# Credit Score Prediction: Cleaning and Transforming Financial Data to Improve Credit Risk Assessment Models

**Name : Raamanjal Singh Gangwar**

**Roll No: 202401100300187**

**Course: Artificial Intelligence**

**Date: 11-03-2025**

# 1. Introduction

Credit risk assessment is an essential process for financial institutions to evaluate a borrower's ability to repay loans. Accurate credit risk models require high-quality financial data. This project focuses on cleaning and transforming financial data to enhance credit score prediction models, ultimately improving risk assessment and decision-making.

## Objective:

- Improve data quality by handling missing values, outliers, and inconsistencies.
- Apply data transformation techniques to optimize model performance.
- Build a predictive model for credit risk assessment.

# 2. Methodology

To develop a reliable credit score prediction model, the following steps were undertaken:

## 2.1 Data Collection

Financial data was sourced from credit reports, customer transactions, and financial statements.

## 2.2 Data Cleaning

• **Handling Missing Values:** Used mean/mode imputation and predictive filling.

• **Removing Duplicates:** Ensured unique records by eliminating redundant data.

• **Handling Outliers:** Used statistical techniques like Z-score and IQR to detect and remove anomalies.

## 2.3 Data Transformation

• **Normalization & Scaling:** Applied Min-Max Scaling to bring numerical features to a standard range.

• **Encoding Categorical Variables:** Used One-Hot Encoding and Label Encoding.

• **Feature Engineering:** Created new relevant features like Debt-to-Income Ratio and Credit Utilization.

### 2.4 Model Selection & Training

• Compared different machine learning models (Logistic Regression, Decision Trees, Random Forest, XGBoost).

• Evaluated models using accuracy, precision, recall, and F1-score.

## 3. Code Implemented

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

%matplotlib inline
# %matplotlib notebook
plt.rcParams["figure.figsize"] = (12, 6)
# plt.rcParams['figure.dpi'] = 100
sns.set_style("whitegrid")
import warnings

warnings.filterwarnings("ignore")
warnings.warn("this will not show")
pd.set_option('display.float_format', lambda x: '%.3f' % x)
# Pre-Processing
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, MinMaxScaler
```

```python
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score, cross_validate
# Metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve,
average_precision_score

# Model relavant libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout,
BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.saving import save_model
from keras.optimizers import Adam
from keras.regularizers import l2
train0 = pd.read_csv('/content/train.csv', sep=',', on_bad_lines='skip')
train = train0.copy()
train.head(3)
test0 = pd.read_csv('/content/test.csv')
test = test0.copy()
test.head(3)
# train + test data all together
df0 = pd.concat([train,test], sort=False).reset_index(drop=True)
df = df0.copy()
df.head(3)
print('Train Data Shape:', train.shape)
print('Test Data Shape:', test.shape)
train.info()
print(f'Data shape (rows, columns): {df.shape}')
print(f'Number of total duplicate rows: {df.duplicated().sum()}')
print(f'Number of missing values in Train: {train.isnull().sum().sum()}')
print(f'Number of missing values in Test: {test.isnull().sum().sum()}')

def get_value_count(df, column_name):
    """
    This function calculates and returns a DataFrame with the value counts and
    their corresponding percentages for a specified column in the DataFrame.
    """


    vc = df[column_name].value_counts()
    vc_norm = df[column_name].value_counts(normalize=True)

    vc = vc.rename_axis(column_name).reset_index(name='counts')
```

```python
    vc_norm = vc_norm.rename_axis(column_name).reset_index(name='percent')
    vc_norm['percent'] = (vc_norm['percent'] * 100).map('{:.2f}%'.format)

    df_result = pd.concat([vc[column_name], vc['counts'], vc_norm['percent']],
axis=1)

    return df_result




# ========== User-Defined-Function for Missing Values ============
def missing_values(df):
    """This function calculates the missing values count and their percentage in
a DataFrame."""

    missing_count = df.isnull().sum()
    value_count = df.isnull().count()
    missing_percentage = round(missing_count / value_count * 100, 2)

    # Format the percentage as '0.00%' with % symbol
    missing_percentage_formatted = missing_percentage.map("{:.2f}%".format)
    # Create a DataFrame to store the results
    missing_df = pd.DataFrame({"count": missing_count, "percentage":
missing_percentage_formatted})

    return missing_df


# ============= Compare Missing Values (Train-Test ==============
def compare_missing_values(train, test):
    """
    Compares missing values between train and test datasets, returning counts,
percentages, and data types.
    """
    def missing_data(df, label):
        missing_count = df.isna().sum()[df.isna().sum() > 0]
        total_count = len(df)
        missing_percentage = (missing_count / total_count *
100).map("{:.2f}%".format)
        return pd.DataFrame({
            f'{label} Missing Values': missing_count,
            f'{label} Missing Percentage': missing_percentage,
            f'{label} dtypes': df.dtypes[missing_count.index]
```

```python
        })

    # Get missing data for train and test
    train_missing_df = missing_data(train, 'Train')
    test_missing_df = missing_data(test, 'Test')

    # Concatenate the missing values side by side
    return pd.concat([train_missing_df, test_missing_df], axis=1)


# ========== Plotting Missing Values  ===========================
def na_ratio_plot(df):
    """Plots the ratio of missing values for each feature and prints the count of
missing values."""

    sns.displot(df.isna().melt(value_name='Missing_data',var_name='Features')\
                ,y='Features',hue='Missing_data',multiple='fill',aspect=9/8)

    print(df.isna().sum()[df.isna().sum()>0])




#========== Detecting Non-Numerical Characters ===========================

import re

def find_non_numeric_values(df, column_name):
    """
    Finds unique non-numeric values in a specified column of the DataFrame.
    """
    pattern = r'\D+'  # Pattern to match non-numeric characters
    # Find and flatten non-numeric values, then ensure uniqueness with set
    return set(re.findall(pattern, ' '.join(df[column_name].astype(str))))

# Comparing missing values in Train and Test data
compare_missing_values(train, test)
# TRAIN DATASET
sns.heatmap(train.isnull())
# TEST DATASET
sns.heatmap(test.isnull())
# Name column unique values and percentage
get_value_count(df, 'Name')
# column unique values and percentage
```

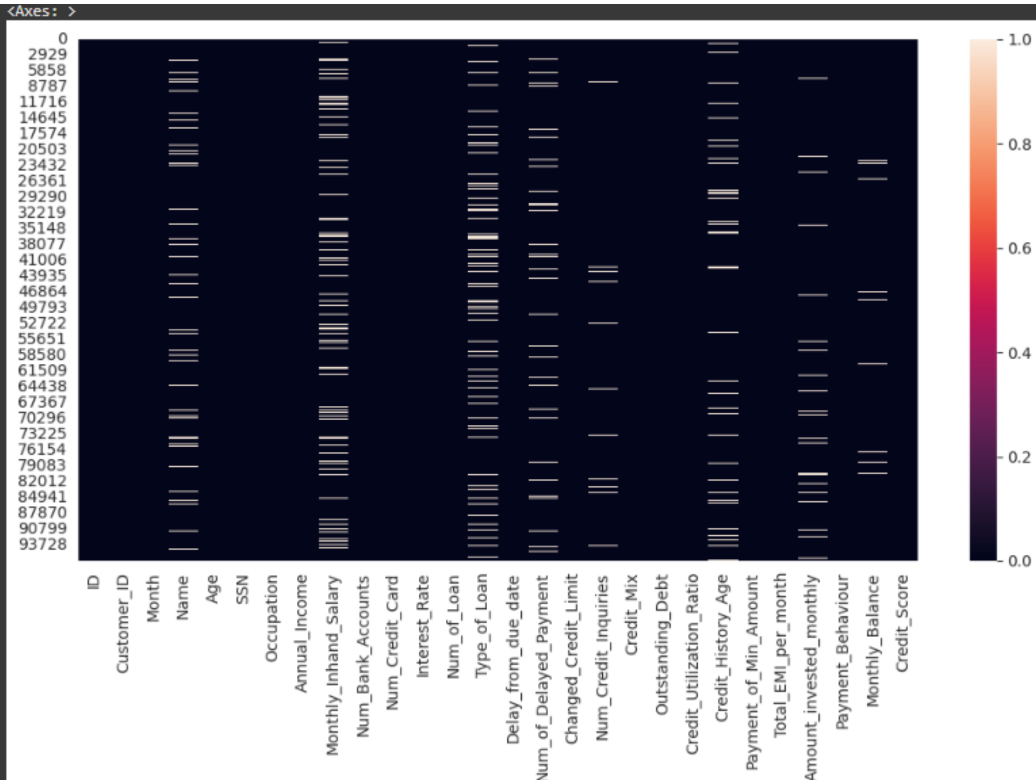
```
get_value_count(train, 'Annual_Income')
# Check missing values and dtype

print('Remaining missing values in Train:', train['Annual_Income'].isna().sum())
print('Remaining missing values in Test:', test['Annual_Income'].isna().sum())
print('dtype: ', train['Monthly_Inhand_Salary'].dtypes)

# Check the unusual-non-numeric values
find_non_numeric_values(train, 'Annual_Income')
# Plot Average Annual Income by Credit Score

plt.figure(figsize=(10, 5))
ax = sns.barplot(x='Credit_Score', y='Annual_Income', data=train, ci=None,
palette='Greens_r')

# Add values on top of the bars
for p in ax.patches: ax.annotate(format(p.get_height(), '.2f'),
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center',
                    xytext=(0, 9), textcoords='offset points')

plt.title('Average Annual Income by Credit Score')
plt.xlabel('Credit Score')
plt.ylabel('Average Annual Income')

plt.show()

# TEST DATA
# For each of the most common loan types (excluding the first one) in the test
dataset
for i in test['Type_of_Loan'].value_counts().head(9).index[1:]:

    # Create a new column for each loan type in the test dataset
    # The new column will be 1 if the loan type is present in 'Type_of_Loan', 0
otherwise
    test[i] = test['Type_of_Loan'].str.contains(i, na=False).astype(int)

# Delete the original 'Type_of_Loan' column after creating binary columns in the
test dataset
del test['Type_of_Loan']

# Display the first few rows of the modified test dataframe
test.head(3)
```

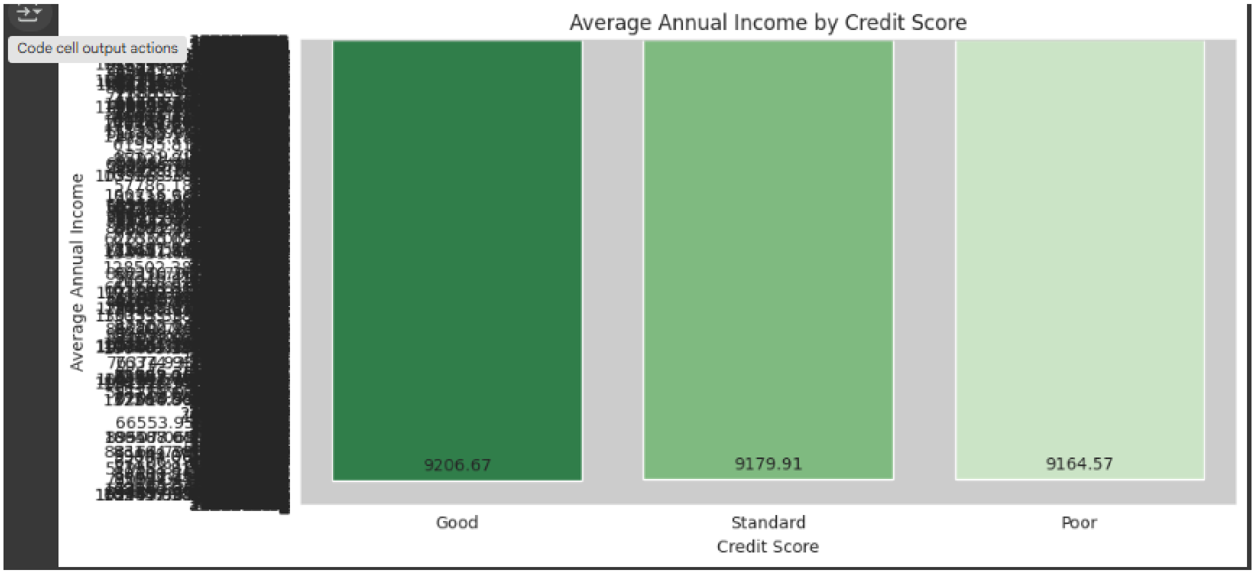
```python
# Check the column' unique values and percentage
get_value_count(train, 'Num_of_Delayed_Payment')
# Check the column' unique values and percentage
get_value_count(train, 'Credit_History_Age')
# Check the column' unique values and percentage
get_value_count(train, 'Age')
```

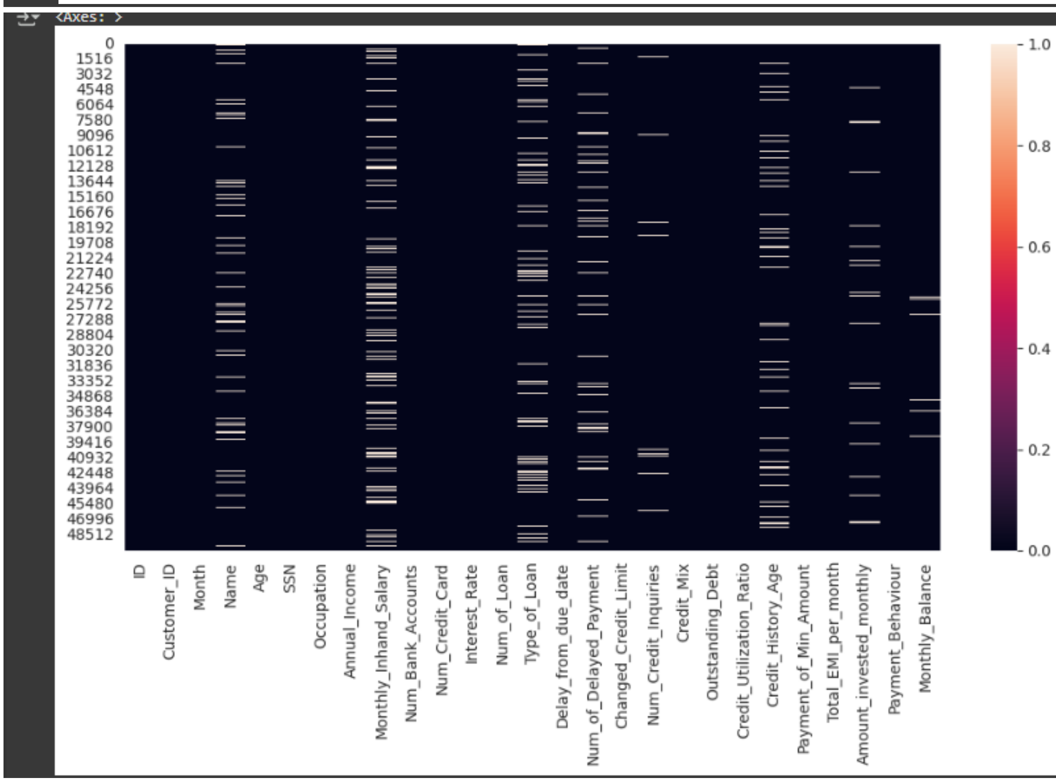
# 4. Output

| | Train Missing Values | Train Missing Percentage | Train dtypes | Test Missing Values | Test Missing Percentage | Test dtypes |
|---|---|---|---|---|---|---|
| Customer_ID | 1 | 0.00% | object | NaN | NaN | NaN |
| Month | 1 | 0.00% | object | NaN | NaN | NaN |
| Name | 9650 | 9.98% | object | 5015.000 | 10.03% | object |
| Age | 2 | 0.00% | object | NaN | NaN | NaN |
| SSN | 2 | 0.00% | object | NaN | NaN | NaN |
| Occupation | 2 | 0.00% | object | NaN | NaN | NaN |
| Annual_Income | 2 | 0.00% | object | NaN | NaN | NaN |
| Monthly_Inhand_Salary | 14517 | 15.02% | object | 7498.000 | 15.00% | float64 |
| Num_Bank_Accounts | 2 | 0.00% | object | NaN | NaN | NaN |
| Num_Credit_Card | 2 | 0.00% | float64 | NaN | NaN | NaN |
| Interest_Rate | 2 | 0.00% | object | NaN | NaN | NaN |
| Num_of_Loan | 4 | 0.00% | object | NaN | NaN | NaN |
| Type_of_Loan | 11041 | 11.42% | object | 5704.000 | 11.41% | object |
| Delay_from_due_date | 4 | 0.00% | object | NaN | NaN | NaN |
| Num_of_Delayed_Payment | 6765 | 7.00% | object | 3498.000 | 7.00% | object |
| Changed_Credit_Limit | 5 | 0.01% | object | NaN | NaN | NaN |
| Num_Credit_Inquiries | 1906 | 1.97% | object | 1035.000 | 2.07% | float64 |
| Credit_Mix | 5 | 0.01% | object | NaN | NaN | NaN |
| Outstanding_Debt | 5 | 0.01% | object | NaN | NaN | NaN |
| Credit_Utilization_Ratio | 5 | 0.01% | object | NaN | NaN | NaN |
| Credit_History_Age | 8771 | 9.08% | object | 4470.000 | 8.94% | object |
| Payment_of_Min_Amount | 5 | 0.01% | object | NaN | NaN | NaN |
| Total_EMI_per_month | 6 | 0.01% | object | NaN | NaN | NaN |
| Amount_invested_monthly | 4331 | 4.48% | object | 2271.000 | 4.54% | object |
| Payment_Behaviour | 7 | 0.01% | object | NaN | NaN | NaN |
| Monthly_Balance | 1169 | 1.21% | object | 562.000 | 1.12% | object |
| Credit_Score | 7 | 0.01% | object | NaN | NaN | NaN |

<Axes: >






# 5. Reference

CSV file used:  Test.csv, Train.csv