Bank Credit Card Launch

Problem Statement

A new banking company wants to launch a credit card in the highly competitive Indian market.The company needed to identify the most promising target market segment and tailor its credit card offering to meet the specific needs and preferences of that segment.The objective is to make data-driven decisions regarding the target market segment and ensure the successful launch of the new credit card within that segment, enabling the banking company to gain a competitive edge in the Indian market.

Phase 1: Bank Credit Card Project

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

Data Import

In [1]:

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

In [3]:

```
df_cust = pd.read_csv('Datasets/customers.csv')
df_cs = pd.read_csv('Datasets/credit_profiles.csv')
df_trans = pd.read_csv('Datasets/transactions.csv')
```

Exploring Customers Table

In [6]:

```
df_cust.head()
```

Out[6]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status
0	1	Manya Acharya	Female	2	City	Business Owner	358211.0	Married
1	2	Anjali Pandey	Female	47	City	Consultant	65172.0	Single
2	3	Aaryan Chauhan	Male	21	City	Freelancer	22378.0	Married
3	4	Rudra Bali	Male	24	Rural	Freelancer	33563.0	Married
4	5	Advait Malik	Male	48	City	Consultant	39406.0	Married

In [8]:

```
df_cust.describe()
```

Out[8]:

	cust_id	age	annual_income

	cust_id	age	annual_income
count	1000.000000	1000.000000	950.000000
mean	500.500000	36.405000	139410.314737
std	288.819436	15.666155	112416.802007
min	1.000000	1.000000	2.000000
25%	250.750000	26.000000	47627.500000
50%	500.500000	32.000000	112218.500000
75%	750.250000	46.000000	193137.500000
max	1000.000000	135.000000	449346.000000

Handle Null Values

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

In [11]:

```
df_cust.isnull().sum() #50 null values in annual_income.
```

Out[11]:

```
cust_id            0
name               0
gender             0
age                0
location           0
occupation         0
annual_income     50
marital_status     0
dtype: int64
```

1. Analyze Income Column

Handle Null Values: Annual income

In [15]:

```
df_cust[df_cust.annual_income.isnull()].head()
```

Out[15]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status
14	15	Sanjana Malik	Female	25	Rural	Artist	NaN	Married
82	83	Reyansh Mukherjee	Male	27	City	Freelancer	NaN	Single
97	98	Virat Puri	Male	47	Suburb	Business Owner	NaN	Married
102	103	Aarav Shah	Male	32	City	Data Scientist	NaN	Married
155	156	Kiaan Saxena	Male	24	City	Fullstack Developer	NaN	Married

In [17]:

```
occupation_wise_sales_median = df_cust.groupby('occupation').annual_income.median()
occupation_wise_sales_median
```

Out[17]:

```
occupation
Accountant              65265.0
Artist                  45794.0
Business Owner         261191.5
Consultant              58017.0
Data Scientist         135759.0
Freelancer              46759.0
```

```
Fullstack Developer      76774.0
Name: annual_income, dtype: float64
```

In [19]:

```
df_cust ['annual_income'] = df_cust.apply(
    lambda row : occupation_wise_sales_median[row['occupation']] if pd.isnull(row['annua
l_income']) else row["annual_income"],
    axis =1)
```

In [21]:

```
df_cust.isnull().sum()  # No null values
```

Out[21]:

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

In [23]:

```
df_cust.iloc[[14,82]]
```

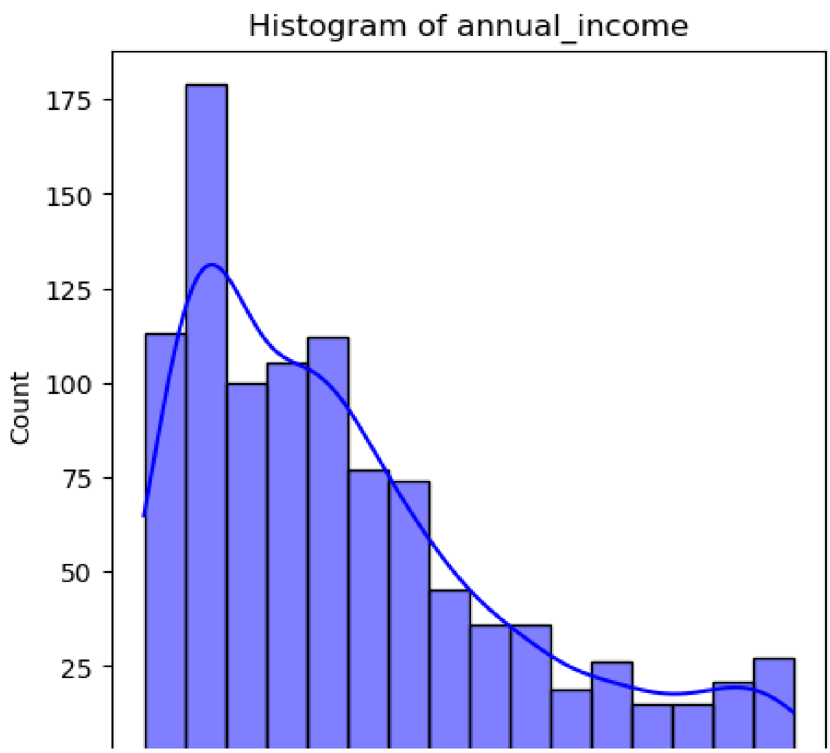
Out[23]:

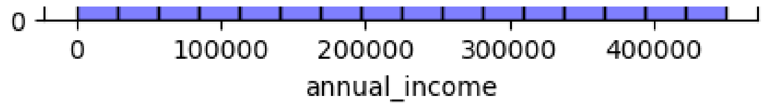
	cust_id	name	gender	age	location	occupation	annual_income	marital_status
14	15	Sanjana Malik	Female	25	Rural	Artist	45794.0	Married
82	83	Reyansh Mukherjee	Male	27	City	Freelancer	46759.0	Single

Previously records at location 14 and 82 had null annual income. Now you have a median value per occupation.

In [26]:

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





In [28]:

```
df_cust.describe()
```

Out[28]:

	cust_id	age	annual_income
count	1000.000000	1000.000000	1000.000000
mean	500.500000	36.405000	138916.765500
std	288.819436	15.666155	110969.408643
min	1.000000	1.000000	2.000000
25%	250.750000	26.000000	48229.500000
50%	500.500000	32.000000	113416.000000
75%	750.250000	46.000000	192614.000000
max	1000.000000	135.000000	449346.000000

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 [31]:

```
df_cust[df_cust.annual_income<100]
```

Out[31]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status
31	32	Veer Mistry	Male	50	City	Business Owner	50.0	Married
262	263	Vivaan Tandon	Male	53	Suburb	Business Owner	50.0	Married
316	317	Yuvraj Saxena	Male	47	City	Consultant	50.0	Married
333	334	Avani Khanna	Female	29	City	Data Scientist	50.0	Married
340	341	Priya Sinha	Female	33	Rural	Fullstack Developer	50.0	Married
543	544	Advait Batra	Male	54	City	Consultant	2.0	Married
592	593	Priya Gandhi	Female	32	City	Business Owner	50.0	Married
633	634	Rudra Mehtani	Male	26	City	Data Scientist	2.0	Married
686	687	Vihaan Jaiswal	Male	40	City	Business Owner	2.0	Married
696	697	Ishan Negi	Male	47	City	Consultant	20.0	Married

In [33]:

```
df_cust[df_cust.annual_income<100].shape
```

Out[33]:

```
(10, 8)
```

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

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 [36]:

```
for index, row in df_cust.iterrows():
    if row["annual_income"] < 100:
        df_cust.at[index , 'annual_income'] = occupation_wise_sales_median[row['occupati
on']]
```

In [38]:

```
df_cust[df_cust.annual_income<100] # No outliers
```

Out[38]:

cust_id	name	gender	age	location	occupation	annual_income	marital_status

In [40]:

```
df_cust.loc[[31,262]]
```

Out[40]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status
31	32	Veer Mistry	Male	50	City	Business Owner	261191.5	Married
262	263	Vivaan Tandon	Male	53	Suburb	Business Owner	261191.5	Married

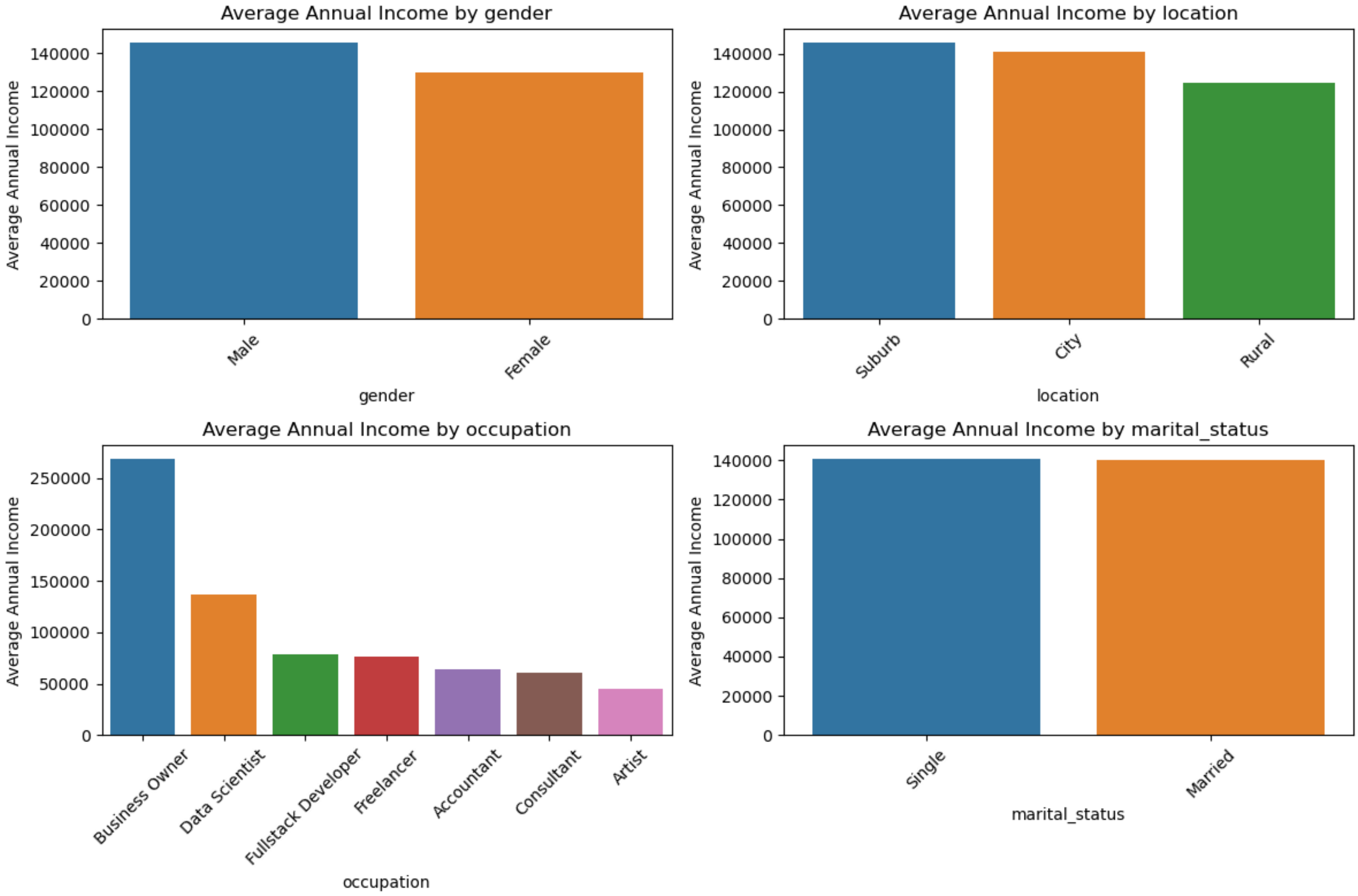
Record at 31,262 location had annual income of < 100$. Now you can see it is replaced by a median income per occupation.

Data Visualization: Annual Income

In [44]:

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

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

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().reset_inde
x()

    # Sort the data in descending order by 'annual_income' before plotting
    sorted_data = avg_income_by_category.sort_values(by='annual_income', ascending=False
)

    sns.barplot(x=cat_col, y='annual_income', data=sorted_data, ci=None, ax=axes[i], pal
ette='tab10')
    axes[i].set_title(f'Average Annual Income by {cat_col}')
    axes[i].set_xlabel(cat_col)
    axes[i].set_ylabel('Average Annual Income')

    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=45)

for i in range(len(cat_cols), len(axes)):
```

```
    fig.delaxes(axes[i])
plt.tight_layout()
plt.show()
```



2. Analyze Age Column

In [47]:

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

Out[47]:

0

No null values

In [50]:

```
df_cust.describe()
```

Out[50]:

	cust_id	age	annual_income
count	1000.000000	1000.000000	1000.000000
mean	500.500000	36.405000	140483.548500
std	288.819436	15.666155	110463.002934
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.

In [53]:

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

Out[53]:

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

In [55]:

```
outliers.shape
```

Out[55]:

```
(20, 8)
```

We cannot remove 20 rows as they are important. So to treat these outliers we will use median age for each of the occupation

In [58]:

```
median_age_per_occupation = df_cust.groupby('occupation').age.median()
median_age_per_occupation
```

Out[58]:

```
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 [60]:

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

In [62]:

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

Out[62]:

cust_id	name	gender	age	location	occupation	annual_income	marital_status

In [64]:

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

Out[64]:

```
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

Data Visualization: Age Column

In [68]:

```
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.head()
```

Out[68]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status	age_group
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	Married	49-65
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	Single	26-48
2	3	Aaryan Chauhan	Male	21.0	City	Freelancer	22378.0	Married	18-25
3	4	Rudra Bali	Male	24.0	Rural	Freelancer	33563.0	Married	18-25
4	5	Advait Malik	Male	48.0	City	Consultant	39406.0	Married	26-48

In [70]:

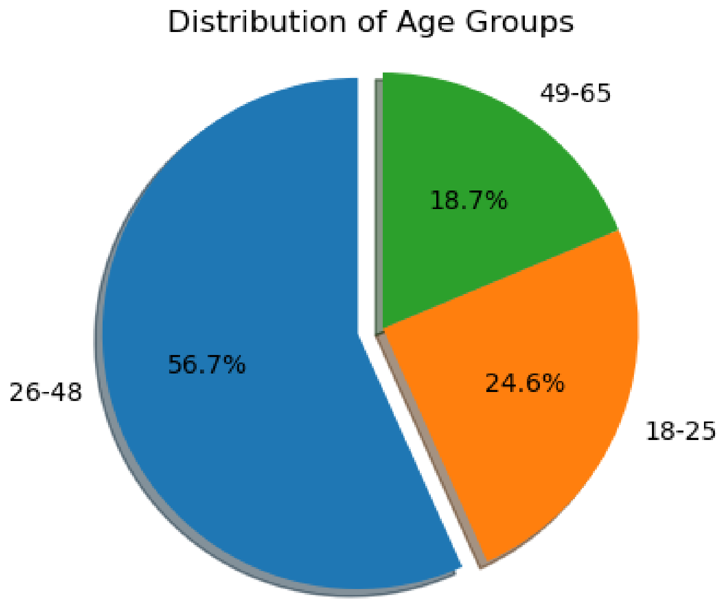
```
age_group_counts = df_cust.age_group.value_counts(normalize = True)*100 # normalize for
converting in into %
age_group_counts
```

Out[70]:

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

In [72]:

```
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=90
)
plt.axis('equal')
plt.title('Distribution of Age Groups')
plt.show()
```



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

3. Analyze Gender and Location Distribution

In [76]:

```
df_cust.location.value_counts()
```

Out[76]:

```
location
City      683
Suburb    232
Rural      85
Name: count, dtype: int64
```

In [78]:

```
df_cust.gender.value_counts()
```

Out[78]:

```
gender
Male      674
Female    326
Name: count, dtype: int64
```

In [80]:

```
customer_location_gender = df_cust.groupby('location').gender.value_counts().unstack()
customer_location_gender
```

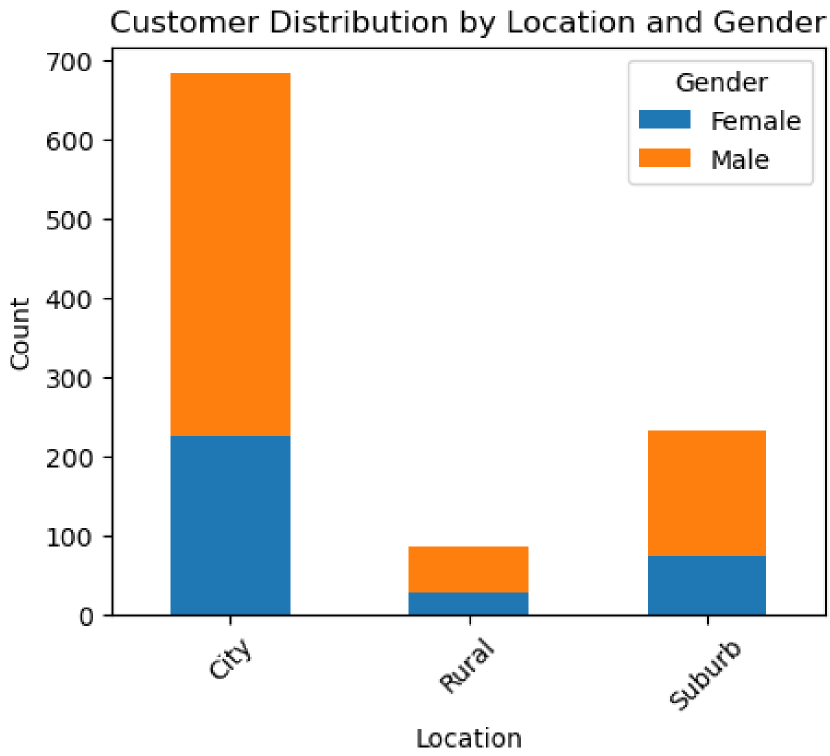
Out[80]:

gender	Female	Male
location		

gender	Female	Male
location		
City	226	457
Rural	26	59
Suburb	74	158

In [82]:

```
customer_location_gender.plot(kind='bar', stacked=True, figsize=(5, 4))

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

plt.legend(title='Gender', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45)

plt.show()
```



Explore Credit Score Table

In [85]:

```
df_cs.head()
```

Out[85]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit
0	1	749	0.585171	19571.0	0.0	40000.0
1	2	587	0.107928	161644.0	2.0	1250.0
2	3	544	0.854807	513.0	4.0	1000.0
3	4	504	0.336938	224.0	2.0	1000.0
4	5	708	0.586151	18090.0	2.0	40000.0

In [87]:

```
df_cs.shape
```

Out[87]:

```
(1004, 6)
```

Credit score table should have same records as customers table. There might be invalid or duplicate data.

In [90]:

```
df_cs.cust_id.nunique() # some records have duplicates
```

Out[90]:

```
1000
```

In [92]:

```
df_cs[df_cs.cust_id.duplicated(keep = False)]
```

Out[92]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit
516	517	308	NaN	NaN	NaN	NaN
517	517	308	0.113860	33.0	3.0	500.0
569	569	344	NaN	NaN	NaN	NaN
570	569	344	0.112599	37.0	0.0	500.0
607	606	734	NaN	NaN	NaN	NaN
608	606	734	0.193418	4392.0	1.0	40000.0
664	662	442	NaN	NaN	NaN	NaN
665	662	442	0.856039	266.0	2.0	500.0

In [94]:

```
df_cs_clean_1 = df_cs.drop_duplicates(subset='cust_id' , keep='last')
df_cs_clean_1.shape
```

Out[94]:

```
(1000, 6)
```

In [96]:

```
df_cs_clean_1[df_cs_clean_1.cust_id.duplicated(keep = False)]
```

Out[96]:

cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit

df_cs_clean_1 looks clean now after cleaning duplicates.

Data Cleaning Step 2: Handle Null Values

In [100]:

```
df_cs_clean_1.isnull().sum() # 65 null values in credit_limit.
```

Out[100]:

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

```
dtype: int64
```

In [102]:

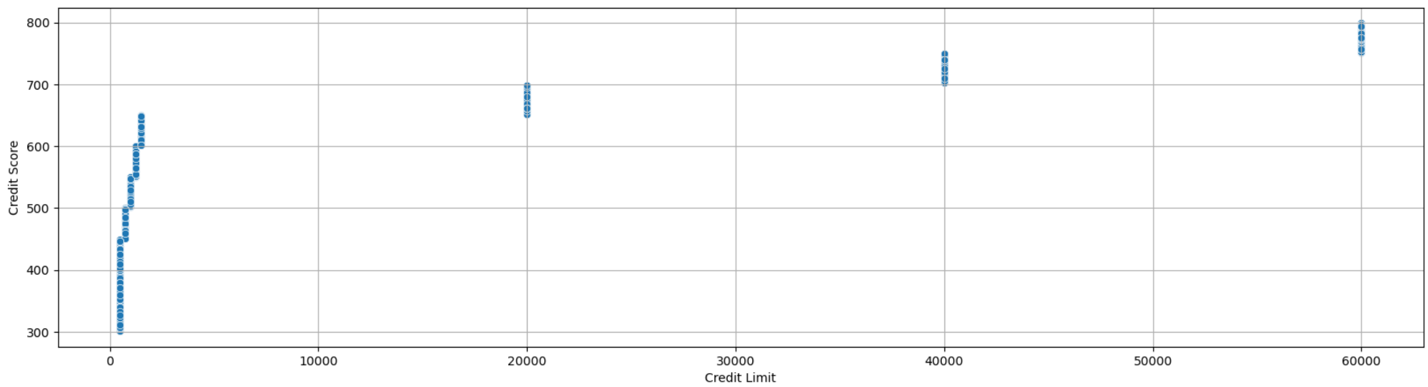
```
df_cs_clean_1[df_cs_clean_1.credit_limit.isnull()].head()
```

Out[102]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit
10	11	679	0.557450	9187.0	2.0	NaN
35	36	790	0.112535	4261.0	1.0	NaN
37	38	514	0.296971	238.0	2.0	NaN
45	46	761	0.596041	24234.0	2.0	NaN
64	65	734	0.473715	13631.0	0.0	NaN

In [104]:

```
plt.figure(figsize=(20, 5))
sns.scatterplot(x=df_cs_clean_1.credit_limit, y=df_cs.credit_score )
plt.xlabel('Credit Limit')
plt.ylabel('Credit Score')
plt.grid(True)
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 [107]:

```
bin_ranges = [300, 450, 500, 550, 600, 650, 700, 750, 800]

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

df_cs_clean_1['credit_score_range'] = pd.cut(df_cs_clean_1['credit_score'], bins=bin_rang
es, labels=bin_labels, include_lowest=True, right=False)
```

In [109]:

```
df_cs_clean_1.head()
```

Out[109]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_score_range
0	1	749	0.585171	19571.0	0.0	40000.0	700-749
1	2	587	0.107928	161644.0	2.0	1250.0	550-599
2	3	544	0.854807	513.0	4.0	1000.0	500-549
3	4	504	0.336938	224.0	2.0	1000.0	500-549
4	5	708	0.586151	18090.0	2.0	40000.0	700-749

In [111]:

```
mode_df = df_cs_clean_1.groupby('credit_score_range')["credit_limit"].agg(lambda x : x.m
ode().iloc[0]).reset_index()
mode_df
```

Out[111]:

	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 [113]:

```
df_cs_clean_2 =pd.merge(df_cs_clean_1 , mode_df , on = 'credit_score_range' , suffixes=
("","_mode"))
df_cs_clean_2.sample(3)
```

Out[113]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_score_range
985	986	425	0.178470	56.0	4.0	500.0	300-449
646	647	498	0.658087	128818.0	3.0	750.0	450-499
735	736	483	0.693349	385.0	0.0	750.0	450-499

In [115]:

```
df_cs_clean_2[df_cs_clean_2.credit_limit.isnull()].sample(3)
```

Out[115]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_score_range
662	663	478	0.154754	84.0	0.0	NaN	450-499
805	806	617	0.421308	416.0	1.0	NaN	600-649
167	168	737	0.489797	12421.0	2.0	NaN	700-749

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 [118]:

```
df_cs_clean_3 = df_cs_clean_2.copy()
df_cs_clean_3['credit_limit'].fillna(df_cs_clean_3['credit_limit_mode'], inplace=True)
df_cs_clean_3.shape
```

Out[118]:

```
(1000, 8)
```

In [120]:

```
df_cs_clean_3.isnull().sum()
```

Out[120]:

```
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
```

In [122]:

```
df_cs_clean_3[df_cs_clean_3.cust_id==211]
```

Out[122]:

	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_score_range
210	211	405	0.633233	160.0	2.0	500.0	300-449

Previously customer id 211 had null value in credit_limit. Now it has a valid value.

Data Cleaning Step 3: Handle Outliers

In [126]:

```
df_cs_clean_3.describe()
```

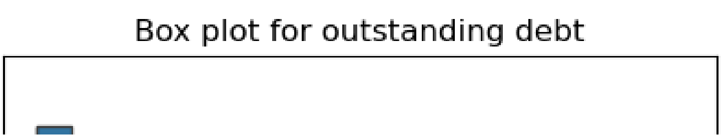
Out[126]:

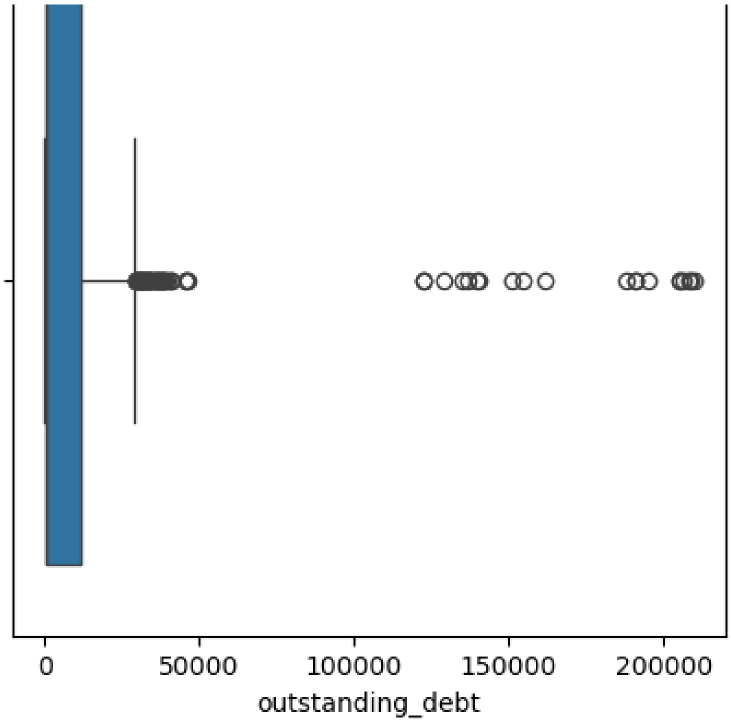
	cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_limit_
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.0
mean	500.500000	589.182000	0.498950	9683.597000	1.955000	19733.75000	19912.5
std	288.819436	152.284929	0.233139	25255.893671	1.414559	24717.43818	24840.9
min	1.000000	300.000000	0.103761	33.000000	0.000000	500.00000	500.0
25%	250.750000	460.000000	0.293917	221.000000	1.000000	750.00000	750.0
50%	500.500000	601.500000	0.487422	550.000000	2.000000	1500.00000	1500.0
75%	750.250000	738.000000	0.697829	11819.500000	3.000000	40000.00000	40000.0
max	1000.000000	799.000000	0.899648	209901.000000	4.000000	60000.00000	60000.0

Outliers are there in outstanding_debt as max outstanding debt is more than credit limit which is not possible

In [129]:

```
# Through box plot we can see the circles which are outliers as they are not in the range
plt.figure(figsize=(5, 5))
sns.boxplot(x=df_cs_clean_3['outstanding_debt'])
plt.title('Box plot for outstanding debt')
plt.show()
```





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

df_cs_clean_3[df_cs_clean_3.outstanding_debt>df_cs_clean_3.credit_limit]

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 [134]:

```
df_cs_clean_3.loc[df_cs_clean_3.outstanding_debt>df_cs_clean_3.credit_limit , 'outstandin
g_debt'] = df_cs_clean_3.credit_limit
```

In [136]:

```
df_cs_clean_3[df_cs_clean_3.outstanding_debt>df_cs_clean_3.credit_limit]
```

Out[136]:

cust_id	credit_score	credit_utilisation	outstanding_debt	credit_inquiries_last_6_months	credit_limit	credit_score_range	cre

All outliers in column outstanding_debt are now GONE.

Data Exploration: Visualizing Correlation in Credit Score Table

In [140]:

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

Out[140]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status	age_group	credit_score	credit_utilisa
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	Married	49-65	749	0.585
1	2	Anjali Pandey	Female	47.0	City	Consultant	65172.0	Single	26-48	587	0.107

In [142]:

```
df_merged[['credit_limit' , 'credit_score']].corr() # Strong correlation
```

Out[142]:

	credit_limit	credit_score
credit_limit	1.000000	0.847952
credit_score	0.847952	1.000000

In [144]:

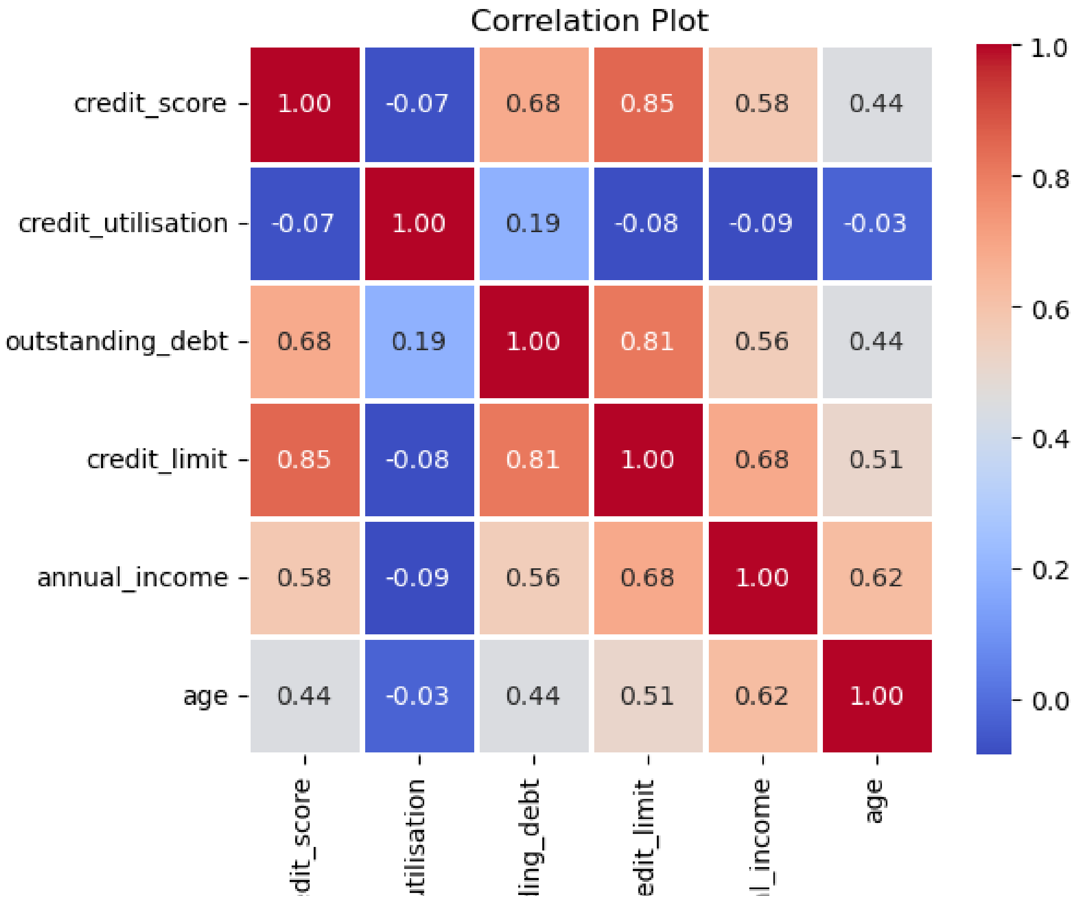
```
numerical_cols = ['credit_score', 'credit_utilisation', 'outstanding_debt', 'credit_limit
', 'annual_income','age']

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

Out[144]:

	credit_score	credit_utilisation	outstanding_debt	credit_limit	annual_income	age
credit_score	1.000000	-0.070445	0.680654	0.847952	0.575751	0.444917
credit_utilisation	-0.070445	1.000000	0.192838	-0.080493	-0.086368	-0.027713
outstanding_debt	0.680654	0.192838	1.000000	0.810581	0.555661	0.444301
credit_limit	0.847952	-0.080493	0.810581	1.000000	0.684775	0.510993
annual_income	0.575751	-0.086368	0.555661	0.684775	1.000000	0.619037
age	0.444917	-0.027713	0.444301	0.510993	0.619037	1.000000

In [146]:

```
# Create a heatmap of the correlation matrix
plt.figure(figsize=(6, 5))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.8)
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

Transactions Table

In [150]:

```
df_trans.head()
```

Out[150]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
0	1	705	2023-01-01	63	Flipkart	Electronics	Phonepe
1	2	385	2023-01-01	99	Alibaba	Fashion & Apparel	Credit Card
2	3	924	2023-01-01	471	Shopify	Sports	Phonepe
3	4	797	2023-01-01	33	Shopify	Fashion & Apparel	Gpay
4	5	482	2023-01-01	68	Amazon	Fashion & Apparel	Net Banking

Data Cleaning Step 1: Handle NULL Values

In [153]:

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

Out[153]:

```
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.

In [156]:

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

Out[156]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
355	356	58	2023-01-01	237	NaN	Electronics	Net Banking
418	419	383	2023-01-01	338	NaN	Electronics	Credit Card
607	608	421	2023-01-01	700	NaN	Electronics	Phonepe
844	845	945	2023-01-01	493	NaN	Sports	Credit Card
912	913	384	2023-01-01	85	NaN	Fashion & Apparel	Phonepe

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
...	...	...	...	...	...	...	...
499579	499580	924	2023-09-05	31	NaN	Fashion & Apparel	Gpay
499646	499647	944	2023-09-05	58445	NaN	Fashion & Apparel	Phonepe
499725	499726	620	2023-09-05	15	NaN	Sports	Net Banking
499833	499834	616	2023-09-05	97	NaN	Fashion & Apparel	Credit Card
499997	499998	57	2023-09-05	224	NaN	Garden & Outdoor	Phonepe

4941 rows × 7 columns

In [158]:

```
df_trans.platform.value_counts()
```

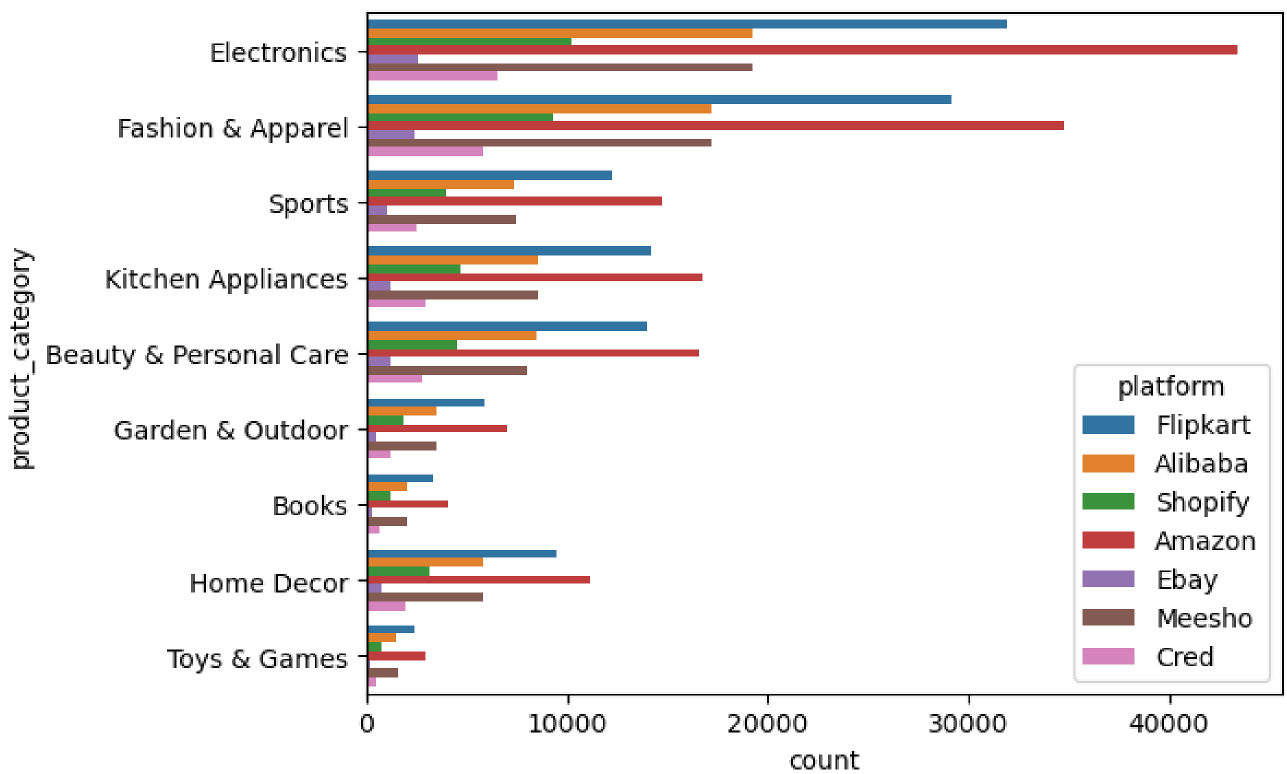
Out[158]:

```
platform
Amazon      151443
Flipkart    122660
Alibaba      73584
Meesho       73271
Shopify      39416
Cred         24741
Ebay          9944
Name: count, dtype: int64
```

As you can see Amazon is the platform that users use the most but we will see for each category with the help of countplot

In [161]:

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



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 [163]:

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

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

In [164]:

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

Out[164]:

tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type

Data Cleaning Step 2: Treat Outliers

In [169]:

```
df_trans.describe()
```

Out[169]:

	tran_id	cust_id	tran_amount
count	500000.000000	500000.000000	500000.00000
mean	250000.500000	501.400428	3225.20733
std	144337.711634	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 in tran_amount column

In [172]:

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

Out[172]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
120	121	440	2023-01-01	0	Amazon	Electronics	Credit Card
141	142	839	2023-01-01	0	Amazon	Electronics	Credit Card
517	518	147	2023-01-01	0	Amazon	Electronics	Credit Card
533	534	891	2023-01-01	0	Amazon	Electronics	Credit Card
586	587	108	2023-01-01	0	Amazon	Electronics	Credit Card

In [174]:

```
df_trans_zero.shape
```

Out[174]:

```
(4734, 7)
```

In [176]:

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

Out[176]:

```
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 [179]:

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

Out[179]:

(15637, 7)

In [181]:

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

Out[181]:

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

10903 rows × 7 columns

In [183]:

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

Out[183]:

554.0

In [185]:

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

In [187]:

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

Out[187]:

tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type

No zero values are left in tran_amount column

In [190]:

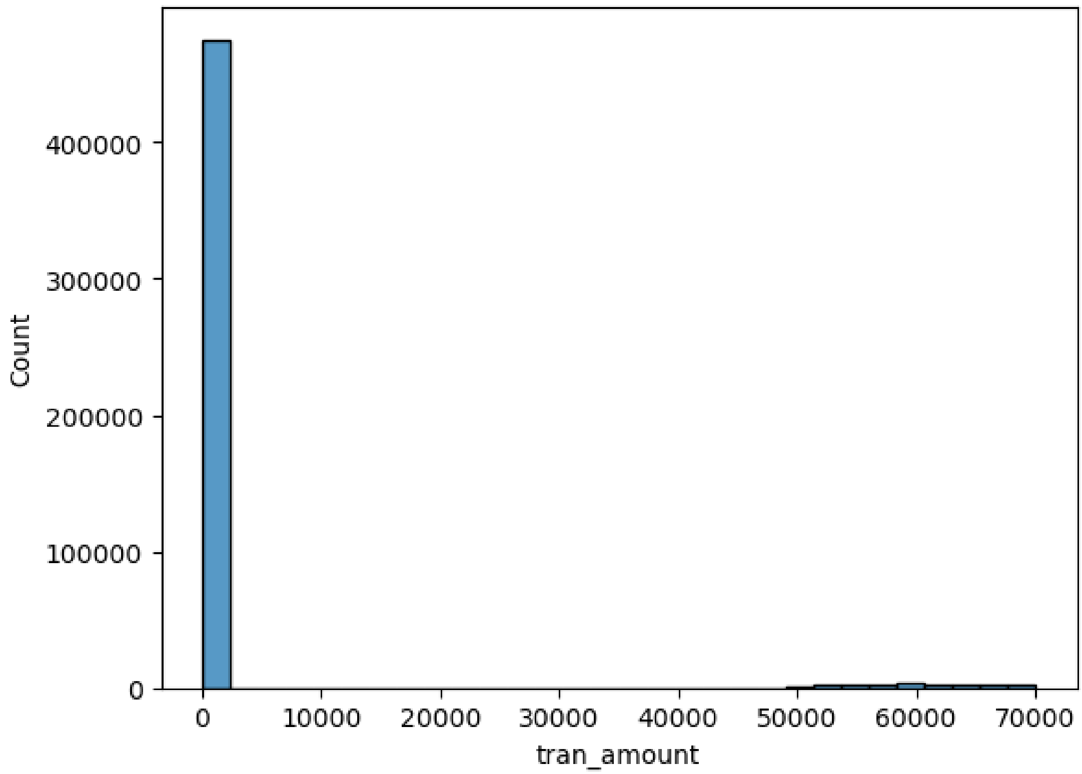
```
df_trans.tran_amount.describe()
```

Out[190]:

```
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 [192]:

```
sns.histplot(df_trans.tran_amount,bins =30) #rightly skewed
plt.show()
```



In [193]:

```
Q1, Q3 = df_trans['tran_amount'].quantile([0.25, 0.75])
IQR = Q3 - Q1
lower = Q1 - 2 * IQR  # Instead of 1.5 for IQR  we used 2 to be more flexible
upper = Q3 + 2 * IQR

lower, upper
```

Out[193]:

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

In [196]:

```
#25000 outliers so we have to handle the outliers
df_trans_outliers = df_trans[df_trans.tran_amount>=upper]
df_trans_outliers
```

Out[196]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
26	27	380	2023-01-01	61963	Shopify	Beauty & Personal Care	Credit Card
49	50	287	2023-01-01	57869	Amazon	Toys & Games	Gpay
94	95	770	2023-01-01	52881	Ebay	Kitchen Appliances	Credit Card

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
104	105	549	2023-01-01	58574	Flipkart	Fashion & Apparel	Gpay
113	114	790	2023-01-01	51669	Shopify	Kitchen Appliances	Credit Card
...	...	...	...	...	...	...	...
499742	499743	868	2023-09-05	55131	Meesho	Fashion & Apparel	Gpay
499888	499889	614	2023-09-05	59679	Meesho	Fashion & Apparel	Net Banking
499900	499901	811	2023-09-05	60184	Flipkart	Sports	Debit Card
499966	499967	662	2023-09-05	54678	Meesho	Sports	Gpay
499996	499997	569	2023-09-05	53022	Meesho	Fashion & Apparel	Net Banking

25000 rows × 7 columns

In [198]:

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

Out[198]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
0	1	705	2023-01-01	63	Flipkart	Electronics	Phonepe
1	2	385	2023-01-01	99	Alibaba	Fashion & Apparel	Credit Card
2	3	924	2023-01-01	471	Shopify	Sports	Phonepe
3	4	797	2023-01-01	33	Shopify	Fashion & Apparel	Gpay
4	5	482	2023-01-01	68	Amazon	Fashion & Apparel	Net Banking
...	...	...	...	...	...	...	...
499994	499995	679	2023-09-05	59	Ebay	Beauty & Personal Care	Gpay
499995	499996	791	2023-09-05	43	Amazon	Books	Phonepe
499997	499998	57	2023-09-05	224	Amazon	Garden & Outdoor	Phonepe
499998	499999	629	2023-09-05	538	Flipkart	Home Decor	Gpay
499999	500000	392	2023-09-05	346	Amazon	Kitchen Appliances	Net Banking

475000 rows × 7 columns

In [200]:

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

Out[200]:

```
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 [202]:

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

In [204]:

```
df_trans.loc[[26 , 49]]
```

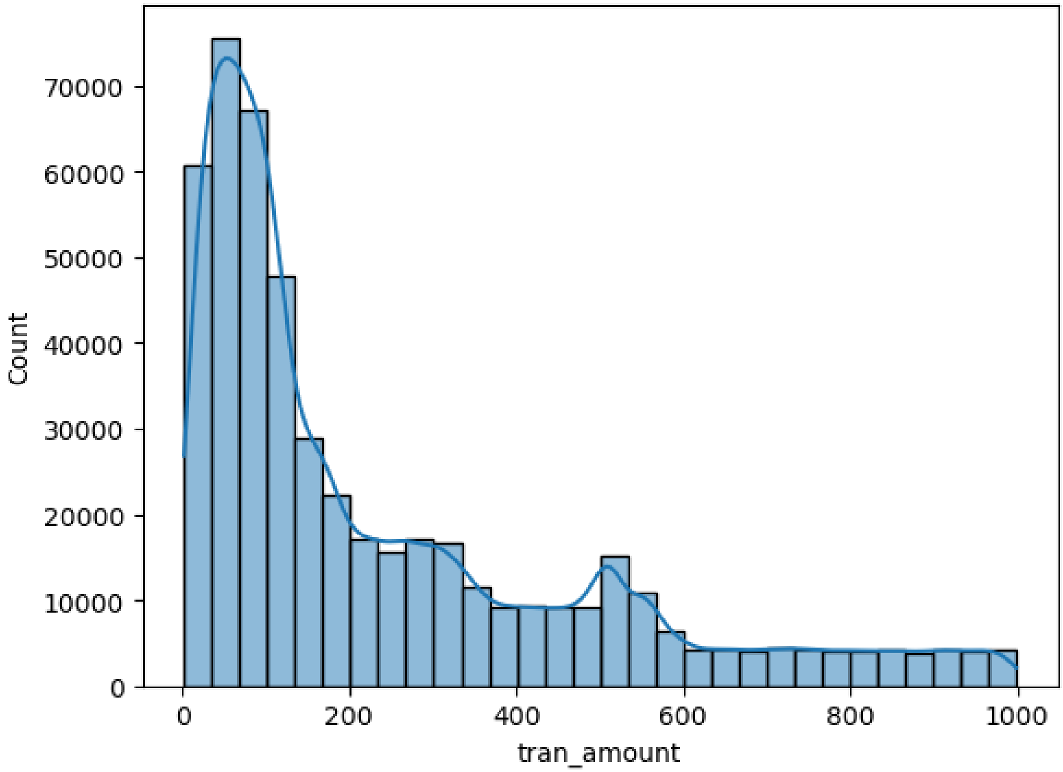
Out[204]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
26	27	380	2023-01-01	92.167205	Shopify	Beauty & Personal Care	Credit Card
49	50	287	2023-01-01	50.333298	Amazon	Toys & Games	Gpay

You can now see that we got rid of outliers from tran_amount column.

In [207]:

```
sns.histplot(df_trans.tran_amount,bins =30 , kde = True)
plt.show()
```



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 [211]:

```
df_trans.head()
```

Out[211]:

	tran_id	cust_id	tran_date	tran_amount	platform	product_category	payment_type
0	1	705	2023-01-01	63.0	Flipkart	Electronics	Phonepe
1	2	385	2023-01-01	99.0	Alibaba	Fashion & Apparel	Credit Card
2	3	924	2023-01-01	471.0	Shopify	Sports	Phonepe
3	4	797	2023-01-01	33.0	Shopify	Fashion & Apparel	Gpay
4	5	482	2023-01-01	68.0	Amazon	Fashion & Apparel	Net Banking

In [213]:

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

```
df_merged_2.head(3)
```

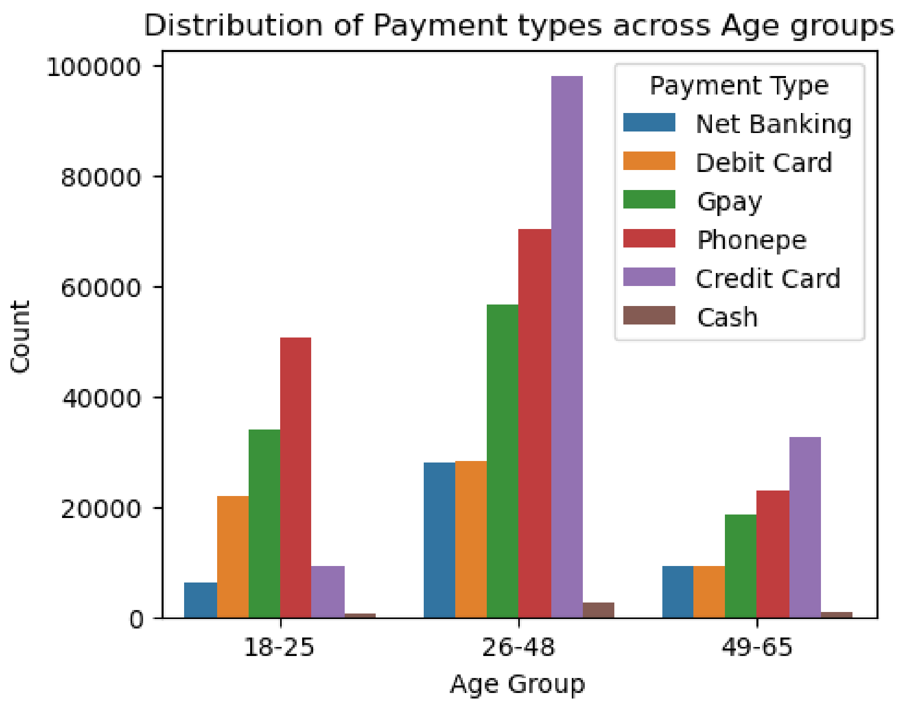
Out[213]:

	cust_id	name	gender	age	location	occupation	annual_income	marital_status	age_group	credit_score	...	credit_inc
0	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	Married	49-65	749	...	
1	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	Married	49-65	749	...	
2	1	Manya Acharya	Female	51.0	City	Business Owner	358211.0	Married	49-65	749	...	

3 rows × 22 columns

In [214]:

```
plt.figure(figsize=(5, 4))
sns.countplot(x='age_group', hue='payment_type', data=df_merged_2)
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()
```

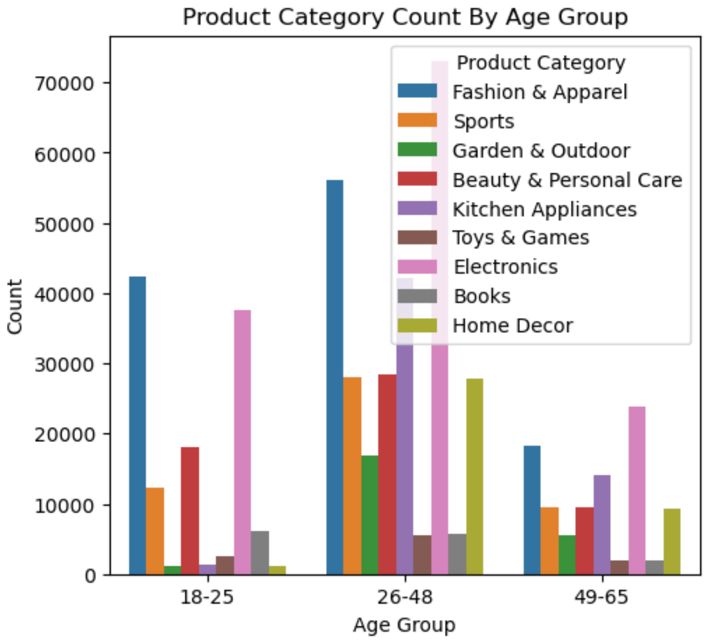


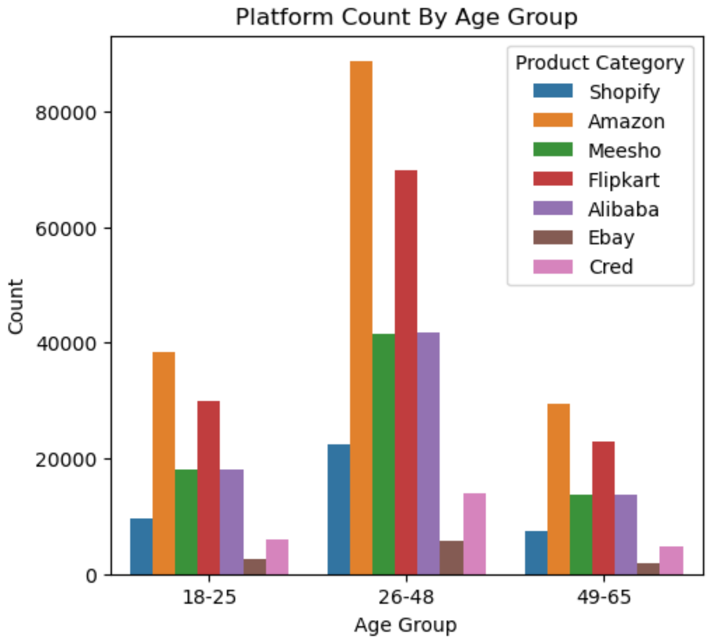
From above analysis, we can see that age group 18-25 has less exposure to credit cards compared to other groups

In [217]:

```
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()
```





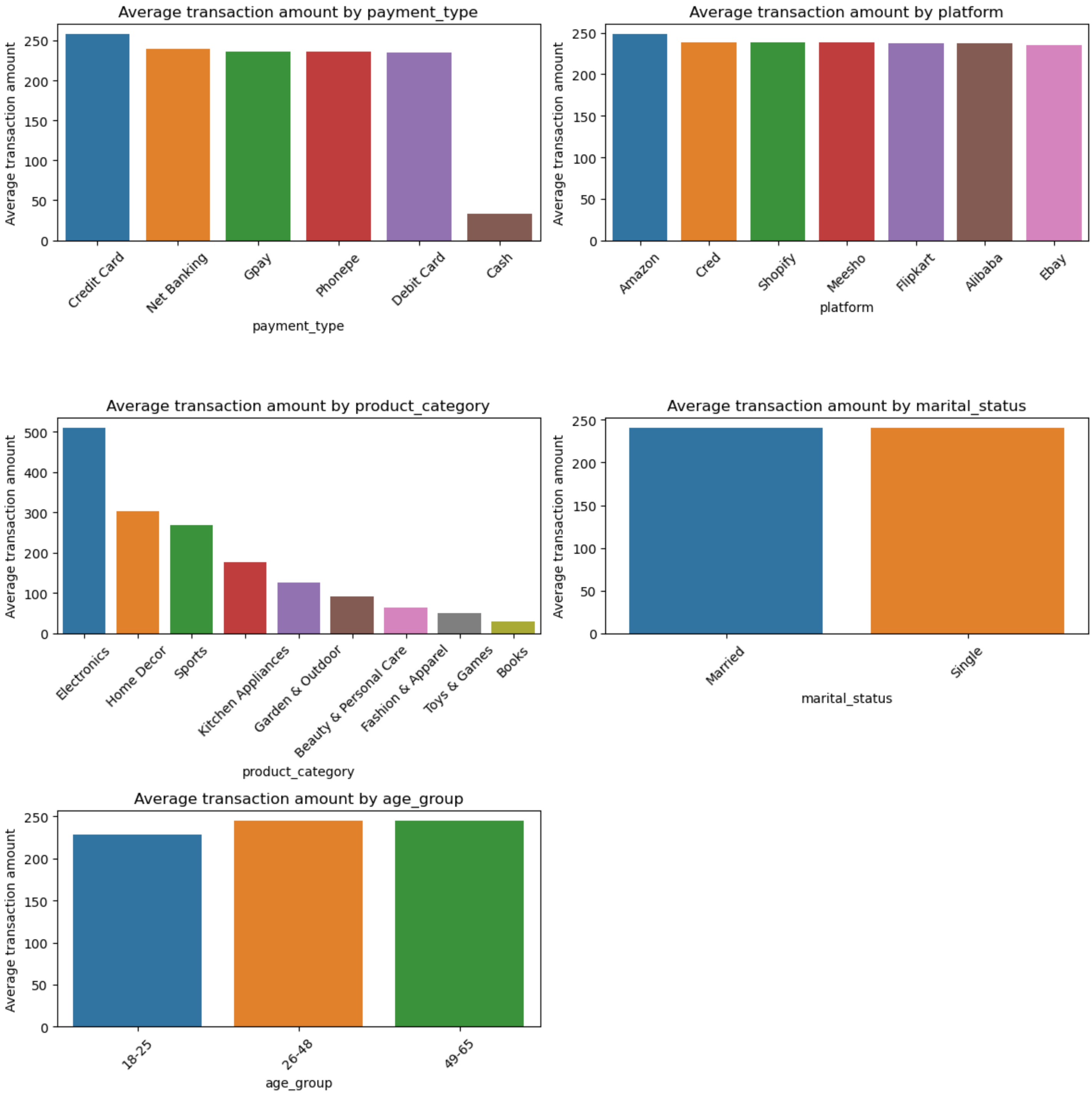
Observations:

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 [221]:

```
# 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))

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().res
et_index()

    # Sort the data in descending order by 'annual_income' before plotting
    sorted_data = avg_tran_amount_by_category.sort_values(by='tran_amount', ascending=Fa
lse)

    sns.barplot(x=cat_col, y='tran_amount', data=sorted_data, ci=None, ax=axes[i], palet
te='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')

    axes[i].set_xticklabels(axes[i].get_xticklabels(), rotation=45)

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



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 [223]:

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

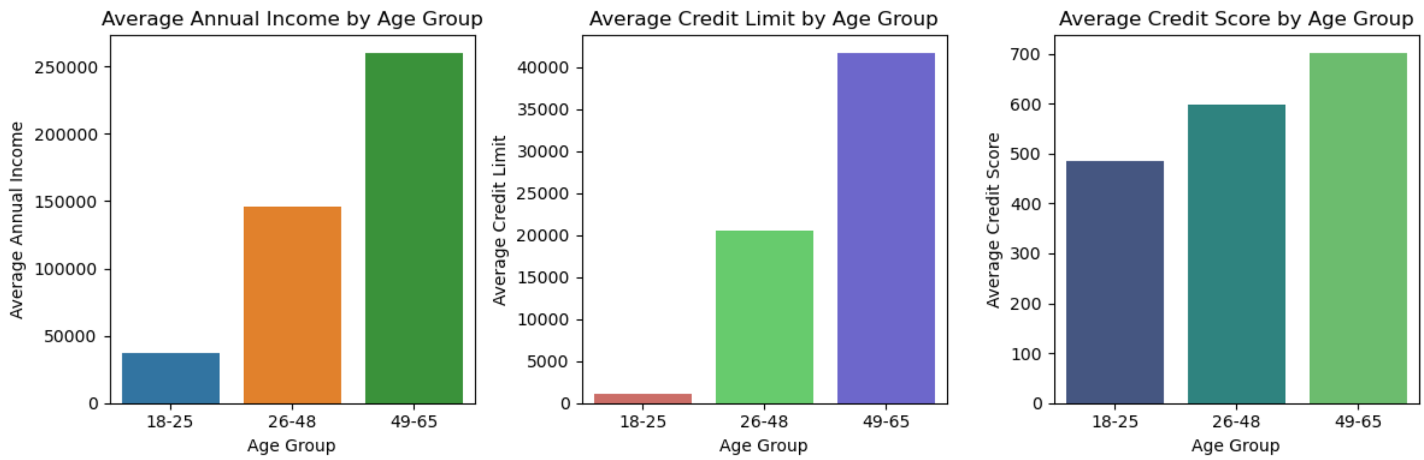
Out[223]:

	age_group	annual_income	credit_limit	credit_score
0	18-25	37091.235772	1130.081301	484.451220
1	26-48	145869.623457	20560.846561	597.569665
2	49-65	260165.925134	41699.197861	701.524064

In [227]:

```
# 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', a
x=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=a
x2)
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()
```



<h2 align="center", style="color:purple">Finalize Target Market For a Trial Credit Card Launch

Targeting Untapped market

1. **People with age group of 18 -25 accounts to ~25% 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**

Phase 2: Bank Credit Card Project

Business Analysis and launch of AB testing: Targeting Untapped Market (18 - 25 age group)

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

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.

In [234]:

```
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 [235]:

```
alpha = 0.05 # 5% significance
power = 0.8 #statistical power
effect_size=0.2

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

Out[235]:

```
393.4056989990335
```

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 [239]:

```
effect_sizes = [0.1, 0.2, 0.3, 0.4, 0.5,1]

for effect_size in effect_sizes:
    sample_size = sms.tt_ind_solve_power(effect_size=effect_size, alpha=alpha, power=pow
er, ratio=1, alternative='two-sided')
    print(f"Effect Size: {effect_size}, Required Sample Size: {int(sample_size)} customer
s")
```

```
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

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 [246]:

```
df = pd.read_csv('Datasets/avg_transactions_after_campaign.csv')
df.head()
```

Out[246]:

	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
4	2023-09-14	158.55	344.08

In [248]:

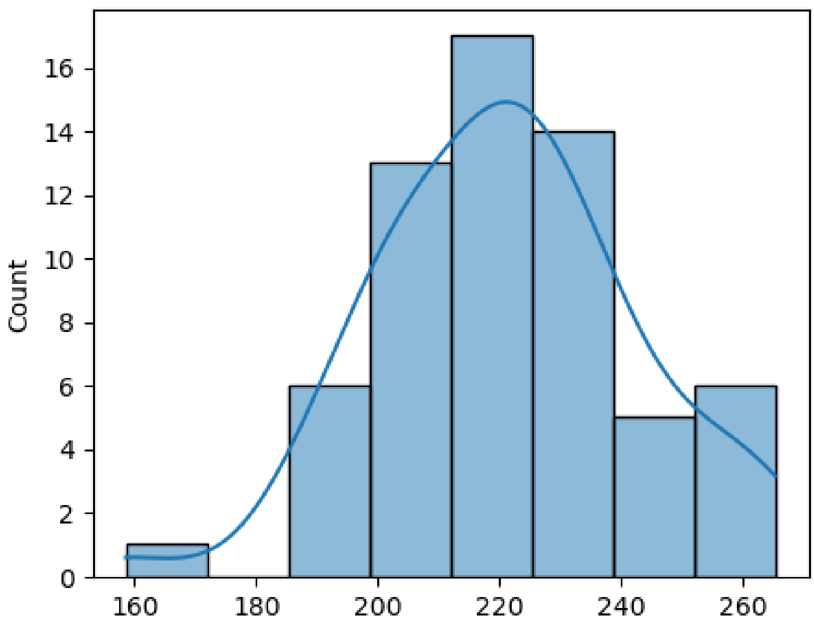
```
df.shape
```

Out[248]:

(62, 3)

In [250]:

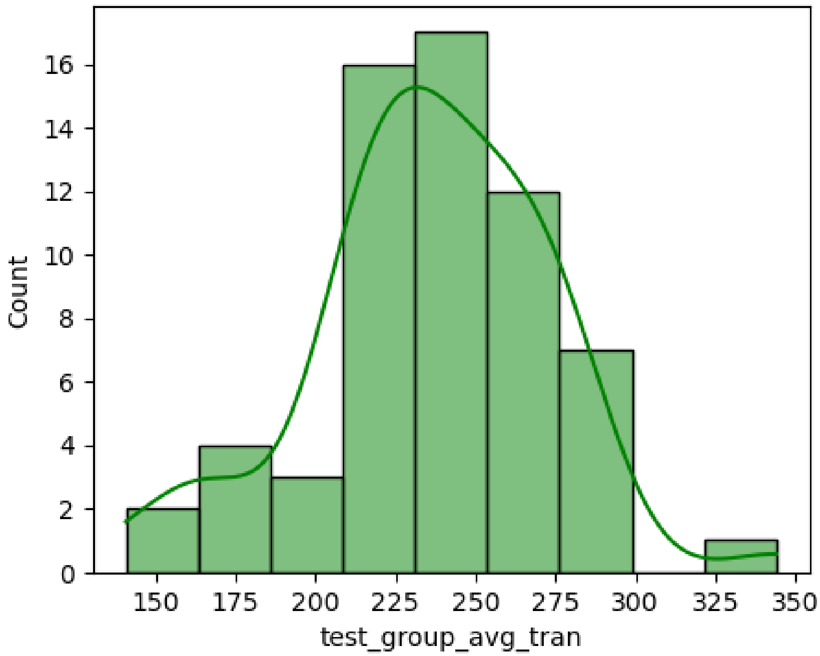
```
plt.figure(figsize=(5, 4))
sns.histplot(df.control_group_avg_tran ,kde=True)
plt.show()
```



control_group_avg_tran

In [252]:

```
plt.figure(figsize=(5, 4))
sns.histplot(df.test_group_avg_tran ,kde=True , color='green')
plt.show()
```



Defining hypothesis :

1. **Null Hypothesis: Old credit card has more transactions and performing well (mean of control_group > mean of test_group).**
2. **Alternate hypothesis : New credit card has more transactions and performing well (mean of test_group > mean of control_group).**

In [255]:

```
control_mean = df["control_group_avg_tran"].mean()
control_std = df["control_group_avg_tran"].std()
control_mean, control_std
```

Out[255]:

```
(221.1751612903226, 21.359192112027014)
```

In [257]:

```
test_mean = df["test_group_avg_tran"].mean()
test_std = df["test_group_avg_tran"].std()
test_mean, test_std
```

Out[257]:

```
(235.9835483870968, 36.65808210918637)
```

In [259]:

```
sample_size = df.shape[0]
sample_size
```

Out[259]:

```
62
```

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

In [262]:

```
control_variance = control_std ** 2 / sample_size
test_variance = test_std ** 2 / sample_size

z_score = (test_mean - control_mean) / np.sqrt(control_variance + test_variance)
z_score
```

Out[262]:

```
2.7482973745691135
```

In [264]:

```
# For a significance level of 5% (0.05) in a right-tailed test, the critical Z-value is a
pproximately 1.645
z_critical = st.norm.ppf(1 - alpha)  # Right-tailed test at 5% significance level
z_critical
```

Out[264]:

```
1.6448536269514722
```

In [266]:

```
z_score > z_critical
```

Out[266]:

```
True
```

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

Test Using p-Value

In [270]:

```
p_value = 1 - st.norm.cdf(z_score)
p_value
```

Out[270]:

```
0.0029952824622024865
```

In [272]:

```
p_value < alpha # p value is less than significance level of 5% (or 0.05 for absolute val
ue)
```

Out[272]:

```
True
```

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

After conducting two tests comparing the new and old credit card, the analysis validated the new credit card performs better and can now be confidently introduced to the market.