Name : Ramdayal

Intern-Id : EI0353 ¶

Position : Data-Analytics

Batch : 7 may 2023

Project : Credit-Risk-Analysis

Project_id : 01

```
In [509]: import pandas as pd
          from dateutil.parser import parse
```

```
In [510]: Customer_Acq = pd.read_csv("Customer_Acq.csv")
```

```
In [511]: Repayment=pd.read_csv("Repayment.csv")
```

```
In [512]: spend = pd.read_csv("spend.csv")
          # Create a copy of the existing dataset
          copied_spend = spend.copy()
```

Sanity Checks – Data Cleaning

Provide a meaningful treatment to all values where age is less than 18.

```
In [513]: # Calculate the mean age
          mean_age = Customer_Acq['Age'].mean()

          # Replace values where age is less than 18 with the mean age
          Customer_Acq.loc[Customer_Acq['Age'] < 18, 'Age'] = 18

          # Output the updated data
          Customer_Acq
```

Out[513]:

	Sl No:	Customer	Age	City	Credit Card Product	Limit	Company	Segment
0	1	A1	18.0000	BANGALORE	Gold	INR 500,000.00	C1	Self Employed
1	2	A2	35.5346	CALCUTTA	Silver	INR 100,000.00	C2	Salaried_MNC
2	3	A3	18.0000	COCHIN	Platimum	INR 10,000.00	C3	Salaried_Pvt
3	4	A4	45.8203	BOMBAY	Platimum	INR 10,001.00	C4	Govt
4	5	A5	69.6639	BANGALORE	Platimum	INR 10,002.00	C5	Normal Salary
...	...	...	...	...	...	...	...	...
95	96	A96	29.6316	CHENNAI	Silver	INR 100,000.00	C19	Salaried_Pvt
96	97	A97	20.6118	TRIVANDRUM	Platimum	INR 10,000.00	C20	Govt
97	98	A98	40.5390	CALCUTTA	Platimum	INR 10,001.00	C21	Normal Salary
98	99	A99	21.5887	CALCUTTA	Platimum	INR 10,002.00	C22	Self Employed
99	100	A100	23.6076	COCHIN	Silver	INR 100,000.00	C5	Salaried_MNC

In [514]:

```
# Remove non-numerical characters from the "Limit" column
Customer_Acq["Limit"] = Customer_Acq["Limit"].str.replace("[^\d.]", "", regex=True)

# Convert the "Limit" column to numeric type
Customer_Acq["Limit"] = pd.to_numeric(Customer_Acq["Limit"], errors="coerce")

# Create a copy of the existing dataset
copied_dataset = Customer_Acq.copy()

Customer_Acq
```

Out[514]:

	Sl No:	Customer	Age	City	Credit Card Product	Limit	Company	Segment
0	1	A1	18.0000	BANGALORE	Gold	500000.0	C1	Self Employed
1	2	A2	35.5346	CALCUTTA	Silver	100000.0	C2	Salaried_MNC
2	3	A3	18.0000	COCHIN	Platimum	10000.0	C3	Salaried_Pvt
3	4	A4	45.8203	BOMBAY	Platimum	10001.0	C4	Govt
4	5	A5	69.6639	BANGALORE	Platimum	10002.0	C5	Normal Salary
...	...	...	...	...	...	...	...	...
95	96	A96	29.6316	CHENNAI	Silver	100000.0	C19	Salaried_Pvt
96	97	A97	20.6118	TRIVANDRUM	Platimum	10000.0	C20	Govt
97	98	A98	40.5390	CALCUTTA	Platimum	10001.0	C21	Normal Salary
98	99	A99	21.5887	CALCUTTA	Platimum	10002.0	C22	Self Employed
99	100	A100	23.6076	COCHIN	Silver	100000.0	C5	Salaried_MNC

100 rows × 8 columns

Spending

In [515]:
```
spend
```

Out[515]:
```
      SINo  Customer      Month        Type  Amount
   0     1        A1  12-Jan-04   JEWELLERY  473776
   1     2        A1  03-Jan-04      PETRO   335579
   2     3        A1  15-Jan-04     CLOTHES  371041
   3     4        A1  25-Jan-04        FOOD  141178
   4     5        A1  17-Jan-05      CAMERA  398404
 ...   ...       ...        ...         ...     ...
1495  1496       A67  04-Feb-06  BUS TICKET  195841
1496  1497       A68  25-Mar-06  BUS TICKET  284263
1497  1498       A69  31-Mar-06  BUS TICKET  206552
1498  1499       A70  23-Mar-06  BUS TICKET   41025
1499  1500       A71  24-Mar-06  BUS TICKET  101398
```

In [516]:
```
# Convert the "Month" column to datetime format
spend['Month'] = spend['Month'].apply(lambda x: parse(str(x)))

# Extract year and month from the "Month" column
spend['Year'] = spend['Month'].dt.year
spend['Month'] = spend['Month'].dt.month

# Create the pivot table with the sum of spend for each customer in each month and year
monthly_spend = pd.pivot_table(spend, index=["Customer", "Year"], columns="Month", values="Amount", aggfunc="sum")
```

```
In [517]: monthly_spend
```

```
Customer  Year
A1        2004  1511173.0    41381.0       NaN       NaN  131197.0  NaN  NaN       NaN  NaN       NaN       NaN  NaN
          2005   398404.0  1404194.0       NaN       NaN       NaN  NaN  NaN  129388.0  NaN       NaN  457317.0  NaN
          2006        NaN        NaN       NaN  564506.0       NaN  NaN  NaN       NaN  NaN  220735.0       NaN  NaN
A10       2004   747428.0        NaN  435159.0       NaN  480729.0  NaN  NaN       NaN  NaN       NaN       NaN  NaN
          2005        NaN   357397.0       NaN       NaN  696068.0  NaN  NaN       NaN  NaN       NaN  484426.0  NaN
...       ...         ...        ...       ...       ...       ...  ...  ...       ...  ...       ...       ...  ...
A95       2004   347834.0        NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN       NaN       NaN  NaN
A96       2004   320364.0        NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN       NaN       NaN  NaN
A97       2004   164330.0        NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN       NaN       NaN  NaN
A98       2004    87484.0        NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN       NaN       NaN  NaN
A99       2004   476020.0        NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN       NaN       NaN  NaN

233 rows × 12 columns
```

```
In [518]: # Fill missing values with 0
          monthly_spend = monthly_spend.fillna(0)
```

In [519]:
```
monthly_spend
```

Out[519]:
```
          Month          1          2         3         4         5    6    7         8    9        10        11   12
Customer  Year
      A1  2004   1511173.0    41381.0       0.0       0.0  131197.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
          2005    398404.0  1404194.0       0.0       0.0       0.0  0.0  0.0  129388.0  0.0       0.0  457317.0  0.0
          2006         0.0        0.0       0.0  564506.0       0.0  0.0  0.0       0.0  0.0  220735.0       0.0  0.0
     A10  2004    747428.0        0.0  435159.0       0.0  480729.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
          2005         0.0   357397.0       0.0       0.0  696068.0  0.0  0.0       0.0  0.0       0.0  484426.0  0.0
     ...   ...         ...        ...       ...       ...       ...  ...  ...       ...  ...       ...       ...  ...
     A95  2004    347834.0        0.0       0.0       0.0       0.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
     A96  2004    320364.0        0.0       0.0       0.0       0.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
     A97  2004    164330.0        0.0       0.0       0.0       0.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
     A98  2004     87484.0        0.0       0.0       0.0       0.0  0.0  0.0       0.0  0.0       0.0       0.0  0.0
```

In [520]:
```
# Define the correct order of months
months_order = [
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
]
```

In [521]:
```
# Define the mapping dictionary
month_mapping = {i+1: month for i, month in enumerate(months_order)}

# Apply the mapping to the "Month" column
monthly_spend.columns = monthly_spend.columns.map(month_mapping)
```

```
In [522]: monthly_spend
```

Out[522]:

	Month	January	February	March	April	May	June	July	August	September	October	November	December
Customer	Year												
A1	2004	1511173.0	41381.0	0.0	0.0	131197.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2005	398404.0	1404194.0	0.0	0.0	0.0	0.0	0.0	129388.0	0.0	0.0	457317.0	0.0
	2006	0.0	0.0	0.0	564506.0	0.0	0.0	0.0	0.0	0.0	220735.0	0.0	0.0
A10	2004	747428.0	0.0	435159.0	0.0	480729.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2005	0.0	357397.0	0.0	0.0	696068.0	0.0	0.0	0.0	0.0	0.0	484426.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
A95	2004	347834.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A96	2004	320364.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A97	2004	164330.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A98	2004	87484.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A99	2004	476020.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

233 rows × 12 columns

Identity where the repayment is more than the spend then give them a credit of 2% of their credit limit in the next month biling.

In [523]:
```
Repayment
```

Out[523]:
```
      SL No:  Costomer      Month          Amount
   0       1        A1  12-Jan-04   331844.007400
   1       2        A1   3-Jan-04   441139.807300
   2       3        A1  15-Jan-04    32480.004010
   3       4        A1  25-Jan-04    90636.966530
   4       5        A1  17-Jan-05     1581.969829
 ...     ...       ...        ...             ...
1495    1496       A67   9-May-05   200288.874100
1496    1497       A68  10-May-06   231194.702300
1497    1498       A69   11-Jul-06  391195.615800
1498    1499       A70  12-Aug-05   357629.618200
1499    1500       A71  13-Sep-04   179771.745200

1500 rows × 4 columns
```

In [524]:
```
Repayment.rename(columns={'Costomer':'Customer'},inplace=True)
```

In [525]:
```
# Convert the "Month" column to datetime format
Repayment['Month'] = Repayment['Month'].apply(lambda x: parse(str(x)))

# Extract year and month from the "Month" column
Repayment['Year'] = Repayment['Month'].dt.year
Repayment['Month'] = Repayment['Month'].dt.month

# Create the pivot table with the sum of spend for each customer in each month and year
monthly_Repayment = pd.pivot_table(Repayment, index=["Customer", "Year"], columns="Month", values="Amount", aggfunc="
In [526]: monthly_Repayment
```

```
Out[526]:
              Month            1             2            3            4            5    6            7    8            9   10            11
Customer  Year
      A1  2004  1.362775e+06  1.911800e+05          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
          2005  1.581970e+03  1.199808e+06          NaN          NaN          NaN  NaN  300581.7031  NaN          NaN  NaN  278486.41560
          2006           NaN           NaN          NaN  371273.2744          NaN  NaN          NaN  NaN  108320.1205  NaN   17931.39929
     A10  2004  1.149997e+06           NaN  266929.3785          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
          2005  4.460683e+05  9.101423e+05          NaN  279491.6147  895696.1747  NaN          NaN  NaN          NaN  NaN           NaN
     ...   ...           ...           ...          ...          ...          ...  ...          ...  ...          ...  ...           ...
     A95  2004  7.510949e+04           NaN          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
     A96  2004  1.101390e+05           NaN          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
     A97  2004  1.746064e+05           NaN          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
     A98  2004  9.780260e+04           NaN          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN
     A99  2004  3.585899e+05           NaN          NaN          NaN          NaN  NaN          NaN  NaN          NaN  NaN           NaN

231 rows × 12 columns
```

```
In [527]: # Fill missing values with 0
          monthly_Repayment = monthly_Repayment.fillna(0)
```

```
In [528]: # Apply the mapping to the "Month" column
          monthly_Repayment.columns = monthly_Repayment.columns.map(month_mapping)
```

```
In [529]: monthly_Repayment
```

Out[529]:

	Month	January	February	March	April	May	June	July	August	September	October	Nove
Customer	**Year**											
A1	**2004**	1.362775e+06	1.911800e+05	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
	2005	1.581970e+03	1.199808e+06	0.0000	0.0000	0.0000	0.0	300581.7031	0.0	0.0000	0.0	278486.
	2006	0.000000e+00	0.000000e+00	0.0000	371273.2744	0.0000	0.0	0.0000	0.0	108320.1205	0.0	17931.
A10	**2004**	1.149997e+06	0.000000e+00	266929.3785	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
	2005	4.460683e+05	9.101423e+05	0.0000	279491.6147	895696.1747	0.0	0.0000	0.0	0.0000	0.0	0.
...	**...**	...	...	...	...	...	...	...	...	...	...	
A95	**2004**	7.510949e+04	0.000000e+00	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
A96	**2004**	1.101390e+05	0.000000e+00	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
A97	**2004**	1.746064e+05	0.000000e+00	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
A98	**2004**	9.780260e+04	0.000000e+00	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.
A99	**2004**	3.585899e+05	0.000000e+00	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0	0.

231 rows × 12 columns

Bonus Table

```
In [530]: # Subtract "Spend" values from "Repayment" values
          bonus = monthly_Repayment - monthly_spend
```

In [531]:
```python
# Display the new table
bonus
```

Out[531]:

	Month	January	February	March	April	May	June	July	August	September	October
Customer	**Year**										
A1	**2004**	-148397.764880	149799.01160	0.0000	0.0000	-131197.0000	0.0	0.0000	0.0	0.0000	0.0
	2005	-396822.030171	-204386.37550	0.0000	0.0000	0.0000	0.0	300581.7031	-129388.0	0.0000	0.0
	2006	0.000000	0.00000	0.0000	-193232.7256	0.0000	0.0	0.0000	0.0	108320.1205	-220735.0
A10	**2004**	402569.083600	0.00000	-168229.6215	0.0000	-480729.0000	0.0	0.0000	0.0	0.0000	0.0
	2005	446068.328500	552745.30313	0.0000	279491.6147	199628.1747	0.0	0.0000	0.0	0.0000	0.0
...	**...**	...	...	...	...	...	...	...	...	...	...
A95	**2004**	-272724.507770	0.00000	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0
A96	**2004**	-210224.985900	0.00000	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0
A97	**2004**	10276.392300	0.00000	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0
A98	**2004**	10318.599000	0.00000	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0
A99	**2004**	-117430.075700	0.00000	0.0000	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0

237 rows × 12 columns

In [532]:
```python
bonus[bonus<=0]=0
bonus=bonus*0.02
```

```
In [533]: bonus
```

Out[533]:

	Month	January	February	March	April	May	June	July	August	September	October	November	De
Customer	**Year**												
A1	**2004**	0.000000	2995.980232	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
	2005	0.000000	0.000000	0.0	0.000000	0.000000	0.0	6011.634062	0.0	0.00000	0.0	0.000000	
	2006	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	2166.40241	0.0	358.627986	
A10	**2004**	8051.381672	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
	2005	8921.366570	11054.906063	0.0	5589.832294	3992.563494	0.0	0.000000	0.0	0.00000	0.0	0.000000	
...	**...**	...	...	...	...	...	...	...	...	...	...	...	
A95	**2004**	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
A96	**2004**	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
A97	**2004**	205.527846	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
A98	**2004**	206.371980	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	
A99	**2004**	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.00000	0.0	0.000000	

237 rows × 12 columns

penality Table

```
In [534]: # Subtract "Spend" values from "Repayment" values
          penality = monthly_spend - monthly_Repayment;
```

```
In [535]: penality[penality<=0]=0
          penality=penality*0.029
          penality
```

Out[535]:

	Month	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**Dec**
Customer	**Year**												
A1	**2004**	4303.535182	0.00000	0.000000	0.000000	3804.713	0.0	0.0	0.000	0.0	0.000	0.000000	
	2005	11507.838875	5927.20489	0.000000	0.000000	0.000	0.0	0.0	3752.252	0.0	0.000	5186.086948	
	2006	0.000000	0.00000	0.000000	5603.749042	0.000	0.0	0.0	0.000	0.0	6401.315	0.000000	
A10	**2004**	0.000000	0.00000	4878.659024	0.000000	13941.141	0.0	0.0	0.000	0.0	0.000	0.000000	
	2005	0.000000	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	14048.354000	
...	**...**	...	...	...	...	...	...	...	...	...	...	...	
A95	**2004**	7909.010725	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	0.000000	
A96	**2004**	6096.524591	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	0.000000	
A97	**2004**	0.000000	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	0.000000	
A98	**2004**	0.000000	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	0.000000	
A99	**2004**	3405.472195	0.00000	0.000000	0.000000	0.000	0.0	0.0	0.000	0.0	0.000	0.000000	

237 rows × 12 columns

Highest Paying 10 Customer

```
In [536]: # Sort the Repayment table by the total repayment amount in descending order
          sorted_repayment = monthly_Repayment.sum(axis=1).sort_values(ascending=False)

          # Get the top 10 highest paying customers
          top_10_customers = sorted_repayment.head(10)

          # Print the top 10 highest paying customers
          top_10_customers
```

```
Out[536]: Customer  Year
          A47       2005    4.996185e+06
          A48       2005    4.320045e+06
          A60       2006    3.994239e+06
          A44       2005    3.895134e+06
          A56       2006    3.866303e+06
          A13       2005    3.807379e+06
          A28       2006    3.788398e+06
          A43       2005    3.787195e+06
          A61       2005    3.708511e+06
          A22       2005    3.644060e+06
          dtype: float64
```

People in which segment are spending more money.

```
In [537]: spend
```

```
Out[537]:
```

	SINo	Customer	Month	Type	Amount	Year
0	1	A1	1	JEWELLERY	473776	2004
1	2	A1	1	PETRO	335579	2004
2	3	A1	1	CLOTHES	371041	2004
3	4	A1	1	FOOD	141178	2004
4	5	A1	1	CAMERA	398404	2005
...	...	...	...	...	...	...
1495	1496	A67	2	BUS TICKET	195841	2006
1496	1497	A68	3	BUS TICKET	284263	2006
1497	1498	A69	3	BUS TICKET	206552	2006
1498	1499	A70	3	BUS TICKET	41025	2006
1499	1500	A71	3	BUS TICKET	101398	2006

1500 rows × 6 columns

```
In [538]: # Group the payment data by customer and calculate the sum of payments
          customer_spending = spend.groupby('Customer')['Amount'].sum()
```

```
In [539]: customer_spending
```

```
Out[539]: Customer
          A1      4858295
          A10     4110789
          A100      42254
          A11     4581924
          A12     6848587
                   ...
          A95      347834
          A96      320364
          A97      164330
          A98       87484
          A99      476020
          Name: Amount, Length: 100, dtype: int64
```

```
In [540]: # Merge the 'Customer_Avq' dataset with the 'customer_spending' data based on the 'Customer' attribute
          new_Customer_Acq_data = Customer_Acq.merge(customer_spending, on='Customer', how='left')
```

In [541]:
```
new_Customer_Acq_data
```

Out[541]:

	Sl No:	Customer	Age	City	Credit Card Product	Limit	Company	Segment	Amount
0	1	A1	18.0000	BANGALORE	Gold	500000.0	C1	Self Employed	4858295
1	2	A2	35.5346	CALCUTTA	Silver	100000.0	C2	Salaried_MNC	3151948
2	3	A3	18.0000	COCHIN	Platimum	10000.0	C3	Salaried_Pvt	2871165
3	4	A4	45.8203	BOMBAY	Platimum	10001.0	C4	Govt	3121669
4	5	A5	69.6639	BANGALORE	Platimum	10002.0	C5	Normal Salary	4816556
...	...	...	...	...	...	...	...	...	...
95	96	A96	29.6316	CHENNAI	Silver	100000.0	C19	Salaried_Pvt	320364
96	97	A97	20.6118	TRIVANDRUM	Platimum	10000.0	C20	Govt	164330
97	98	A98	40.5390	CALCUTTA	Platimum	10001.0	C21	Normal Salary	87484
98	99	A99	21.5887	CALCUTTA	Platimum	10002.0	C22	Self Employed	476020
99	100	A100	23.6076	COCHIN	Silver	100000.0	C5	Salaried_MNC	42254

100 rows × 9 columns

In [542]:
```
# Group the spending data by segment and calculate the sum of spending for each segment
seg_data = new_Customer_Acq_data.groupby('Segment')['Amount'].sum().reset_index()
```

In [543]:
```
seg_data
```

Out[543]:

	Segment	Amount
0	Govt	67325627
1	Normal Salary	107707139
2	Salaried_MNC	63639486
3	Salaried_Pvt	71704312
4	Self Employed	70975481

```
In [544]: highest_spending_segment = seg_data['Amount'].idxmax()
```

```
In [545]: # Get the segment name corresponding to the index
          highest_spending_segment_name = seg_data.loc[highest_spending_segment, 'Segment']

          # Print the segment name with the highest spending amount
          print("Segment with the highest spending amount:", highest_spending_segment_name)
```

```
Segment with the highest spending amount: Normal Salary
```

Which age group is spending more money?

```
In [546]: new_Customer_Acq_data
```

Out[546]:

	Sl No:	Customer	Age	City	Credit Card Product	Limit	Company	Segment	Amount
0	1	A1	18.0000	BANGALORE	Gold	500000.0	C1	Self Employed	4858295
1	2	A2	35.5346	CALCUTTA	Silver	100000.0	C2	Salaried_MNC	3151948
2	3	A3	18.0000	COCHIN	Platimum	10000.0	C3	Salaried_Pvt	2871165
3	4	A4	45.8203	BOMBAY	Platimum	10001.0	C4	Govt	3121669
4	5	A5	69.6639	BANGALORE	Platimum	10002.0	C5	Normal Salary	4816556
...	...	...	...	...	...	...	...	...	...
95	96	A96	29.6316	CHENNAI	Silver	100000.0	C19	Salaried_Pvt	320364
96	97	A97	20.6118	TRIVANDRUM	Platimum	10000.0	C20	Govt	164330
97	98	A98	40.5390	CALCUTTA	Platimum	10001.0	C21	Normal Salary	87484
98	99	A99	21.5887	CALCUTTA	Platimum	10002.0	C22	Self Employed	476020
99	100	A100	23.6076	COCHIN	Silver	100000.0	C5	Salaried_MNC	42254

100 rows × 9 columns

In [547]:
```python
# Define the age group categories
age_groups = {
    'Young Adults (18-39)': range(18, 45),
    'Middle-Age Adults (40-70)': range(45, 71),
    'Senior Citizens (>=71)': range(71, 201)  # Assuming 200 as an upper limit for senior citizens
}

# Calculate the total spending for each age group
age_group_spending = {}

# Loop through each row in the dataset
for index, row in new_Customer_Acq_data.iterrows():
    age = int(row['Age'])
    spending = row['Amount']

    # Find the age group category
    age_group_category = None
    for group, age_range in age_groups.items():
        if age in age_range:
            age_group_category = group
            break

    # Add the spending to the corresponding age group category
    if age_group_category:
        if age_group_category in age_group_spending:
            age_group_spending[age_group_category] += spending
        else:
            age_group_spending[age_group_category] = spending

# Create a DataFrame from the age group spending dictionary
age_group_spending_df = pd.DataFrame(age_group_spending.items(), columns=['Age Group', 'Total Spending'])

# Display the age group spending table
print(age_group_spending_df)
```

```
                   Age Group  Total Spending
0       Young Adults (18-39)       184011447
1  Middle-Age Adults (40-70)       139944034
2     Senior Citizens (>=71)        57396564
```

In which category the customers are spending more money?

```
In [548]: copied_spend
```

```
Out[548]:
```

	SlNo	Customer	Month	Type	Amount
0	1	A1	12-Jan-04	JEWELLERY	473776
1	2	A1	03-Jan-04	PETRO	335579
2	3	A1	15-Jan-04	CLOTHES	371041
3	4	A1	25-Jan-04	FOOD	141178
4	5	A1	17-Jan-05	CAMERA	398404
...	...	...	...	...	...
1495	1496	A67	04-Feb-06	BUS TICKET	195841
1496	1497	A68	25-Mar-06	BUS TICKET	284263
1497	1498	A69	31-Mar-06	BUS TICKET	206552
1498	1499	A70	23-Mar-06	BUS TICKET	41025
1499	1500	A71	24-Mar-06	BUS TICKET	101398

1500 rows × 5 columns

In which category the customers are spending more money?

In [549]:
```
# Group the data by 'TYPE' and calculate the sum of 'amount'
grouped_data1 = copied_spend.groupby('Type')['Amount'].sum()

# Create a new dataset with the grouped data
new_dataset = pd.DataFrame({'Type': grouped_data1.index, 'Total Amount': grouped_data1.values})

# Find the type that spends the most money
max_spending_type = new_dataset.loc[new_dataset['Total Amount'].idxmax(), 'Type']
max_spending_amount = new_dataset['Total Amount'].max()

print("Type with the highest spending:", max_spending_type)
print("Total spending in the type:", max_spending_amount)

# Print the new dataset
new_dataset
```

```
Type with the highest spending: PETRO
Total spending in the type: 51022578
```

Out[549]:

	Type	Total Amount
0	AIR TICKET	37435466
1	AUTO	10505088
2	BIKE	13152641
3	BUS TICKET	24905901
4	CAMERA	43721016
5	CAR	7018277
6	CLOTHES	24791096
7	FOOD	38296468
8	JEWELLERY	25247944
9	MOVIE TICKET	18784583
10	PETRO	51022578
11	RENTAL	20914668
12	SANDALS	6325018
13	SHOPPING	27418682
14	TRAIN TICKET	31812619

Monthly profit for the bank.

In [550]:
```python
#penaluty-bonus
bank_monthly_profit = penality - bonus;
bank_monthly_profit
```

Out[550]:

	Month	January	February	March	April	May	June	July	August	September	October	Novemb
ner	**Year**											
A1	**2004**	4303.535182	-2995.980232	0.000000	0.000000	3804.713000	0.0	0.000000	0.000	0.00000	0.000	0.00000
	2005	11507.838875	5927.204890	0.000000	0.000000	0.000000	0.0	-6011.634062	3752.252	0.00000	0.000	5186.08694
	2006	0.000000	0.000000	0.000000	5603.749042	0.000000	0.0	0.000000	0.000	-2166.40241	6401.315	-358.62798
\10	**2004**	-8051.381672	0.000000	4878.659024	0.000000	13941.141000	0.0	0.000000	0.000	0.00000	0.000	0.00000
	2005	-8921.366570	-11054.906063	0.000000	-5589.832294	-3992.563494	0.0	0.000000	0.000	0.00000	0.000	14048.3540
...	**...**	...	...	...	...	...	...	...	...	...	...	
\95	**2004**	7909.010725	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000	0.00000	0.000	0.00000
\96	**2004**	6096.524591	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000	0.00000	0.000	0.00000
\97	**2004**	-205.527846	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000	0.00000	0.000	0.00000
\98	**2004**	-206.371980	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000	0.00000	0.000	0.00000
\99	**2004**	3405.472195	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000	0.00000	0.000	0.00000

vs × 12 columns

In [551]:
```
# Group the original table by 'Year' and sum the values for each month
year_month_totals = bank_monthly_profit.groupby(['Year']).sum()

# Reset the index to convert 'Year' to a regular column
year_month_totals = year_month_totals.reset_index()

# Display the new table
year_month_totals
```

Out[551]:

Month	Year	January	February	March	April	May	June	July	August	Septembe
0	2004	330894.030748	92326.667881	21062.169741	42306.602845	4606.033326	0.000000	0.000000	0.000000	2007.32626
1	2005	50294.886888	99738.896954	0.000000	60724.800188	113580.765585	106566.004235	23112.088580	9916.482352	28770.98217
2	2006	49063.020049	81497.491328	181151.470734	149187.208076	-91120.460990	59378.646041	26857.275139	28896.793122	50524.67927

Which is the most profitable segment?

```
In [552]: # Sum up all the values in a single row of the bonus table
          row_totals = bonus.sum(axis=1)

          # Convert the result to a DataFrame
          row_totals = pd.DataFrame(row_totals, columns=['Total'])

          # Display the new table
          row_totals
```

```
Out[552]:
                               Total
Customer  Year
      A1  2004       2995.980232
          2005       6011.634062
          2006       2525.030396
     A10  2004       8051.381672
          2005      29558.668421
     ...   ...               ...
     A95  2004          0.000000
     A96  2004          0.000000
     A97  2004        205.527846
     A98  2004        206.371980
     A99  2004          0.000000

237 rows × 1 columns
```

```
In [553]: Customer_Acq
```

Out[553]:

	Sl No:	Customer	Age	City	Credit Card Product	Limit	Company	Segment
0	1	A1	18.0000	BANGALORE	Gold	500000.0	C1	Self Employed
1	2	A2	35.5346	CALCUTTA	Silver	100000.0	C2	Salaried_MNC
2	3	A3	18.0000	COCHIN	Platimum	10000.0	C3	Salaried_Pvt
3	4	A4	45.8203	BOMBAY	Platimum	10001.0	C4	Govt
4	5	A5	69.6639	BANGALORE	Platimum	10002.0	C5	Normal Salary
...	...	...	...	...	...	...	...	...
95	96	A96	29.6316	CHENNAI	Silver	100000.0	C19	Salaried_Pvt
96	97	A97	20.6118	TRIVANDRUM	Platimum	10000.0	C20	Govt
97	98	A98	40.5390	CALCUTTA	Platimum	10001.0	C21	Normal Salary
98	99	A99	21.5887	CALCUTTA	Platimum	10002.0	C22	Self Employed
99	100	A100	23.6076	COCHIN	Silver	100000.0	C5	Salaried_MNC

100 rows × 8 columns

In [555]:

```python
# Reset the index of row_totals DataFrame
row_totals.reset_index(inplace=True)

# Merge the row_totals and Customer_Acq tables based on the 'Customer' column
merged_table = pd.merge(row_totals, Customer_Acq[['Customer', 'Segment']], on='Customer', how='left')

# Display the merged table
merged_table
```

Out[555]:

	Customer	Year	Total	Segment
0	A1	2004	2995.980232	Self Employed
1	A1	2005	6011.634062	Self Employed
2	A1	2006	2525.030396	Self Employed
3	A10	2004	8051.381672	Normal Salary
4	A10	2005	29558.668421	Normal Salary
...	...	...	...	...
232	A95	2004	0.000000	Salaried_MNC
233	A96	2004	0.000000	Salaried_Pvt
234	A97	2004	205.527846	Govt
235	A98	2004	206.371980	Normal Salary
236	A99	2004	0.000000	Self Employed

237 rows × 4 columns

In [556]:
```
# Group the merged_table by 'Segment' and calculate the sum of 'Total' within each segment
grouped_table = merged_table.groupby('Segment').agg({'Total': 'sum'}).reset_index()

# Display the grouped table
grouped_table
```

Out[556]:

	Segment	Total
0	Govt	6.014154e+05
1	Normal Salary	1.036330e+06
2	Salaried_MNC	5.318797e+05
3	Salaried_Pvt	5.958515e+05
4	Self Employed	6.124568e+05

In [557]:
```
# Sort the grouped_table by 'Total' in descending order
sorted_table = grouped_table.sort_values('Total', ascending=False)

# Get the segment with the highest total
highest_segment = sorted_table.iloc[0]

# Display the segment with the highest total
highest_segment
```

Out[557]:
```
Segment       Normal Salary
Total        1036330.430555
Name: 1, dtype: object
```

In [ ]:
```

```