Individual Assignment 1 Task 1

Name: Rohit Panda

UOW ID: 8943060

Before Running the code, ensure you have the kaggle.json file to upload into the Colab.

Download the kaggle.json from this link: [Download Now]

```
from google.colab import files  # if using Colab
files.upload()  # upload kaggle.json

# Move to the correct path
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle.json to kaggle.json

```
# Step 0: Import necessary libraries
import pandas as pd
import numpy as np
import os


# Step 1: Set up Kaggle API credentials (Ensure kaggle.json is placed in ~/.kaggle or current directory)
# If not done already, download your API token from Kaggle Account > API > Create New Token

# Optional: Automatically set Kaggle API key location if in current directory
if not os.path.exists(os.path.expanduser("~/.kaggle/kaggle.json")) and os.path.exists("kaggle.json"):
    os.makedirs(os.path.expanduser("~/.kaggle"), exist_ok=True)
    os.rename("kaggle.json", os.path.expanduser("~/.kaggle/kaggle.json"))
    os.chmod(os.path.expanduser("~/.kaggle/kaggle.json"), 0o600)


# Step 2: Download dataset from Kaggle
print("=== Downloading dataset from Kaggle ===")
os.system("kaggle datasets download -d parisrohan/credit-score-classification")
```

```
=== Downloading dataset from Kaggle ===
0
```

```
# Step 3: Unzip the downloaded file
import zipfile
import os

zip_file_name = "credit-score-classification.zip"
if os.path.exists(zip_file_name):
    with zipfile.ZipFile(zip_file_name, "r") as zip_ref:
        zip_ref.extractall("credit_data")
    print(f"Successfully unzipped {zip_file_name} to credit_data/")

    # Step 4: Load the train.csv file
    # Assuming 'train.csv' is inside the unzipped folder 'credit_data'
    train_csv_path = "credit_data/train.csv"
    if os.path.exists(train_csv_path):
        df = pd.read_csv(train_csv_path)
        print("Dataset loaded from Kaggle successfully!")
    else:
        print(f"Error: train.csv not found in {train_csv_path}")

else:
    print(f"Error: {zip_file_name} not found. Please ensure the dataset was downloaded correctly in the previous step.")
```

```
Successfully unzipped credit-score-classification.zip to credit_data/
Dataset loaded from Kaggle successfully!
```

```
/tmp/ipython-input-29-3166859202.py:15: DtypeWarning: Columns (26) have mixed types. Specify dtype option on import or set low_memo
  df = pd.read_csv(train_csv_path)
```

Step 1: Creating Pandas DataFrame

In this step, I loaded the `train.csv` file from the Kaggle dataset "Credit Score Classification" using the Kaggle API. I used the Pandas library to read the CSV into a DataFrame, which allows for easy manipulation and analysis of the data. I then displayed the shape of the dataset, column names, and the first few rows to get an overview of the data.

```python
# (1) Create one Pandas data frame for this data set
print("=== Task 1: Credit Score Classification Dataset ===")
print("\n1. Creating Pandas DataFrame")

# Load the dataset
# Note: Replace 'credit_score_data.csv' with your actual file path
df = pd.read_csv('credit_data/train.csv')

print(f"Dataset shape: {df.shape}")
print(f"Column names: {list(df.columns)}")
print("\nFirst 5 rows:")
print(df.head())
```

```
=== Task 1: Credit Score Classification Dataset ===

1. Creating Pandas DataFrame
Dataset shape: (100000, 28)
Column names: ['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation', 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bar

First 5 rows:
       ID Customer_ID     Month           Name   Age          SSN Occupation  \
0  0x1602   CUS_0xd40   January  Aaron Maashoh    23  821-00-0265  Scientist
1  0x1603   CUS_0xd40  February  Aaron Maashoh    23  821-00-0265  Scientist
2  0x1604   CUS_0xd40     March  Aaron Maashoh  -500  821-00-0265  Scientist
3  0x1605   CUS_0xd40     April  Aaron Maashoh    23  821-00-0265  Scientist
4  0x1606   CUS_0xd40       May  Aaron Maashoh    23  821-00-0265  Scientist

  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  ...  Credit_Mix  \
0      19114.12            1824.843333                  3  ...           _
1      19114.12                    NaN                  3  ...        Good
2      19114.12                    NaN                  3  ...        Good
3      19114.12                    NaN                  3  ...        Good
4      19114.12            1824.843333                  3  ...        Good

   Outstanding_Debt Credit_Utilization_Ratio     Credit_History_Age  \
0            809.98                26.822620  22 Years and 1 Months
1            809.98                31.944960                    NaN
2            809.98                28.609352  22 Years and 3 Months
3            809.98                31.377862  22 Years and 4 Months
4            809.98                24.797347  22 Years and 5 Months

   Payment_of_Min_Amount Total_EMI_per_month Amount_invested_monthly  \
0                     No           49.574949       80.41529543900253
1                     No           49.574949      118.28022162236736
2                     No           49.574949         81.699521264648
3                     No           49.574949       199.4580743910713
4                     No           49.574949      41.420153086217326

                 Payment_Behaviour     Monthly_Balance Credit_Score
0   High_spent_Small_value_payments  312.49408867943663         Good
1    Low_spent_Large_value_payments  284.62916249607184         Good
2   Low_spent_Medium_value_payments   331.2098628537912         Good
3    Low_spent_Small_value_payments  223.45130972736786         Good
4  High_spent_Medium_value_payments  341.48923103222177         Good

[5 rows x 28 columns]
/tmp/ipython-input-30-3257610822.py:7: DtypeWarning: Columns (26) have mixed types. Specify dtype option on import or set low_memor
  df = pd.read_csv('credit_data/train.csv')
```

Step 2: Identifying Missing Values and Cleaning One Attribute

I checked for missing values in all columns using `isnull().sum()` . If any missing values are found, I selected one column with missing data and proposed a cleaning method based on the column's data type:

- If the column is **numerical**, I applied **mean imputation**, replacing missing values with the mean of the column.
- If the column is **categorical**, I applied **mode imputation**, replacing missing values with the most frequent value.

This ensures that no information is lost by dropping rows and helps retain the dataset's structure for future analysis.

```
# (2) Identify the attributes with missing values
print("\n2. Identifying Missing Values")
print("Missing values per column:")
missing_values = df.isnull().sum()
print(missing_values)

# Show columns with missing values
columns_with_missing = missing_values[missing_values > 0]
print(f"\nColumns with missing values: {len(columns_with_missing)}")
print(columns_with_missing)

# Select one attribute with missing values and propose cleaning method
if len(columns_with_missing) > 0:
    # Select the first column with missing values
    selected_column = columns_with_missing.index[0]
    print(f"\nSelected column for cleaning: {selected_column}")
    print(f"Missing values in {selected_column}: {columns_with_missing[selected_column]}")

    # Check if it's numerical or categorical
    if df[selected_column].dtype in ['int64', 'float64']:
        # For numerical data, use mean imputation
        mean_value = df[selected_column].mean()
        df[f'{selected_column}_cleaned'] = df[selected_column].fillna(mean_value)
        print(f"Cleaning method: Mean imputation (value: {mean_value:.2f})")
    else:
        # For categorical data, use mode imputation
        mode_value = df[selected_column].mode()[0]
        df[f'{selected_column}_cleaned'] = df[selected_column].fillna(mode_value)
        print(f"Cleaning method: Mode imputation (value: {mode_value})")

    # Verify cleaning
    print(f"Missing values after cleaning: {df[f'{selected_column}_cleaned'].isnull().sum()}")
else:
    print("No missing values found in the dataset")
```

```
2. Identifying Missing Values
Missing values per column:
ID                              0
Customer_ID                     0
Month                           0
Name                         9985
Age                             0
SSN                             0
Occupation                      0
Annual_Income                   0
Monthly_Inhand_Salary       15002
Num_Bank_Accounts               0
Num_Credit_Card                 0
Interest_Rate                   0
Num_of_Loan                     0
Type_of_Loan                11408
Delay_from_due_date             0
Num_of_Delayed_Payment       7002
Changed_Credit_Limit            0
Num_Credit_Inquiries         1965
Credit_Mix                      0
Outstanding_Debt                0
Credit_Utilization_Ratio        0
Credit_History_Age           9030
Payment_of_Min_Amount           0
Total_EMI_per_month             0
Amount_invested_monthly      4479
Payment_Behaviour               0
Monthly_Balance              1200
Credit_Score                    0
dtype: int64

Columns with missing values: 8
Name                        9985
Monthly_Inhand_Salary      15002
Type_of_Loan               11408
Num_of_Delayed_Payment      7002
Num_Credit_Inquiries        1965
Credit_History_Age          9030
Amount_invested_monthly     4479
Monthly_Balance             1200
dtype: int64
```

```
Selected column for cleaning: Name
Missing values in Name: 9985
Cleaning method: Mode imputation (value: Langep)
Missing values after cleaning: 0
```

Step 3: Z-score Normalization of "Amount_invested_monthly"

Z-score normalization transforms the "Amount_invested_monthly" column so that its values have a **mean of 0** and a **standard deviation of 1**. This is done using the formula:

[Z = \frac{X - \mu}{\sigma}]

Where:

- (X) is the original value
- (\mu) is the mean
- (\sigma) is the standard deviation

This transformation is important when comparing features with different scales or for use in algorithms that are sensitive to feature magnitude.

```
# (3) Perform z-score normalization on "Amount_invested_monthly"
print("\n3. Z-score Normalization of Amount_invested_monthly")

if 'Amount_invested_monthly' in df.columns:
    # Convert the column to numeric, coercing errors
    df['Amount_invested_monthly'] = pd.to_numeric(df['Amount_invested_monthly'], errors='coerce')

    # Calculate mean and standard deviation, ignoring NaN values
    mean_amount = df['Amount_invested_monthly'].mean()
    std_amount = df['Amount_invested_monthly'].std()

    print(f"Original mean: {mean_amount:.6f}")
    print(f"Original standard deviation: {std_amount:.6f}")

    # Perform z-score normalization, handling potential NaN values
    df['Amount_invested_monthly_zscore'] = (df['Amount_invested_monthly'] - mean_amount) / std_amount

    # Calculate mean and variance of normalized values, ignoring NaN values
    normalized_mean = df['Amount_invested_monthly_zscore'].mean()
    normalized_variance = df['Amount_invested_monthly_zscore'].var()

    print(f"Normalized mean: {normalized_mean:.10f}")
    print(f"Normalized variance: {normalized_variance:.10f}")

    print("\nFirst 10 normalized values:")
    print(df['Amount_invested_monthly_zscore'].head(10))
else:
    print("Column 'Amount_invested_monthly' not found in dataset")
```

```
3. Z-score Normalization of Amount_invested_monthly
Original mean: 195.539456
Original standard deviation: 199.564527
Normalized mean: 0.0000000000
Normalized variance: 1.0000000000

First 10 normalized values:
0   -0.576877
1   -0.387139
2   -0.570442
3    0.019636
4   -0.772278
5   -0.666999
6   -0.086165
7   -0.855634
8   -0.457234
9   -0.777434
Name: Amount_invested_monthly_zscore, dtype: float64
```

Step 4: Creating Four Equal-Frequency Bins for "Amount_invested_monthly"

I used the `pd.qcut()` function to divide the "Amount_invested_monthly" column into four bins with approximately equal numbers of records in each bin. This is known as **equal-frequency binning** or **quantile binning**.

The bin ranges are automatically determined based on the quartiles of the data, and each record is assigned to one of the bins (Bin1 through Bin4). This method helps in categorizing continuous variables into discrete groups for analysis or visualization.

```
# (4) Create four bins for "Amount_invested_monthly" with equivalent numbers of records
print("\n4. Creating Four Equal-Frequency Bins")

if 'Amount_invested_monthly' in df.columns:
    # Use pd.qcut for equal-frequency binning
    df['Amount_invested_monthly_bins'] = pd.qcut(df['Amount_invested_monthly'],
                                                 q=4,
                                                 labels=['Bin1', 'Bin2', 'Bin3', 'Bin4'])

    # Check the distribution of bins
    bin_counts = df['Amount_invested_monthly_bins'].value_counts().sort_index()
    print("Bin distribution:")
    print(bin_counts)

    # Show bin ranges
    print("\nBin ranges:")
    bin_ranges = pd.qcut(df['Amount_invested_monthly'], q=4, retbins=True)[1]
    for i, (start, end) in enumerate(zip(bin_ranges[:-1], bin_ranges[1:])):
        print(f"Bin{i+1}: [{start:.2f}, {end:.2f}]")
else:
    print("Column 'Amount_invested_monthly' not found in dataset")
```

```
4. Creating Four Equal-Frequency Bins
Bin distribution:
Amount_invested_monthly_bins
Bin1    22804
Bin2    22804
Bin3    22804
Bin4    22804
Name: count, dtype: int64

Bin ranges:
Bin1: [0.00, 72.24]
Bin2: [72.24, 128.95]
Bin3: [128.95, 236.82]
Bin4: [236.82, 1977.33]
```

Step 5: One-Hot Encoding of "Credit_Mix"

One-hot encoding is a method to convert categorical data into a numerical format that can be used in analysis or modeling. For the "Credit_Mix" column, I applied one-hot encoding using `pd.get_dummies()`.

Each unique value in "Credit_Mix" is converted into a separate binary column:

- If the original value is present in a row, the column has a value of 1.
- Otherwise, it has a value of 0.

This avoids introducing any ordinal relationship between categories and ensures that models (in future work) interpret each category independently.

```
# (5) Apply one-hot-encoding to "Credit_Mix"
print("\n5. One-Hot Encoding of Credit_Mix")

if 'Credit_Mix' in df.columns:
    # Check unique values
    print(f"Unique values in Credit_Mix: {df['Credit_Mix'].unique()}")

    # Apply one-hot encoding
    credit_mix_encoded = pd.get_dummies(df['Credit_Mix'], prefix='Credit_Mix')

    # Append to dataframe
    df = pd.concat([df, credit_mix_encoded], axis=1)

    print(f"One-hot encoded columns: {list(credit_mix_encoded.columns)}")
    print("\nFirst 5 rows of encoded data:")
    print(credit_mix_encoded.head())
else:
    print("Column 'Credit_Mix' not found in dataset")
```

```
# Final dataframe summary
print("\n=== Final DataFrame Summary ===")
print(f"Final shape: {df.shape}")
print(f"Final columns: {list(df.columns)}")

# Display first few rows of the final dataframe
print("\nFirst 3 rows of final dataframe:")
print(df.head(3))

# Check for any remaining missing values
print(f"\nTotal missing values in final dataframe: {df.isnull().sum().sum()}")
```

```
5. One-Hot Encoding of Credit_Mix
Unique values in Credit_Mix: ['_' 'Good' 'Standard' 'Bad']
One-hot encoded columns: ['Credit_Mix_Bad', 'Credit_Mix_Good', 'Credit_Mix_Standard', 'Credit_Mix__']

First 5 rows of encoded data:
   Credit_Mix_Bad  Credit_Mix_Good  Credit_Mix_Standard  Credit_Mix__
0           False            False                False          True
1           False             True                False         False
2           False             True                False         False
3           False             True                False         False
4           False             True                False         False

=== Final DataFrame Summary ===
Final shape: (100000, 35)
Final columns: ['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation', 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Ba

First 3 rows of final dataframe:
       ID Customer_ID     Month           Name   Age          SSN Occupation  \
0  0x1602   CUS_0xd40   January  Aaron Maashoh    23  821-00-0265  Scientist
1  0x1603   CUS_0xd40  February  Aaron Maashoh    23  821-00-0265  Scientist
2  0x1604   CUS_0xd40     March  Aaron Maashoh  -500  821-00-0265  Scientist

  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  ...  \
0      19114.12            1824.843333                  3  ...
1      19114.12                    NaN                  3  ...
2      19114.12                    NaN                  3  ...

                 Payment_Behaviour     Monthly_Balance Credit_Score  \
0  High_spent_Small_value_payments  312.49408867943663         Good
1   Low_spent_Large_value_payments  284.62916249607184         Good
2  Low_spent_Medium_value_payments   331.2098628537912         Good

    Name_cleaned  Amount_invested_monthly_zscore Amount_invested_monthly_bins  \
0  Aaron Maashoh                       -0.576877                         Bin2
1  Aaron Maashoh                       -0.387139                         Bin2
2  Aaron Maashoh                       -0.570442                         Bin2

  Credit_Mix_Bad  Credit_Mix_Good Credit_Mix_Standard Credit_Mix__
0          False            False               False         True
1          False             True               False        False
2          False             True               False        False

[3 rows x 35 columns]

Total missing values in final dataframe: 81944
```

END OF ASSIGNMENT 1 TASK 1