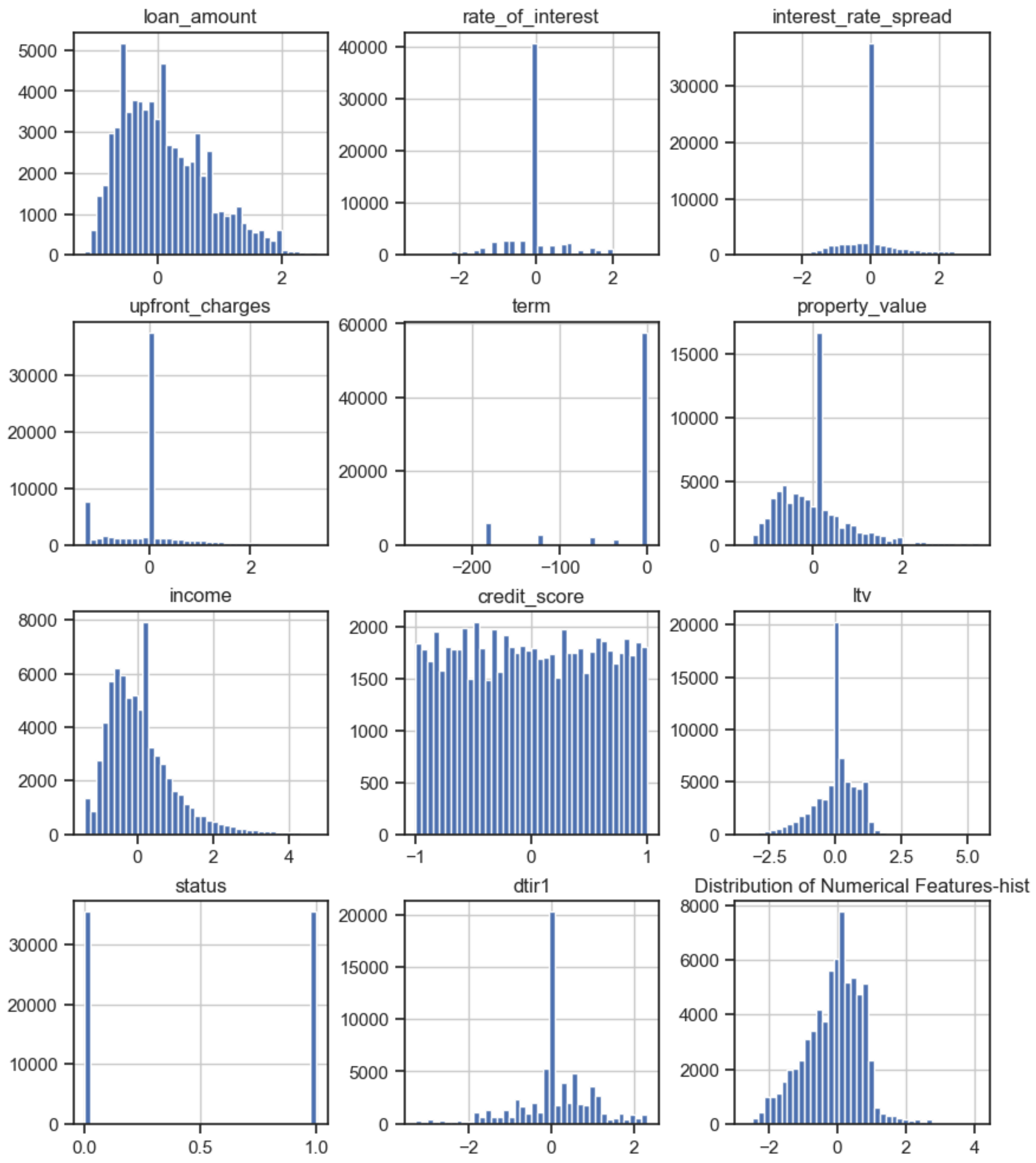
```python
# count the data entries in the 'status' column
print(dict(df_downsampled['status'].value_counts()))
```

```
{1.0: 35471, 0.0: 35471}
```

4. Exploratory Data Analysis (EDA)

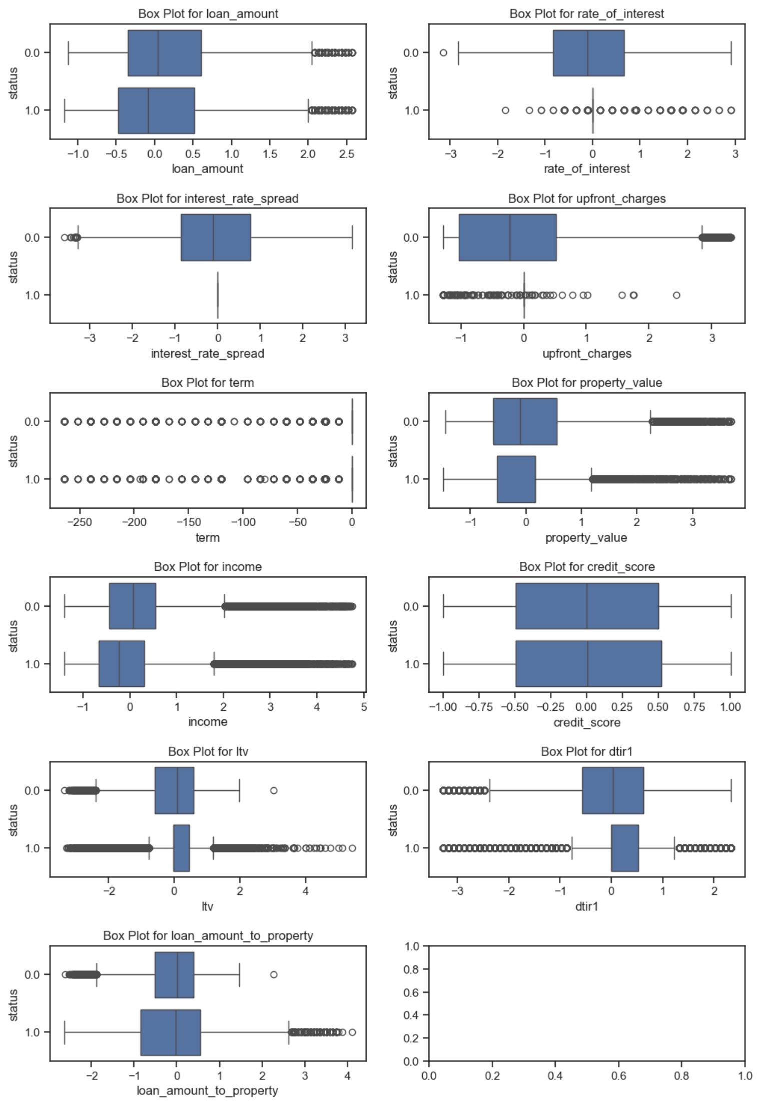
4.1. Analyzing numerical features distribution

We plotted a histogram for all numerical features as below. It can be observed that the distribution of ‘loan_amount’ and ‘income’ is left-skewed, indicating that the majority of ‘loan_amount’ is small. Moreover, the ‘credit_score’ appears to be relatively uniform across the dataset, while the ‘debt-to-income ratio (DTIR)’ is primarily centered around 0.



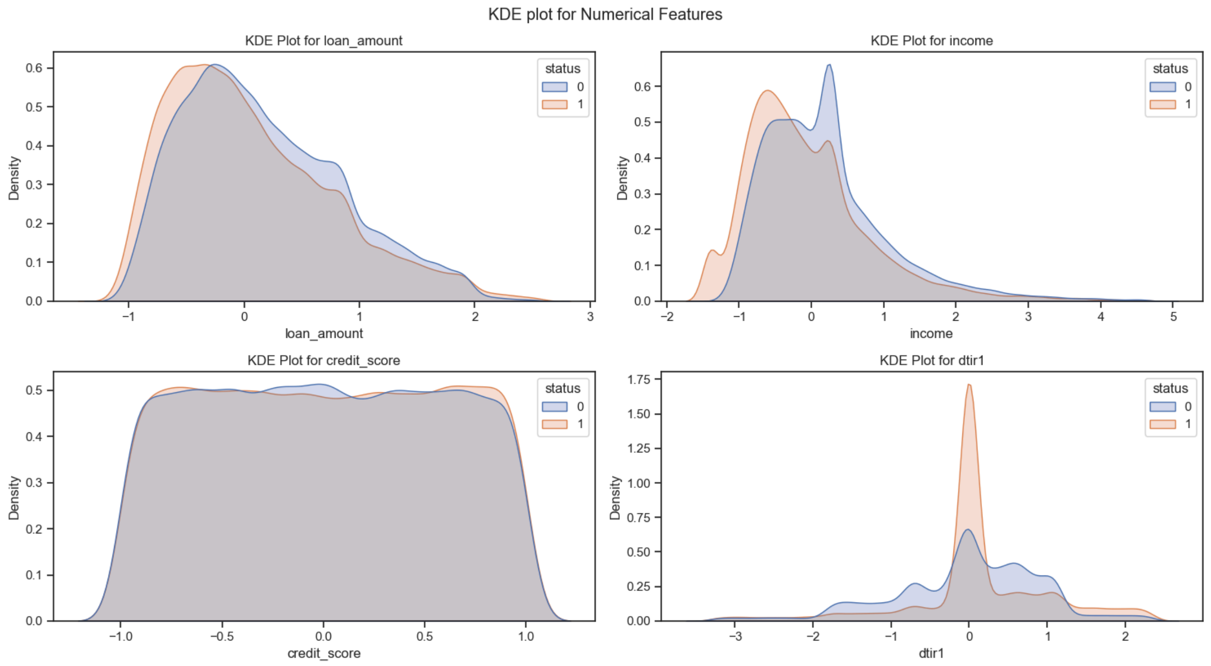
4.2. Analyzing the Outlier Distribution using BoxPlot

This is carried out in the numerical feature preprocessing step. Each numerical feature's distribution is shown in the box plot below. There is a slight left skew in features like income, upfront charges, and property values. Other characteristics, such as interest rate and credit score, on the other hand, are primarily centered around uniform distributions. Notably, there is no discernible difference between loans that have been sanctioned and those that have not in terms of interest rate spread. Moreover, there are a lot of outliers in the data, as indicated by the box plot.



4.3. Analyzing the density of numerical features by status

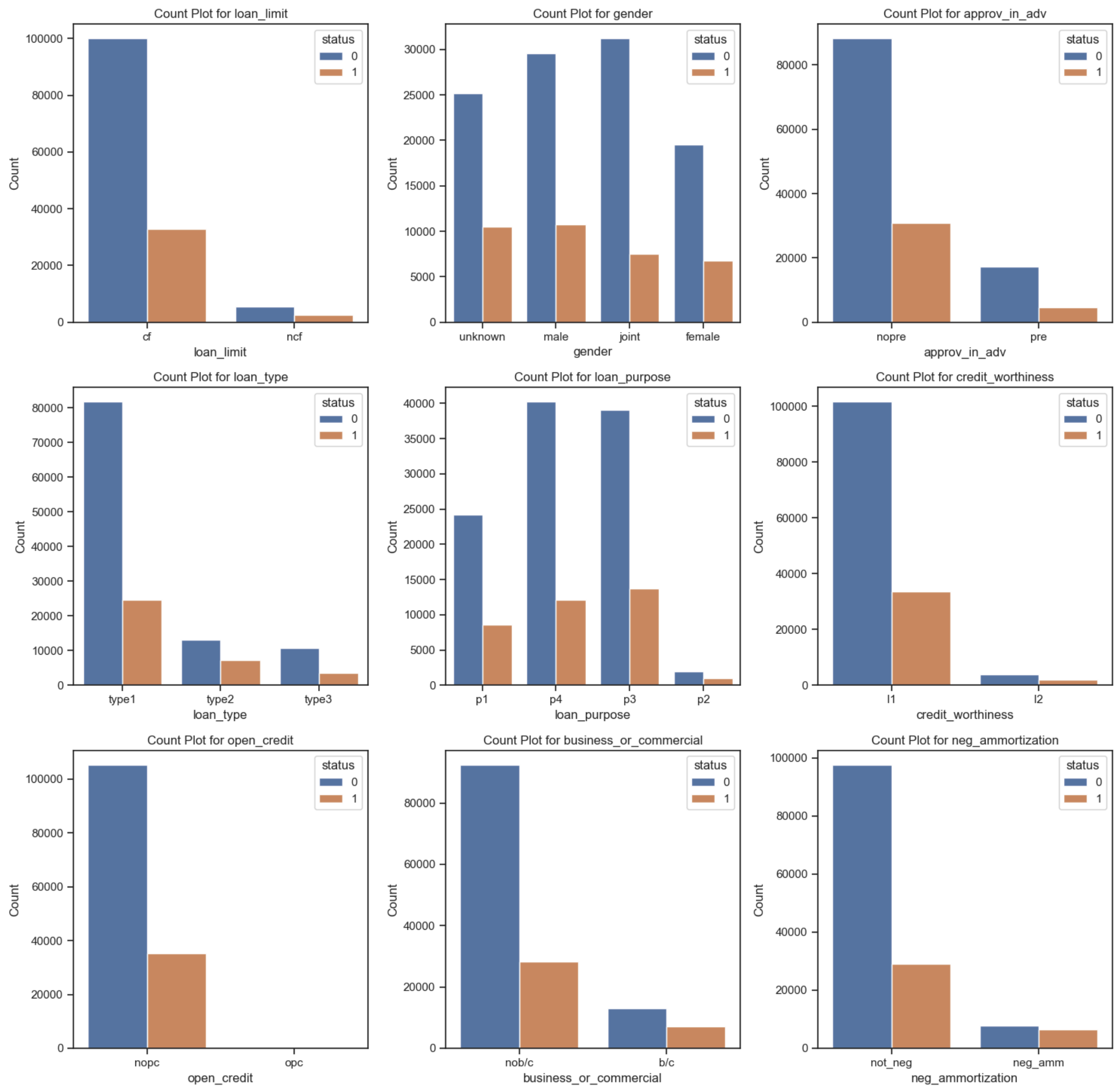
The KDE plot for ‘loan_amount’ is skewed towards the left, indicating that the majority of the loans are smaller. This observation is consistent with the ‘income’ plot below, which also exhibits a left skewness. Furthermore, the KDE plot for the ‘Debt-to-Income Ratio (DTIR)’ suggests that values near zero have the highest loan approvals.



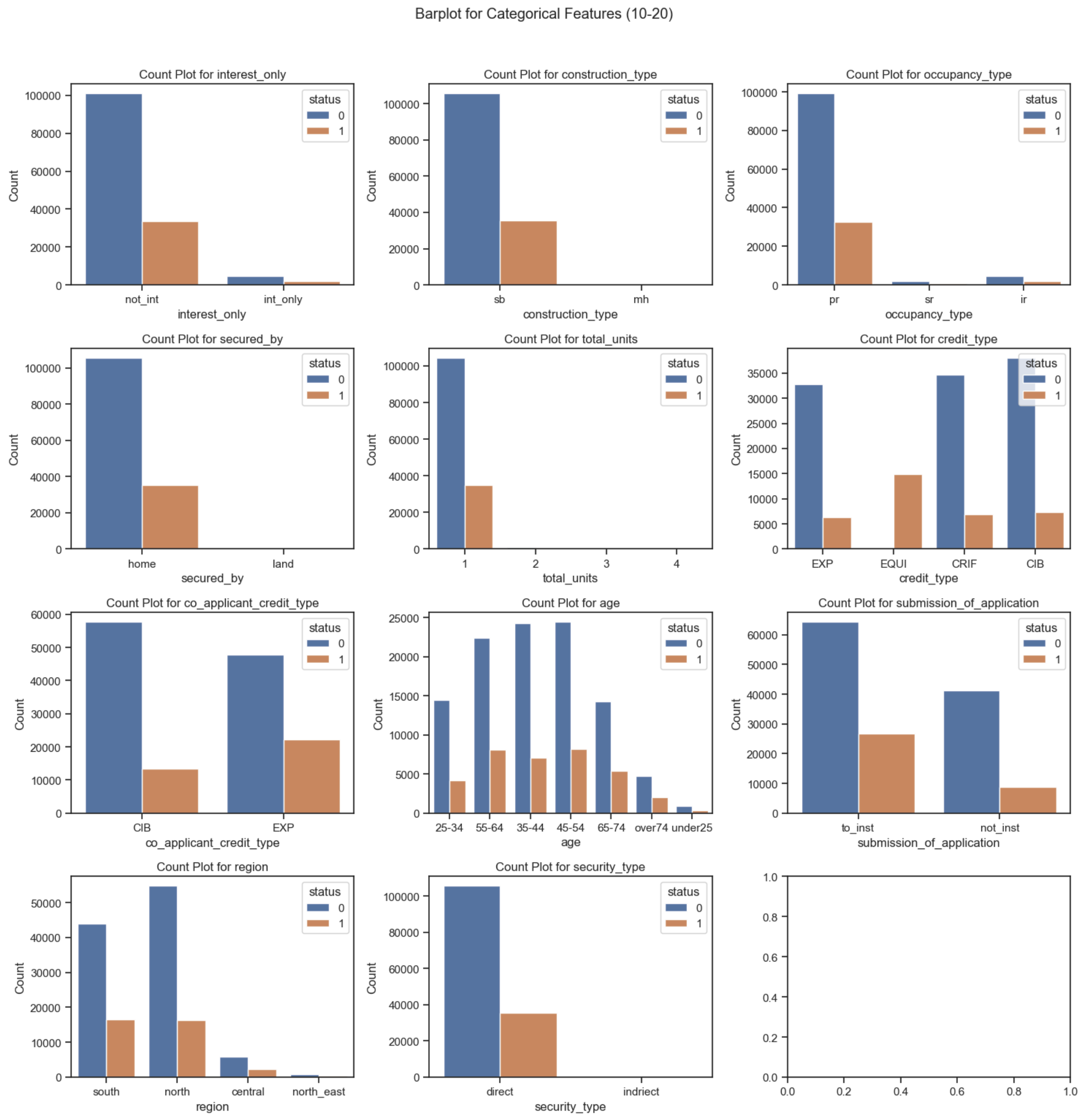
4.4. Analysing the frequency of categorical features by status

The graph below represents the frequency of occurrence by status for all categorical features in the non-downsampled dataset. As there are imbalanced data in the ‘status’ column, status=0 has a high frequency of occurrence of all categorical features. In the first nine bar plots for the categorical features(1-9), several features have large distributional differences between categories in the respective feature, such as ‘loan_limit’, ‘approv_in_adv’, ‘credit_worthiness’, ‘open_credit’, ‘bussiness_or_commercial’, and ‘neg_ammortization’. In the rest of the eleven bar plots for the categorical features(10-20), several features have large distributional differences between categories in the respective feature, such as ‘interest_only’, ‘construction_type’, ‘occupancy_type’, ‘secured_by’, ‘total_units’, and ‘security_type’. In the ‘credit_type’ feature, all credit type for ‘EQUI’ represents status=1.

Barplot for Categorical Features (1-9)



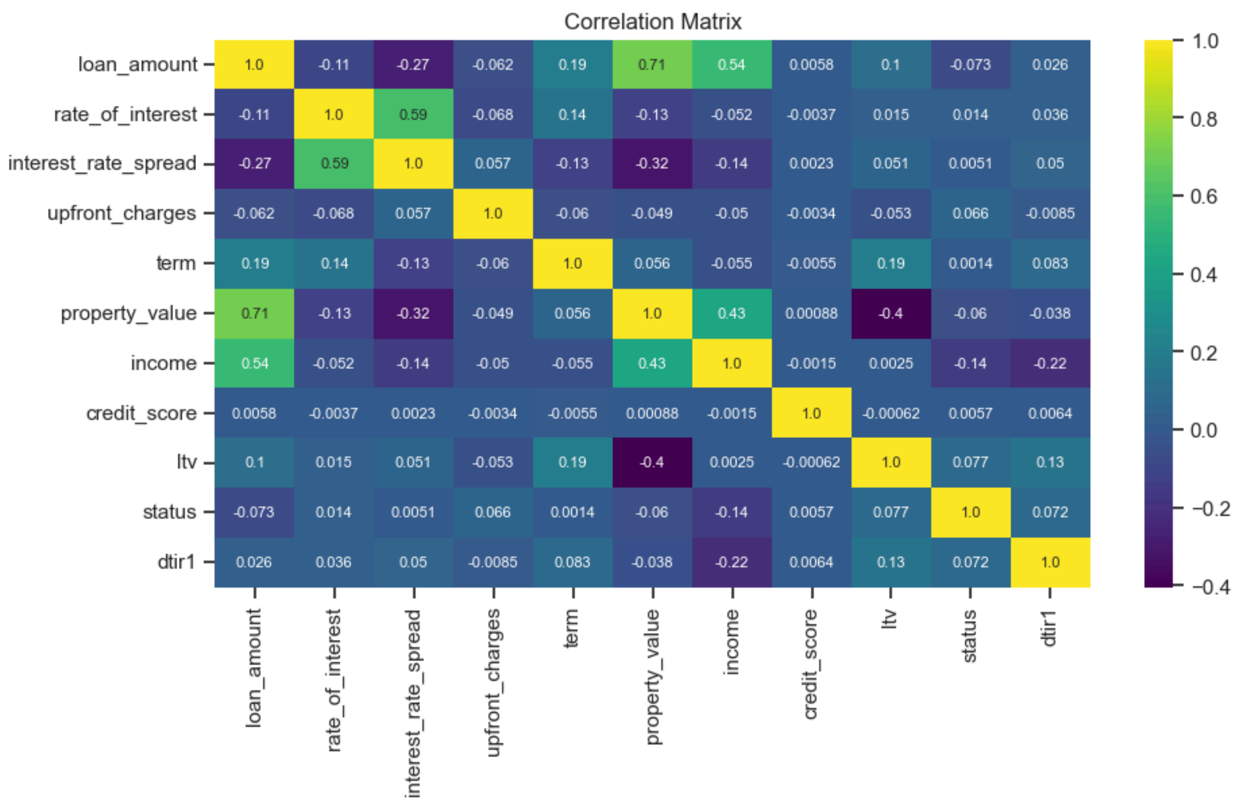
It can be inferred that 'male' and 'unknown' gender types have a higher rate of loan sanctions compared to other genders such as joint and female. Moreover, non-preapproved loans exhibit a higher sanction rate than pre-approved ones, suggesting that the pre-approval process may not be accurate in identifying potential approvals. In the case of loan types, type 2 loans have a higher approval rate. Additionally, most of the loan applications are for non-business or commercial purposes.



In the above graphs, features such as ‘interest_only’, ‘construction_type’, ‘secured_by’, and ‘security_type’ have attributes with an insignificant number of samples. These features may be considered for exclusion in later stages of modeling, depending on their relevance.

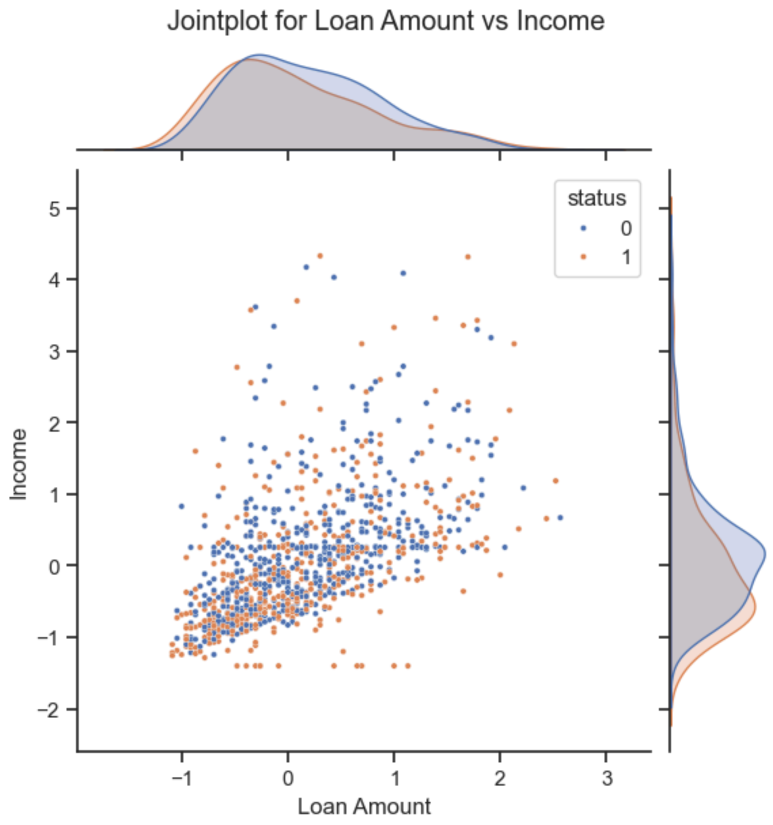
4.5. Correlation Matrix Analysis

The correlation matrix below reveals a strong correlation between *‘loan_amount’, ‘property_value’, and ‘income’,* indicating that higher incomes and property values are associated with larger loan amounts. Moreover, there is a slight positive correlation between ‘LTV (Loan to Property Value)’ and ‘term’. Suggesting that the higher the LTV the higher the term required. In contrast, the correlation with the rest of the features is not noticeably strong.



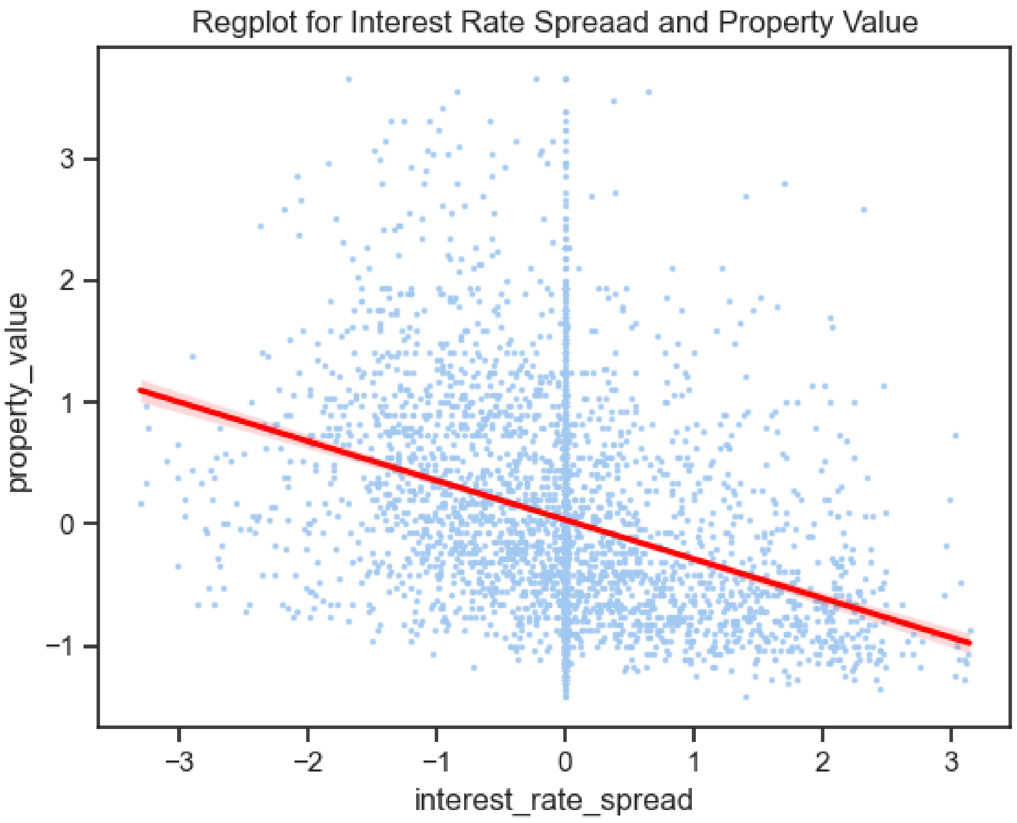
4.6. Analyzing loan amount with income

The below joint plot is used to analyze the trend between ‘income’ and ‘loan_amount’. There is no clear linear relationship between ‘income’ and ‘loan_amount’ when the entire dataset is used, likely due to overcrowding of data points. However, when a ***sample*** from the dataset is utilized, ***a noticeable linear trend emerges.***



4.7. Exploring the relationship between Interest Rate Spreaad and Property Value

We used a Regression plot(Regplot) to analyze the trend between '*interest_rate_spread'* and *'property_value'*. The regression line(Red) from the ***regplot*** indicates that both features are negatively correlated. This is also evident from the correlation matrix above.

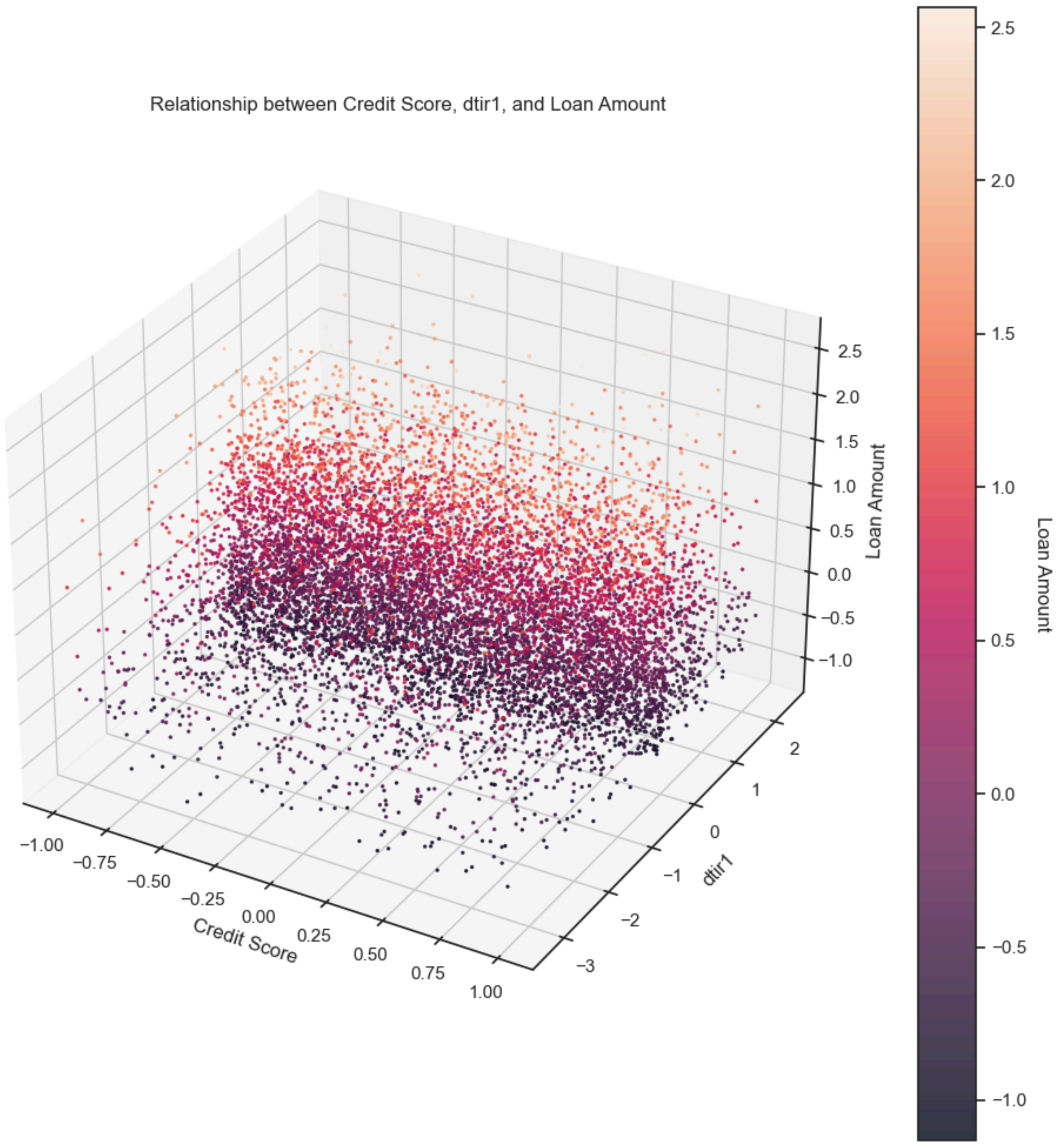


4.8. Exploring the relationship between Credit Score, Dept to Income Ratio, and Interest Rate

In the 3D - scatter plot below, we tried to analyze the relationship between, i.e., to analyze the influence of ‘credit_score’ and ‘dtir1’on ‘loan_amount’.

1. ‘credit_score’ on X-Axis
2. ‘Dept to Income Ratio(dtir1)’ on Y-Axis
3. ‘loan_amount’ on Z-Axis

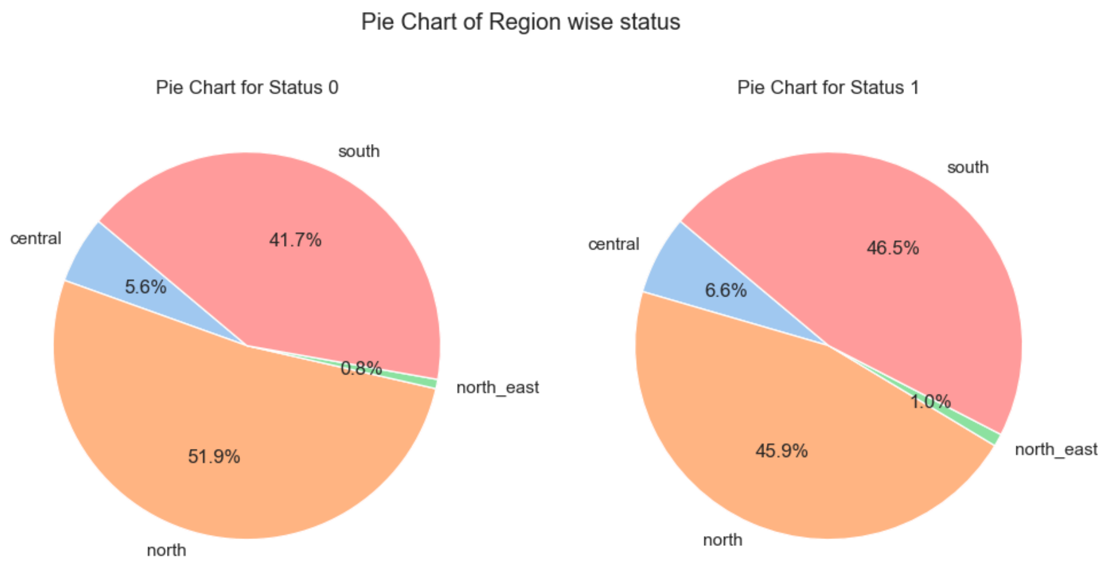
It can be observed that ‘*dtir1’* with values greater than zero has higher loan amounts while the impact of ‘credit_score’ on ‘dtir1’ is not clearly visible.



4.9. Exploring region distribution using a piechart

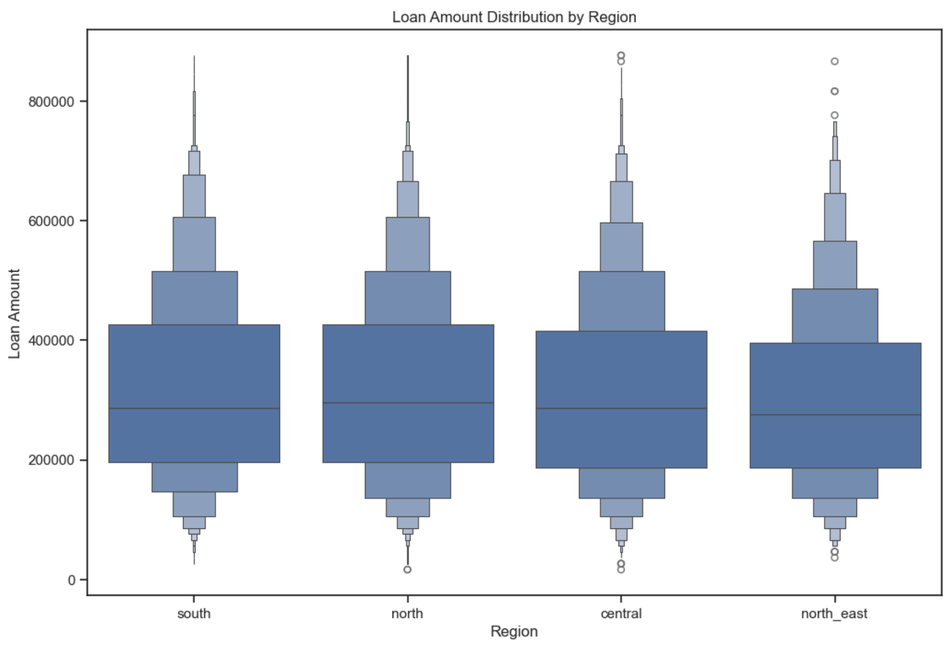
The below piechart represents the total distribution of status with respect to regions. The idea is to understand the underlying relationship in status for the given regions. It can be inferred from the below graph that the loan approvals for the South, Central, and Northeast regions are higher than rejections whereas the North region is lower than rejections.

Note: This is plotted by using the dataframe before downsampling and categorical encoding. To understand the trend better.



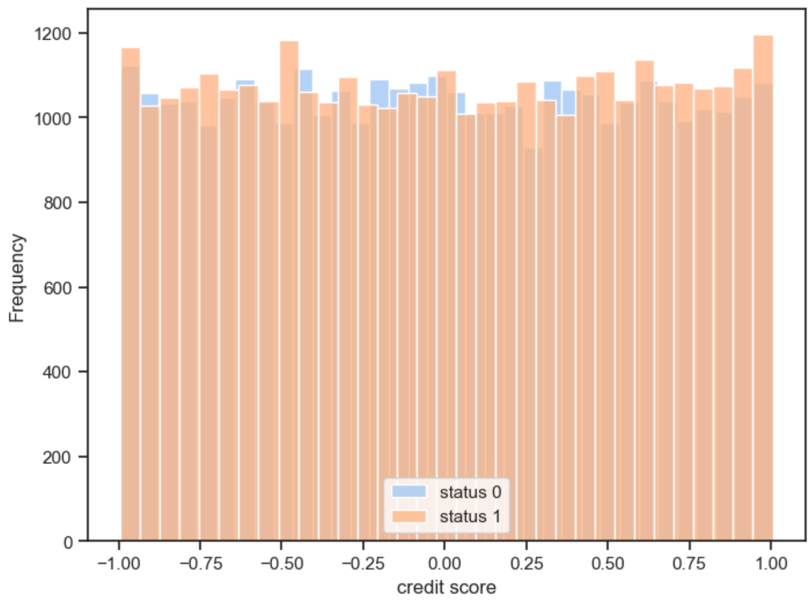
4.10. Analyzing loan amount and region

The below boxen plot provides a better distribution of data, especially in the presence of outliers. It can be inferred from the below boxen plot that the mean loan amount in the northeast region is slightly lower than in other regions. In addition, it can be inferred that the frequency of maximum loan amounts in the northeast and central regions is relatively low compared to other regions. Moreover, the higher number of boxes in all the individual plots indicates the presence of outliers.



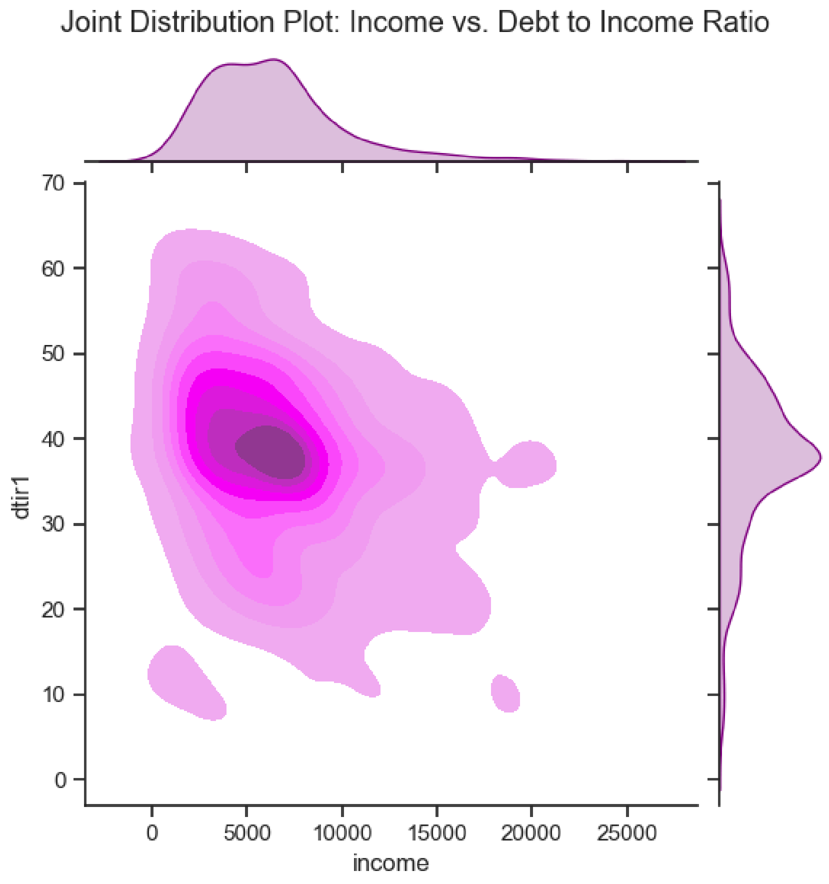
4.11. Analysis of credit score

The graph below represents the frequency of occurrence by status for the 'credit_score' feature in the downsampled dataset. According to this graph, there is no huge difference in credit scores between status 0 and 1. However, there is a slight difference, and based on this, it can be inferred that the higher the credit score, the more likely it is to be approved.



4.12. Analyzing Dept To Income Ration and Income

The Joint plot for Income Ratio and Income shows the probability density among the considered features. The darker region indicates high data concentration. Here, most of the data points lie in regions with a dtir of 40 and income of 5000.



5. References

[1] https://www.kaggle.com/datasets/yasserh/loan-default-dataset

[2] What Is an Amortization Schedule? How to Calculate with Formula (investopedia.com)

[3] https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html