AtliQo BANK Credit Card Launch

PHASE 1 - Figure out the Target Market

- Distributions : Normal, Skewness
- Data Cleaning : Handling Null Values
- EDA : Pandas, Seaborn, Matplotlib
- EDA : Measures of Central Tendency
- EDA : Measures of Dispersion
- Outlier Treatment : IQR, StdDev, Mode
- Data Visualization : Histogram, Countplot

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

df_cust = pd.read_csv('E:/Study/Data Analysis/DS - BC 2025/2 - Math & Statistics/5 - PROJECT/PHASE 1/datasets/customers.csv')
df_cp = pd.read_csv('E:/Study/Data Analysis/DS - BC 2025/2 - Math & Statistics/5 - PROJECT/PHASE 1/datasets/credit_profiles.csv')
df_trans = pd.read_csv('E:/Study/Data Analysis/DS - BC 2025/2 - Math & Statistics/5 - PROJECT/PHASE 1/datasets/transactions.csv')

df_cust.head(5)
```

	cust_id	name	gender	age	location	occupation
0	1	Manya Acharya	Female	2	City	Business Owner
1	2	Anjali Pandey	Female	47	City	Consultant
2	3	Aaryan Chauhan	Male	21	City	Freelancer
3	4	Rudra Bali	Male	24	Rural	Freelancer
4	5	Advait Malik	Male	48	City	Consultant

	annual_income	marital_status
0	358211.0	Married
1	65172.0	Single
2	22378.0	Married
3	33563.0	Married
4	39406.0	Married

```python
df_cp.head(5)
```

	cust_id	credit_score	credit_utilisation	outstanding_debt
0	1	749	0.585171	19571.0
1	2	587	0.107928	161644.0
2	3	544	0.854807	513.0
3	4	504	0.336938	224.0
4	5	708	0.586151	18090.0

```
   credit_inquiries_last_6_months  credit_limit
0                             0.0       40000.0
1                             2.0        1250.0
2                             4.0        1000.0
3                             2.0        1000.0
4                             2.0       40000.0
```

```
df_trans.head(5)
```

```
   tran_id  cust_id   tran_date  tran_amount  platform
product_category  \
0        1      705  2023-01-01           63  Flipkart
Electronics
1        2      385  2023-01-01           99   Alibaba  Fashion &
Apparel
2        3      924  2023-01-01          471   Shopify
Sports
3        4      797  2023-01-01           33   Shopify  Fashion &
Apparel
4        5      482  2023-01-01           68    Amazon  Fashion &
Apparel

  payment_type
0      Phonepe
1  Credit Card
2      Phonepe
3         Gpay
4  Net Banking
```

SQL Connection

```
import mysql.connector

conn = mysql.connector.connect(
    host='localhost',
    user='root',
    passwd='root',
    database='e_master_card'
)

df_cust = pd.read_sql("SELECT * FROM customers", conn)
df_cust.head(3)
```

```
   cust_id            name  gender  age location      occupation  \
0        1   Manya Acharya  Female    2     City  Business Owner
1        2   Anjali Pandey  Female   47     City      Consultant
2        3  Aaryan Chauhan    Male   21     City      Freelancer
```

```
   annual_income marital_status
0         358211        Married
1          65172         Single
2          22378        Married
```

```
df_cp = pd.read_sql("SELECT * FROM credit_profiles", conn)
df_cp.head(3)
```

```
   cust_id  credit_score  credit_utilisation  outstanding_debt  \
0        1           749            0.585171           19571.0   
1        2           587            0.107928          161644.0   
2        3           544            0.854807             513.0   

   credit_inquiries_last_6_months  credit_limit  
0                             0.0       40000.0  
1                             2.0        1250.0  
2                             4.0        1000.0  
```

```
df_trans = pd.read_sql("SELECT * FROM transactions", conn)
df_trans.head(3)
```

```
   tran_id  cust_id   tran_date  tran_amount  platform 
product_category  \
0        1      705  2023-01-01           63  Flipkart 
Electronics   
1        2      385  2023-01-01           99   Alibaba  Fashion & 
Apparel   
2        3      924  2023-01-01          471   Shopify 
Sports   

  payment_type  
0      Phonepe  
1  Credit Card  
2      Phonepe  
```

```
# when you are done importing the data, close the connection
conn.close()
```

Overview of Data

```
print("Customers data - ",df_cust.shape)
print("Credit Score data - ",df_cp.shape)
print("Transactions data - ",df_trans.shape)
```

```
Customers data -  (1000, 8)
Credit Score data -  (1004, 6)
Transactions data -  (500000, 7)
```

```
df_cust.describe()
```

	cust_id	age	annual_income
count	1000.000000	1000.000000	1000.000000
mean	500.500000	36.405000	132439.799000
std	288.819436	15.666155	113706.313793
min	1.000000	1.000000	0.000000
25%	250.750000	26.000000	42229.750000
50%	500.500000	32.000000	107275.000000
75%	750.250000	46.000000	189687.500000
max	1000.000000	135.000000	449346.000000

```
df_cp.describe()
```

	cust_id	credit_score	credit_utilisation	outstanding_debt \
count	1004.000000	1004.000000	1000.000000	1000.000000
mean	500.850598	588.655378	0.498950	9683.597000
std	288.315670	152.575244	0.233139	25255.893671
min	1.000000	300.000000	0.103761	33.000000
25%	251.750000	459.000000	0.293917	221.000000
50%	502.500000	601.000000	0.487422	550.000000
75%	749.250000	737.250000	0.697829	11819.500000
max	1000.000000	799.000000	0.899648	209901.000000

	credit_inquiries_last_6_months	credit_limit
count	1000.000000	935.000000
mean	1.955000	19235.561497
std	1.414559	24489.997195
min	0.000000	500.000000
25%	1.000000	750.000000
50%	2.000000	1250.000000
75%	3.000000	40000.000000
max	4.000000	60000.000000

```
df_trans.describe()
```

	tran_id	cust_id	tran_amount
count	500000.000000	500000.000000	500000.00000
mean	250000.500000	501.400428	3225.20733
std	144337.711635	288.641924	13098.74276
min	1.000000	1.000000	0.00000
25%	125000.750000	252.000000	64.00000

```
50%     250000.500000     502.000000     141.00000
75%     375000.250000     752.000000     397.00000
max     500000.000000    1000.000000   69999.00000
```

Data Discrepancy

- For a bank customer, the min. age cannot be 1, and max. age cannot be 135
- The min. annual income for a bank customer, cannot be 0

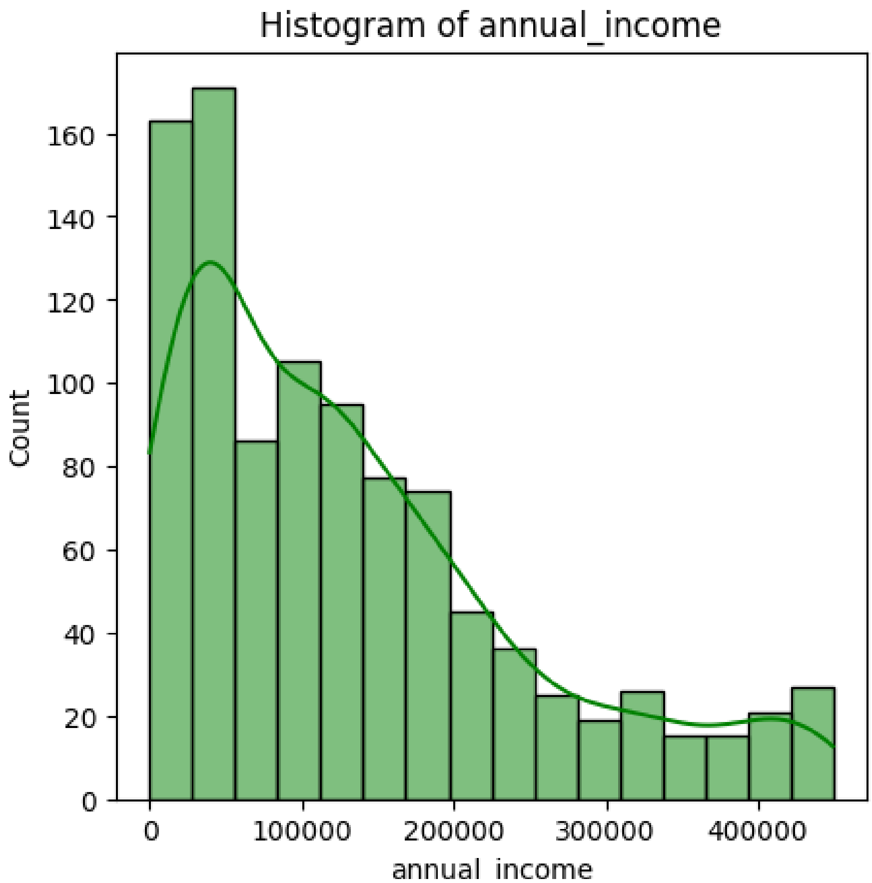
Therefore, the above discrepancies will be taken as **OUTLIERS**.

Treating Null Values

```
df_cust.isnull().sum()
```

```
cust_id           0
name              0
gender            0
age               0
location          0
occupation        0
annual_income     0
marital_status    0
dtype: int64
```

```
plt.figure(figsize=(5, 5))
sns.histplot(df_cust['annual_income'], kde=True, color='green',
label='Data')
plt.title('Histogram of annual_income')
plt.show()
```



We have following observations from the above,

1. **Age**: min = 1, max = 135
2. **Annual Income**: min = 2, max = 447 k

Age column has outliers. Annual income also seem to have outliers in terms of minimum value because business suggested that minimum income should be atleast 100

```
df_cust.annual_income.describe()

count      1000.000000
mean     132439.799000
std      113706.313793
min           0.000000
25%       42229.750000
50%      107275.000000
75%      189687.500000
max      449346.000000
Name: annual_income, dtype: float64

df_cp.isnull().sum()
```

```
cust_id                          0
credit_score                     0
credit_utilisation               4
outstanding_debt                 4
credit_inquiries_last_6_months   4
credit_limit                    69
dtype: int64
```

Outlier Detection: Annual income

Let us use standard deviation to detect outliers. Common practice is to treat anything that +/- 3 std dev as an outlier

```
df_cust['annual_income'].mean(), df_cust['annual_income'].std()

(132439.799, 113706.31379289791)

lower_sd = df_cust['annual_income'].mean() - 
3*df_cust['annual_income'].std()
upper_sd = df_cust['annual_income'].mean() + 
3*df_cust['annual_income'].std()

lower_sd, upper_sd

(-208679.14237869374, 473558.74037869374)

df_cust[df_cust['annual_income']>upper_sd]

Empty DataFrame
Columns: [cust_id, name, gender, age, location, occupation, 
annual_income, marital_status]
Index: []
```

We are seeing two outliers as per our statistical criteria of +/- 3 std dev.

But we don't always assume these as outliers all the time. We have to use business knowledge and our sense of judgement. Here after discussing with the business we concluded that having this type of higher income for business owners is usual and we will keep these data points as is to stay close to the reality while doing our analysis.

On the lower end however, we see minimum income as 2. Our business manager has told us that the income should be at least 100. We can use this as our criteria to find out the outliers on the lower end. These outliers could have occured due to a data error.

```
df_cust[df_cust.annual_income < 100]

     cust_id               name  gender  age location  
occupation  \
```

14	15	Sanjana Malik	Female	25	Rural	Artist
31	32	Veer Mistry	Male	50	City	Business Owner
82	83	Reyansh Mukherjee	Male	27	City	Freelancer
97	98	Virat Puri	Male	47	Suburb	Business Owner
102	103	Aarav Shah	Male	32	City	Data Scientist
155	156	Kiaan Saxena	Male	24	City	Fullstack Developer
170	171	Advait Verma	Male	52	City	Business Owner
186	187	Samar Sardar	Male	53	City	Consultant
192	193	Ishan Joshi	Male	37	Suburb	Data Scientist
227	228	Advait Mukherjee	Male	48	City	Business Owner
232	233	Aditya Goel	Male	26	City	Freelancer
240	241	Aaryan Bose	Male	24	Suburb	Freelancer
262	263	Vivaan Tandon	Male	53	Suburb	Business Owner
272	273	Kunal Sahani	Male	50	Suburb	Business Owner
275	276	Ananya Bali	Female	47	City	Consultant
312	313	Ritvik Gupta	Male	50	City	Consultant
315	316	Amara Jha	Female	25	City	Data Scientist
316	317	Yuvraj Saxena	Male	47	City	Consultant
333	334	Avani Khanna	Female	29	City	Data Scientist
340	341	Priya Sinha	Female	33	Rural	Fullstack Developer
402	403	Arnav Singh	Male	60	City	Business Owner
404	405	Arnav Banerjee	Male	26	City	Data Scientist
409	410	Kiaan Jain	Male	45	Rural	Consultant
440	441	Rudra Bose	Male	36	Suburb	Data Scientist
446	447	Aahan Gambhir	Male	60	City	Business

```
Owner
449       450        Anika Rathod  Female   24   Suburb  Fullstack
Developer
461       462          Kunal Nair    Male   33     City       Data
Scientist
474       475          Neha Verma  Female   28     City       Data
Scientist
502       503         Samar Dewan    Male   38   Suburb       Data
Scientist
508       509          Advait Das    Male   55     City       Business
Owner
516       517     Rehan Kulkarni    Male   29    Rural  Fullstack
Developer
530       531           Aarya Ver    Male   32     City       Business
Owner
536       537        Ritvik Patil    Male   33     City       Data
Scientist
543       544        Advait Batra    Male   54     City
Consultant
592       593        Priya Gandhi  Female   32     City       Business
Owner
599       600       Ishan Goswami  Female   38     City
Consultant
603       604     Kunal Malhotra    Male   25   Suburb  Fullstack
Developer
608       609       Kriti Lalwani  Female   25     City       Data
Scientist
633       634       Rudra Mehtani    Male   26     City       Data
Scientist
634       635         Anaya Dutta  Female   21     City
Freelancer
644       645           Dhruv Das    Male   64     City       Business
Owner
648       649       Kunal Rathore    Male   41     City
Consultant
650       651        Gauri Mittal  Female   47    Rural
Consultant
664       665        Ayush Khanna    Male   32    Rural  Fullstack
Developer
681       682        Arya Jaiswal    Male   37   Suburb       Data
Scientist
686       687      Vihaan Jaiswal    Male   40     City       Business
Owner
688       689         Dhruv Dewan    Male   26     City
Artist
693       694     Aditi Mehrotra  Female   37   Suburb       Data
Scientist
694       695         Rohan Mehta    Male   28     City       Data
Scientist
```

```
696      697         Ishan Negi    Male   47      City
Consultant
744      745         Swara Kaul  Female   39      City       Data
Scientist
784      785         Rohan Jain    Male   27      City       Data
Scientist
788      789     Vihaan Singhal    Male   20      City  Fullstack
Developer
791      792        Sara Mhatre  Female   38      City       Data
Scientist
817      818    Akshay Mehrotra    Male   47      City
Consultant
932      933     Avinash Tiwari    Male   35      City       Data
Scientist
955      956       Aahan Gandhi    Male   39    Suburb       Business
Owner
956      957        Priya Malik  Female   24      City
Artist
995      996     Manya Vasudeva  Female   26      City
Freelancer
998      999      Amara Rathore  Female   47      City       Business
Owner

     annual_income marital_status
14               0        Married
31              50        Married
82               0         Single
97               0        Married
102              0        Married
155              0        Married
170              0         Single
186              0         Single
192              0        Married
227              0        Married
232              0        Married
240              0        Married
262             50        Married
272              0        Married
275              0         Single
312              0        Married
315              0        Married
316             50        Married
333             50        Married
340             50        Married
402              0        Married
404              0         Single
409              0        Married
440              0        Married
446              0        Married
```

```
449               0       Married
461               0       Married
474               0        Single
502               0        Single
508               0       Married
516               0        Single
530               0       Married
536               0       Married
543               2       Married
592              50       Married
599               0        Single
603               0       Married
608               0        Single
633               2       Married
634               0       Married
644               0        Single
648               0       Married
650               0       Married
664               0       Married
681               0       Married
686               2       Married
688               0       Married
693               0       Married
694               0       Married
696              20       Married
744               0       Married
784               0        Single
788               0        Single
791               0        Single
817               0        Single
932               0       Married
955               0       Married
956               0       Married
995               0       Married
998               0       Married
```

Outlier Treatment: Annual income

Above records (with <100$ income) are outliers. We have following options to treat them :

1. **Remove them**: After discussion with business, we decided not to remove them as these are valid customers and we want to include them in our analysis
2. **Replace them with mean or median** : Mean is sensitive to outliers. It is better to use median for income values
3. **Replace them with occupation wise median**: Income level may vary based on occupation. For example median income for data scientist can be different from a

median income of a business owner. It is better to use occupation wise median income for replacement

```
occ_wise_inc_median = df_cust.groupby("occupation")
["annual_income"].median()
occ_wise_inc_median

occupation
Accountant               65265.0
Artist                   44915.0
Business Owner          254881.0
Consultant               51175.0
Data Scientist          127889.0
Freelancer               45189.5
Fullstack Developer      74457.0
Name: annual_income, dtype: float64

occ_wise_inc_median['Artist']

44915.0

for index, row in df_cust.iterrows():
    if row["annual_income"] < 100:
        occupation = df_cust.at[index, "occupation"]
        df_cust.at[index, "annual_income"] =
occ_wise_inc_median[occupation]

df_cust[df_cust.annual_income < 100]

Empty DataFrame
Columns: [cust_id, name, gender, age, location, occupation,
annual_income, marital_status]
Index: []

df_cust.loc[[240, 474, 502]]    # Now the customers with income <
$100, is updated with a median occupation income

     cust_id         name  gender  age location      occupation  \
240      241  Aaryan Bose    Male   24   Suburb      Freelancer
474      475   Neha Verma  Female   28     City  Data Scientist
502      503  Samar Dewan    Male   38   Suburb  Data Scientist

     annual_income marital_status
240        45189.5        Married
474       127889.0         Single
502       127889.0         Single
```

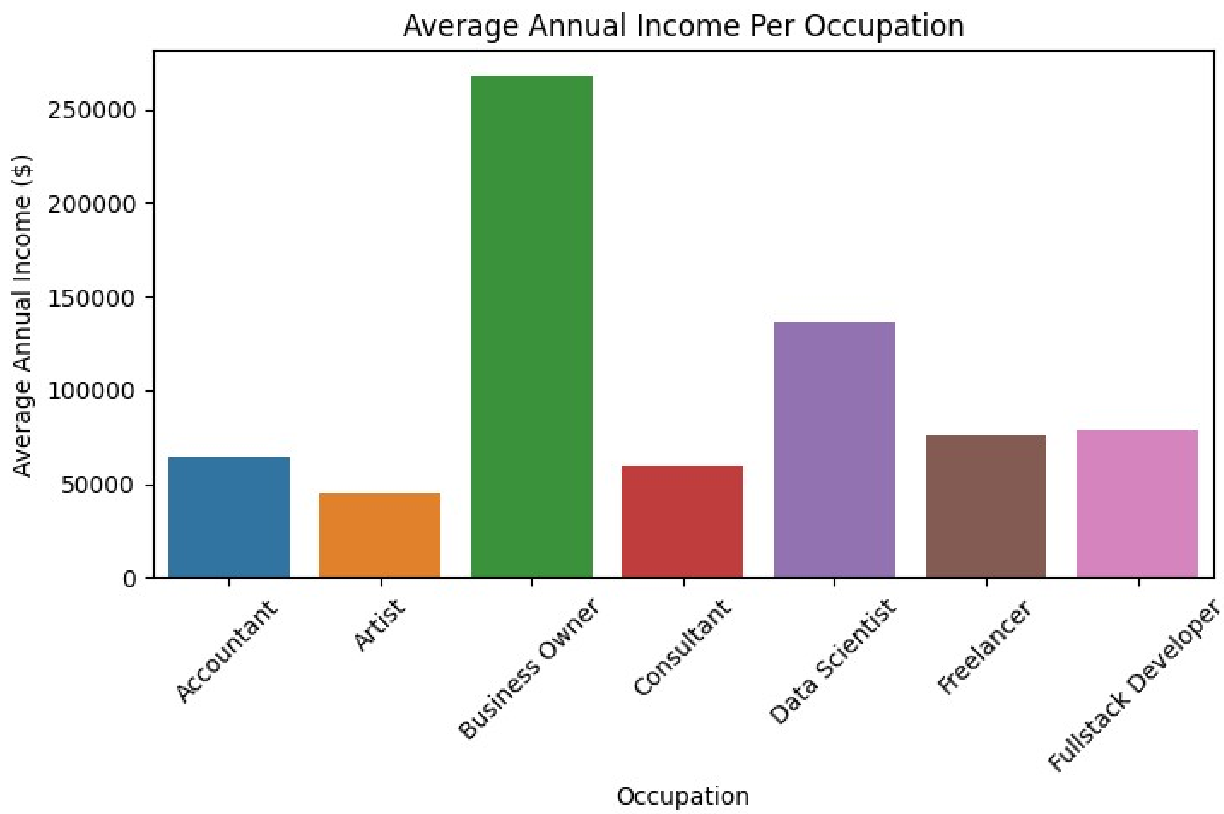
- The customers having income < $100 can be seen in **cell[21]**
- The 'median' of the occ. wise income can be seen in **cell[27]**

Data Visualization - Annual Income

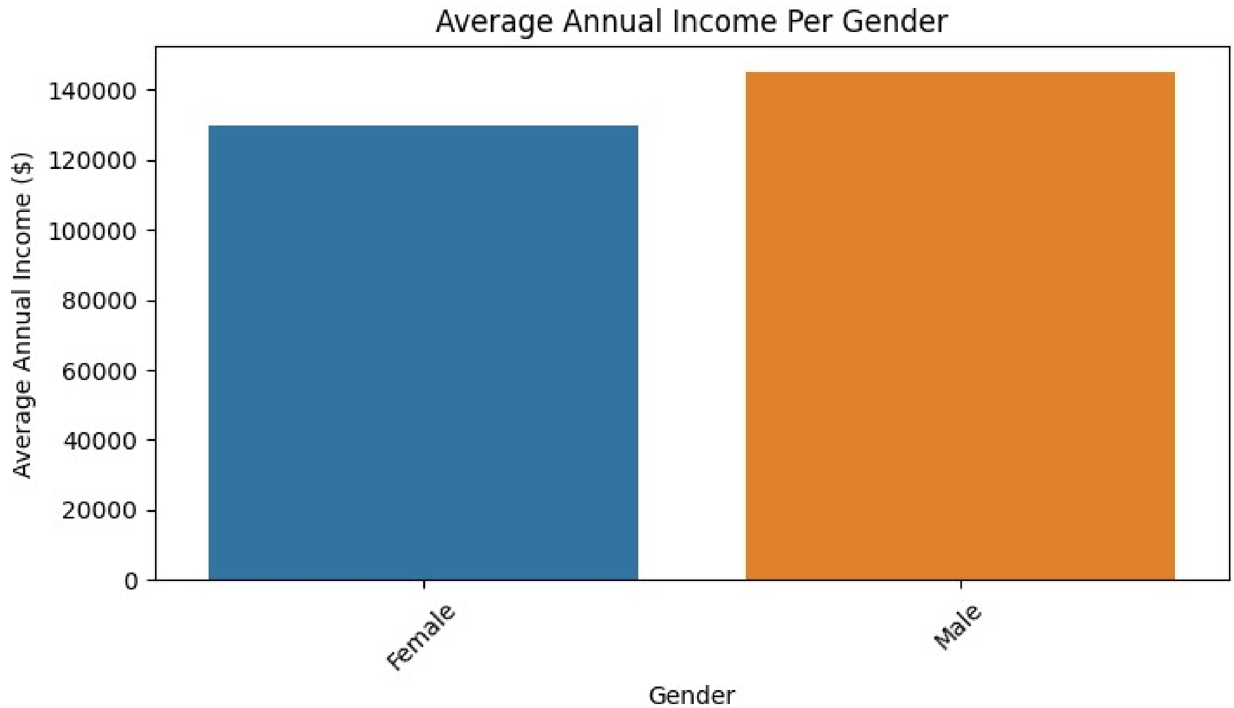
```
avg_income_per_occ = df_cust.groupby("occupation")
["annual_income"].mean().round(2)
avg_income_per_occ
```

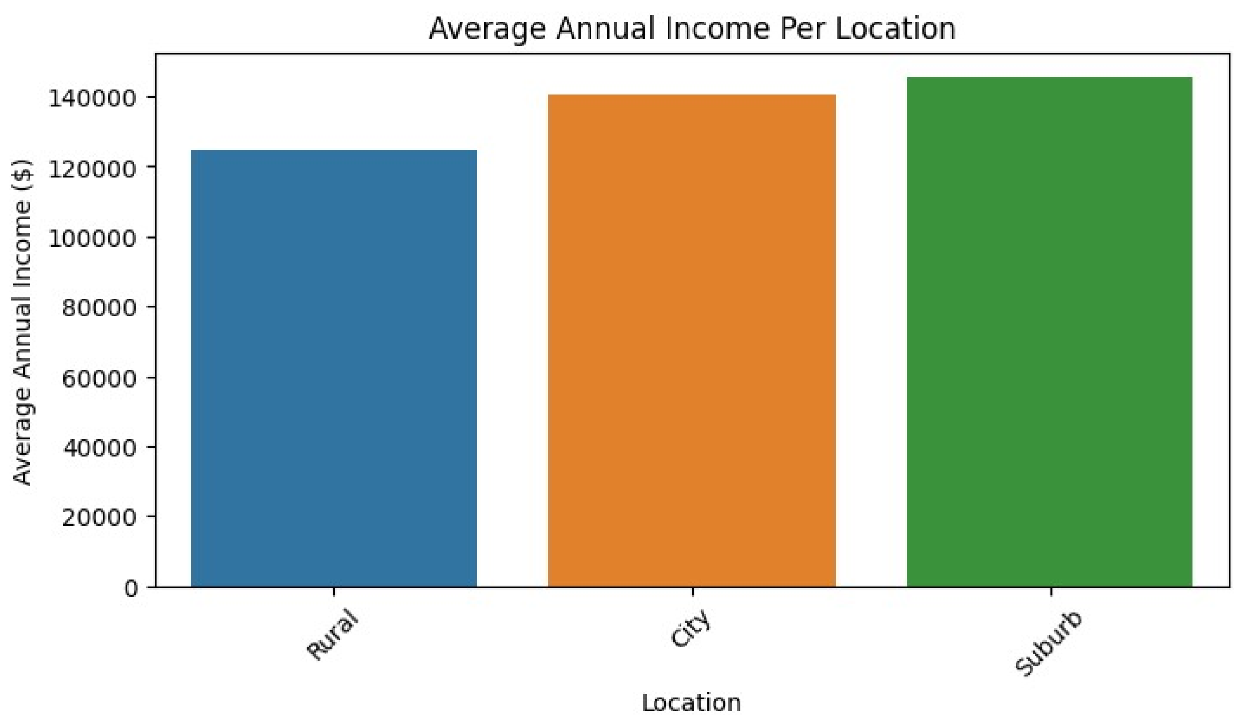
```
occupation
Accountant               64123.56
Artist                   45239.84
Business Owner          268119.83
Consultant               59927.26
Data Scientist          136208.60
Freelancer               76293.09
Fullstack Developer      78618.39
Name: annual_income, dtype: float64
```

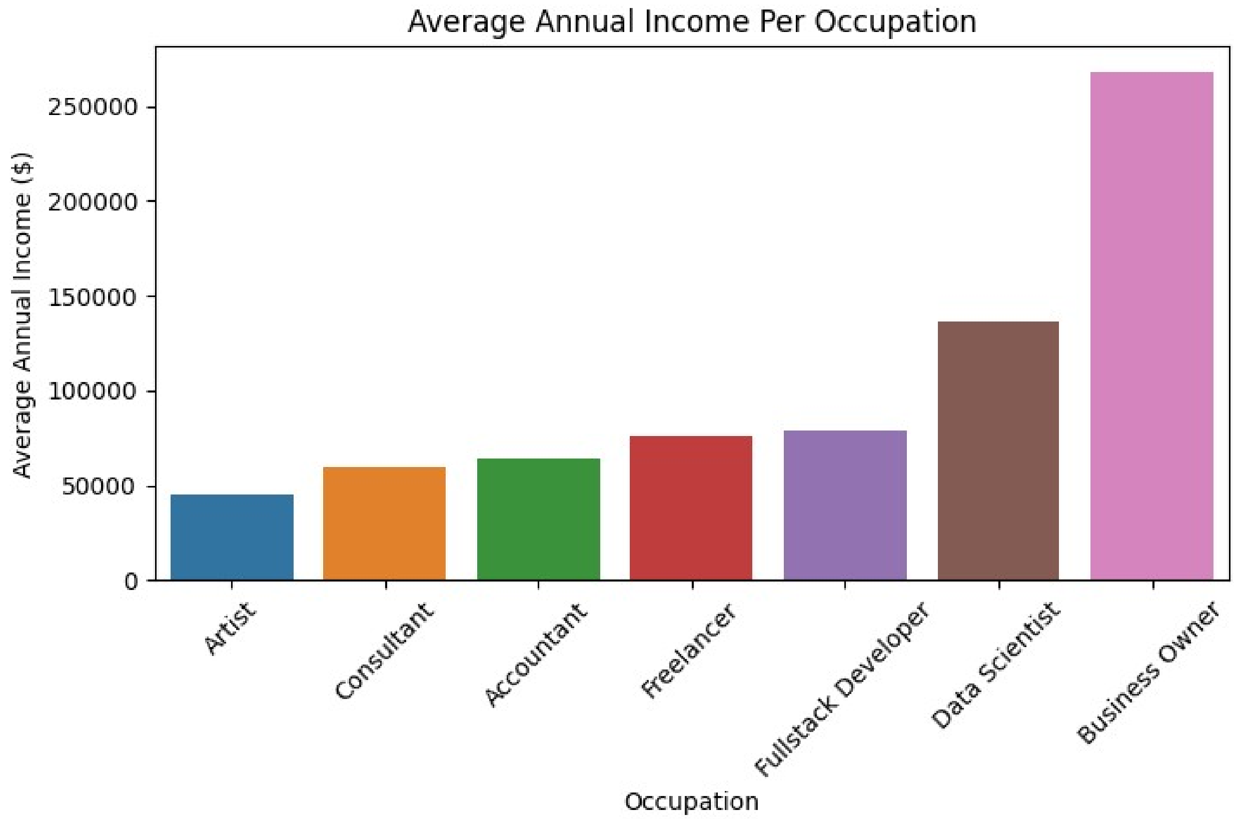
```
plt.figure(figsize = (8,4))
sns.barplot(x = avg_income_per_occ.index, y =
avg_income_per_occ.values, palette = 'tab10')
plt.xticks(rotation = 45)
plt.title('Average Annual Income Per Occupation')
plt.xlabel('Occupation')
plt.ylabel('Average Annual Income ($)')
plt.show()
```

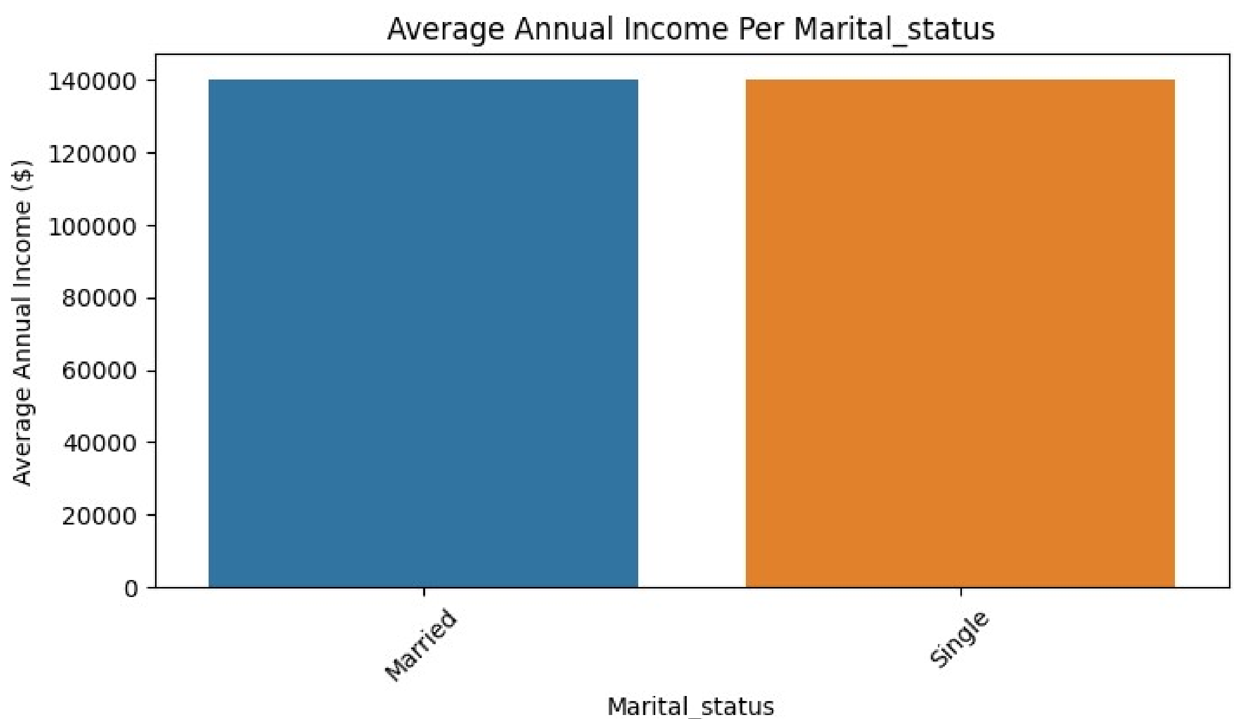



```
# List of categorical columns
categorical_columns = ['gender', 'location', 'occupation',
'marital_status']

# Loop through each categorical column and plot a bar chart of average
annual income
for col in categorical_columns:
    plt.figure(figsize = (8, 4))
    avg_income_per_group = df_cust.groupby(col)
['annual_income'].mean().sort_values()
    sns.barplot(x = avg_income_per_group.index, y =
avg_income_per_group.values, palette = 'tab10')
    plt.xticks(rotation=45)
    plt.title(f'Average Annual Income Per {col.capitalize()}')
    plt.xlabel(col.capitalize())
    plt.ylabel('Average Annual Income ($)')
    plt.show()
```


Average Annual Income Per Gender
Average Annual Income ($)
140000
120000
100000
80000
60000
40000
20000
0
Female
Male
Gender


Average Annual Income Per Location
Average Annual Income ($)
140000
120000
100000
80000
60000
40000
20000
0
Rural
City
Suburb
Location


Average Annual Income Per Occupation
Average Annual Income ($)
0
50000
100000
150000
200000
250000
Artist
Consultant
Accountant
Freelancer
Fullstack Developer
Data Scientist
Business Owner
Occupation


Average Annual Income Per Marital_status
Average Annual Income ($)
0
20000
40000
60000
80000
100000
120000
140000
Married
Single
Marital_status

Analysis of Age Column

```
df_cust.age.isnull().sum()
```

```
0
```

```
df_cust.describe()
```

	cust_id	age	annual_income
count	1000.000000	1000.000000	1000.000000
mean	500.500000	36.405000	140137.395500
std	288.819436	15.666155	110450.464107
min	1.000000	1.000000	5175.000000
25%	250.750000	26.000000	49620.500000
50%	500.500000	32.000000	115328.000000
75%	750.250000	46.000000	195514.250000
max	1000.000000	135.000000	449346.000000

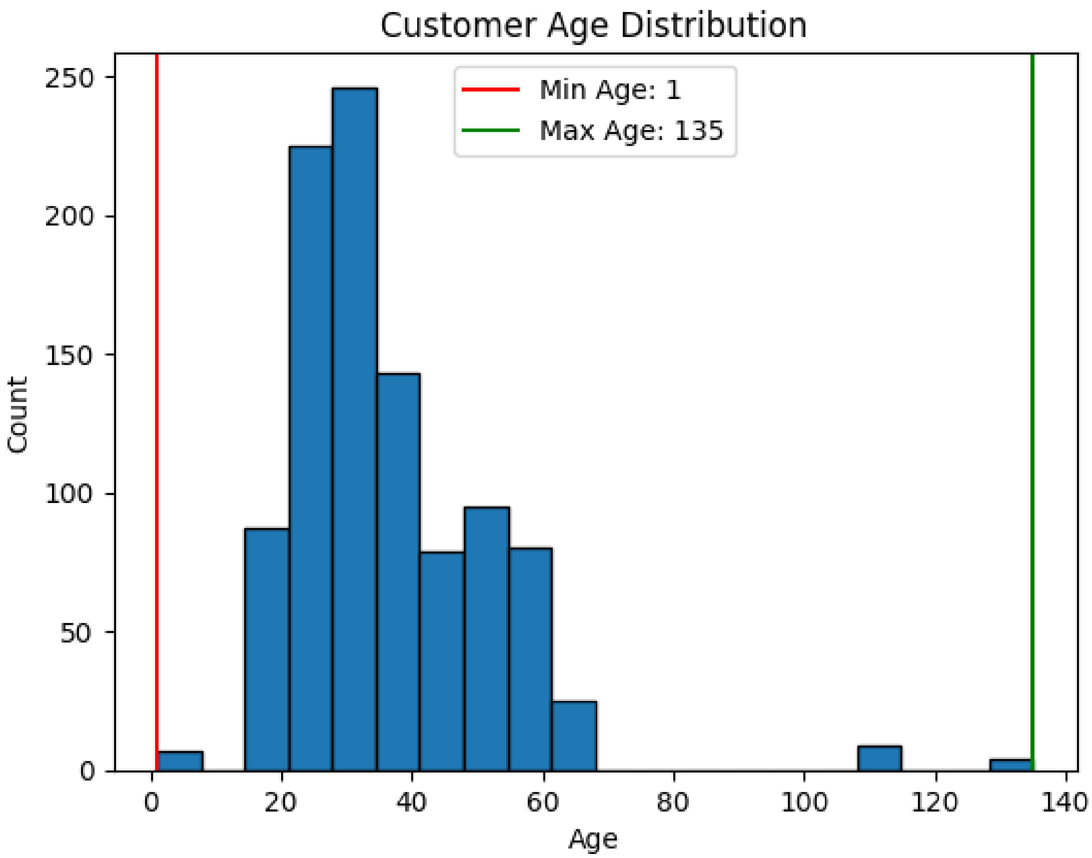
Outlier Treatment: Age

Above we see that min age is 1 and max age is 135. These seem to be outliers. So let's find out age distribution.

```
min_age = df_cust.age.min()
max_age = df_cust.age.max()

min_age, max_age
```

```
(1, 135)
```

```
plt.hist(df_cust.age, bins=20, edgecolor='black')
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Customer Age Distribution")

plt.axvline(min_age, color="red", label=f"Min Age: {min_age}")
plt.axvline(max_age, color="green", label=f"Max Age: {max_age}")

plt.legend()
plt.show()
```




```
df_cust[(df_cust.age<15)|(df_cust.age>80)]      # Client told that age
< 15 and age > 80 are invalid for the data

     cust_id             name  gender  age location
occupation  \
0          1    Manya Acharya  Female    2     City       Business
Owner
41        42      Aaryan Shah    Male  110     City
Artist
165      166        Sia Dutta  Female    1     City
Freelancer
174      175     Rohan Sharma    Male  110     City
Freelancer
222      223      Arjun Batra    Male  110   Suburb
Freelancer
277      278     Aarav Tandon    Male  110     City
Consultant
295      296     Ayush Pandey    Male    1    Rural
Accountant
325      326       Virat Goel    Male  110     City
Accountant
610      611      Rehan Verma    Male  135    Rural       Business
Owner
```

```
692      693         Dhruv Jha    Male    1     City       Business
Owner
703      704      Aanya Sharma  Female  110     City
Freelancer
709      710       Anika Verma  Female  110     City       Data
Scientist
728      729       Rehan Yadav    Male  135     City       Business
Owner
832      833         Ridhi Raj  Female  110     City  Fullstack
Developer
845      846    Rohan Jaiswal    Male    1     City
Consultant
855      856      Aanya Taneja  Female    2     City  Fullstack
Developer
895      896  Krishna Goswami    Male    1     City
Freelancer
923      924       Kunal Patel    Male  110     City
Freelancer
951      952      Virat Shetty    Male  135     City       Data
Scientist
991      992         Arya Dube    Male  135     City  Fullstack
Developer

     annual_income marital_status
0         358211.0        Married
41          7621.0        Married
165        39721.0         Single
174        23723.0        Married
222       210987.0        Married
277        96522.0         Single
295        55254.0        Married
325        61021.0         Single
610       444776.0        Married
692        83045.0        Married
703        43404.0         Single
709        98417.0        Married
728       382836.0        Married
832        95379.0         Single
845        20838.0        Married
855        30689.0        Married
895        31533.0        Married
923        51629.0        Married
951        49677.0        Married
991        93267.0         Single
```

```
outliers = df_cust[(df_cust.age<15)|(df_cust.age>80)]    # storing
outliers in a separate DF
outliers.shape
```

```
(20, 8)
```

Total 20 outliers for age. Now how can we handle these outliers?

Possible options,

- Remove them: This doesn't sound like a good option as we will loose important information
- Replace outlier values with some appropriate value: We can use mean or median for this

```
df_cust.age.median()

32.0
```

Instead of replacing it with a **median** age for all customers, how about we calculate **median age per occupation**?

```
# Just like we did for the Income part

median_age_per_occupation = df_cust.groupby('occupation')
['age'].median()
median_age_per_occupation

occupation
Accountant              31.5
Artist                  26.0
Business Owner          51.0
Consultant              46.0
Data Scientist          32.0
Freelancer              24.0
Fullstack Developer     27.5
Name: age, dtype: float64

for index, row in outliers.iterrows():
    if pd.notnull(row['age']):
        occupation = df_cust.at[index, 'occupation']
        df_cust.at[index, 'age'] =
median_age_per_occupation[occupation]
```

```
for index, row in outliers.iterrows():
    if pd.notnull(row['age']):
        occupation = df_cust.at[index, 'occupation']
        df_cust.at[index, 'age'] =
median_age_per_occupation[occupation]

df_cust[(df_cust.age<15)|(df_cust.age>80)]

Empty DataFrame
Columns: [cust_id, name, gender, age, location, occupation,
annual_income, marital_status]
Index: []
```

```
df_cust.age.describe()
```

```
count    1000.000000
mean       35.541500
std        12.276634
min        18.000000
25%        26.000000
50%        32.000000
75%        44.250000
max        64.000000
Name: age, dtype: float64
```

We can see above, we don't have any outliers left. Min age is 18 and Max age is 64

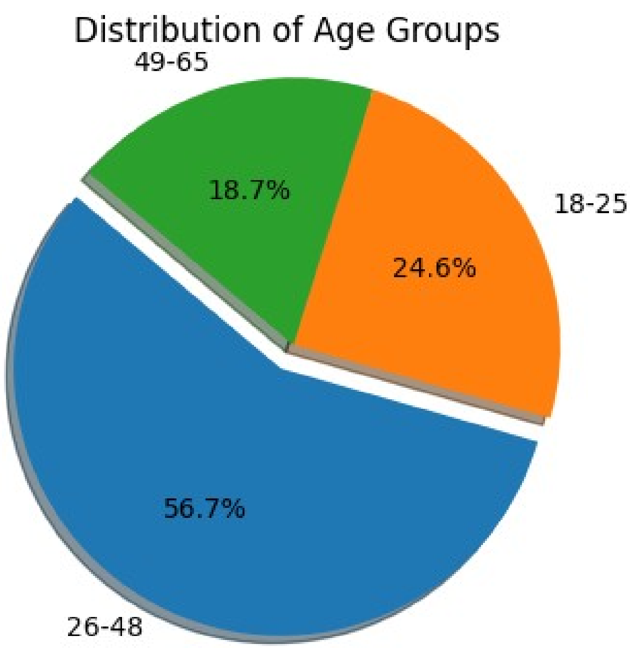
Data Visualization - Age and Gender

```
# Defining the bin edges and labels

bin_edges = [17, 25, 48, 65]
bin_labels = ['18-25', '26-48', '49-65']

# Using the cut function to bin and label the age column

pd.cut(df_cust['age'], bins=bin_edges, labels=bin_labels)
```

```
0      49-65
1      26-48
2      18-25
3      18-25
4      26-48
       ...
995    26-48
996    49-65
997    26-48
998    26-48
999    26-48
Name: age, Length: 1000, dtype: category
Categories (3, object): ['18-25' < '26-48' < '49-65']
```

```
bin_edges = [17, 25, 48, 65]
bin_labels = ['18-25', '26-48', '49-65']

df_cust['age_group'] = pd.cut(df_cust['age'], bins=bin_edges,
labels=bin_labels)

df_cust['age_group'].value_counts(normalize=True)*100
```

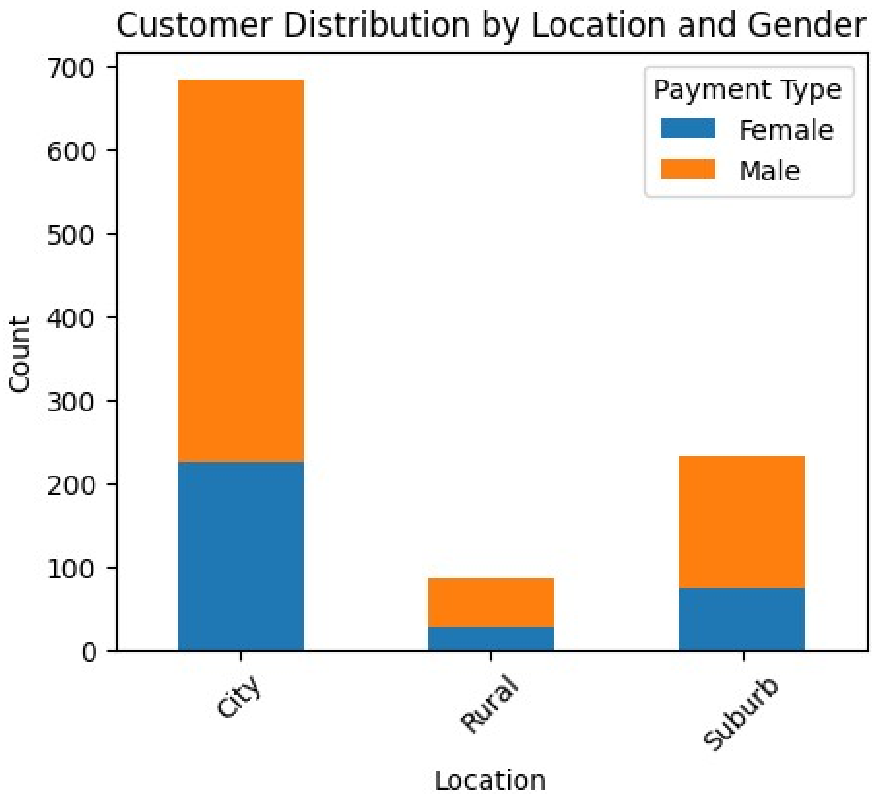
```
age_group
26-48    56.7
18-25    24.6
49-65    18.7
Name: proportion, dtype: float64

# Calculate the count of values in each age group
age_group_counts = df_cust['age_group'].value_counts(normalize=True) *
100

# Plot the pie chart
plt.figure(figsize=(4, 4))
plt.pie(
    age_group_counts,
    labels=age_group_counts.index,
    explode=(0.1,0,0),
    autopct='%1.1f%%',
    shadow=True,
    startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.title('Distribution of Age Groups')
plt.show()
```



More than 50% of customer base are in the age group of 26 - 48 and ~26% are of age group 18 - 25

Analyze Gender and Location Distribution

```
customer_location_gender = df_cust.groupby(['location',
'gender']).size().unstack(fill_value=0)

# Create a stacked bar chart to visualize the distribution of payment
types for each occupation
customer_location_gender.plot(kind='bar', stacked=True, figsize=(5,
4))

# Add labels and title
plt.xlabel('Location')
plt.ylabel('Count')
plt.title('Customer Distribution by Location and Gender')

# Show the bar chart
plt.legend(title='Payment Type', bbox_to_anchor=(1, 1))  # Add a
legend

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

plt.show()
```



Exploring Credit_Score Table - 1

1 - Removing Duplicates

```
df_cp.head()
```

```
   cust_id  credit_score  credit_utilisation  outstanding_debt  \
0        1           749            0.585171           19571.0
1        2           587            0.107928          161644.0
2        3           544            0.854807             513.0
3        4           504            0.336938             224.0
4        5           708            0.586151           18090.0

   credit_inquiries_last_6_months  credit_limit
0                             0.0       40000.0
1                             2.0        1250.0
2                             4.0        1000.0
3                             2.0        1000.0
4                             2.0       40000.0
```

```
df_cp.shape
```

```
(1004, 6)
```

```
df_cust.shape
```

```
(1000, 9)
```

```
df_cp['cust_id'].nunique()
```

```
1000
```

There are **4 extra** values in **credit profile**

```
df_cp.duplicated('cust_id')
```

```
0       False
1       False
2       False
3       False
4       False
        ...
999     False
1000    False
1001    False
1002    False
1003    False
Length: 1004, dtype: bool
```

```
df_cp[df_cp.duplicated('cust_id', keep=False)]
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
516      517           308                 NaN               NaN
517      517           308            0.113860              33.0
569      569           344                 NaN               NaN
570      569           344            0.112599              37.0
607      606           734                 NaN               NaN
608      606           734            0.193418            4392.0
664      662           442                 NaN               NaN
665      662           442            0.856039             266.0

     credit_inquiries_last_6_months  credit_limit
516                             NaN           NaN
517                             3.0         500.0
569                             NaN           NaN
570                             0.0         500.0
607                             NaN           NaN
608                             1.0       40000.0
664                             NaN           NaN
665                             2.0         500.0
```

```
df_cp_upd_1 = df_cp.drop_duplicates(subset='cust_id', keep="last")
df_cp_upd_1.shape
```

```
(1000, 6)
```

```python
df_cp_upd_1[df_cp_upd_1.duplicated('cust_id', keep=False)]
```

```
Empty DataFrame
Columns: [cust_id, credit_score, credit_utilisation, outstanding_debt, credit_inquiries_last_6_months, credit_limit]
Index: []
```

df_cp_upd_1 looks clean now after cleaning duplicates.

Next step would be to see if there are any **null values**

2 - Handling Null Values

```python
df_cp_upd_1.isnull().sum()
```

```
cust_id                            0
credit_score                       0
credit_utilisation                 0
outstanding_debt                   0
credit_inquiries_last_6_months     0
credit_limit                      65
dtype: int64
```

credit_limit has a bunch of null values.

From the business knowledge we know that credit limit depends on credit score of a customer.

We will try to find out if we can figure out a mathematical relationship between credit score and credit limit and use credit score to full NULL values in credit limit.

```python
df_cp_upd_1[df_cp_upd_1.credit_limit.isnull()]
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
10        11           679            0.557450            9187.0
35        36           790            0.112535            4261.0
37        38           514            0.296971             238.0
45        46           761            0.596041           24234.0
64        65           734            0.473715           13631.0
..       ...           ...                 ...               ...
912      909           479            0.487555             320.0
931      928           311            0.832244             316.0
948      945           526            0.272734             227.0
954      951           513            0.175914             131.0
957      954           783            0.867421           46451.0

     credit_inquiries_last_6_months  credit_limit
10                              2.0           NaN
```

```
35                                1.0           NaN
37                                2.0           NaN
45                                2.0           NaN
64                                0.0           NaN
..                                ...           ...
912                               3.0           NaN
931                               2.0           NaN
948                               1.0           NaN
954                               3.0           NaN
957                               0.0           NaN

[65 rows x 6 columns]
```

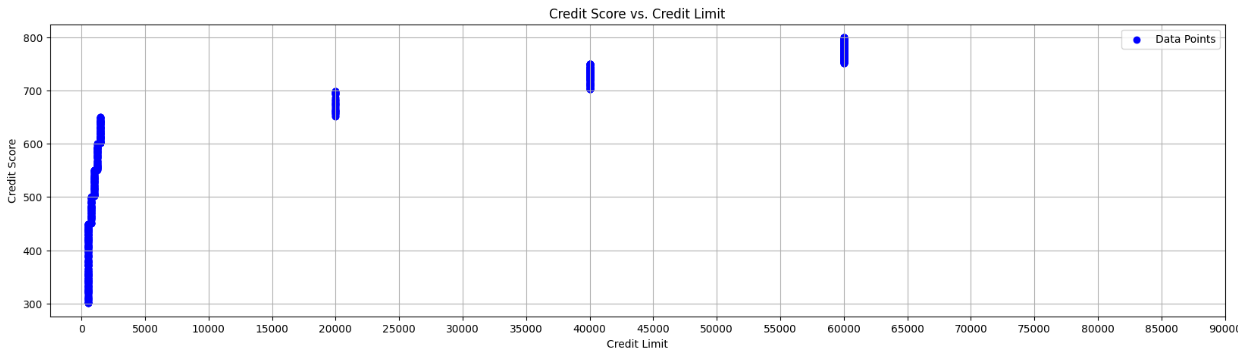
```
df_cp_upd_1['credit_limit'].unique()
```

```
array([40000.,  1250.,  1000.,   500.,   750.,    nan,  1500., 60000.,
       20000.])
```

```
df_cp_upd_1['credit_limit'].value_counts()
```

```
credit_limit
500.0      229
60000.0    186
40000.0    137
1500.0     100
1000.0      90
750.0       76
1250.0      75
20000.0     42
Name: count, dtype: int64
```

```
# Looking at scatter plot for credit score vs credit_limit again
(after handling oultiers)

# Creating the plot
plt.figure(figsize=(20, 5))
plt.scatter(df_cp_upd_1['credit_limit'], df_cp_upd_1['credit_score'],
c='blue', marker='o', label='Data Points')

# Customizing the plot
plt.title('Credit Score vs. Credit Limit')
plt.xlabel('Credit Limit')
plt.ylabel('Credit Score')

# Adjusting the y-axis bin interval to 1000
plt.xticks(range(0, 90001, 5000))
plt.grid(True)

# Showing the plot
plt.legend()
plt.show()
```



Above, we can see clear relationship between credit score and credit limit.

Where there are levels for example, upto 650 score is getting a very minor credit limit (<1000$) whereas a score between 650 to 700 is getting around 20000. Score between 700 to 750 is getting around 40K etc.

```
# Defining the bin ranges
bin_ranges = [300, 450, 500, 550, 600, 650, 700, 750, 800]

# Creating labels for the bins
bin_labels = [f'{start}-{end-1}' for start, end in zip(bin_ranges,
bin_ranges[1:])]

# Using pd.cut to assign data to bins
df_cp_upd_1['credit_score_range'] =
pd.cut(df_cp_upd_1['credit_score'], bins=bin_ranges,
labels=bin_labels, include_lowest=True, right=False)

df_cp_upd_1.head()
```

	cust_id	credit_score	credit_utilisation	outstanding_debt \
0	1	749	0.585171	19571.0
1	2	587	0.107928	161644.0
2	3	544	0.854807	513.0
3	4	504	0.336938	224.0
4	5	708	0.586151	18090.0

	credit_inquiries_last_6_months	credit_limit	credit_score_range
0	0.0	40000.0	700-749
1	2.0	1250.0	550-599
2	4.0	1000.0	500-549
3	2.0	1000.0	500-549
4	2.0	40000.0	700-749

We can now see a **new column called credit_score_range** which is calculated based on the **credit_score** column

```
df_cp_upd_1[['credit_score','credit_score_range',
'credit_limit']].head(3)
```

```
   credit_score credit_score_range  credit_limit
0           749            700-749       40000.0
1           587            550-599        1250.0
2           544            500-549        1000.0
```

```
df_cp_upd_1[df_cp_upd_1['credit_score_range']=="750-799"]
```

```
      cust_id  credit_score  credit_utilisation  outstanding_debt  \
21         22           785            0.897089           36083.0
25         26           758            0.250811          190838.0
26         27           766            0.830908           31344.0
29         30           798            0.222597            7238.0
31         32           768            0.747793           35109.0
...       ...           ...                 ...               ...
988       985           770            0.628088           33405.0
993       990           772            0.259958           11937.0
996       993           782            0.477170           20305.0
1000      997           774            0.465462           17139.0
1003     1000           775            0.696050           33956.0

      credit_inquiries_last_6_months  credit_limit credit_score_range

21                               3.0       60000.0            750-799

25                               2.0       60000.0            750-799

26                               3.0       60000.0            750-799

29                               2.0       60000.0            750-799

31                               2.0       60000.0            750-799

...                              ...           ...                ...

988                              2.0       60000.0            750-799

993                              2.0       60000.0            750-799

996                              2.0       60000.0            750-799

1000                             0.0       60000.0            750-799

1003                             1.0       60000.0            750-799


[213 rows x 7 columns]
```

```
df_cp_upd_1[df_cp_upd_1['credit_score_range']=="300-449"]
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
5          6           442            0.705409             246.0   
11        12           429            0.627645             263.0   
15        16           347            0.531660             190.0   
18        19           447            0.795650             292.0   
20        21           381            0.714710             307.0   
..       ...           ...                 ...               ...   
981      978           371            0.435307             183.0   
982      979           332            0.150815              65.0   
984      981           327            0.377202             108.0   
989      986           425            0.178470              56.0   
998      995           360            0.594345             242.0   

     credit_inquiries_last_6_months  credit_limit credit_score_range  
5                               4.0         500.0            300-449  
11                              0.0         500.0            300-449  
15                              0.0         500.0            300-449  
18                              1.0         500.0            300-449  
20                              0.0         500.0            300-449  
..                              ...           ...                ...  
981                             2.0         500.0            300-449  
982                             1.0         500.0            300-449  
984                             3.0         500.0            300-449  
989                             4.0         500.0            300-449  
998                             0.0         500.0            300-449  

[237 rows x 7 columns]
```

Above we can see that for credit score range "750-799" the credit limit is 60K whereas for "300-449" it is 500.

We can use MODE function to find out most frequently occuring credit limit for a given score range.

```
mode_df = df_cp_upd_1.groupby('credit_score_range')
['credit_limit'].agg(lambda x: x.mode().iloc[0]).reset_index()
mode_df

  credit_score_range  credit_limit
0            300-449         500.0
1            450-499         750.0
2            500-549        1000.0
3            550-599        1250.0
4            600-649        1500.0
5            650-699       20000.0
6            700-749       40000.0
7            750-799       60000.0

df_cp_upd_1[df_cp_upd_1.credit_limit.isnull()].sample(3)
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
430      431           610            0.741063             628.0   
83        84           733            0.525567           16663.0   
650      648           405            0.231599              63.0   

     credit_inquiries_last_6_months  credit_limit credit_score_range  
430                             4.0           NaN            600-649  
83                              1.0           NaN            700-749  
650                             0.0           NaN            300-449  
```

```
# Merging the mode values back with the original DataFrame

df_cp_upd_2 = pd.merge(df_cp_upd_1, mode_df, on='credit_score_range',
suffixes=('', '_mode'))
df_cp_upd_2.sample(3)
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
483      484           708            0.244153            7599.0   
797      798           725            0.766200           23055.0   
341      342           625            0.647972             611.0   

     credit_inquiries_last_6_months  credit_limit
credit_score_range  \
483                             0.0       40000.0            700-749

797                             2.0       40000.0            700-749

341                             4.0        1500.0            600-649


     credit_limit_mode  
483            40000.0  
797            40000.0  
341             1500.0  
```

```
df_cp_upd_2[df_cp_upd_2.credit_limit.isnull()].sample(3)
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
849      850           787            0.293520           11195.0   
841      842           490            0.555309             249.0   
114      115           619            0.128910             151.0   

     credit_inquiries_last_6_months  credit_limit
credit_score_range  \
849                             3.0           NaN            750-799

841                             1.0           NaN            450-499

114                             1.0           NaN            600-649
```

```
     credit_limit_mode
849            60000.0
841              750.0
114             1500.0
```

Above, we can simple replace NaN value in credit_limit column with credit_limit_mode value.

This value indicates most frequently occuring credit limit for a given credit_score_range. Hence it can be used as a replacement value.

We will create a new copy of the dataframe so that we have reproducibility and access of the older dataframe in this notebook.

```
df_cp_upd_3 = df_cp_upd_2.copy()
df_cp_upd_3['credit_limit'].fillna(df_cp_upd_3['credit_limit_mode'],
inplace=True)
df_cp_upd_3.shape

(1000, 8)

df_cp_upd_3.isnull().sum()

cust_id                        0
credit_score                   0
credit_utilisation             0
outstanding_debt               0
credit_inquiries_last_6_months 0
credit_limit                   0
credit_score_range             0
credit_limit_mode              0
dtype: int64
```

You can now see **ZERO outliers** in credit_limit column which means we successfully got rid of all NULL values.

```
df_cp_upd_3[df_cp_upd_3.cust_id==431]

     cust_id  credit_score  credit_utilisation  outstanding_debt  \
430      431           610            0.741063             628.0

     credit_inquiries_last_6_months  credit_limit
credit_score_range  \
430                             4.0        1500.0            600-649


     credit_limit_mode
430             1500.0
```

Previously customer id **431** had null value in credit_limit. Now it has a valid value.

3 - Handling Outliers

```
# outstanding_debt

df_cp_upd_3.describe()
```

	cust_id	credit_score	credit_utilisation	outstanding_debt
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	500.500000	589.182000	0.498950	9683.597000
std	288.819436	152.284929	0.233139	25255.893671
min	1.000000	300.000000	0.103761	33.000000
25%	250.750000	460.000000	0.293917	221.000000
50%	500.500000	601.500000	0.487422	550.000000
75%	750.250000	738.000000	0.697829	11819.500000
max	1000.000000	799.000000	0.899648	209901.000000

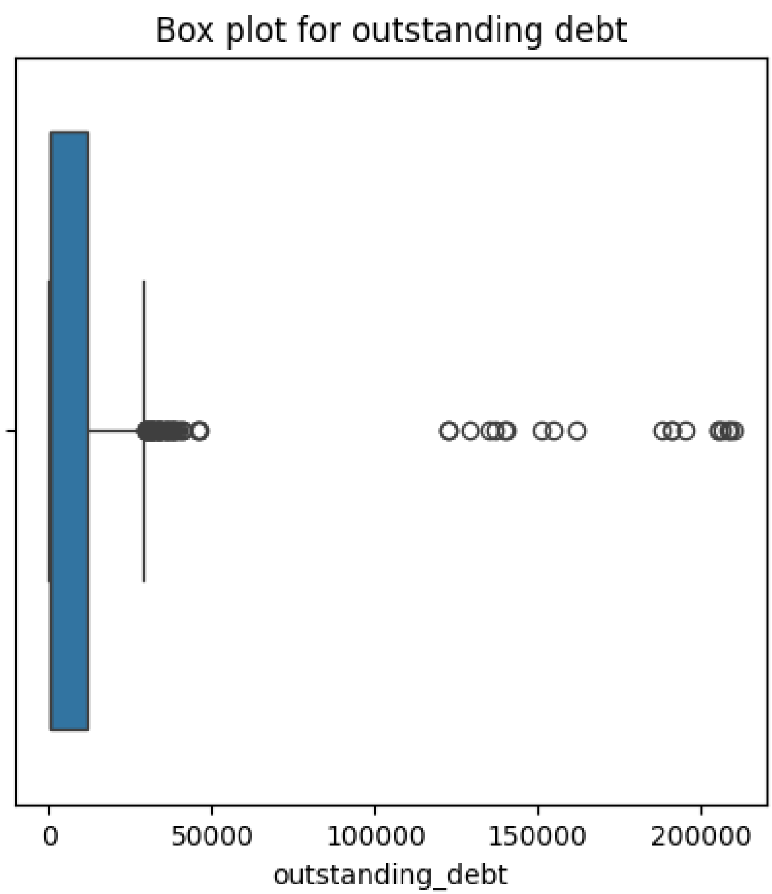
	credit_inquiries_last_6_months	credit_limit	credit_limit_mode
count	1000.000000	1000.00000	1000.000000
mean	1.955000	19733.75000	19912.500000
std	1.414559	24717.43818	24840.914633
min	0.000000	500.00000	500.000000
25%	1.000000	750.00000	750.000000
50%	2.000000	1500.00000	1500.000000
75%	3.000000	40000.00000	40000.000000
max	4.000000	60000.00000	60000.000000

When we observe min and max for various columns, we realize that **outstanding_debt's max > the max of credit_limit**.

Based on the business understanding, we know that the maximum debt that a customer can have is equal to credit limit. They would not be allowed to spend more than their credit limit.

```
# Checking Outliers

plt.figure(figsize=(5, 5))
sns.boxplot(x = df_cp_upd_3['outstanding_debt'])
plt.title('Box plot for outstanding debt')
```

```
Text(0.5, 1.0, 'Box plot for outstanding debt')
```



Instead of using any statistical approach (such as standard deviation or IQR), here too we will use a business knowledge. We will mark any outstanding debt that is greater than credit limit as an outlier

And, we will replace these outliers with credit_limit. We can assume that there was some data processing error due to we got these high numbers and it is ok to replace them with a credit_limit

```
df_cp_upd_3[df_cp_upd_3.outstanding_debt>df_cp_upd_3.credit_limit]
```

	cust_id	credit_score	credit_utilisation	outstanding_debt	\
1	2	587	0.107928	161644.0	
19	20	647	0.439132	205014.0	
25	26	758	0.250811	190838.0	

```
38        39          734          0.573023         122758.0
93        94          737          0.739948         137058.0
204      205          303          0.364360         187849.0
271      272          703          0.446886         154568.0
301      302          722          0.608076         122402.0
330      331          799          0.363420         208898.0
350      351          320          0.285081         150860.0
446      447          754          0.178394         206191.0
544      545          764          0.337769         135112.0
636      637          420          0.323984         140063.0
646      647          498          0.658087         128818.0
698      699          775          0.385100         190717.0
723      724          465          0.658173         140008.0
725      726          737          0.136048         205404.0
730      731          626          0.762245         209901.0
766      767          473          0.611750         195004.0
862      863          792          0.399555         208406.0

     credit_inquiries_last_6_months  credit_limit
credit_score_range  \
1                                2.0        1250.0            550-599

19                               3.0        1500.0            600-649

25                               2.0       60000.0            750-799

38                               3.0       40000.0            700-749

93                               2.0       40000.0            700-749

204                              0.0         500.0            300-449

271                              1.0       40000.0            700-749

301                              4.0       40000.0            700-749

330                              4.0       60000.0            750-799

350                              0.0         500.0            300-449

446                              2.0       60000.0            750-799

544                              2.0       60000.0            750-799

636                              4.0         500.0            300-449

646                              3.0         750.0            450-499

698                              2.0       60000.0            750-799

723                              3.0         750.0            450-499
```

725	4.0	40000.0	700-749
730	2.0	1500.0	600-649
766	1.0	750.0	450-499
862	3.0	60000.0	750-799

	credit_limit_mode
1	1250.0
19	1500.0
25	60000.0
38	40000.0
93	40000.0
204	500.0
271	40000.0
301	40000.0
330	60000.0
350	500.0
446	60000.0
544	60000.0
636	500.0
646	750.0
698	60000.0
723	750.0
725	40000.0
730	1500.0
766	750.0
862	60000.0

```
df_cp_upd_3.loc[df_cp_upd_3['outstanding_debt'] >
df_cp_upd_3['credit_limit'], 'outstanding_debt']
```

```
1      161644.0
19     205014.0
25     190838.0
38     122758.0
93     137058.0
204    187849.0
271    154568.0
301    122402.0
330    208898.0
350    150860.0
446    206191.0
544    135112.0
636    140063.0
646    128818.0
698    190717.0
```

```
723    140008.0
725    205404.0
730    209901.0
766    195004.0
862    208406.0
Name: outstanding_debt, dtype: float64
```

```
df_cp_upd_3.loc[df_cp_upd_3['outstanding_debt'] >
df_cp_upd_3['credit_limit'], 'outstanding_debt'] =
df_cp_upd_3['credit_limit']
```

```
df_cp_upd_3.loc[[204, 544]]
```

```
     cust_id  credit_score  credit_utilisation  outstanding_debt  \
204      205           303            0.364360             500.0
544      545           764            0.337769           60000.0

     credit_inquiries_last_6_months  credit_limit
credit_score_range  \
204                             0.0         500.0            300-449

544                             2.0       60000.0            750-799


     credit_limit_mode
204              500.0
544            60000.0
```

```
df_cp_upd_3[df_cp_upd_3.outstanding_debt > df_cp_upd_3.credit_limit]
```

```
Empty DataFrame
Columns: [cust_id, credit_score, credit_utilisation, outstanding_debt,
credit_inquiries_last_6_months, credit_limit, credit_score_range,
credit_limit_mode]
Index: []
```

```
df_cp_upd_3.describe()
```

```
           cust_id  credit_score  credit_utilisation  outstanding_debt
\
count  1000.000000   1000.000000         1000.000000       1000.000000

mean    500.500000    589.182000            0.498950       6850.084000

std     288.819436    152.284929            0.233139      10683.473561

min       1.000000    300.000000            0.103761         33.000000

25%     250.750000    460.000000            0.293917        221.000000

50%     500.500000    601.500000            0.487422        541.500000
```

```
75%     750.250000    738.000000             0.697829      10924.500000
max    1000.000000    799.000000             0.899648      60000.000000

       credit_inquiries_last_6_months  credit_limit  credit_limit_mode
count                     1000.000000    1000.00000        1000.000000
mean                         1.955000   19733.75000       19912.500000
std                          1.414559   24717.43818       24840.914633
min                          0.000000     500.00000         500.000000
25%                          1.000000     750.00000         750.000000
50%                          2.000000    1500.00000        1500.000000
75%                          3.000000   40000.00000       40000.000000
max                          4.000000   60000.00000       60000.000000
```

4 - Data Exploration : Visualizing Correlation in Credit Score Table

```
df_cust.head(2)
```

```
   cust_id           name  gender   age location       occupation  \
0        1  Manya Acharya  Female  51.0     City  Business Owner
1        2  Anjali Pandey  Female  47.0     City      Consultant

   annual_income marital_status age_group
0       358211.0        Married     49-65
1        65172.0         Single     26-48
```

```
df_cp_upd_3.head(2)
```

```
   cust_id  credit_score  credit_utilisation  outstanding_debt  \
0        1           749            0.585171           19571.0
1        2           587            0.107928            1250.0

   credit_inquiries_last_6_months  credit_limit credit_score_range  \
0                             0.0       40000.0            700-749
1                             2.0        1250.0            550-599

   credit_limit_mode
0            40000.0
1             1250.0
```

```
df_merged = df_cust.merge(df_cp_upd_3, on='cust_id', how='inner')
df_merged.head(2)
```

```
   cust_id           name  gender   age location      occupation  \
0        1  Manya Acharya  Female  51.0     City  Business Owner
1        2  Anjali Pandey  Female  47.0     City      Consultant

   annual_income marital_status age_group  credit_score
credit_utilisation  \
0       358211.0        Married     49-65           749
0.585171
1        65172.0         Single     26-48           587
0.107928

   outstanding_debt  credit_inquiries_last_6_months  credit_limit  \
0           19571.0                             0.0       40000.0
1            1250.0                             2.0        1250.0

  credit_score_range  credit_limit_mode
0            700-749            40000.0
1            550-599             1250.0
```

```
numerical_cols = ['credit_score', 'credit_utilisation',
'outstanding_debt', 'credit_limit', 'annual_income','age']

corr_matx = df_merged[numerical_cols].corr()
corr_matx
```

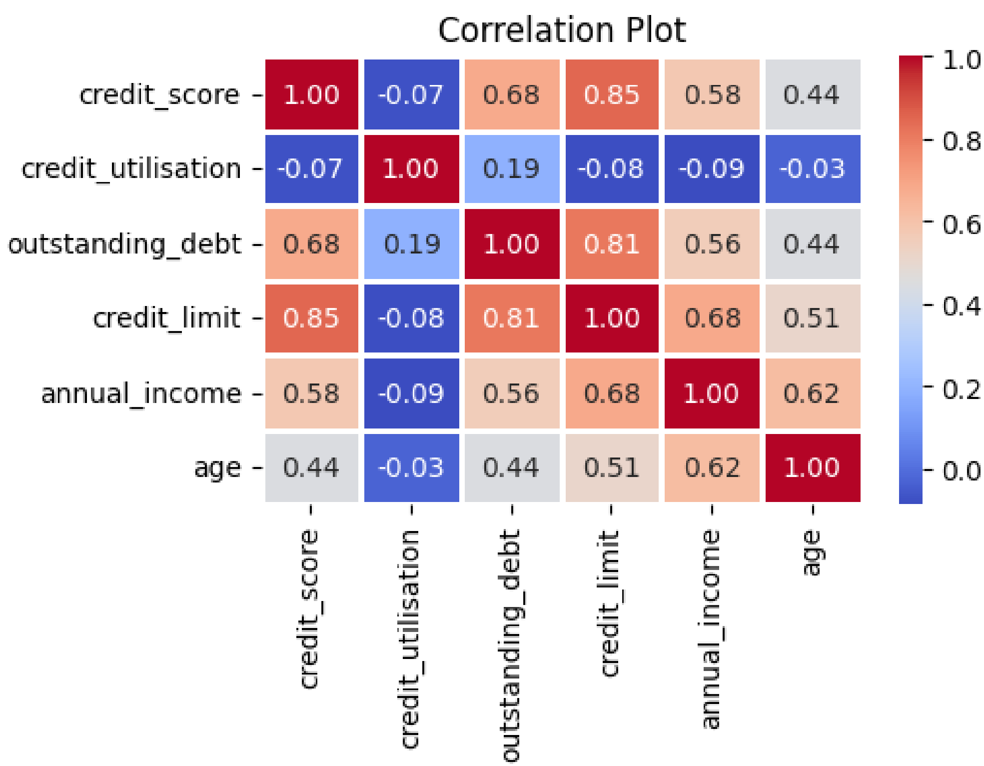
```
                    credit_score  credit_utilisation  outstanding_debt
\
credit_score            1.000000           -0.070445          0.680654

credit_utilisation     -0.070445            1.000000          0.192838

outstanding_debt        0.680654            0.192838          1.000000

credit_limit            0.847952           -0.080493          0.810581

annual_income           0.575685           -0.086816          0.555077

age                     0.444917           -0.027713          0.444301


                    credit_limit  annual_income       age
credit_score            0.847952       0.575685  0.444917
credit_utilisation     -0.080493      -0.086816 -0.027713
outstanding_debt        0.810581       0.555077  0.444301
credit_limit            1.000000       0.684627  0.510993
annual_income           0.684627       1.000000  0.618136
age                     0.510993       0.618136  1.000000
```

Creating a list of numerical columns you're interested in — basically telling Python, "These are the columns I want to study for relationships."

The above code does 3 things:

- df_merged[numerical_cols] — selects just those columns from the DataFrame df_merged.
- .corr() — calculates the correlation between every pair of those columns.
- Stores the result in a new variable called corr_matx.

```
# Creating a heatmap of the correlation matrix

plt.figure(figsize=(5, 3))
sns.heatmap(corr_matx, annot=True, fmt=".2f", cmap='coolwarm',
linewidths=0.8)
plt.title('Correlation Plot')
plt.show()
```

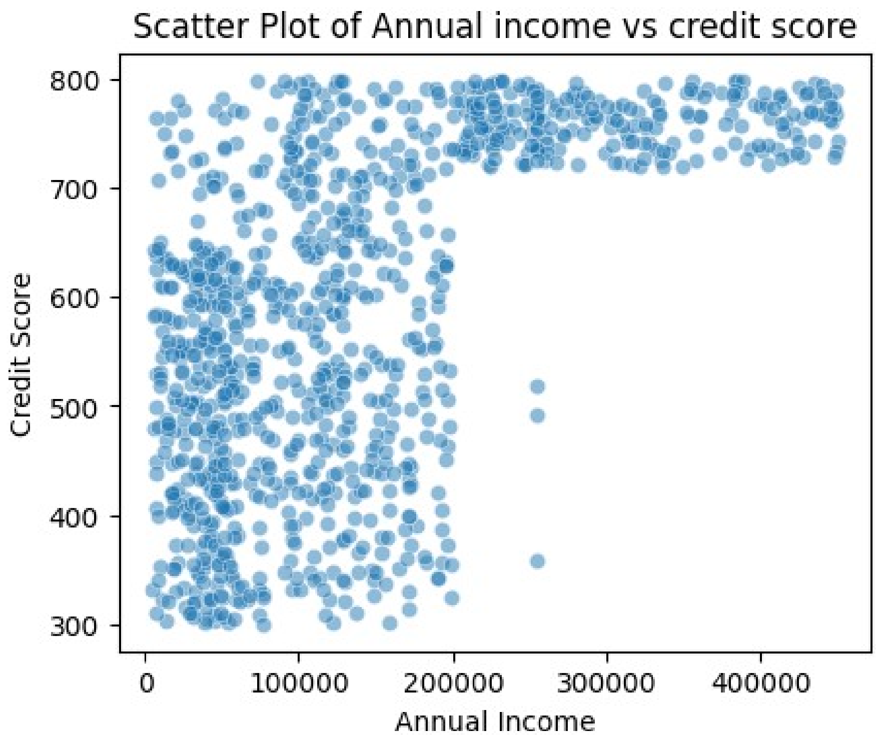


You can see a **high correlation** between credit limit & credit score (~0.85), credit limit & annual income.

This correlation table can be used for further analysis. It shows if one variable has relationship with the other variable

```
# Checking if there is any relation between annual_income and credit
score
```

```
plt.figure(figsize=(5, 4))
sns.scatterplot(x='annual_income', y='credit_score', data=df_merged,
alpha=0.5)
plt.title('Scatter Plot of Annual income vs credit score')
plt.xlabel('Annual Income')
plt.ylabel('Credit Score')
plt.show()
```



No clear pattern observed.

Transactions Table

```
df_trans.head(10)

   tran_id  cust_id   tran_date  tran_amount  platform  \
0        1      705  2023-01-01           63  Flipkart
1        2      385  2023-01-01           99   Alibaba
2        3      924  2023-01-01          471   Shopify
3        4      797  2023-01-01           33   Shopify
4        5      482  2023-01-01           68    Amazon
5        6      527  2023-01-01           38   Shopify
6        7      388  2023-01-01          720   Alibaba
7        8        8  2023-01-01          140   Shopify
8        9      939  2023-01-01          144   Alibaba
```

```
9       10      228  2023-01-01          836      Ebay

         product_category payment_type
0             Electronics      Phonepe
1       Fashion & Apparel  Credit Card
2                  Sports      Phonepe
3       Fashion & Apparel         Gpay
4       Fashion & Apparel  Net Banking
5       Fashion & Apparel   Debit Card
6             Electronics  Credit Card
7      Kitchen Appliances         Gpay
8  Beauty & Personal Care      Phonepe
9             Electronics         Gpay
```

```
df_trans.shape
```

```
(500000, 7)
```

```
df_trans.isnull().sum()
```

```
tran_id                0
cust_id                0
tran_date              0
tran_amount            0
platform            4941
product_category       0
payment_type           0
dtype: int64
```

1 - Handling Null Values

```
df_trans[df_trans.platform.isnull()]
```

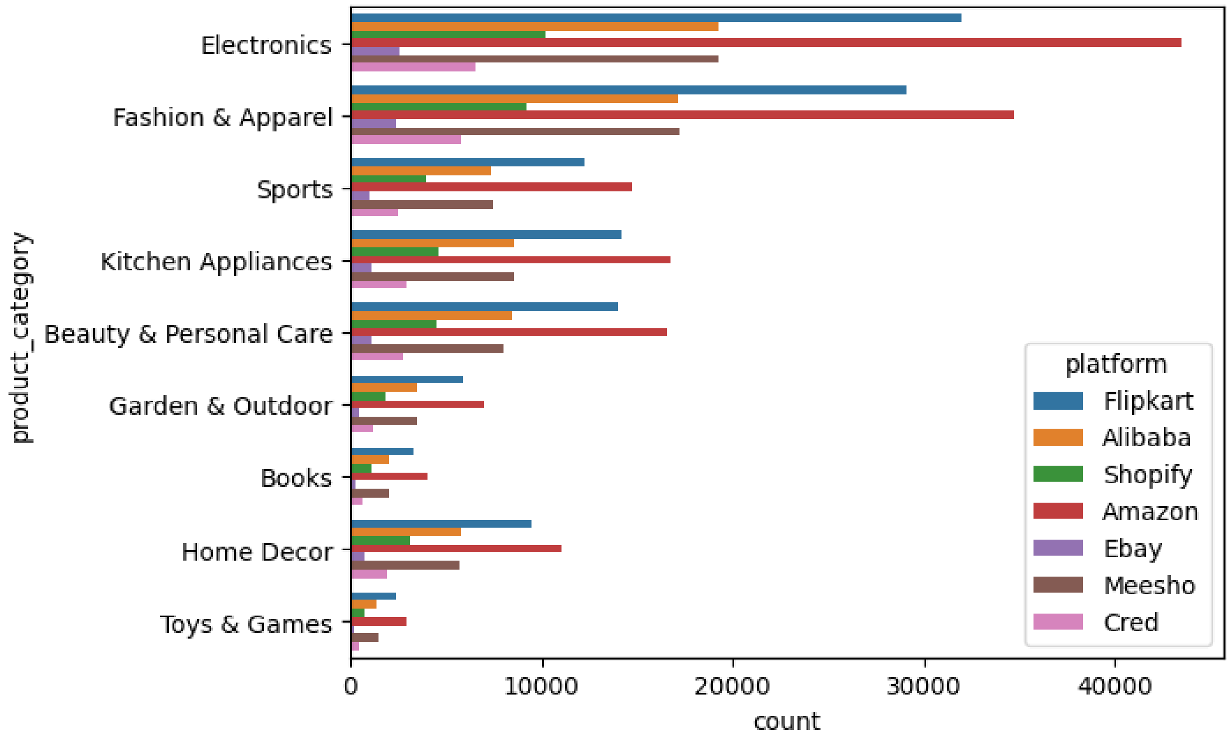
```
        tran_id  cust_id   tran_date  tran_amount platform  
product_category  \
355         356       58  2023-01-01          237     None   
Electronics
418         419      383  2023-01-01          338     None   
Electronics
607         608      421  2023-01-01          700     None   
Electronics
844         845      945  2023-01-01          493     None   
Sports
912         913      384  2023-01-01           85     None   Fashion & 
Apparel
...         ...      ...         ...          ...      ...   
...
499579   499580      924  2023-09-05           31     None   Fashion & 
```

```
Apparel
499646   499647       944  2023-09-05         58445      None  Fashion &
Apparel
499725   499726       620  2023-09-05            15      None
Sports
499833   499834       616  2023-09-05            97      None  Fashion &
Apparel
499997   499998        57  2023-09-05           224      None   Garden &
Outdoor

       payment_type
355     Net Banking
418     Credit Card
607         Phonepe
844     Credit Card
912         Phonepe
...             ...
499579         Gpay
499646      Phonepe
499725  Net Banking
499833  Credit Card
499997      Phonepe

[4941 rows x 7 columns]
```

For above, we cannot replace null values with Mean or Median.

We'll go with **Mode**. And by segmenting the **product_category** we can assign the platform respectively.

```
sns.countplot(y = 'product_category', hue = 'platform', data =
df_trans)

<Axes: xlabel='count', ylabel='product_category'>
```



In the above chart, we can see that in all *product_category*, **Amazon** is the platform that is used the most for making purchases.

For handling null values in **platform**, we can just replace them using **Amazon** as a product platform just because it is used most frequently.

```
df_trans.platform.mode()[0]
```

```
'Amazon'
```

```
df_trans['platform'].fillna(df_trans.platform.mode()[0], inplace=True)
```

```
df_trans.isnull().sum()
```

```
tran_id             0
cust_id             0
tran_date           0
tran_amount         0
platform            0
product_category    0
payment_type        0
dtype: int64
```

2 - Treating Outliers for tran_amount

```
df_trans.describe()
```

```
             tran_id        cust_id   tran_amount
count  500000.000000  500000.000000  500000.00000
mean   250000.500000     501.400428    3225.20733
std    144337.711635     288.641924   13098.74276
min         1.000000       1.000000       0.00000
25%    125000.750000     252.000000      64.00000
50%    250000.500000     502.000000     141.00000
75%    375000.250000     752.000000     397.00000
max    500000.000000    1000.000000   69999.00000
```

We can see transactions with **0 amount**. These seem to be invalid

```
df_trans_zero = df_trans[df_trans.tran_amount == 0]
df_trans_zero.head(3)
```

```
     tran_id  cust_id   tran_date  tran_amount platform
product_category  \
120      121      440  2023-01-01            0   Amazon
Electronics
141      142      839  2023-01-01            0   Amazon
Electronics
517      518      147  2023-01-01            0   Amazon
Electronics

    payment_type
120  Credit Card
141  Credit Card
517  Credit Card
```

```
df_trans_zero.shape
```

```
(4734, 7)
```

```
df_trans_zero.platform.value_counts()
```

```
platform
Amazon    4734
Name: count, dtype: int64
```

```
df_trans_zero.product_category.value_counts()
```

```
product_category
Electronics    4734
Name: count, dtype: int64
```

```
df_trans_zero.payment_type.value_counts()
```

```
payment_type
Credit Card    4734
Name: count, dtype: int64
```

It appears that when **platform=Amazon, product_category=Eletronics and payment_type=Credit Card**, at that time we get all these zero transactions.

We need to find other transactions in this group and find its median to replace these zero values. We are not using mean because we can see some outliers as well in this column.

```
df_trans_1 =
df_trans[(df_trans.platform=='Amazon')&(df_trans.product_category=="El
ectronics")&(df_trans.payment_type=="Credit Card")]
df_trans_1.shape
```

```
(15637, 7)
```

```
df_trans_1[df_trans_1.tran_amount>0]
```

```
        tran_id  cust_id   tran_date  tran_amount platform
product_category  \
109         110      887  2023-01-01          635   Amazon
Electronics
173         174      676  2023-01-01        60439   Amazon
Electronics
190         191      763  2023-01-01          697   Amazon
Electronics
263         264      528  2023-01-01          421   Amazon
Electronics
311         312      936  2023-01-01          537   Amazon
Electronics
...         ...      ...         ...          ...      ...
...
499766   499767      723  2023-09-05          909   Amazon
Electronics
499793   499794      586  2023-09-05          304   Amazon
Electronics
499812   499813      688  2023-09-05          425   Amazon
Electronics
499860   499861      373  2023-09-05          480   Amazon
Electronics
499885   499886      520  2023-09-05          643   Amazon
Electronics

       payment_type
109     Credit Card
```

```
173     Credit Card
190     Credit Card
263     Credit Card
311     Credit Card
...             ...
499766  Credit Card
499793  Credit Card
499812  Credit Card
499860  Credit Card
499885  Credit Card

[10903 rows x 7 columns]
```

```
median_to_replace =
df_trans_1[df_trans_1.tran_amount>0].tran_amount.median()
median_to_replace
```

```
554.0
```

```
df_trans['tran_amount'].replace(0,median_to_replace, inplace=True)

df_trans[df_trans.tran_amount==0]
```

```
Empty DataFrame
Columns: [tran_id, cust_id, tran_date, tran_amount, platform,
product_category, payment_type]
Index: []
```

No 0 values is present in **tran_amount** column

```
df_trans.tran_amount.describe()
```

```
count    500000.000000
mean       3230.452602
std       13097.561071
min           2.000000
25%          66.000000
50%         146.000000
75%         413.000000
max       69999.000000
Name: tran_amount, dtype: float64
```

```
df_trans[df_trans['tran_amount']<1000].describe()
```

	tran_id	cust_id	tran_amount
count	475000.000000	475000.000000	475000.000000
mean	250041.699922	501.375499	240.667608
std	144285.259913	288.606185	244.487110
min	1.000000	1.000000	2.000000
25%	125126.750000	252.000000	63.000000
50%	250100.500000	502.000000	131.000000

```
75%    374928.250000    751.000000    348.000000
max    500000.000000   1000.000000    999.000000
```

```python
Q1, Q3 = df_trans['tran_amount'].quantile([0.25, 0.75])
IQR = Q3 - Q1
lower = Q1 - 2 * IQR      # 2 instead of 1.5 (litlle flexible for business)
upper = Q3 + 2 * IQR

lower, upper
```

```
(-628.0, 1107.0)
```

```python
df_trans[df_trans.tran_amount<upper].tran_amount.max()
```

```
999
```

```python
df_trans[df_trans.tran_amount<upper].tran_amount.min()
```

```
2
```

```python
df_trans_outliers = df_trans[df_trans.tran_amount>=upper]
df_trans_outliers
```

```
        tran_id  cust_id   tran_date  tran_amount  platform  \
26           27      380  2023-01-01        61963   Shopify
49           50      287  2023-01-01        57869    Amazon
94           95      770  2023-01-01        52881      Ebay
104         105      549  2023-01-01        58574  Flipkart
113         114      790  2023-01-01        51669   Shopify
...         ...      ...         ...          ...       ...
499742   499743      868  2023-09-05        55131    Meesho
499888   499889      614  2023-09-05        59679    Meesho
499900   499901      811  2023-09-05        60184  Flipkart
499966   499967      662  2023-09-05        54678    Meesho
499996   499997      569  2023-09-05        53022    Meesho

              product_category payment_type
26      Beauty & Personal Care  Credit Card
49               Toys & Games         Gpay
94          Kitchen Appliances  Credit Card
104          Fashion & Apparel         Gpay
113         Kitchen Appliances  Credit Card
...                        ...          ...
499742       Fashion & Apparel         Gpay
499888       Fashion & Apparel  Net Banking
499900                  Sports   Debit Card
499966                  Sports         Gpay
499996       Fashion & Apparel  Net Banking

[25000 rows x 7 columns]
```

```
df_trans_normal = df_trans[df_trans.tran_amount < upper]
df_trans_normal
```

```
        tran_id  cust_id   tran_date  tran_amount  platform  \
0             1      705  2023-01-01           63  Flipkart
1             2      385  2023-01-01           99   Alibaba
2             3      924  2023-01-01          471   Shopify
3             4      797  2023-01-01           33   Shopify
4             5      482  2023-01-01           68    Amazon
...         ...      ...         ...          ...       ...
499994   499995      679  2023-09-05           59      Ebay
499995   499996      791  2023-09-05           43    Amazon
499997   499998       57  2023-09-05          224    Amazon
499998   499999      629  2023-09-05          538  Flipkart
499999   500000      392  2023-09-05          346    Amazon

              product_category payment_type
0                  Electronics      Phonepe
1             Fashion & Apparel  Credit Card
2                       Sports      Phonepe
3             Fashion & Apparel         Gpay
4             Fashion & Apparel  Net Banking
...                        ...          ...
499994  Beauty & Personal Care         Gpay
499995                   Books      Phonepe
499997        Garden & Outdoor      Phonepe
499998              Home Decor         Gpay
499999      Kitchen Appliances  Net Banking

[475000 rows x 7 columns]
```

```
tran_mean_per_category = df_trans_normal.groupby("product_category")
["tran_amount"].mean()
tran_mean_per_category
```

```
product_category
Beauty & Personal Care     92.167205
Books                      29.553515
Electronics               510.172685
Fashion & Apparel          64.553463
Garden & Outdoor          125.630277
Home Decor                302.487561
Kitchen Appliances        176.773288
Sports                    269.181631
Toys & Games               50.333298
Name: tran_amount, dtype: float64
```

```
df_trans.loc[df_trans_outliers.index]
```

```
    tran_id  cust_id   tran_date  tran_amount  platform  \
26       27      380  2023-01-01        61963   Shopify
```

```
49          50      287  2023-01-01        57869    Amazon
94          95      770  2023-01-01        52881      Ebay
104        105      549  2023-01-01        58574  Flipkart
113        114      790  2023-01-01        51669   Shopify
...        ...      ...         ...          ...       ...
499742  499743      868  2023-09-05        55131    Meesho
499888  499889      614  2023-09-05        59679    Meesho
499900  499901      811  2023-09-05        60184  Flipkart
499966  499967      662  2023-09-05        54678    Meesho
499996  499997      569  2023-09-05        53022    Meesho

              product_category payment_type
26      Beauty & Personal Care  Credit Card
49               Toys & Games         Gpay
94          Kitchen Appliances  Credit Card
104          Fashion & Apparel         Gpay
113         Kitchen Appliances  Credit Card
...                        ...          ...
499742       Fashion & Apparel         Gpay
499888       Fashion & Apparel  Net Banking
499900                  Sports   Debit Card
499966                  Sports         Gpay
499996       Fashion & Apparel  Net Banking

[25000 rows x 7 columns]
```

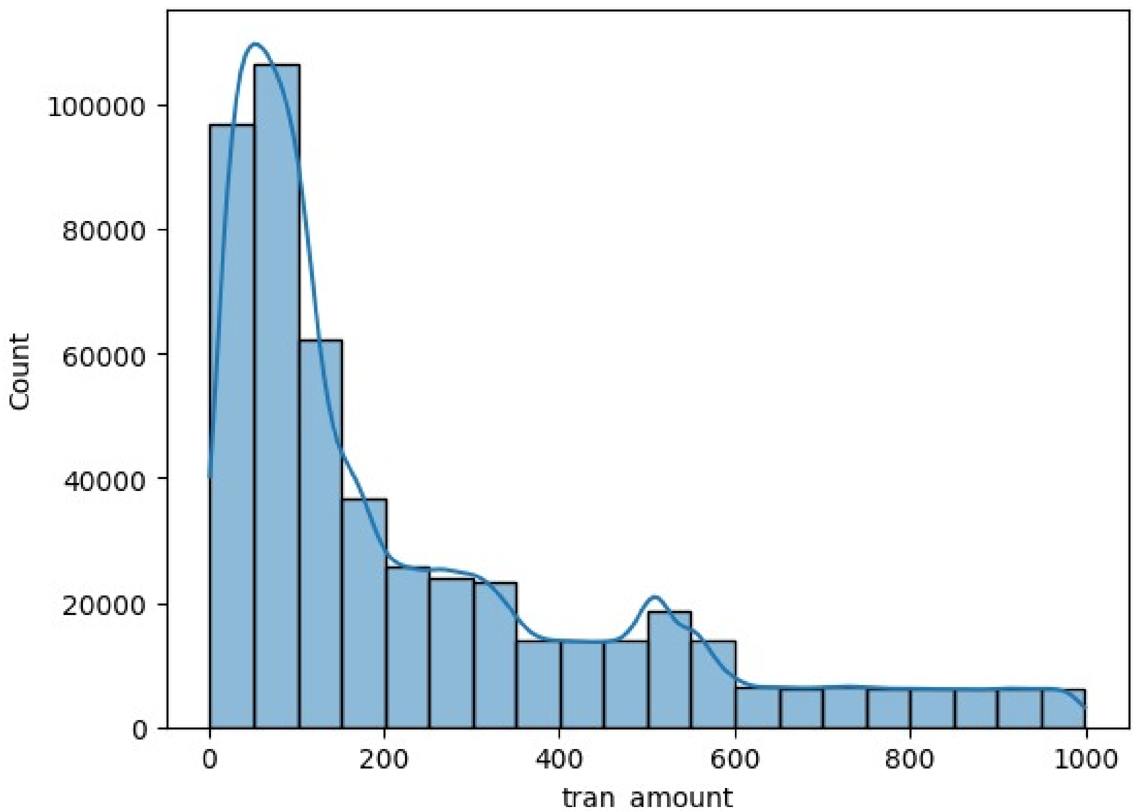
```
df_trans.loc[df_trans_outliers.index, 'tran_amount'] =
df_trans_outliers['product_category'].map(tran_mean_per_category)

df_trans.loc[df_trans_outliers.index]
```

```
        tran_id  cust_id   tran_date  tran_amount  platform  \
26           27      380  2023-01-01    92.167205   Shopify
49           50      287  2023-01-01    50.333298    Amazon
94           95      770  2023-01-01   176.773288      Ebay
104         105      549  2023-01-01    64.553463  Flipkart
113         114      790  2023-01-01   176.773288   Shopify
...         ...      ...         ...          ...       ...
499742   499743      868  2023-09-05    64.553463    Meesho
499888   499889      614  2023-09-05    64.553463    Meesho
499900   499901      811  2023-09-05   269.181631  Flipkart
499966   499967      662  2023-09-05   269.181631    Meesho
499996   499997      569  2023-09-05    64.553463    Meesho

              product_category payment_type
26      Beauty & Personal Care  Credit Card
49               Toys & Games         Gpay
94          Kitchen Appliances  Credit Card
104          Fashion & Apparel         Gpay
113         Kitchen Appliances  Credit Card
```

```
...                             ...           ...
499742        Fashion & Apparel          Gpay
499888        Fashion & Apparel   Net Banking
499900                   Sports    Debit Card
499966                   Sports          Gpay
499996        Fashion & Apparel   Net Banking

[25000 rows x 7 columns]
```

We now got rid of **outliers** from **tran_amount** column.

```
sns.histplot(x='tran_amount', data=df_trans, bins=20, kde=True)

<Axes: xlabel='tran_amount', ylabel='Count'>
```



Above shows the histogram of transactions after the removal of outliers.

We can see that distribution is right skewed. Transaction amount now is **< 1000**

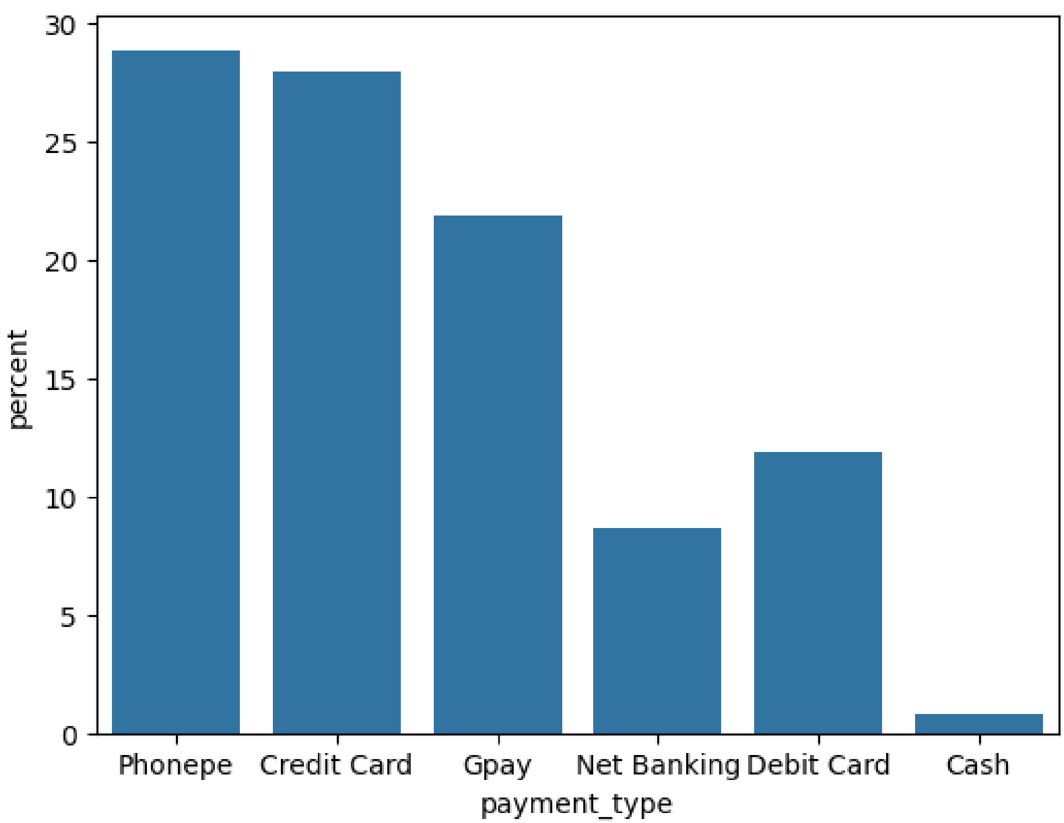
Data Visualization : Payment Type Distribution

```
df_trans.head(3)
```

```
   tran_id  cust_id   tran_date  tran_amount  platform
product_category  \
0        1      705  2023-01-01         63.0  Flipkart
Electronics
1        2      385  2023-01-01         99.0   Alibaba  Fashion &
Apparel
2        3      924  2023-01-01        471.0   Shopify
Sports

  payment_type
0      Phonepe
1  Credit Card
2      Phonepe
```

```
sns.countplot(x=df_trans.payment_type, stat='percent')
```

```
<Axes: xlabel='payment_type', ylabel='percent'>
```



The above plot shows the **Distribution of payment types across age groups**.

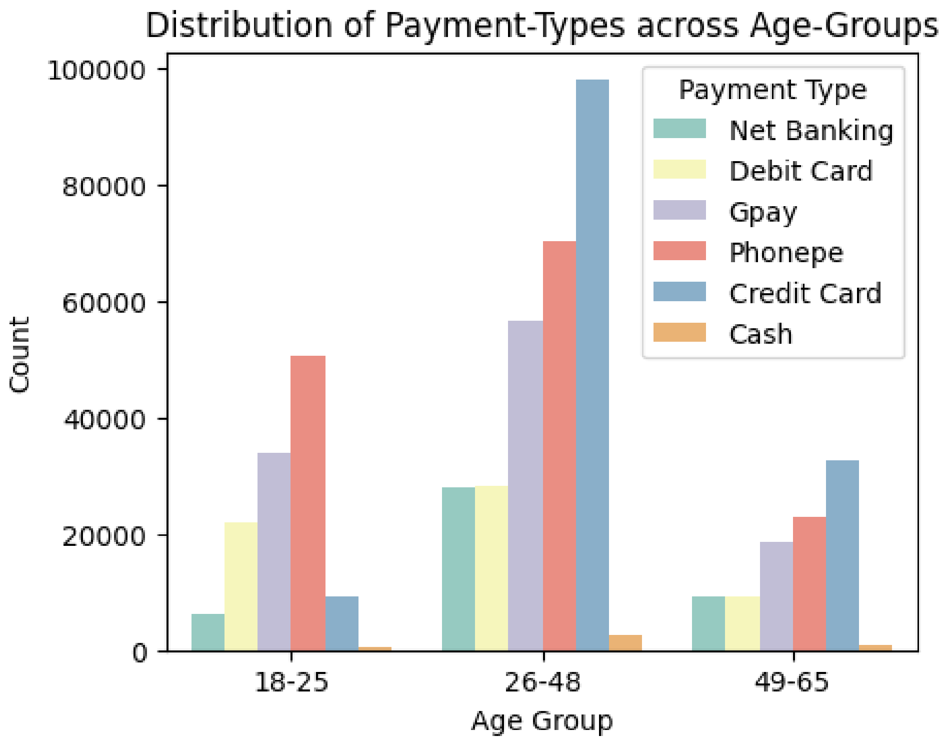
```
df_merged_2 = df_merged.merge(df_trans, on='cust_id', how='inner')
df_merged_2.head(3)
```

```
   cust_id           name  gender   age location      occupation  \
0        1  Manya Acharya  Female  51.0     City  Business Owner
1        1  Manya Acharya  Female  51.0     City  Business Owner
2        1  Manya Acharya  Female  51.0     City  Business Owner

   annual_income marital_status age_group  credit_score  ...  \
0       358211.0        Married     49-65           749  ...
1       358211.0        Married     49-65           749  ...
2       358211.0        Married     49-65           749  ...

   credit_inquiries_last_6_months  credit_limit  credit_score_range  \
0                             0.0       40000.0             700-749
1                             0.0       40000.0             700-749
2                             0.0       40000.0             700-749

   credit_limit_mode tran_id   tran_date  tran_amount platform  \
0            40000.0    1283  2023-01-01         30.0  Shopify
1            40000.0    1382  2023-01-01         96.0   Amazon
2            40000.0    1521  2023-01-01         86.0   Meesho

    product_category payment_type
0  Fashion & Apparel  Net Banking
1             Sports   Debit Card
2    Garden & Outdoor        Gpay

[3 rows x 22 columns]
```

```
df_merged_2.shape
```

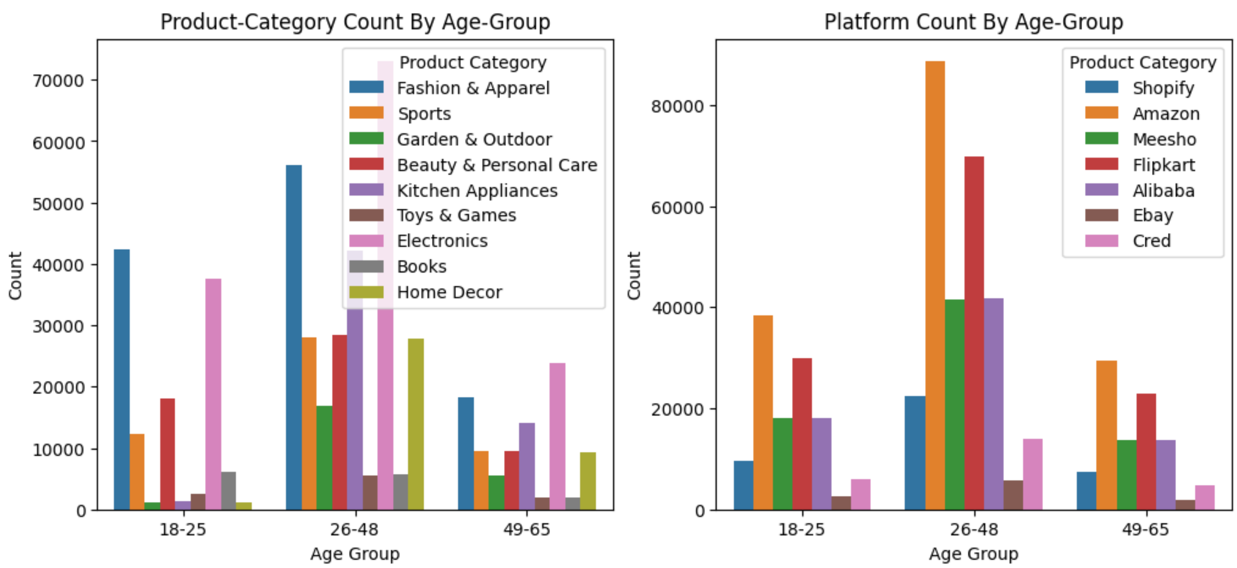
```
(500000, 22)
```

```
plt.figure(figsize=(5, 4))
sns.countplot(x='age_group', hue='payment_type', data=df_merged_2,
palette='Set3')
plt.title('Distribution of Payment-Types across Age-Groups')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Payment Type', loc='upper right')

plt.show()
```



From above analysis, we can see that **age group 18-25** has less exposure to credit cards compared to other groups

```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))

sns.countplot(x='age_group', hue="product_category", data=df_merged_2,
ax=ax1)
ax1.set_title("Product-Category Count By Age-Group")
ax1.set_xlabel("Age Group")
ax1.set_ylabel("Count")
ax1.legend(title="Product Category", loc='upper right')

sns.countplot(x='age_group', hue="platform", data=df_merged_2, ax=ax2)
ax2.set_title("Platform Count By Age-Group")
ax2.set_xlabel("Age Group")
ax2.set_ylabel("Count")
ax2.legend(title="Product Category", loc='upper right')

plt.show()
```



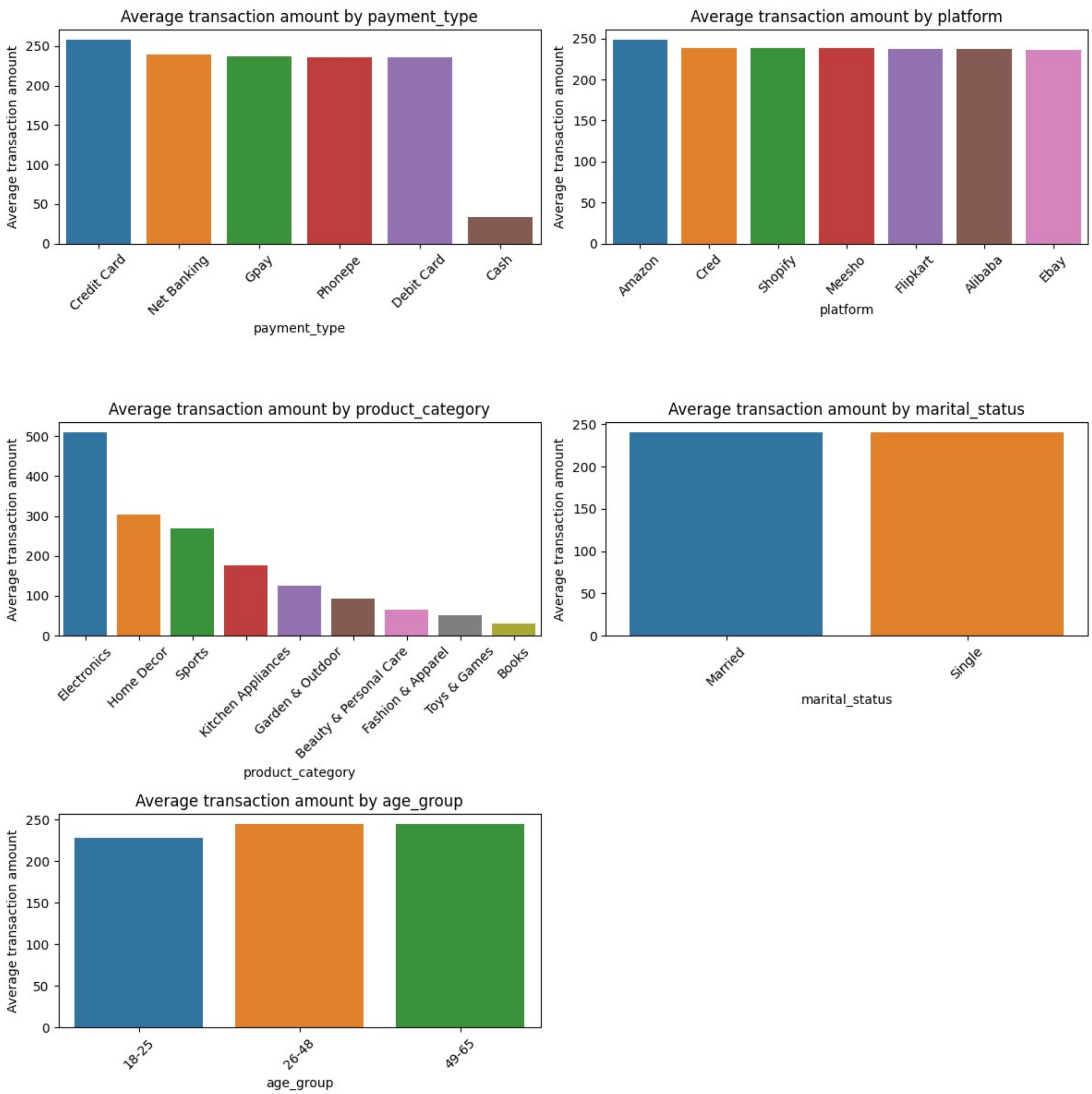
Observations:

- **Top 3** purchasing categories of customers in age group (18 -25) : Electronics, Fashion & Apparel, Beauty & personal care
- **Top 3** platforms : Amazon, Flipkart, Alibaba

Data Visualization : Average Transaction Amount

```
# List of categorical columns
cat_cols = ['payment_type', 'platform', 'product_category',
'marital_status', 'age_group']

num_rows = 3
# Create subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, 4 * num_rows))

# Flatten the axes array to make it easier to iterate
axes = axes.flatten()

# Create subplots for each categorical column
for i, cat_col in enumerate(cat_cols):
    # Calculate the average annual income for each category
    avg_tran_amount_by_category = df_merged_2.groupby(cat_col)
['tran_amount'].mean().reset_index()

    # Sort the data by 'annual_income' before plotting
    sorted_data =
avg_tran_amount_by_category.sort_values(by='tran_amount',
ascending=False)

    sns.barplot(x=cat_col, y='tran_amount', data=sorted_data, ci=None,
```

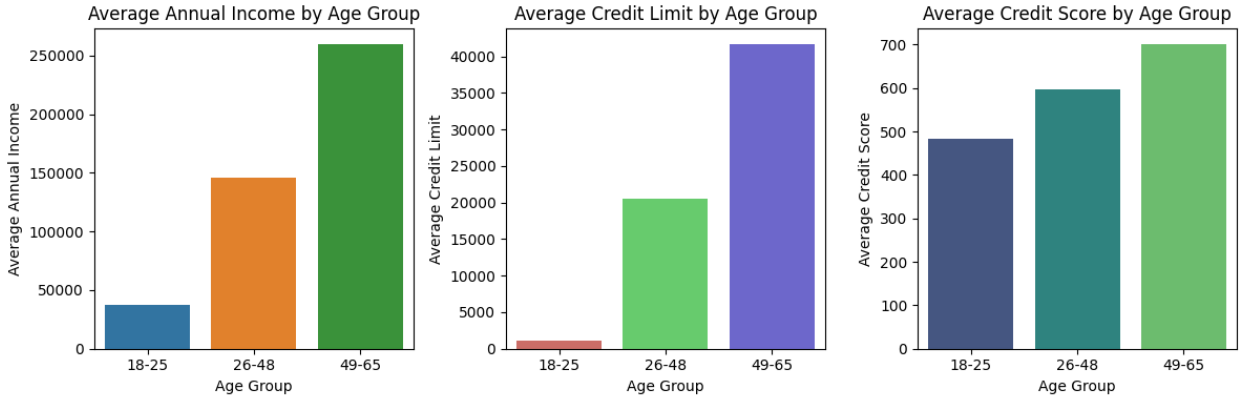
```
ax=axes[i], palette='tab10')
    axes[i].set_title(f'Average transaction amount by {cat_col}')
    axes[i].set_xlabel(cat_col)
    axes[i].set_ylabel('Average transaction amount')

    # Rotate x-axis labels for better readability
    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=45)

# Hide any unused subplots
for i in range(len(cat_cols), len(axes)):
    fig.delaxes(axes[i])
plt.tight_layout()
plt.show()
```



Let us do further analysis on age group to figure out their average income, credit limit, credit score etc

```
# Group the data by age group and calculate the average credit_limit
and credit_score

age_group_metrics = df_merged.groupby('age_group')[['annual_income',
'credit_limit', 'credit_score']].mean().reset_index()
age_group_metrics
```

```
  age_group  annual_income  credit_limit  credit_score
0     18-25   36969.670732   1130.081301    484.451220
1     26-48  145437.104938  20560.846561    597.569665
2     49-65  259786.192513  41699.197861    701.524064

# Create subplots

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))


# Plot 1: Average annual income by age group

sns.barplot(x='age_group', y='annual_income', data=age_group_metrics,
palette='tab10', ax=ax1)
ax1.set_title('Average Annual Income by Age Group')
ax1.set_xlabel('Age Group')
ax1.set_ylabel('Average Annual Income')
ax1.tick_params(axis='x', rotation=0)


# Plot 2: Average Max Credit Limit by Age Group

sns.barplot(x='age_group', y='credit_limit', data=age_group_metrics,
palette='hls', ax=ax2)
ax2.set_title('Average Credit Limit by Age Group')
ax2.set_xlabel('Age Group')
ax2.set_ylabel('Average Credit Limit')
ax2.tick_params(axis='x', rotation=0)


# Plot 3: Average Credit Score by Age Group

sns.barplot(x='age_group', y='credit_score', data=age_group_metrics,
palette='viridis', ax=ax3)
ax3.set_title('Average Credit Score by Age Group')
ax3.set_xlabel('Age Group')
ax3.set_ylabel('Average Credit Score')
ax3.tick_params(axis='x', rotation=0)

plt.tight_layout()
plt.show()
```



Finalize Target Market For a Trial Credit Card Launch

- People with **age group of 18-25** accounts to ~26% of customer base in the data
- Avg annual income of this group is **<50k**
- They **don't have much credit history** which is getting reflected in their credit score and credit limit
- **Usage of credit cards** as payment type is relatively **low** compared to other groups
- **Top 3** most shopping products categories : Electronics, Fashion & Apparel, Beauty & Personal care