In [1]:
```
import logging
from enum import Enum

logging.basicConfig(
    filename='auto_log.log',  # Log file location and name
    level=logging.INFO,  # Minimum log level to capture
    format='%(asctime)s [%(name)s] [%(levelname)s] %(message)s'
)

PaymentType = {'DAILY': 0.1, 'D_PARTIAL_RECURRING': 0.3, 'D_FULL_RECURRING': 0.3
               'MONTHLY_MINIMUM_DUE': 0.045, 'MONTHLY_MISS_DUE': 0.225}

# Enums classes
class TransactionType(Enum):
    PURCHASE = 'PURCHASE'
    CASH_ADVANCE = 'CASH_ADVANCE'
    PAYMENT = 'PAYMENT'
    PENALTY = 'PENALTY'
    INTEREST = 'INTEREST'
    BLOCKED = 'BLOCKED'


class EducationLevel(Enum):
    NO_EDUCATION = 0
    HIGH_SCHOOL = 1
    BACHELORS = 2
    MASTERS = 3
    PHD = 4
```

In [2]:
```
customer_logger = logging.getLogger('Customer')
account_logger = logging.getLogger('Account')
transaction_logger = logging.getLogger('Transaction')
report_logger = logging.getLogger('Report')


# Classes needed for our mapping

class Customer:
    def __init__(self, customer_id, age, gender, marital_status, number_of_child
                 number_of_accounts, total_credit_line):
        self.customer_id = customer_id
        self.age = age
        self.gender = gender
        self.marital_status = marital_status
        self.number_of_children = number_of_children
        self.education_level = education_level
        self.annual_income = annual_income
        self.number_of_accounts = number_of_accounts
        self.total_credit_line = total_credit_line

    def get_in_list_format(self):
        """
        Converts Customer object to list of values to store in the csv file
        :return: List of customer details in specific order
        """
        customer_logger.info(f'Converting from Customer object to list to store
        return [self.customer_id, self.age, self.gender, self.marital_status, se
                self.education_level.name, '{:.2f}'.format(self.annual_income),
```

```
                '{:.2f}'.format(self.total_credit_line)]

    @staticmethod
    def get_customer_from_list(customer):
        """
        Converts list of values from csv file to Customer object
        :param customer: List of customer details
        :return: Customer object from given values
        """
        customer_logger.info(f'Converting list of value to Customer object for :
        return Customer(customer[0], int(customer[1]), customer[2], customer[3],
                        EducationLevel[customer[5]],
                        float(customer[6]), int(customer[7]), float(customer[8])

    @staticmethod
    def get_customer_from_list2(customer):
        """
        Converts list of values from csv file to Customer object
        :param customer: List of customer details
        :return: Customer object from given values
        """
        customer_logger.info(f'Converting list of value to Customer object for :
        return Customer(customer[0], int(customer[1]), customer[2], customer[3],
                        customer[5],
                        float(customer[6]), int(customer[7]), float(customer[8])


class Account:
    def __init__(self, customer_id, account_number, date_opened, account_credit_
                 balance):
        self.account_number = account_number
        self.customer_id = customer_id
        self.date_opened = date_opened
        self.account_credit_line = account_credit_line
        self.annual_fee = annual_fee
        self.annual_interest_rate = annual_interest_rate
        self.balance = balance
        self.delinquency = 0

    def get_in_list_format(self):
        """
        Converts Account object to list of values to store in the csv file
        :return: List of account details in specific order
        """
        account_logger.info(
            f'Converting from Account object to list to store for customer : {se
        return [self.customer_id, self.account_number, self.date_opened, '{:.2f}
                '{:.2f}'.format(self.annual_fee),
                self.annual_interest_rate, '{:.2f}'.format(self.balance)]

    @classmethod
    def get_Account_from_list(cls, account):
        """
        Converts list of values from csv file to Account object
        :param account: List of customer details
        :return: Account object from given values
        """
        customer_logger.info(
            f'Converting list of value to Account object for customer : {account
        return Account(account[0], account[1], account[2], float(account[3]), fl
```

```
                        float(account[6]))


class Transaction:
    def __init__(self):
        self.customer_id = None
        self.account_number = None
        self.transaction_id = None
        self.transaction_date = None
        self.transaction_type = None
        self.opening_balance = None
        self.transaction_amount = 0
        self.closing_balance = None
        self.available_credit_line = None

    def get_in_list_format(self):
        """
        Converts Transaction object to list of values to store in the csv file
        :return: List of transaction details in specific order
        """
        account_logger.info(
            f'Converting from Transaction object to list to store for transactio
        return [self.customer_id, self.account_number, self.transaction_id, self
                self.transaction_type, '{:.2f}'.format(self.available_credit_lin
                '{:.2f}'.format(self.opening_balance), '{:.2f}'.format(self.tran
                '{:.2f}'.format(self.closing_balance)]


class MonthlyReport:
    def __init__(self):
        self.customer_id = 0
        self.account_number = 0
        self.month = 0
        self.closing_balance = 0
        self.total_purchases = 0
        self.total_cash_advances = 0
        self.total_payments = 0
        self.total_interests_charged = 0

    def get_in_list_format(self):
        """
        Converts MonthlyReport object to list of values to store in the csv file
        :return: List of monthly report details in specific order
        """
        account_logger.info(
            f'Converting from Transaction object to list to store for account :
        return [self.customer_id, self.account_number, self.month, '{:.2f}'.form
                '{:.2f}'.format(self.total_purchases),
                '{:.2f}'.format(self.total_cash_advances), '{:.2f}'.format(self.
                '{:.2f}'.format(self.total_interests_charged)]
```

In [3]:

```
# Excel utility functions to store and retrieve data needed
import csv
import os


def store_customer_data(customers):
    """
    Store customers data into csv file in data folder in the same location
    :param customers: List of customers with list of values each to store in csv
```

```
    """
    if not os.path.exists("data"):
        customer_logger.info('Creating data folder if not exists')
        os.mkdir('data')
    with open('data/customer_data.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        customer_logger.info('Writing headers into customer_data.csv file')
        csv_writer.writerow(
            ['CustomerID', 'Age', 'Gender', 'MaritalStatus', 'NumChildren', 'Edu
             'NumAccounts', 'TotalCreditLine'])
        customer_logger.info('Writing customer details into customer_data.csv fi
        csv_writer.writerows(customers)
    customer_logger.info(f'Customer details have successfully stored into csv fi


def get_customers():
    """
    Get customers data from customer_data.csv file
    :return: List of Customer objects from the csv file
    """
    with open('data/customer_data.csv', 'r') as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=',')
        next(csv_reader)
        customers = []
        for row in csv_reader:
            customers.append(Customer.get_customer_from_list(row))
    customer_logger.info(f'Fetched customer details from customer_data.csv. Cust
    return customers


def get_customers_for_report():
    """
    Get customers data from customer_data.csv file
    :return: List of Customer objects from the csv file
    """
    with open('data/customer_data.csv', 'r') as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=',')
        next(csv_reader)
        customers = []
        for row in csv_reader:
            customers.append(Customer.get_customer_from_list2(row))
    customer_logger.info(f'Fetched customer details from customer_data.csv. Cust
    return customers


def store_account_data(accounts):
    """
    Store accounts data into csv file in data folder in the same location
    :param accounts: List of accounts with list of values each to store in csv f
    """
    path = 'data/'
    if not os.path.exists(path):
        account_logger.info('Creating data folder if not exists')
        os.mkdir(path)
    with open(path + 'account_data.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        customer_logger.info('Writing headers into account_data.csv file')
        csv_writer.writerow(
            ['CustomerID', 'AccountID', 'DateOpened', 'AccountCreditLine', 'Annu
             'Balance'])
```

```
        account_logger.info('Writing account details into account_data.csv file'
        csv_writer.writerows(accounts)
    account_logger.info(f'Account details have successfully stored into csv file


def get_accounts_for_customer():
    """
    Get account data from account_data.csv file
    :return: List of Account objects from the csv file
    """
    path = 'data/'
    with open(path + 'account_data.csv', 'r') as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=',')
        next(csv_reader)
        accounts_data = dict()
        for row in csv_reader:
            row_account = Account.get_Account_from_list(row)
            if row_account.customer_id not in accounts_data:
                accounts_data[row_account.customer_id] = []
            accounts_data[row_account.customer_id].append(row_account)
    account_logger.info('Fetched account details from account_data.csv successfu
    return accounts_data


def get_accounts_for_customer_for_report():
    """
    Get account data from account_data.csv file
    :return: List of Account objects from the csv file
    """
    path = 'data/'
    with open(path + 'account_data.csv', 'r') as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=',')
        next(csv_reader)
        accounts_data = []
        for row in csv_reader:
            row_account = Account.get_Account_from_list(row)
            accounts_data.append(row_account)
    account_logger.info('Fetched account details from account_data.csv successfu
    return accounts_data


def store_transaction_data(transactions):
    """
    Store transactions data into csv file in data folder in the same location
    :param transactions: List of transactions with list of values each to store
    """
    path = 'data/'
    if not os.path.exists(path):
        transaction_logger.info('Creating data folder if not exists')
        os.mkdir(path)
    with open(path + 'transactions_data.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        transaction_logger.info('Writing headers into transactions_data.csv file
        csv_writer.writerow(['Customer ID',
                             'Account Number',
                             'Transaction Id',
                             'Transaction Date',
                             'Transaction Type',
                             'Available Credit',
                             'Opening Balance',
```

```
                                'Transaction Amount',
                                'Closing Balance'])
            transaction_logger.info('Writing transaction details into transactions_d
            csv_writer.writerows(transactions)
        transaction_logger.info(
            f'Transaction details have successfully stored into csv file. Transactio


def store_monthly_report_data(monthly_reports):
    """
    Store monthly report data into csv file in data folder in the same location
    :param monthly_reports: List of reports with list of values each to store in
    """
    path = 'data/'
    if not os.path.exists(path):
        report_logger.info('Creating data folder if not exists')
        os.mkdir(path)
    with open(path + 'monthly_reports_data.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        transaction_logger.info('Writing headers into monthly_reports_data.csv f
        csv_writer.writerow(
            ['Customer Id', 'Account Number', 'Month', 'Closing Balance', 'Total
             'Total Payments', 'Total Interests Charged'])
        transaction_logger.info('Writing transaction details into monthly_report
        csv_writer.writerows(monthly_reports)
    report_logger.info(
        f'Monthly report details have successfully stored into csv file. Reports
```

In [4]:

```
# Customer related functionalities

def get_customer_id(customer_index):
    """
    Return customer id for a customer
    :param customer_index: index of customer to create id
    :return: Customer id
    """
    customer_logger.info(f'Getting customer id for customer : {customer_index}')
    return 1000000 + customer_index


def get_age():
    """
    Return random age from 20 to 80 both included
    :return: random age
    """
    age = random.randint(20, 80)
    customer_logger.info(f'Random age for customer : {age}')
    return age


def get_gender():
    """
    Return male or female with half probability
    :return: random gender
    """
    gender = random.choice(['Male', 'Female'])
    customer_logger.info(f'Random gender for customer : {gender}')
    return gender
```

```
def get_martial_status(age):
    """
    Return marital status according to the age group of customer with specific p
    :param age: age of the customer
    :return: marital status - Single or Married
    """
    if 20 <= age <= 30:
        marital_status = random.choices(['Single', 'Married'], weights=[0.75, 0.
    elif 30 < age <= 60:
        marital_status = random.choices(['Single', 'Married'], weights=[0.25, 0.
    else:
        marital_status = random.choices(['Single', 'Married'], weights=[0.5, 0.5
    customer_logger.info(f'Marital status of the customer : {marital_status}')
    return marital_status


def get_number_of_children(age):
    """
    Return number of children according to the age group of customer with specif
    :param age: age of the customer
    :return: number of children
    """
    num_children = 0
    if 20 <= age <= 40:
        num_children = random.choices(range(5), weights=[0.4, 0.3, 0.2, 0.1, 0])
    elif 40 < age <= 80:
        num_children = random.choices(range(5), weights=[0.1, 0.3, 0.3, 0.2, 0.1
    customer_logger.info(f'Number of children for the customer : {num_children}'
    return num_children


def get_education_level(age):
    """
    Return education level according to the age group of customer with specific
                    NO_EDUCATION, HIGH_SCHOOL, BACHELORS, MASTERS, PHD
    :param age: age of the customer
    :return: education level
    """
    education_level = EducationLevel.NO_EDUCATION
    education_level_types = [EducationLevel.NO_EDUCATION, EducationLevel.HIGH_SC
                             EducationLevel.MASTERS, EducationLevel.PHD]
    if 20 <= age <= 25:
        education_level = random.choices(education_level_types, weights=[0.1, 0.
    elif 25 < age <= 35:
        education_level = random.choices(education_level_types, weights=[0.1, 0.
    elif 35 < age <= 80:
        education_level = random.choices(education_level_types, weights=[0.1, 0.
    customer_logger.info(f'Education level of the customer : {education_level}')
    return education_level


def get_annual_income(age, education_level):
    """
    Return annual income according to the age and education of the customer with
        annual_income = 40 * 52 * (15 + education_level.value * 10 + (age / 10)
    :param age: age of the customer
    :param education_level: education level of the customer
    :return: annual income of the customer
    """
    annual_income = 40 * 52 * (15 + education_level.value * 10 + (age / 10) * 2)
```

```
    customer_logger.info(f'Annual income of the customer : {annual_income}')
    return annual_income


def get_num_of_accounts(marital_status, num_children):
    """
    Return number of accounts according to the marital status and number of chil
    :param marital_status: marital status of the custmer
    :param num_children: number of children
    :return: number of accounts
    """
    marital_status_factor = 0 if marital_status == 'Single' else 1
    num_accounts = marital_status_factor + num_children + 1
    customer_logger.info(f'Number of accounts for the customer : {num_accounts}'
    return num_accounts


def get_total_credit_line(num_accounts, annual_income):
    """
    Return total credit customer has according to his annual income
    :param num_accounts: number of accounts
    :param annual_income: annual income
    :return: total credit line for the customer
    """
    total_credit_line = num_accounts * (annual_income / 10)
    customer_logger.info(f'Total credit line for the customer : {total_credit_li
    return total_credit_line


def generate_customer(customer_index):
    """
    Generate a new customer with all the requirements
    :param customer_index: index of the customer
    :return: Newly created customer object
    """
    customer_logger.info(f'Generating new customer data for customer : {customer
    customer_id = get_customer_id(customer_index)
    age = get_age()
    gender = get_gender()
    marital_status = get_martial_status(age)
    number_of_children = get_number_of_children(age)
    education_level = get_education_level(age)
    annual_income = get_annual_income(age, education_level)
    number_of_accounts = get_num_of_accounts(marital_status, number_of_children)
    total_credit_line = get_total_credit_line(number_of_accounts, annual_income)
    customer = Customer(customer_id, age, gender, marital_status, number_of_chil
                        number_of_accounts, total_credit_line)
    customer_logger.info(f'New customer data generated successfully for customer
    return customer
```

In [5]:

```
# Account related functionalities

def get_date_opened(age):
    """
    Return random date opened for account according to the age of the customer
    :param age: age
    :return: Random date opened
    """
    min_year = 2022 - (age - 19)
    date_opened = date(random.randint(min_year, 2021), random.randint(1, 12), ra
```

```
    account_logger.info(f'Random date opened for the account : {date_opened}')
    return date_opened


def get_account_number(customer_id, account_index):
    """
    Return account number for given account index and customer with below calcul
            'customer_id' + 'account_index'
    :param customer_id: customer id
    :param account_index: account index
    :return: account number
    """
    account_number = int(f"{customer_id}{account_index}")
    account_logger.info(f'Account number for given account index and customer :
    return account_number


def get_account_credit_line(total_credit_line):
    """
    Return account credit which is some random portion of the total credit line
    :param total_credit_line: total credit line of the customer
    :return:
    """
    account_credit_line = random.uniform(0.0, 1.0) * total_credit_line
    account_logger.info(f'Account credit line for given account : {account_credi
    return account_credit_line


def get_annual_fee(account_credit_line):
    """
    Return annual fee according to the account credit with below calculation
                account_credit_line * 0.01
    :param account_credit_line: account credit line
    :return: annual fee
    """
    annual_fee = account_credit_line * 0.01
    account_logger.info(f'Annual fee for given account : {annual_fee}')
    return annual_fee


def get_annual_interest_rate():
    """
    Return random interest rate from 15 to 30 percent
    :return: annual interest rate
    """
    annual_interest_rate = round(random.uniform(15, 30), 2)
    account_logger.info(f'Annual interest rate for given account : {annual_inter
    return annual_interest_rate


def generate_account(customer, account_index, total_credit_line_available,number
    """
    Generate new account data with all requirements
    :param customer: customer data
    :param account_index: index of the account
    :return: Newly created account object
    """
    date_opened = get_date_opened(customer.age)
    account_number = get_account_number(customer.customer_id, account_index)
    if account_index == number_of_accounts:
```

```
        account_credit_line = total_credit_line_available
    else:
        account_credit_line = get_account_credit_line(total_credit_line_availabl
    temp_total_credit_line_available = total_credit_line_available - account_cre
    annual_fee = get_annual_fee(account_credit_line)
    annual_interest_rate = get_annual_interest_rate()
    account = Account(customer.customer_id, account_number, date_opened, account
                      annual_interest_rate, 0.0)
    account_logger.info(
        f'New account data generated successfully for given customer : {customer
    return account, temp_total_credit_line_available
```

In [6]:
```
# Transaction related functionalities

import random
from datetime import date


def get_random_num_of_days_1_to_7():
    """
    Return random number from 1 to 7
    :return: random number
    """
    d = random.randint(1, 7)
    transaction_logger.info(f'Random generated number from 1 to 7 : {d}')
    return d


def get_random_num_of_days_1_to_10():
    """
    Return random number from 1 to 10
    :return: random number
    """
    d = random.randint(1, 10)
    transaction_logger.info(f'Random generated number from 1 to 10 : {d}')
    return d


def get_random_portion_of_balance():
    """
    Return random portion from 0 to 1
    :return: random portion
    """
    d = random.random()
    transaction_logger.info(f'Random portion from 0 to 1 : {d}')
    return d


def get_purchase_or_cash():
    """
    Return transaction type from PURCHASE or CASH with probabilities
    :return: transaction type
    """
    transaction_type = TransactionType.PURCHASE if random.random() <= 0.95 else
    transaction_logger.info(f'Type of the transaction : {transaction_type}')
    return transaction_type


def get_available_credit_line(account_credit_line, balance):
    """
```

```
    Return available credit limit with below calculation
                account_credit_line - balance
    :param account_credit_line: total account credit limit
    :param balance: credit balance for a customer
    :return: available credit limit
    """
    available_credit_line = account_credit_line - balance
    transaction_logger.info(f'Available credit limit for the transaction : {avai
    return available_credit_line


def get_available_cash(account_credit_line, balance):
    """
    Return available cash advance with below calculation
                   min(0.1 * account_credit_line, account_credit_line - balance
    :param account_credit_line: total account credit limit
    :param balance: credit balance for a customer
    :return: available cash advance
    """
    available_cash = min(0.1 * account_credit_line, account_credit_line - balanc
    transaction_logger.info(f'Available cash advance for the transaction : {avai
    return available_cash


def get_interest_for_the_days(annual_interest_rate: float, balance: float, num_o
    """
    Return interest for given number of days for the balance
    :param annual_interest_rate: annual interest rate
    :param balance: credit balance for a customer
    :param num_of_days: number of days for interest to be calculated
    :return: interest for the days
    """
    day_interest_rate = annual_interest_rate / 36500
    interest = balance * (pow((1 + day_interest_rate), num_of_days) - 1)
    transaction_logger.info(f'Interest to be added for the days : {interest}')
    return interest


def purchase_something(available_credit_line):
    """
    Return random purchase amount within available credit limit
    :param available_credit_line: available credit
    :return: random purchase amount
    """
    purchase_amount = random.uniform(0, available_credit_line)
    transaction_logger.info(f'Random purchase amount : {purchase_amount}')
    return purchase_amount


def withdraw_some_cash(available_cash):
    """
    Return random cash amount within available cash advance
    :param available_cash: available cash
    :return: random cash amount
    """
    cash_amount = random.uniform(0, available_cash)
    transaction_logger.info(f'Random cash amount : {cash_amount}')
    return cash_amount
```

```
def get_payment_type():
    """
    Return payment type according to the probabilities mentioned in @PaymentType
    :return: payment type
    """
    payment_types = [type_temp for type_temp in PaymentType]
    weights = [PaymentType[type_temp] for type_temp in payment_types]
    payment_type = random.choices(payment_types, weights=weights)[0]
    transaction_logger.info(f'Payment type for next payment : {payment_type}')
    return payment_type


def get_next_payment_day(current_payment_day, payment_type):
    """
    Return next payment day according to payment type and current payment day
    :param current_payment_day: current payment day
    :param payment_type: payment type
    :return: next payment day
    """
    next_payment_day = 10000
    if payment_type == 'DAILY':
        next_payment_day = current_payment_day + 1
    elif payment_type == 'D_FULL_RECURRING' or payment_type == 'D_PARTIAL_RECURR
        next_payment_day = current_payment_day + get_random_num_of_days_1_to_7()
    elif payment_type == 'MONTHLY_ENTIRE_BALANCE' or payment_type == 'MONTHLY_MI
        next_payment_day = get_random_num_of_days_1_to_10()
    transaction_logger.info(f'Next payment day : {next_payment_day} for given pa
    return next_payment_day


def generate_transaction(temp_id, month, account, current_transaction_day):
    """
    Generate transaction for current day with given data
    :param temp_id: id to create transaction id
    :param month: month
    :param account: account
    :param current_transaction_day: current transaction day
    :return: transaction data, interest transaction, purchase amount, cash advan
    """
    purchase_amount = 0
    cash_advance = 0
    # Get basic transaction
    current_transaction = get_basic_transaction(account, current_transaction_day
    # Get transaction type
    current_transaction_type = get_purchase_or_cash()
    main_logger.info(f'Current transaction type is {current_transaction_type}')
    # If transaction type is PURCHASE
    if current_transaction_type is TransactionType.PURCHASE:
        # Get available credit limit
        available_credit_line = get_available_credit_line(account.account_credit
        current_transaction.available_credit_line = available_credit_line
        # if available credit is greater than zero
        if available_credit_line > 0:
            # Purchase something for transaction
            purchase_amount = purchase_something(available_credit_line)
            main_logger.info(f'Customer purchased an amount of {purchase_amount}
            current_transaction.transaction_amount = purchase_amount
            account.balance += purchase_amount
    # if transaction type is CASH
    else:
```

```
        # Get available cash advance
        available_cash = get_available_cash(account.account_credit_line, account
        current_transaction.available_credit_line = available_cash
        # if available cash advance is greater than zero
        if available_cash > 0:
            # Withdraw some cash for the transaction
            cash_advance = withdraw_some_cash(available_cash)
            main_logger.info(f'Customer has withdrawn cash of {cash_advance}')
            current_transaction.transaction_amount = cash_advance
            account.balance += cash_advance
    current_transaction.transaction_type = current_transaction_type.name
    current_transaction.closing_balance = account.balance
    # Calculate the interest
    main_logger.info('Calculate the interest to be added')
    interest = get_interest_for_the_days(account.annual_interest_rate, account.b
                                         current_transaction_day)
    # Generate interest transaction
    interest_transaction = get_interest_transaction(account, current_transaction
    return current_transaction, interest_transaction, purchase_amount, cash_adva


def generate_monthly_activity(account, month, payment_type, last_month_report):
    """
    Generate monthly activity for given month and given account information
    :param account: account data
    :param month: month
    :param payment_type: payment type
    :param last_month_report: last month report
    :return: monthly report, transactions for given month
    """
    main_logger.info('Generating monthly activity data')
    monthly_report = MonthlyReport()
    # Get random current transaction day
    current_trans_day = get_random_num_of_days_1_to_7()
    # Get random payment day
    payment_day = get_next_payment_day(0, payment_type)
    months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    transactions = []
    # Create basic id for creating transaction ids by incrementing
    temp_id = (month * 100) + (int(account.account_number) * 10000)
    is_blocked = False
    # Iterate over days for looping
    for day in range(1, months[month - 1] + 1):
        # account delinquency should not be greater than or equal to 3
        if account.delinquency < 3:
            # If day is transaction day create transaction
            if day == current_trans_day:
                main_logger.info(f'Generating transaction for day : {day}')
                temp_id += 1
                # Create transaction and calculate interest
                current_transaction, interest_transaction, purchase_amount, cash
                    temp_id, month,
                    account,
                    current_trans_day)
                # Update monthly report values
                monthly_report.total_purchases += purchase_amount
                monthly_report.total_cash_advances += cash_advance
                monthly_report.total_interests_charged += interest
                # Collect transactions and interest transactions to store finall
                transactions.append(current_transaction.get_in_list_format())
```

```
                transactions.append(interest_transaction.get_in_list_format())
                # update current transaction day for upcoming transaction
                current_trans_day += get_random_num_of_days_1_to_7()
                # if current transaction day exceeds month days exit the loop
                if current_trans_day > months[month - 1]:
                    break
            # If day is payment day and account balance is not zero proceed to p
            if day == payment_day and account.balance != 0.0:
                main_logger.info(f'Generating payment transaction for day : {day
                temp_id += 1
                # Create payment transaction
                payment_transaction = generate_payment_transaction(account, day,
                                                                   month, paymen
                # update monthly reports data
                monthly_report.total_payments += payment_transaction.transaction
                # Collect transactions to store finally
                transactions.append(payment_transaction.get_in_list_format())
                # Get next payment day
                payment_day = get_next_payment_day(payment_day, payment_type)
        else:
            # If account is blocked exit the loop
            main_logger.info('Account is blocked due to delinquency greater than
            is_blocked = True
            break
    # update the monthly report
    monthly_report.month = month
    monthly_report.customer_id = account.customer_id
    monthly_report.account_number = account.account_number
    monthly_report.closing_balance = account.balance
    return monthly_report, transactions, is_blocked


def generate_payment_transaction(account, current_transaction_day, last_month_re
    """
    Create payment transaction with given details according to payment_type.
    :param account: account data
    :param current_transaction_day: current transaction day
    :param last_month_report: last month report
    :param month: month
    :param payment_type: payment type
    :param temp_id: id to create transaction id
    :return: payment transaction
    """
    # Get basic transaction
    payment_transaction = get_basic_transaction(account, current_transaction_day
    payment_transaction.transaction_type = TransactionType.PAYMENT.name
    # Get available credit limit
    payment_transaction.available_credit_line = get_available_credit_line(accoun
    # if payment type is DAILY or D_FULL_RECURRING, pay full account balance
    if payment_type == 'DAILY' or payment_type == 'D_FULL_RECURRING':
        payment_transaction.transaction_amount = account.balance
        account.delinquency = 0
        account.balance = 0
    # if payment type is D_PARTIAL_RECURRING, pay partial account balance
    elif payment_type == 'D_PARTIAL_RECURRING':
        payment_transaction.transaction_amount = account.balance * get_random_po
        account.delinquency = 0
        account.balance -= payment_transaction.transaction_amount
    # if payment type is MONTHLY_ENTIRE_BALANCE, pay full last month due
    elif payment_type == 'MONTHLY_ENTIRE_BALANCE':
```

```
            payment_transaction.transaction_amount = last_month_report.closing_balan
            account.delinquency = 0
            account.balance -= payment_transaction.transaction_amount
        # if payment type is MONTHLY_MINIMUM_DUE, pay last month minimum due
        elif payment_type == 'MONTHLY_MINIMUM_DUE':
            payment_transaction.transaction_amount = last_month_report.closing_balan
            account.delinquency = 0
            account.balance -= payment_transaction.transaction_amount
        # if payment type not among above, Add 30 fine and change payment to PENALTY
        else:
            payment_transaction.transaction_amount = 30
            payment_transaction.transaction_type = TransactionType.PENALTY.name
            account.balance += 30
            account.delinquency += 1
            # if delinquency greater than 3, block the account and update transactio
            if account.delinquency >= 3:
                payment_transaction.transaction_type = TransactionType.BLOCKED.name
        payment_transaction.closing_balance = account.balance
        return payment_transaction


    def get_basic_transaction(account, current_transaction_day, month, temp_id):
        """
        Create basic transaction
        :param account: account data
        :param current_transaction_day: current transaction day
        :param month: month
        :param temp_id: id for transaction id
        :return: sample transaction
        """
        transaction = Transaction()
        transaction.customer_id = account.customer_id
        transaction.account_number = account.account_number
        transaction.transaction_id = temp_id
        transaction.transaction_date = date(2022, month, current_transaction_day)
        transaction.opening_balance = account.balance
        transaction_logger.info(f'Transaction has been created successfully. Id : {t
        return transaction


    def get_interest_transaction(account, current_transaction_day, month, temp_id, i
        """
        Create basic interest transaction to store interest information at end of th
        :param account: account data
        :param current_transaction_day: current transaction day
        :param month: month
        :param temp_id: id to create transaction id
        :param interest: interest amount
        :return: interest transaction
        """
        temp_id += 1
        # Get basic transaction
        interest_transaction = get_basic_transaction(account, current_transaction_da
        # Update transaction type to INTEREST
        interest_transaction.transaction_type = TransactionType.INTEREST.name
        # Get available credit limit
        interest_transaction.available_credit_line = get_available_credit_line(accou
        interest_transaction.transaction_amount = interest
        interest_transaction.opening_balance = account.balance
        account.balance += interest
```

```
    interest_transaction.closing_balance = account.balance
    transaction_logger.info(f'Interest transaction has been created successfully
    return interest_transaction


def generate_account_activity_for_all_customers():
    """
    Generate account activity for twelve months for all accounts for all custome
    And store transaction and monthly report data in different respective csv fi
    """
    months = range(1, 13)
    # Get customer data which was stored previously from customer_data.csv file
    customers = get_customers()
    last_month_report = MonthlyReport()
    monthly_reports = []
    total_transactions = []
    # Iterate through from all customers
    accounts_data = get_accounts_for_customer()
    for customer in customers:
        # Get accounts data for given customer which was stored previously from
        main_logger.info(f'Generating account activity for customer : {customer.
        # Iterate through all accounts for customer
        for account in accounts_data[customer.customer_id]:
            main_logger.info(f'Generating account activity for account : {accoun
            # Iterate through months to generate monthly activity
            for month in months:
                main_logger.info(f'Generating account activity for month : {mont
                # Get payment type for upcoming month
                payment_type = get_payment_type()
                # Generate monthly activity and get the transaction, monthly rep
                monthly_report, transactions, is_blocked = generate_monthly_acti

                # Collect monthly reports to store finally
                monthly_reports.append(monthly_report.get_in_list_format())
                # Collect transactions to store finally
                total_transactions.extend(transactions)
                last_month_report = monthly_report
                if is_blocked:
                    break
    # Store transactions data into transaction_data.csv
    main_logger.info('Storing transactions data')
    store_transaction_data(total_transactions)
    # Store monthly reports data into monthly_reports_data.csv
    main_logger.info('Storing monthly reports data')
    store_monthly_report_data(monthly_reports)
```

In [7]:

```
## Generate customer activity data for given number of customers
# Main Code

main_logger = logging.getLogger('main')

# Input from user for number of customer
num_of_customers = int(input('Enter the number of customers the account data nee
main_logger.info(
    f'==================================  Generating activity for {num_of_cust
customers_data = []
accounts_data = []
main_logger.info(
    f'================================================  Generating customer D
for customer_index in range(1, num_of_customers + 1):
```

```
    main_logger.info(
        f'--------------------------------------------------------------------
    # Generate new customer
    new_customer = generate_customer(customer_index)
    # Collect newly customer data generated into list to store finally
    customers_data.append(new_customer.get_in_list_format())
    main_logger.info(
        f'----------------------------------------  Generating Account Data fo
    total_credit_line_available = new_customer.total_credit_line
    for account_index in range(1, new_customer.number_of_accounts + 1):
        # Generate new account for given customer
        new_account, total_credit_line_available = generate_account(new_customer
        # Collect newly account data generated into list to store finally
        accounts_data.append(new_account.get_in_list_format())
# Store accounts data into account_data.csv file
main_logger.info('Storing accounts data')
store_account_data(accounts_data)
# Store customers data into customer_data.csv file
main_logger.info('Storing customers data')
store_customer_data(customers_data)
# Generate account activity for all accounts for all customers
main_logger.info('Generating the account activity for all customers')
generate_account_activity_for_all_customers()
```

In [8]:

```
# customer report
import pandas

customers_report_list = []

for c in get_customers():
    customers_report_list.append(c.__dict__)

customer_dataframe = pandas.DataFrame(customers_report_list)

# 1. Number of customers in your results
print(f'Total number of customers : {len(customers_report_list)}')

# 2. The minimum and maximum Customer ID in your results
print(f'Minimum value of the customer id field : {customer_dataframe["customer_i
print(f'Maximum value of the customer id field : {customer_dataframe["customer_i

# 3. Number of unique Customer IDs
print(f'Number of unique customer ids : {customer_dataframe.customer_id.nunique(

# 4. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the age of cus
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the age of c
print(customer_dataframe.age.describe())


def create_frequency_table(field, dataframe):
    max_count = len(dataframe)
    temp_freq = dataframe[field].value_counts().reset_index()
    temp_freq.columns = [field, 'Frequency']
    temp_freq['Percentage Frequency'] = temp_freq['Frequency'] *100/ max_count
    temp_freq['Cumulative Frequency'] = temp_freq['Frequency'].cumsum()
    temp_freq['Cumulative Percentage Frequency'] = temp_freq['Cumulative Frequen
    return temp_freq

# 5. Frequency table for Gender of the customers
```

```
gender_freq = create_frequency_table('gender',customer_dataframe)
print()
print('Frequency table for Gender of the customers')
print(gender_freq)

# 6. Frequency table for Marital Status of the customer
marital_status_freq = create_frequency_table('marital_status',customer_dataframe
print()
print('Frequency table for Marital Status of the customer')
print(marital_status_freq)

# 7. Provide percent frequency of Marital Status for the following categories se
# a. For customers with age in [20, 30]
marital_status_age_20_30 = len(customer_dataframe[customer_dataframe.age.between
# b. For customers with age in (30, 60]
marital_status_age_30_60 = len(customer_dataframe[customer_dataframe.age.between
# c. For customers with age in (60, 80]
marital_status_age_60_80 = len(customer_dataframe[customer_dataframe.age.between
print()
print('PercentGW frequency of Marital Status')
print('                                          Percent-Frequency ')
print(f'For customers with age in [20, 30]        {marital_status_age_20_30 * 10
print(f'For customers with age in (30, 60]        {marital_status_age_30_60 * 10
print(f'For customers with age in (60, 80]        {marital_status_age_60_80 * 10

# 8. Frequency table for Number of Children of the customers
number_of_children_freq = create_frequency_table('number_of_children',customer_d
print()
print('Frequency table for Number of Children of the customers')
print(number_of_children_freq)

# 9. Provide percent frequency of Number of Children for the following categorie
print()
# a. For customers with age in [20, 40]
num_children_age_20_40_freq = create_frequency_table('number_of_children',custom
print('Percent frequency of Number of Children for customers with age in [20, 40
print(num_children_age_20_40_freq)
# b. For customers with age in (40, 80]
num_children_age_40_80_freq = create_frequency_table('number_of_children',custom
print('Percent frequency of Number of Children for customers with age in (40, 80
print(num_children_age_40_80_freq)

# 10. Frequency table for Education Level of the customers
education_level_freq = create_frequency_table('education_level',customer_datafra
print()
print('Frequency table for Education Level of the customers')
print(education_level_freq)

# 11. Provide percent frequency of Education Level for the following categories
# a. For customers with age in [20, 25]
education_level_age_20_25_freq = create_frequency_table('education_level',custom
print()
print('Percent frequency of Education Level for customers with age in [20, 25]')
print(education_level_age_20_25_freq)
# # b. For customers with age in (25, 35]
education_level_age_25_35_freq = create_frequency_table('education_level',custom
print()
print('Percent frequency of Education Level for customers with age in (25, 35]')
print(education_level_age_25_35_freq)
# # c. For customers with age in (35, 80]
```

```
education_level_age_35_80_freq = create_frequency_table('education_level',custom
print()
print('Percent frequency of Education Level for customers with age in (25, 35]')
print(education_level_age_35_80_freq)

# 12. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual In
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual I
print(customer_dataframe.annual_income.describe())

# 13. Number of accounts
# number_of_accounts_freq = dict(customer_dataframe.number_of_accounts.value_cou
number_of_accounts_freq = create_frequency_table('number_of_accounts',customer_d
print()
print('Frequency table for Number of accounts of the customers')
print(number_of_accounts_freq)

# 14. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account C
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account
print(customer_dataframe.total_credit_line.describe())
```

```
Total number of customers : 1500
Minimum value of the customer id field : 1000001
Maximum value of the customer id field : 1001500
Number of unique customer ids : 1500

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the age of customers
:
count    1500.000000
mean       50.069333
std        17.206953
min        20.000000
25%        36.000000
50%        50.000000
75%        64.000000
max        80.000000
Name: age, dtype: float64

Frequency table for Gender of the customers
   gender  Frequency  Percentage Frequency  Cumulative Frequency  \
0  Female        762                  50.8                   762
1    Male        738                  49.2                  1500

   Cumulative Percentage Frequency
0                             50.8
1                            100.0

Frequency table for Marital Status of the customer
  marital_status  Frequency  Percentage Frequency  Cumulative Frequency  \
0        Married        874             58.266667                   874
1         Single        626             41.733333                  1500

   Cumulative Percentage Frequency
0                        58.266667
1                       100.000000

PercentGW frequency of Marital Status
                                         Percent-Frequency
For customers with age in [20, 30]       17.00
For customers with age in (30, 60]       49.93
For customers with age in (60, 80]       33.07

Frequency table for Number of Children of the customers
   number_of_children  Frequency  Percentage Frequency  Cumulative Frequency  \
0                   1        453             30.200000                   453
1                   2        411             27.400000                   864
2                   0        296             19.733333                  1160
3                   3        249             16.600000                  1409
4                   4         91              6.066667                  1500

   Cumulative Percentage Frequency
0                        30.200000
1                        57.600000
2                        77.333333
3                        93.933333
4                       100.000000

Percent frequency of Number of Children for customers with age in [20, 40]
   number_of_children  Frequency  Percentage Frequency  Cumulative Frequency  \
0                   0        205             40.196078                   205
1                   1        152             29.803922                   357
```

```
2                   2        107             20.980392                    464
3                   3         46              9.019608                    510

   Cumulative Percentage Frequency
0                      40.196078
1                      70.000000
2                      90.980392
3                     100.000000
Percent frequency of Number of Children for customers with age in (40, 80]
   number_of_children  Frequency  Percentage Frequency  Cumulative Frequency  \
0                   2        304             30.707071                    304
1                   1        301             30.404040                    605
2                   3        203             20.505051                    808
3                   4         91              9.191919                    899
4                   0         91              9.191919                    990

   Cumulative Percentage Frequency
0                      30.707071
1                      61.111111
2                      81.616162
3                      90.808081
4                     100.000000

Frequency table for Education Level of the customers
              education_level  Frequency  Percentage Frequency  \
0   EducationLevel.HIGH_SCHOOL        770             51.333333
1     EducationLevel.BACHELORS        362             24.133333
2  EducationLevel.NO_EDUCATION        152             10.133333
3       EducationLevel.MASTERS        147              9.800000
4           EducationLevel.PHD         69              4.600000

   Cumulative Frequency  Cumulative Percentage Frequency
0                   770                        51.333333
1                  1132                        75.466667
2                  1284                        85.600000
3                  1431                        95.400000
4                  1500                       100.000000

Percent frequency of Education Level for customers with age in [20, 25]
              education_level  Frequency  Percentage Frequency  \
0   EducationLevel.HIGH_SCHOOL         63             46.666667
1     EducationLevel.BACHELORS         38             28.148148
2       EducationLevel.MASTERS         18             13.333333
3  EducationLevel.NO_EDUCATION         16             11.851852

   Cumulative Frequency  Cumulative Percentage Frequency
0                    63                        46.666667
1                   101                        74.814815
2                   119                        88.148148
3                   135                       100.000000

Percent frequency of Education Level for customers with age in (25, 35]
              education_level  Frequency  Percentage Frequency  \
0   EducationLevel.HIGH_SCHOOL        115             49.568966
1     EducationLevel.BACHELORS         68             29.310345
2  EducationLevel.NO_EDUCATION         23              9.913793
3           EducationLevel.PHD         15              6.465517
4       EducationLevel.MASTERS         11              4.741379

   Cumulative Frequency  Cumulative Percentage Frequency
```

```
0                  115                        49.568966
1                  183                        78.879310
2                  206                        88.793103
3                  221                        95.258621
4                  232                       100.000000

Percent frequency of Education Level for customers with age in (25, 35]
              education_level  Frequency  Percentage Frequency  \
0   EducationLevel.HIGH_SCHOOL        592             52.250662
1     EducationLevel.BACHELORS        256             22.594881
2       EducationLevel.MASTERS        118             10.414828
3  EducationLevel.NO_EDUCATION        113              9.973522
4           EducationLevel.PHD         54              4.766108

   Cumulative Frequency  Cumulative Percentage Frequency
0                   592                        52.250662
1                   848                        74.845543
2                   966                        85.260371
3                  1079                        95.233892
4                  1133                       100.000000

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual Income of
customers
count      1500.000000
mean      82688.042667
std       21549.553865
min       39520.000000
25%       67808.000000
50%       79040.000000
75%       94432.000000
max      147264.000000
Name: annual_income, dtype: float64

Frequency table for Number of accounts of the customers
   number_of_accounts  Frequency  Percentage Frequency  Cumulative Frequency  \
0                   3        435             29.000000                   435
1                   4        350             23.333333                   785
2                   2        350             23.333333                  1135
3                   5        180             12.000000                  1315
4                   1        131              8.733333                  1446
5                   6         54              3.600000                  1500

   Cumulative Percentage Frequency
0                        29.000000
1                        52.333333
2                        75.666667
3                        87.666667
4                        96.400000
5                       100.000000

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account Credit L
ine
count     1500.000000
mean     26425.984000
std      12968.276017
min       3993.600000
25%      16723.200000
50%      24710.400000
75%      34216.000000
```

```
max      85862.400000
Name: total_credit_line, dtype: float64
```

In [9]:

```
# Account report
import pandas

accounts_report_list = []
accounts_dict = get_accounts_for_customer_for_report()
for account in accounts_dict:
    accounts_report_list.append(account.__dict__)
accounts_dataframe = pandas.DataFrame(accounts_report_list)

# 1. The minimum and maximum for the Date Opened across the entire cohort
print(f'The minimum date opened across all accounts {accounts_dataframe.date_ope
print(f'The maximum date opened across all accounts {accounts_dataframe.date_ope

# 2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Age of the
accounts_dataframe.date_opened = pandas.to_datetime(accounts_dataframe.date_open
accounts_dataframe['age_of_account'] = (
            (pandas.to_datetime('2022-01-01') - accounts_dataframe.date_opened).
age_of_account_statistics = accounts_dataframe.age_of_account.describe()
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Age of t
print(age_of_account_statistics)


def get_age_for_customer_for_account(customer_dataframe, customer_id):
    return customer_dataframe.age[customer_dataframe.index[customer_dataframe.cu
        int(customer_id) - 1000001]



# 3. Frequency table for Account Age Flag
customer_ages = []
for account in accounts_dataframe['customer_id']:
    temp_age = get_age_for_customer_for_account(customer_dataframe, account)
    customer_ages.append(temp_age)
accounts_dataframe['customer_age'] = customer_ages
accounts_dataframe['accounts_age_flag'] = accounts_dataframe.customer_age - acco
accounts_age_flag_freq = create_frequency_table('accounts_age_flag',accounts_dat
print()
print('Frequency table for Account Age Flag')
print(accounts_age_flag_freq)

# 4. The minimum and maximum Account Number in your results
print()
print(f'Minimum value of the account number field : {accounts_dataframe["account
print(f'Maximum value of the account number field : {accounts_dataframe["account

# 5. Frequency table for last digit of the Account Number
accounts_dataframe['account_last_digit'] = accounts_dataframe['account_number'].
last_digit_distribution = accounts_dataframe['account_last_digit'].value_counts(
last_digit_distribution_freq = create_frequency_table('account_last_digit',accou
print()
print('Frequency table for last digit of the Account Number')
print(last_digit_distribution_freq)

# 6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account Cr
account_credit_line_statistics = accounts_dataframe.account_credit_line.describe
print()
```

```
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account
print(account_credit_line_statistics)

# 7. Frequency table for Account Credit Line Flag
accounts_dataframe['account_credit_line_flag'] = accounts_dataframe.account_cred
    'customer_id').account_credit_line.transform('sum')
account_credit_line_flag_freq = create_frequency_table('account_credit_line_flag
print()
print('Frequency table for Account Credit Line Flag')
print(account_credit_line_flag_freq)

# 8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual Fee
annual_fee_freq = accounts_dataframe.annual_fee.describe()
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual F
print(annual_fee_freq)

# 9. Frequency table for Annual Fee Flag
accounts_dataframe['annual_fee_flag'] = accounts_dataframe.annual_fee == account
# annual_fee_flag_freq = accounts_dataframe.annual_fee_flag.value_counts()
annual_fee_flag_freq = create_frequency_table('annual_fee_flag',accounts_datafra
print()
print('Frequency table for Annual Fee Flag')
print(annual_fee_flag_freq)

# 10. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual In
annual_interest_rate_freq = accounts_dataframe.annual_interest_rate.describe()
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual I
print(annual_interest_rate_freq)
```

```
The minimum date opened across all accounts 1962-01-20
The maximum date opened across all accounts 2021-12-28

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Age of the accou
nts
count    4760.000000
mean       16.098739
std        13.449270
min         0.000000
25%         5.000000
50%        13.000000
75%        25.000000
max        59.000000
Name: age_of_account, dtype: float64

Frequency table for Account Age Flag
   accounts_age_flag  Frequency  Percentage Frequency  Cumulative Frequency  \
0               True       4757             99.936975                  4757
1              False          3              0.063025                  4760

   Cumulative Percentage Frequency
0                        99.936975
1                       100.000000

Minimum value of the account number field : 10000011
Maximum value of the account number field : 10015003

Frequency table for last digit of the Account Number
  account_last_digit  Frequency  Percentage Frequency  Cumulative Frequency  \
0                  1       1500             31.512605                  1500
1                  2       1369             28.760504                  2869
2                  3       1019             21.407563                  3888
3                  4        584             12.268908                  4472
4                  5        234              4.915966                  4706
5                  6         54              1.134454                  4760

   Cumulative Percentage Frequency
0                        31.512605
1                        60.273109
2                        81.680672
3                        93.949580
4                        98.865546
5                       100.000000

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Account Credit L
ine
count     4760.000000
mean      8327.515943
std       8772.926133
min          0.460000
25%       1688.952500
50%       5623.625000
75%      11972.142500
max      61837.460000
Name: account_credit_line, dtype: float64

Frequency table for Account Credit Line Flag
   account_credit_line_flag  Frequency  Percentage Frequency  \
0                      True       1422                  94.8
1                     False         78                   5.2
```

```
   Cumulative Frequency  Cumulative Percentage Frequency
0                  1422                            94.8
1                  1500                           100.0

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual Fee
count    4760.000000
mean       83.275172
std        87.729295
min         0.000000
25%        16.890000
50%        56.235000
75%       119.722500
max       618.370000
Name: annual_fee, dtype: float64

Frequency table for Annual Fee Flag
   annual_fee_flag  Frequency  Percentage Frequency  Cumulative Frequency  \
0            False       4685              98.42437                  4685
1             True         75               1.57563                  4760

   Cumulative Percentage Frequency
0                         98.42437
1                        100.00000

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Annual Interest
Rate
count    4760.000000
mean       22.413158
std         4.344592
min        15.010000
25%        18.590000
50%        22.420000
75%        26.200000
max        30.000000
Name: annual_interest_rate, dtype: float64
```

In [10]:

```
# C. Account Activity Report

# Load the activity data
transaction_dataframe = pandas.read_csv('data/transactions_data.csv')

# 1. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number o
print(transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['PUR
    'Account Number')['Transaction Type'].value_counts().describe())

# 2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number o
print(transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['PUR
          'Transaction Type'].value_counts().describe())

# 3. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number o
print(transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['CAS
          'Transaction Type'].value_counts().describe())
```

```
# 4. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Purcha
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Purc
print(transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['PUR
          'Transaction Amount'].describe())

# 5. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Cash A
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Cash
print(transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['CAS
          'Transaction Amount'].describe())

# 6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Paymen
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Paym
print(
    transaction_dataframe[transaction_dataframe['Transaction Type'].isin(['PAYME

# 7. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Closin
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Clos
print(transaction_dataframe['Closing Balance'].describe())

monthly_report_dataframe = pandas.read_csv('data/monthly_reports_data.csv')

# 8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Minimu
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Mini
monthly_report_dataframe['minimum_amount_due'] = monthly_report_dataframe['Closi
print(monthly_report_dataframe.minimum_amount_due.describe())

# 9. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Tota
print(monthly_report_dataframe['Total Purchases'].describe())

# 10. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Tota
print(monthly_report_dataframe['Total Cash Advances'].describe())

# 11. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Payme
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Paym
print(monthly_report_dataframe['Total Payments'].describe())

# 12. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total
print()
print('Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Tota
print(monthly_report_dataframe['Total Interests Charged'].describe())
```

```
Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of Transa
ctions (Purchase or Cash Advance) for each card during the activity period
count    9456.000000
mean       44.363050
std        39.819081
min         1.000000
25%         4.000000
50%        71.000000
75%        84.000000
max       103.000000
Name: count, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of Purcha
ses
count    4760.000000
mean       83.722689
std         5.098936
min        67.000000
25%        80.000000
50%        84.000000
75%        87.000000
max       103.000000
Name: count, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for the Number of Cash A
dvances
count    4696.000000
mean        4.466993
std         2.013209
min         1.000000
25%         3.000000
50%         4.000000
75%         6.000000
max        17.000000
Name: count, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Purchase Amo
unts
count    398520.000000
mean       1578.149342
std        3521.657729
min           0.000000
25%           0.000000
50%         135.990000
75%        1414.105000
max       72121.210000
Name: Transaction Amount, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Cash Advance
Amounts
count    20977.000000
mean       257.466438
std        466.748016
min          0.000000
25%          0.000000
50%         44.510000
75%        310.630000
max       4580.260000
Name: Transaction Amount, dtype: float64
```

```
Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Payments Amo
unts
count    167426.000000
mean       3747.781107
std        5711.823462
min      -15338.490000
25%         287.712500
50%        1439.525000
75%        4768.947500
max       72510.610000
Name: Transaction Amount, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Closing Bala
nces
count    1.006420e+06
mean     6.040454e+03
std      7.714735e+03
min     -1.162080e+05
25%      5.875600e+02
50%      3.185385e+03
75%      8.600833e+03
max      7.614088e+04
Name: Closing Balance, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Minimum Amou
nts Due
count    57120.000000
mean       752.172722
std        849.785454
min      -1898.482000
25%        127.272500
50%        466.219500
75%       1069.763750
max       7614.088000
Name: minimum_amount_due, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total Purcha
se Amounts of the month
count     57120.000000
mean      11010.575512
std       16346.062140
min           0.000000
25%         507.892500
50%        4742.260000
75%       14753.882500
max      195180.130000
Name: Total Purchases, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total Cash A
dvance Amounts of the month
count    57120.000000
mean        94.553104
std        345.109848
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max       8826.370000
Name: Total Cash Advances, dtype: float64
```

```
Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Payment Amou
nts of the month
count     57120.000000
mean      10985.224046
std       17715.805707
min        -465.850000
25%           0.000000
50%        3373.455000
75%       14752.462500
max      241173.320000
Name: Total Payments, dtype: float64

Min, P25, Median, P75, Max, Mean, and Standard Deviation for all the Total Intere
sts of the month
count    57120.000000
mean       511.733897
std        618.016358
min      -1235.120000
25%         89.215000
50%        305.055000
75%        699.777500
max       9512.770000
Name: Total Interests Charged, dtype: float64
```