- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

Walmart Business Case Study

Importing the required libraries and packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats
from scipy.stats import kstest
import statsmodels.api as sm

from dateutil.parser import parse

import statistics
```

```
import statistics
from scipy.stats import norm
from tqdm import tqdm
```

Downloading the dataset

```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?1641285094
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?1641285094
To: /content/walmart_data.csv?1641285094
100% 23.0M/23.0M [00:00<00:00, 82.9MB/s]
```

```
df=pd.read_csv('walmart_data.csv?1641285094')
df
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Categor
0	1000001	P00069042	F	0-17	10	A	2	0	
1	1000001	P00248942	F	0-17	10	A	2	0	
2	1000001	P00087842	F	0-17	10	A	2	0	
3	1000001	P00085442	F	0-17	10	A	2	0	
4	1000002	P00285442	M	55+	16	C	4+	0	
...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	1	1	
550064	1006035	P00375436	F	26-35	1	C	3	0	
550065	1006036	P00375436	F	26-35	15	B	4+	1	
550066	1006038	P00375436	F	55+	1	C	2	0	
550067	1006039	P00371644	F	46-50	0	B	4+	1	

550068 rows × 10 columns

Shape

```
df.shape
```

```
(550068, 10)
```

```
rows, columns = df.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
Number of rows: 550068
Number of columns: 10
```

Data Types

```
df.dtypes
```

```
User_ID                        int64
Product_ID                    object
Gender                        object
Age                           object
Occupation                     int64
City_Category                 object
Stay_In_Current_City_Years    object
Marital_Status                 int64
Product_Category               int64
Purchase                       int64
dtype: object
```

Basic Information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     550068 non-null  int64
```

```
 1   Product_ID                  550068 non-null  object
 2   Gender                      550068 non-null  object
 3   Age                         550068 non-null  object
 4   Occupation                  550068 non-null  int64
 5   City_Category               550068 non-null  object
 6   Stay_In_Current_City_Years  550068 non-null  object
 7   Marital_Status              550068 non-null  int64
 8   Product_Category            550068 non-null  int64
 9   Purchase                    550068 non-null  int64
dtypes: int64(5), object(5)
memory usage: 42.0+ MB
```

Finding Null Values

```
missing_values = df.isnull().sum()
print(missing_values)
```

```
User_ID                       0
Product_ID                    0
Gender                        0
Age                           0
Occupation                    0
City_Category                 0
Stay_In_Current_City_Years    0
Marital_Status                0
Product_Category              0
Purchase                      0
dtype: int64
```

There are no null values

Number of Unique Values

```
df.nunique()
```

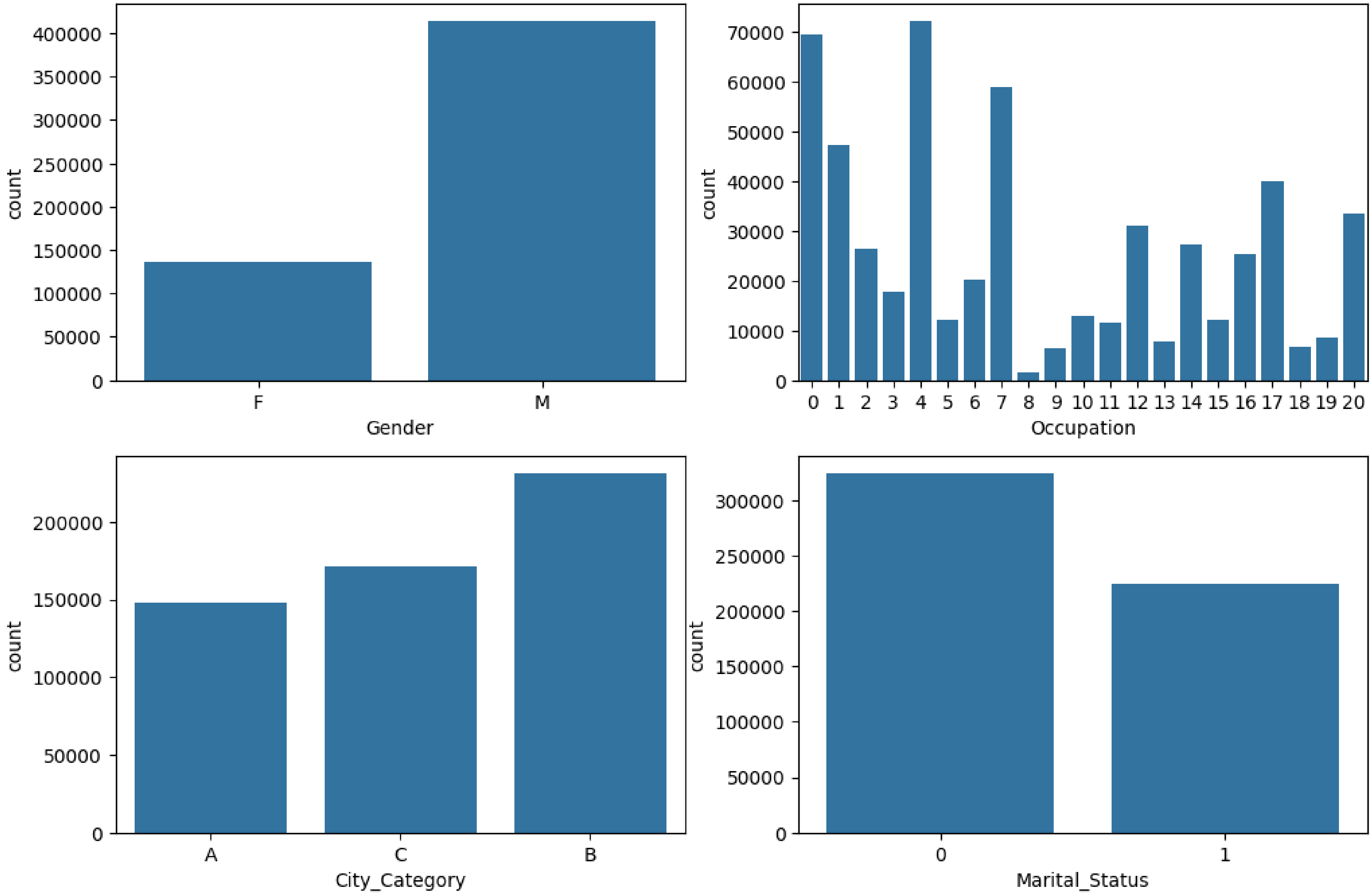
```
User_ID                        5891
Product_ID                     3631
Gender                            2
Age                               7
Occupation                       21
City_Category                     3
Stay_In_Current_City_Years        5
Marital_Status                    2
Product_Category                 20
Purchase                      18105
dtype: int64
```

```
df.describe(include='all')
```

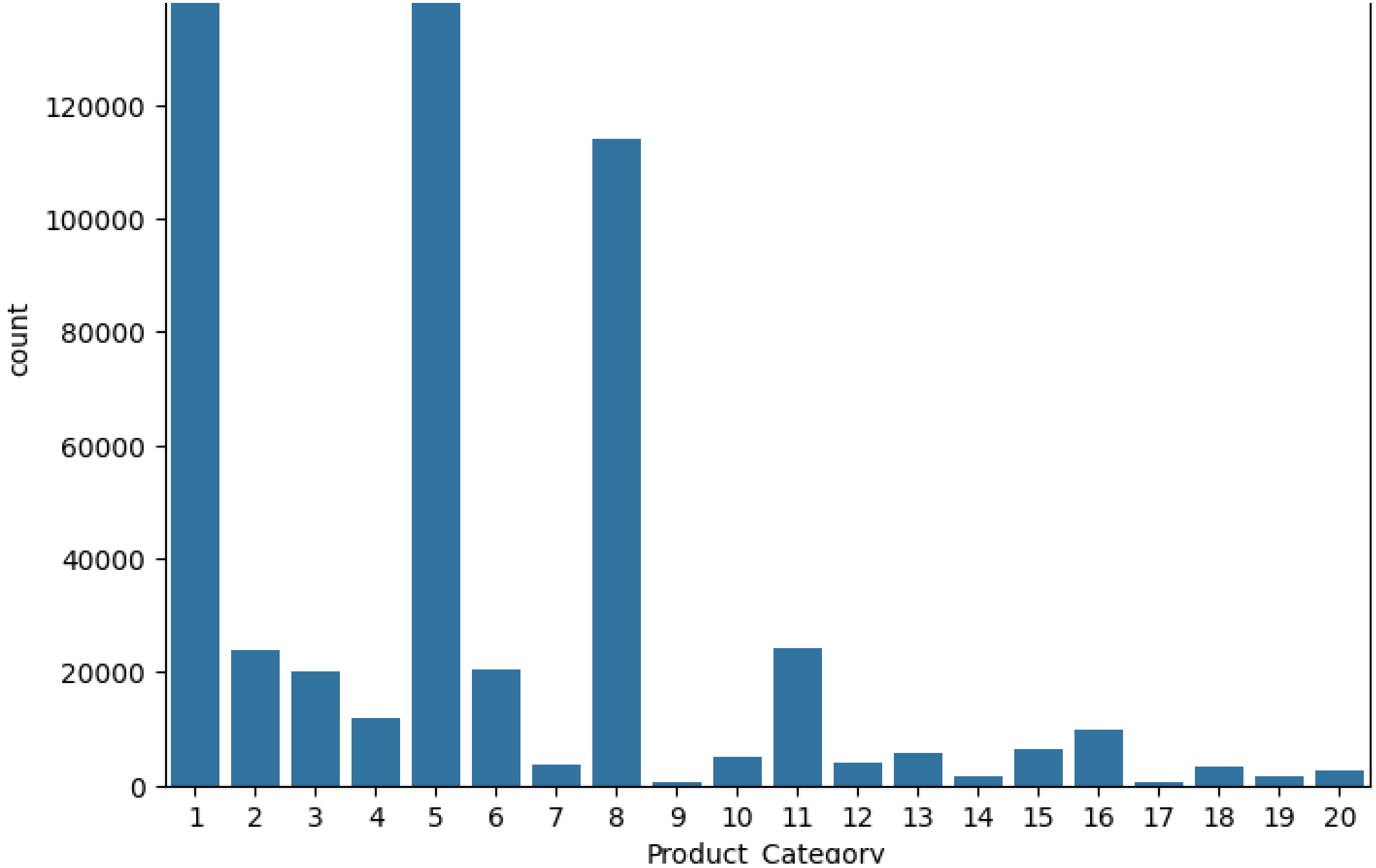
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Prod
count	5.500680e+05	550068	550068	550068	550068.000000	550068	550068	550068.000000	5
unique	NaN	3631	2	7	NaN	3	5	NaN	
top	NaN	P00265242	M	26-35	NaN	B	1	NaN	
freq	NaN	1880	414259	219587	NaN	231173	193821	NaN	
mean	1.003029e+06	NaN	NaN	NaN	8.076707	NaN	NaN	0.409653	
std	1.727592e+03	NaN	NaN	NaN	6.522660	NaN	NaN	0.491770	
min	1.000001e+06	NaN	NaN	NaN	0.000000	NaN	NaN	0.000000	
25%	1.001516e+06	NaN	NaN	NaN	2.000000	NaN	NaN	0.000000	
50%	1.003077e+06	NaN	NaN	NaN	7.000000	NaN	NaN	0.000000	
75%	1.004478e+06	NaN	NaN	NaN	14.000000	NaN	NaN	1.000000	
max	1.006040e+06	NaN	NaN	NaN	20.000000	NaN	NaN	1.000000	

Uni-Variate Analysis

```
categorical_cols = ['Gender', 'Occupation','City_Category','Marital_Status','Product_Category']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(12,8))
sns.countplot(data=df, x='Gender', ax=axs[0,0])
sns.countplot(data=df, x='Occupation', ax=axs[0,1])
sns.countplot(data=df, x='City_Category', ax=axs[1,0])
sns.countplot(data=df, x='Marital_Status', ax=axs[1,1])
plt.show()

plt.figure(figsize=(8,6))
sns.countplot(data=df, x='Product_Category')
plt.show()
```


count
Gender
F
M
Occupation
City_Category
A
C
B
Marital_Status

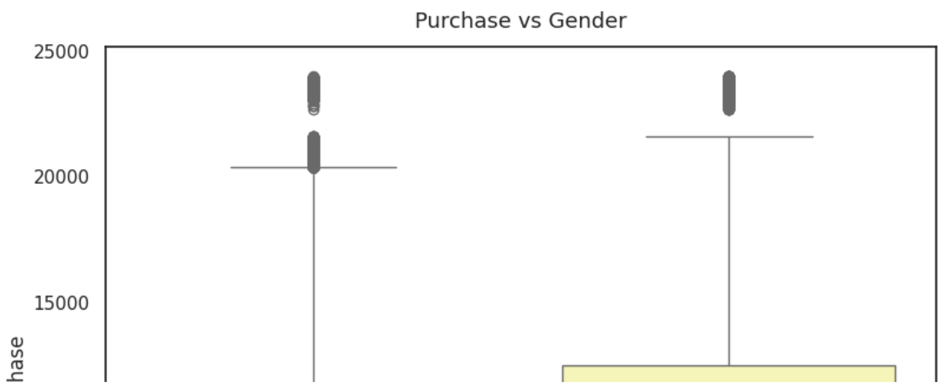

140000

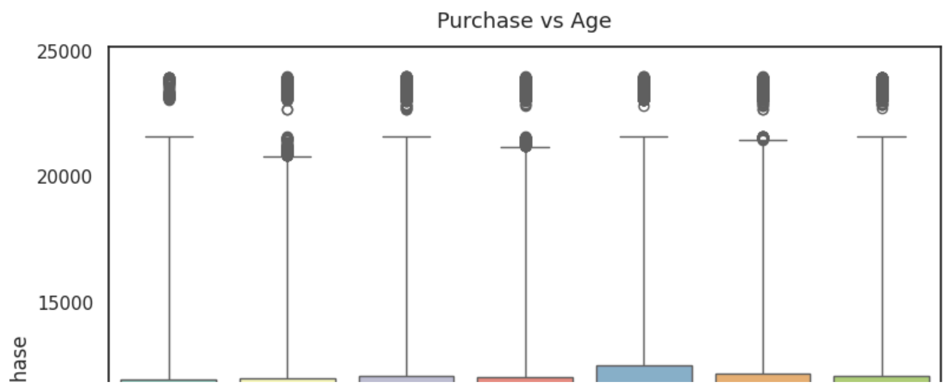


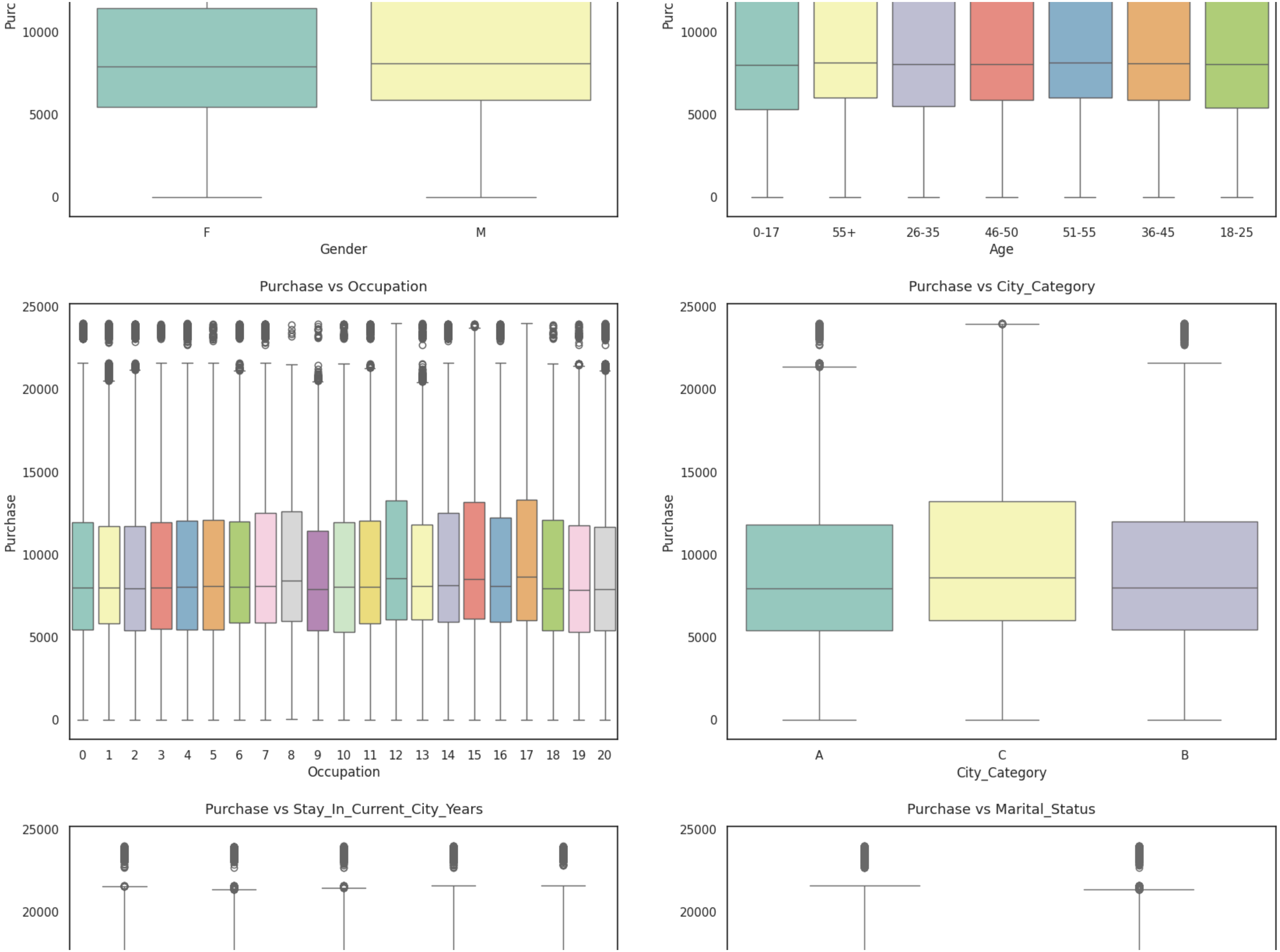
Bi-Variate Analysis

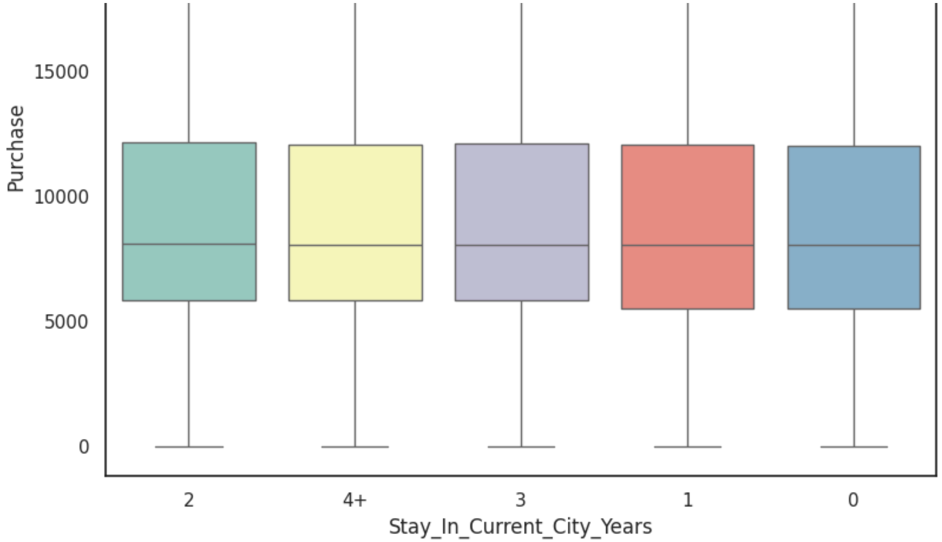
```
attrs = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']
sns.set_style("white")

fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(20, 16))
fig.subplots_adjust(top=1.3)
count = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
        axs[row,col].set_title(f"Purchase vs {attrs[count]}", pad=12, fontsize=13)
        count += 1
plt.show()

plt.figure(figsize=(10, 8))
sns.boxplot(data=df, y='Purchase', x=attrs[-1], palette='Set3')
plt.show()
```

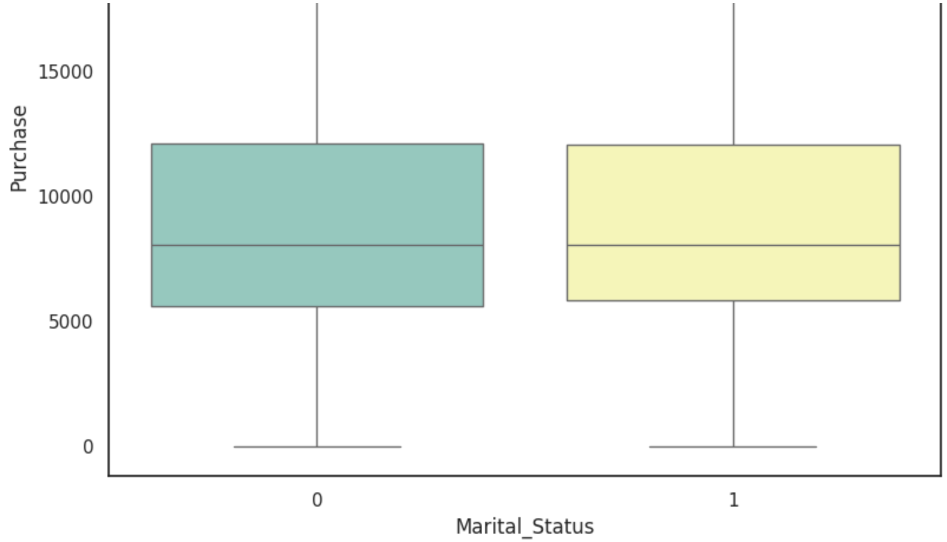
```
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
<ipython-input-33-8c207d598a9a>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[count], ax=axs[row, col], palette='Set3')
```



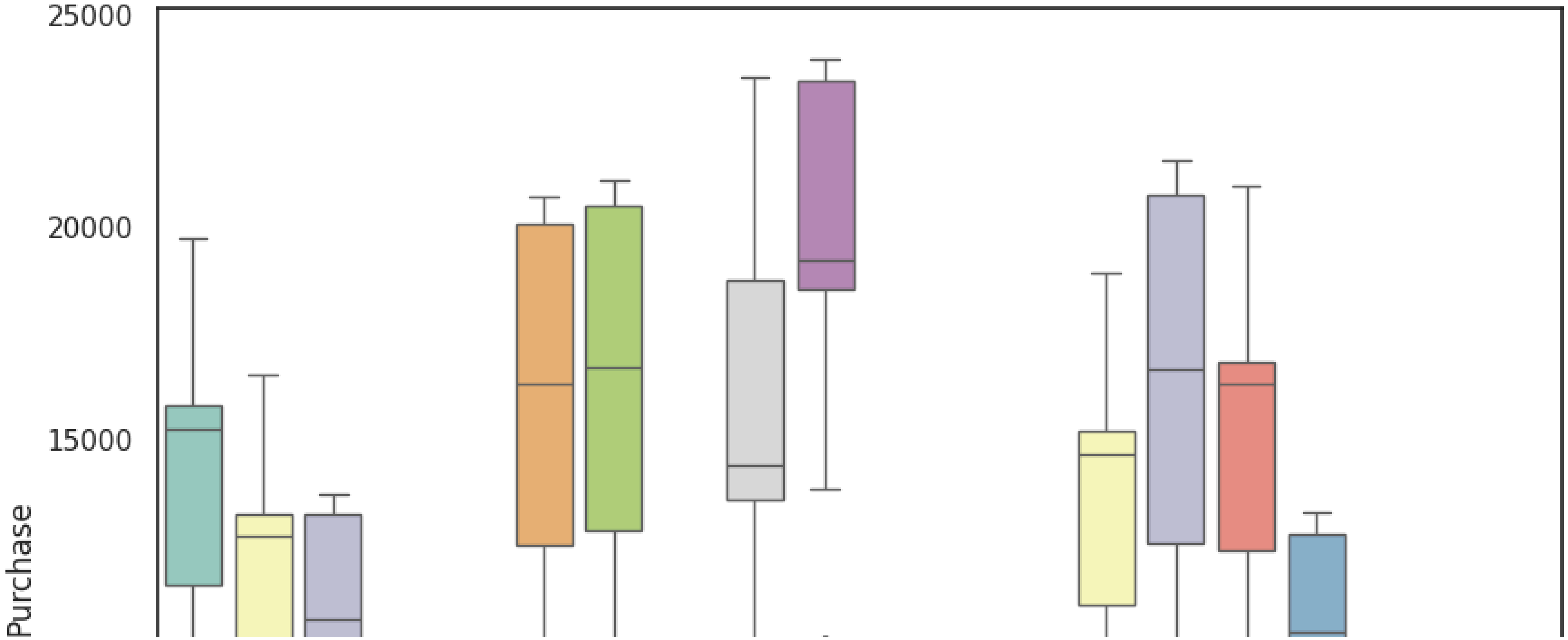


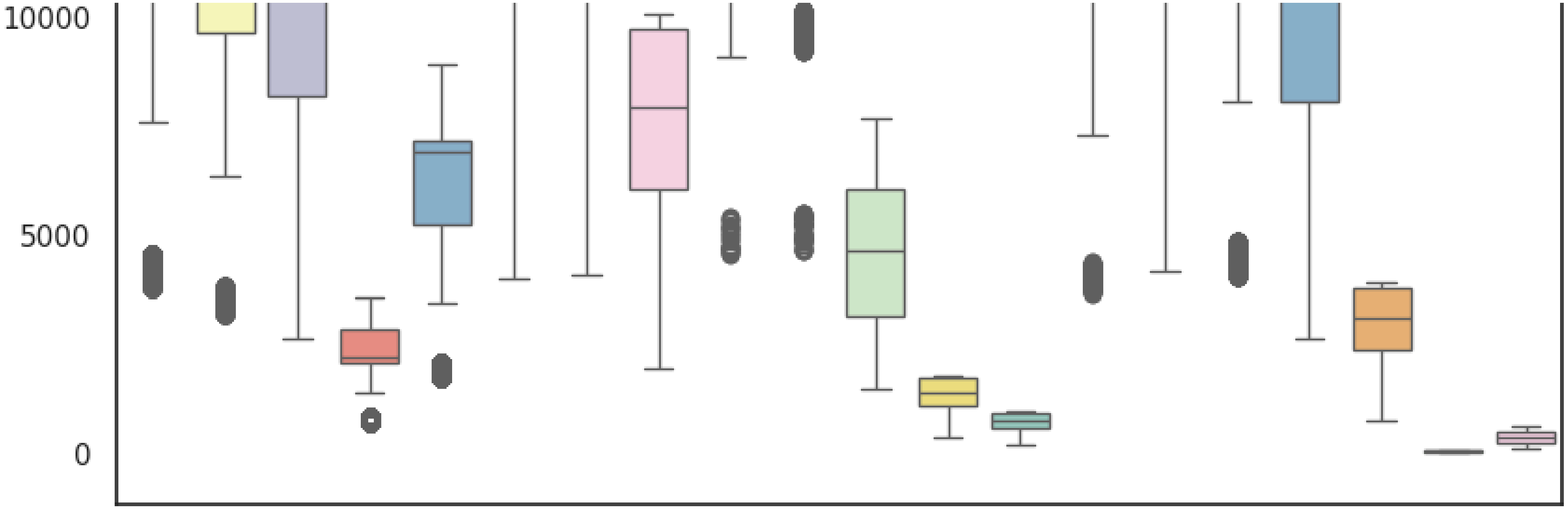

Purc
10000
5000
0
F
M
Gender
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age
Purchase vs Occupation
25000
20000
15000
10000
5000
0
Purchase
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Occupation
Purchase vs City_Category
A
C
B
City_Category
Purchase vs Stay_In_Current_City_Years
Purchase vs Marital_Status





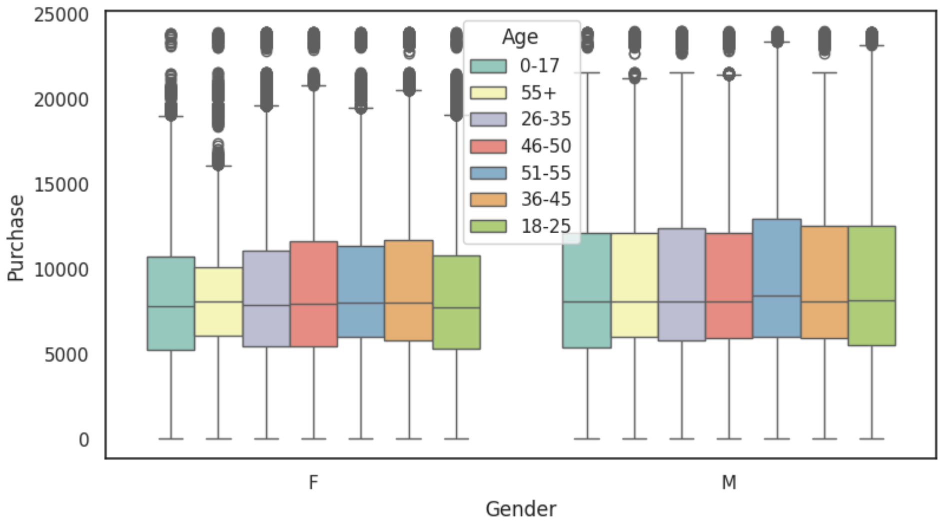

```
<ipython-input-33-8c207d598a9a>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(data=df, y='Purchase', x=attrs[-1], palette='Set3')
```

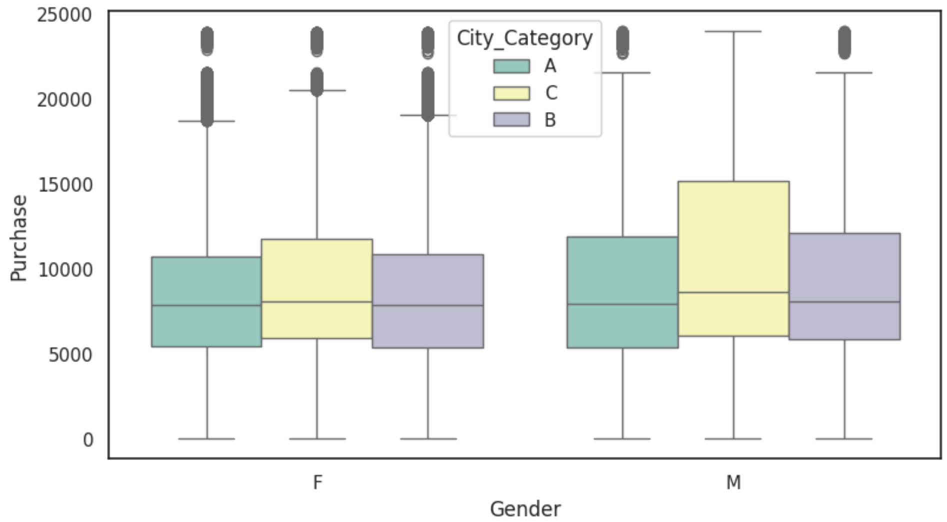


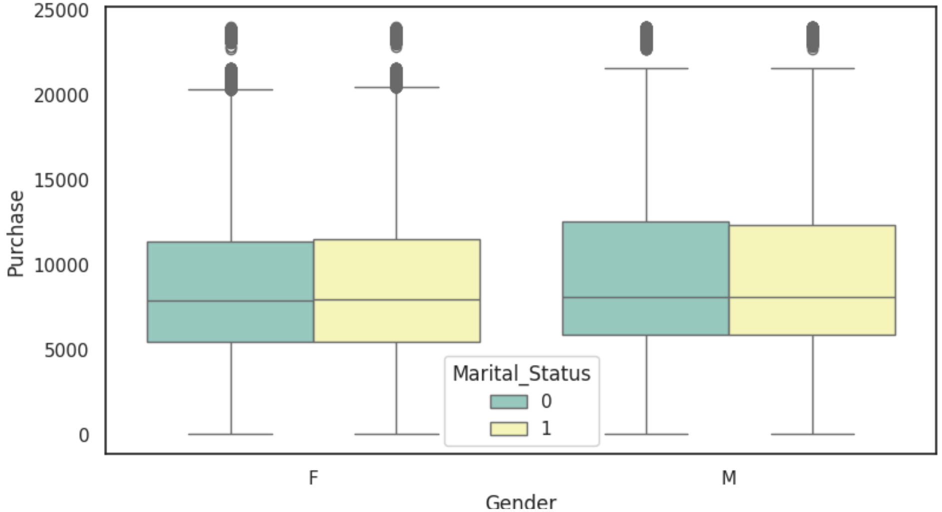


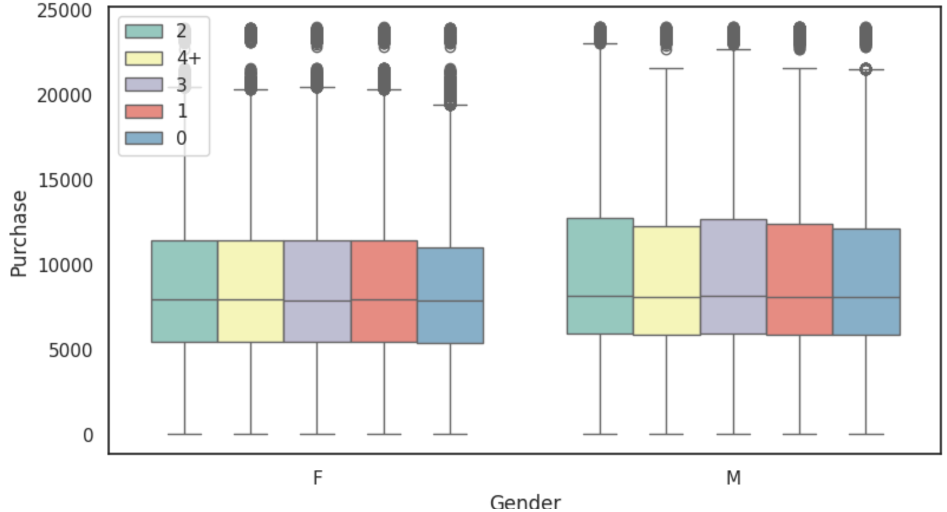
Multi-Variate Analysis

```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 6))
fig.subplots_adjust(top=1.5)
sns.boxplot(data=df, y='Purchase', x='Gender', hue='Age', palette='Set3', ax=axs[0,0])
sns.boxplot(data=df, y='Purchase', x='Gender', hue='City_Category', palette='Set3', ax=axs[0,1])

sns.boxplot(data=df, y='Purchase', x='Gender', hue='Marital_Status', palette='Set3', ax=axs[1,0])
sns.boxplot(data=df, y='Purchase', x='Gender', hue='Stay_In_Current_City_Years', palette='Set3', ax=axs[1,1])
axs[1,1].legend(loc='upper left')

plt.show()
```









Analysing each column

```
categorical_cols = ['Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Catego
df[categorical_cols].melt().groupby(['variable', 'value'])[['value']].count()/len(df)
```

		value
variable	value	
Age	0-17	0.027455
	18-25	0.181178
	26-35	0.399200
	36-45	0.199999
	46-50	0.083082
	51-55	0.069993
	55+	0.039093
City_Category	A	0.268549
	B	0.420263
	C	0.311189
Gender	F	0.246895
	M	0.753105
Marital_Status	0	0.590347
	1	0.409653
Occupation	0	0.126599
	1	0.086218
	2	0.048336
	3	0.032087
	4	0.131453
	5	0.022137
	6	0.037005

	7	0.107501
	8	0.002811
	9	0.011437
	10	0.023506
	11	0.021063
	12	0.056682
	13	0.014049
	14	0.049647
	15	0.022115
	16	0.046123
	17	0.072796
	18	0.012039
	19	0.015382
	20	0.061014
Product_Category	**1**	0.255201
	2	0.043384
	3	0.036746
	4	0.021366
	5	0.274390
	6	0.037206
	7	0.006765
	8	0.207111
	9	0.000745

	9	[illegible]
	10	0.009317
	11	0.044153
	12	0.007175
	13	0.010088
	14	0.002769
	15	0.011435
	16	0.017867
	17	0.001051
	18	0.005681
	19	0.002914
	20	0.004636
Stay_In_Current_City_Years	**0**	0.135252
	1	0.352358
	2	0.185137
	3	0.173224
	4+	0.154028

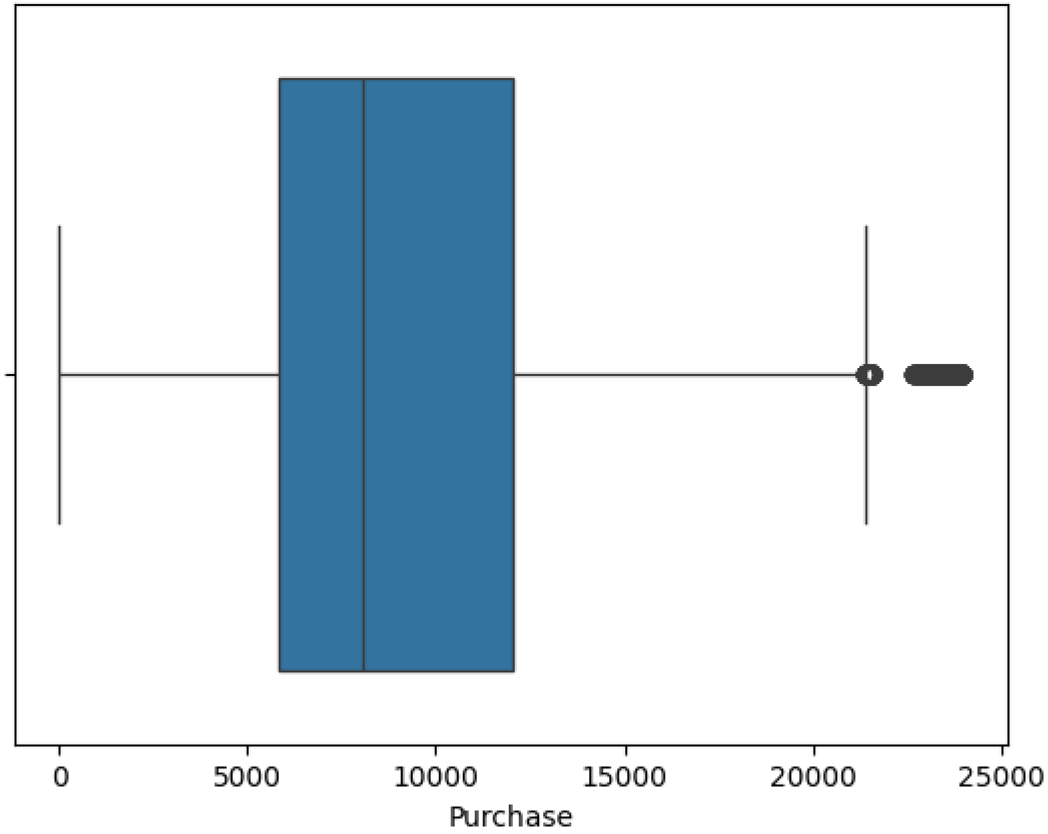
Observations

- 80% of the users are between the age 18-50 (40%: 26-35, 18%: 18-25, 20%: 36-45)
- 75% of the users are Male and 25% are Female
- 60% Single, 40% Married
- 35% Staying in the city from 1 year, 18% from 2 years, 17% from 3 years
- Total of 20 product categories are there

- There are 20 differnent types of occupations in the city

Checking for Outliers- Box Plot

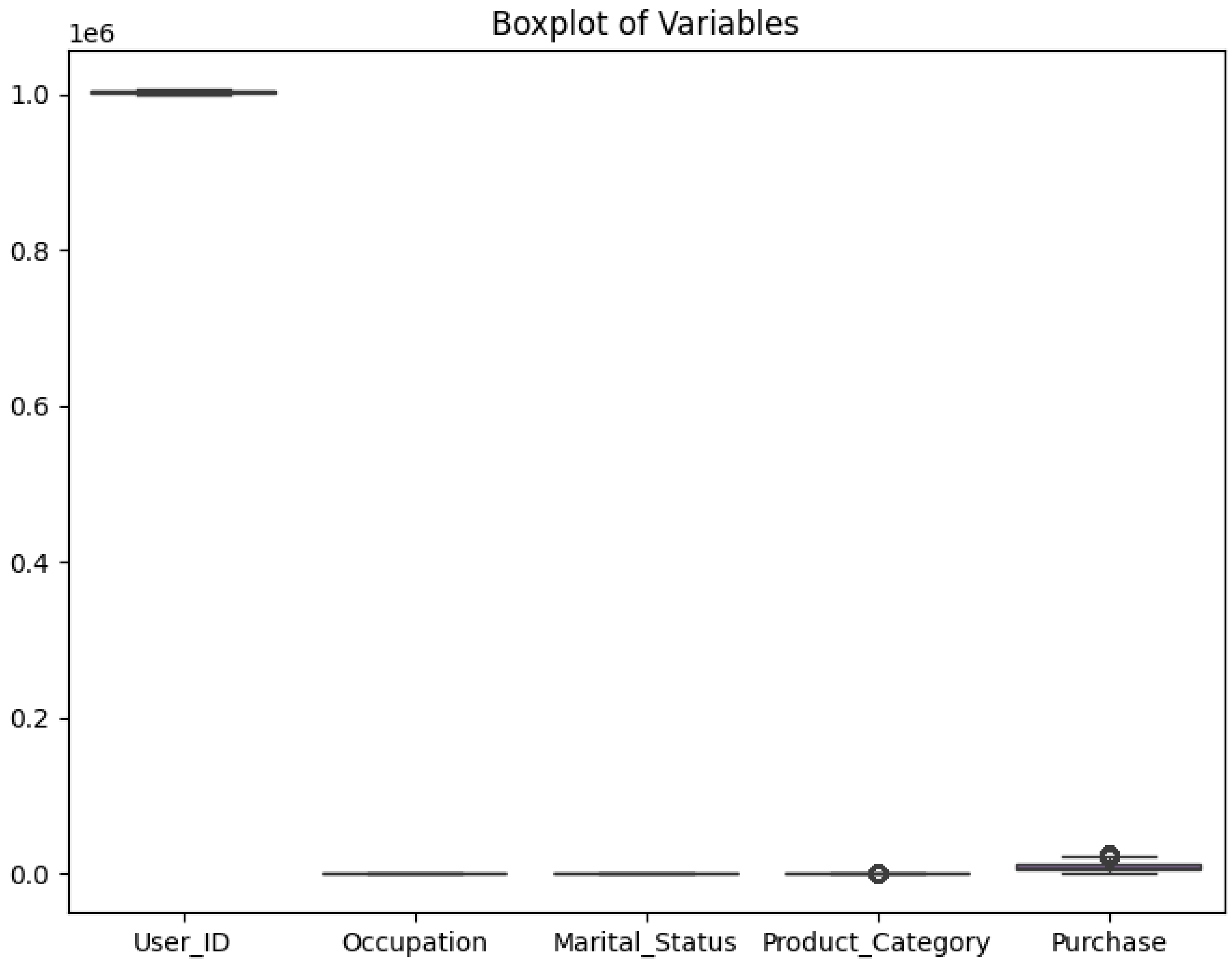
```
sns.boxplot(data=df, x='Purchase', orient='h')
plt.show()
```



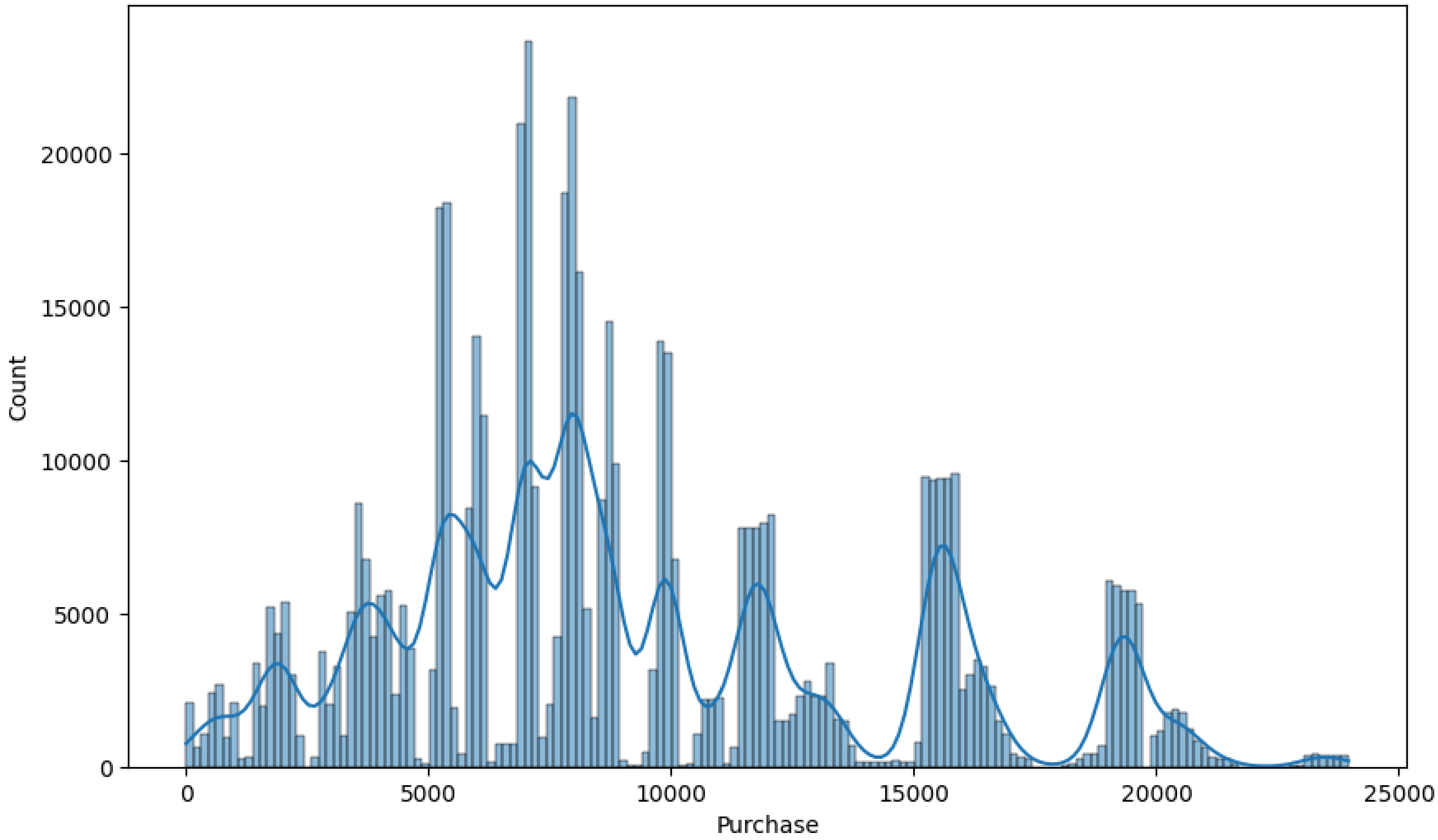
Observation: Purchase has outliers

```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df.select_dtypes(include=[np.number]))
```

```
plt.title("Boxplot of Variables")
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Purchase', kde=True)
plt.show()
```



Observations:

1. Predominantly, the dataset consists of male users.
2. The dataset encompasses 20 distinct categories for both Occupation and Product_Category.
3. The majority of users are affiliated with City_Category B.
4. There is a higher prevalence of single users compared to married ones.

5. Product_Category 1, 5, 8, and 11 exhibit the highest purchasing frequency.

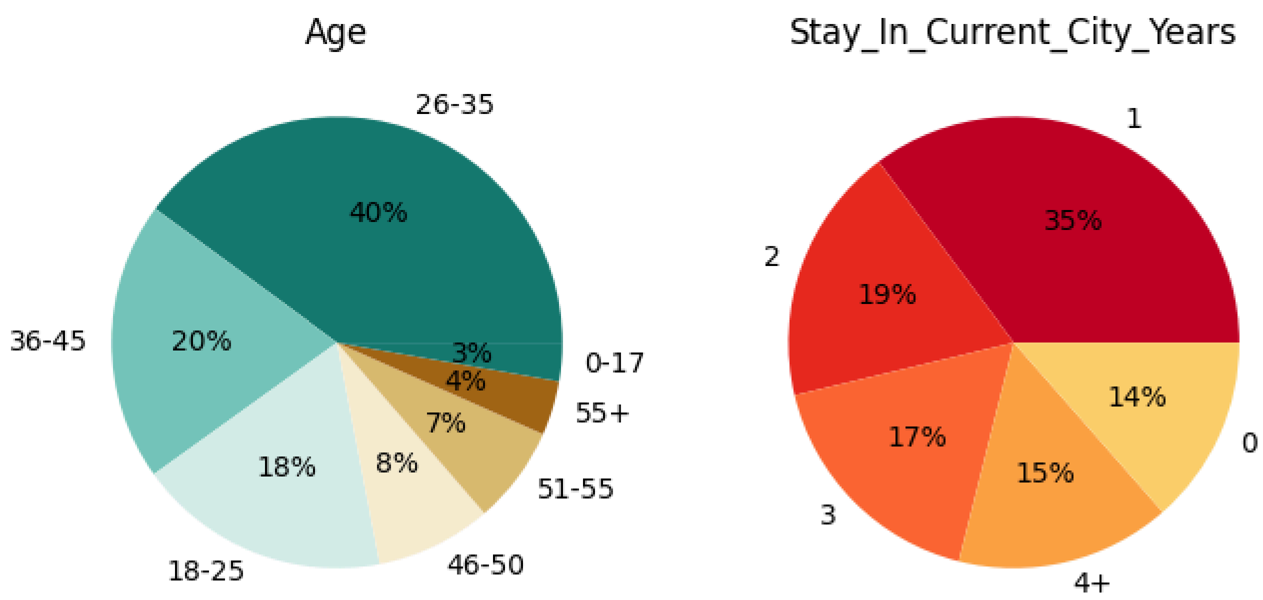
Remove/clip the data between the 5 percentile and 95 percentile

```
df_clipped = df.select_dtypes(include=[np.number]).apply(lambda x: np.clip(x, x.quantile(0.05), x.quantile(0.95)))

df_clipped = pd.concat([df.select_dtypes(exclude=[np.number]), df_clipped], axis=1)
df_clipped
```

	Product_ID	Gender	Age	City_Category	Stay_In_Current_City_Years	User_ID	Occupation	Marital_Status	Product_Categor
0	P00069042	F	0-17	A	2	1000329	10	0	
1	P00248942	F	0-17	A	2	1000329	10	0	
2	P00087842	F	0-17	A	2	1000329	10	0	
3	P00085442	F	0-17	A	2	1000329	10	0	
4	P00285442	M	55+	C	4+	1000329	16	0	
...	...	...	...	...	...	...	...	...	
550063	P00372445	M	51-55	B	1	1005747	13	1	
550064	P00375436	F	26-35	C	3	1005747	1	0	
550065	P00375436	F	26-35	B	4+	1005747	15	1	
550066	P00375436	F	55+	C	2	1005747	1	0	
550067	P00371644	F	46-50	B	4+	1005747	0	1	

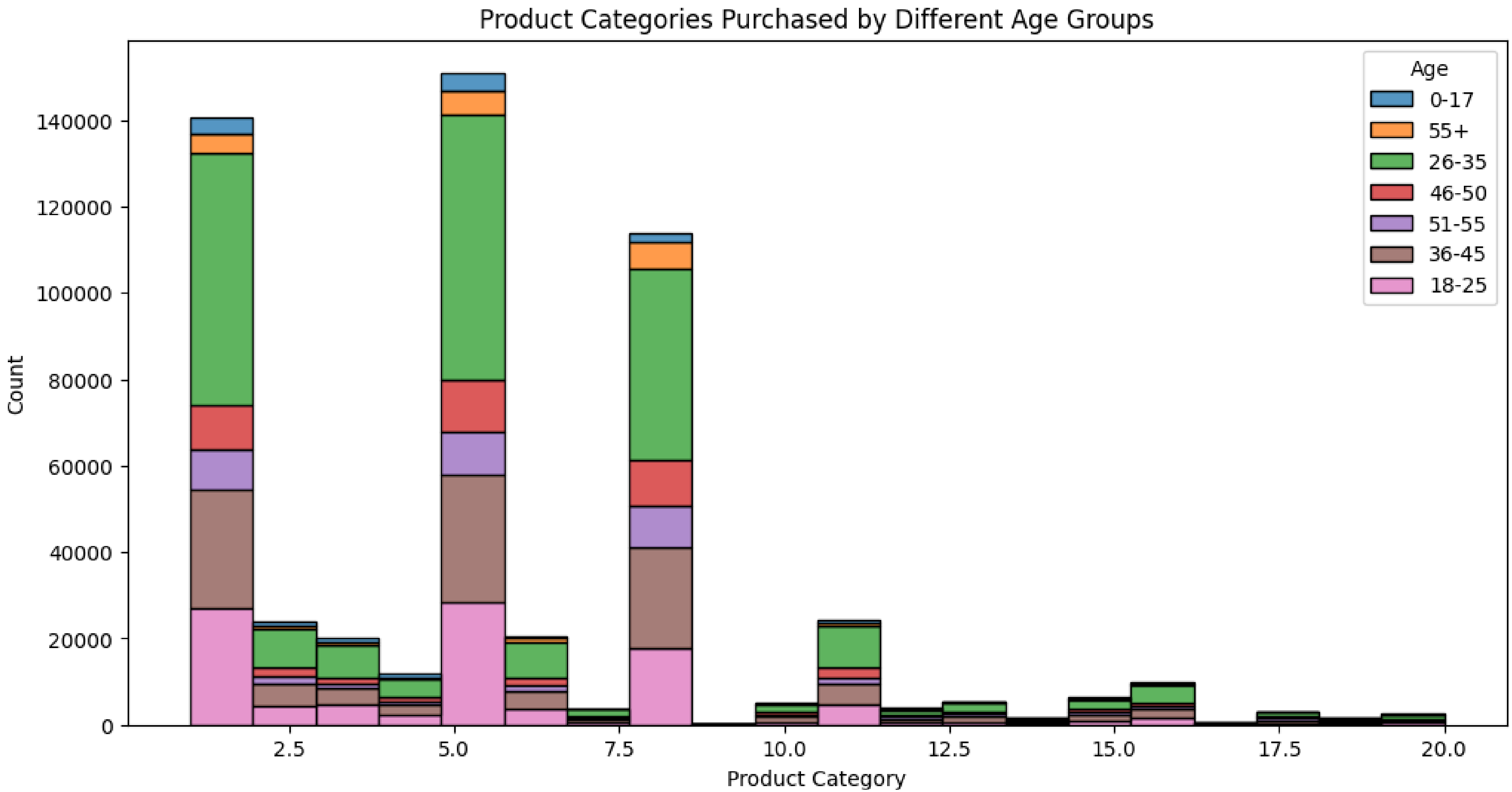
550068 rows × 10 columns

```
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(8, 8))

data = df['Age'].value_counts(normalize=True)*100
palette_color = sns.color_palette('BrBG_r')
axs[0].pie(x=data.values, labels=data.index, autopct='%.0f%%', colors=palette_color)
axs[0].set_title("Age")

data = df['Stay_In_Current_City_Years'].value_counts(normalize=True)*100
palette_color = sns.color_palette('YlOrRd_r')
axs[1].pie(x=data.values, labels=data.index, autopct='%.0f%%', colors=palette_color)
axs[1].set_title("Stay_In_Current_City_Years")
plt.show()
```



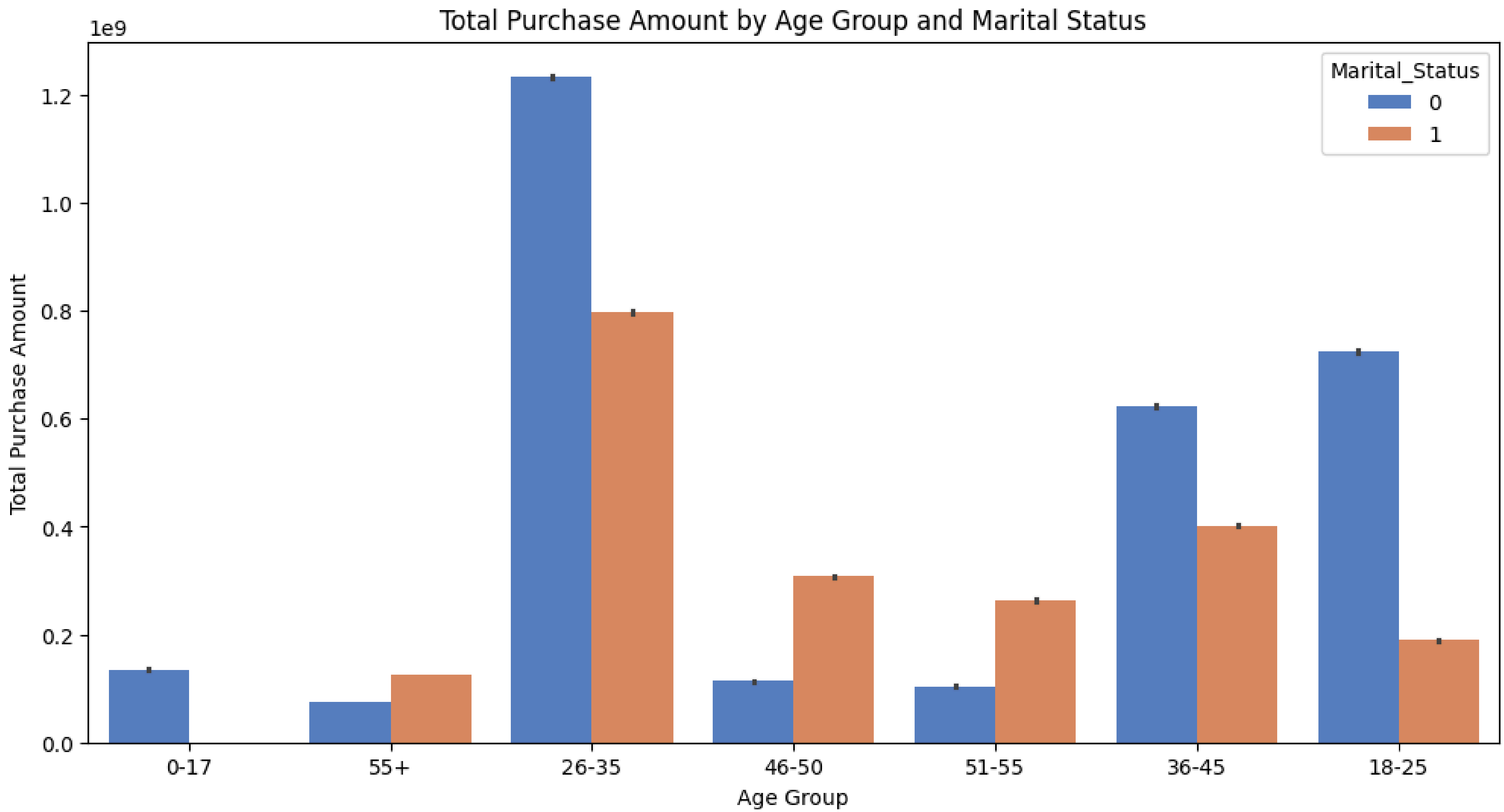
Product Categories Purchased by Different Age Groups

```
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='Product_Category', hue='Age', multiple='stack', bins=20, edgecolor='black')
plt.title('Product Categories Purchased by Different Age Groups')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.show()
```



Relationship between age, marital status, and the amount spent

```
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Age', y='Purchase', hue='Marital_Status', estimator=sum, palette='muted')
plt.title('Total Purchase Amount by Age Group and Marital Status')
plt.xlabel('Age Group')
plt.ylabel('Total Purchase Amount')
plt.show()
```

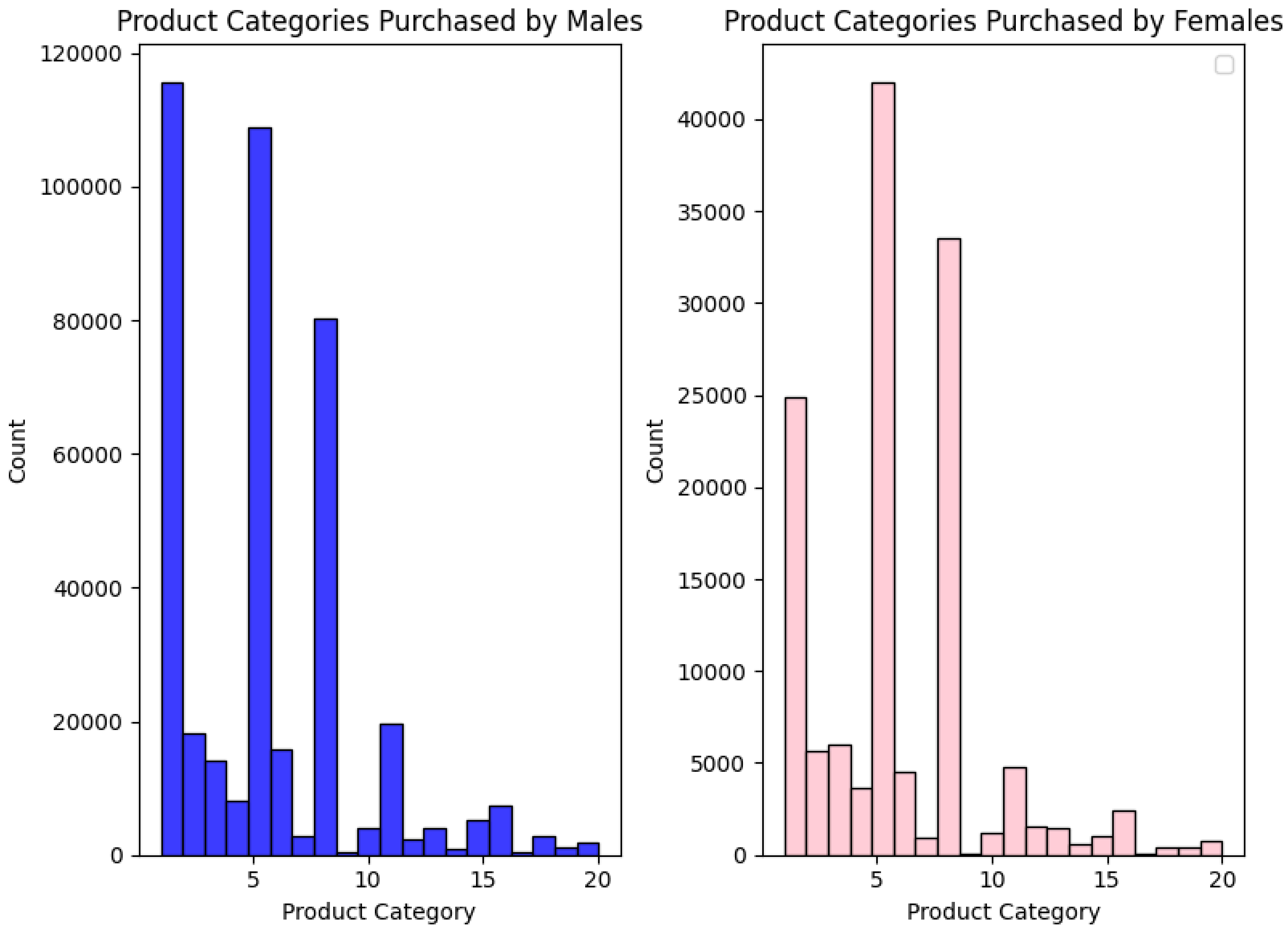


Observation: Maximum purchases are of unmarried people(marital status=0) in the age group of 26-35

Preferred product categories for different genders

```
plt.figure(figsize=(8,6))

# For Male users
plt.subplot(1, 2, 1)
sns.histplot(data=df[df['Gender'] == 'M'], x='Product_Category', bins=20, color='blue', edgecolor='black')
plt.title('Product Categories Purchased by Males')
plt.xlabel('Product Category')
plt.ylabel('Count')

# For Female users
plt.subplot(1, 2, 2)
sns.histplot(data=df[df['Gender'] == 'F'], x='Product_Category', bins=20, color='pink', edgecolor='black')
plt.title('Product Categories Purchased by Females')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.legend()
plt.tight_layout()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



The maximum Product Category purchased by Males is from 0-5, while for Females its from 5-10

Purchases by Males/Females

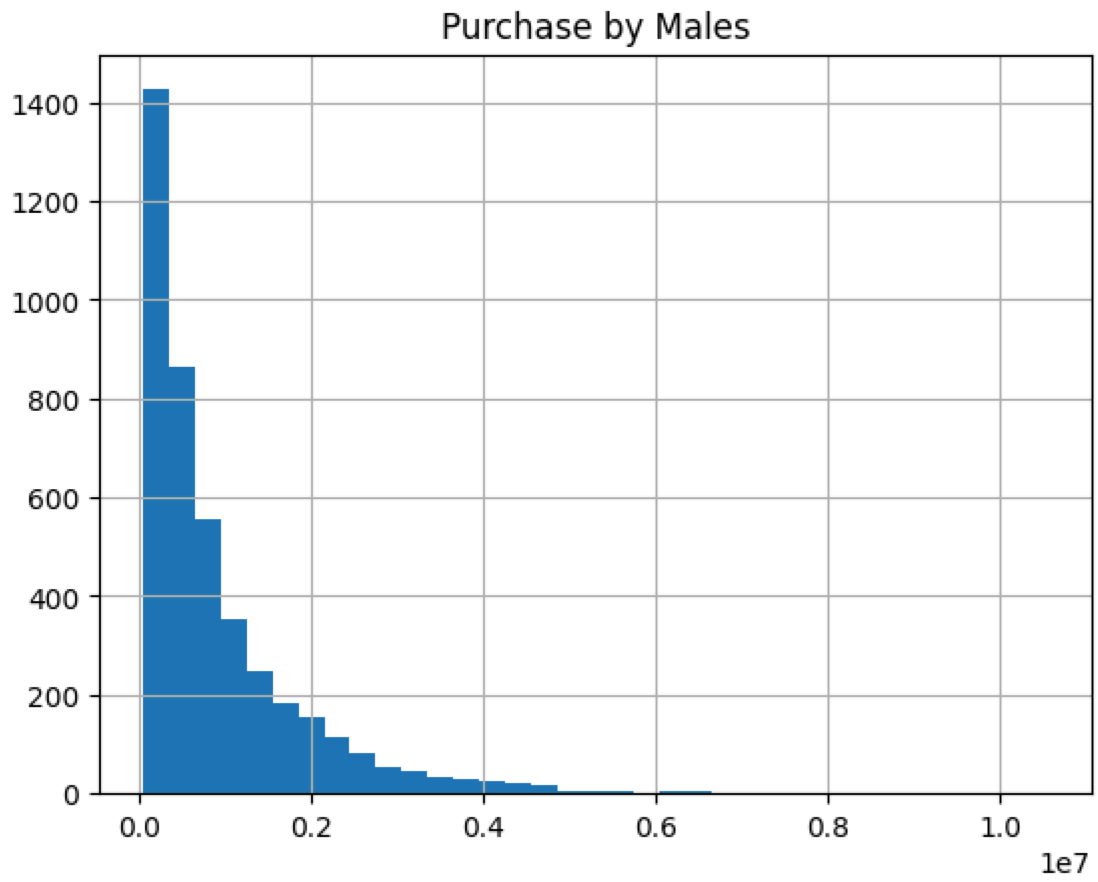
```
amt_df = df.groupby(['User_ID', 'Gender'])[['Purchase']].sum()
amt_df = amt_df.reset_index()
amt_df
```

	User_ID	Gender	Purchase
0	1000001	F	334093
1	1000002	M	810472
2	1000003	M	341635
3	1000004	M	206468
4	1000005	M	821001
...	...	...	...
5886	1006036	F	4116058
5887	1006037	F	1119538
5888	1006038	F	90034
5889	1006039	F	590319
5890	1006040	M	1653299

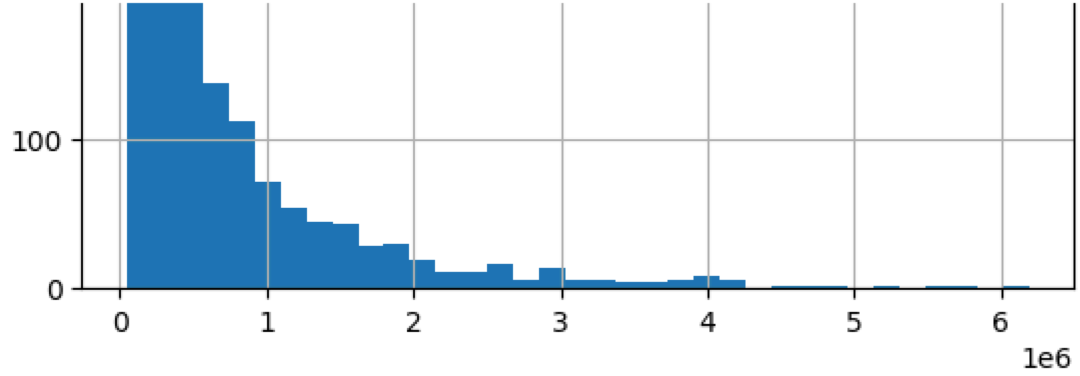
5891 rows × 3 columns

Next steps: Generate code with `amt_df` View recommended plots

```
amt_df[amt_df['Gender']=='M']['Purchase'].hist(bins=35)
plt.title('Purchase by Males')
plt.show()

amt_df[amt_df['Gender']=='F']['Purchase'].hist(bins=35)
plt.title('Purchase by Females')
plt.show()
```


Purchase by Males
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
1e7


Purchase by Females
400
300
200




```
male_avg = amt_df[amt_df['Gender']=='M']['Purchase'].mean()
female_avg = amt_df[amt_df['Gender']=='F']['Purchase'].mean()

print("Average amount spend by Male customers: {:.2f}".format(male_avg))
print("Average amount spend by Female customers: {:.2f}".format(female_avg))
```

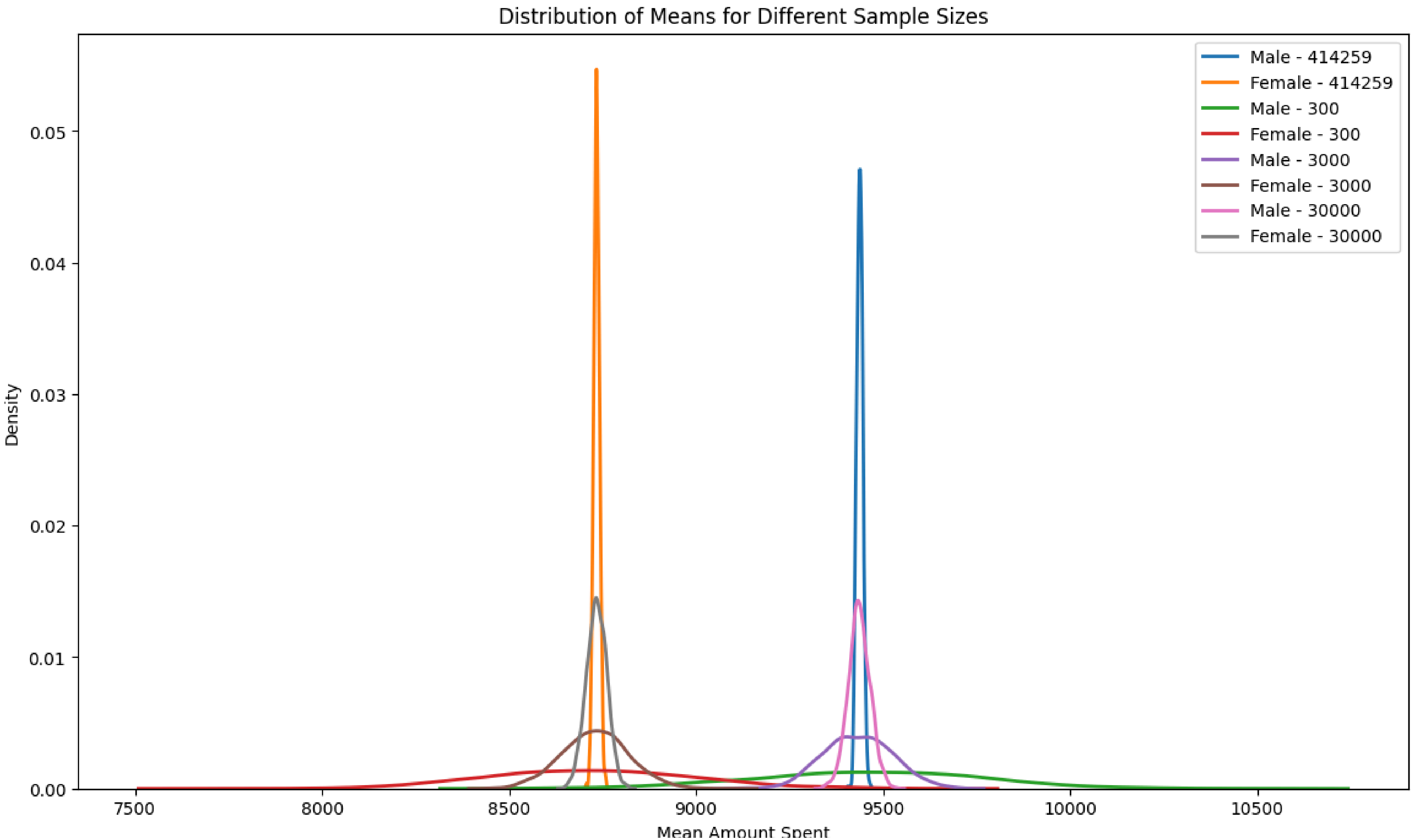
```
Average amount spend by Male customers: 925344.40
Average amount spend by Female customers: 712024.39
```

Observation: Observing the graph above and the average purchase of Male customers ie. 925344.40 and Female customers ie. 712024.39, we can conclude that- Males purchase more than Females

```
# Function to calculate bootstrap confidence interval
def bootstrap_ci(data, n_bootstrap, alpha=0.05):
    means = np.zeros(n_bootstrap)
    for i in range(n_bootstrap):
        sample = np.random.choice(data, size=len(data), replace=len(data) > 1)
        means[i] = np.mean(sample)
    lower_bound = np.percentile(means, 100 * alpha / 2)
    upper_bound = np.percentile(means, 100 * (1 - alpha / 2))
    return lower_bound, upper_bound, means

np.random.seed(42)

# Original dataset
amount_spent_male = df[df['Gender'] == 'M']['Purchase']
amount_spent_female = df[df['Gender'] == 'F']['Purchase']

# Bootstrap confidence interval for the entire dataset
lower_male, upper_male, _ = bootstrap_ci(amount_spent_male, 1000)
lower_female, upper_female, _ = bootstrap_ci(amount_spent_female, 1000)

# Print confidence intervals for the entire dataset
print(f"Male CI: ({lower_male:.2f}, {upper_male:.2f})")
print(f"Female CI: ({lower_female:.2f}, {upper_female:.2f})")

# Function to perform analysis for different sample sizes
def analyze_sample_size(sample_size):
    # Bootstrap confidence interval for specified sample size
    lower_male, upper_male, _ = bootstrap_ci(amount_spent_male.sample(sample_size, replace=True), 1000)
    lower_female, upper_female, _ = bootstrap_ci(amount_spent_female.sample(sample_size, replace=True), 1000)
    return lower_male, upper_male, lower_female, upper_female

# Sample sizes to analyze
sample_sizes = [len(amount_spent_male), 300, 3000, 30000]

# Results for different sample sizes
results = []
```

```
    for size in tqdm(sample_sizes, desc="Analyzing Sample Sizes"):
        lower_male, upper_male, lower_female, upper_female = analyze_sample_size(size)
        results.append({
            'Sample Size': size,
            'Male CI': (lower_male, upper_male),
            'Female CI': (lower_female, upper_female)
        })

    # Plot distributions of means for different sample sizes
    plt.figure(figsize=(14, 8))

    for result in results:
        means_male = np.zeros(1000)
        means_female = np.zeros(1000)

        for i in range(1000):
            sample_male = amount_spent_male.sample(result['Sample Size'], replace=True)
            sample_female = amount_spent_female.sample(result['Sample Size'], replace=True)

            means_male[i] = np.mean(sample_male)
            means_female[i] = np.mean(sample_female)

        sns.kdeplot(means_male, label=f'Male - {result["Sample Size"]}', linewidth=2)
        sns.kdeplot(means_female, label=f'Female - {result["Sample Size"]}', linewidth=2)

    plt.title('Distribution of Means for Different Sample Sizes')
    plt.xlabel('Mean Amount Spent')
    plt.ylabel('Density')
    plt.legend()
    plt.show()
```

```
Male CI: (9422.34, 9453.75)
Female CI: (8707.44, 8760.40)
Analyzing Sample Sizes: 100%|██████████| 4/4 [00:16<00:00,  4.19s/it]
```



i. Wider Confidence Interval: The confidence interval computed using the entire dataset may be wider for males. If one gender has a more diverse range of spending amounts, it can lead to a wider confidence interval.

ii. Effect of Sample Size on Interval Width: Generally, as the sample size increases, the width of the confidence interval decreases. Larger sample sizes provide more information about the population, resulting in a more precise estimate of the mean.

iii. Overlap of Confidence Intervals: Depending on the data, the confidence intervals for different sample sizes may overlap. However, as the sample size increases, the overlap is likely to decrease, indicating more confidence in the estimates.

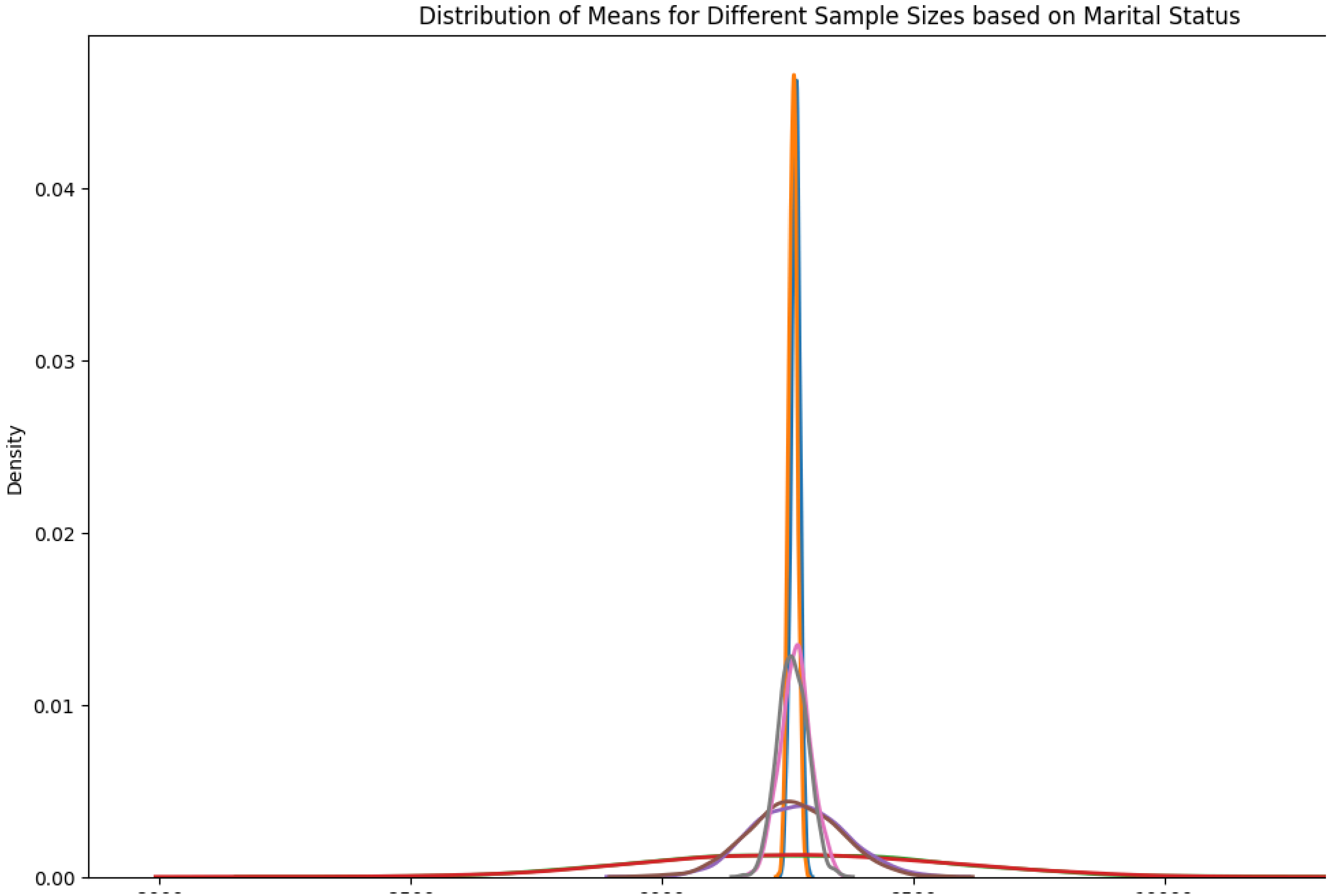
iv. Effect of Sample Size on Distribution Shape: As the sample size increases, the distribution of means becomes more normal (following the central limit theorem). With smaller sample sizes, the distribution may be more skewed or have heavier tails.

Marital_Status affecting the amount spent

```
# Function to perform analysis for different sample sizes based on Marital_Status
def analyze_marital_status(sample_size):
    # Compute bootstrap confidence interval for specified sample size
    lower_single, upper_single, _ = bootstrap_ci(amount_spent_single.sample(sample_size, replace=True), 1000)
    lower_married, upper_married, _ = bootstrap_ci(amount_spent_married.sample(sample_size, replace=True), 1000)
    return lower_single, upper_single, lower_married, upper_married

# Marital Status dataset
amount_spent_single = df[df['Marital_Status'] == 0]['Purchase']
amount_spent_married = df[df['Marital_Status'] == 1]['Purchase']

# Bootstrap confidence interval for the entire dataset based on Marital_Status
lower_single, upper_single, _ = bootstrap_ci(amount_spent_single, 1000)
lower_married, upper_married, _ = bootstrap_ci(amount_spent_married, 1000)

# Print confidence intervals for the entire dataset
print(f"Single CI: ({lower_single:.2f}, {upper_single:.2f})")
print(f"Married CI: ({lower_married:.2f}, {upper_married:.2f})")

# Sample sizes to analyze
sample_sizes = [len(amount_spent_single), 300, 3000, 30000]

# Results for different sample sizes based on Marital_Status
results_marital_status = []

for size in tqdm(sample_sizes, desc="Analyzing Sample Sizes - Marital Status"):
    lower_single, upper_single, lower_married, upper_married = analyze_marital_status(size)
    results_marital_status.append({
        'Sample Size': size,
        'Single CI': (lower_single, upper_single),
        'Married CI': (lower_married, upper_married)
    })

# Plot distributions of means for different sample sizes based on Marital_Status
plt.figure(figsize=(14, 8))

for result in results_marital_status:
```

```
    means_single = np.zeros(1000)
    means_married = np.zeros(1000)

    for i in range(1000):
        sample_single = amount_spent_single.sample(result['Sample Size'], replace=True)
        sample_married = amount_spent_married.sample(result['Sample Size'], replace=True)

        means_single[i] = np.mean(sample_single)
        means_married[i] = np.mean(sample_married)

    sns.kdeplot(means_single, label=f'Single - {result["Sample Size"]}', linewidth=2)
    sns.kdeplot(means_married, label=f'Married - {result["Sample Size"]}', linewidth=2)

plt.title('Distribution of Means for Different Sample Sizes based on Marital Status')
plt.xlabel('Mean Amount Spent')
plt.ylabel('Density')
plt.legend()
plt.show()
```

```
Single CI: (9249.61, 9283.28)
Married CI: (9242.41, 9281.14)
Analyzing Sample Sizes - Marital Status: 100%|██████████| 4/4 [00:23<00:00,  5.98s/it]
```



Observation: Married have more amount spent

```
# Function to perform analysis for different sample sizes based on Age
def analyze_age(sample_size):
    # Compute bootstrap confidence interval for specified sample size
    lower_age_0_17, upper_age_0_17, _ = bootstrap_ci(amount_spent_age_0_17.sample(sample_size, replace=True), 1000)
    lower_age_18_25, upper_age_18_25, _ = bootstrap_ci(amount_spent_age_18_25.sample(sample_size, replace=True), 1000)
    lower_age_26_35, upper_age_26_35, _ = bootstrap_ci(amount_spent_age_26_35.sample(sample_size, replace=True), 1000)
    lower_age_36_45, upper_age_36_45, _ = bootstrap_ci(amount_spent_age_36_45.sample(sample_size, replace=True), 1000)
    lower_age_46_50, upper_age_46_50, _ = bootstrap_ci(amount_spent_age_46_50.sample(sample_size, replace=True), 1000)
    lower_age_51_55, upper_age_51_55, _ = bootstrap_ci(amount_spent_age_51_55.sample(sample_size, replace=True), 1000)
    lower_age_55plus, upper_age_55plus, _ = bootstrap_ci(amount_spent_age_55plus.sample(sample_size, replace=True), 1000)

    return (lower_age_0_17, upper_age_0_17), (lower_age_18_25, upper_age_18_25), (lower_age_26_35, upper_age_26_35), \
           (lower_age_36_45, upper_age_36_45), (lower_age_46_50, upper_age_46_50), (lower_age_51_55, upper_age_51_55), \
           (lower_age_55plus, upper_age_55plus)

# Age-based datasets
amount_spent_age_0_17 = df[df['Age'] == '0-17']['Purchase']
amount_spent_age_18_25 = df[df['Age'] == '18-25']['Purchase']
amount_spent_age_26_35 = df[df['Age'] == '26-35']['Purchase']
amount_spent_age_36_45 = df[df['Age'] == '36-45']['Purchase']
amount_spent_age_46_50 = df[df['Age'] == '46-50']['Purchase']
amount_spent_age_51_55 = df[df['Age'] == '51-55']['Purchase']
amount_spent_age_55plus = df[df['Age'] == '55+']['Purchase']

# Compute bootstrap confidence interval for the entire dataset based on Age
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.