Data Preparation

Every data science endeavor begins with source data that will hopefully provide insights on a question (business, technical, scientific, etc).

Each data set will present with its own characteristic data quality issues that must be identified, characterized, and (if problematic) corrected or mitigated.

The objective of data preparation is to yield a data set that can be effectively analyzed and, if desired, used as a training resource to make predictions with machine learning methods.

Background of this project

This work was originally performed to create a submission to the **MIT Applied Data Science Program** Mega Hackathon for Wilson Analytics in 2022 Winter.

Problem Statement

One of the leading financial institutions in India wants to leverage Machine Learning techniques to determine the client's loan repayment abilities and take proactive steps to reduce the magnitude of exposure to default

Goal

The goal of the problem is to predict whether a client will default on the loan payment or not, given the recent data of all the loan transactions. This can help the institution to distinguish future applicants who might default. For each ID in the Test Dataset, you must predict the "Default" level.

Data Cleaning Workflow

- **Data Collection:** Collect the data from various sources like databases, spreadsheets, web scraping, or APIs. This step may also include combining data from multiple sources.
- **Data Exploration and Preprocessing:** Explore the data to understand its structure and characteristics. Preprocess the data by handling missing values, identifying outliers, and transforming variables.
- **Data Cleaning:** Clean the data by removing duplicate entries, correcting typos, standardizing variables, and dealing with inconsistencies in the data.
- **Data Transformation:** Transform the data by normalizing, scaling, or encoding variables as required.
- **Feature Engineering:** Engineer new features that might improve model performance. This might include creating new variables based on existing variables, aggregating data, or extracting features from text or images.
- **Data Sampling:** Sample the data to ensure that the data is representative of the population it comes from.
- **Data Splitting:** Split the data into training, validation, and testing sets for model building and evaluation.
- **Data Visualization:** Visualize the data to understand patterns, correlations, and outliers in the data.
- **Iterative Refinement:** Iterate through the previous steps to refine the data as needed. This process involves going back and forth between steps until the data is ready for modeling.
- **Data Reporting:** Report on the cleaning process, including any changes made to the data, and document the final cleaned dataset.

Tech Stack

A representative tech stack for data cleaning might include the following tools:

- **Data Wrangling Libraries:** Libraries like **pandas** in **Python** or **data.table** in **R** are commonly used to manipulate and transform data, which is a key step in the data cleaning process.
- **Data Visualization Libraries:** Libraries like **matplotlib** or **ggplot2** are used for data visualization, which can help identify outliers, inconsistencies, and other data quality issues.
- **Text Processing Libraries:** Text data is often a major source of data cleaning challenges, so libraries like **NLTK** or **spaCy** can be used for cleaning, preprocessing, and feature engineering on textual data.
- **Data Quality Tools:** Data quality tools like **OpenRefine** or **Trifacta** are used for identifying and correcting errors in data, handling missing data, and dealing with inconsistencies.
- **Version Control Tools:** Version control tools like **Git** are used to track changes made to the data cleaning process and to collaborate with other team members.
- **Cloud Storage and Computing:** Cloud platforms like **Amazon Web Services**, **Microsoft Azure**, or **Google Cloud Platform** can be used for storing large datasets and for accessing computing resources needed to process data at scale.
- **Data Cleaning Frameworks:** Some data cleaning frameworks like **Dora** or **Great Expectations** can automate some parts of the data cleaning process and ensure that the data is cleaned and prepared for analysis according to the best practices and guidelines.

Tech Stack specific to this project

Python

collection of libraries and tools for tasks such as data cleaning, visualization, statistical analysis, and machine learning

numpy

Provides efficient array-based computing capabilities used to handle missing values, reshape data, and transform variables

pandas

Provides functions for handling missing data, merging and reshaping datasets, and filtering and transforming data

matplotlib

Used to create customizable and high-quality plots and visualizations to aid in identifying patterns and trends in the data

pyplot

Matplotlib module used to create interactive visualizations and facilitate data exploration

seaborn

Visualization capabilities that can help identify outliers and other data quality issues

sklearn(scikit-learn)

Supports various data preprocessing techniques, such as handling missing values and scaling features.

SciPy

Supports various data preprocessing techniques, such as handling missing values and scaling features.

parquet

A columnar storage format, compatible with a wide range of data processing tools, including Hadoop, Spark, and SQL-based databases)

Data Quality Assessment

Data Dictionary

To effectively manage data in data science, a Data Dictionary is crucial.

It captures essential parameters of data, regardless of the source (e.g., consumer surveys, sensors, web scraping, etc.). This includes the label, attributes, and description of each row and column in a table.

Our project includes a Data Dictionary, which is a table that outlines these parameters and provides guidance on encoding categorical data.

See the following slide for the data dictionary as supplied at the beginning of the project. It is contains many less relevant data and some that are confusing.

Data Dictionary

Features	Description
ID	Id of the Applicant.
Date_Of_Disbursement	The Date when the Loan is Disbursed.
Business	Type of Business. Existing or New. **ENCODE: Existing = 0, New = 1**
Jobs_Reatained	The total number of Jobs Retained by the business.
Jobs_Created	The total number of Jobs Created by the business.
Year_Of_CommitmentXS	Fiscal year of commitment.
Guaranteed_ApprovedXS_Loan	The Guaranteed Amount of Loan that has been approved by the Financial Company.
Borrower_NameXS	The Name of the borrower.
Low_Documentation_Loan	Whether the Documentation is low or not? **ENCODE: No = 0, Yes = 1**
Demography	Whether the borrower belongs to urban or rural locality? **ENCODE: urban = 0, rural = 1**
State_Of_Bank	The State of the Bank which has approved the Loan
ChargedOff_Amount	The Amount that has been charged off
Borrower_City	The City where the borrower lives.
Borrower_State	The State where the borrower lives.
Gross_Amount_Balance	The Gross amount that has been outstanding in the Loan.
Count_Employees	The total number of employees in the business.
Classification_CodeXS	North American Industry Classification Code.
Loan_Approved_Gross	Application process day
Gross_Amount_Disbursed	The total Loan Amount that has been disbursed.
Loan_Term	The total Loan term in months.
Commitment_Date	The date when the SBA commitment is issued.
Primary_Loan_Digit	The Primary Key Identifier of the Loan Account.
Code_Franchise	The Franchise Code.
Name_Of_Bank	The Name of the Bank that has approved the Loan.
Revolving_Credit_Line	Revolving Line of Credit. (Yes/No) **ENCODE: Yes = 0, No = 1**
Default (TARGET VARIABLE)	Did not default = 0, Defaulted = 1

XS = extra space to be removed

Identify data issues

Source data CSV files are ingested into dataframes (Pandas) and then displayed to provide an initial analysis of data quality issues.

train_data

ID	0	1	2	3	4
Date_Of_Disbursement	31-Jul-91	30-Apr-06	30-Jun-04	31-Jan-06	31-Dec-04
Business	Existing	New	Existing	New	Existing
Jobs_Reatained	0	0	4	9	4
Jobs_Created	0	6	0	1	0
Year_Of_Commitment	1991	2006	2004	2006	2005
Guaranteed_Approved _Loan	Rs.33121600.0	Rs.32735520.0	Rs.1422400.0	Rs.2032000.0	Rs.22981920.0
Borrower_Name	STANDARD PARTS CORPORATION	FRANK & KERI AMESTOY	TELECOMMQC L L C	K & A AUTOMOTIVE, INC. DBA MUF	SUNBEAM DELI
Low_Documentation_Loan	No	No	No	No	No
Demography	Undefined	Urban	Urban	Urban	Urban
State_Of_Bank	AP	TR	AS	BR	TR
ChargedOff_Amount	Rs.0.0	Rs.38283367.68	Rs.0.0	Rs.0.0	Rs.22862519.68
Borrower_City	Mumbai	Delhi	Bengaluru	Ahmedabad	Hyderabad
Borrower_State	Maharashtra	Delhi	Karnataka	Gujarat	Telangana
Gross_Amount_Balance	Rs.0.0	Rs.0.0	Rs.0.0	Rs.0.0	Rs.0.0
Count_Employees	38	6	4	7	4
Classification_Code	0	451120	541618	811112	722211
Loan_Approved_Gross	Rs.40640000.0	Rs.43647360.0	Rs.2844800.0	Rs.4064000.0	Rs.30642560.0
Gross_Amount_Disbursed	Rs.40640000.0	Rs.43647360.0	Rs.5961400.32	Rs.4064000.0	Rs.30642560.0
Loan_Term	126	123	90	126	104
Commitment_Date	2-Apr-91	10-Apr-06	25-May-04	21-Dec-05	2-Nov-04
Primary_Loan_Digit	4419763001	1709796003	7464754008	1588745006	8037734002
Code_Franchise	1	1	1	0	1
Name_Of_Bank	Axis Bank Ltd.	Bandhan Bank Ltd.	CSB Bank Limited	City Union Bank Ltd.	DCB Bank Ltd.
Revolving_Credit_Line	No	0	Yes	Yes	0
Default Target Variable	0	1	0	0	1

Column label "Jobs_Reatained" is misspelled

Several column labels have inappropriate extra spaces that need to be removed (blue arrows)

Date data appears in two different formats

Some of the monetary data contains both numerical and string data

Same in the testing data

test_data

	0	1	2	3	4
ID	105000	105001	105002	105003	105004
Date_Of_Disbursement	31-Mar-06	31-Jan-95	30-Sep-06	31-Jul-00	30-Jun-05
Business	Existing	Existing	Existing	New	Existing
Jobs_Reatained	19	0	7	2	0
Jobs_Created	0	0	5	0	0
Year_Of_Commitment	2006	1995	2006	2000	2005
Guaranteed_Approved _Loan	Rs.4064000.0	Rs.1463040.0	Rs.812800.0	Rs.2032000.0	Rs.23469600.0
Borrower_Name	Diversified Display Products o	FOOTE CONSULTING GROUP, INC.	INTEGRATED COMERCIAL ENTERPRIS	FIRST IN RESCUE EQUIPMENT	GLASGOW AUTOMOTIVE, INC.
Low_Documentation_Loan	No	Yes	No	No	No
Demography	Urban	Undefined	Urban	Urban	Rural
State_Of_Bank	GJ	AS	ML	TR	TR
ChargedOff_Amount	Rs.8050784.0	Rs.0.0	Rs.1625600.0	Rs.0.0	Rs.0.0
Borrower_City	Safidon	Nanjikottai	Tonk	Musabani	Adityapur
Borrower_State	Haryana	Tamil Nadu	Rajasthan	Jharkhand	Jharkhand
Gross_Amount_Balance	Rs.0.0	Rs.0.0	Rs.0.0	Rs.0.0	Rs.0.0
Count_Employees	17	2	2	2	6
Classification_Code	326199	0	541611	0	441310
Loan_Approved_Gross	Rs.8128000.0	Rs.1625600.0	Rs.1625600.0	Rs.4064000.0	Rs.31292800.0
Gross_Amount_Disbursed	Rs.9403852.16	Rs.1625600.0	Rs.3450336.0	Rs.6916196.48	Rs.31292800.0
Loan_Term	57	90	81	18	219
Commitment_Date	9-Mar-06	14-Dec-94	25-Aug-06	28-Jun-00	2-May-05
Primary_Loan_Digit	1702825000	7908833003	2361626001	3814664008	8830244003
Code_Franchise	0	1	1	1	1
Name_Of_Bank	ICICI Bank Ltd.	South Indian Bank Ltd.	IDBI Bank Limited	Aryavart Bank	Paschim Banga Gramin Bank
Revolving_Credit_Line	Yes	No	Yes	Yes	No

Check for dataframe shape

```
# Determine the shape of the TRAINING dataset
train_data.shape
```

(105000, 26)

```
# Determine the shape of the TESTING dataset
test_data.shape
```

(45000, 25)

Training Data

- Rows = **105,000**
- Columns = **26**

Testing Data

- Rows = **45,000**
- Columns = **25**
  - As expected, the testing data set lacks the target variable column

Observations on Initial Data

These columns will be dropped:

- 'Jobs_Retained'
- 'Jobs_Created '
- 'Count_Employees'
- 'ID'
- 'Date_Of_Disbursement'
- 'Commitment_Date'
- 'Code_Franchise'
- 'Year_Of_Commitment'
- 'Classification_Code'
- 'Borrower_Name'
- 'Borrower_City'
- 'Gross_Amount_Balance'
- 'Revolving_Credit_Line'
- 'State_Of_Bank'
- 'Borrower_State'
- 'Name_Of_Bank'
- 'Primary_Loan_Digit'
- ‘Loan_Approved_Gross’

Target Variable = 'Default' (Did not default = 0, Defaulted = 1)

Categorical variables that need to be encoded:

- 'Business' (Existing or New)
- 'Low_Documentation_Loan' (Low or Not)
- 'Demography' (Undecided, Urban or Rural)

Data Pre-Processing

Renaming misspelled and poorly formatted column names

```
columns = {'ChargedOff_Amount ': 'ChargedOff_Amount', 'Gross_Amount_Disbursed  ': 'Gross_Amount_Disbursed', 'Guaranteed_Approved _Loan': 'Guaranteed_Approved_Loan', 'Jobs_Reatained': 'Jobs_Retained', 'Borrower_Name ': 'Borrower_Name', 'Classification_Code ': 'Classification_Code', 'Year_Of_Commitment ': 'Year_Of_Commitment'}

train_data = train_data.rename(columns, axis = 1)
test_data = test_data.rename(columns, axis = 1)
```

Note:

- Many of these renamed columns will be dropped in the next step
- I have included this here as a demonstration of one important aspect of the data cleaning workflow

Dropping columns

```
cols_to_drop = ['Jobs_Retained', 'Jobs_Created ', 'Count_Employees', 'ID', 'Date_Of_Disbursement',
'Commitment_Date', 'Code_Franchise', 'Year_Of_Commitment', 'Classification_Code',
'Borrower_Name', 'Borrower_City', 'Gross_Amount_Balance', 'Revolving_Credit_Line',
'State_Of_Bank', 'Borrower_State', 'Name_Of_Bank', 'Primary_Loan_Digit', 'Loan_Approved_Gross']

train_data.drop(columns=cols_to_drop, inplace=True)
test_data.drop(columns=cols_to_drop, inplace=True)
```

After dropping these columns

train_data.shape = 8 columns and 105,000 rows
test_data.shape = 7 columns and 45,000 rows

New Data Dictionary, after pre-processing

Features	Description
Business	Type of business. **ENCODE: Existing = 0, New = 1**
Guaranteed_Approved_Loan	The guaranteed amount of loan that has been approved by the financial company.
Low_Documentation_Loan	Whether the loan documentation is low or not. **ENCODE: No = 0, Yes = 1**
Demography	Whether the borrower lives in an urban or rural locality? **ENCODE: Undefined = 0, Urban = 1, Rural = 2**
ChargedOff_Amount	The amount that has been charged off (loss to financial company due to default)
Gross_Amount_Disbursed	The total loan amount that has been disbursed.
Loan_Term	The total loan term in months.
Default (TARGET VARIABLE)	Did not default = 0, Defaulted = 1

Checking for data types

train_data

```
train_data.dtypes

Business                    object
Guaranteed_Approved_Loan    object
Low_Documentation_Loan      object
Demography                  object
ChargedOff_Amount           object
Gross_Amount_Disbursed      object
Loan_Term                    int64
Default                      int64
dtype: object
```

test_data

```
test_data.dtypes

Business                    object
Guaranteed_Approved_Loan    object
Low_Documentation_Loan      object
Demography                  object
ChargedOff_Amount           object
Gross_Amount_Disbursed      object
Loan_Term                    int64
dtype: object
```

Fields in the red boxes SHOULD be numeric but are being detected as ‘object’ (string)

Correcting Monetary Data Issues

All of the following columns have entries that relate to monetary amounts:

- 'Guaranteed_Approved_Loan'
- 'Gross_Amount_Balance'
- 'Gross_Amount_Disbursed'
- 'ChargedOff_Amount'

These monetary columns are detected as an ‘**object**’ because, in addition to the numerical data, a prefix indicating that these numbers are in Rupees ('Rs.') is present

Various monetary columns, listing rupee amounts, are detected as ‘object’ due to the string ‘Rs.” being included.

train_data

```
train_data.head().T
```

	0	1	2	3	4
Business	Existing	New	Existing	New	Existing
Guaranteed_Approved_Loan	Rs.33121600.0	Rs.32735520.0	Rs.1422400.0	Rs.2032000.0	Rs.22981920.0
Low_Documentation_Loan	No	No	No	No	No
Demography	Undefined	Urban	Urban	Urban	Urban
ChargedOff_Amount	Rs.0.0	Rs.38283367.68	Rs.0.0	Rs.0.0	Rs.22862519.68
Gross_Amount_Disbursed	Rs.40640000.0	Rs.43647360.0	Rs.5961400.32	Rs.4064000.0	Rs.30642560.0
Loan_Term	126	123	90	126	104
Default	0	1	0	0	1

test_data

```
test_data.head().T
```

	0	1	2	3	4
Business	Existing	Existing	Existing	New	Existing
Guaranteed_Approved_Loan	Rs.4064000.0	Rs.1463040.0	Rs.812800.0	Rs.2032000.0	Rs.23469600.0
Low_Documentation_Loan	No	Yes	No	No	No
Demography	Urban	Undefined	Urban	Urban	Rural
ChargedOff_Amount	Rs.8050784.0	Rs.0.0	Rs.1625600.0	Rs.0.0	Rs.0.0
Gross_Amount_Disbursed	Rs.9403852.16	Rs.1625600.0	Rs.3450336.0	Rs.6916196.48	Rs.31292800.0
Loan_Term	57	90	81	18	219

Code to address the data heterogeneity

```
def replace_and_cast_to_int(data, columns, replace_dict):
   for column in columns:
       data[column] = data[column].replace(replace_dict, regex=True).astype(float)
       data[column] = data[column].apply(lambda x: int(round(x)))
```

```
columns = ['Guaranteed_Approved_Loan', 'Gross_Amount_Disbursed', 'ChargedOff_Amount']
replace_dict = {'Rs.': '', ',': ''}

replace_and_cast_to_int(train_data, columns, replace_dict)
replace_and_cast_to_int(test_data, columns, replace_dict)
```

Corrected monetary columns

train_data

```
train_data.head().T
```

	0	1	2	3	4
Business	Existing	New	Existing	New	Existing
Guaranteed_Approved_Loan	33121600.0	32735520.0	1422400.0	2032000.0	22981920.0
Low_Documentation_Loan	No	No	No	No	No
Demography	Undefined	Urban	Urban	Urban	Urban
ChargedOff_Amount	0.0	38283367.68	0.0	0.0	22862519.68
Gross_Amount_Disbursed	40640000.0	43647360.0	5961400.32	4064000.0	30642560.0
Loan_Term	126	123	90	126	104
Default	0	1	0	0	1

test_data

```
test_data.head().T
```

	0	1	2	3	4
Business	Existing	Existing	Existing	New	Existing
Guaranteed_Approved_Loan	4064000.0	1463040.0	812800.0	2032000.0	23469600.0
Low_Documentation_Loan	No	Yes	No	No	No
Demography	Urban	Undefined	Urban	Urban	Rural
ChargedOff_Amount	8050784.0	0.0	1625600.0	0.0	0.0
Gross_Amount_Disbursed	9403852.16	1625600.0	3450336.0	6916196.48	31292800.0
Loan_Term	57	90	81	18	219

Check data type

Check data type by using this code:

train_data.dtypes
test_data.dtypes

Once the ‘Rs.’ string was deleted from the monetary columns the data type has changed to the correct type (from ‘object’ to ‘int64’)

train_data

```
train_data.dtypes

Business                     object
Guaranteed_Approved_Loan     object
Low_Documentation_Loan       object
Demography                   object
ChargedOff_Amount            object
Gross_Amount_Disbursed       object
Loan_Term                     int64
Default                       int64
dtype: object
```

test_data

```
test_data.dtypes

Business                     object
Guaranteed_Approved_Loan     object
Low_Documentation_Loan       object
Demography                   object
ChargedOff_Amount            object
Gross_Amount_Disbursed       object
Loan_Term                     int64
dtype: object
```

train_data

```
train_data.dtypes

Business                     object
Guaranteed_Approved_Loan      int64
Low_Documentation_Loan       object
Demography                   object
ChargedOff_Amount             int64
Gross_Amount_Disbursed        int64
Loan_Term                     int64
Default                       int64
dtype: object
```

test_data

```
test_data.dtypes

Business                     object
Guaranteed_Approved_Loan      int64
Low_Documentation_Loan       object
Demography                   object
ChargedOff_Amount             int64
Gross_Amount_Disbursed        int64
Loan_Term                     int64
dtype: object
```

Check for missing data

Missing Data

train_data

```
train_data.isnull().sum()*100/len(train_data)
```

```
Business                    0.014286
Guaranteed_Approved_Loan    0.000000
Low_Documentation_Loan      0.349524
Demography                  0.000000
ChargedOff_Amount           0.000000
Gross_Amount_Disbursed      0.000000
Loan_Term                   0.000000
Default                     0.000000
dtype: float64
```

test_data

```
test_data.isnull().sum()*100/len(test_data)
```

```
Business                    0.013333
Guaranteed_Approved_Loan    0.000000
Low_Documentation_Loan      0.295556
Demography                  0.000000
ChargedOff_Amount           0.000000
Gross_Amount_Disbursed      0.000000
Loan_Term                   0.000000
dtype: float64
```

Observations

Missing Values in **TRAINING data:**

- 'Business' = 0.014 %
- 'Low_Documentation_Loan' = 0.35 %

Missing Values in **TESTING data:**

- 'Business' = 0.013 %
- 'Low_Documentation_Loan' = 0.30 %

Examine unique values

Unique Values

train_data

```
train_data.nunique()
```

```
Business                         3
Guaranteed_Approved_Loan     10138
Low_Documentation_Loan           7
Demography                       3
ChargedOff_Amount            23059
Gross_Amount_Disbursed       23443
Loan_Term                      344
Default                          2
dtype: int64
```

test_data

```
test_data.nunique()
```

```
Business                         3
Guaranteed_Approved_Loan      6151
Low_Documentation_Loan           7
Demography                       3
ChargedOff_Amount            10833
Gross_Amount_Disbursed       11723
Loan_Term                      329
dtype: int64
```

Observations

Analysis of **unique values** reveals that in the ‘Demography’ column there are substantial numbers of entries that are “Undefined”:

- ‘train_data’
  - Of the 105,000 rows, 35,099 are undefined.
  - That is **33%** or a 1/3 of the data.

- ‘test_data’
  - Of the 45,000 rows, 15,020 are undefined.
  - That is **33%** or a 1/3 of the data.

Dropping all rows that have "Undecided" would cause a 33% reduction in the entire dataset and is not desirable

For this reason, **the ‘Demography’ column is dropped**

Encode Categorical Data

Encoding of Categorical Values

As shown at the beginning of this project, certain columns must be translated from a text string (eg, 'Yes', 'No', etc) into a numeric quantity to make it accessible to further statistical analysis and inclusion in prediction models.

A method called **Nominal Label Encoding** will be used to prepare the categorical data for machine learning methods

In Nominal Label Encoding, a specific value is assigned to a specific string found in a specific column of data. This method **uses dictionaries to map the assigned numeric value in the place of the specified string value**.

This method can be considered a simple solution as it **does NOT create new columns** as is the case of One Hot Encoding.

As has been the case throughout this project, all data preparation done on the training data set is also done on the testing data set.

Encoding of Categorical Values

'Business'

- '**Existing**' = 0
- '**New**' = 1

'Low_Documentation_Loan'

- '**No**' = 0
- '**Yes**' = 1
- All **173** of the '**O**' entries remain '**0**' (UNCHANGED DURING ENCODING)
- All **95** of the '**S**' entries are assigned '**0**'
- All **60** of the '**A**' entries are assigned '**0**'
- All **89** of the '**C**' entries are assigned '**1**'
- All **6** of the '**R**' entries are assigned '**1**'

Code

```
def encode_column(data, column, encoding_dict):
  return data[column].map(encoding_dict)
```

```
# Encoding dictionaries
business_encoding_dict = {'Existing': 0, 'New': 1}
low_encoding_dict = {'No': 0, 'Yes': 1, 'S': 0, 'A': 0, 'C': 1, 'R': 1}

# Encode columns in train_data
train_data['Business'] = encode_column(train_data, 'Business', business_encoding_dict)
train_data['Low_Documentation_Loan'] = encode_column(train_data, 'Low_Documentation_Loan', low_encoding_dict)

# Encode columns in test_data
test_data['Business'] = encode_column(test_data, 'Business', business_encoding_dict)
test_data['Low_Documentation_Loan'] = encode_column(test_data, 'Low_Documentation_Loan', low_encoding_dict)
```

After encoding

train_data

	Business	Guaranteed_Approved_Loan	Low_Documentation_Loan	ChargedOff_Amount	Gross_Amount_Disbursed	Loan_Term	Default
0	0.0	33121600	0.0	0	40640000	126	0
1	1.0	32735520	0.0	38283368	43647360	123	1
2	0.0	1422400	0.0	0	5961400	90	0
3	1.0	2032000	0.0	0	4064000	126	0
4	0.0	22981920	0.0	22862520	30642560	104	1

test_data

	Business	Guaranteed_Approved_Loan	Low_Documentation_Loan	ChargedOff_Amount	Gross_Amount_Disbursed	Loan_Term
0	0.0	4064000	0.0	8050784	9403852	57
1	0.0	1463040	1.0	0	1625600	90
2	0.0	812800	0.0	1625600	3450336	81
3	1.0	2032000	0.0	0	6916196	18
4	0.0	23469600	0.0	0	31292800	219

After encoding

After encoding, nulls still exist

```
train_data.isnull().sum()
```

```
Business                     120
Guaranteed_Approved_Loan       0
Low_Documentation_Loan       540
ChargedOff_Amount              0
Gross_Amount_Disbursed         0
Loan_Term                      0
Default                        0
dtype: int64
```

```
test_data.isnull().sum()
```

```
Business                      60
Guaranteed_Approved_Loan       0
Low_Documentation_Loan       197
ChargedOff_Amount              0
Loan_Approved_Gross            0
Gross_Amount_Disbursed         0
Loan_Term                      0
dtype: int64
```

Removing the nulls

```
def fill_na(df, columns):
    for col in columns:
        df[col] = df[col].fillna(0)
    return df

columns_to_fill = ['Business', 'Low_Documentation_Loan']
train_data = fill_na(train_data, columns_to_fill)

columns_to_fill = ['Business', 'Low_Documentation_Loan']
test_data = fill_na(test_data, columns_to_fill)
```

```
train_data.isnull().sum()

Business                    0
Guaranteed_Approved_Loan    0
Low_Documentation_Loan      0
ChargedOff_Amount           0
Gross_Amount_Disbursed      0
Loan_Term                   0
Default                     0
dtype: int64
```

```
test_data.isnull().sum()

Business                    0
Guaranteed_Approved_Loan    0
Low_Documentation_Loan      0
ChargedOff_Amount           0
Loan_Approved_Gross         0
Gross_Amount_Disbursed      0
Loan_Term                   0
dtype: int64
```

Revised Data Dictionary

Features	Description
Business	Type of business. **ENCODE: Existing = 0, New = 1**
Guaranteed_Approved_Loan	The guaranteed amount of loan that has been approved by the financial company.
Low_Documentation_Loan	Whether the loan documentation is low or not. **ENCODE: No = 0, Yes = 1**
ChargedOff_Amount	The amount that has been charged off (loss to financial company due to default)
Gross_Amount_Disbursed	The total loan amount that has been disbursed.
Loan_Term	The total loan term in months.
Default (TARGET VARIABLE)	Did not default = 0, Defaulted = 1

Final DataFrame shapes

train_data
Rows = 105,000
Columns = 7

test_data
Rows = 45,000
Columns = 6

Outliers

Addressing Outliers

Rationale for normalization to address outliers

- Some machine learning methods assume normal distributions in the input data
- Data with significant variance/outliers may compromise ML performance
- Outliers and variance are often important aspects of data so they should not be simply dropped

- I will be comparing the impact of the use of no normalization, winsorization, and log transform on data distribution and then, later, on ML performance in another project.

Addressing Outliers - Winsorization

Rationale for using Winsorization to address outliers

Winsorization works by identifying extreme values, which are typically defined as values that are a certain number of standard deviations away from the mean of the dataset. Once the extreme values have been identified, they are replaced with values that are less extreme but still within a certain range of the original values.

Two types:

- **Minimum Winsorization**
  - extreme values that are below a certain threshold are replaced with the value of the threshold
- **Maximum Winsorization**
  - the extreme values that are above a certain threshold are replaced with the value of the threshold.

Addressing Outliers - Winsorization

With respect to the code used to winsorize these data sets:

- The set limits=[0.2, 0.2] applies the same 20% trimming limit to both tails of the distribution, which means that the function will replace the 10% lowest values and 10% highest values with the adjacent values.

- There is no distinction between minimum and maximum limits. The limits are symmetrical, so the same fraction of values will be trimmed from both ends of the distribution.

Winsorization Code

```
# Define the columns to Winsorize

columns_to_winsorize = ['Guaranteed_Approved_Loan', 'ChargedOff_Amount', 'Gross_Amount_Disbursed']


# Create a new DataFrame for the Winsorized data

train_data_win = pd.DataFrame()


# Apply Winsorization to the selected columns and store the results in the new DataFrame

for column in columns_to_winsorize:
   train_data_win[column + '_win'] = winsorize(train_data[column], limits=[0.2, 0.2])


# Add the remaining columns from the original DataFrame to the new DataFrame

train_data_win = pd.concat([train_data_win, train_data.drop(columns_to_winsorize, axis=1)], axis=1)
```

Addressing Outliers - Log Transform

Rationale for using Log Transformation to address outliers

- Some machine learning methods assume normal distributions in the input data
- Data with significant variance/outliers may compromise ML performance
- Outliers and variance are often important aspects of data so they should not be simply dropped
- Log Transformation brings data into a distribution more effectively approximating a standard curve
- It preserves relative changes and magnitude of change

Log transform code

```
# Define the columns to log transform
columns_to_log = ['Guaranteed_Approved_Loan', 'ChargedOff_Amount', 'Gross_Amount_Disbursed']

# Create a new DataFrame for the Winsorized data
train_data_log = pd.DataFrame()

# Apply Winsorization to the selected columns and store the results in the new DataFrame

for column in columns_to_log:
  train_data_log[column + '_log'] = np.log(train_data[column].where(train_data[column] > 0, 1))

# Add the remaining columns from the original DataFrame to the new DataFrame

train_data_log = pd.concat([train_data_log, train_data.drop(columns_to_log, axis=1)], axis=1)
```

Visualization code - Box and Histplots

```
fig, ax = plt.subplots(1, 2, figsize=(20, 10))
sns.boxplot(x='Default', y='Guaranteed_Approved_Loan_win', palette='flare', data=train_data_win, ax=ax[0])
sns.histplot(train_data_win, x='Guaranteed_Approved_Loan_win', hue='Default', multiple='stack',
palette='flare', edgecolor='.3', linewidth=.5, ax=ax[1])
ax[1].set(xlabel='Guaranteed_Approved_Loan_win')
ax[0].set(title='Winsorized', ylabel='Guaranteed_Approved_Loan_win')
ax[1].set(title='Winsorized', ylabel='Count')
fig.suptitle('Loan Approval and Default: Analysis of Winsorized Loan Amount', fontsize=16)
plt.show()

fig, ax = plt.subplots(1, 2, figsize=(20, 10))
sns.boxplot(x='Default', y='Guaranteed_Approved_Loan_log', palette='flare', data=train_data_log, ax=ax[0])
sns.histplot(train_data_log, x='Guaranteed_Approved_Loan_log', hue='Default', multiple='stack',
palette='flare', edgecolor='.3', linewidth=.5, ax=ax[1])
ax[1].set(xlabel='Guaranteed_Approved_Loan_log')
ax[0].set(title='Log Transformed', ylabel='Guaranteed_Approved_Loan_log')
ax[1].set(title='Log Transformed', ylabel='Count')
fig.suptitle('Loan Approval and Default: Analysis of Log Transformed Loan Amount', fontsize=16)
plt.show()
```

Observations

Winsorized data:

- The distribution has shifted from right skew to non-normal distribution that reflects the transforms done on the bottom and top 20% of the data.
- The box plots have shifted their main body to a more central location and encompass a wide range
- There are fewer 'outliers' but they will remain as relevant data

Log transformed data:

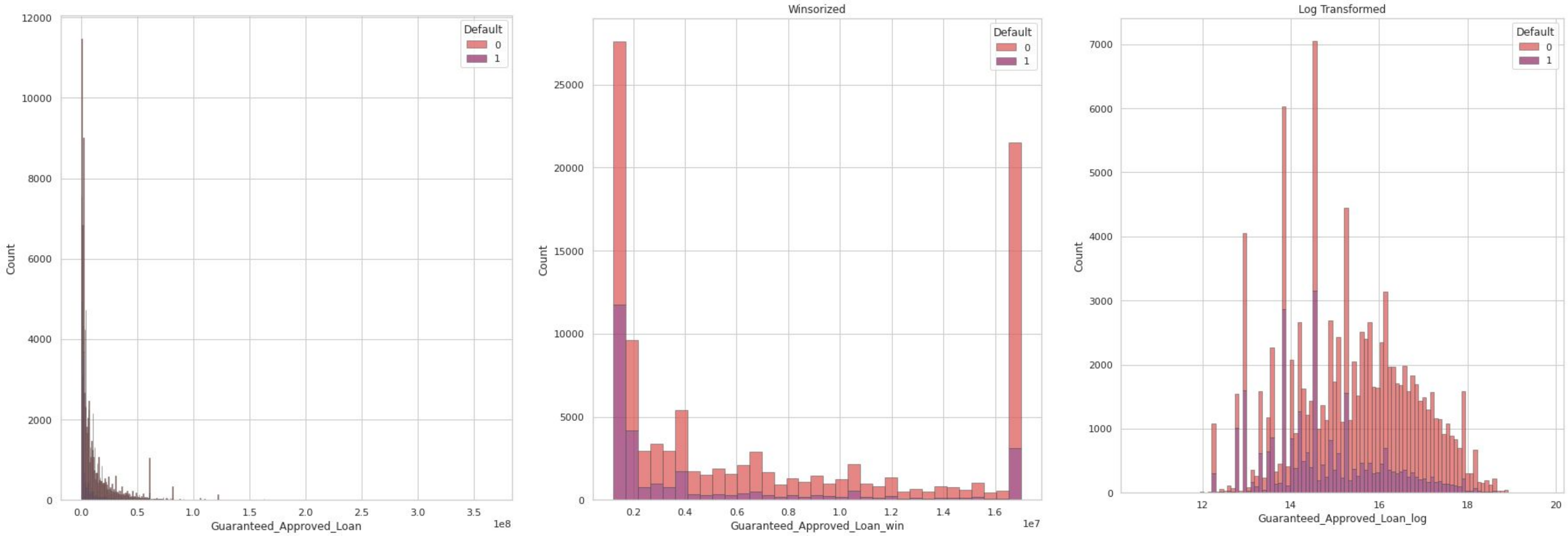
- The distribution has shifted from right skew to a more normal distribution.
- The box plots have shifted their main body to a more central location
- There are fewer 'outliers' but they will remain as relevant data

It remains to be seen how winsorization and log transforms impact machine learning performance. This will be assess in another project.

‘Guaranteed_Approved_Loan’

Impact of normalization (log transformation) on ‘Guaranteed_Approved_Loan’ variable

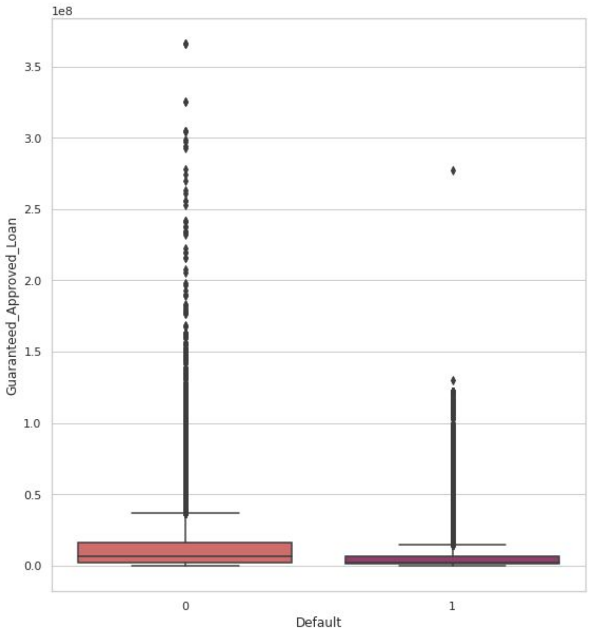
Distribution



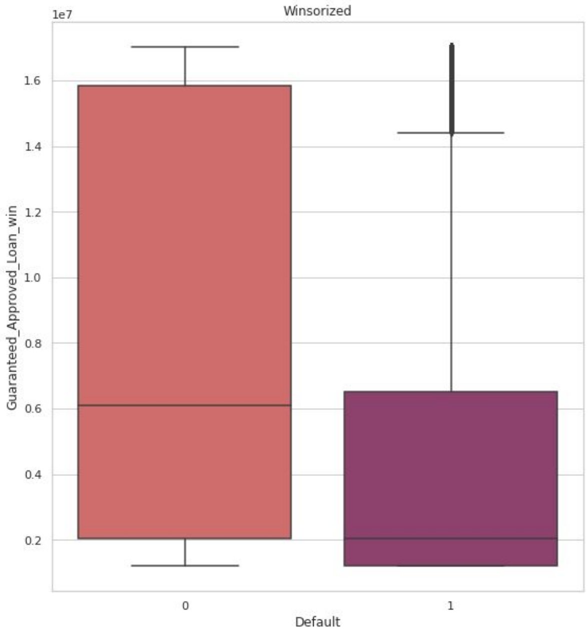
none | winsorized | logged

Impact of normalization (log transformation) on ‘Guaranteed_Approved_Loan’ variable

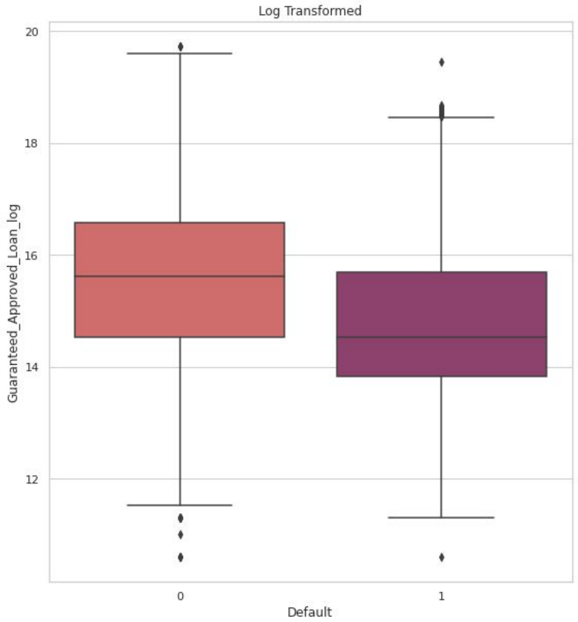
Box Plot



none



winsorized

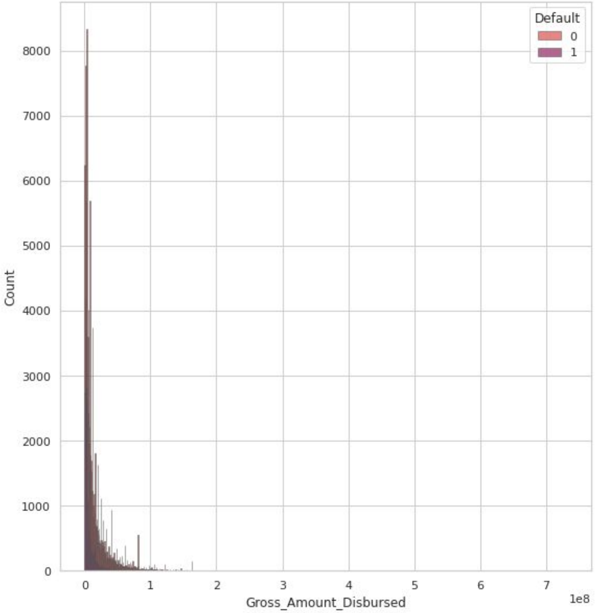


logged

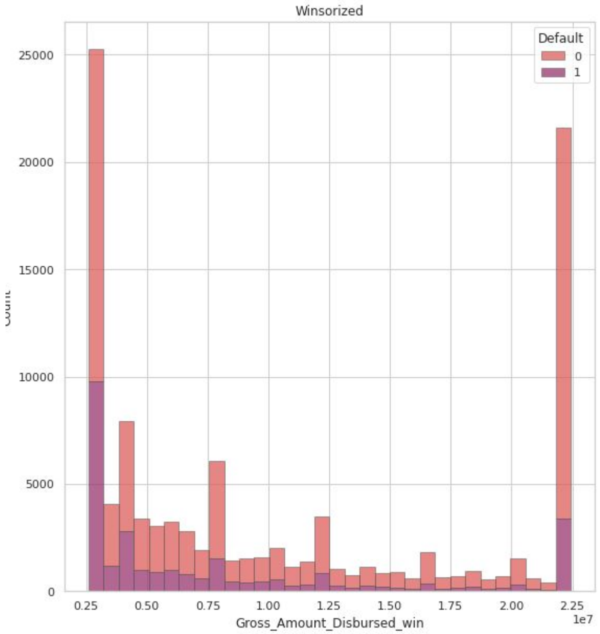
‘Gross_Amount_Disbursed’

Impact of normalization (log transformation) on 'Gross_Amount_Disbursed' variable

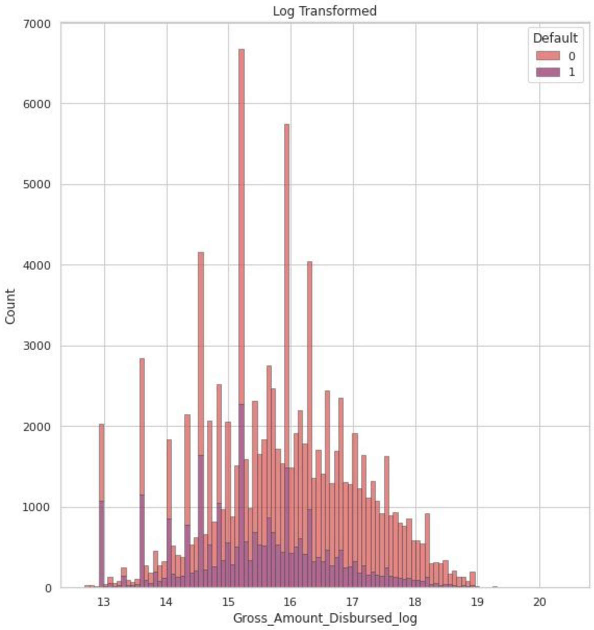
Distribution



none



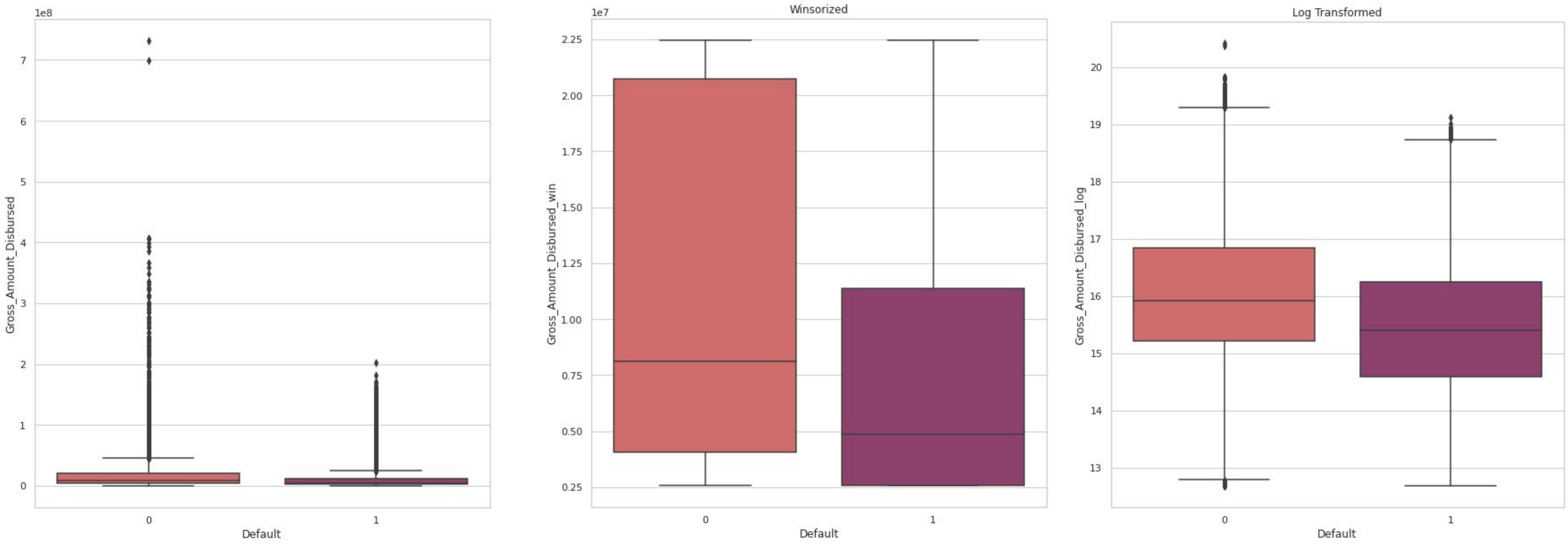
winsorized



logged

Impact of normalization (log transformation) on ‘Gross_Amount_Disbursed’ variable

Box Plot



none

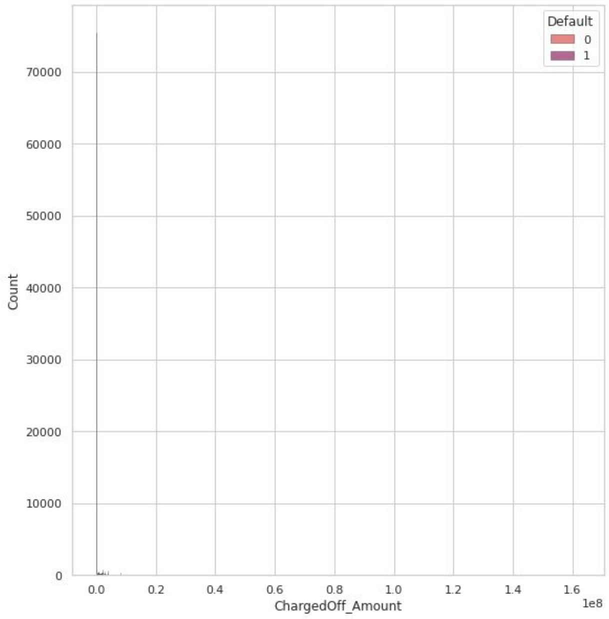
winsorized

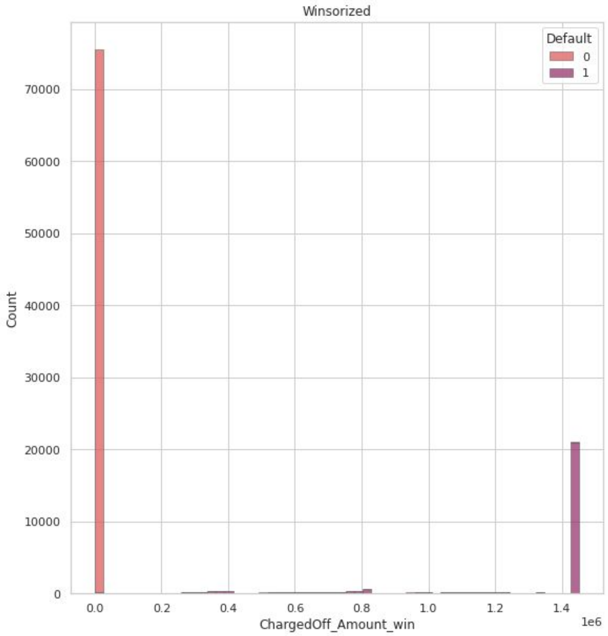
logged

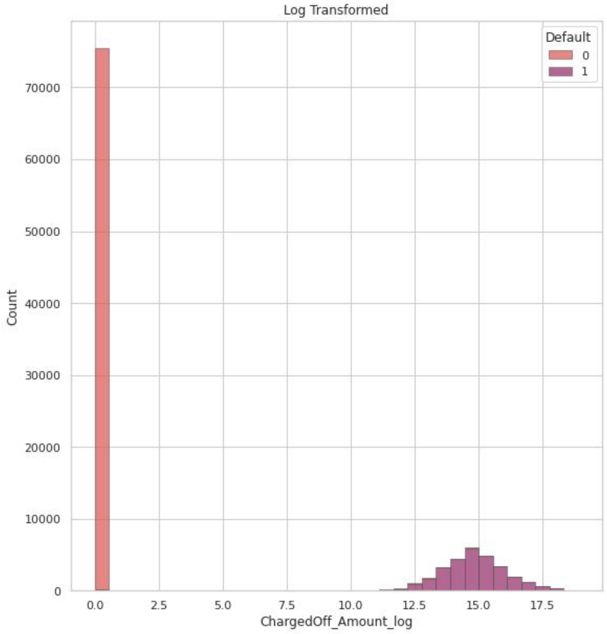
‘ChargedOff_Amount’

Impact of normalization (log transformation) on 'ChargedOff_Amount' variable

Distribution







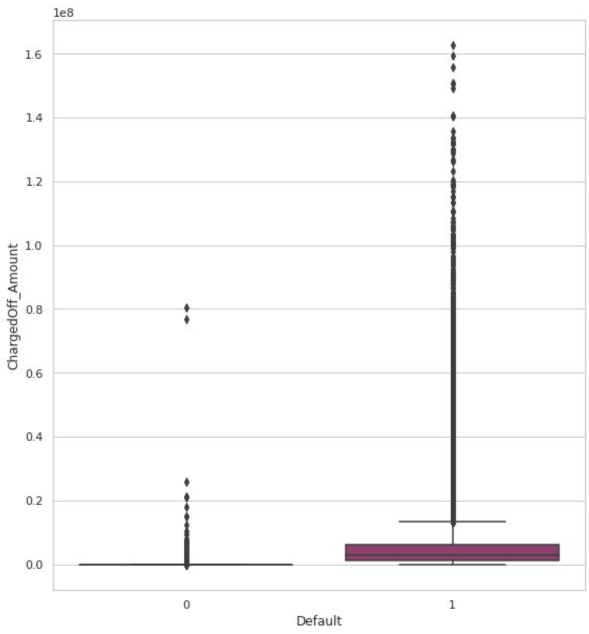
none

winsorized

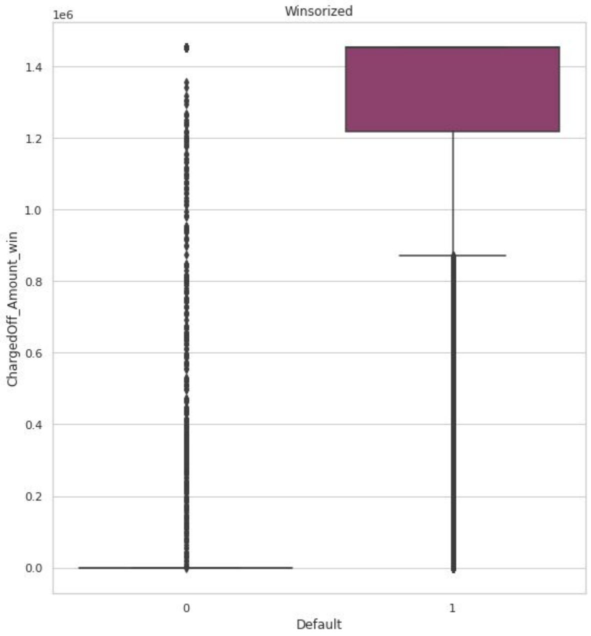
logged

Impact of normalization (log transformation) on ‘ChargedOff_Amount’ variable

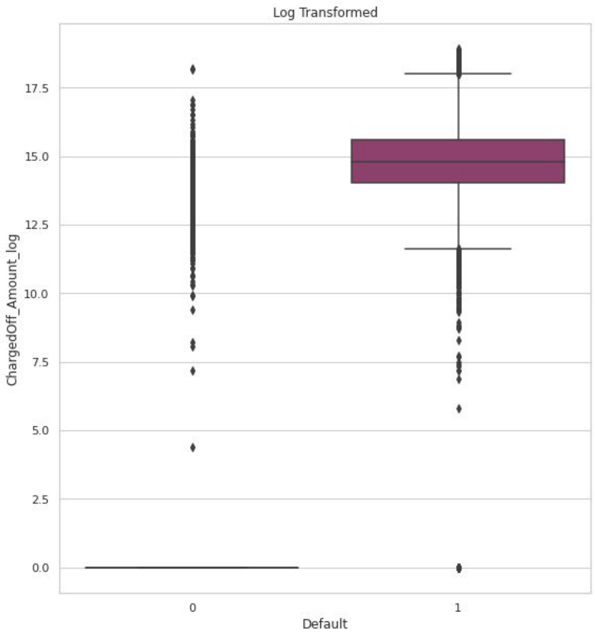
Box Plot



none



winsorized



logged