```
import random
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
import calendar
import logging
import time
from memory_profiler import memory_usage
import warnings

warnings.filterwarnings('ignore', message='No such comm target
registered')

logging.basicConfig(filename='app.log', level=logging.INFO,
                    format='%(asctime)s - %(name)s - %(levelname)s - %
(message)s')
logger = logging.getLogger(__name__)

#Generating Customer Data
def generate_customer():
    logger.info(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}-
Generating customer...")
    #Customer_ID, age, gender
    customer_id = random.randint(1000000, 9999999)
    age = random.randint(20, 80)
    gender = random.choice(['Male', 'Female'])
    age_categories = {
        (20, 30): [0.75, 0.25],
        (30, 60): [0.25, 0.75],
        (60, 80): [0.5, 0.5]
    }
    #Marital Status based on age
    for age_range, marital_probs in age_categories.items():
        if age_range[0] <= age <= age_range[1]:
            marital_status = random.choices(['Single', 'Married'],
weights=marital_probs)[0]
            break

    children_probs = {
        (20, 40): [0.4, 0.3, 0.2, 0.1, 0],
        (40, 80): [0.1, 0.3, 0.3, 0.2, 0.1]
    }

    #generating number of children based on age and children
probability
    num_children = random.choices(range(5),
weights=children_probs[(20, 40)] if age <= 40 else children_probs[(40,
80)])[0]

    education_probs = {
```

```
        (20, 25): [0.1, 0.5, 0.3, 0.1, 0],
        (25, 35): [0.1, 0.5, 0.3, 0.05, 0.05],
        (35, 80): [0.1, 0.5, 0.25, 0.1, 0.05]
    }

    #Education level numeric mapping
    for age_range, education_probs_list in education_probs.items():
        if age_range[0] <= age <= age_range[1]:
            education_level_mapping = {
                'No Education': 0,
                'High School': 1,
                "Bachelor's Degree": 2,
                "Master's Degree": 3,
                'Ph.D.': 4
            }
            education_level = random.choices(['No Education', 'High
School', "Bachelor's Degree", "Master's Degree", 'Ph.D.'],
                             weights=education_probs_list)[0]
            education_level_numeric =
education_level_mapping[education_level]
            break

    #Calculate Annual Income
    annual_income = 40 * 52 * ((15 + education_level_numeric * 10) +
(age / 10) * 2)

    #Define marital status numeric value
    marital_status_numeric = 1 if marital_status == 'Married' else 0

    #Generating Number of accounts
    num_accounts = marital_status_numeric + num_children + 1

    #For each customer, his acccounts are defined
    accounts = []

    #Calculate total credit line
    total_credit_line = num_accounts * (annual_income / 10)

    #Generate account info for each of the customer's account
    for i in range(num_accounts):
        account = generate_account_info(customer_id, i + 1, age,
total_credit_line,num_accounts)
        accounts.append(account)
        total_credit_line += account['Account_Credit_Line']

    logger.info("Customer generation completed.")
    return {
        'Customer_ID': customer_id,
        'Age': age,
        'Gender': gender,
```

```
        'Marital_Status': marital_status,
        'Number_of_Children': num_children,
        'Education_Level': education_level,
        'Annual_Income': annual_income,
        'Number_of_Accounts': num_accounts,
        'Total_Credit_Line': total_credit_line,
        'Accounts': accounts
    }

#Generating Account data for each customer
def generate_account_info(customer_id, account_number, age,
total_credit_line,num_accounts):
    logger.info(
        f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}-Generating
account info for Customer ID: {customer_id}, Account Number:
{account_number}...")
    max_date = min(datetime.now() - timedelta(days=365 * (age - 20)),
datetime(2022, 1, 1))
    date_opened = datetime(random.randint(max_date.year, 2022),
random.randint(1, 12), random.randint(1, 28))
    #Date opened should be before Jan 2022 and the customer should be
20 years old before he can open any account...

    #Generate AccountID by adding a number to the end of customer_id
    account_id = int(str(customer_id) + str(account_number))

    #Generate Account Credit Line, Annual Fee, Annual Interest Rate
    account_credit_line = random.uniform(0.1, 1.0) *
(total_credit_line)

    annual_fee = account_credit_line * 0.01
    annual_interest_rate = random.uniform(0.15, 0.3)
    #credit line for each account of the customer
    num_accounts = 0
    total_credit_line_of_accounts = 0
    monthly_details = []

    logger.info("Account info generation completed.")
    return {
        'Account_ID': account_id,
        'Date_Opened': date_opened,
        'Account_Credit_Line': account_credit_line,
        'Annual_Fee': annual_fee,
        'Annual_Interest_Rate': annual_interest_rate,
        'Number_of_Accounts': num_accounts,
        'Total_Credit_Line_of_Accounts':
total_credit_line_of_accounts,
        'Account_Status': 'open',
        'Monthly_Details': monthly_details
    }
```

```
#simulate account activity for each account and store monthly
transactions data into account_activity_data.csv file
def simulate_account_activity(account):
    logger.info(
        f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}-Simulating
account activity for Account ID: {account['Account_ID']}...")
    total_available_credit = account['Account_Credit_Line']  #
Starting balance
    daily_interest_rate = account['Annual_Interest_Rate'] / 365
    credit_line = total_available_credit

    total_purchases_month = 0
    total_cash_advances_month = 0
    total_payments_month = 0
    total_interest_charged_month = 0
    current_balance = 0
    purchase_amount = 0

    # Delinquency variables
    past_due_amount = 0
    late_payment_fee = 30
    delinquency_counter = 0
    paymentMissed = False

    for month in range(1, 13):
        closing_balance = current_balance
        total_purchases_month = 0
        total_cash_advances_month = 0
        total_payments_month = 0
        total_interest_charged_month = 0
        current_balance = 0
        purchase_amount = 0
        total_payments_month = 0
        interest_charged = 0
        transaction_values = []
        transaction_Purchase_Counter = 0
        transaction_Cashout_Counter = 0
        total_transaction_amount = 0
        d = 0
        k = 0
        purchases_monthly = []
        cash_advances_monthly = []
        payments_monthly = []
        #purchases_monthly, cash_advances_monthly and payments_monthly
are arrays containing the individual transaction amounts made
        num_days_in_month = calendar.monthrange(2022, month)[1]

        for day in range(1, num_days_in_month):
            # Purchase or Cash Advance
```

```
            d = random.randint(1, 7)   #d is transaction made in
random day from 1 to 7
            #k is an integer value, k is added the value of d. such
that it limits the days of transactions stays under total days in
month
            k = k + d
            if k <= num_days_in_month and delinquency_counter < 3:
                chosen_event = random.choices(['Make a Purchase',
'Take Cash Out'], weights=[0.95, 0.05], k=1)[0]
                if chosen_event == 'Make a Purchase' and
total_available_credit > 0 and delinquency_counter < 3:
                    # Make a purchase
                    purchase_amount = random.uniform(0,
total_available_credit)
                    total_available_credit = credit_line -
purchase_amount
                    current_balance += purchase_amount
                    interest_charged = interest_charged +
(current_balance * daily_interest_rate * (day - 1))
                    total_purchases_month += purchase_amount
                    transaction_Purchase_Counter =
transaction_Purchase_Counter + 1
                    purchases_monthly.append(purchase_amount)

                elif chosen_event == 'Take Cash Out' and
total_available_credit > 0 and delinquency_counter < 3 and k <=
num_days_in_month:
                    # Take cash advance
                    available_cash = min(total_available_credit, 0.1 *
account['Account_Credit_Line'])
                    cash_advance_amount = random.uniform(0,
available_cash)
                    current_balance += cash_advance_amount
                    interest_charged = interest_charged +
(current_balance * daily_interest_rate * (day - 1))
                    total_cash_advances_month += cash_advance_amount
                    total_available_credit = credit_line -
cash_advance_amount
                    transaction_Cashout_Counter =
transaction_Cashout_Counter + 1
                    cash_advances_monthly.append(cash_advance_amount)

            # Apply daily interest
            current_balance *= (1 + daily_interest_rate)

            # Payment scenarios
            if random.randint(1, 10) == 1 and k < num_days_in_month
and delinquency_counter < 3:
                individual_payments_record = current_balance
                total_payments_month = total_payments_month +
```

```
current_balance
                payments_monthly.append(individual_payments_record)
                # 10% pay the exact amount they spent every day
                current_balance = 0
                paymentMissed = False
            elif random.randint(1, 7) == 1 and k < num_days_in_month
and delinquency_counter < 3:
                individual_payments_record = current_balance
                total_payments_month = total_payments_month +
current_balance
                payments_monthly.append(individual_payments_record)
                # 30% pay entire balance every d days
                current_balance = 0
                paymentMissed = False
            elif random.randint(1, 7) == 1 and k < num_days_in_month
and delinquency_counter < 3:
                # 30% pay a proportion every d days
                days_to_pay = random.randint(1, 7)
                if day % days_to_pay == 0:
                    payment_proportion = random.uniform(0, 1)
                    payments_monthly.append(payment_proportion *
current_balance)
                    current_balance *= (1 - payment_proportion)
                    paymentMissed = False
            elif delinquency_counter < 3 and k < num_days_in_month:
                # 30% follow payment period scenarios
                payment_period = random.randint(1, 10)
                if day <= payment_period and k < 32:
                    if random.random() < 0.1:
                        # 10% pay entire balance within the Payment
Period
                        individual_payments_record = current_balance
                        total_payments_month += current_balance

payments_monthly.append(individual_payments_record)
                        current_balance = 0
                        paymentMissed = False
                    elif random.random() < 0.15:
                        # 15% pay only the Minimum Amount Due within
the Payment Period
                        minimum_amount_due = 0.1 * current_balance
                        total_payments_month += minimum_amount_due
                        current_balance -= minimum_amount_due
                        payments_monthly.append(minimum_amount_due)
                        paymentMissed = False
                    else:
                        # miss payment
                        paymentMissed = True
```

```
        if day > 10 and paymentMissed == True and delinquency_counter
< 3:
            past_due_amount = 0.1 * current_balance
            current_balance += past_due_amount + late_payment_fee
            delinquency_counter += 1

        # Calculate end-of-month metrics
        closing_balance = current_balance
        minimum_amount_due = 0.1 * closing_balance
        total_interest_charged_month += interest_charged
        total_transaction_amount = total_purchases_month +
total_cash_advances_month

        account.setdefault('Monthly_Details', []).append({
            'Month': f"{calendar.month_name[month]} 2022",
            'Closing_Balance': closing_balance,
            'Minimum_Amount_Due': minimum_amount_due,
            'Total_Purchases': total_purchases_month,
            'Total_Cash_Advances': total_cash_advances_month,
            'Total_Payments': total_payments_month,
            'Total_Interest_Charged': total_interest_charged_month,
            'Total_Transaction_amount_in_month':
total_transaction_amount,
            'Delinquency_Count': delinquency_counter,
            'Purchase_Transactions_of_month':
transaction_Purchase_Counter,
            'Cash_advance_Transactions_of_month':
transaction_Cashout_Counter,
            'Total_Transactions_of_month':
transaction_Purchase_Counter + transaction_Cashout_Counter,
            'Total_Individual_Payments_of_month': payments_monthly,
            'Total_Individual_purchases_of_month': purchases_monthly,
            'Total_Individual_Cash_Advances_of_month':
cash_advances_monthly
        })

        if delinquency_counter >= 3:
            logger.info(
                f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}-
Account closed due to 3 consecutive missed payments. Account ID:
{account['Account_ID']}")
            break

        # Reset monthly metrics
        total_purchases_month = 0
        total_cash_advances_month = 0
        total_payments_month = 0
        total_interest_charged_month = 0
        # print(delinquency_counter,'delinquency')
```

```
    logger.info(
        f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}-Account
activity simulation completed for Account ID:
{account['Account_ID']}.")


if __name__ == "__main__":
    start_time = time.time()

    # Generate 20,000 customers
    customers = [generate_customer() for _ in range(20000)]

    # Simulate account activity for each customer's accounts
    for customer in customers:
        for account in customer['Accounts']:
            simulate_account_activity(account)

    # Convert the data to dataframes
    customers_df = pd.DataFrame(customers)
    customers_df.drop('Accounts', axis=1, inplace=True)
    customers_df.to_csv('customers_data.csv', index=False)

    accounts_data = []
    for customer in customers:
        for account in customer['Accounts']:
            account_status = account.get('Account_Status', 'Open')  #
'Account_Status' default set to 'Open'
            accounts_data.append({
                'Customer_ID': customer['Customer_ID'],
                **account  # Unpack account details
            })

    accounts_df = pd.DataFrame(accounts_data)
    accounts_df.drop(['Number_of_Accounts',
'Total_Credit_Line_of_Accounts', 'Monthly_Details'], axis=1,
inplace=True)
    accounts_df.to_csv('accounts_data.csv', index=False)

    account_activity_data = []
    for customer in customers:
        for account in customer['Accounts']:
            for monthly_detail in account.get('Monthly_Details', []):
                account_activity_data.append({
                    'Customer_ID': customer['Customer_ID'],
                    'Account_ID': account['Account_ID'],
                    **monthly_detail
                })

    account_activity_df = pd.DataFrame(account_activity_data)
    account_activity_df.to_csv('account_activity_data.csv',
```

```
index=False)

    end_time = time.time()
    elapsed_time = end_time - start_time

    logging.info(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} -
Main - Execution completed in {elapsed_time:.2f} seconds.")
    logging.info(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} -
Main - Total memory used: {memory_usage()[0]} MB")

    # Sample results in Jupyter Notebook
    print("Sample results:")
    print(customers_df.head())
    print(accounts_df.head())
    print(account_activity_df.head())
    #logs are to be printed at the end



#Following code is to calculate and display results of Functional
Testing

# Customer Data
#Cohort information

# Load the customers data from the CSV file
file_path = 'customers_data.csv'
data = pd.read_csv(file_path)

# 1. Number of customers
num_customers = len(data)

# 2. Minimum and Maximum Customer ID
min_customer_id = data['Customer_ID'].min()
max_customer_id = data['Customer_ID'].max()

# 3. Number of unique Customer IDs
unique_customer_ids = data['Customer_ID'].nunique()

# 4. Descriptive statistics for the age of customers
age_stats = data['Age'].describe(percentiles=[.25, .5, .75])

# 5. Frequency table for Gender
gender_freq_table = data['Gender'].value_counts()
gender_percent_freq_table =
data['Gender'].value_counts(normalize=True)
gender_cumulative_freq_table = gender_freq_table.cumsum()
gender_cumulative_percent_freq_table =
gender_percent_freq_table.cumsum()

# 6. Frequency table for Marital Status
```

```
marital_status_freq_table = data['Marital_Status'].value_counts()
marital_status_percent_freq_table =
data['Marital_Status'].value_counts(normalize=True)
marital_status_cumulative_freq_table =
marital_status_freq_table.cumsum()
marital_status_cumulative_percent_freq_table =
marital_status_percent_freq_table.cumsum()

# 7. Percent frequency of Marital Status for specific age categories
age_bins = [20, 30, 60, 80]
data['Age_Category'] = pd.cut(data['Age'], bins=age_bins, labels=['20-
30', '30-60', '60-80'])

# 8. Frequency table for Number of Children
children_freq_table = data['Number_of_Children'].value_counts()
children_percent_freq_table =
data['Number_of_Children'].value_counts(normalize=True)
children_cumulative_freq_table = children_freq_table.cumsum()
children_cumulative_percent_freq_table =
children_percent_freq_table.cumsum()

# 9. Percent frequency of Number of Children for specific age
categories
age_bins_children = [20, 40, 80]
data['Age_Category_Children'] = pd.cut(data['Age'],
bins=age_bins_children, labels=['20-40', '40-80'])
children_age_category_percent_freq =
pd.crosstab(data['Age_Category_Children'], data['Number_of_Children'],
                                            normalize='index')
children_age_category_cross_tab =
pd.crosstab(data['Age_Category_Children'], data['Number_of_Children'],
                                               normalize='index')

# 10. Frequency table for Education Level
education_freq_table = data['Education_Level'].value_counts()
education_percent_freq_table =
data['Education_Level'].value_counts(normalize=True)
education_cumulative_freq_table = education_freq_table.cumsum()
education_cumulative_percent_freq_table =
education_percent_freq_table.cumsum()

# 11. Percent frequency of Education Level for specific age categories
age_bins_education = [20, 25, 35, 80]
data['Age_Category_Education'] = pd.cut(data['Age'],
bins=age_bins_education, labels=['20-25', '25-35', '35-80'])
education_age_category_percent_freq =
pd.crosstab(data['Age_Category_Education'], data['Education_Level'],
                                             normalize='index')
education_age_category_cross_tab =
pd.crosstab(data['Age_Category_Education'], data['Education_Level'],
```

```
                                                normalize='index')

# 12. Descriptive statistics for Annual Income
annual_income_stats = data['Annual_Income'].describe(percentiles=[.25,
.5, .75])

# 13. Frequency table for Number of Accounts
num_accounts_freq_table = data['Number_of_Accounts'].value_counts()
num_accounts_percent_freq_table =
data['Number_of_Accounts'].value_counts(normalize=True)
num_accounts_cumulative_freq_table = num_accounts_freq_table.cumsum()
num_accounts_cumulative_percent_freq_table =
num_accounts_percent_freq_table.cumsum()

# 14. Descriptive statistics for Total Credit Line
total_credit_line_stats =
data['Total_Credit_Line'].describe(percentiles=[.25, .5, .75])
total_credit_line_stats =
total_credit_line_stats.to_frame().transpose()

# Create DataFrames for results
results = pd.DataFrame({
    'Number of customers': [num_customers],
    'Minimum Customer ID': [min_customer_id],
    'Maximum Customer ID': [max_customer_id],
    'Number of unique Customer IDs': [unique_customer_ids],
})

results_age = age_stats.to_frame().transpose()
results_gender = pd.DataFrame({'Category': gender_freq_table.index,
                               'Frequency': gender_freq_table.values,
                               'Percent Frequency':
gender_percent_freq_table.values * 100,
                               'Cumulative Frequency':
gender_cumulative_freq_table.values,
                               'Cumulative Percent Frequency':
gender_cumulative_percent_freq_table.values * 100
                               })

marital_status_table = pd.DataFrame({
    'Category': marital_status_freq_table.index,
    'Frequency': marital_status_freq_table.values,
    'Percent Frequency': marital_status_percent_freq_table.values *
100,
    'Cumulative Frequency':
marital_status_cumulative_freq_table.values,
    'Cumulative Percent Frequency':
marital_status_cumulative_percent_freq_table.values * 100
})
marital_status_age_category_percent_freq =
```

```
pd.crosstab(data['Age_Category'], data['Marital_Status'],
normalize='index')

children_table = pd.DataFrame({
    'Category': children_freq_table.index,
    'Frequency': children_freq_table.values,
    'Percent Frequency': children_percent_freq_table.values * 100,
    'Cumulative Frequency': children_cumulative_freq_table.values,
    'Cumulative Percent Frequency':
children_cumulative_percent_freq_table.values * 100
})

education_table = pd.DataFrame({
    'Category': education_freq_table.index,
    'Frequency': education_freq_table.values,
    'Percent Frequency': education_percent_freq_table.values * 100,
    'Cumulative Frequency': education_cumulative_freq_table.values,
    'Cumulative Percent Frequency':
education_cumulative_percent_freq_table.values * 100
})

accounts_table = pd.DataFrame({
    'Category': num_accounts_freq_table.index,
    'Frequency': num_accounts_percent_freq_table.values,
    'Percent Frequency': num_accounts_percent_freq_table.values * 100,
    'Cumulative Frequency': num_accounts_cumulative_freq_table.values,
    'Cumulative Percent Frequency':
num_accounts_cumulative_percent_freq_table.values * 100
})

#Also printing them to console
print('\n1-3. Customer Results: min, max and unique customers:')
display(results)
print('\n4. Age Results')
display(results_age)
print('\n5. Gender')
display(results_gender)
print('\n6. Marital Status')
display(marital_status_table)
print('\n7. Marital Status for specific age:')
print(marital_status_age_category_percent_freq)
print('\n8. Frequency table for children')
print(children_table)
print('\n9. Frequency of Number of Children')
print(children_age_category_cross_tab)
print('\n10. Frequency of Education Level')
print(education_table)
print('\n11. Education level for ages')
print(education_age_category_cross_tab)
print('\n12. Annual Income calculations')
```

```
print(annual_income_stats)
print('\n13. Accounts Table')
print(accounts_table)
print('\n14. Credit line Results')
print(total_credit_line_stats)




# B. Accounts information
# Load the data from the CSV file
file_path = 'accounts_data.csv'
accounts_data = pd.read_csv(file_path)

# 1. Min and Max Date Opened
min_date_opened = accounts_data['Date_Opened'].min()
max_date_opened = accounts_data['Date_Opened'].max()

# 2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for Age
of the accounts (in years)
accounts_data['Date_Opened'] =
pd.to_datetime(accounts_data['Date_Opened'])
accounts_data['Account_Age'] = (pd.to_datetime('2022-01-01') -
accounts_data['Date_Opened']).dt.days // 365

age_stats = accounts_data['Account_Age'].describe(percentiles=[.25,
.5, .75])

# 3. Frequency table for Account Age Flag
accounts_data['Account_Age_Flag'] = accounts_data['Account_Age'] >= 20
account_age_flag_freq_table =
accounts_data['Account_Age_Flag'].value_counts()
account_age_flag_percent_freq_table =
accounts_data['Account_Age_Flag'].value_counts(normalize=True)
account_age_flag_cumulative_freq_table =
account_age_flag_freq_table.cumsum()
account_age_flag_cumulative_percent_freq_table =
account_age_flag_percent_freq_table.cumsum()

# 4. Min and Max Account Number
min_account_number = accounts_data['Account_ID'].min()
max_account_number = accounts_data['Account_ID'].max()

# 5. Frequency table for last digit of the Account Number
accounts_data['Last_Digit_Account_Number'] =
accounts_data['Account_ID'] % 10
last_digit_freq_table =
accounts_data['Last_Digit_Account_Number'].value_counts()
last_digit_percent_freq_table =
```

```
accounts_data['Last_Digit_Account_Number'].value_counts(normalize=True
)
last_digit_cumulative_freq_table = last_digit_freq_table.cumsum()
last_digit_cumulative_percent_freq_table =
last_digit_percent_freq_table.cumsum()

# 6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for
Account Credit Line
credit_line_stats =
accounts_data['Account_Credit_Line'].describe(percentiles=[.25, .5, .7
5])

# 7. Frequency table for Account Credit Line Flag
accounts_data['Account_Credit_Line_Flag'] =
accounts_data['Account_Credit_Line'] == accounts_data[
    'Account_Credit_Line'].sum()
credit_line_flag_freq_table =
accounts_data['Account_Credit_Line_Flag'].value_counts()
credit_line_flag_percent_freq_table =
accounts_data['Account_Credit_Line_Flag'].value_counts(normalize=True)
credit_line_flag_cumulative_freq_table =
credit_line_flag_freq_table.cumsum()
credit_line_flag_cumulative_percent_freq_table =
credit_line_flag_percent_freq_table.cumsum()

# 8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for
Annual Fee
annual_fee_stats =
accounts_data['Annual_Fee'].describe(percentiles=[.25, .5, .75])

# 9. Frequency table for Annual Fee Flag
accounts_data['Annual_Fee_Flag'] = accounts_data['Annual_Fee'] ==
accounts_data['Account_Credit_Line'] * 0.01
annual_fee_flag_freq_table =
accounts_data['Annual_Fee_Flag'].value_counts()
annual_fee_flag_percent_freq_table =
accounts_data['Annual_Fee_Flag'].value_counts(normalize=True)
annual_fee_flag_cumulative_freq_table =
annual_fee_flag_freq_table.cumsum()
annual_fee_flag_cumulative_percent_freq_table =
annual_fee_flag_percent_freq_table.cumsum()

# 10. Min, P25, Median, P75, Max, Mean, and Standard Deviation for
Annual Interest Rate
interest_rate_stats =
accounts_data['Annual_Interest_Rate'].describe(percentiles=[.25, .5, .
75])


# Display the results
```

```
print("\n1. Min and Max Date Opened:")
print(f"Min Date Opened: {min_date_opened}")
print(f"Max Date Opened: {max_date_opened}")

print("\n2. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for Age of the accounts (in years):")
print(age_stats)

print("\n3. Frequency table for Account Age Flag:")
print(account_age_flag_freq_table)
print(account_age_flag_percent_freq_table)
print(account_age_flag_cumulative_freq_table)
print(account_age_flag_cumulative_percent_freq_table)

print("\n4. Min and Max Account Number:")
print(f"Min Account Number: {min_account_number}")
print(f"Max Account Number: {max_account_number}")

print("\n5. Frequency table for last digit of the Account Number:")
print(last_digit_freq_table)
print(last_digit_percent_freq_table)
print(last_digit_cumulative_freq_table)
print(last_digit_cumulative_percent_freq_table)

print("\n6. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for Account Credit Line:")
print(credit_line_stats)

print("\n7. Frequency table for Account Credit Line Flag:")
print(credit_line_flag_freq_table)
print(credit_line_flag_percent_freq_table)
print(credit_line_flag_cumulative_freq_table)
print(credit_line_flag_cumulative_percent_freq_table)

print("\n8. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for Annual Fee:")
print(annual_fee_stats)

print("\n9. Frequency table for Annual Fee Flag:")
print(annual_fee_flag_freq_table)
print(annual_fee_flag_percent_freq_table)
print(annual_fee_flag_cumulative_freq_table)
print(annual_fee_flag_cumulative_percent_freq_table)

print("\n10. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for Annual Interest Rate:")
print(interest_rate_stats)
```

```
# C. Account Activity Information
# Load the data from the CSV file
file_path = 'account_activity_data.csv'
activity_data = pd.read_csv(file_path)

# 1. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Transactions
transaction_stats =
activity_data['Total_Transactions_of_month'].describe(percentiles=[.25
, .5, .75])

# 2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Purchases
purchase_stats =
activity_data['Purchase_Transactions_of_month'].describe(percentiles=[
.25, .5, .75])

# 3. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Cash Advances
cash_advance_stats =
activity_data['Cash_advance_Transactions_of_month'].describe(percentil
es=[.25, .5, .75])

# 4. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Purchase Amounts
purchase_amount_stats =
activity_data['Total_Purchases'].describe(percentiles=[.25, .5, .75])

# 5. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Cash Advance Amounts
cash_advance_amount_stats =
activity_data['Total_Cash_Advances'].describe(percentiles=[.25, .5, .7
5])

# 6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Payments Amounts
payment_amount_stats =
activity_data['Total_Payments'].describe(percentiles=[.25, .5, .75])

# 7. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Closing Balances
closing_balance_stats =
activity_data['Closing_Balance'].describe(percentiles=[.25, .5, .75])

# 8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Minimum Amounts Due
min_amount_due_stats =
activity_data['Minimum_Amount_Due'].describe(percentiles=[.25, .5, .75
])
```

```
# Convert the string representation of arrays to actual arrays
activity_data['Total_Individual_Payments_of_month'] =
activity_data['Total_Individual_Payments_of_month'].apply(eval)
activity_data['Total_Individual_Purchases_of_month'] =
activity_data['Total_Individual_purchases_of_month'].apply(eval)
activity_data['Total_Individual_Cash_Advances_of_month'] =
activity_data[
    'Total_Individual_Cash_Advances_of_month'].apply(eval)


def calculate_stats(array):
    if len(array) == 0:
        return [np.nan] * 7  # Return NaN values for empty arrays

    stats = np.percentile(array, [0, 25, 50, 75, 100])  # Percentiles:
0, 25, 50 (median), 75, 100
    stats = np.append(stats, [np.mean(array), np.std(array)])  # Mean
and Standard Deviation
    return stats


# Calculate statistics for each array column
payment_stats =
activity_data['Total_Individual_Payments_of_month'].apply(calculate_st
ats).apply(pd.Series)
purchase_stats =
activity_data['Total_Individual_Purchases_of_month'].apply(calculate_s
tats).apply(pd.Series)
cash_advance_stats =
activity_data['Total_Individual_Cash_Advances_of_month'].apply(calcula
te_stats).apply(pd.Series)

# Rename the columns for better clarity
payment_stats.columns = ['Min_Payment', '25th_Percentile_Payment',
'Median_Payment', '75th_Percentile_Payment',
                         'Max_Payment', 'Mean_Payment',
'Std_Dev_Payment']
purchase_stats.columns = ['Min_Purchase', '25th_Percentile_Purchase',
'Median_Purchase', '75th_Percentile_Purchase',
                          'Max_Purchase', 'Mean_Purchase',
'Std_Dev_Purchase']
cash_advance_stats.columns = ['Min_Cash_Advance',
'25th_Percentile_Cash_Advance', 'Median_Cash_Advance',
                              '75th_Percentile_Cash_Advance',
'Max_Cash_Advance', 'Mean_Cash_Advance',
                              'Std_Dev_Cash_Advance']

# 12. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Total Interests of the month
total_interest_stats =
```

```
activity_data['Total_Interest_Charged'].describe(percentiles=[.25, .5,
.75])

# 13. Frequency table for Delinquency Counter
delinquency_counter_freq_table =
activity_data['Delinquency_Count'].value_counts()
delinquency_counter_percent_freq_table =
activity_data['Delinquency_Count'].value_counts(normalize=True)
delinquency_counter_cumulative_freq_table =
delinquency_counter_freq_table.cumsum()
delinquency_counter_cumulative_percent_freq_table =
delinquency_counter_percent_freq_table.cumsum()

# 14. Frequency table for Annual Active Flag
activity_data['Annual_Active_Flag'] =
activity_data['Delinquency_Count'] < 3
annual_active_flag_freq_table =
activity_data['Annual_Active_Flag'].value_counts()
annual_active_flag_percent_freq_table =
activity_data['Annual_Active_Flag'].value_counts(normalize=True)
annual_active_flag_cumulative_freq_table =
annual_active_flag_freq_table.cumsum()
annual_active_flag_cumulative_percent_freq_table =
annual_active_flag_percent_freq_table.cumsum()

# Display the results of Min, P25, Median, P75, Max, Mean, and
Standard Deviations of calculated data
print("\n1. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for the Number of Transactions:")
print(transaction_stats)

print("\n2. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for the Number of Purchases:")
print(purchase_stats)

print("\n3. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for the Number of Cash Advances:")
print(cash_advance_stats)

print("\n4. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for all the Purchase Amounts:")
print(purchase_amount_stats.to_frame().transpose())

print("\n5. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for all the Cash Advance Amounts:")
print(cash_advance_amount_stats)

print("\n6. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for all the Payments Amounts:")
print(payment_amount_stats.to_frame().transpose())
```

```
print("\n7. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for all the Closing Balances:")
print(closing_balance_stats.to_frame().transpose())

print("\n8. Min, P25, Median, P75, Max, Mean, and Standard Deviation
for all the Minimum Amounts Due:")
print(min_amount_due_stats)

print(
    "\n9. Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Payment Amounts of the month:")
print(payment_stats.head())

print(
    "\n10.Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Total Purchase Amounts of the month:")
print(purchase_stats.head())

print("\n11.Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Total Cash Advance Amounts of the month:")
print(cash_advance_stats.head())

print("\n13. Frequency table for Delinquency Counter:")
print(delinquency_counter_freq_table)
print(delinquency_counter_percent_freq_table)
print(delinquency_counter_cumulative_freq_table)
print(delinquency_counter_cumulative_percent_freq_table)

print("\n14. Frequency table for Annual Active Flag:")
print(annual_active_flag_freq_table)
print(annual_active_flag_percent_freq_table)
print(annual_active_flag_cumulative_freq_table)
print(annual_active_flag_cumulative_percent_freq_table)

#Printing the logs into the console
#sometimes the data rate exceeds and logs are not printed to console
but are stored in log file
print("Logs:")
with open('app.log', 'r') as log_file:
    print(log_file.read())
```

```
Sample results:
   Customer_ID  Age  Gender Marital_Status  Number_of_Children  \
0      5061525   52    Male        Married                   3
1      1826965   34    Male         Single                   2
2      3147168   33    Male         Single                   0
3      4980124   75  Female         Single                   1
4      4868155   76    Male         Single                   2
```

```
     Education_Level  Annual_Income  Number_of_Accounts
Total_Credit_Line
0        High School        73632.0                   5
260312.772869
1              Ph.D.       128544.0                   3
119482.191085
2  Bachelor's Degree        86528.0                   1
12931.487789
3    Master's Degree       124800.0                   2
67210.126783
4        High School        83616.0                   3
69387.064042
   Customer_ID  Account_ID Date_Opened  Account_Credit_Line
Annual_Fee  \
0      5061525    50615251  2009-08-16         12446.135223
124.461352
1      5061525    50615252  2016-06-05         32442.815229
324.428152
2      5061525    50615253  2003-09-01         32583.612755
325.836128
3      5061525    50615254  2000-05-20        111745.413004
1117.454130
4      5061525    50615255  1997-12-28         34278.796657
342.787967

   Annual_Interest_Rate Account_Status
0              0.294196           open
1              0.161480           open
2              0.266961           open
3              0.158613           open
4              0.293124           open
   Customer_ID  Account_ID          Month  Closing_Balance  \
0      5061525    50615251   January 2022         0.000000
1      5061525    50615251  February 2022     10141.130829
2      5061525    50615251     March 2022     20028.606552
3      5061525    50615251     April 2022      1174.758857
4      5061525    50615251       May 2022     22995.959369

   Minimum_Amount_Due  Total_Purchases  Total_Cash_Advances
Total_Payments  \
0            0.000000     25695.063787              0.000000
25730.957497
1         1014.113083     37453.362687              0.000000
27503.695956
2         2002.860655     18798.029667              0.000000
1016.369482
3          117.475886     24815.862242              0.000000
23029.692422
4         2299.595937     43970.399619            507.318914
```

```
21959.785867

   Total_Interest_Charged  Total_Transaction_amount_in_month  \
0               91.508918                       25695.063787
1              138.552131                       37453.362687
2              103.563177                       18798.029667
3              156.788066                       24815.862242
4              442.212713                       44477.718533

   Delinquency_Count  Purchase_Transactions_of_month  \
0                  0                               6
1                  0                               8
2                  1                               5
3                  1                               6
4                  1                               8

   Cash_advance_Transactions_of_month  Total_Transactions_of_month  \
0                                   0                            6
1                                   0                            8
2                                   0                            5
3                                   0                            6
4                                   1                            9

                   Total_Individual_Payments_of_month  \
0  [11243.356986130048, 6430.6940657631385, 8056....
1  [8398.189430013848, 8120.070218803819, 8519.85...
2            [238.34343622631854, 778.0260460208698]
3            [689.5784601455291, 23029.692422416734]
4                               [21959.78586657235]

                  Total_Individual_purchases_of_month  \
0  [4239.269894369461, 4689.229753377601, 2295.18...
1  [4236.73185983685, 4151.279056439217, 3435.139...
2  [2381.514820080194, 2712.6933177427, 2912.2947...
3  [2239.9725174326186, 9833.853425109888, 566.64...
4  [1590.3615007100816, 10153.275890581639, 1540....

  Total_Individual_Cash_Advances_of_month
0                                      []
1                                      []
2                                      []
3                                      []
4                      [507.31891403905286]

1-3. Customer Results: min, max and unique customers:

   Number of customers  Minimum Customer ID  Maximum Customer ID  \
0                20000              1001695              9999711
```

```
   Number of unique Customer IDs
0                          19982


4. Age Results

       count      mean        std   min   25%   50%   75%   max
Age  20000.0  50.03035  17.604713  20.0  35.0  50.0  65.0  80.0


5. Gender

  Category  Frequency  Percent Frequency  Cumulative Frequency  \
0     Male      10080               50.4                 10080
1   Female       9920               49.6                 20000

   Cumulative Percent Frequency
0                          50.4
1                         100.0


6. Marital Status

  Category  Frequency  Percent Frequency  Cumulative Frequency  \
0  Married      11574              57.87                 11574
1   Single       8426              42.13                 20000

   Cumulative Percent Frequency
0                         57.87
1                        100.00


7. Marital Status for specific age:
Marital_Status   Married    Single
Age_Category
20-30           0.242481  0.757519
30-60           0.747542  0.252458
60-80           0.508694  0.491306

8. Frequency table for children
   Category  Frequency  Percent Frequency  Cumulative Frequency  \
0         1       6012             30.060                  6012
1         2       5327             26.635                 11339
2         0       4010             20.050                 15349
3         3       3298             16.490                 18647
4         4       1353              6.765                 20000

   Cumulative Percent Frequency
0                        30.060
1                        56.695
2                        76.745
3                        93.235
```

```
4                        100.000

9. Frequency of Number of Children
Number_of_Children            0         1         2         3
4
Age_Category_Children

20-40                    0.393995  0.304850  0.205543  0.095612
0.000000
40-80                    0.100053  0.298183  0.297879  0.201019
0.102866

10. Frequency of Education Level
            Category  Frequency  Percent Frequency  Cumulative
Frequency  \
0        High School      10138              50.690
10138
1  Bachelor's Degree       5145              25.725
15283
2       No Education       2007              10.035
17290
3    Master's Degree       1830               9.150
19120
4              Ph.D.         880               4.400
20000

   Cumulative Percent Frequency
0                         50.690
1                         76.415
2                         86.450
3                         95.600
4                        100.000

11. Education level for ages
Education_Level          Bachelor's Degree  High School  Master's
Degree  \
Age_Category_Education

20-25                                0.290221     0.499054
0.108517
25-35                                0.285757     0.508714
0.057392
35-80                                0.246759     0.506956
0.096980

Education_Level          No Education     Ph.D.
Age_Category_Education
20-25                        0.102208  0.000000
25-35                        0.099760  0.048377
35-80                        0.100509  0.048795
```

```
12. Annual Income calculations
count     20000.000000
mean      82628.145600
std       21169.285873
min       39520.000000
25%       67392.000000
50%       80288.000000
75%       94432.000000
max      147680.000000
Name: Annual_Income, dtype: float64

13. Accounts Table
   Category  Frequency  Percent Frequency  Cumulative Frequency  \
0         3    0.28365             28.365                  5673
1         2    0.22985             22.985                 10270
2         4    0.22410             22.410                 14752
3         5    0.12490             12.490                 17250
4         1    0.09585              9.585                 19167
5         6    0.04165              4.165                 20000

   Cumulative Percent Frequency
0                       28.365
1                       51.350
2                       73.760
3                       86.250
4                       95.835
5                      100.000

14. Credit line Results
                      count           mean             std          min
\
Total_Credit_Line  20000.0  156664.524845  190477.046579  4576.672724


                             25%          50%             75%
max
Total_Credit_Line  41396.25961  92588.89556  196154.699352
2.792347e+06

1. Min and Max Date Opened:
Min Date Opened: 1963-02-11
Max Date Opened: 2022-12-28

2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for Age of
the accounts (in years):
count    63546.000000
mean        14.608724
std         13.391270
min         -1.000000
```

```
25%          3.000000
50%         11.000000
75%         23.000000
max         58.000000
Name: Account_Age, dtype: float64

3. Frequency table for Account Age Flag:
Account_Age_Flag
False    43737
True     19809
Name: count, dtype: int64
Account_Age_Flag
False    0.688273
True     0.311727
Name: proportion, dtype: float64
Account_Age_Flag
False    43737
True     63546
Name: count, dtype: int64
Account_Age_Flag
False    0.688273
True     1.000000
Name: proportion, dtype: float64

4. Min and Max Account Number:
Min Account Number: 10016951
Max Account Number: 99997112

5. Frequency table for last digit of the Account Number:
Last_Digit_Account_Number
1    20000
2    18083
3    13486
4     7813
5     3331
6      833
Name: count, dtype: int64
Last_Digit_Account_Number
1    0.314733
2    0.284566
3    0.212224
4    0.122950
5    0.052419
6    0.013109
Name: proportion, dtype: float64
Last_Digit_Account_Number
1    20000
2    38083
3    51569
4    59382
```

```
5    62713
6    63546
Name: count, dtype: int64
Last_Digit_Account_Number
1    0.314733
2    0.599298
3    0.811522
4    0.934473
5    0.986891
6    1.000000
Name: proportion, dtype: float64

6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for
Account Credit Line:
count    6.354600e+04
mean     4.094768e+04
std      5.535326e+04
min      4.166727e+02
25%      1.179472e+04
50%      2.358716e+04
75%      4.747595e+04
max      1.333053e+06
Name: Account_Credit_Line, dtype: float64

7. Frequency table for Account Credit Line Flag:
Account_Credit_Line_Flag
False    63546
Name: count, dtype: int64
Account_Credit_Line_Flag
False    1.0
Name: proportion, dtype: float64
Account_Credit_Line_Flag
False    63546
Name: count, dtype: int64
Account_Credit_Line_Flag
False    1.0
Name: proportion, dtype: float64

8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for Annual
Fee:
count    63546.000000
mean       409.476802
std        553.532611
min          4.166727
25%        117.947247
50%        235.871569
75%        474.759459
max      13330.533003
Name: Annual_Fee, dtype: float64
```

```
9. Frequency table for Annual Fee Flag:
Annual_Fee_Flag
True     50875
False    12671
Name: count, dtype: int64
Annual_Fee_Flag
True     0.800601
False    0.199399
Name: proportion, dtype: float64
Annual_Fee_Flag
True     50875
False    63546
Name: count, dtype: int64
Annual_Fee_Flag
True     0.800601
False    1.000000
Name: proportion, dtype: float64

10. Min, P25, Median, P75, Max, Mean, and Standard Deviation for
Annual Interest Rate:
count    63546.000000
mean         0.224813
std          0.043111
min          0.150000
25%          0.187614
50%          0.224749
75%          0.261976
max          0.299994
Name: Annual_Interest_Rate, dtype: float64

1. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Transactions:
count    463982.000000
mean          7.323347
std           1.468484
min           4.000000
25%           6.000000
50%           7.000000
75%           8.000000
max          17.000000
Name: Total_Transactions_of_month, dtype: float64

2. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Purchases:
        Min_Purchase  25th_Percentile_Purchase  Median_Purchase  \
0        2295.186092               2856.143781      4464.249824
1        1273.795944               3192.253496      4194.005458
2        2381.514820               2712.693318      2912.294784
3         566.641445               1424.189031      2382.930571
4         461.888548               1577.808796      4880.423476
```

```
...              ...                    ...              ...
463977    714.691508               1627.008912      9646.522207
463978    256.071679               5988.783716     10781.286784
463979   4395.196168               5652.525842      8819.844831
463980   1086.530080               2884.260408      4042.237738
463981   6638.992955               8534.236198     11734.499166

        75th_Percentile_Purchase  Max_Purchase  Mean_Purchase  \
0                    5410.378424   6425.514989    4282.510631
1                    5316.259432   9979.027696    4681.670336
2                    5286.797109   5504.729636    3759.605933
3                    7004.405927   9833.853425    4135.977040
4                    9015.954963  11827.028060    5496.299952
...                          ...           ...            ...
463977              18769.528910  24852.680435   10858.138542
463978              11515.807948  17465.198517    9515.060432
463979              16451.328646  23500.215713   11075.978138
463980               7175.460744  15751.012137    5940.025678
463981              14515.450157  20299.224956   12219.521990

        Std_Dev_Purchase
0            1535.341382
1            2577.894639
2            1348.396600
3            3635.849016
4            4010.281549
...                  ...
463977       9242.206330
463978       5088.083055
463979       6406.337006
463980       4885.994874
463981       4655.040959

[463982 rows x 7 columns]

3. Min, P25, Median, P75, Max, Mean, and Standard Deviation for the
Number of Cash Advances:
        Min_Cash_Advance  25th_Percentile_Cash_Advance
Median_Cash_Advance  \
0                    NaN                           NaN
NaN
1                    NaN                           NaN
NaN
2                    NaN                           NaN
NaN
3                    NaN                           NaN
NaN
4             507.318914                    507.318914
507.318914
...                  ...                           ...
```

```
...
463977                NaN                           NaN
NaN
463978       1450.091077                  1450.091077
1450.091077
463979                NaN                           NaN
NaN
463980        553.842786                   553.842786
553.842786
463981                NaN                           NaN
NaN

        75th_Percentile_Cash_Advance  Max_Cash_Advance
Mean_Cash_Advance  \
0                                NaN               NaN
NaN
1                                NaN               NaN
NaN
2                                NaN               NaN
NaN
3                                NaN               NaN
NaN
4                         507.318914        507.318914
507.318914
...                              ...               ...
...
463977                           NaN               NaN
NaN
463978                  1450.091077       1450.091077
1450.091077
463979                           NaN               NaN
NaN
463980                   553.842786        553.842786
553.842786
463981                           NaN               NaN
NaN

        Std_Dev_Cash_Advance
0                        NaN
1                        NaN
2                        NaN
3                        NaN
4                        0.0
...                      ...
463977                   NaN
463978                   0.0
463979                   NaN
463980                   0.0
463981                   NaN
```

```
[463982 rows x 7 columns]

4. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Purchase Amounts:
                     count          mean            std        min  \
Total_Purchases  463982.0  96948.327842  138308.952155  27.452179

                           25%           50%            75%
max
Total_Purchases  26092.992786  53916.069284  111351.370052
5.295504e+06

5. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Cash Advance Amounts:
count    463982.000000
mean        743.643805
std        2528.290646
min           0.000000
25%           0.000000
50%           0.000000
75%         320.816826
max       98437.050490
Name: Total_Cash_Advances, dtype: float64

6. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Payments Amounts:
                    count          mean            std  min
25%  \
Total_Payments  463982.0  60739.550197  102447.592613  0.0
10303.089582

                          50%           75%           max
Total_Payments  29267.127983  69151.806295  3.657854e+06

7. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Closing Balances:
                     count         mean           std  min          25%
\
Closing_Balance  463982.0  35334.82927  73749.316041  0.0  1665.149654


                          50%           75%           max
Closing_Balance  12296.519537  37653.583839  2.477997e+06

8. Min, P25, Median, P75, Max, Mean, and Standard Deviation for all
the Minimum Amounts Due:
count    463982.000000
mean       3533.482927
std        7374.931604
min           0.000000
```

```
25%         166.514965
50%        1229.651954
75%        3765.358384
max      247799.670618
Name: Minimum_Amount_Due, dtype: float64

9. Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Payment Amounts of the month:
    Min_Payment  25th_Percentile_Payment  Median_Payment  \
0   6430.694066              7243.800256     8056.906445
1   2465.579872              6706.447632     8259.129824
2    238.343436               373.264089      508.184741
3    689.578460              6274.606951    11859.635441
4  21959.785867             21959.785867    21959.785867

   75th_Percentile_Payment   Max_Payment  Mean_Payment
Std_Dev_Payment
0              9650.131716  11243.356986   8576.985832
1998.881858
1              8428.606181   8519.856435   6875.923989
2550.433347
2               643.105394    778.026046    508.184741
269.841305
3             17444.663932  23029.692422  11859.635441
11170.056981
4             21959.785867  21959.785867  21959.785867
0.000000

10.Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Total Purchase Amounts of the month:
   Min_Purchase  25th_Percentile_Purchase  Median_Purchase  \
0   2295.186092               2856.143781      4464.249824
1   1273.795944               3192.253496      4194.005458
2   2381.514820               2712.693318      2912.294784
3    566.641445               1424.189031      2382.930571
4    461.888548               1577.808796      4880.423476

   75th_Percentile_Purchase  Max_Purchase  Mean_Purchase
Std_Dev_Purchase
0              5410.378424   6425.514989    4282.510631
1535.341382
1              5316.259432   9979.027696    4681.670336
2577.894639
2              5286.797109   5504.729636    3759.605933
1348.396600
3              7004.405927   9833.853425    4135.977040
3635.849016
4              9015.954963  11827.028060    5496.299952
4010.281549
```

```
11.Min, 25percent, Median, 75percent, Max, Mean, and Standard
Deviation for all the Total Cash Advance Amounts of the month:
   Min_Cash_Advance  25th_Percentile_Cash_Advance  Median_Cash_Advance
\
0               NaN                           NaN                  NaN

1               NaN                           NaN                  NaN

2               NaN                           NaN                  NaN

3               NaN                           NaN                  NaN

4        507.318914                    507.318914           507.318914


   75th_Percentile_Cash_Advance  Max_Cash_Advance
Mean_Cash_Advance  \
0                           NaN               NaN                  NaN

1                           NaN               NaN                  NaN

2                           NaN               NaN                  NaN

3                           NaN               NaN                  NaN

4                    507.318914        507.318914           507.318914


   Std_Dev_Cash_Advance
0                   NaN
1                   NaN
2                   NaN
3                   NaN
4                   0.0

13. Frequency table for Delinquency Counter:
Delinquency_Count
1    158821
2    150990
0     96353
3     57818
Name: count, dtype: int64
Delinquency_Count
1    0.342300
2    0.325422
0    0.207665
3    0.124613
Name: proportion, dtype: float64
Delinquency_Count
1    158821
2    309811
```

```
0    406164
3    463982
Name: count, dtype: int64
Delinquency_Count
1    0.342300
2    0.667722
0    0.875387
3    1.000000
Name: proportion, dtype: float64

14. Frequency table for Annual Active Flag:
Annual_Active_Flag
True     406164
False     57818
Name: count, dtype: int64
Annual_Active_Flag
True     0.875387
False    0.124613
Name: proportion, dtype: float64
Annual_Active_Flag
True     406164
False    463982
Name: count, dtype: int64
Annual_Active_Flag
True     0.875387
False    1.000000
Name: proportion, dtype: float64
Logs:

IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)
```