Financial Transactions Dataset Analytics

January 29, 2025

1 Financial Transactions Dataset Analytics

Dataset Description

This comprehensive financial dataset combines transaction records, customer information, and card data from a banking institution, spanning across the 2010s decade.

1. **Transactions Data:** Details of financial transactions, including amounts, dates, merchant information, and transaction types.
2. **Users Data:** Customer demographic and financial details, including income, debt, credit scores, and age groups.
3. **Cards Data:** Information on issued cards, including card type, credit limits, account opening dates, and expiry dates.
4. **Merchant Categories (MCC Codes):** Mapping of merchant category codes to their respective business categories.

The dataset can be found here: **Financial Transactions Dataset Analytics**

Business Scenario

The dataset represents the banking industry's transactional and operational data, helping:

- Understand customer behavior and spending patterns.
- Identify risks like potential fraud and defaults.
- Enhance operational efficiency by addressing transaction failures and customer inactivity.
- Develop targeted marketing strategies for high-value customer segments.
- Support strategic decision-making based on key trends and insights.

Methodology

1. **Data Preparation:**
   - Cleaned, standardized, and merged the datasets for analysis.
   - Added calculated fields (e.g., credit utilization, debt-to-income ratio, transaction categories).
2. **Exploratory Data Analysis (EDA):**
   - Filtered, grouped, and sorted data to examine specific trends.
   - Created calculated metrics like total revenue, transaction volume, and customer spending patterns.
3. **Data Visualization:**

- Used Python libraries like Matplotlib, Seaborn, and Plotly to create visualizations (e.g., bar charts, boxplots, heatmaps, and line plots).
- Employed SQL (with SQLAlchemy) to create a database and execute complex queries for deeper insights.

1.1 Import necessary libraries

[9]:
```
# For data manipulation and analysis
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None) # Set pandas to check all columns
pd.options.display.float_format = '{:.2f}'.format # Set the global float format␣
↪to show 2 decimal places

from janitor import clean_names

# For data visualization
import matplotlib.pyplot as plt
import seaborn as sns

import plotly.express as px
import plotly.graph_objects as go


# For SQL Integration
from sqlalchemy import create_engine, text

# To suppress warnings in jupyter notebook
import warnings
warnings.filterwarnings("ignore")
```

1.2 Load data files

[11]:
```
transaction_file_path = "transactions_data.csv"  # File path to the␣
↪transactions data
users_file_path = "users_data.csv"  # File path to the users data
cards_file_path = "cards_data.csv"  # File path to the cards data
mcc_codes_file_path = "mcc_codes.json" # File path to the MCC codes data
```

[12]:
```
# Load data files to data frames
transactions_df = pd.read_csv(transaction_file_path)
users_df = pd.read_csv(users_file_path)
cards_df = pd.read_csv(cards_file_path)

mcc_codes_df = pd.read_json(mcc_codes_file_path, typ='series').reset_index()
mcc_codes_df.columns = ['mcc_code', 'category']
```

1.3 Inspect the datasets

1.3.1 Transactions data:

Dataframe information

[16]:
```
transactions_df = clean_names(transactions_df)
transactions_df.columns
```

[16]:
```
Index(['id', 'date', 'client_id', 'card_id', 'amount', 'use_chip',
       'merchant_id', 'merchant_city', 'merchant_state', 'zip', 'mcc',
       'errors'],
      dtype='object')
```

[17]:
```
transactions_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13305915 entries, 0 to 13305914
Data columns (total 12 columns):
 #   Column          Dtype
---  ------          -----
 0   id              int64
 1   date            object
 2   client_id       int64
 3   card_id         int64
 4   amount          object
 5   use_chip        object
 6   merchant_id     int64
 7   merchant_city   object
 8   merchant_state  object
 9   zip             float64
 10  mcc             int64
 11  errors          object
dtypes: float64(1), int64(5), object(6)
memory usage: 1.2+ GB
```

Display initial rows

[19]:
```
transactions_df.head()
```

[19]:
```
        id                 date  client_id  card_id   amount  \
0  7475327  2010-01-01 00:01:00       1556     2972  $-77.00
1  7475328  2010-01-01 00:02:00        561     4575   $14.57
2  7475329  2010-01-01 00:02:00       1129      102   $80.00
3  7475331  2010-01-01 00:05:00        430     2860  $200.00
4  7475332  2010-01-01 00:06:00        848     3915   $46.41

           use_chip  merchant_id merchant_city merchant_state      zip   mcc  \
0  Swipe Transaction        59935        Beulah             ND 58523.00  5499
1  Swipe Transaction        67570    Bettendorf             IA 52722.00  5311
```

```
2  Swipe Transaction        27092         Vista             CA 92084.00   4829
3  Swipe Transaction        27092   Crown Point             IN 46307.00   4829
4  Swipe Transaction        13051       Harwood             MD 20776.00   5813

  errors
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
```

Display unique categorical values

[21]:
```
transactions_df.use_chip.value_counts()
```

[21]:
```
use_chip
Swipe Transaction     6967185
Chip Transaction      4780818
Online Transaction    1557912
Name: count, dtype: int64
```

[22]:
```
transactions_df.errors.value_counts()
```

[22]:
```
errors
Insufficient Balance                                  130902
Bad PIN                                                32119
Technical Glitch                                       26271
Bad Card Number                                         7767
Bad Expiration                                          6161
Bad CVV                                                 6106
Bad Zipcode                                             1126
Bad PIN,Insufficient Balance                             293
Insufficient Balance,Technical Glitch                    243
Bad Card Number,Insufficient Balance                      71
Bad PIN,Technical Glitch                                  70
Bad CVV,Insufficient Balance                              57
Bad Expiration,Insufficient Balance                       47
Bad Card Number,Bad CVV                                   38
Bad Card Number,Bad Expiration                            33
Bad Expiration,Bad CVV                                    32
Bad Expiration,Technical Glitch                           21
Bad Card Number,Technical Glitch                          15
Bad CVV,Technical Glitch                                   8
Bad Zipcode,Insufficient Balance                           7
Bad Zipcode,Technical Glitch                               5
Bad Card Number,Bad Expiration,Insufficient Balance        1
Name: count, dtype: int64
```

Display missing values

[24]:
```
transactions_df.isnull().sum()
```

[24]:
```
id                       0
date                     0
client_id                0
card_id                  0
amount                   0
use_chip                 0
merchant_id              0
merchant_city            0
merchant_state     1563700
zip                1652706
mcc                      0
errors            13094522
dtype: int64
```

Check for duplicate rows

[26]:
```
print(f"Number of Duplicate Rows: {transactions_df.duplicated().sum()}")
```

```
Number of Duplicate Rows: 0
```

Display dataset shape

[28]:
```
print(f"Dataset Shape: {transactions_df.shape[0]} rows, {transactions_df.
  ↪shape[1]} columns")
```

```
Dataset Shape: 13305915 rows, 12 columns
```

1.3.2 Users data:

Dataframe information

[31]:
```
users_df = clean_names(users_df)
users_df.columns
```

[31]:
```
Index(['id', 'current_age', 'retirement_age', 'birth_year', 'birth_month',
       'gender', 'address', 'latitude', 'longitude', 'per_capita_income',
       'yearly_income', 'total_debt', 'credit_score', 'num_credit_cards'],
      dtype='object')
```

[32]:
```
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 2000 non-null   int64
 1   current_age        2000 non-null   int64
 2   retirement_age     2000 non-null   int64
```

```
 3   birth_year         2000 non-null   int64
 4   birth_month        2000 non-null   int64
 5   gender             2000 non-null   object
 6   address            2000 non-null   object
 7   latitude           2000 non-null   float64
 8   longitude          2000 non-null   float64
 9   per_capita_income  2000 non-null   object
 10  yearly_income      2000 non-null   object
 11  total_debt         2000 non-null   object
 12  credit_score       2000 non-null   int64
 13  num_credit_cards   2000 non-null   int64
dtypes: float64(2), int64(7), object(5)
memory usage: 218.9+ KB
```

[33]:
```
users_df.head()
```

[33]:
```
     id  current_age  retirement_age  birth_year  birth_month  gender  \
0   825           53              66        1966           11  Female
1  1746           53              68        1966           12  Female
2  1718           81              67        1938           11  Female
3   708           63              63        1957            1  Female
4  1164           43              70        1976            9    Male

                    address  latitude  longitude per_capita_income  \
0             462 Rose Lane     34.15    -117.76            $29278
1    3606 Federal Boulevard     40.76     -73.74            $37891
2           766 Third Drive     34.02    -117.89            $22681
3          3 Madison Street     40.71     -73.99           $163145
4  9620 Valley Stream Drive     37.76    -122.44            $53797

  yearly_income total_debt  credit_score  num_credit_cards
0        $59696    $127613           787                 5
1        $77254    $191349           701                 5
2        $33483       $196           698                 5
3       $249925    $202328           722                 4
4       $109687    $183855           675                 1
```

Display unique categorical values

[35]:
```
users_df.gender.value_counts()
```

[35]:
```
gender
Female    1016
Male       984
Name: count, dtype: int64
```

[36]:
```
users_df.num_credit_cards.value_counts()
```

[36]:
```
num_credit_cards
3    449
1    416
2    388
4    376
5    206
6    105
7     40
8     17
9      3
Name: count, dtype: int64
```

[37]:
```python
users_df.isnull().sum()
```

[37]:
```
id                   0
current_age          0
retirement_age       0
birth_year           0
birth_month          0
gender               0
address              0
latitude             0
longitude            0
per_capita_income    0
yearly_income        0
total_debt           0
credit_score         0
num_credit_cards     0
dtype: int64
```

[38]:
```python
print(f"Number of Duplicate Rows: {users_df.duplicated().sum()}")
```

```
Number of Duplicate Rows: 0
```

[39]:
```python
print(f"Dataset Shape: {users_df.shape[0]} rows, {users_df.shape[1]} columns")
```

```
Dataset Shape: 2000 rows, 14 columns
```

1.3.3 Cards data:

Dataframe information

[42]:
```python
cards_df = clean_names(cards_df)
cards_df.columns
```

[42]:
```
Index(['id', 'client_id', 'card_brand', 'card_type', 'card_number', 'expires',
       'cvv', 'has_chip', 'num_cards_issued', 'credit_limit', 'acct_open_date',
       'year_pin_last_changed', 'card_on_dark_web'],
      dtype='object')
```

[43]:
```python
cards_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6146 entries, 0 to 6145
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   id                     6146 non-null   int64 
 1   client_id              6146 non-null   int64 
 2   card_brand             6146 non-null   object
 3   card_type              6146 non-null   object
 4   card_number            6146 non-null   int64 
 5   expires                6146 non-null   object
 6   cvv                    6146 non-null   int64 
 7   has_chip               6146 non-null   object
 8   num_cards_issued       6146 non-null   int64 
 9   credit_limit           6146 non-null   object
 10  acct_open_date         6146 non-null   object
 11  year_pin_last_changed  6146 non-null   int64 
 12  card_on_dark_web       6146 non-null   object
dtypes: int64(6), object(7)
memory usage: 624.3+ KB
```

[44]:
```python
cards_df.head()
```

[44]:
```
     id  client_id  card_brand        card_type       card_number  expires  \
0  4524        825        Visa            Debit  4344676511950444  12/2022   
1  2731        825        Visa            Debit  4956965974959986  12/2020   
2  3701        825        Visa            Debit  4582313478255491  02/2024   
3    42        825        Visa           Credit  4879494103069057  08/2024   
4  4659        825  Mastercard  Debit (Prepaid)  5722874738736011  03/2009   

   cvv has_chip  num_cards_issued credit_limit acct_open_date  \
0  623      YES                 2       $24295        09/2002   
1  393      YES                 2       $21968        04/2014   
2  719      YES                 2       $46414        07/2003   
3  693       NO                 1       $12400        01/2003   
4   75      YES                 1          $28        09/2008   

   year_pin_last_changed card_on_dark_web  
0                   2008               No  
1                   2014               No  
2                   2004               No  
3                   2012               No  
4                   2009               No  
```

Display unique categorical values

[46]:
```python
cards_df.card_brand.value_counts()
```

[46]:
```
card_brand
Mastercard    3209
Visa          2326
Amex           402
Discover       209
Name: count, dtype: int64
```

[47]:
```python
cards_df.card_type.value_counts()
```

[47]:
```
card_type
Debit              3511
Credit             2057
Debit (Prepaid)     578
Name: count, dtype: int64
```

[48]:
```python
cards_df.has_chip.value_counts()
```

[48]:
```
has_chip
YES    5500
NO      646
Name: count, dtype: int64
```

[49]:
```python
cards_df.card_on_dark_web.value_counts()
```

[49]:
```
card_on_dark_web
No    6146
Name: count, dtype: int64
```

[50]:
```python
cards_df.isnull().sum()
```

[50]:
```
id                       0
client_id                0
card_brand               0
card_type                0
card_number              0
expires                  0
cvv                      0
has_chip                 0
num_cards_issued         0
credit_limit             0
acct_open_date           0
year_pin_last_changed    0
card_on_dark_web         0
dtype: int64
```

[51]:
```python
print(f"Number of Duplicate Rows: {cards_df.duplicated().sum()}")
```

```
Number of Duplicate Rows: 0
```

[52]:
```python
print(f"Dataset Shape: {cards_df.shape[0]} rows, {cards_df.shape[1]} columns")
```

```
Dataset Shape: 6146 rows, 13 columns
```

1.3.4 MCC Codes data:

Dataframe information

[55]:
```python
mcc_codes_df = clean_names(mcc_codes_df)
mcc_codes_df.columns
```

[55]:
```
Index(['mcc_code', 'category'], dtype='object')
```

[56]:
```python
mcc_codes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109 entries, 0 to 108
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   mcc_code  109 non-null    int64
 1   category  109 non-null    object
dtypes: int64(1), object(1)
memory usage: 1.8+ KB
```

[57]:
```python
mcc_codes_df.head()
```

[57]:

	mcc_code	category
0	5812	Eating Places and Restaurants
1	5541	Service Stations
2	7996	Amusement Parks, Carnivals, Circuses
3	5411	Grocery Stores, Supermarkets
4	4784	Tolls and Bridge Fees

[58]:
```python
mcc_codes_df.isnull().sum()
```

[58]:
```
mcc_code    0
category    0
dtype: int64
```

[59]:
```python
print(f"Number of Duplicate Rows: {mcc_codes_df.duplicated().sum()}")
```

```
Number of Duplicate Rows: 0
```

[60]:
```python
print(f"Dataset Shape: {mcc_codes_df.shape[0]} rows, {mcc_codes_df.shape[1]}␣
↪columns")
```

```
Dataset Shape: 109 rows, 2 columns
```

1.4 Data Cleaning

Clean each dataset, to ensure data is standardised, column names are consistent, and unnecessary fields are removed.

Cleaning transactions_data:

Clean missing values by filling with appropriate values

[65]:
```
# Replace missing values in the 'errors' column with 'No Error'
transactions_df['errors'] = transactions_df['errors'].fillna('No Error')

# Replace missing values in the 'merchant_state' column with 'Unknown'
transactions_df['merchant_state'] = transactions_df['merchant_state'].
↪fillna('Unknown')

# Replace missing values in the 'zip' column with 'Unknown'
transactions_df['zip'] = (transactions_df['zip'].apply(lambda x: str(int(x)) if␣
↪pd.notna(x) else 'Unknown'))
```

[66]:
```
pd.isnull(transactions_df).sum()
```

[66]:
```
id                0
date              0
client_id         0
card_id           0
amount            0
use_chip          0
merchant_id       0
merchant_city     0
merchant_state    0
zip               0
mcc               0
errors            0
dtype: int64
```

Remove special characters from 'amount' and convert it to numeric

[68]:
```
def clean_amount_column(df, column_name):
    """
    Cleans a specified column in a DataFrame by removing special characters␣
↪(like $ and commas) and converting it to numeric values.
    Parameters:
        df (pd.DataFrame): The DataFrame containing the column to clean.
        column_name (str): The name of the column to clean.
    Returns:
        pd.Series: A cleaned pandas Series with numeric values.
    """
```

```python
    # Ensure the column exists in the DataFrame
    if column_name not in df.columns:
        raise ValueError(f"Column '{column_name}' does not exist in the␣
↪DataFrame.")

    # Remove special characters and convert to float
    cleaned_column = (
        df[column_name]
        .astype(str)  # Ensure it's a string
        .str.replace('[$,]', '', regex=True)  # Remove $ and commas
        .str.strip()  # Remove any surrounding whitespace
        .replace('', '0')  # Replace empty strings with 0
        .astype(float)  # Convert to numeric type
    )

    return cleaned_column
```

[69]:
```python
transactions_df['amount'] = clean_amount_column(transactions_df, 'amount')
```

[70]:
```python
transactions_df['amount']
```

[70]:
```
0          -77.00
1           14.57
2           80.00
3          200.00
4           46.41
            …
13305910     1.11
13305911    12.80
13305912    40.44
13305913     4.00
13305914    12.88
Name: amount, Length: 13305915, dtype: float64
```

Converting 'date' column to datetime type

[72]:
```python
transactions_df['date'] = pd.to_datetime(transactions_df['date'])
```

Extract year, month, day, hour, and day of the week

[74]:
```python
transactions_df['transaction_year'] = transactions_df['date'].dt.year
transactions_df['transaction_month'] = transactions_df['date'].dt.month
transactions_df['transaction_day'] = transactions_df['date'].dt.day
transactions_df['transaction_hour'] = transactions_df['date'].dt.hour
transactions_df['transaction_day_of_week'] = transactions_df['date'].dt.
↪day_name()  # day name
```

[75]:
```python
transactions_df.dtypes
```

[75]:
```
id                                  int64
date                       datetime64[ns]
client_id                           int64
card_id                             int64
amount                            float64
use_chip                           object
merchant_id                         int64
merchant_city                      object
merchant_state                     object
zip                                object
mcc                                 int64
errors                             object
transaction_year                    int32
transaction_month                   int32
transaction_day                     int32
transaction_hour                    int32
transaction_day_of_week            object
dtype: object
```

Convert categorical values with numeric datatype to string

[77]:
```python
transactions_df['id'] = transactions_df['id'].astype('str')
transactions_df['client_id'] = transactions_df['client_id'].astype('str')
transactions_df['card_id'] = transactions_df['card_id'].astype('str')
transactions_df['merchant_id'] = transactions_df['merchant_id'].astype('str')
```

[78]:
```python
transactions_df.dtypes
```

[78]:
```
id                                 object
date                       datetime64[ns]
client_id                          object
card_id                            object
amount                            float64
use_chip                           object
merchant_id                        object
merchant_city                      object
merchant_state                     object
zip                                object
mcc                                 int64
errors                             object
transaction_year                    int32
transaction_month                   int32
transaction_day                     int32
transaction_hour                    int32
transaction_day_of_week            object
dtype: object
```

Merge transactions_df with mcc_codes_df on the 'mcc' column

[80]:
```python
transactions_df = transactions_df.merge(
    mcc_codes_df,
    how='left',              # Perform a left join to keep all rows in␣
 ↪transactions_df
    left_on='mcc',           # Column in transactions_df to join on
    right_on='mcc_code'      # Column in mcc_codes_df to join on
)

# Rename category column
transactions_df = transactions_df.rename(columns={'category': 'mcc_category'}).
 ↪drop(columns=['mcc_code'], axis=1)
```

[81]:
```python
transactions_df.head()
```

[81]:
```
        id                date  client_id  card_id  amount           use_chip  \
0  7475327 2010-01-01 00:01:00       1556     2972  -77.00  Swipe Transaction
1  7475328 2010-01-01 00:02:00        561     4575   14.57  Swipe Transaction
2  7475329 2010-01-01 00:02:00       1129      102   80.00  Swipe Transaction
3  7475331 2010-01-01 00:05:00        430     2860  200.00  Swipe Transaction
4  7475332 2010-01-01 00:06:00        848     3915   46.41  Swipe Transaction

  merchant_id merchant_city merchant_state    zip   mcc    errors  \
0       59935        Beulah             ND  58523  5499  No Error
1       67570    Bettendorf             IA  52722  5311  No Error
2       27092         Vista             CA  92084  4829  No Error
3       27092   Crown Point             IN  46307  4829  No Error
4       13051       Harwood             MD  20776  5813  No Error

   transaction_year  transaction_month  transaction_day  transaction_hour  \
0              2010                  1                1                 0
1              2010                  1                1                 0
2              2010                  1                1                 0
3              2010                  1                1                 0
4              2010                  1                1                 0

  transaction_day_of_week                        mcc_category
0                  Friday            Miscellaneous Food Stores
1                  Friday                    Department Stores
2                  Friday                       Money Transfer
3                  Friday                       Money Transfer
4                  Friday  Drinking Places (Alcoholic Beverages)
```

Cleaning users_data:

Convert income and debt columns to numeric after removing $.

[84]:
```python
users_df['per_capita_income'] = clean_amount_column(users_df,␣
 ↪'per_capita_income')
```

```python
users_df['yearly_income'] = clean_amount_column(users_df, 'yearly_income')
users_df['total_debt'] = clean_amount_column(users_df, 'total_debt')
```

convert categorical values with numeric datatype to string

[86]:
```python
users_df['id'] = users_df['id'].astype('str')
```

[87]:
```python
users_df.dtypes
```

[87]:
```
id                    object
current_age            int64
retirement_age         int64
birth_year             int64
birth_month            int64
gender                object
address               object
latitude             float64
longitude            float64
per_capita_income    float64
yearly_income        float64
total_debt           float64
credit_score           int64
num_credit_cards       int64
dtype: object
```

[88]:
```python
users_df.head()
```

[88]:
```
     id  current_age  retirement_age  birth_year  birth_month  gender  \
0   825           53              66        1966           11  Female
1  1746           53              68        1966           12  Female
2  1718           81              67        1938           11  Female
3   708           63              63        1957            1  Female
4  1164           43              70        1976            9    Male

                    address  latitude  longitude  per_capita_income  \
0             462 Rose Lane     34.15    -117.76           29278.00
1    3606 Federal Boulevard     40.76     -73.74           37891.00
2           766 Third Drive     34.02    -117.89           22681.00
3          3 Madison Street     40.71     -73.99          163145.00
4  9620 Valley Stream Drive     37.76    -122.44           53797.00

   yearly_income  total_debt  credit_score  num_credit_cards
0       59696.00   127613.00           787                 5
1       77254.00   191349.00           701                 5
2       33483.00      196.00           698                 5
3      249925.00   202328.00           722                 4
4      109687.00   183855.00           675                 1
```

Cleaning cards_data:

Convert credit_limit column to numeric after removing $.

[91]:
```
cards_df['credit_limit'] = clean_amount_column(cards_df, 'credit_limit')
```

Convert categorical values with numeric datatype to string

[93]:
```
cards_df['id'] = cards_df['id'].astype('str')
cards_df['client_id'] = cards_df['client_id'].astype('str')
cards_df['card_number'] = cards_df['card_number'].astype('str')
cards_df['cvv'] = cards_df['cvv'].astype('str')
```

Convert 'expires' and 'acct_open_date' columns to datetime

[95]:
```
cards_df['expires'] = pd.to_datetime(cards_df['expires'], format='%m/%Y',␣
↪errors='coerce')
cards_df['acct_open_date'] = pd.to_datetime(cards_df['acct_open_date'],␣
↪format='%m/%Y', errors='coerce')
```

[96]:
```
cards_df.dtypes
```

[96]:
```
id                                 object
client_id                          object
card_brand                         object
card_type                          object
card_number                        object
expires                    datetime64[ns]
cvv                                object
has_chip                           object
num_cards_issued                    int64
credit_limit                      float64
acct_open_date             datetime64[ns]
year_pin_last_changed               int64
card_on_dark_web                   object
dtype: object
```

[97]:
```
cards_df.head()
```

[97]:
```
     id client_id  card_brand        card_type       card_number    expires  \
0  4524       825        Visa            Debit  4344676511950444 2022-12-01
1  2731       825        Visa            Debit  4956965974959986 2020-12-01
2  3701       825        Visa            Debit  4582313478255491 2024-02-01
3    42       825        Visa           Credit  4879494103069057 2024-08-01
4  4659       825  Mastercard  Debit (Prepaid)  5722874738736011 2009-03-01

   cvv has_chip  num_cards_issued  credit_limit acct_open_date  \
0  623      YES                 2      24295.00     2002-09-01
1  393      YES                 2      21968.00     2014-04-01
2  719      YES                 2      46414.00     2003-07-01
3  693       NO                 1      12400.00     2003-01-01
```

```
4   75      YES                  1         28.00     2008-09-01

   year_pin_last_changed card_on_dark_web
0                   2008               No
1                   2014               No
2                   2004               No
3                   2012               No
4                   2009               No
```

Flag expired cards

[99]:
```
cards_df['is_expired'] = cards_df['expires'] < pd.Timestamp.today()
```

[100]:
```
print(f"{cards_df['is_expired'].sum()} out of {cards_df.shape[0]} cards are␣
↪inactive as of {pd.Timestamp.today().date()}")
```

```
6146 out of 6146 cards are inactive as of 2025-01-29
```

[101]:
```
del transaction_file_path
del users_file_path
del cards_file_path
del mcc_codes_file_path
del mcc_codes_df
```

1.5 Exploratory Data Analysis

1.5.1 Summary Statistics

[104]:
```
transactions_df.describe(include='float').T
```

[104]:
```
              count  mean   std     min  25%   50%   75%     max
amount 13305915.00 42.98 81.66 -500.00 8.93 28.99 63.71 6820.20
```

[105]:
```
transactions_df.describe(include='O').T
```

[105]:
```
                          count    unique                         top  \
id                     13305915  13305915                     7475327
client_id              13305915      1219                        1098
card_id                13305915      4071                        4938
use_chip               13305915         3           Swipe Transaction
merchant_id            13305915     74831                       59935
merchant_city          13305915     12492                      ONLINE
merchant_state         13305915       200                     Unknown
zip                    13305915     25257                     Unknown
errors                 13305915        23                    No Error
transaction_day_of_week 13305915         7                    Thursday
mcc_category           13305915       108  Grocery Stores, Supermarkets

                           freq
```

```
id                               1
client_id                    48479
card_id                      31552
use_chip                   6967185
merchant_id                 610053
merchant_city              1563700
merchant_state             1563700
zip                        1652706
errors                    13094522
transaction_day_of_week    1918666
mcc_category               1592584
```

[106]:
```python
users_df.describe(include=['float', 'int']).T
```

[106]:
```
                   count     mean      std      min      25%      50%  \
current_age      2000.00    45.39    18.41    18.00    30.00    44.00
retirement_age   2000.00    66.24     3.63    50.00    65.00    66.00
birth_year       2000.00  1973.80    18.42  1918.00  1961.00  1975.00
birth_month      2000.00     6.44     3.57     1.00     3.00     7.00
latitude         2000.00    37.39     5.11    20.88    33.84    38.25
longitude        2000.00   -91.55    16.28  -159.41   -97.39   -86.44
per_capita_income 2000.00 23141.93 11324.14     0.00 16824.50 20581.00
yearly_income    2000.00 45715.88 22992.62     1.00 32818.50 40744.50
total_debt       2000.00 63709.69 52254.45     0.00 23986.75 58251.00
credit_score     2000.00   709.73    67.22   480.00   681.00   711.50
num_credit_cards 2000.00     3.07     1.64     1.00     2.00     3.00

                       75%       max
current_age          58.00    101.00
retirement_age       68.00     79.00
birth_year         1989.00   2002.00
birth_month          10.00     12.00
latitude             41.20     61.20
longitude           -80.13    -68.67
per_capita_income 26286.00 163145.00
yearly_income     52698.50 307018.00
total_debt        89070.50 516263.00
credit_score        753.00    850.00
num_credit_cards      4.00      9.00
```

[107]:
```python
users_df.describe(include='O').T
```

[107]:
```
        count unique                  top  freq
id       2000   2000                  825     1
gender   2000      2               Female  1016
address  2000   1999  506 Washington Lane     2
```

[108]:
```
cards_df.describe(include=['float', 'int']).T
```

[108]:
```
                      count     mean      std     min      25%      50%  \
num_cards_issued     6146.00     1.50     0.52    1.00     1.00     1.00
credit_limit         6146.00 14347.49 12014.46    0.00  7042.75 12592.50
year_pin_last_changed 6146.00  2013.44     4.27 2002.00  2010.00  2013.00

                          75%       max
num_cards_issued         2.00      3.00
credit_limit         19156.50 151223.00
year_pin_last_changed  2017.00   2020.00
```

[109]:
```
cards_df.describe(include='O').T
```

[109]:
```
               count unique               top  freq
id              6146   6146              4524     1
client_id       6146   2000              1741     9
card_brand      6146      4        Mastercard  3209
card_type       6146      3             Debit  3511
card_number     6146   6146  4344676511950444     1
cvv             6146    998               877    15
has_chip        6146      2               YES  5500
card_on_dark_web 6146     1                No  6146
```

1.5.2 Some Insights from the data

Filter high-value transactions (amount > $1000)

[112]:
```
high_value_transactions = transactions_df[transactions_df["amount"] > 1000]
high_value_transactions
```

[112]:
```
                id                 date client_id card_id  amount  \
363        7475749  2010-01-01 06:51:00      1133    2586 1153.61
2642       7478491  2010-01-01 17:25:00       556       2 1309.71
4235       7480358  2010-01-02 09:32:00      1150    1225 1411.14
4559       7480766  2010-01-02 11:12:00      1498    2232 1037.26
4572       7480780  2010-01-02 11:17:00      1640    5171 1091.70
…              …                  …        …       …       …
13296575  23750302 2019-10-29 12:48:00      1616    4734 1248.56
13296656  23750400 2019-10-29 13:05:00      1913    5877 1262.58
13298485  23752703 2019-10-30 03:37:00       556    5654 1208.88
13301323  23756188 2019-10-30 18:52:00       134    6080 1152.88
13305601  23761488 2019-10-31 20:47:00       313    4804 1265.67

                   use_chip merchant_id merchant_city merchant_state    zip  \
363       Swipe Transaction       29742    Coraopolis             PA  15108
2642      Swipe Transaction       38489    Springboro             OH  45066
4235      Swipe Transaction       57386    Fort Worth             TX  76111
```

```
4559      Swipe Transaction       49814  Williamstown            NJ   8094
4572      Swipe Transaction       35503        Adrian            MI  49221
…                         …         …            …            …    …
13296575  Swipe Transaction       38489       Houston            TX  77035
13296656  Swipe Transaction        5594       Farwell            MI  48622
13298485   Chip Transaction       29742     Massillon            OH  44646
13301323   Chip Transaction       56903       El Paso            TX  79912
13305601   Chip Transaction       22792          Kyle            TX  78640

           mcc    errors  transaction_year  transaction_month  \
363       3256  No Error              2010                  1
2642      3058  No Error              2010                  1
4235      3132  No Error              2010                  1
4559      8111  No Error              2010                  1
4572      8111  No Error              2010                  1
…          …       …               …                 …
13296575  3058  No Error              2019                 10
13296656  3001  No Error              2019                 10
13298485  3256  No Error              2019                 10
13301323  8111  No Error              2019                 10
13305601  8062  No Error              2019                 10

          transaction_day  transaction_hour transaction_day_of_week  \
363                     1                 6                  Friday
2642                    1                17                  Friday
4235                    2                 9                Saturday
4559                    2                11                Saturday
4572                    2                11                Saturday
…                     …                …                     …
13296575               29                12                 Tuesday
13296656               29                13                 Tuesday
13298485               30                 3               Wednesday
13301323               30                18               Wednesday
13305601               31                20                Thursday

                                 mcc_category
363        Brick, Stone, and Related Materials
2642      Tools, Parts, Supplies Manufacturing
4235                             Leather Goods
4559             Legal Services and Attorneys
4572             Legal Services and Attorneys
…                                         …
13296575  Tools, Parts, Supplies Manufacturing
13296656          Steel Products Manufacturing
13298485   Brick, Stone, and Related Materials
13301323         Legal Services and Attorneys
13305601                             Hospitals
```

```
[9185 rows x 18 columns]
```

This subset identifies transactions that involve significant spending, useful for premium product offers or fraud detection.

Sort transactions by amount in descending order to find the top transactions

[115]:
```python
sorted_transactions = transactions_df.sort_values(by="amount", ascending=False)
sorted_transactions.head(10)
```

[115]:
```
                id                date  client_id  card_id  amount  \
892174     8544734 2010-09-22 06:37:00        708     5165  6820.20
12248570  22453398 2019-01-27 17:52:00       1081     3427  6613.44
2888921   10973185 2012-04-10 11:05:00       1259     5406  5913.37
4373878   12783563 2013-05-22 17:28:00       1487     4946  5813.78
6314617   15155601 2014-10-24 13:11:00        278     5619  5696.78
833431     8473892 2010-09-05 08:14:00       1699     2204  5694.44
11243848  21211758 2018-05-09 17:38:00       1156      175  5682.22
6388103   15245857 2014-11-13 10:27:00       1259     5406  5654.50
12192436  22383851 2019-01-13 07:09:00        708     5621  5591.73
10594708  20412330 2017-11-21 09:19:00        742     3943  5155.36

                    use_chip merchant_id  merchant_city merchant_state  \
892174     Swipe Transaction       34524  Staten Island             NY
12248570  Online Transaction        9026         ONLINE        Unknown
2888921    Swipe Transaction       85983         Wilton             CT
4373878   Online Transaction        9026         ONLINE        Unknown
6314617   Online Transaction        7202         ONLINE        Unknown
833431    Online Transaction        9026         ONLINE        Unknown
11243848  Online Transaction        9026         ONLINE        Unknown
6388103    Swipe Transaction       76639       Stamford             CT
12192436    Chip Transaction       84324       New York             NY
10594708  Online Transaction        7202         ONLINE        Unknown

              zip   mcc    errors  transaction_year  transaction_month  \
892174      10302  5712  No Error              2010                  9
12248570  Unknown  4411  No Error              2019                  1
2888921      6897  5932  No Error              2012                  4
4373878   Unknown  4411  No Error              2013                  5
6314617   Unknown  4411  No Error              2014                 10
833431    Unknown  4411  No Error              2010                  9
11243848  Unknown  4411  No Error              2018                  5
6388103      6907  5732  No Error              2014                 11
12192436    10069  5712  No Error              2019                  1
10594708  Unknown  4411  No Error              2017                 11

          transaction_day  transaction_hour transaction_day_of_week  \
```

```
892174              22          6           Wednesday
12248570            27         17              Sunday
2888921             10         11             Tuesday
4373878             22         17           Wednesday
6314617             24         13              Friday
833431               5          8              Sunday
11243848             9         17           Wednesday
6388103             13         10            Thursday
12192436            13          7              Sunday
10594708            21          9             Tuesday

                                           mcc_category
892174    Furniture, Home Furnishings, and Equipment Stores
12248570                                       Cruise Lines
2888921                                       Antique Shops
4373878                                        Cruise Lines
6314617                                        Cruise Lines
833431                                         Cruise Lines
11243848                                       Cruise Lines
6388103                                   Electronics Stores
12192436  Furniture, Home Furnishings, and Equipment Stores
10594708                                       Cruise Lines
```

This sorting highlights the highest-value transactions, which could reveal trends in luxury spending or high-value customers.

Group by category to calculate total revenue per category

[118]:
```
category_revenue = transactions_df.groupby("mcc_category")["amount"].sum().
↪sort_values(ascending=False).reset_index()
category_revenue
```

[118]:
```
                        mcc_category       amount
0                     Money Transfer  53158515.64
1         Grocery Stores, Supermarkets  40970754.15
2                    Wholesale Clubs  37697546.74
3           Drug Stores and Pharmacies  35113527.69
4                   Service Stations  29570426.66
..                               ...          ...
103          Household Appliance Stores    160285.62
104  Music Stores - Musical Instruments    148540.92
105                   Cosmetic Stores     76821.92
106              Sporting Goods Stores     47568.77
107          Gift, Card, Novelty Stores     25225.92

[108 rows x 2 columns]
```

This grouping reveals which categories contribute most to revenue, helping prioritize partnerships or promotions.

Filter customers with high income (yearly_income > $100,000)

```
[121]: high_income_users = users_df[users_df["yearly_income"] > 100000]
high_income_users
```

[121]:

	id	current_age	retirement_age	birth_year	birth_month	gender	\
3	708	63	63	1957	1	Female	
4	1164	43	70	1976	9	Male	
21	777	18	65	2002	1	Male	
58	1452	46	59	1973	5	Female	
84	1014	54	70	1965	9	Female	
85	290	27	66	1992	3	Female	
126	165	34	65	1986	2	Male	
142	1799	32	55	1987	4	Male	
167	1427	34	66	1985	10	Female	
215	1147	80	69	1939	3	Male	
216	1625	24	66	1995	5	Female	
240	1543	31	68	1988	10	Female	
249	995	40	64	1979	5	Female	
370	1897	38	67	1981	3	Male	
377	1201	60	66	1959	11	Male	
453	236	36	65	1983	3	Female	
481	1156	56	69	1963	6	Female	
496	599	26	71	1993	3	Male	
529	1223	53	67	1966	6	Male	
592	1865	19	66	2000	9	Male	
651	715	37	75	1983	1	Female	
654	842	61	65	1959	2	Male	
656	763	35	61	1985	1	Male	
658	704	51	67	1968	7	Female	
693	1259	64	69	1955	7	Female	
745	856	54	59	1965	7	Male	
822	115	61	69	1958	7	Male	
834	1426	22	66	1997	4	Female	
883	1600	62	66	1957	9	Female	
1000	453	26	67	1994	2	Female	
1001	341	50	66	1969	10	Male	
1010	440	43	50	1976	11	Female	
1033	1517	37	71	1982	3	Male	
1036	696	74	67	1945	7	Female	
1231	944	58	71	1961	10	Male	
1293	342	18	68	2001	6	Male	
1366	1079	65	60	1954	11	Female	
1484	700	21	68	1998	12	Female	
1537	1988	59	67	1960	8	Male	
1618	414	45	66	1975	1	Female	
1647	952	29	66	1990	5	Male	

1677	959	35	67	1984	11	Female
1683	1692	81	59	1938	8	Female
1757	278	59	66	1960	9	Male
1780	1648	66	69	1953	5	Female
1811	1325	23	66	1996	3	Female
1880	989	78	66	1941	9	Male
1882	1983	50	67	1969	3	Male
1888	1168	51	68	1968	10	Male
1924	1790	21	69	1998	3	Male
1952	1395	58	65	1961	9	Male
1965	628	57	66	1963	1	Male

	address	latitude	longitude	per_capita_income	\
3	3 Madison Street	40.71	-73.99	163145.00	
4	9620 Valley Stream Drive	37.76	-122.44	53797.00	
21	970 Essex Drive	37.37	-122.21	106305.00	
58	524 Ocean Drive	29.76	-95.38	95039.00	
84	393 Mountain View Lane	33.60	-117.82	96516.00	
85	293 Wessex Street	42.40	-83.60	49458.00	
126	95266 Bayview Drive	37.83	-122.22	52813.00	
142	3249 12th Drive	47.75	-122.04	51751.00	
167	326 Elm Lane	35.19	-80.83	49477.00	
215	614 Spruce Avenue	40.60	-74.76	46827.00	
216	625 Washington Lane	47.67	-122.18	49629.00	
240	5857 12th Avenue	34.14	-118.46	51976.00	
249	1752 Martin Luther King Avenue	35.46	-97.51	49868.00	
370	683 Washington Street	40.87	-73.40	51032.00	
377	175 Valley Drive	42.36	-71.36	56632.00	
453	4137 Bayview Drive	29.70	-95.46	79100.00	
481	9603 South Lane	40.74	-74.33	137428.00	
496	4872 Lexington Avenue	40.36	-75.09	49534.00	
529	822 Ocean Street	41.80	-87.92	92938.00	
592	616 Catherine Avenue	40.04	-75.42	58297.00	
651	94 Ocean Avenue	41.76	-88.15	49195.00	
654	945 Third Avenue	47.58	-122.03	50607.00	
656	583 Seventh Street	39.99	-75.27	50579.00	
658	6840 North Lane	41.20	-73.73	55274.00	
693	729 Littlewood Avenue	41.18	-73.42	94302.00	
745	275 Tenth Street	38.98	-77.12	56069.00	
822	386 11th Lane	40.93	-73.72	49546.00	
834	37 Norfolk Boulevard	40.33	-74.03	56252.00	
883	314 Fourth Street	47.67	-122.18	49629.00	
1000	682 Martin Luther King Avenue	32.79	-96.76	63159.00	
1001	517 Eighth Drive	40.87	-73.40	51032.00	
1010	689 Valley Stream Lane	42.67	-70.98	52066.00	
1033	7954 Wessex Boulevard	38.90	-77.26	60593.00	
1036	5064 North Lane	40.63	-73.72	45685.00	

1231	3817 Martin Luther King Avenue	33.77	-118.34	58517.00
1293	3033 Maple Lane	38.94	-77.19	62840.00
1366	422 Madison Lane	40.66	-73.63	48994.00
1484	649 Second Avenue	40.71	-73.99	91487.00
1537	763 Essex Avenue	25.77	-80.20	53676.00
1618	471 Eighth Lane	40.79	-74.47	55362.00
1647	7135 Ninth Lane	42.31	-71.16	62177.00
1677	5223 Lafayette Drive	40.22	-74.93	50208.00
1683	97339 Lake Avenue	41.03	-73.86	74205.00
1757	910 Eighth Drive	29.76	-95.38	95039.00
1780	211 Valley Street	40.76	-74.59	91180.00
1811	459 East Avenue	37.44	-122.20	150583.00
1880	6283 Rose Avenue	38.87	-77.40	46175.00
1882	655 George Boulevard	33.55	-117.78	69236.00
1888	207 Ocean View Street	40.67	-74.42	53790.00
1924	727 Valley Stream Boulevard	41.24	-73.31	55814.00
1952	2687 Burns Avenue	40.98	-74.11	75378.00
1965	4 George Lane	40.00	-75.26	52517.00

	yearly_income	total_debt	credit_score	num_credit_cards
3	249925.00	202328.00	722	4
4	109687.00	183855.00	675	1
21	216740.00	0.00	700	2
58	193773.00	241571.00	660	1
84	196784.00	437533.00	729	3
85	100837.00	61377.00	687	2
126	107683.00	225017.00	694	3
142	105515.00	192458.00	646	4
167	100880.00	210445.00	770	1
215	104692.00	6955.00	704	2
216	101191.00	290730.00	659	1
240	105963.00	106266.00	684	4
249	101679.00	307856.00	592	1
370	104049.00	247623.00	741	2
377	115465.00	195657.00	715	4
453	161276.00	317964.00	540	1
481	280199.00	91367.00	752	5
496	100991.00	0.00	722	3
529	189490.00	448929.00	717	3
592	118862.00	276156.00	782	1
651	100303.00	53919.00	821	2
654	103185.00	206422.00	564	1
656	103126.00	130160.00	642	1
658	112695.00	35135.00	840	6
693	192269.00	100192.00	700	6
745	114318.00	328089.00	748	3
822	101018.00	78115.00	748	6

```
834       114692.00   91575.00          805                 2
883       101193.00  124771.00          747                 3
1000      128775.00  232506.00          655                 1
1001      104045.00       0.00          734                 3
1010      106159.00  144773.00          703                 5
1033      123540.00  236393.00          764                 4
1036      110570.00    5700.00          766                 8
1231      119308.00   89328.00          789                 6
1293      128123.00  135808.00          699                 1
1366      103294.00   39076.00          831                 3
1484      186534.00  233746.00          590                 1
1537      109440.00  180865.00          737                 5
1618      112875.00   44432.00          709                 2
1647      126775.00   28869.00          685                 2
1677      102369.00  189558.00          738                 2
1683      162709.00    5642.00          542                 3
1757      193768.00  150896.00          686                 5
1780      185909.00  461854.00          621                 5
1811      307018.00  516263.00          745                 2
1880      113514.00   16524.00          727                 8
1882      141161.00       0.00          773                 3
1888      109673.00  242379.00          505                 1
1924      113797.00  169684.00          660                 1
1952      153691.00  197377.00          604                 2
1965      107075.00   75999.00          815                 3
```

High-income customers are potential candidates for premium products or exclusive services.

Sort by credit_score to find customers with the highest and lowest scores

[124]:
```
sorted_users = users_df.sort_values(by="credit_score", ascending=False)
sorted_users.head(10)  # Top credit scores
```

[124]:
```
        id  current_age  retirement_age  birth_year  birth_month  gender  \
490   1104           91              66        1928            3    Male
1729   492           41              70        1978            9    Male
1313   497           63              65        1956           11    Male
773   1221           64              69        1955           11  Female
1527  1090           59              66        1960           10  Female
1750   202           20              66        1999            7  Female
1671   464           36              64        1983           11    Male
30    1884           18              64        2001            5    Male
1678  1049           66              67        1953           11  Female
404    678           63              67        1957            2  Female

                          address  latitude  longitude  \
490         120 Lafayette Boulevard     26.23     -80.13
1729           2397 Madison Avenue     34.95     -82.12
```

```
1313                 8414 Tenth Drive     43.54     -96.73
773               6606 Jefferson Avenue   33.92    -117.24
1527                102 Burns Boulevard   40.68     -75.22
1750                    75 Third Avenue   39.33     -84.40
1671            2352 Bayview Boulevard    28.50     -81.37
30                   660 Seventh Drive    39.98     -82.98
1678                289 Ocean View Avenue 25.77     -80.20
404   861 Martin Luther King Boulevard    42.88     -78.85

      per_capita_income  yearly_income  total_debt  credit_score  \
490            18266.00       40141.00      805.00           850
1729           18857.00       38450.00    51430.00           850
1313           25692.00       52380.00     8764.00           850
773            15079.00       30747.00    51667.00           850
1527           21005.00       42825.00   105122.00           850
1750           31920.00       65082.00    88347.00           850
1671           33078.00       67444.00    93513.00           850
30             28092.00       57281.00    89114.00           850
1678           17893.00       36481.00    81550.00           850
404            14456.00       29477.00    56355.00           850

      num_credit_cards
490                  6
1729                 1
1313                 6
773                  4
1527                 5
1750                 1
1671                 1
30                   1
1678                 6
404                  4
```

[125]:
```python
sorted_users.tail(10)  # Lowest credit scores
```

[125]:
```
        id  current_age  retirement_age  birth_year  birth_month  gender  \
1905   897           30              64        1989            8  Female
399   1818           85              65        1934            6  Female
925   1436           21              55        1998            8  Female
1756  1733           47              66        1972            6  Female
1638    80           82              67        1937            8  Female
806    630           60              66        1959            7    Male
1510   559           56              59        1963            5    Male
908   1163           45              61        1975            1    Male
150   1987           63              62        1956            9    Male
1642  1801           18              64        2001            9  Female
```

```
                          address  latitude  longitude  per_capita_income  \
1905    847 Martin Luther King Lane     31.07     -83.19           13238.00
399                  23 11th Avenue     36.73     -76.04           29485.00
925                729 Wessex Avenue     41.52     -87.42           21992.00
1756                 6613 Hill Drive     36.73     -84.16           12814.00
1638  362 Martin Luther King Street     41.01     -84.47           15090.00
806                1583 Grant Avenue     30.33     -81.65           19382.00
1510         593 Valley Stream Drive     35.97     -97.02           17394.00
908                  2270 Sixth Lane     29.99     -95.26           32943.00
150                   786 12th Drive     42.13     -87.92           23098.00
1642           3371 Madison Boulevard     33.29    -117.30           15520.00

      yearly_income  total_debt  credit_score  num_credit_cards
1905       26996.00    33332.00           501                 2
399        41843.00     1741.00           500                 3
925        44842.00    54189.00           500                 4
1756       26130.00    54290.00           498                 1
1638       21815.00     1597.00           498                 5
806        39521.00    61861.00           491                 3
1510       35468.00    53185.00           490                 2
908        67170.00   114251.00           489                 3
150        33686.00    24997.00           488                 2
1642       31644.00    70064.00           480                 1
```

- Top credit score customers are low-risk and ideal for loan or credit card offers.
- Customers with low scores may require financial counseling or credit-building products.

Group by current_age to calculate average income and debt

[128]:
```
age_group_metrics = users_df.groupby("current_age")[["yearly_income",␣
↪"total_debt"]].mean()
age_group_metrics
```

[128]:
```
             yearly_income  total_debt
current_age
18                47490.47    70240.12
19                50036.68    80231.62
20                44429.22    76167.65
21                52598.53    83731.57
22                41921.56    70694.42
…                      …         …
93                10782.00      346.00
94                50433.50     1690.50
98                38087.50      421.00
99                51110.00     3781.00
101               15348.00     1396.00

[80 rows x 2 columns]
```

This reveals how income and debt levels vary by age, helping tailor financial products to different age groups.

Further we can also bin the ages as 18-28, 28-38, 38-48, so on to get a more consolidated insights.

Filter cards with high credit limits (credit_limit > $50,000)

[131]:
```
high_limit_cards = cards_df[cards_df["credit_limit"] > 50000]
high_limit_cards
```

[131]:
```
        id  client_id  card_brand card_type       card_number     expires  cvv  \
15     281        708        Visa    Credit  4017261190134817  2015-05-01  877
17    5621        708        Visa     Debit  4032240655674503  2022-06-01   53
18    5165        708        Visa     Debit  4935974646456357  2020-06-01  649
68     748        777        Visa     Debit  4832328468851061  2023-08-01  580
69     441        777  Mastercard     Debit  5278075482033392  2023-04-01  437
…      …          …         …         …               …          …   …
5504  5794        484  Mastercard     Debit  5814688468697564  2023-11-01  196
5572  1434       1325        Amex    Credit   337243144975638  2020-07-01   64
5780  2907       1983  Mastercard    Credit  5268892141706910  2023-03-01  905
6012  4089       1395  Mastercard     Debit  5088932138697648  2024-02-01  904
6080  2786       1616        Visa     Debit  4474440079220409  2022-10-01  166

     has_chip  num_cards_issued  credit_limit acct_open_date  \
15        YES                 2      98100.00     2011-01-01
17        YES                 1     132439.00     2010-11-01
18        YES                 1     125723.00     2009-10-01
68        YES                 1      68400.00     2020-01-01
69        YES                 2      77237.00     2020-01-01
…         …                 …            …             …
5504      YES                 2      61262.00     2003-12-01
5572      YES                 1      89900.00     2020-02-01
5780      YES                 1      51900.00     2006-05-01
6012      YES                 1      51412.00     1997-08-01
6080       NO                 1      56039.00     2018-04-01

      year_pin_last_changed card_on_dark_web  is_expired
15                     2011               No        True
17                     2011               No        True
18                     2010               No        True
68                     2020               No        True
69                     2020               No        True
…                      …                …         …
5504                   2016               No        True
5572                   2020               No        True
5780                   2009               No        True
6012                   2008               No        True
6080                   2018               No        True
```

```
[92 rows x 14 columns]
```

High-limit cards indicate high-value customers or potential credit risks.

Identify high-value transactions made by high-income customers

[134]:
```python
# Merge transactions and users data
transactions_users = transactions_df.merge(users_df.rename(columns={"id":␣
 ↪"client_id"}), on="client_id", how="left")
```

[135]:
```python
# Filter high-income customers and high-value transactions
high_value_high_income = transactions_users[(transactions_users["amount"] >␣
 ↪1000) & (transactions_users["yearly_income"] > 100000)]
high_value_high_income
```

[135]:
```
                id                date client_id card_id  amount  \
10463      7487841 2010-01-04 08:24:00       944    5550 1459.48
31988      7513627 2010-01-10 17:48:00      1147    2042 1205.14
32047      7513692 2010-01-10 18:22:00      1223    1042 1414.16
32750      7514512 2010-01-11 05:42:00       165    5146 1501.95
42241      7525882 2010-01-13 20:07:00       704    2168 1393.13
…              …                   …       …       …      …
13278343  23727786 2019-10-24 16:05:00      1223     384 1684.57
13279065  23728677 2019-10-24 21:03:00      1452    3801 1351.19
13285358  23736480 2019-10-26 13:13:00      1259    5406 1713.72
13286357  23737724 2019-10-26 17:44:00      1223     384 1418.08
13296007  23749600 2019-10-29 10:34:00      1452    3801 1800.78

                    use_chip merchant_id merchant_city merchant_state  \
10463     Online Transaction        6063        ONLINE        Unknown
31988      Swipe Transaction       75894      New York             NY
32047      Swipe Transaction       56271      Hinsdale             IL
32750      Swipe Transaction       60569   Morgan Hill             CA
42241      Swipe Transaction       95826       Memphis             TN
…                          …           …             …              …
13278343    Chip Transaction       60569     La Grange             IL
13279065    Chip Transaction       34524       Houston             TX
13285358  Online Transaction       72813        ONLINE        Unknown
13286357  Online Transaction       27350        ONLINE        Unknown
13296007    Chip Transaction       39695       Houston             TX

              zip   mcc    errors  transaction_year  transaction_month  \
10463     Unknown  4511  No Error              2010                  1
31988       10010  8062  No Error              2010                  1
32047       60521  6300  No Error              2010                  1
32750       95037  5300  No Error              2010                  1
42241       38135  3006  No Error              2010                  1
```

```
…             …    …        …              …                …
13278343    60525  5300  No Error             2019                  10
13279065    77059  5712  No Error             2019                  10
13285358  Unknown  6300  No Error             2019                  10
13286357  Unknown  6300  No Error             2019                  10
13296007    77056  6300  No Error             2019                  10

          transaction_day  transaction_hour transaction_day_of_week  \
10463                   4                 8                  Monday
31988                  10                17                  Sunday
32047                  10                18                  Sunday
32750                  11                 5                  Monday
42241                  13                20               Wednesday
…                     …                …                     …
13278343               24                16                Thursday
13279065               24                21                Thursday
13285358               26                13                Saturday
13286357               26                17                Saturday
13296007               29                10                 Tuesday

                                             mcc_category  current_age  \
10463                                            Airlines           58
31988                                           Hospitals           80
32047                          Insurance Sales, Underwriting           53
32750                                     Wholesale Clubs           34
42241               Miscellaneous Fabricated Metal Products           51
…                                                     …          …
13278343                                  Wholesale Clubs           53
13279065  Furniture, Home Furnishings, and Equipment Stores           46
13285358                    Insurance Sales, Underwriting           64
13286357                    Insurance Sales, Underwriting           53
13296007                    Insurance Sales, Underwriting           46

          retirement_age  birth_year  birth_month  gender  \
10463                 71        1961           10    Male
31988                 69        1939            3    Male
32047                 67        1966            6    Male
32750                 65        1986            2    Male
42241                 67        1968            7  Female
…                    …         …           …     …
13278343              67        1966            6    Male
13279065              59        1973            5  Female
13285358              69        1955            7  Female
13286357              67        1966            6    Male
13296007              59        1973            5  Female

                       address  latitude  longitude  \
```

```
10463      3817 Martin Luther King Avenue     33.77    -118.34
31988                   614 Spruce Avenue     40.60     -74.76
32047                    822 Ocean Street     41.80     -87.92
32750                  95266 Bayview Drive     37.83    -122.22
42241                     6840 North Lane     41.20     -73.73
…                                       …        …         …
13278343                 822 Ocean Street     41.80     -87.92
13279065                   524 Ocean Drive     29.76     -95.38
13285358             729 Littlewood Avenue     41.18     -73.42
13286357                 822 Ocean Street     41.80     -87.92
13296007                   524 Ocean Drive     29.76     -95.38

          per_capita_income  yearly_income  total_debt  credit_score  \
10463              58517.00      119308.00    89328.00           789
31988              46827.00      104692.00     6955.00           704
32047              92938.00      189490.00   448929.00           717
32750              52813.00      107683.00   225017.00           694
42241              55274.00      112695.00    35135.00           840
…                         …              …           …             …
13278343           92938.00      189490.00   448929.00           717
13279065           95039.00      193773.00   241571.00           660
13285358           94302.00      192269.00   100192.00           700
13286357           92938.00      189490.00   448929.00           717
13296007           95039.00      193773.00   241571.00           660

          num_credit_cards
10463                    6
31988                    2
32047                    3
32750                    3
42241                    6
…                        …
13278343                 3
13279065                 1
13285358                 6
13286357                 3
13296007                 1

[1554 rows x 31 columns]
```

Combines income and spending data to identify high-value, high-income customers for premium product targeting.

Identify the top 5 customers by transaction volume.

[138]:
```
# Group by customer and count transactions
customer_transaction_count = transactions_df.groupby("client_id")["id"].count().
↪sort_values(ascending=False)
```

[139]:
```python
# Top 5 customers
customer_transaction_count.head(5)
```

[139]:
```
client_id
1098    48479
909     43381
1963    42462
1776    41350
114     40286
Name: id, dtype: int64
```

This highlights the most active customers, enabling loyalty or engagement campaigns.

Calculate the total transaction value

[142]:
```python
total_revenue = transactions_df["amount"].sum()
print(f"Total Revenue: ${total_revenue:.2f}")
```

```
Total Revenue: $571835522.28
```

Calculate the average transaction amount

[144]:
```python
average_transaction = transactions_df["amount"].mean()
print(f"Average Transaction Amount: ${average_transaction:.2f}")
```

```
Average Transaction Amount: $42.98
```

Calculate the percentage of declined transactions

[146]:
```python
transactions_df["declined"] = transactions_df["errors"] != 'No Error'
declined_percentage = (transactions_df["declined"].mean()) * 100
print(f"Percentage of Declined Transactions: {declined_percentage:.2f}%")
```

```
Percentage of Declined Transactions: 1.59%
```

Calculate debt-to-income ratio

[148]:
```python
users_df["debt_to_income_ratio"] = users_df["total_debt"] /␣
 ↪users_df["yearly_income"]
average_dti = users_df["debt_to_income_ratio"].mean()
print(f"Average Debt-to-Income Ratio: {average_dti:.2f}")
```

```
Average Debt-to-Income Ratio: 1.38
```

Calculate the percentage of cards flagged on the dark web

[150]:
```python
dark_web_percentage = ((cards_df["card_on_dark_web"]=='Yes').mean()) * 100
print(f"Percentage of Cards Compromised on Dark Web: {dark_web_percentage:.
 ↪2f}%")
```

```
Percentage of Cards Compromised on Dark Web: 0.00%
```

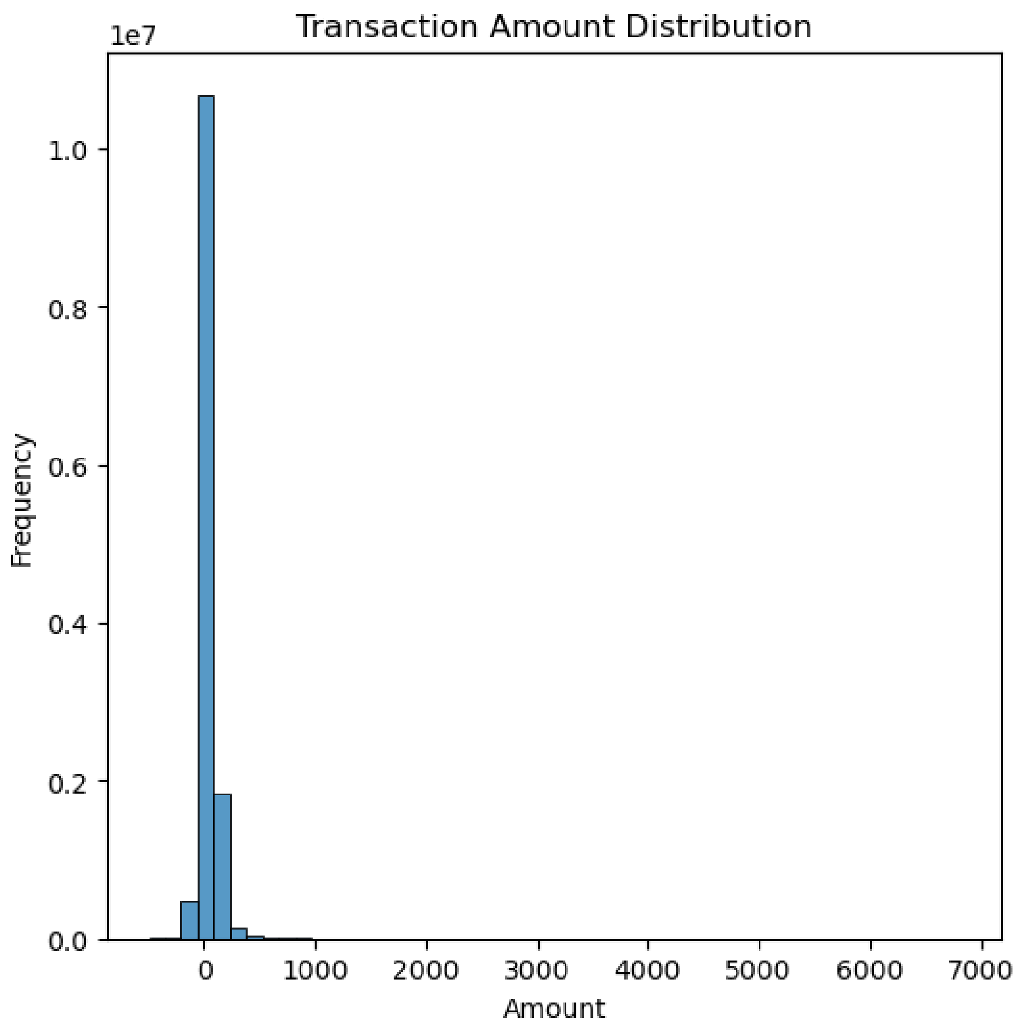
Deleting few values for memory purpose

del age_group_metrics del average_dti del average_transaction del category_revenue del customer_transaction_count del dark_web_percentage del declined_percentage del high_income_users del high_limit_cards del high_value_high_income del sorted_transactions del sorted_users del total_revenue del transactions_users

Data Visualization

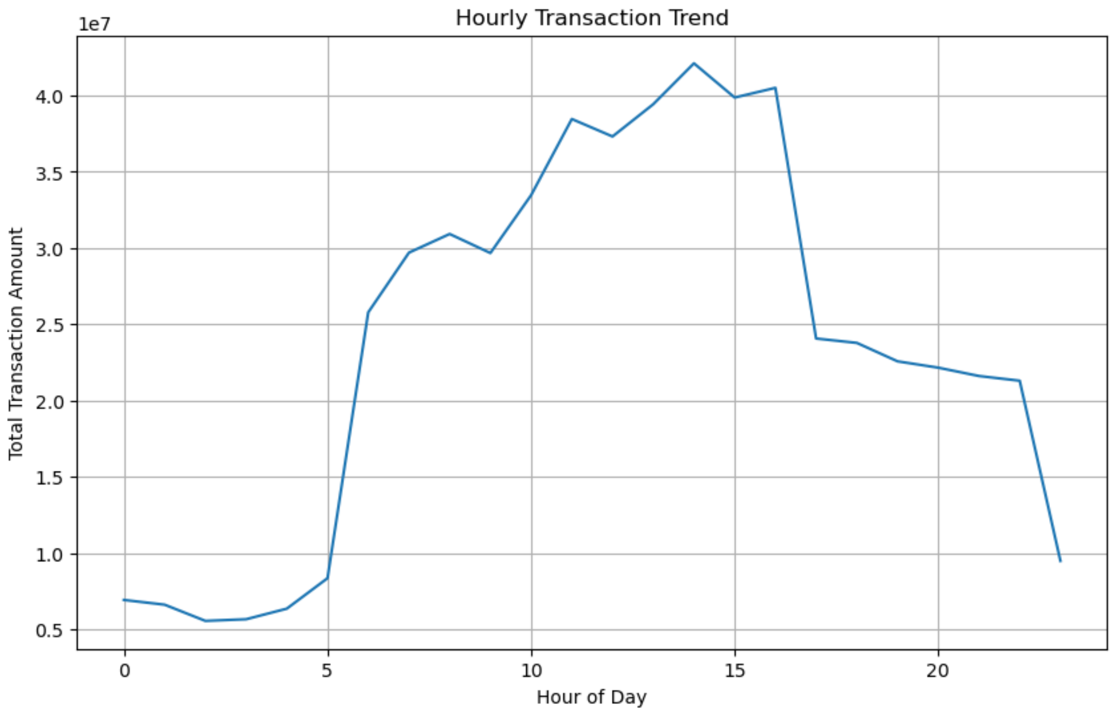
Transaction amount distribution

[155]:
```
plt.figure(figsize=(6, 6))
sns.histplot(transactions_df['amount'], bins=50)
plt.title("Transaction Amount Distribution")
plt.xlabel("Amount")
plt.ylabel("Frequency")
plt.show()
```



Analyze transactions over time (hourly trend)

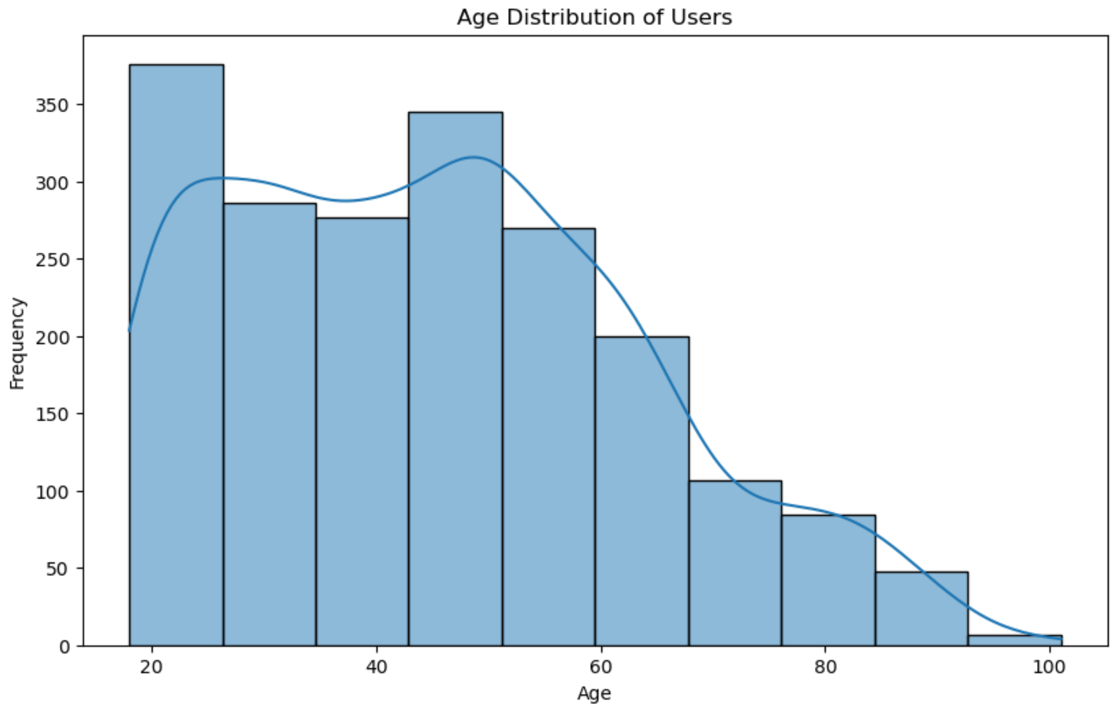
[157]:
```
hourly_transactions = transactions_df.groupby('transaction_hour')['amount'].
  ↪sum()
hourly_transactions.plot(kind='line', figsize=(10, 6), title="Hourly␣
  ↪Transaction Trend")
plt.xlabel("Hour of Day")
plt.ylabel("Total Transaction Amount")
plt.grid()
plt.show()
```



Insights: - Transaction amounts have a skewed distribution with a high frequency of small values. - Most transactions occur during 10 am to 4 pm, suggesting peaks in customer activity.

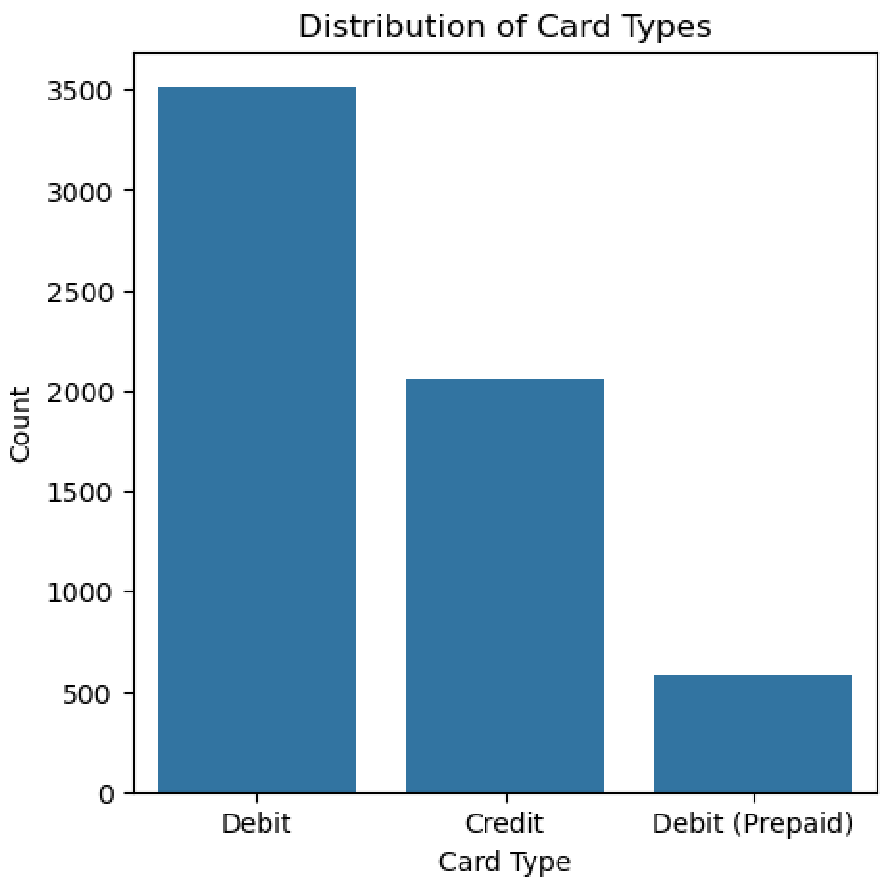
age distribution

[160]:
```
plt.figure(figsize=(10, 6))
sns.histplot(users_df['current_age'], kde=True, bins=10)
plt.title("Age Distribution of Users")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



card type distribution

[162]:
```
plt.figure(figsize=(5, 5))
sns.countplot(x=cards_df['card_type'])
plt.title("Distribution of Card Types")
plt.xlabel("Card Type")
plt.ylabel("Count")
plt.show()
```



Merge transactions_data with users_data to analyze customer demographics

[164]:
```
users_transactions_data = transactions_df.merge(
    users_df.rename(columns={"id": "client_id"}), on="client_id", how="left"
)
```

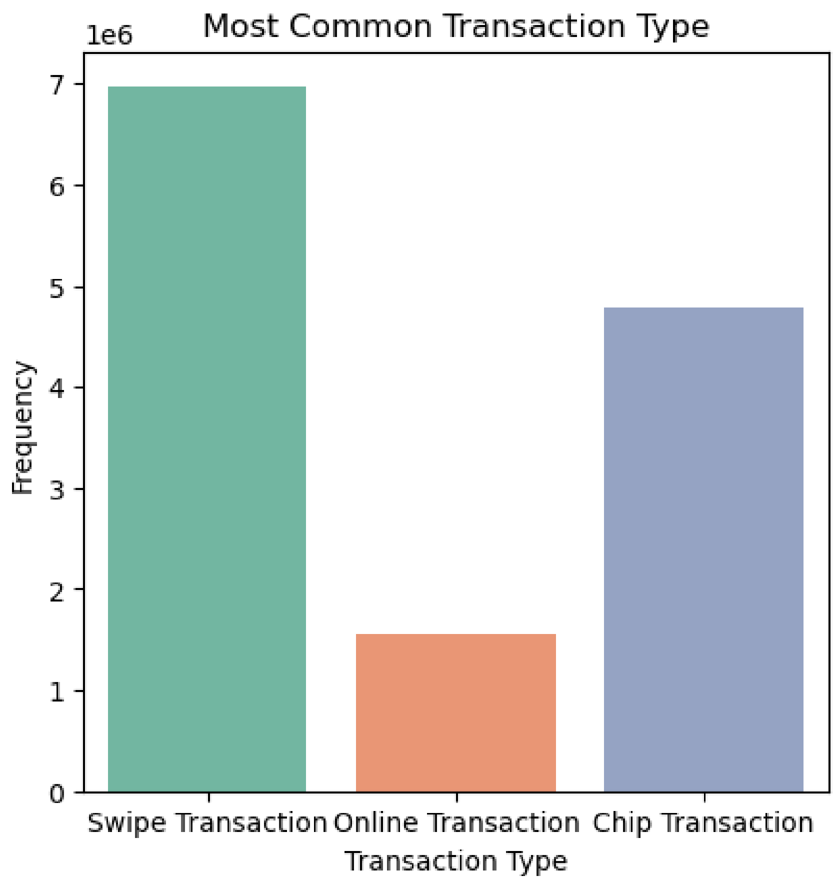
What is the most common transaction type?

[166]:
```
common_transaction_type = users_transactions_data["use_chip"].value_counts()
print(f"Most Common Transaction Type:\n{common_transaction_type}\n")

# Transaction type distribution
plt.figure(figsize=(5, 5))
sns.countplot(data=users_transactions_data, x="use_chip", palette="Set2")
plt.title("Most Common Transaction Type")
plt.xlabel("Transaction Type")
plt.ylabel("Frequency")
plt.show()
```

```
Most Common Transaction Type:
use_chip
```

```
Swipe Transaction     6967185
Chip Transaction      4780818
Online Transaction    1557912
Name: count, dtype: int64
```



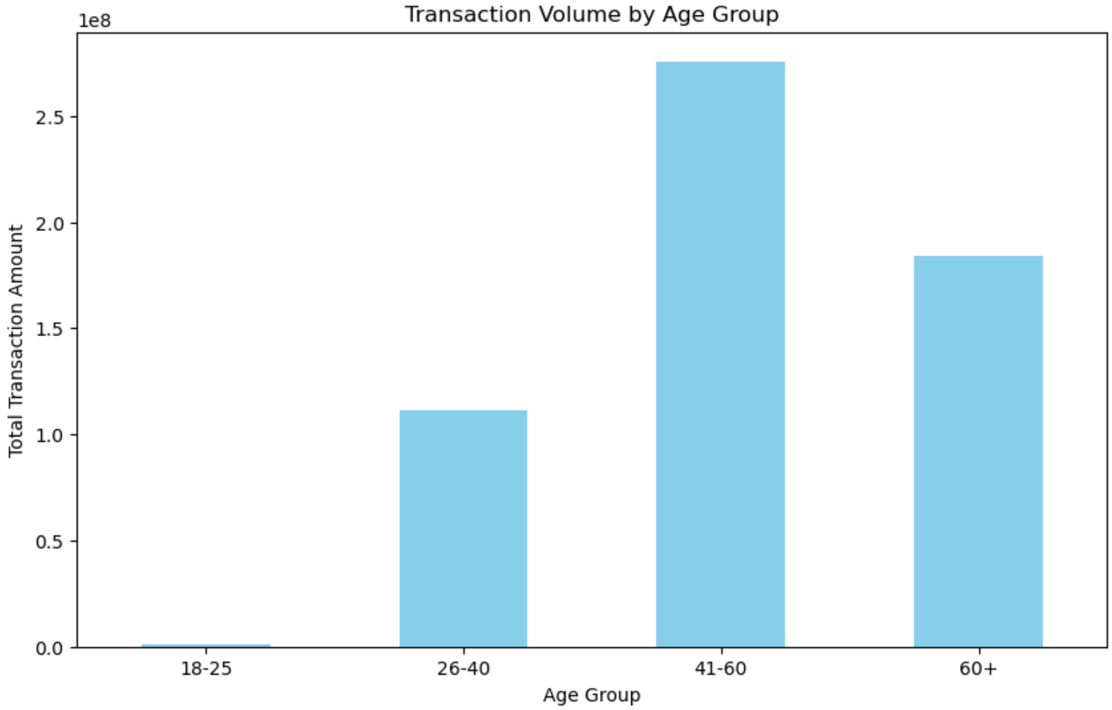
Majority of transactions are either "Swipe" or "Chip" based, showing customer preference.

Which customer segments (age groups, demographics) have the highest transaction volume

[169]:
```
# Create age groups
bins = [0, 25, 40, 60, 100]
labels = ["18-25", "26-40", "41-60", "60+"]
users_transactions_data["age_group"] = pd.
  ↪cut(users_transactions_data["current_age"], bins=bins, labels=labels,␣
  ↪right=False)

age_group_volume = users_transactions_data.groupby("age_group")["amount"].sum()
```

```
# Age group transaction volume
plt.figure(figsize=(10, 6))
age_group_volume.plot(kind="bar", color="skyblue")
plt.title("Transaction Volume by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Total Transaction Amount")
plt.xticks(rotation=0)
plt.show()
```

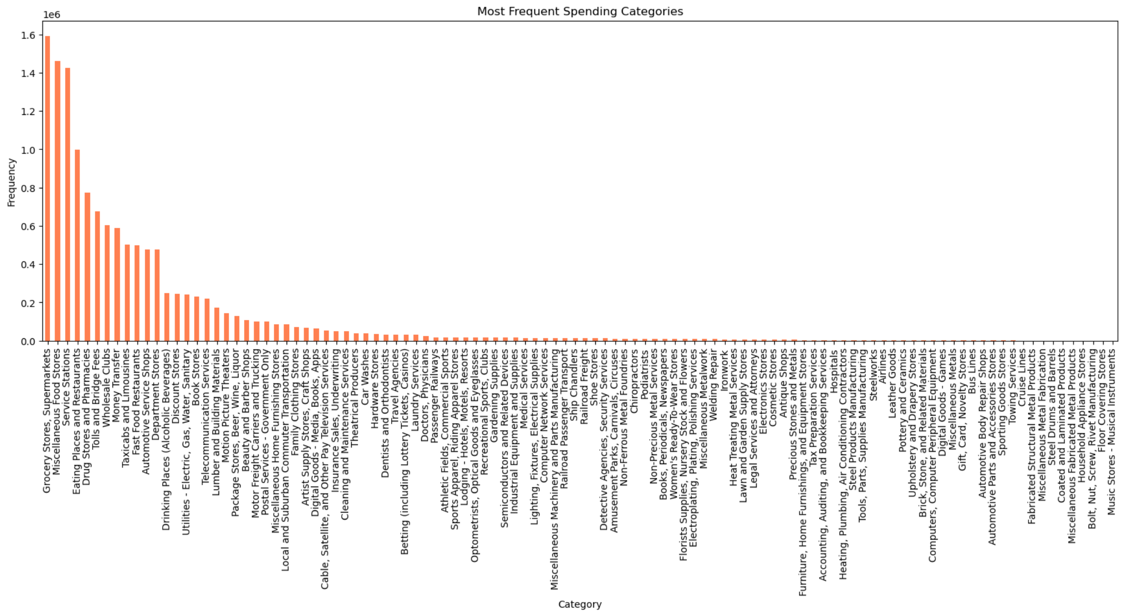


Age group between 41-60 has the highest transaction volumes and represent more active spending customers.

What are the most frequent spending categories?

[172]:
```
category_counts = users_transactions_data["mcc_category"].value_counts()

# Spending categories
plt.figure(figsize=(20, 6))
category_counts.plot(kind="bar", color="coral")
plt.title("Most Frequent Spending Categories")
plt.xlabel("Category")
plt.ylabel("Frequency")
plt.xticks(rotation=90)
plt.show()
```



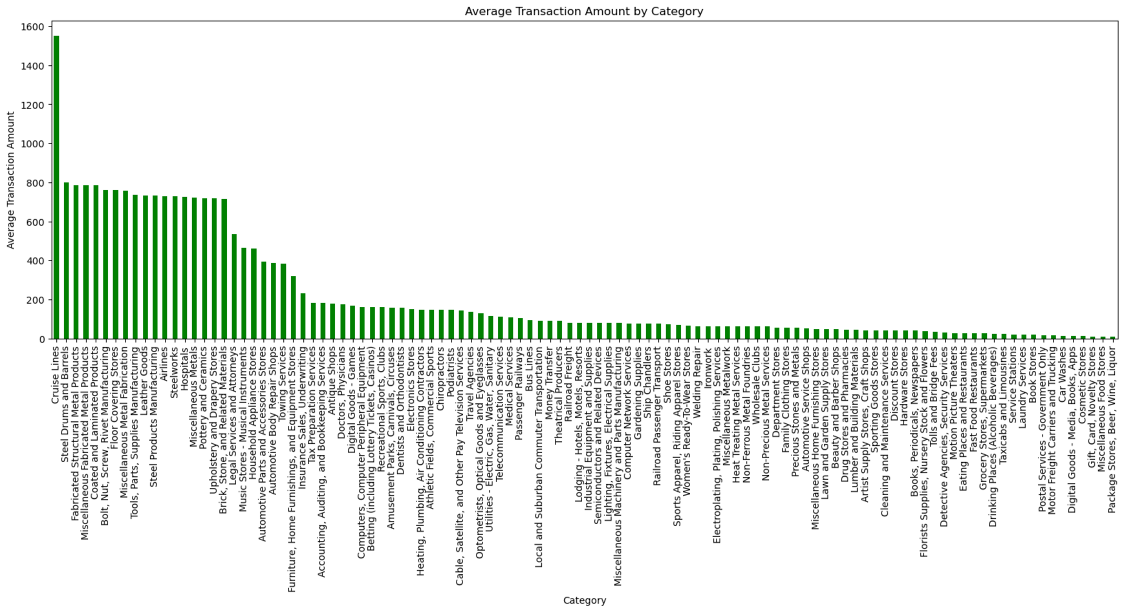
Common spending categories help businesses target popular sectors for promotions or partnerships.

Grocery Stores, Food Stores and Service Stations transactions are dominating the MCC categories.

What is the average transaction amount for each category?

[175]:
```python
avg_transaction_per_category = users_transactions_data.
 ↪groupby("mcc_category")["amount"].mean()

# Average transaction amount by category
plt.figure(figsize=(20, 6))
avg_transaction_per_category.sort_values(ascending=False).plot(kind="bar",␣
 ↪color="green")
plt.title("Average Transaction Amount by Category")
plt.xlabel("Category")
plt.ylabel("Average Transaction Amount")
plt.xticks(rotation=90)
plt.show()
```



Categories with higher average transaction amounts indicate opportunities for premium products or services.

Cruise Lines transactions have higher average amounts.

Recommendations:

- Tailor marketing efforts to customers aged 26-40 for higher transaction volumes.
- Partner with restaurants and food stores to offer loyalty programs for premium services.
- Focus on improving digital transaction experiences, as chip-based transactions remain popular.

What is the total transaction value over the years?

[179]:
```
total_value_yearly = users_transactions_data.
 ↪groupby("transaction_year")["amount"].sum()

# Total Transaction Value by Year
fig = go.Figure(data=[go.Pie(
    labels=total_value_yearly.index,
    values=total_value_yearly.values,
    hole=.5
)])
fig.update_layout(title_text="Total Transaction Value by Year")
fig.show()
```

The donut chart provides a clear proportional view of yearly revenue distribution. Yearly transaction trends indicate the overall growth or decline in total revenue. As we see an equal distribution over the years, there is a dip in 2019

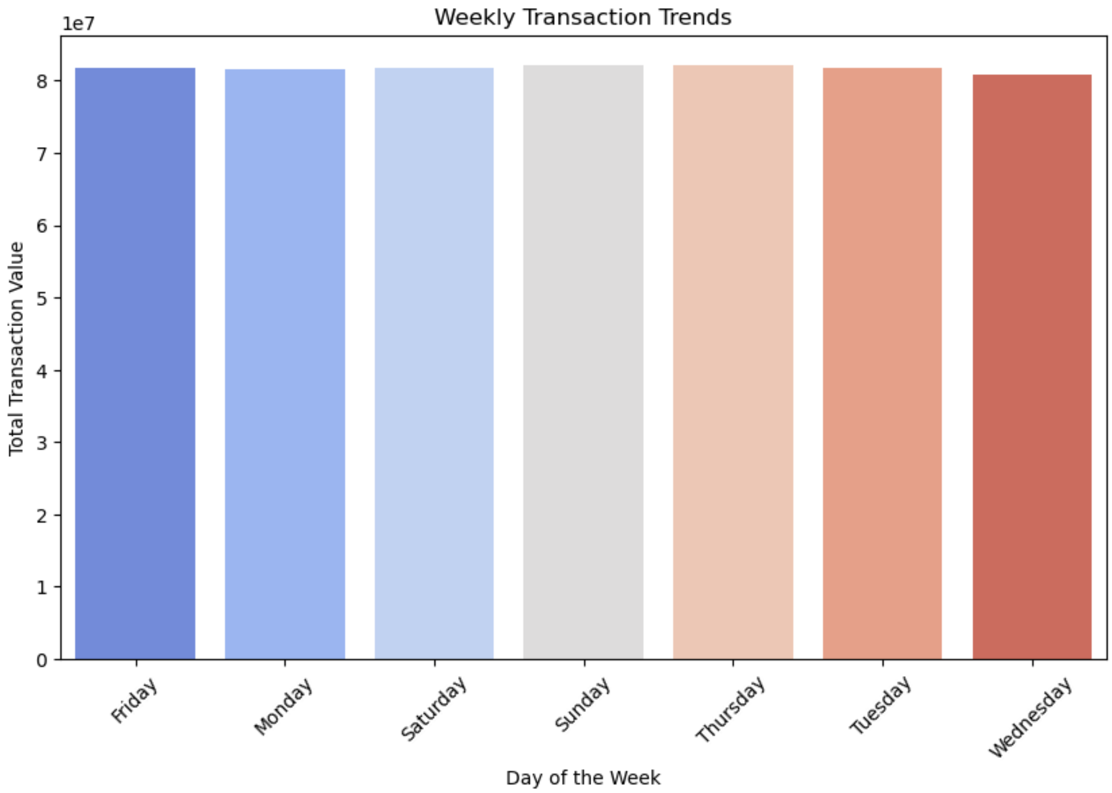
Are there any seasonal trends in transactions?

[182]:
```
# Monthly transaction value
monthly_revenue = users_transactions_data.
 ↪groupby("transaction_month")["amount"].sum().reset_index()

# Monthly transaction trends
fig = px.line(
    monthly_revenue,
    x="transaction_month",
    y="amount",
    title="Monthly Transaction Trends",
    labels={"month": "Month", "amount": "Total Transaction Value"},
    template="plotly_white",
    markers=True
)
fig.show()
```

[183]:
```
# Weekly transaction trends
weekly_revenue = users_transactions_data.
 ↪groupby("transaction_day_of_week")["amount"].sum()

# Weekly transaction trends
plt.figure(figsize=(10, 6))
sns.barplot(x=weekly_revenue.index, y=weekly_revenue.values, palette="coolwarm")
plt.title("Weekly Transaction Trends")
plt.xlabel("Day of the Week")
plt.ylabel("Total Transaction Value")
plt.xticks(rotation=45)
plt.show()
```



We observe a low activity during the months of February, November and December and peak activity during July, August and October.

Businesses must target these months for marketing campaigns or operational adjustments.

We also observe the day of the week not affecting the transaction activity.

Which customers or customer segments contribute the most to the bank's revenue?

[186]:
```
# Calculate total revenue per customer
customer_revenue = users_transactions_data.groupby("client_id")["amount"].sum().
↪sort_values(ascending=False)

# Top customers by revenue (Bar Chart using Plotly)
top_customers = customer_revenue.head(10).reset_index()
fig = px.bar(
    top_customers,
    x="client_id",
    y="amount",
    text="amount",
    title="Top 10 Customers by Revenue Contribution",
    labels={"client_id": "Customer ID", "amount": "Total Revenue"},
    template="plotly_white"
```

```
)
fig.update_traces(texttemplate='%{text:.2f}', textposition="outside")
fig.show()
```

A small number of customers contribute disproportionately to the revenue.

These high-value customers should be targeted with loyalty programs.

Identify high-value customers for retention strategies and targeted offers.

del age_group_volume del avg_transaction_per_category del bins del category_counts del common_transaction_type del customer_revenue del high_value_transactions del hourly_transactions del labels del monthly_revenue del top_customers del total_value_yearly del weekly_revenue

Which regions experience the most transaction failures?

[190]:
```
# Transaction failure flag
transactions_df["transaction_failure"] = transactions_df["errors"]!="No Error"

# Count transaction failures by region
failure_by_region = transactions_df.
↪groupby(["merchant_state"])["transaction_failure"].sum().sort_values()
```

[191]:
```
# For ease of visualization, filtering only transactions with atleast 1 failure
failure_by_region = failure_by_region[failure_by_region > 0]

# Transaction failures by region
fig = px.bar(
    failure_by_region,
    title="Transaction Failures by Region",
    labels={"value": "Number of Failures", "index": "Region"},
    text=failure_by_region.values,
    orientation='h',
    height=1500,
)
fig.update_traces(texttemplate='%{text}', textposition="outside")
fig.show()
```

Customers or regions with frequent failures may need targeted support to improve operational efficiency.

Failure rates vary by region. Los Angeles (CA) shows higher failure rates, possibly due to infrastructure or service issues.

Recommendations: - Provide targeted support to customers and regions with frequent failures to improve transaction success rates. - Investigate infrastructure issues or service reliability in regions with high failure rates.

Which customer demographics should the bank target for promotions or new products?

[195]:
```
# Group by demographic attributes and calculate total revenue
demographic_revenue = users_transactions_data.groupby(["gender",␣
↪"age_group"])["amount"].sum().reset_index()

# Revenue by demographics
fig = px.bar(
    demographic_revenue,
    x="age_group",
    y="amount",
    color="gender",
    barmode="group",
    title="Revenue by Customer Demographics",
    labels={"amount": "Total Revenue", "age_group": "Age Group", "gender":␣
↪"Gender"},
    text="amount",
    template="plotly_white"
)
fig.update_traces(texttemplate='%{text:.2f}', textposition="outside")
fig.show()
```

How loyal are customers to specific transaction categories over time?

[197]:
```
# Group transactions by client_id and category to calculate total spending
customer_category_loyalty = users_transactions_data.groupby(["client_id",␣
↪"mcc_category"])["amount"].sum().reset_index().sort_values('amount',␣
↪ascending=False)

# Calculate the percentage of spending per category for each customer
customer_category_loyalty["percent_spending"] = customer_category_loyalty.
↪groupby("client_id")["amount"].transform(lambda x: x / x.sum() * 100)

# Loyalty to categories (example for one customer)
loyalty_sample =␣
↪customer_category_loyalty[customer_category_loyalty["client_id"] == '1556']

fig = px.bar(
    loyalty_sample,
    x="mcc_category",
    y="percent_spending",
    title="Customer Loyalty to Categories (Client ID: 1556)",
    labels={"percent_spending": "Percent Spending", "category": "Category"},
    text="percent_spending",
    template="plotly_white",
    width=2500,
    height=500
)
fig.update_traces(texttemplate='%{text:.2f}%', textposition="outside")
```

```
fig.show()
```

Customers with higher spending proportions in specific categories are more loyal, allowing targeted campaigns for those categories.

What is the average transaction count per customer annually?

[200]:
```
# Extract year from transaction dates
users_transactions_data["transaction_year"] = users_transactions_data["date"].
    ↪dt.year

# Group by customer and year to calculate transaction counts
customer_transaction_counts = users_transactions_data.groupby(["client_id",␣
    ↪"transaction_year"])["id"].count().reset_index()
customer_transaction_counts.rename(columns={"id": "transaction_count"},␣
    ↪inplace=True)

# Calculate the average transaction count per customer annually
avg_transaction_count = customer_transaction_counts.
    ↪groupby("transaction_year")["transaction_count"].mean()

# Average transaction count per year
fig = px.line(
    avg_transaction_count,
    x=avg_transaction_count.index,
    y=avg_transaction_count.values,
    title="Average Transaction Count Per Customer Annually",
    labels={"x": "Year", "y": "Average Transaction Count"},
    markers=True,
    template="plotly_white"
)
fig.show()
```

We observe the transaction frequency increase over the years except during 2019 where we observe a significant dip

Customer retention trends

[203]:
```
customer_retention = users_transactions_data.groupby(["client_id",␣
    ↪"transaction_year"])["id"].count().reset_index()
customer_retention["retention_flag"] = customer_retention.
    ↪groupby("client_id")["transaction_year"].diff().fillna(1).astype(bool)

# Retention summary
retention_rate = customer_retention["retention_flag"].mean() * 100
print(f"Customer Retention Rate: {retention_rate:.2f}%")

# Top loyal customers
```

```
top_loyal_customers = customer_category_loyalty.
 ↪groupby("client_id")["percent_spending"].max().sort_values(ascending=False).
 ↪head(10)

# Top loyal customers
fig = px.bar(
    top_loyal_customers,
    x=top_loyal_customers.index,
    y=top_loyal_customers.values,
    title="Top Loyal Customers",
    labels={"x": "Client ID", "y": "Max Percent Spending"},
    text=top_loyal_customers.values,
    template="plotly_white"
)
fig.update_traces(texttemplate='%{text:.2f}%', textposition="outside")
fig.show()
```

```
Customer Retention Rate: 100.00%
```

Recommendations:

Loyal customers with the highest spending proportions in specific categories can be targeted for rewards or exclusive offers.

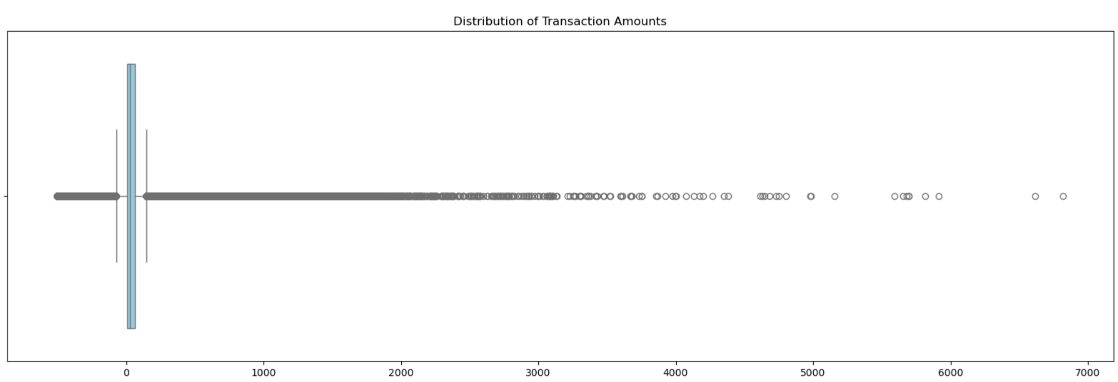
The bank's retention rate is approximately X%, indicating a strong customer base.

Strategies: Focus on retaining high-value customers through loyalty programs.

The top loyal customers are identified by their highest spending proportions in certain categories. These customers should receive targeted engagement to ensure retention and growth.

Distribution of transaction amounts

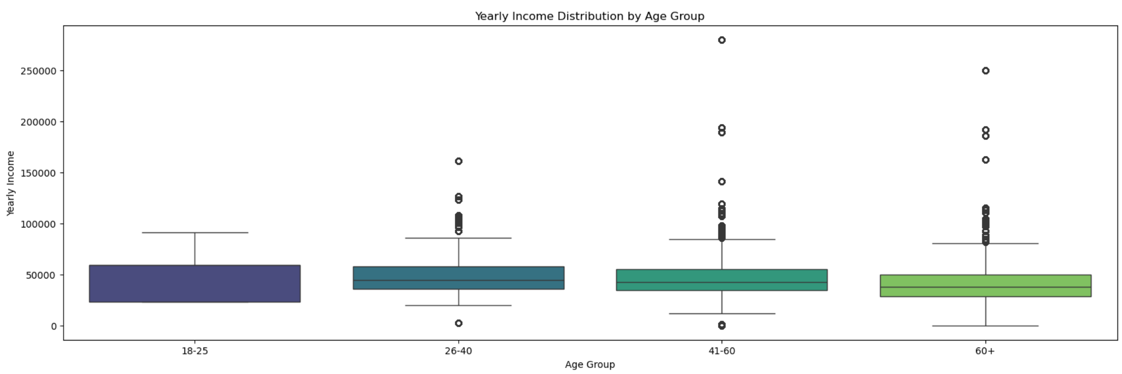
[206]:
```
plt.figure(figsize=(20, 6))
sns.boxplot(data=users_transactions_data, x="amount", color="skyblue")
plt.title("Distribution of Transaction Amounts")
plt.xlabel("Transaction Amount")
plt.show()
```



Distribution of user income by age group

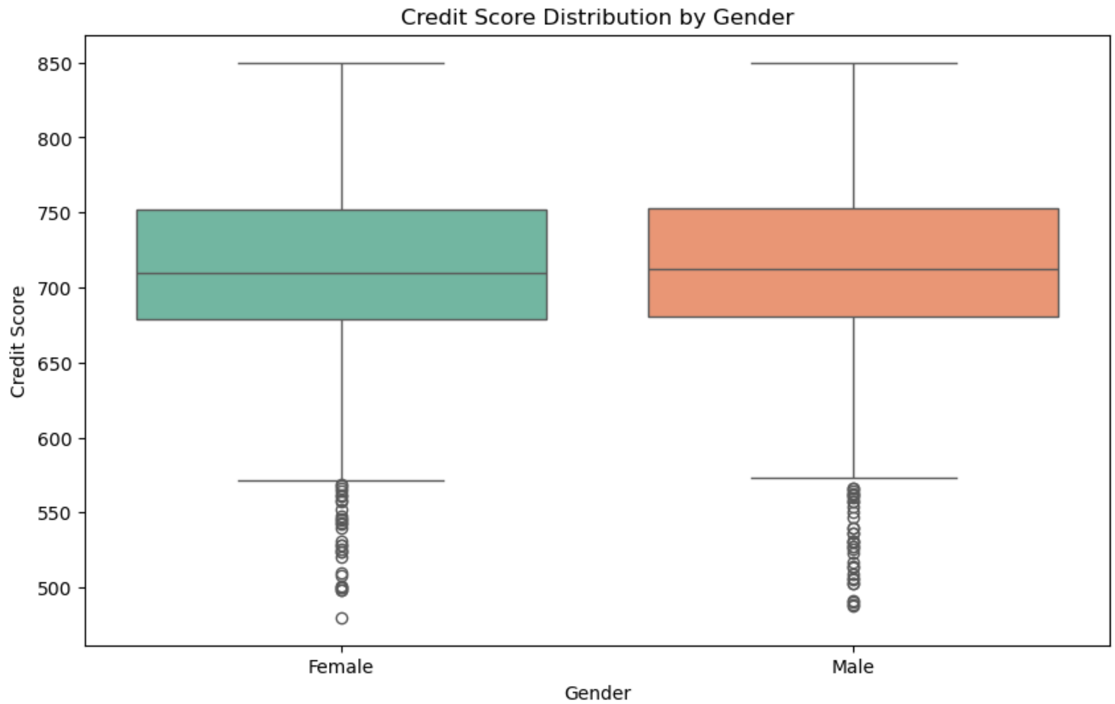
[208]:
```
plt.figure(figsize=(20, 6))
sns.boxplot(data=users_transactions_data, x="age_group", y="yearly_income",␣
↪palette="viridis")
plt.title("Yearly Income Distribution by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Yearly Income")
plt.show()

# Insights:
# - Boxplots show the spread of income across different age groups, helping␣
↪identify high-income segments for targeted marketing.
# - Outliers may represent high-net-worth individuals who warrant exclusive␣
↪products or services.
```



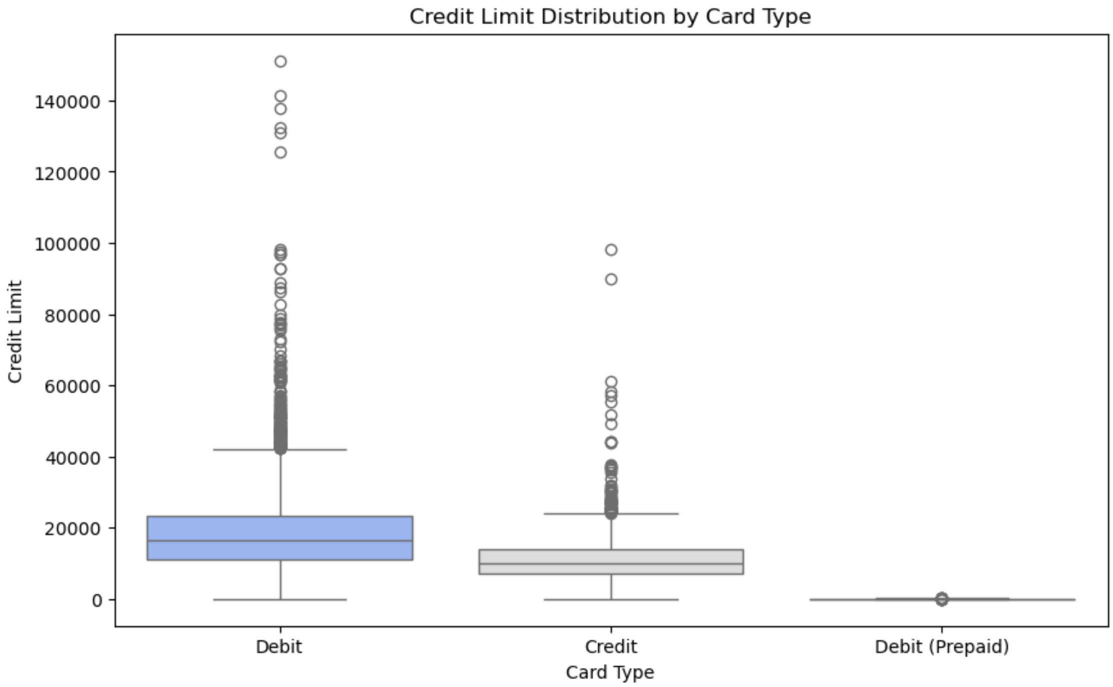
Credit score distribution by gender

[210]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=users_df, x="gender", y="credit_score", palette="Set2")
plt.title("Credit Score Distribution by Gender")
plt.xlabel("Gender")
plt.ylabel("Credit Score")
plt.show()
```



Credit limit distribution by card type

[212]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=cards_df, x="card_type", y="credit_limit", palette="coolwarm")
plt.title("Credit Limit Distribution by Card Type")
plt.xlabel("Card Type")
plt.ylabel("Credit Limit")
plt.show()
```



del avg_transaction_count del customer_category_loyalty del customer_retention del customer_transaction_counts del demographic_revenue del failure_by_region del loyalty_sample del retention_rate del top_loyal_customers

1.6 SQL Integration

1.6.1 1. Create an SQLite database

[216]:
```
engine = create_engine("sqlite:///financial_transactions_data.db")
```

1.6.2 2. Store data in the database

[ ]:
```
transactions_df.to_sql("transactions", engine, if_exists="replace", index=False)
users_df.to_sql("users", engine, if_exists="replace", index=False)
cards_df.to_sql("cards", engine, if_exists="replace", index=False)
mcc_codes_df.to_sql("mcc_codes", engine, if_exists="replace", index=False)

print("Data successfully stored in the database.")
```

1.6.3 3. Query the Database for Insights

Connect to the database

[221]:
```
connection = engine.connect()
```

What is the most common transaction type?

[223]:

```
query = """
SELECT use_chip, COUNT(*) AS count
FROM transactions
GROUP BY use_chip
ORDER BY count DESC
LIMIT 1;
"""
result = connection.execute(text(query)).fetchall()
print("Most Common Transaction Type:", result)
```

```
Most Common Transaction Type: [('Swipe Transaction', 6967185)]
```

Which customer segments (age groups) have the highest transaction volume?

[225]:

```
query = """
SELECT u.current_age, COUNT(t.id) AS transaction_count
FROM transactions t
JOIN users u ON t.client_id = u.id
GROUP BY u.current_age
ORDER BY transaction_count DESC;
"""
result = connection.execute(text(query)).fetchall()
print("Transaction Volume by Age Group:", result)
```

```
Transaction Volume by Age Group: [(47, 541132), (51, 409971), (52, 389245), (49,
372906), (50, 350911), (43, 350010), (48, 348784), (46, 342266), (59, 328724),
(58, 324342), (54, 323299), (42, 317454), (61, 316061), (44, 307188), (38,
304951), (63, 304782), (36, 292683), (56, 287372), (39, 277863), (57, 274192),
(41, 273420), (40, 266052), (53, 246499), (67, 239473), (31, 238640), (62,
237557), (32, 228989), (55, 227427), (37, 219987), (34, 216297), (68, 203976),
(66, 202268), (35, 199390), (45, 192713), (64, 180845), (65, 153775), (81,
152705), (76, 152140), (70, 148547), (75, 141483), (83, 139178), (78, 133658),
(33, 132710), (60, 131961), (82, 130233), (30, 128867), (69, 118100), (80,
107700), (77, 101150), (28, 94909), (85, 93684), (86, 89423), (84, 86373), (74,
83023), (79, 79674), (29, 77690), (73, 77567), (98, 62210), (72, 53666), (26,
53488), (90, 52013), (94, 46394), (87, 45972), (91, 41896), (71, 40597), (92,
39535), (88, 39268), (89, 35806), (27, 28932), (25, 28564), (101, 15412), (99,
13722), (24, 11891), (23, 4330)]
```

What are the top 5 merchant categories contributing to revenue?

[227]:

```
query = """
SELECT mcc_category, SUM(amount) AS total_revenue
FROM transactions
GROUP BY mcc_category
ORDER BY total_revenue DESC
LIMIT 5;
"""
result = connection.execute(text(query)).fetchall()
```

```
print("Top 5 Merchant Categories by Revenue:", result)
```

```
Top 5 Merchant Categories by Revenue: [('Money Transfer', 53158515.64),
('Grocery Stores, Supermarkets', 40970754.15), ('Wholesale Clubs', 37697546.74),
('Drug Stores and Pharmacies', 35113527.69), ('Service Stations', 29570426.66)]
```

What is the total transaction value over the years?

[229]:
```
query = """
SELECT strftime('%Y', date) AS year, SUM(amount) AS total_revenue
FROM transactions
GROUP BY transaction_year
ORDER BY transaction_year;
"""
result = connection.execute(text(query)).fetchall()
print("Total Transaction Value Over the Years:", result)
```

```
Total Transaction Value Over the Years: [('2010', 54232556.12), ('2011',
55778904.96), ('2012', 56832410.86), ('2013', 58284939.62), ('2014',
58617820.51), ('2015', 59514007.43), ('2016', 59844028.9), ('2017',
59628480.63), ('2018', 59627317.94), ('2019', 49475055.31)]
```

Which card_id has the latest expiry date?

[231]:
```
query = """
SELECT id, MAX(expires) AS latest_expiry
FROM cards;
"""
result = connection.execute(text(query)).fetchall()
print("Card with the Latest Expiry Date:", result)
```

```
Card with the Latest Expiry Date: [('5924', '2024-12-01 00:00:00.000000')]
```

Which customers have the highest and lowest credit limits?

[233]:
```
query = """
SELECT c.client_id, c.credit_limit
FROM cards c
ORDER BY c.credit_limit DESC
LIMIT 1;
"""
result_high = connection.execute(text(query)).fetchall()
print("Customer with the Highest Credit Limit:", result_high)
```

```
Customer with the Highest Credit Limit: [('1156', 151223.0)]
```

[234]:
```
query = """
SELECT c.client_id, c.credit_limit
FROM cards c
ORDER BY c.credit_limit ASC
LIMIT 1;
```

```
"""
result_low = connection.execute(text(query)).fetchall()
print("Customer with the Lowest Credit Limit:", result_low)
```

```
Customer with the Lowest Credit Limit: [('668', 0.0)]
```

Percentage of transactions declined

[236]:
```
query = """
SELECT
    (CAST(SUM(CASE WHEN errors IS NOT NULL THEN 1 ELSE 0 END) AS FLOAT) /␣
 ↪COUNT(*)) * 100 AS declined_percentage
FROM transactions;
"""
result = connection.execute(text(query)).fetchall()
print("Percentage of Transactions Declined:", result)
```

```
Percentage of Transactions Declined: [(100.0,)]
```

Top 5 Customers by Revenue Contribution

[238]:
```
query = """
WITH customer_revenue AS (
    SELECT
        t.client_id,
        SUM(t.amount) AS total_revenue
    FROM transactions t
    GROUP BY t.client_id
),
total_revenue AS (
    SELECT SUM(total_revenue) AS grand_total_revenue FROM customer_revenue
)
SELECT
    cr.client_id,
    cr.total_revenue,
    (cr.total_revenue * 100.0 / tr.grand_total_revenue) AS␣
 ↪contribution_percentage
FROM customer_revenue cr
CROSS JOIN total_revenue tr
ORDER BY cr.total_revenue DESC
LIMIT 5;
"""

result = connection.execute(text(query)).fetchall()
print("Top 5 Customers by Revenue Contribution:", result)
```

```
Top 5 Customers by Revenue Contribution: [('96', 2445773.25,
0.42770572213637753), ('1686', 2167880.9, 0.3791091696011313), ('1340',
2039921.23, 0.3567321634491168), ('840', 1956340.84, 0.34211600430133393),
('464', 1882901.35, 0.3292732397058109)]
```

What it reveals: Top customers by revenue contribution: Identifies the top 5 customers contributing the most to the bank's revenue.

Business Scenario: Helps prioritize loyalty programs and targeted offers for high-value customers to ensure retention and increase revenue.

Revenue Contribution by Card Type

[241]:
```
query = """
SELECT
    c.card_type,
    SUM(t.amount) AS total_revenue,
    (SUM(t.amount) * 100.0 / (SELECT SUM(amount) FROM transactions)) AS␣
↪revenue_percentage
FROM transactions t
JOIN cards c ON t.card_id = c.id
GROUP BY c.card_type
ORDER BY total_revenue DESC;
"""

result = connection.execute(text(query)).fetchall()
print("Average Credit Utilization by Customer:", result)
```

```
Average Credit Utilization by Customer: [('Debit', 326091498.38,
57.02540077954949), ('Credit', 225811820.01, 39.48894589647947), ('Debit
(Prepaid)', 19932203.89, 3.4856533239710443)]
```

What it reveals: Card type revenue contribution: Determines which card types (e.g., credit or debit) generate the most revenue.

Business Scenario: Guides product development and marketing strategies to enhance the value of top-performing card types.

1.6.4 4. Close the Connection

[244]:
```
connection.close()
```