AtliQ Bank Credit Card Project

Objective: Analyze customers' transactions and credit profiles to figure out a target group for the launch of AtliQo bank credit card

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import pandoc
```

Data Import

We got the dataset in two formats (1) CSV (2) MySQL. I will show you how to import data from both. You can use only one method out of these two.

Read it from MySQL

In [2]:
```
import mysql.connector

conn = mysql.connector.connect(
    host='localhost',
    user='root',
    passwd='@Sree05092001varshan',
    database='e_master_card'
)

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

Out[2]:

	cust_id	name	gender	age	location	occupation	annual_income	marita
0	1	Manya Acharya	Female	2	City	Business Owner	358211	
1	2	Anjali Pandey	Female	47	City	Consultant	65172	
2	3	Aaryan Chauhan	Male	21	City	Freelancer	22378	

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

Out[3]:

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

In [4]:
```
df_cs = pd.read_sql("SELECT * FROM credit_profiles", conn)
df_cs.head(3)
```

Out[4]:

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

In [5]:
```
# when you are done importing the data, close the connection
conn.close()
```

In [6]:
```
print("Customers data",df_cust.shape)
print("Credit Score data",df_cs.shape)
print("Transactions data",df_trans.shape)
```

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

In [7]:
```
df_cust.head()
```

Out[7]:

	cust_id	name	gender	age	location	occupation	annual_income	marit
0	1	Manya Acharya	Female	2	City	Business Owner	358211	
1	2	Anjali Pandey	Female	47	City	Consultant	65172	
2	3	Aaryan Chauhan	Male	21	City	Freelancer	22378	
3	4	Rudra Bali	Male	24	Rural	Freelancer	33563	
4	5	Advait Malik	Male	48	City	Consultant	39406	

In [8]:
```
df_cs.head()
```

Out[8]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_l
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	

In [9]:
```
df_trans.head()
```

Out[9]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	paym
0	1	705	2023-01-01	63	Flipkart	Electronics	
1	2	385	2023-01-01	99	Alibaba	Fashion & Apparel	C
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	N

Explore Customers Table

In [10]:
```
df_cust.head(3)
```

Out[10]:

	cust_id	name	gender	age	location	occupation	annual_income	marita
0	1	Manya Acharya	Female	2	City	Business Owner	358211	
1	2	Anjali Pandey	Female	47	City	Consultant	65172	
2	3	Aaryan Chauhan	Male	21	City	Freelancer	22378	

In [11]:
```
df_cust.describe()
```

Out[11]:

	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

1. Analyze Income Column

Handle Null Values: Annual income

Now let us check if any of our dataframe columns contain null values

In [12]:
```
df_cust.isnull().sum()
```

Out[12]:
```
cust_id           0
name              0
gender            0
age               0
location          0
occupation        0
annual_income     0
marital_status    0
dtype: int64
```

Ahh.. 50 null values in annual_income. Let's quickly explore those rows

In [13]:
```
df_cust[df_cust.annual_income.isna()].head(4)
```

Out[13]:

cust_id	name	gender	age	location	occupation	annual_income	marital_st

We can handle these null values using different ways,

1. **Remove them**: Since there are 50 of them in a dataframe of 1000, we will not remove them as we don't want to loose some important records
2. **Replace them with mean or median**: It is suggested with use median in the case of income. This is because in an income data there could be outliers and median is more robust to these outliers

3. **Replace them with median per occupation**: Occupation wise median income can vary. It is best to use a median per occupation for replacement

In [14]:
```
occupation_wise_inc_median = df_cust.groupby("occupation")["annual_income"].
occupation_wise_inc_median
```

Out[14]:
```
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
```

In [15]:
```
occupation_wise_inc_median['Artist']
```

Out[15]:
```
np.float64(44915.0)
```

In [16]:
```
# 2. Replace null values in annual_income with the median income of their oc
df_cust['annual_income'] = df_cust.apply(
    lambda row: occupation_wise_inc_median[row['occupation']] if pd.isnull(r
    axis=1
)
```

In [17]:
```
df_cust.iloc[[1,29]]
```

Out[17]:

	cust_id	name	gender	age	location	occupation	annual_income	marit
1	2	Anjali Pandey	Female	47	City	Consultant	65172	
29	30	Aditya Kulkarni	Male	31	Rural	Data Scientist	105583	

Previously records at location 1 and 29 had null annual income. Now you have a median value per occupation

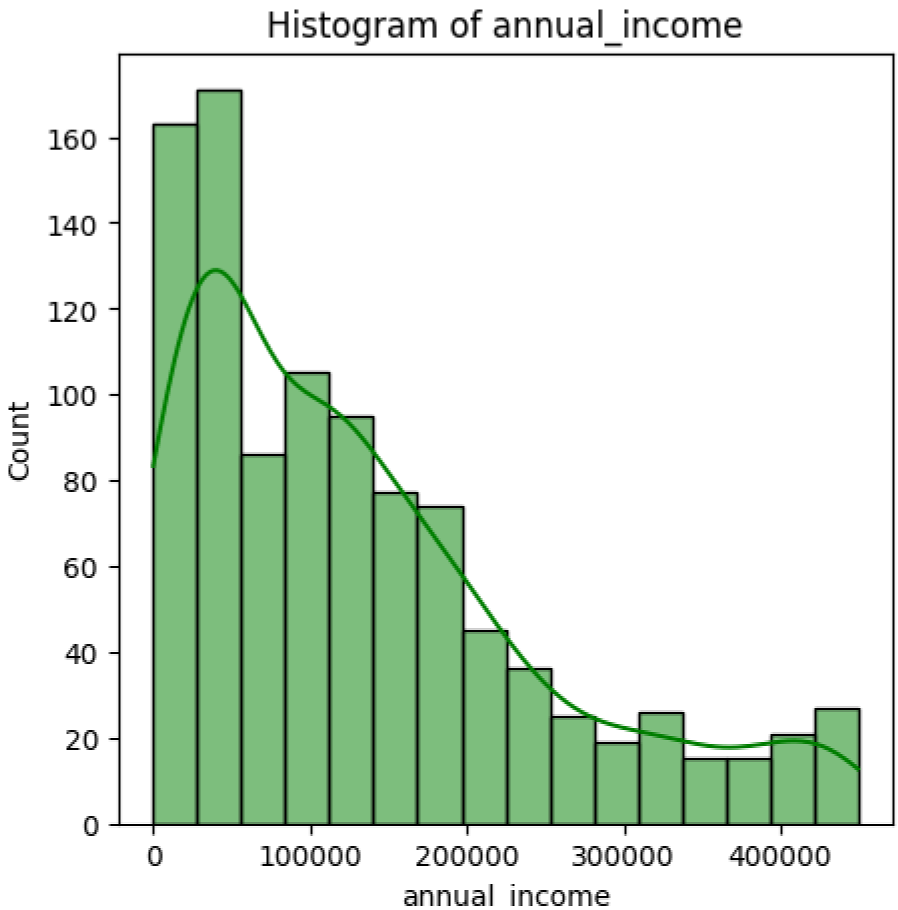
In [18]:
```
df_cust.isnull().sum()
```

Out[18]:
```
cust_id           0
name              0
gender            0
age               0
location          0
occupation        0
annual_income     0
marital_status    0
dtype: int64
```

Awesome 😀 Number of null values in all the columns is zero now! Hurray 🎉

Now that there are no null values, let us view the distribution of annual income

In [19]:
```
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()
```



You can see above that the income distribution is right skewed

Let us now use describe() function to check some quick stats

In [20]:
```
df_cust.describe()
```

Out[20]:

	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

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

In [21]:
```
df_cust.annual_income.describe()
```

Out[21]:
```
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
```

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

In [22]:
```
df_cust['annual_income'].mean(), df_cust['annual_income'].std()
```

Out[22]:
```
(np.float64(132439.799), np.float64(113706.31379289791))
```

In [23]:
```
lower = df_cust['annual_income'].mean() - 3*df_cust['annual_income'].std()
upper = df_cust['annual_income'].mean() + 3*df_cust['annual_income'].std()

lower, upper
```

Out[23]: (np.float64(-208679.14237869374), np.float64(473558.74037869374))

In [24]:
```
df_cust[df_cust['annual_income']>upper]
```

Out[24]:

cust_id	name	gender	age	location	occupation	annual_income	marital_st

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

In [25]:
```
df_cust[df_cust.annual_income<100]
```

Out[25]:

	cust_id	name	gender	age	location	occupation	annual_income	m
14	15	Sanjana Malik	Female	25	Rural	Artist	0	
31	32	Veer Mistry	Male	50	City	Business Owner	50	
82	83	Reyansh Mukherjee	Male	27	City	Freelancer	0	
97	98	Virat Puri	Male	47	Suburb	Business Owner	0	
102	103	Aarav Shah	Male	32	City	Data Scientist	0	
155	156	Kiaan Saxena	Male	24	City	Fullstack Developer	0	
170	171	Advait Verma	Male	52	City	Business Owner	0	
186	187	Samar Sardar	Male	53	City	Consultant	0	
192	193	Ishan Joshi	Male	37	Suburb	Data Scientist	0	
227	228	Advait Mukherjee	Male	48	City	Business Owner	0	
232	233	Aditya Goel	Male	26	City	Freelancer	0	
240	241	Aaryan Bose	Male	24	Suburb	Freelancer	0	
262	263	Vivaan Tandon	Male	53	Suburb	Business Owner	50	
272	273	Kunal Sahani	Male	50	Suburb	Business Owner	0	
275	276	Ananya Bali	Female	47	City	Consultant	0	
312	313	Ritvik Gupta	Male	50	City	Consultant	0	
315	316	Amara Jha	Female	25	City	Data Scientist	0	
316	317	Yuvraj Saxena	Male	47	City	Consultant	50	
333	334	Avani Khanna	Female	29	City	Data Scientist	50	
340	341	Priya Sinha	Female	33	Rural	Fullstack Developer	50	
402	403	Arnav Singh	Male	60	City	Business Owner	0	

	cust_id	name	gender	age	location	occupation	annual_income	m
404	405	Arnav Banerjee	Male	26	City	Data Scientist	0	
409	410	Kiaan Jain	Male	45	Rural	Consultant	0	
440	441	Rudra Bose	Male	36	Suburb	Data Scientist	0	
446	447	Aahan Gambhir	Male	60	City	Business Owner	0	
449	450	Anika Rathod	Female	24	Suburb	Fullstack Developer	0	
461	462	Kunal Nair	Male	33	City	Data Scientist	0	
474	475	Neha Verma	Female	28	City	Data Scientist	0	
502	503	Samar Dewan	Male	38	Suburb	Data Scientist	0	
508	509	Advait Das	Male	55	City	Business Owner	0	
516	517	Rehan Kulkarni	Male	29	Rural	Fullstack Developer	0	
530	531	Aarya Ver	Male	32	City	Business Owner	0	
536	537	Ritvik Patil	Male	33	City	Data Scientist	0	
543	544	Advait Batra	Male	54	City	Consultant	2	
592	593	Priya Gandhi	Female	32	City	Business Owner	50	
599	600	Ishan Goswami	Female	38	City	Consultant	0	
603	604	Kunal Malhotra	Male	25	Suburb	Fullstack Developer	0	
608	609	Kriti Lalwani	Female	25	City	Data Scientist	0	
633	634	Rudra Mehtani	Male	26	City	Data Scientist	2	
634	635	Anaya Dutta	Female	21	City	Freelancer	0	
644	645	Dhruv Das	Male	64	City	Business Owner	0	
648	649	Kunal Rathore	Male	41	City	Consultant	0	
650	651	Gauri Mittal	Female	47	Rural	Consultant	0	

	cust_id	name	gender	age	location	occupation	annual_income	m
664	665	Ayush Khanna	Male	32	Rural	Fullstack Developer	0	
681	682	Arya Jaiswal	Male	37	Suburb	Data Scientist	0	
686	687	Vihaan Jaiswal	Male	40	City	Business Owner	2	
688	689	Dhruv Dewan	Male	26	City	Artist	0	
693	694	Aditi Mehrotra	Female	37	Suburb	Data Scientist	0	
694	695	Rohan Mehta	Male	28	City	Data Scientist	0	
696	697	Ishan Negi	Male	47	City	Consultant	20	
744	745	Swara Kaul	Female	39	City	Data Scientist	0	
784	785	Rohan Jain	Male	27	City	Data Scientist	0	
788	789	Vihaan Singhal	Male	20	City	Fullstack Developer	0	
791	792	Sara Mhatre	Female	38	City	Data Scientist	0	
817	818	Akshay Mehrotra	Male	47	City	Consultant	0	
932	933	Avinash Tiwari	Male	35	City	Data Scientist	0	
955	956	Aahan Gandhi	Male	39	Suburb	Business Owner	0	
956	957	Priya Malik	Female	24	City	Artist	0	
995	996	Manya Vasudeva	Female	26	City	Freelancer	0	
998	999	Amara Rathore	Female	47	City	Business Owner	0	

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

In [26]:
```
occupation_wise_inc_median["Artist"]
```

Out[26]:
```
np.float64(44915.0)
```

In [27]:
```
for index, row in df_cust.iterrows():
    if row["annual_income"] < 100:
        occupation = df_cust.at[index, "occupation"]
        df_cust.at[index, "annual_income"] = occupation_wise_inc_median[occu
```

In [28]:
```
df_cust[df_cust.annual_income<100]
```

Out[28]:

cust_id	name	gender	age	location	occupation	annual_income	marital_st

In [29]:
```
df_cust.loc[[112,256]]
```

Out[29]:

	cust_id	name	gender	age	location	occupation	annual_income	marit
112	113	Yash Sethi	Male	55	City	Business Owner	303207.0	
256	257	Rohan Sethi	Male	28	City	Freelancer	205791.0	

Record at 112 and 256 location had annual income of < 100$. Now you can see it is replaced by a median income per occupation

Data Visualization: Annual Income

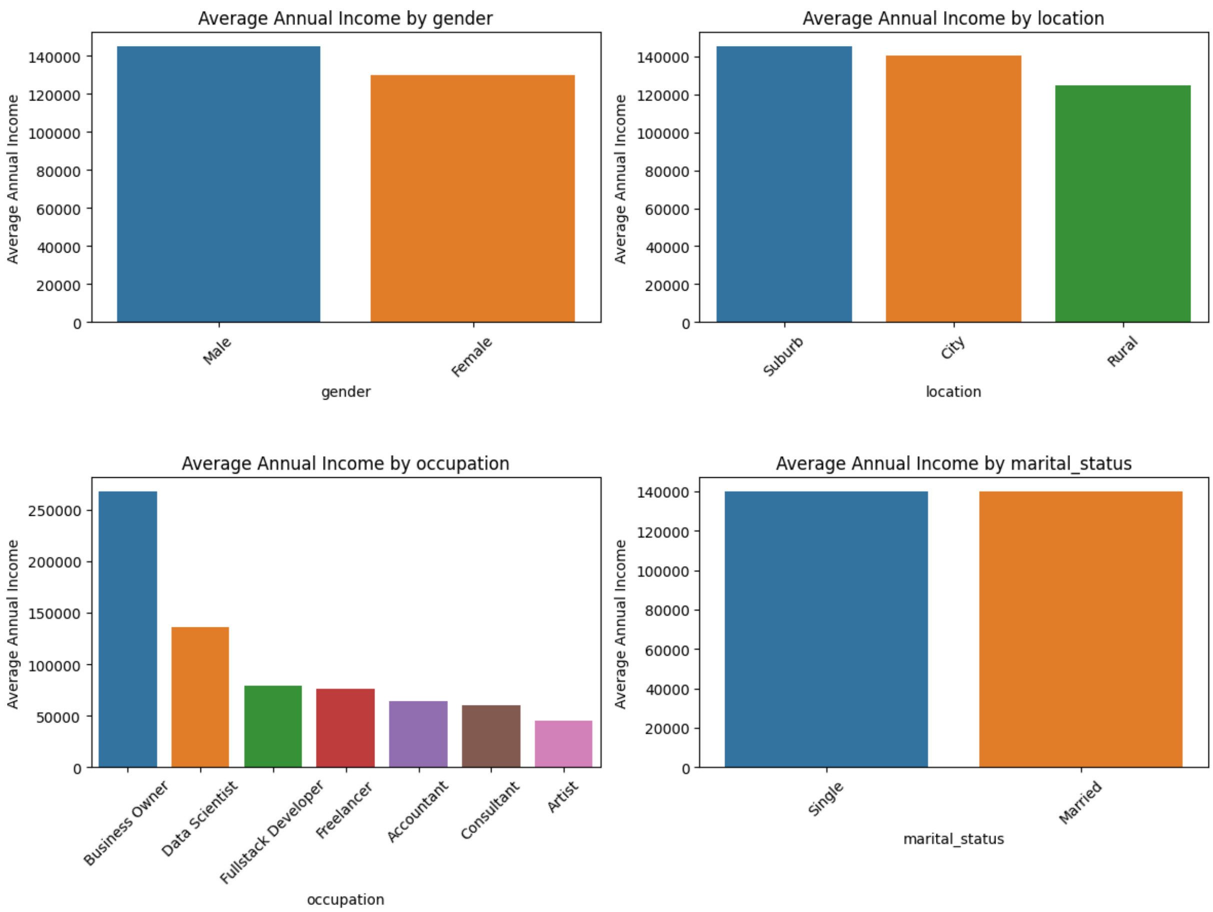
We will explore average income level based on occupation, gender, location and marital status

In [30]:
```
avg_income_per_occupation  = df_cust.groupby("occupation")["annual_income"].
avg_income_per_occupation
```

Out[30]:
```
occupation
Accountant              64123.562500
Artist                  45239.842105
Business Owner         268119.833910
Consultant              59927.257732
Data Scientist         136208.603261
Freelancer              76293.089912
Fullstack Developer     78618.385135
Name: annual_income, dtype: float64
```

In [31]:

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

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_income_by_category = df_cust.groupby(cat_col)['annual_income'].mean(

    # Sort the data by 'annual_income' before plotting
    sorted_data = avg_income_by_category.sort_values(by='annual_income', asc

    sns.barplot(x=cat_col, y='annual_income', data=sorted_data, ci=None, ax=
    axes[i].set_title(f'Average Annual Income by {cat_col}')
    axes[i].set_xlabel(cat_col)
    axes[i].set_ylabel('Average Annual Income')

    # 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()
```



2. Analyze Age Column

Handle Null Values: Age Column

First let us check if there are any NULL values in the Age column

In [32]:
```
df_cust.age.isnull().sum()
```

Out[32]:
```
np.int64(0)
```

No null values are found in age column. This means we don't need to worry about handling them.

In [33]:
```
df_cust.describe()
```

Out[33]:

	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

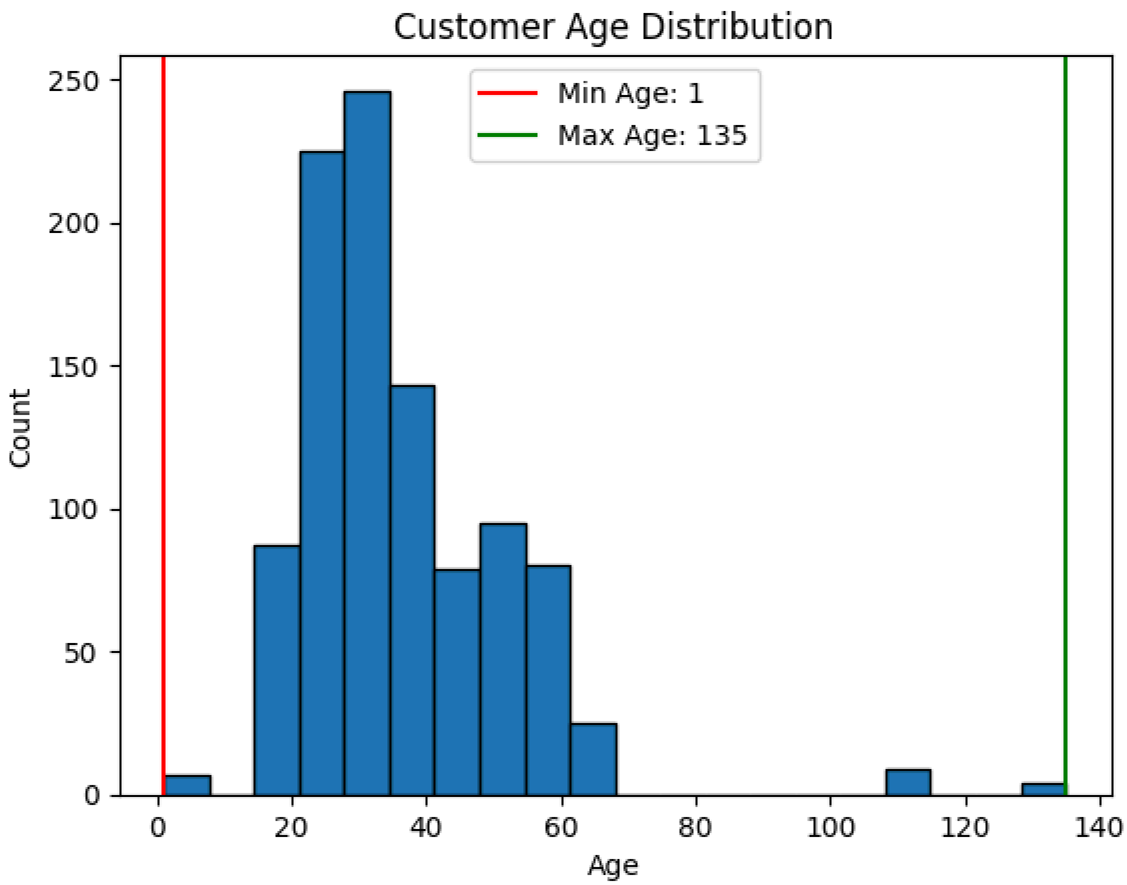
Outlier Treatment: Age

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.

In [34]:
```
min_age = df_cust.age.min()
max_age = df_cust.age.max()

min_age, max_age
```

Out[34]: (np.int64(1), np.int64(135))

In [35]:
```
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()
```



From above we will try to find out all customers above 80 and below 15.

In [36]:
```
df_cust[(df_cust.age<15)|(df_cust.age>80)]
```

Out[36]:

	cust_id	name	gender	age	location	occupation	annual_income	ma
0	1	Manya Acharya	Female	2	City	Business Owner	358211.0	
41	42	Aaryan Shah	Male	110	City	Artist	7621.0	
165	166	Sia Dutta	Female	1	City	Freelancer	39721.0	
174	175	Rohan Sharma	Male	110	City	Freelancer	23723.0	
222	223	Arjun Batra	Male	110	Suburb	Freelancer	210987.0	
277	278	Aarav Tandon	Male	110	City	Consultant	96522.0	
295	296	Ayush Pandey	Male	1	Rural	Accountant	55254.0	
325	326	Virat Goel	Male	110	City	Accountant	61021.0	
610	611	Rehan Verma	Male	135	Rural	Business Owner	444776.0	
692	693	Dhruv Jha	Male	1	City	Business Owner	83045.0	
703	704	Aanya Sharma	Female	110	City	Freelancer	43404.0	
709	710	Anika Verma	Female	110	City	Data Scientist	98417.0	
728	729	Rehan Yadav	Male	135	City	Business Owner	382836.0	
832	833	Ridhi Raj	Female	110	City	Fullstack Developer	95379.0	
845	846	Rohan Jaiswal	Male	1	City	Consultant	20838.0	
855	856	Aanya Taneja	Female	2	City	Fullstack Developer	30689.0	
895	896	Krishna Goswami	Male	1	City	Freelancer	31533.0	
923	924	Kunal Patel	Male	110	City	Freelancer	51629.0	
951	952	Virat Shetty	Male	135	City	Data Scientist	49677.0	
991	992	Arya Dube	Male	135	City	Fullstack Developer	93267.0	

In [37]:

```
outliers = df_cust[(df_cust.age<15)|(df_cust.age>80)]
outliers.shape
```

Out[37]: (20, 8)

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

Possible options,

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

In [38]:
```
df_cust.age.median()
```

Out[38]: np.float64(32.0)

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

In [39]:
```
outliers.head(3)
```

Out[39]:

	cust_id	name	gender	age	location	occupation	annual_income	mar
0	1	Manya Acharya	Female	2	City	Business Owner	358211.0	
41	42	Aaryan Shah	Male	110	City	Artist	7621.0	
165	166	Sia Dutta	Female	1	City	Freelancer	39721.0	

As you can see, for business owners median age is 49 whereas artists have youngest age

We will calculte median per occupation and then use that for replacing outliers

In [40]:
```
median_age_per_occupation = df_cust.groupby('occupation')['age'].median()
median_age_per_occupation
```

Out[40]:
```
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
```

In [41]:
```
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]
```

In [42]:
```
df_cust[(df_cust.age<15)|(df_cust.age>80)]
```

Out[42]:

cust_id	name	gender	age	location	occupation	annual_income	marital_st

In [43]:
```
df_cust.age.describe()
```

Out[43]:
```
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
```

As you can see above, now we don't have any outliers left. min age is 18 and max is 64

In [44]:
```
df_cust.head()
```

Out[44]:

	cust_id	name	gender	age	location	occupation	annual_income	marit
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	
2	3	Aaryan Chauhan	Male	21.0	City	Freelancer	22378.0	
3	4	Rudra Bali	Male	24.0	Rural	Freelancer	33563.0	
4	5	Advait Malik	Male	48.0	City	Consultant	39406.0	

Data Visualization: Age Column

In [45]:
```
# Define the bin edges and labels
bin_edges = [17, 25, 48, 65]  # Adjust as needed
bin_labels = ['18-25', '26-48', '49-65']

# Use the cut function to bin and label the age column
pd.cut(df_cust['age'], bins=bin_edges, labels=bin_labels)
```

Out[45]:
```
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']
```

In [46]:
```
# Define the bin edges and labels
bin_edges = [17, 25, 48, 65]  # Adjust as needed
bin_labels = ['18-25', '26-48', '49-65']

# Use the cut function to bin and label the age column
df_cust['age_group'] = pd.cut(df_cust['age'], bins=bin_edges, labels=bin_lab
```

In [47]:
```
df_cust.head()
```

Out[47]:

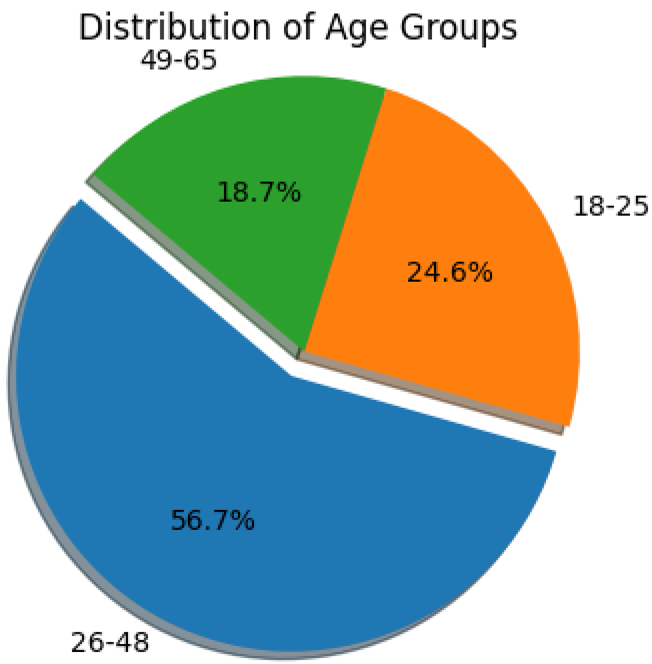
	cust_id	name	gender	age	location	occupation	annual_income	marit
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	
2	3	Aaryan Chauhan	Male	21.0	City	Freelancer	22378.0	
3	4	Rudra Bali	Male	24.0	Rural	Freelancer	33563.0	
4	5	Advait Malik	Male	48.0	City	Consultant	39406.0	

In [48]:
```
df_cust['age_group'].value_counts(normalize=True)*100
```

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

In [49]:
```
# 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),
```

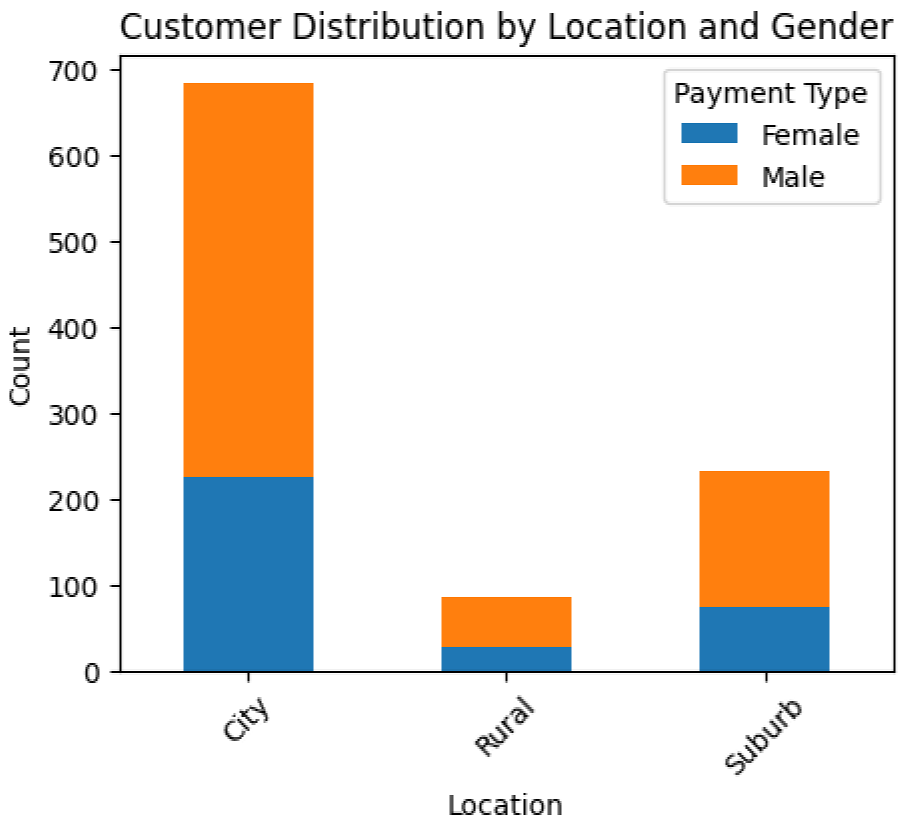
```
    autopct='%1.1f%%',
    shadow=True,
    startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circl
plt.title('Distribution of Age Groups')
plt.show()
```



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

3. Analyze Gender and Location Distribution

In [50]:
```
customer_location_gender = df_cust.groupby(['location', 'gender']).size().ur

# Create a stacked bar chart to visualize the distribution of payment types
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()
```



Explore Credit Score Table

In [51]:
```
df_cs.head()
```

Out[51]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_l
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	

Data Cleaning Step 1: Remove Duplicates

In [52]:
```
df_cs.shape
```

Out[52]: (1004, 6)

Hmmm... there are 1004 rows in this dataframe whereas customers dataframe had only 1000. There might be invalid or duplicate data in df_cs

In [53]:
```
df_cs['cust_id'].nunique()
```

Out[53]:
```
1000
```

In [54]:
```
df_cs.duplicated('cust_id')
```

Out[54]:
```
0       False
1       False
2       False
3       False
4       False
        ...
999     False
1000    False
1001    False
1002    False
1003    False
Length: 1004, dtype: bool
```

In [55]:
```
df_cs[df_cs.duplicated('cust_id', keep=False)]
```

Out[55]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
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	

In [56]:
```
df_cs_clean_1 = df_cs.drop_duplicates(subset='cust_id', keep="last")
df_cs_clean_1.shape
```

Out[56]:
```
(1000, 6)
```

In [57]:
```
df_cs_clean_1[df_cs_clean_1.duplicated('cust_id', keep=False)]
```

Out[57]:

cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_la

df_cs_clean_1 looks clean now after cleaning duplicates.

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

Data Cleaning Step 2: Handle Null Values

In [58]:
```
df_cs_clean_1.isnull().sum()
```

Out[58]:
```
cust_id                              0
credit_score                         0
credit_utilisation                   0
outstanding_debt                     0
credit_inquiries_last_6_months       0
credit_limit                        65
dtype: int64
```

Ahh... look at credit_limit. It has a bunch of null values. we need to clean them up! 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. Let's explore a few things here!

In [59]:
```
df_cs_clean_1[df_cs_clean_1.credit_limit.isnull()]
```

Out[59]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
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	

65 rows × 6 columns

In [60]:
```
df_cs_clean_1['credit_limit'].unique()
```

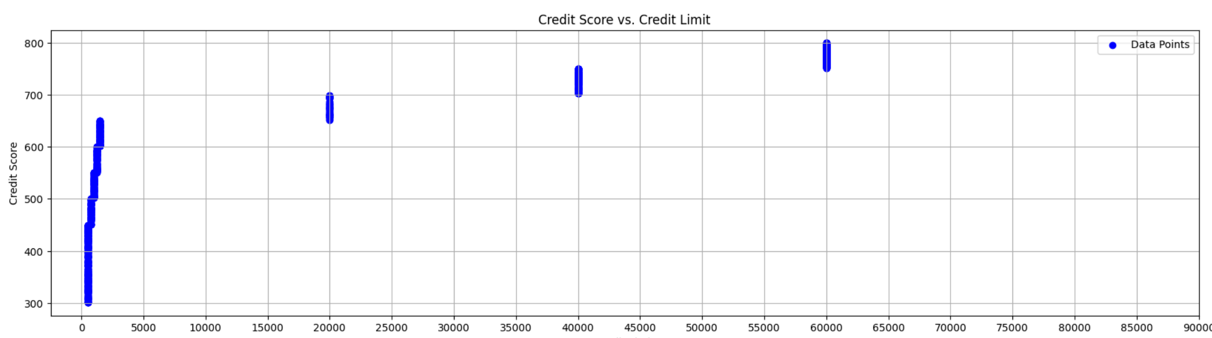
Out[60]:
```
array([40000.,  1250.,  1000.,   500.,   750.,    nan,  1500., 60000.,
       20000.])
```

Credit limit has only few unique values. Let's check the count for each of these unique values

In [61]:
```
df_cs_clean_1['credit_limit'].value_counts()
```

Out[61]:
```
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
```

In [62]:
```
# Looking at scatter plot for credit score vs credit_limit again (after hand
# Create a scatter plot
plt.figure(figsize=(20, 5))
plt.scatter(df_cs_clean_1['credit_limit'], df_cs_clean_1['credit_score'], c=

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

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

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



Here 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$) where as a score between 650 to 700 is getting around 20000. Score between 700 to 750 is getting around 40K etc.

In [63]:
```
# Define bin ranges
bin_ranges = [300, 450, 500, 550, 600, 650, 700, 750, 800]

# Create labels for the bins
bin_labels = [f'{start}-{end-1}' for start, end in zip(bin_ranges, bin_range

# Use pd.cut to assign data to bins
df_cs_clean_1['credit_score_range'] = pd.cut(df_cs_clean_1['credit_score'],
```

In [64]:
```
df_cs_clean_1.head()
```

Out[64]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_l
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	

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

In [65]:
```
df_cs_clean_1[['credit_score','credit_score_range', 'credit_limit']].head(3)
```

Out[65]:

	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

In [66]:
```
df_cs_clean_1[df_cs_clean_1['credit_score_range']=="750-799"]
```

Out[66]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiri
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	

213 rows × 7 columns

In [67]:
```
df_cs_clean_1[df_cs_clean_1['credit_score_range']=="300-449"]
```

Out[67]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
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	

237 rows × 7 columns

Above you 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.

In [68]:

```
mode_df = df_cs_clean_1.groupby('credit_score_range')['credit_limit'].agg(la
mode_df
```

Out[68]:

	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

In [69]:

```
df_cs_clean_1[df_cs_clean_1.credit_limit.isnull()].sample(3)
```

Out[69]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
301	302	722	0.608076	122402.0	
856	853	497	0.873269	416.0	
690	687	736	0.738382	17882.0	

In [70]:
```
# Merge the mode values back with the original DataFrame
df_cs_clean_2 = pd.merge(df_cs_clean_1, mode_df, on='credit_score_range', su
df_cs_clean_2.sample(3)
```

Out[70]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
20	21	381	0.714710	307.0	
196	197	516	0.215087	146.0	
300	301	489	0.575409	357.0	

In [71]:
```
df_cs_clean_2[df_cs_clean_2.credit_limit.isnull()].sample(3)
```

Out[71]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
325	326	599	0.791918	501.0	
258	259	427	0.339428	136.0	
908	909	479	0.487555	320.0	

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

In [72]:
```
df_cs_clean_3 = df_cs_clean_2.copy()
df_cs_clean_3['credit_limit'].fillna(df_cs_clean_3['credit_limit_mode'], inp
df_cs_clean_3.shape
```

Out[72]:
```
(1000, 8)
```

In [73]:
```
df_cs_clean_3.isnull().sum()
```

Out[73]:
```
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. Hurray!

In [74]:
```
df_cs_clean_3[df_cs_clean_3.cust_id==117]
```

Out[74]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
116	117	372	0.604427	252.0	

Previously customer id 5 had null value in credit_limit. Now it has a valid value

Data Cleaning Step 3: Handle Outliers: outstanding_debt

In [75]:
```
df_cs_clean_3.describe()
```

Out[75]:

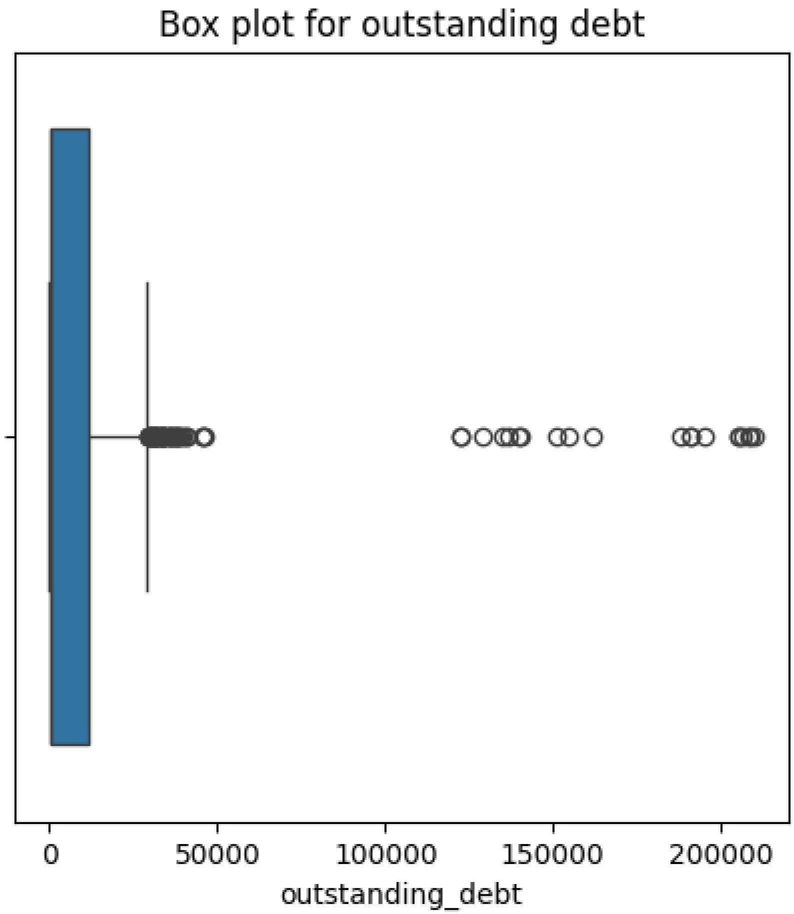
	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_i
count	1000.000000	1000.000000	1000.000000	1000.000000	
mean	500.500000	589.182000	0.498950	9683.597000	
std	288.819436	152.284929	0.233139	25255.893671	
min	1.000000	300.000000	0.103761	33.000000	
25%	250.750000	460.000000	0.293917	221.000000	
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	

When we observe min and max for various columns, we realize that outstanding_debt's max is greater than 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. Let's see how many such cases are present in our dataset

Visualizing outliers

In [76]:
```
plt.figure(figsize=(5, 5))
sns.boxplot(x=df_cs_clean_3['outstanding_debt'])
plt.title('Box plot for outstanding debt')
```

Out[76]: 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

In [77]:
```
df_cs_clean_3[df_cs_clean_3.outstanding_debt>df_cs_clean_3.credit_limit]
```

Out[77]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquirie
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	

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

In [78]:

```
df_cs_clean_3.loc[df_cs_clean_3['outstanding_debt'] > df_cs_clean_3['credit_
```

Out[78]:
```
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
```

In [79]:
```
df_cs_clean_3.loc[df_cs_clean_3['outstanding_debt'] > df_cs_clean_3['credit_
```

In [80]:
```
df_cs_clean_3.loc[[55,66]]
```

Out[80]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_
55	56	429	0.198374	74.0	
66	67	429	0.229638	69.0	

In [81]:
```
df_cs_clean_3[df_cs_clean_3.outstanding_debt>df_cs_clean_3.credit_limit]
```

Out[81]:

cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_la

All outliers in column outstanding_debt are now GONE. Hurray 🙌🏻😀

In [82]:
```
df_cs_clean_3.describe()
```

Out[82]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_i
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	

Data Exploration: Visualizing Correlation in Credit Score Table

In [83]:

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

Out[83]:

	cust_id	name	gender	age	location	occupation	annual_income	marita
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	

In [84]:

```
df_cs_clean_3.head(2)
```

Out[84]:

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

In [85]:

```
df_merged = df_cust.merge(df_cs_clean_3, on='cust_id', how='inner')
df_merged.head(2)
```

Out[85]:

	cust_id	name	gender	age	location	occupation	annual_income	marita
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	

In [86]:

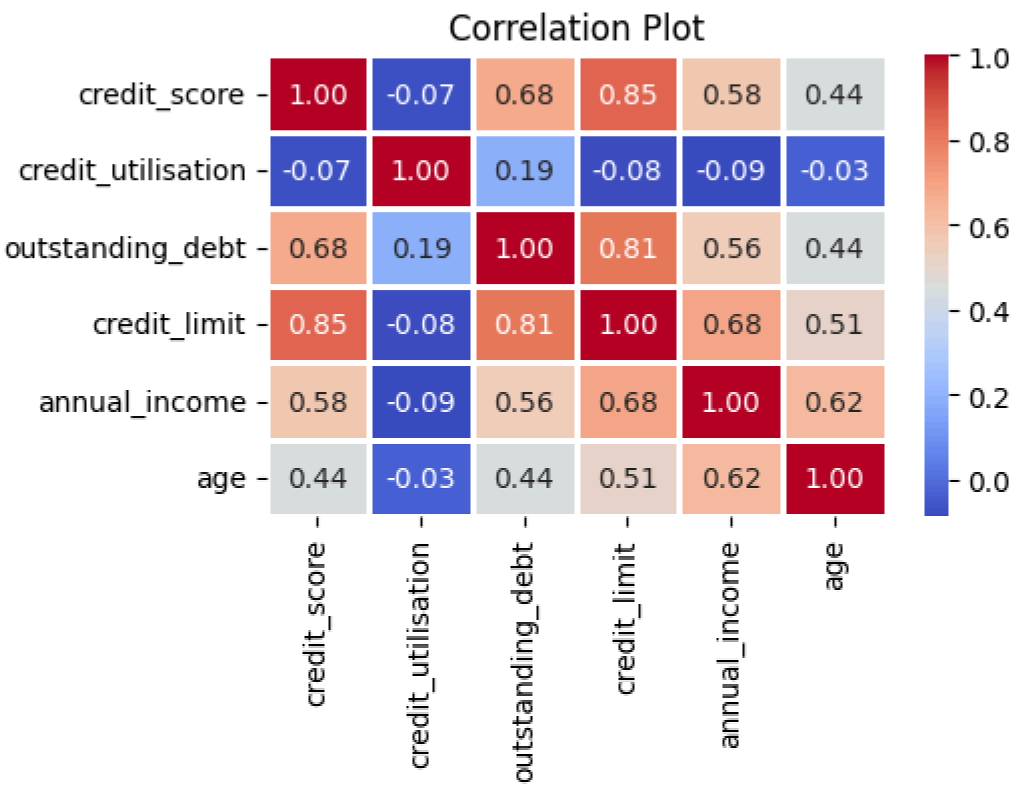
```
numerical_cols = ['credit_score', 'credit_utilisation', 'outstanding_debt',

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

Out[86]:

	credit_score	credit_utilisation	outstanding_debt	credit_lim
credit_score	1.000000	-0.070445	0.680654	0.84795
credit_utilisation	-0.070445	1.000000	0.192838	-0.08049
outstanding_debt	0.680654	0.192838	1.000000	0.81058
credit_limit	0.847952	-0.080493	0.810581	1.00000
annual_income	0.575685	-0.086816	0.555077	0.68462
age	0.444917	-0.027713	0.444301	0.51099

In [87]:

```
# Create a heatmap of the correlation matrix
plt.figure(figsize=(5, 3))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', line
plt.title('Correlation Plot')
plt.show()
```



You can see a high correlation between credit limit and credit score (~0.85)

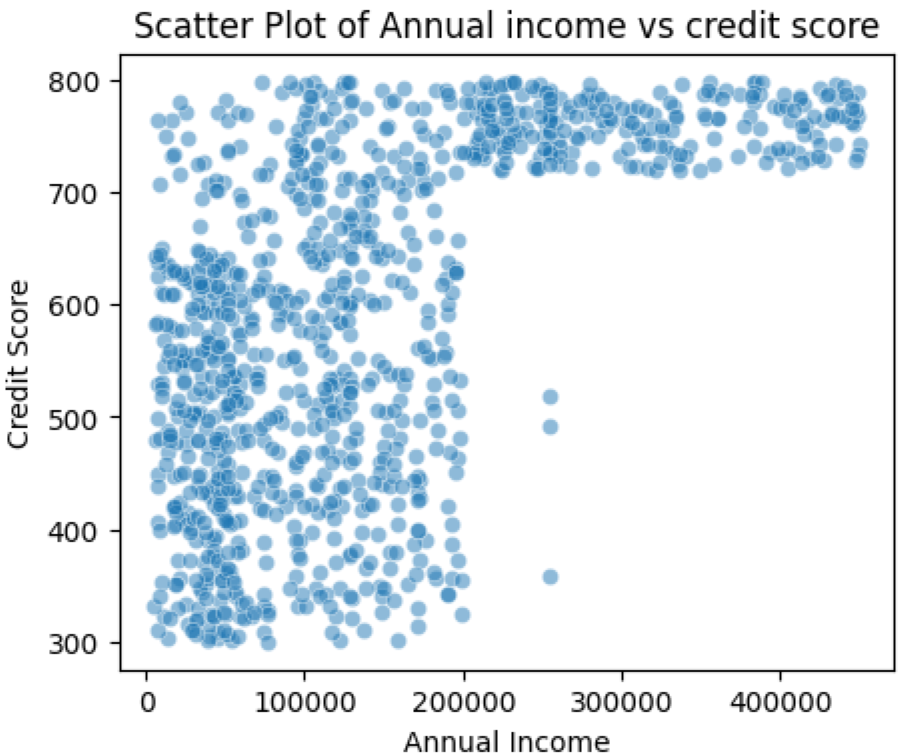
Also credit limit and annual income has a high correlation.

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

In [88]:

```
# Just looking if there is any relation between annual_income and credit sco
plt.figure(figsize=(5, 4))
sns.scatterplot(x='annual_income', y='credit_score', data=df_merged, alpha=0
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

In [89]:
```
df_trans.head(2)
```

Out[89]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payn
0	1	705	2023-01-01	63	Flipkart	Electronics	
1	2	385	2023-01-01	99	Alibaba	Fashion & Apparel	C

In [90]:
```
df_trans.shape
```

Out[90]: (500000, 7)

Data Cleaning Step 1: Handle NULL Values: platform column

In [91]:
```
df_trans.isnull().sum()
```

Out[91]:
```
tran_id                0
cust_id                0
tran_date              0
tran_amount            0
platform            4941
product_category       0
payment_type           0
dtype: int64
```

platform has a lot of null values. Let's check them further

In [92]:
```
df_trans[df_trans.platform.isnull()]
```

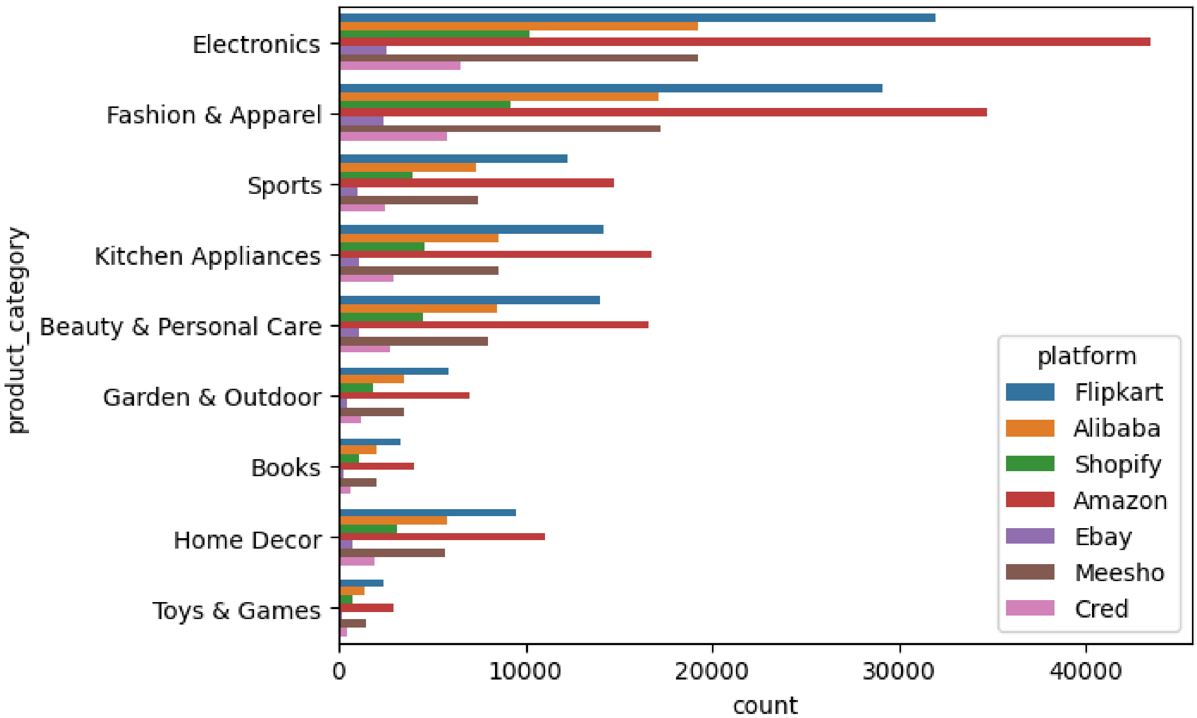
Out[92]:

	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

4941 rows × 7 columns

In [93]:
```
sns.countplot(y='product_category', hue='platform', data=df_trans)
```

Out[93]:
```
<Axes: xlabel='count', ylabel='product_category'>
```



In the above chart, you can see that in all product categories Amazon is the platform that is used the most for making purchases. For handling null values in platform may be we can just replace them using "Amazon" as a product platform just because it is used most frequently

In [94]:
```
df_trans.platform.mode()
```

Out[94]:
```
0    Amazon
Name: platform, dtype: object
```

In [95]:
```
df_trans.platform.mode()[0]
```

Out[95]:
```
'Amazon'
```

In [96]:
```
df_trans['platform'].fillna(df_trans.platform.mode()[0], inplace=True)
```

In [97]:
```
df_trans.isnull().sum()
```

Out[97]:
```
tran_id             0
cust_id             0
tran_date           0
tran_amount         0
platform            0
product_category    0
payment_type        0
dtype: int64
```

Once again we got rid of NULL values □□□□

Data Cleaning Step 2: Treat Outliers: tran_amount

In [98]:
```
df_trans.describe()
```

Out[98]:

	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

In [99]:
```
df_trans_zero = df_trans[df_trans.tran_amount==0]
df_trans_zero.head(3)
```

Out[99]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	pa
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	

In [100…
```
df_trans_zero.shape
```

Out[100…
```
(4734, 7)
```

In [101…
```
df_trans_zero.platform.value_counts()
```

Out[101…
```
platform
Amazon    4734
Name: count, dtype: int64
```

In [102…
```
df_trans_zero[['platform','product_category','payment_type']].value_counts()
```

Out[102…
```
platform  product_category  payment_type
Amazon    Electronics       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

In [103…
```
df_trans_1 = df_trans[(df_trans.platform=='Amazon')&(df_trans.product_catego
df_trans_1.shape
```

Out[103…
```
(15637, 7)
```

In [104…
```
df_trans_1[df_trans_1.tran_amount>0]
```

Out[104…

	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

10903 rows × 7 columns

In [105…
```
median_to_replace = df_trans_1[df_trans_1.tran_amount>0].tran_amount.median(
median_to_replace
```

Out[105…
```
np.float64(554.0)
```

In [106…
```
df_trans['tran_amount'].replace(0,median_to_replace, inplace=True)
```

In [107…
```
df_trans[df_trans.tran_amount==0]
```

Out[107…

tran_id	cust_id	tran_date	tran_amount	platform	product_category	payme

As you can see above, no zero values are left in tran_amount column

In [108…
```
df_trans.tran_amount.describe()
```

Out[108…
```
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
```

In [109…
```
df_trans[df_trans['tran_amount']<1000].describe()
```

Out[109…

	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

In [110…
```
Q1, Q3 = df_trans['tran_amount'].quantile([0.25, 0.75])
IQR = Q3 - Q1
lower = Q1 - 2 * IQR
upper = Q3 + 2 * IQR

lower, upper
```

Out[110…
```
(-628.0, 1107.0)
```

In [111…
```
df_trans[df_trans.tran_amount<upper].tran_amount.max()
```

Out[111…
```
np.int64(999)
```

In [112…
```
df_trans[df_trans.tran_amount>upper].tran_amount.min()
```

Out[112…
```
np.int64(50000)
```

In [113…
```
df_trans_outliers = df_trans[df_trans.tran_amount>=upper]
df_trans_outliers
```

Out[113…

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

25000 rows × 7 columns

In [114…

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

Out[114…

	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
...	...	...	...	...	...	...
499994	499995	679	2023-09-05	59	Ebay	Beauty & Personal Care
499995	499996	791	2023-09-05	43	Amazon	Books
499997	499998	57	2023-09-05	224	Amazon	Garden & Outdoor
499998	499999	629	2023-09-05	538	Flipkart	Home Decor
499999	500000	392	2023-09-05	346	Amazon	Kitchen Appliances

475000 rows × 7 columns

In [115…

```
tran_mean_per_category = df_trans_normal.groupby("product_category")["tran_a
tran_mean_per_category
```

Out[115…

```
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
```

In [116…

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

Out[116…

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

25000 rows × 7 columns

In [117…

```
df_trans.loc[df_trans_outliers.index, 'tran_amount'] = df_trans_outliers['pr
```

In [118…

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

Out[118…

	tran_id	cust_id	tran_date	tran_amount	platform	product_category
26	27	380	2023-01-01	92.167205	Shopify	Beauty & Personal Care
49	50	287	2023-01-01	50.333298	Amazon	Toys & Games
94	95	770	2023-01-01	176.773288	Ebay	Kitchen Appliances
104	105	549	2023-01-01	64.553463	Flipkart	Fashion & Apparel
113	114	790	2023-01-01	176.773288	Shopify	Kitchen Appliances
...	...	...	...	...	...	...
499742	499743	868	2023-09-05	64.553463	Meesho	Fashion & Apparel
499888	499889	614	2023-09-05	64.553463	Meesho	Fashion & Apparel
499900	499901	811	2023-09-05	269.181631	Flipkart	Sports
499966	499967	662	2023-09-05	269.181631	Meesho	Sports
499996	499997	569	2023-09-05	64.553463	Meesho	Fashion & Apparel

25000 rows × 7 columns

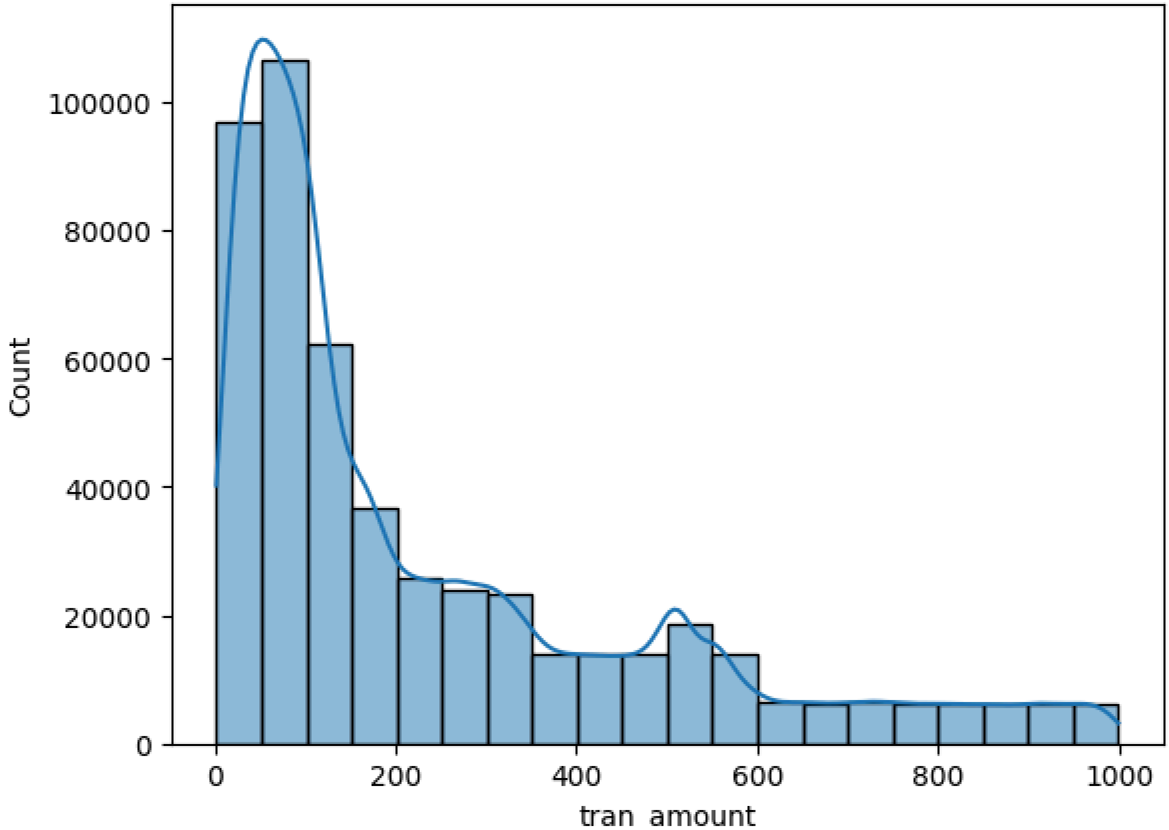
You can now see that we got rid of outliers from tran_amount column. Great job folks □□□□

In [119…

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

Out[119…

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



Above shows the histogram of transactions after the removal of outliers. You can see that distribution is right skewed. Transaction amount now is less than 1000

Data Visualization: Payment Type Distribution

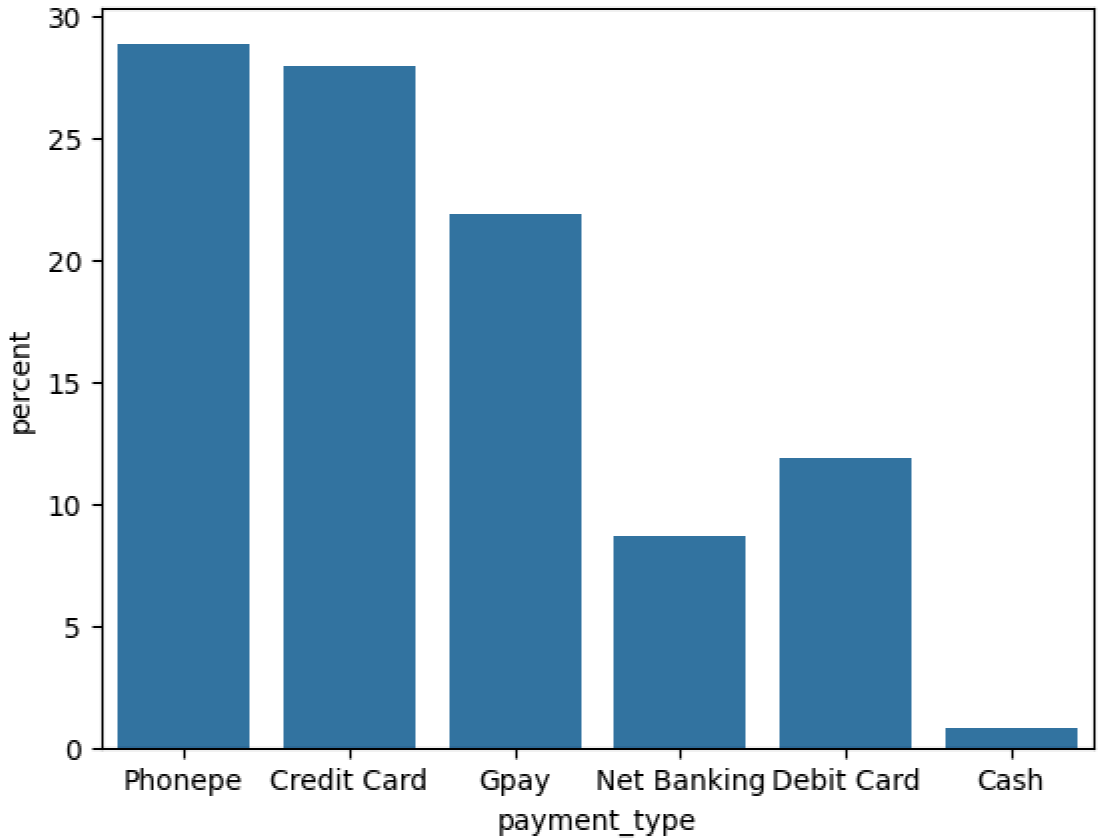
In [120…
```
df_trans.head(3)
```

Out[120…

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

In [121…
```
sns.countplot(x=df_trans.payment_type, stat='percent')
```

Out[121…
```
<Axes: xlabel='payment_type', ylabel='percent'>
```



Distribution of payment types across age groups

In [122…
```
df_merged_2 = df_merged.merge(df_trans, on='cust_id', how='inner')
df_merged_2.head(3)
```

Out[122…

	cust_id	name	gender	age	location	occupation	annual_income	marita
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
1	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	
2	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	

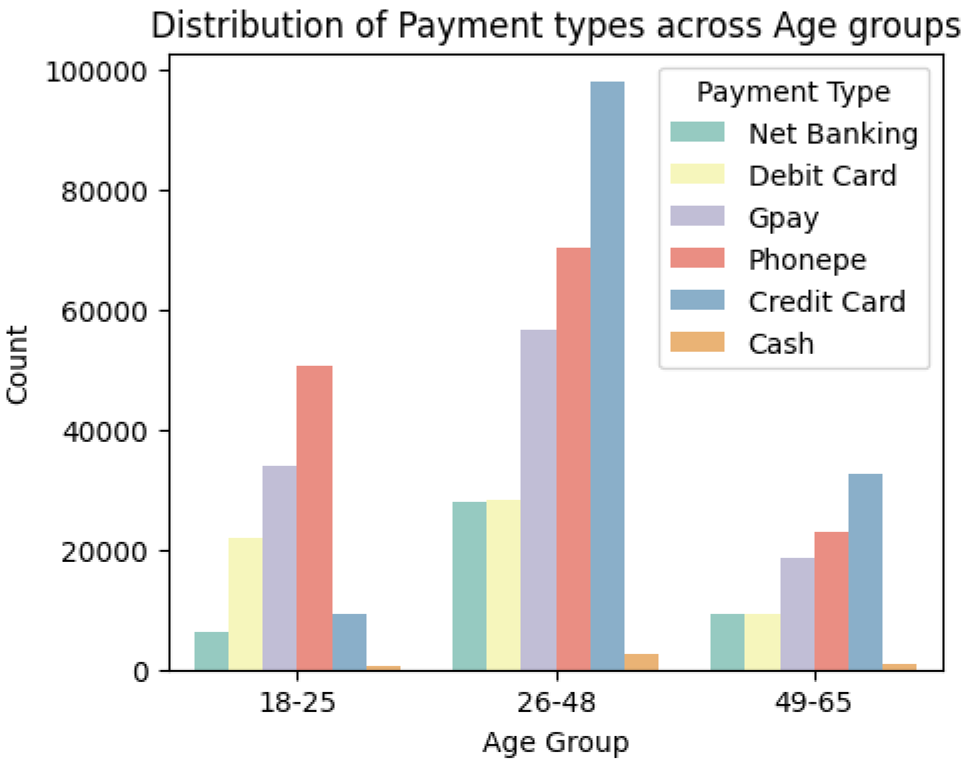
3 rows × 22 columns

In [123…
```
df_merged_2.shape
```

Out[123…
```
(500000, 22)
```

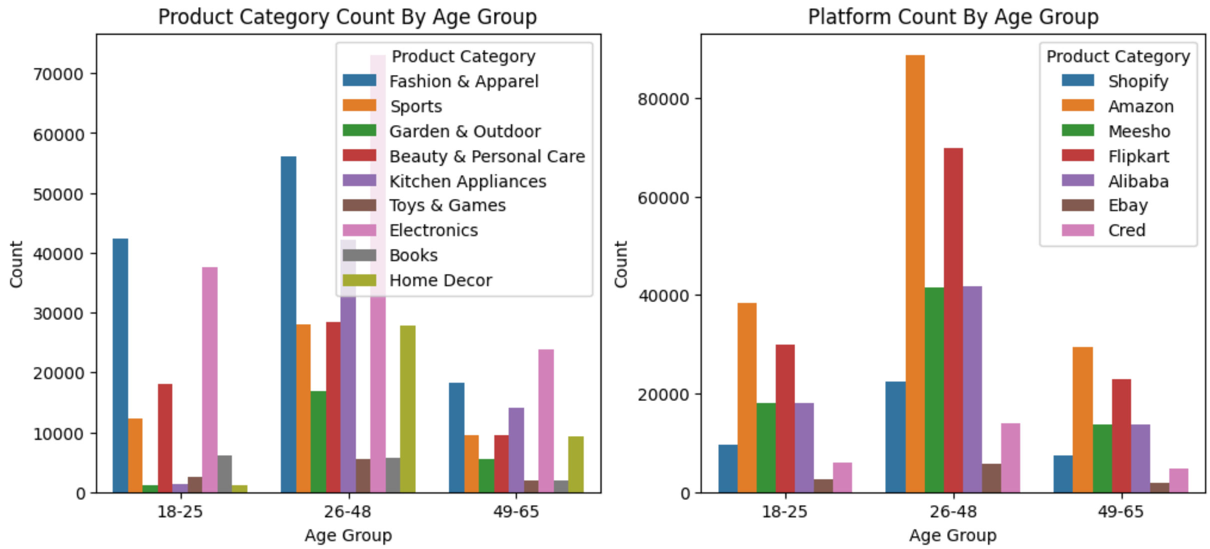
In [124…
```
plt.figure(figsize=(5, 4))
sns.countplot(x='age_group', hue='payment_type', data=df_merged_2, palette='
plt.title('Distribution of Payment types across Age groups')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Payment Type', loc='upper right')
```

```
plt.show()
```



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

In [125…
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))

sns.countplot(x='age_group', hue="product_category", data=df_merged_2, ax=ax
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:

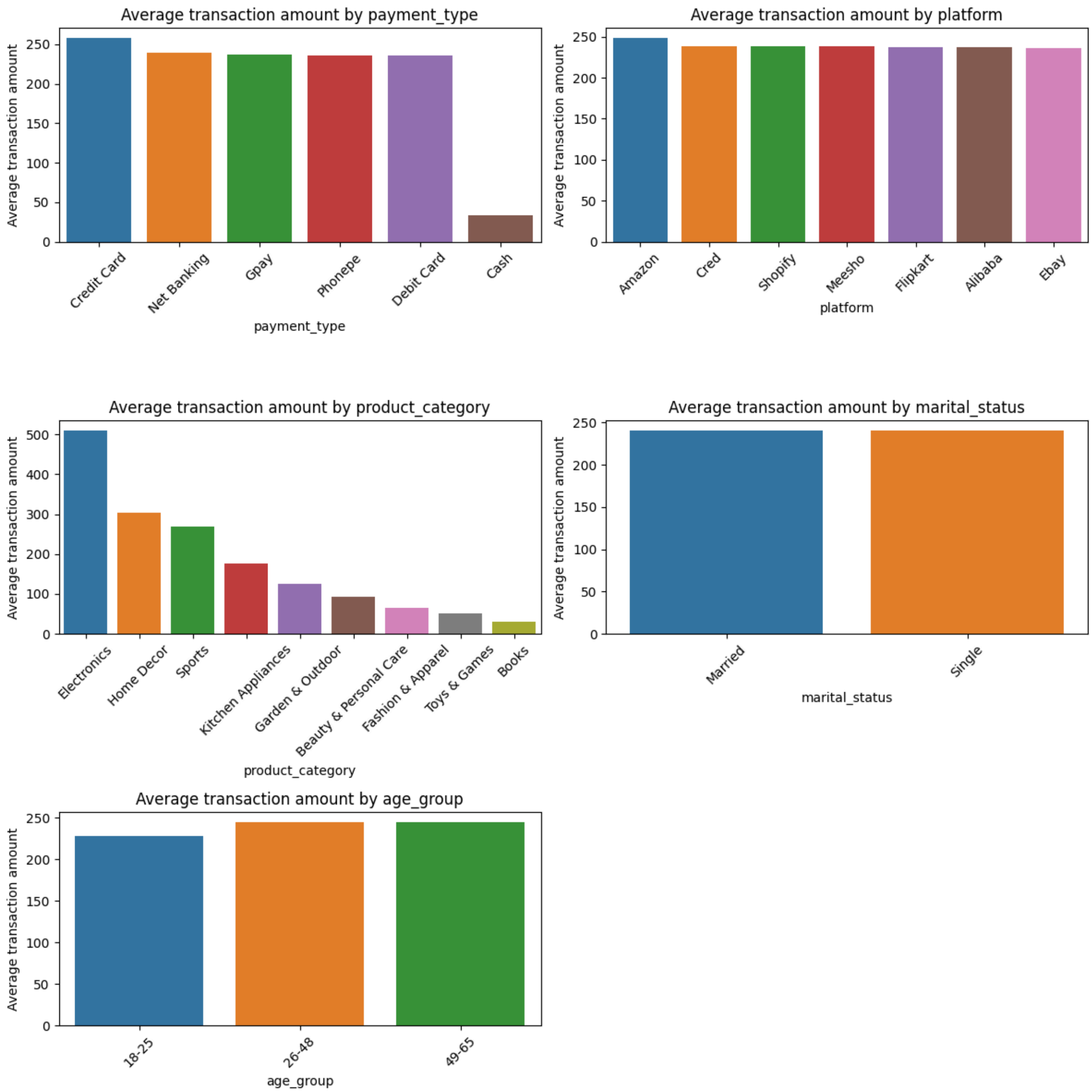
1. Top 3 purchasing categories of customers in age group (18 -25) : Electronics, Fashion & Apparel, Beauty & personal care
2. Top platforms : Amazon, Flipkart, Alibaba

Data Visualization: Average Transaction Amount

In [126…

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

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'

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

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



In [127…

```
df_trans.describe()
```

Out[127…

	tran_id	cust_id	tran_amount
count	500000.000000	500000.000000	500000.000000
mean	250000.500000	501.400428	240.672998
std	144337.711635	288.641924	241.696597
min	1.000000	1.000000	2.000000
25%	125000.750000	252.000000	64.553463
50%	250000.500000	502.000000	133.000000
75%	375000.250000	752.000000	349.000000
max	500000.000000	1000.000000	999.000000

Further Analysis On Age Group

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

In [128…

```
# Group the data by age group and calculate the average credit_limit and cre
age_group_metrics = df_merged.groupby('age_group')[['annual_income', 'credit
age_group_metrics
```

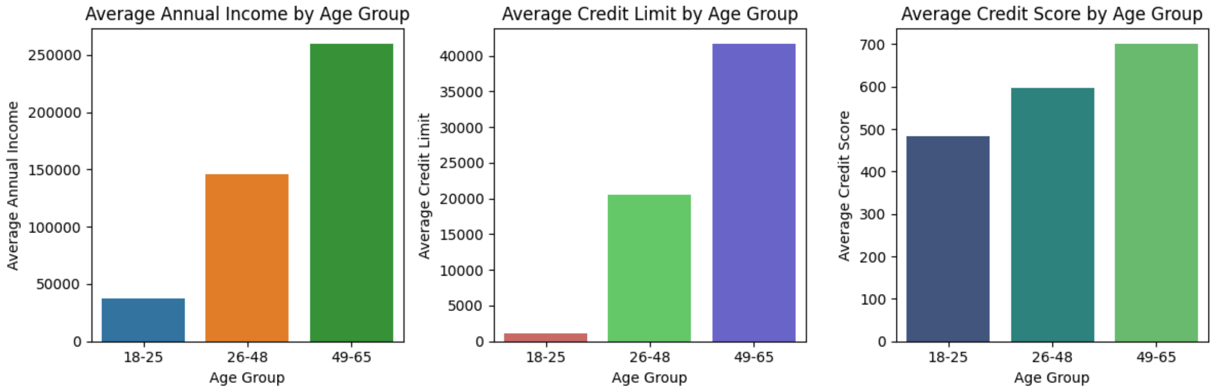
Out[128…

	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

In [129…

```
# 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, palett
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
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
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

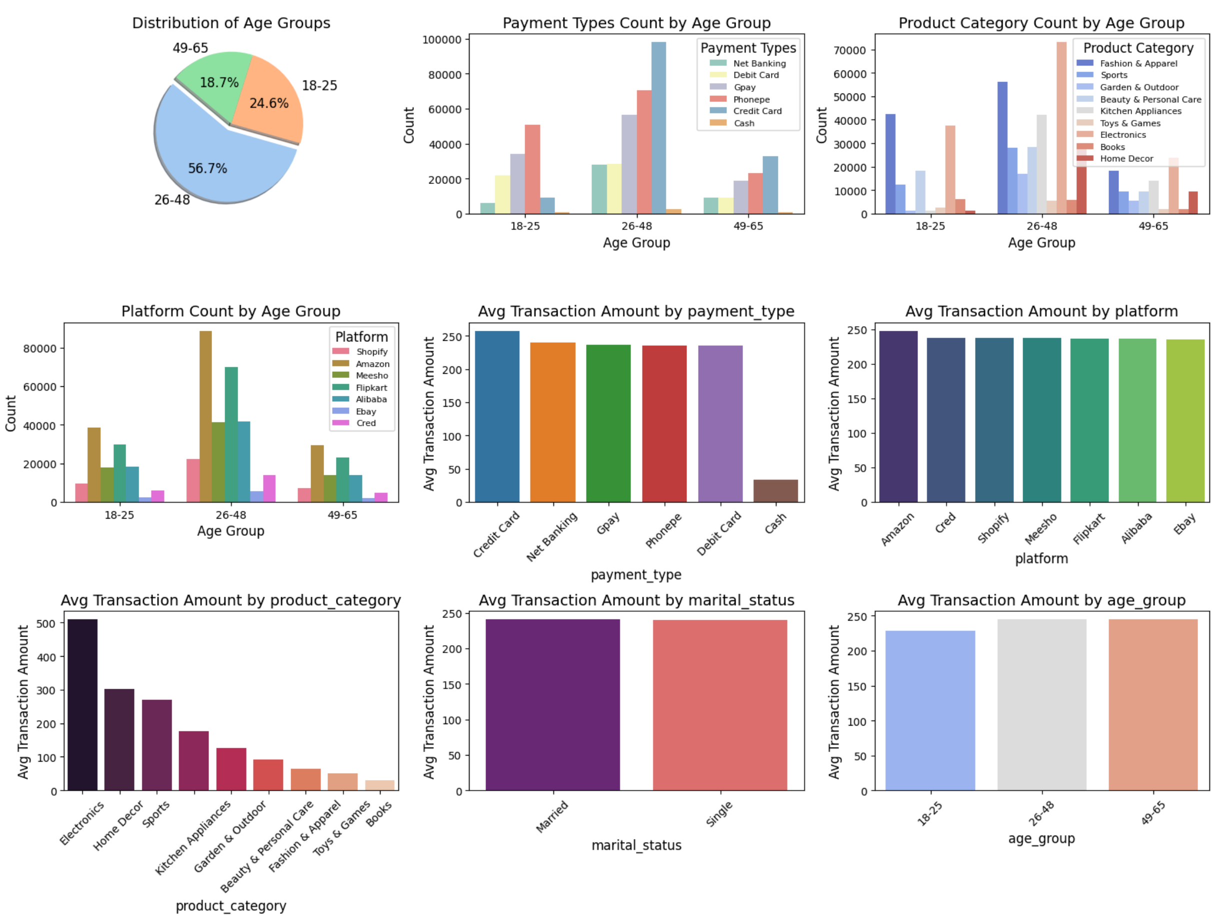
1. People with age group of 18 -25 accounts to ~26% of customer base in the data 2. Avg annual income of this group is less than 50k 3. They don't have much credit history which is getting reflected in their credit score and credit limit 4. Usage of credit cards as payment type is relatively low compared to other groups 5. Top 3 most shopping products categories : Electronics, Fashion & Apparel, Beauty & Personal care

Data Visualization

In [134…

```
# Create 3x3 grid layout
fig, axes = plt.subplots(3, 3, figsize=(16, 12))
axes = axes.flatten()

# === Pie Chart (First Plot) ===
age_group_counts = df_cust['age_group'].value_counts(normalize=True) * 100
colors = sns.color_palette("pastel")  # Soft color tones
axes[0].pie(age_group_counts, labels=age_group_counts.index, explode=(0.1, 0
            autopct='%1.1f%%', shadow=True, startangle=140, colors=colors, t
axes[0].set_title('Distribution of Age Groups', fontsize=14)

# === Categorical Count Plots ===
plots = [
    ('payment_type', 'Payment Types', 'Set3'),
    ('product_category', 'Product Category', 'coolwarm'),
    ('platform', 'Platform', 'husl')
]
#For loop  Function
for i, (col, title, palette) in enumerate(plots, start=1):
    sns.countplot(x='age_group', hue=col, data=df_merged_2, ax=axes[i], pale
    axes[i].set_title(f'{title} Count by Age Group', fontsize=14)
    axes[i].set_xlabel("Age Group", fontsize=12)
    axes[i].set_ylabel("Count", fontsize=12)
```

```
    axes[i].tick_params(axis='x', rotation=0)
    axes[i].legend(title=title, fontsize=8, title_fontsize=12, loc='upper ri

# === Avg Transaction Amount by Category ===
cat_cols = [
    ('payment_type', 'tab10'),
    ('platform', 'viridis'),
    ('product_category', 'rocket'),
    ('marital_status', 'magma'),
    ('age_group', 'coolwarm')
]

for i, (col, palette) in enumerate(cat_cols, start=4):
    avg_tran = df_merged_2.groupby(col)['tran_amount'].mean().reset_index().
    sns.barplot(x=col, y='tran_amount', data=avg_tran, ax=axes[i], palette=p
    axes[i].set_title(f'Avg Transaction Amount by {col}', fontsize=14)
    axes[i].set_xlabel(col, fontsize=12)
    axes[i].set_ylabel('Avg Transaction Amount', fontsize=12)
    axes[i].tick_params(axis='x', rotation=45)

# === Formatting Adjustments ===
plt.tight_layout()
plt.savefig("Analysis.png", dpi=400, bbox_inches="tight", pad_inches=0.1)
plt.show()
```



In [ ]:

AtliQ Bank Credit Card Project

(1) Pre-Campaign

We want to do a trial run for our new credit card. For this we need to figure out (1) How many customers do we need for our A/B testing. We will form a control and test group. For both of these groups we can figure out number of customers we need based on the statistical power and effect size that we agree upon after discussing with business. We will use

In [2]:
```
#import required libraries
import statsmodels.stats.api as sms
import statsmodels.api as sm
import pandas as pd
import numpy as np
from scipy import stats as st
from matplotlib import pyplot as plt
import seaborn as sns
```

In [3]:
```
alpha = 0.05
power = 0.8
effect_size=0.2

sms.tt_ind_solve_power(
    effect_size=0.2,
    alpha=alpha,
    power=power,
    ratio=1,
    alternative='two-sided'
)
```

Out[3]:
```
393.40569300025135
```

For effect size 2 we need 393 customers. We have to keep in mind budgeting restrictions while running this campaign hence let us run this for different effect sizes and discuss with business to find out which sample size would be optimal

In [4]:
```
# Calculate the required sample size for different effect sizes
effect_sizes = [0.1, 0.2, 0.3, 0.4, 0.5,1]   #  standard deviations greater t

for effect_size in effect_sizes:
    sample_size = sms.tt_ind_solve_power(effect_size=effect_size, alpha=alph
    print(f"Effect Size: {effect_size}, Required Sample Size: {int(sample_si
```

```
Effect Size: 0.1, Required Sample Size: 1570 customers
Effect Size: 0.2, Required Sample Size: 393 customers
Effect Size: 0.3, Required Sample Size: 175 customers
Effect Size: 0.4, Required Sample Size: 99 customers
Effect Size: 0.5, Required Sample Size: 63 customers
Effect Size: 1, Required Sample Size: 16 customers
```

Based on business requirements, the test should be capable of detecting a minimum 0.4 standard deviation difference between the control and test groups. For the effect size 0.4, we need 100 customers and when we discussed with business, 100 customers is ok in terms of their budgeting constraints for this trail run

Forming control and test groups

1.We have identified approximately 246 customers within the age group of 18 to 25. From this pool, we will select 100 customers for the initial campaign launch.

2.The campaign is launched for 100 customers, as determined by the effective size calculation and by considering budgeting costs, and will run campaign for a duration of 2 months

3.Got a conversion rate of ~40% (implies 40 out of 100 customers in test group started using credit card)

4.To maintain a similar sample size, a control group consisting of 40 customers will be created. Importantly, this control group will be completely exclusive of initial 100 customers used as test group.

5.So now we have 40 customers in each of control and test groups

At the end of the 2-month campaign period (from 09-10-23 to 11-10-23), we obtained daily data showing the average transaction amounts made by the entire group of 40 customers in both the control and test groups using existing and newly launched credit cards respectively

The key performance indicator (KPI) for this AB test aims to enhance average transaction amounts facilitated by the new card

(2) Post-Campaign

Two Sample Z Test for Our Hypothesis Testing

In [5]:
```
# Loading campaign results data
df = pd.read_csv('data/avg_transactions_after_campaign.csv')
df.head(4)
```

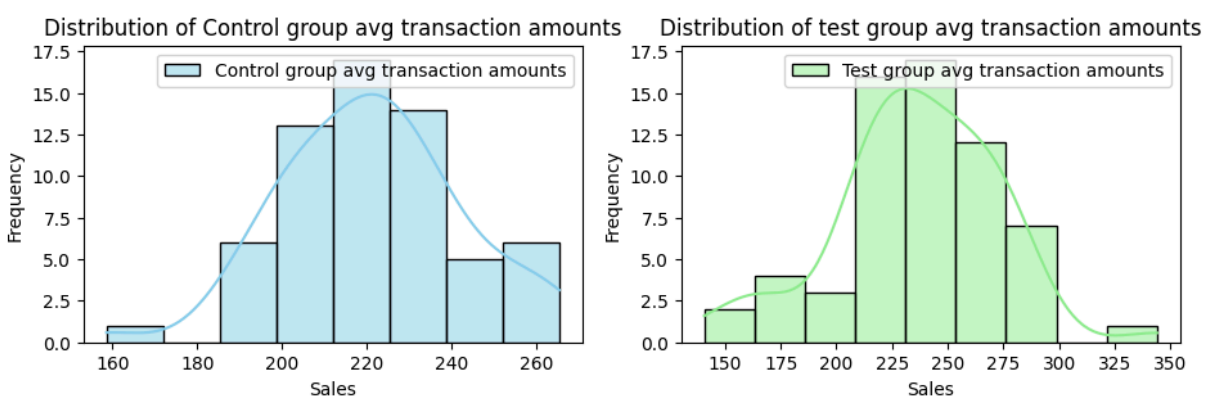
Out[5]:

	campaign_date	control_group_avg_tran	test_group_avg_tran
0	2023-09-10	259.83	277.32
1	2023-09-11	191.27	248.68
2	2023-09-12	212.41	286.61
3	2023-09-13	214.92	214.85

In [6]:
```
df.shape
```

Out[6]: (62, 3)

In [7]:
```
# Let's look at distributions of avg transactions amounts in both groups
# Create a 1x2 grid of subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(11, 3))

# Plot the distribution of Campaign A Sales
sns.histplot(df['control_group_avg_tran'], kde=True, color='skyblue', label=
ax1.set_xlabel('Sales')
ax1.set_ylabel('Frequency')
ax1.set_title('Distribution of Control group avg transaction amounts')
ax1.legend()

# Plot the distribution of Campaign B Sales
sns.histplot(df['test_group_avg_tran'], kde=True, color='lightgreen', label=
ax2.set_xlabel('Sales')
ax2.set_ylabel('Frequency')
ax2.set_title('Distribution of test group avg transaction amounts')
ax2.legend()

# Show the plots
plt.show()
```



Perform Hypothesis Testing Using Two Sample Z-test

In [8]:
```
control_mean = df["control_group_avg_tran"].mean().round(2)
control_std = df["control_group_avg_tran"].std().round(2)
control_mean, control_std
```

Out[8]: (np.float64(221.18), np.float64(21.36))

In [9]:
```
test_mean = df["test_group_avg_tran"].mean().round(2)
test_std = df["test_group_avg_tran"].std().round(2)
test_mean, test_std
```

Out[9]:
```
(np.float64(235.98), np.float64(36.66))
```

In [10]:
```
sample_size = df.shape[0]
sample_size
```

Out[10]:
```
62
```

Test Using Rejection Region (i.e. Critical Z Value)

In [11]:
```
a = (control_std**2/sample_size)
b = (test_std**2/sample_size)

Z_score = (test_mean-control_mean)/np.sqrt(a+b)
Z_score
```

Out[11]:
```
np.float64(2.7466072001806734)
```

In [12]:
```
# For a significance level of 5% (0.05) in a right-tailed test, the critical
critical_z_value = st.norm.ppf(1 - alpha)  # Right-tailed test at 5% signifi
critical_z_value
```

Out[12]:
```
np.float64(1.6448536269514722)
```

In [13]:
```
Z_score > critical_z_value
```

Out[13]:
```
np.True_
```

Since Z score is higher than critical Z value, we can reject the null hypothesis.

Test Using p-Value

In [14]:
```
# Calculate the p-value corresponding to z score for a right-tailed test
p_value = 1 - st.norm.cdf(Z_score)
p_value
```

Out[14]:
```
np.float64(0.0030107601919702187)
```

In [15]:
```
p_value < alpha # p value is less than significance level of 5% (or 0.05 for
```

Out[15]:
```
np.True_
```

Since p value is less than significance level (i.e. alpha), we can reject the null hypothesis.

Using Ready Made API call

1.We will now use stats module from statmodels for doing Z-test

2.The order of passing control and test group data to sm.stats.ztest(test_data, control_data) defines the direction of the test and influences the test results.

3.When you pass test group data first, z-test module assumes that alternative hypothesis as mean of the test group is greater than the mean of the control group and conversely if you switch the order z-test module assumes alternative hypothesis as control group average is more than test group

4.In here we will be using order as sm.stats.ztest(test_group_data, control_group_data) based on our alternative hypothesis considered above.

5.By default z-test module in statmodels performs two tailed test. As we are doing one-tailed test in our case based on the direction and alternate hypothesis we have to set "alternative" parameter.

6.In out case based on test direction we will set "alternative" parameter to "larger"

How to choose right Alternative parameter

a.Two-tailed, meaning you are interested in identifying deviations across control and test groups in either direction

b.larger, This is a one-tailed test, specifically looking for whether the first group is significantly larger than the second

c.smaller, This is another one-tailed test, specifically looking for whether the first group is significantly smaller than the second

image.png

You can check more details about this z-test module and paramteres in here https://statsmodels.org/devel/generated/statsmodels.stats.weightstats.ztest.html

In [16]:
```
# Performing Z-test with above considerations
z_statistic, p_value = sm.stats.ztest( df['test_group_avg_tran'],df['control
z_statistic, p_value
```

Out[16]:
```
(np.float64(2.7482973745691135), np.float64(0.002995282462202502))
```