walmart-Dataset@Dhanureddy

April 11, 2024

1 Walmart dataset exploration

About walmart:

Walmart is an American multinational retail corporation that operates a chain of supercenters, discount departmental stores, and grocery stores from the United States. Walmart has more than 100 million customers worldwide.

Business problem:

The Management team at Walmart Inc. wants to analyze the customer purchase behavior (specifically, purchase amount) against the customer's gender and the various other factors to help the business make better decisions. They want to understand if the spending habits differ between male and female customers: Do women spend more on Black Friday than men? (Assume 50 million customers are male and 50 million are female).

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

[ ]:
```
df = pd.read_csv("walmart_data.csv")
```

[ ]:
```
df.head(5)
```

[ ]:
```
   User_ID Product_ID Gender   Age  Occupation City_Category  \
0  1000001  P00069042      F  0-17          10             A
1  1000001  P00248942      F  0-17          10             A
2  1000001  P00087842      F  0-17          10             A
3  1000001  P00085442      F  0-17          10             A
4  1000002  P00285442      M   55+          16             C

  Stay_In_Current_City_Years  Marital_Status  Product_Category  Purchase
0                          2               0                 3      8370
1                          2               0                 1     15200
2                          2               0                12      1422
3                          2               0                12      1057
4                         4+               0                 8      7969
```

[ ]:
```
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
```

Inference:

1. there are in total ten columns, with no null values.
2. There are in total five each string and integer datatype columns.

Finding unique values of each column in the dataframe

[ ]:
```
for columns in df.columns:
  unique_count = df[columns].nunique()
  print(columns, "-", unique_count)
```

```
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
```

Checking if there are any possible null values in the dataframe

[ ]:
```
df.isna().isna().sum()
```

[ ]:
```
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
```

Shape of the dataframe

[ ]:
```
df.shape
```

[ ]:
```
(550068, 10)
```

Summary of the dataframe describing statistical information of categorical variables

[ ]:
```
summary = df.describe()
summary
```

[ ]:
```
            User_ID     Occupation  Marital_Status  Product_Category  \
count  5.500680e+05  550068.000000   550068.000000     550068.000000
mean   1.003029e+06       8.076707        0.409653          5.404270
std    1.727592e+03       6.522660        0.491770          3.936211
min    1.000001e+06       0.000000        0.000000          1.000000
25%    1.001516e+06       2.000000        0.000000          1.000000
50%    1.003077e+06       7.000000        0.000000          5.000000
75%    1.004478e+06      14.000000        1.000000          8.000000
max    1.006040e+06      20.000000        1.000000         20.000000

            Purchase
count  550068.000000
mean     9263.968713
std      5023.065394
min        12.000000
25%      5823.000000
50%      8047.000000
75%     12054.000000
max     23961.000000
```

Finiding if there are any outliers through use of boxplots

[ ]:
```
plt.figure(figsize =(18,8))
plt.subplots_adjust(left=0.4, right=0.9, top=0.9, bottom=0.1, wspace=0.4,␣
 ↪hspace=0.4)

plt.subplot(2,2,1)
sns.boxplot(data = df.Age)
```

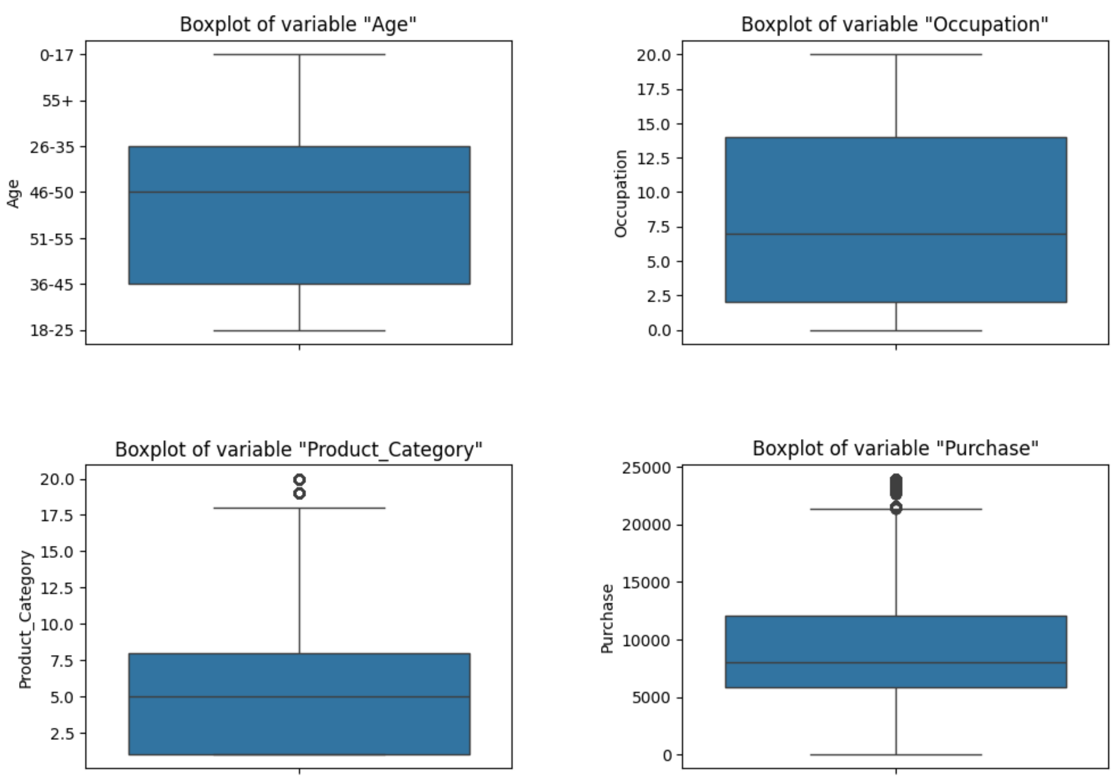
```
plt.title('Boxplot of variable "Age"')

plt.subplot(2,2,2)
sns.boxplot(data = df.Occupation)
plt.title('Boxplot of variable "Occupation"')

plt.subplot(2,2,3)
sns.boxplot(data = df.Product_Category)
plt.title('Boxplot of variable "Product_Category"')

plt.subplot(2,2,4)
sns.boxplot(data = df.Purchase)
plt.title('Boxplot of variable "Purchase"')
```

[ ]: Text(0.5, 1.0, 'Boxplot of variable "Purchase"')



Finding total number of outliers in each categorical varible by setting boundaries

[ ]:
```
Q1 = summary.loc["25%"]
Q3 = summary.loc["75%"]
IQR = Q3 - Q1
print(IQR)
```

```
User_ID             2962.0
Occupation            12.0
Marital_Status         1.0
Product_Category       7.0
Purchase            6231.0
dtype: float64
```

[ ]:
```
Lower_bound = Q1 - 1.5*IQR
Upper_bound = Q3 + 1.5*IQR

bounds_df = pd.DataFrame({"LowerBound" :Lower_bound, "UpperBound" :Upper_bound})
print(bounds_df)
```

```
                  LowerBound  UpperBound
User_ID             997073.0   1008921.0
Occupation             -16.0        32.0
Marital_Status          -1.5         2.5
Product_Category        -9.5        18.5
Purchase             -3523.5     21400.5
```

[ ]:
```
outliers_lower = (df < Lower_bound).sum()
outliers_upper = (df > Upper_bound).sum()
total_outliers = outliers_lower  + outliers_upper

ouliers_count_df = pd.DataFrame({"LowerBound_outliers" :outliers_lower,␣
 ↪"UpperBound_outliers" :outliers_upper, "Total" : total_outliers})
print(ouliers_count_df)
```

```
<ipython-input-12-e2fa83975e56>:1: FutureWarning: Automatic reindexing on
DataFrame vs Series comparisons is deprecated and will raise ValueError in a
future version. Do `left, right = left.align(right, axis=1, copy=False)` before
e.g. `left == right`
  outliers_lower = (df < Lower_bound).sum()
<ipython-input-12-e2fa83975e56>:2: FutureWarning: Automatic reindexing on
DataFrame vs Series comparisons is deprecated and will raise ValueError in a
future version. Do `left, right = left.align(right, axis=1, copy=False)` before
e.g. `left == right`
  outliers_upper = (df > Upper_bound).sum()
                            LowerBound_outliers  UpperBound_outliers  Total
Age                                           0                    0      0
City_Category                                 0                    0      0
Gender                                        0                    0      0
Marital_Status                                0                    0      0
Occupation                                    0                    0      0
Product_Category                              0                 4153   4153
Product_ID                                    0                    0      0
Purchase                                      0                 2677   2677
```

```
Stay_In_Current_City_Years                        0                    0       0
User_ID                                           0                    0       0
```

Finding Difference between mean and median

```
[ ]: difference_between_Mean_and_Median = (summary.loc["mean"] - summary.loc["50%"])
     difference_between_Mean_and_Median
```

```
[ ]: User_ID              -48.157599
     Occupation             1.076707
     Marital_Status         0.409653
     Product_Category       0.404270
     Purchase            1216.968713
     dtype: float64
```

«——————————————————————» «——————————————————————»

«——————————————————————» «——————————————————————»

1. Count of Products categories grouped under different age bins

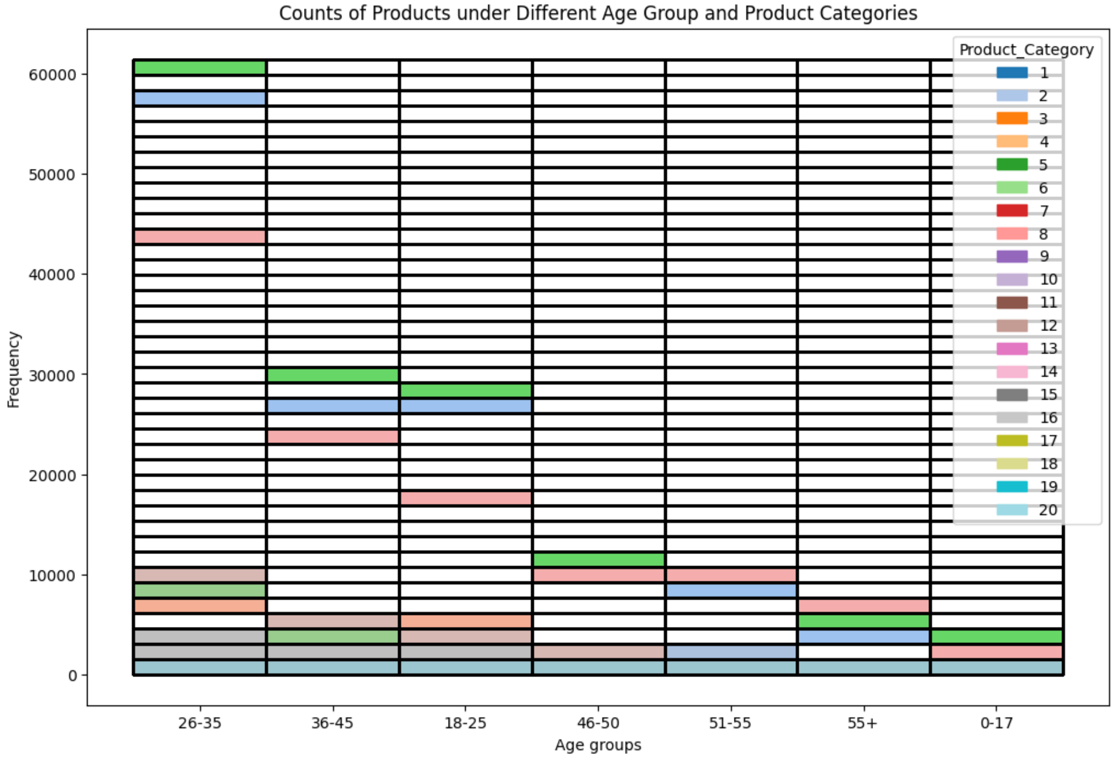
```
[ ]: count_of_products_under_agegroup = df.groupby(["Age", "Product_Category"]).
      ↪size()
     count_of_products_under_agegroup = count_of_products_under_agegroup.
      ↪sort_values(ascending = False).reset_index(name = "Count")
     count_of_products_under_agegroup
```

```
[ ]:        Age  Product_Category  Count
     0    26-35                 5  61473
     1    26-35                 1  58249
     2    26-35                 8  44256
     3    36-45                 5  29377
     4    18-25                 5  28522
     ..     …                 …    …
     135  51-55                 9     29
     136   0-17                18     27
     137   0-17                 9     16
     138    55+                 9      8
     139   0-17                17      6

     [140 rows x 3 columns]
```

```
[ ]: plt.figure(figsize = (12, 8))
     sns.histplot(data = count_of_products_under_agegroup, x = "Age", y = "Count",␣
      ↪hue = "Product_Category", bins = 40, palette = 'tab20', edgecolor='black')
     plt.title("Counts of Products under Different Age Group and Product Categories")
     plt.xlabel('Age groups')
```

```
plt.ylabel('Frequency')
plt.show()
```



Inference: In almost every age bin, category 5 tops the place in terms of purchase count. Along with category 5, we have category 1 and 8 with significant contributions in almost every age bin category.

Recommendation: For every age bin, it is highly suggested to target on product categories (1,5,8) combinedly to increase the demand further and improve the business performance.

«——————————————————————————————» «——————————————————————————————»

2. Relationship between age, marital status and amount spent

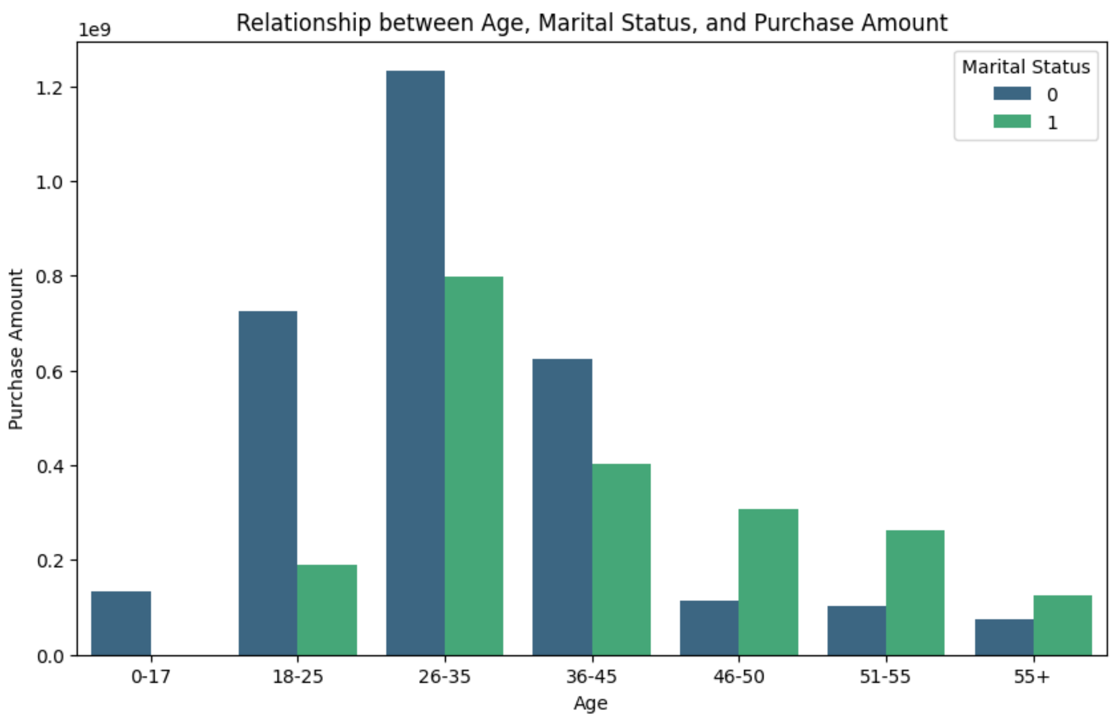
[ ]:
```
Total_Purchase_amount = df.groupby(["Age", "Marital_Status"])["Purchase"].sum()
Total_Purchase_amount = Total_Purchase_amount.sort_index(ascending = True).
↪reset_index()
Total_Purchase_amount
```

[ ]:

	Age	Marital_Status	Purchase
0	0-17	0	134913183
1	18-25	0	723920602
2	18-25	1	189928073

```
3    26-35                0  1233330102
4    26-35                1   798440476
5    36-45                0   624110760
6    36-45                1   402459124
7    46-50                0   113658360
8    46-50                1   307185043
9    51-55                0   103792394
10   51-55                1   263307250
11     55+                0    75202046
12     55+                1   125565329
```

[ ]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=Total_Purchase_amount, x='Age', y='Purchase',␣
 ↪hue='Marital_Status', palette='viridis')
plt.title('Relationship between Age, Marital Status, and Purchase Amount')
plt.xlabel('Age')
plt.ylabel('Purchase Amount')
plt.legend(title='Marital Status')
plt.show()
```



Inference: From this above graph we can infer that Non married people dominate the purchases below the age bin 45, and married couple does more purchases above the age bin 45.

Also from overall point of view, people between 18-45 age group does more purchases than rest of others. In particular people in the age group of 26-35 are actively purchasing more than any other age groups amounting to 39.87% of total purchases.

Recommednation: Focusing on 26-35 age groups yields better results for maintaining purchases demand and also majority of purchases are from people below 45 specifically belonging to non-married; Thus, targetting with some specific type of products or employing certain preferences which cater to them will increase the overall business turnover.

«——————————————————————» «——————————————————————»

3. Preferred product categories grouped based on gender

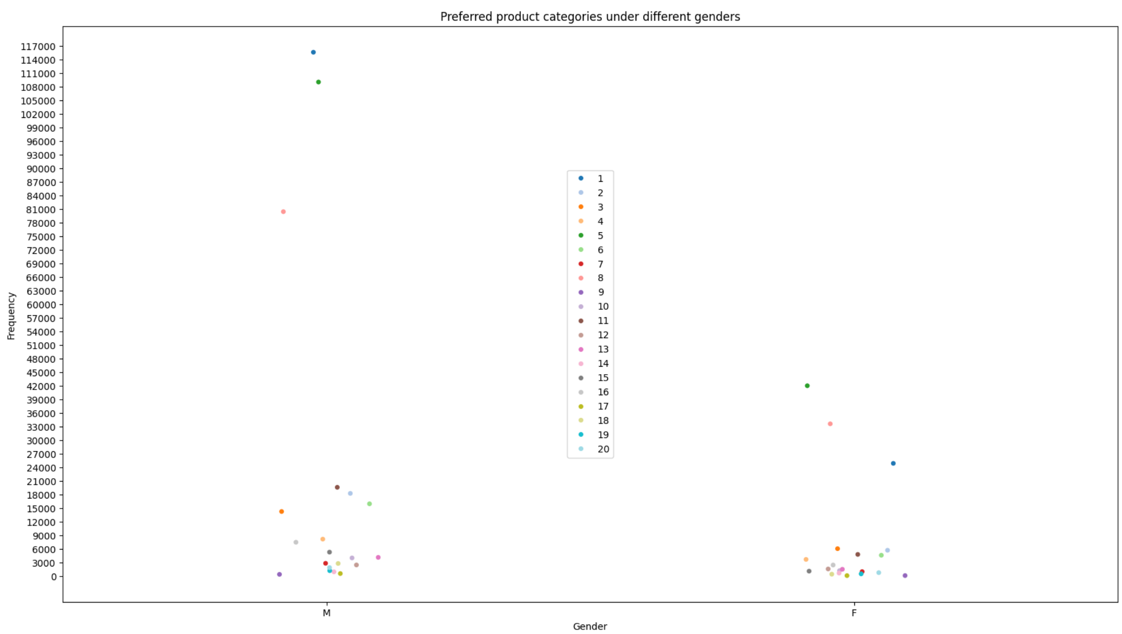
[ ]:
```
count_of_products_under_gender = df.groupby(["Gender", "Product_Category"]).
 ↪size()
count_of_products_under_gender = count_of_products_under_gender.
 ↪reset_index(name = 'Count').sort_values(by = ["Gender", "Product_Category"],␣
 ↪ascending = [False, True])
count_of_products_under_gender
```

[ ]:

	Gender	Product_Category	Count
20	M	1	115547
21	M	2	18206
22	M	3	14207
23	M	4	8114
24	M	5	108972
25	M	6	15907
26	M	7	2778
27	M	8	80367
28	M	9	340
29	M	10	3963
30	M	11	19548
31	M	12	2415
32	M	13	4087
33	M	14	900
34	M	15	5244
35	M	16	7426
36	M	17	516
37	M	18	2743
38	M	19	1152
39	M	20	1827
0	F	1	24831
1	F	2	5658
2	F	3	6006
3	F	4	3639
4	F	5	41961
5	F	6	4559
6	F	7	943
7	F	8	33558
8	F	9	70

9	F	10	1162
10	F	11	4739
11	F	12	1532
12	F	13	1462
13	F	14	623
14	F	15	1046
15	F	16	2402
16	F	17	62
17	F	18	382
18	F	19	451
19	F	20	723

[ ]:
```
plt.figure(figsize = (20, 11))
sns.stripplot(data = count_of_products_under_gender, x = "Gender", y =
↪"Count",hue = "Product_Category", palette = 'tab20', edgecolor='black')
plt.title("Preferred product categories under different genders")
plt.xlabel('Gender')
plt.ylabel('Frequency')
plt.yticks(range(0,120000,3000), fontsize = 10)
plt.legend(loc = 'center')
plt.show()
```



Inference: Based on the above graph we can infer that males(75.31% of Total products) domiante the purchases than females. In particular there are three specific categories(1,5,8) stood apart in both males(73.59% of Total Male products) and females(73.89% of Total Female products) purchasing history. Especially for males, both categories 1 and 5 crossed the mark of 100000 in total, whereas in females the highest sales stood below 42000 mark.

Rest of product category purchases in both males and females were below the mark of 21000, and majority of them were below 9000.

Recommendation: To increase overall sales, the company should focus more on males specifically from (1,5,8) categories. If there are proper strategies being installed in place to increase the demand of sales from males, then focus should also shift to females for the same categories.

It was best to decrease unwarranted expenditure on cluster of product categories below the sales of 9000.

«——————————————————————————————» «——————————————————————————————»

4. Relationship between Purchase amount, Gender, City_Category and Product Category

[ ]:
```
Amount_under_city_and_product = df.groupby(["Gender", "City_Category",
 ↪"Product_Category"])["Purchase"].sum()
Amount_under_city_and_product = Amount_under_city_and_product.reset_index().
 ↪sort_values(by=["Gender", "City_Category", "Product_Category"],
 ↪ascending=[False, True, True])
Amount_under_city_and_product
```

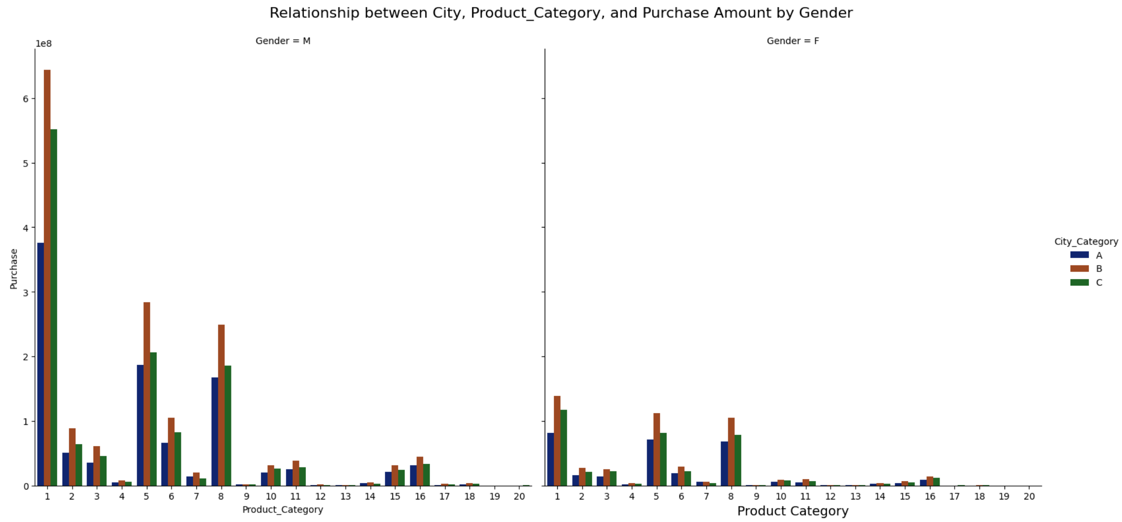
[ ]:
```
   Gender City_Category  Product_Category   Purchase
60      M             A                 1  376181738
61      M             A                 2   50930747
62      M             A                 3   35592296
63      M             A                 4    4893956
64      M             A                 5  186780661
..    …             …               …        …
55      F             C                16   11749084
56      F             C                17     277799
57      F             C                18     361278
58      F             C                19       8449
59      F             C                20     137224

[120 rows x 4 columns]
```

[ ]:
```
g = sns.catplot(data=Amount_under_city_and_product, kind="bar",
 ↪x="Product_Category", y="Purchase",
                hue="City_Category", col="Gender", palette="dark", height=8,
 ↪aspect=1)

plt.subplots_adjust(top=0.9)
plt.suptitle("Relationship between City, Product_Category, and Purchase Amount
 ↪by Gender", fontsize=16)
plt.xlabel("Product Category", fontsize=14)
plt.ylabel("Purchase Amount", fontsize=14)
```

[ ]:
```
Text(875.1887119622879, 0.5, 'Purchase Amount')
```



Inference: (Male) From the above graph we can infer that category B purchases(41.48% of total male purchase amount) stood top for almost every product category. And if we consider product categories in specific then (1,5,8) combined amounts to 57.67% of total male purchases.

(Female) from the above graph we can infer that again category B purchases(41.61% of Total female purchases) stood top fro almost every product category. And if we consider product categories in specific then (1,5,8) combined amounts to 71.99% of total female purchases.

Recommendation: Concentrating on **City_category B** for both male and femlaes and in specific, categories (1,5,8) combinedly will yield high business performance.

«——————————————————» «——————————————————»

5. Relationship between Purchase amount, Gender and Occupation

[ ]:
```
Amount_under_gender_occupation_product = df.groupby(["Gender",
 ↪"Occupation"])["Purchase"].sum()
Amount_under_gender_occupation_product = Amount_under_gender_occupation_product.
 ↪reset_index().sort_values(by=["Gender", "Occupation"], ascending=[False,
 ↪True])
Amount_under_gender_occupation_product
```

[ ]:

	Gender	Occupation	Purchase
21	M	0	475523125
22	M	1	271807418
23	M	2	165459113
24	M	3	90294529
25	M	4	513980163
26	M	5	94054709
27	M	6	114336992
28	M	7	466193977

```
29      M           8   11357904
30      M           9    4133559
31      M          10   83040876
32      M          11   93115418
33      M          12  273687444
34      M          13   59092473
35      M          14  201444632
36      M          15   96506412
37      M          16  201526828
38      M          17  355785294
39      M          18   58404301
40      M          19   56693467
41      M          20  223141466
0       F           0  159883833
1       F           1  152806726
2       F           2   72569470
3       F           3   71707639
4       F           4  152264321
5       F           5   19595050
6       F           6   74079792
7       F           7   91177610
8       F           8    3379484
9       F           9   50206487
10      F          10   32803589
11      F          11   13636200
12      F          12   31762002
13      F          13   12827008
14      F          14   58010060
15      F          15   22453799
16      F          16   36820127
17      F          17   37496159
18      F          18    2317160
19      F          19   17007150
20      F          20   73428976
```

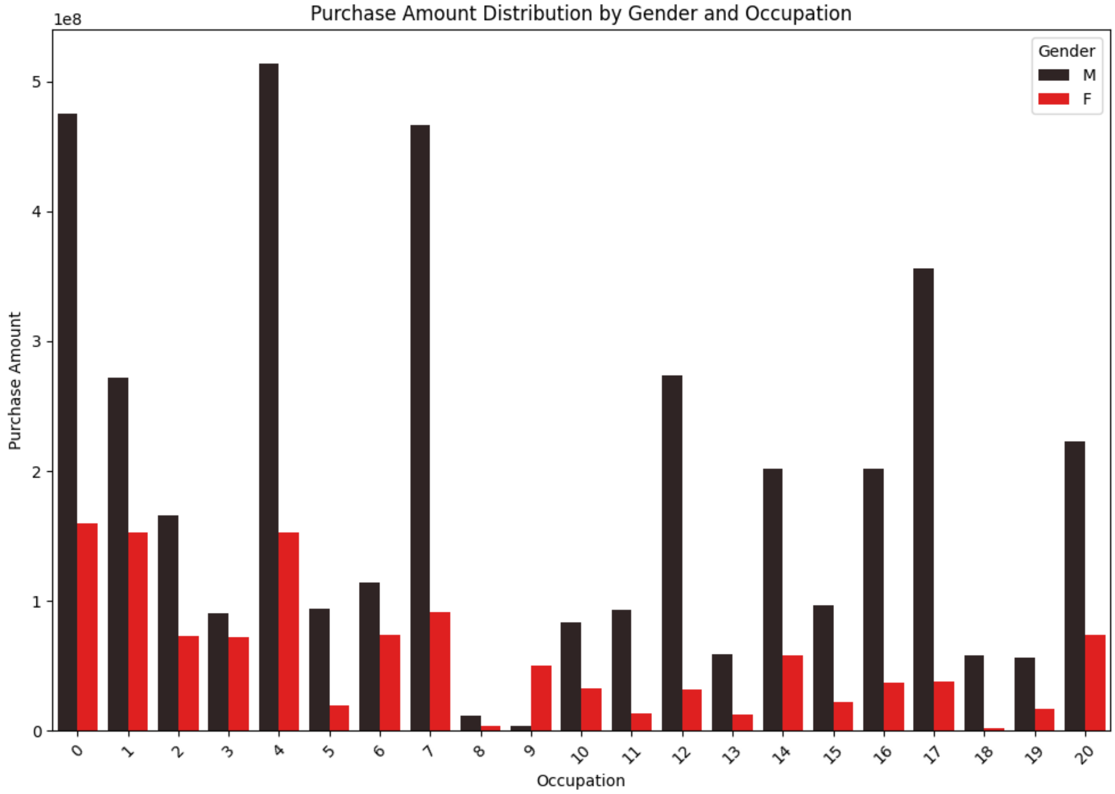
[ ]:

```
plt.figure(figsize=(12, 8))
sns.barplot(data=Amount_under_gender_occupation_product, x="Occupation",␣
 ↪y="Purchase", hue="Gender", color = "red")
plt.title("Purchase Amount Distribution by Gender and Occupation")
plt.xlabel("Occupation")
plt.ylabel("Purchase Amount")
plt.legend(title="Gender")
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-51-c93cc4ca4da3>:2: FutureWarning:
```

```
Setting a gradient palette using color= is deprecated and will be removed in
v0.14.0. Set `palette='dark:red'` for the same effect.

  sns.barplot(data=Amount_under_gender_occupation_product, x="Occupation",
y="Purchase", hue="Gender", color = "red")
```



Inference: (Male) From the above graph we can infer that occupations(0,1,4,7,12,14,16,17,20) combined have a Total Purchases of 76.30%

(Female) From the above graph we can infer that occupations(0,1,4,7,9,14,20) combined have a Total Purchases of 62.19%

Recommednation: It is observed that from both genders; Occupations like (0,1,4,7,14,20) have equally high percenatege contribution to purchases. Thus targetting these occupations in both genders yields good business returns.

«——————————————————————————————» «——————————————————————————————»

6. Affect of Gender affecting the purchases made

[ ]:
```
male_data = df[df["Gender"] == 'M']['Purchase']
female_data = df[df["Gender"] == 'F']['Purchase']
```

```python
def bootstrap_CI(data, bootstrap_samples, alpha):
  boot_means = []
  for _ in range(bootstrap_samples):
    sample = np.random.choice(data, size = len(data), replace = True)
    boot_means.append(np.mean(sample))

  lower_bound = np.percentile(boot_means, 100 * alpha/2)
  upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
  return lower_bound, upper_bound

bootstrap_samples= 10000
alpha = 0.05
male_CI = bootstrap_CI(male_data, bootstrap_samples, alpha)
female_CI = bootstrap_CI(female_data, bootstrap_samples, alpha)

print("95% Confidence Interval for Males:", male_CI)
print("95% Confidence Interval for Females:", female_CI)
```

```
95% Confidence Interval for Males: (9421.968385116557, 9453.346517335773)
95% Confidence Interval for Females: (8709.088316127798, 8759.819427836152)
```

Inference: (Random samples drawn 10000 from entire data set considered as sample)

1. t can be concluded from the above observation of confidence intervals that there was no wider gap between intervals and infact the difference are very low in both females and males regarding their purcahses. Here, we can conclude that the mean calculated from the random 10000 samples from the entire dataset truly represents the population characteristics of the data.
2. As the sample size was entire dataset, the width of the intervals is quite low, but if the smaple size was been lower, we can observe that the width increases gradually to an extent.
3. There has been no evidence of overlapping of male and female samples of mean purchases; Thus, we can conclude that there was significant difference of purchasing behaviour between males and females.
4. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.
5. With larger samples the variability also decreases, this was because larger samples provide more information about the population, thus gap between the intervals gets reduced.

For smaller smaple sizes

[ ]:
```python
def bootstrap_CI(data, bootstrap_samples, sample_size, alpha):
    boot_means = []
    for _ in range(bootstrap_samples):
        sample = np.random.choice(data, size=sample_size, replace=True)
        boot_means.append(np.mean(sample))
```

```python
    lower_bound = np.percentile(boot_means, 100 * alpha / 2)
    upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
    return lower_bound, upper_bound


sample_sizes = [300, 3000, 30000]
bootstrap_samples= 10000
alpha = 0.05

cis_data = []

# Calculate confidence intervals for each gender and sample size
for gender, gender_data in {'Male': male_data, 'Female': female_data}.items():
    print(f"Confidence Intervals for Gender: {gender}")
    for sample_size in sample_sizes:
        ci = bootstrap_CI(gender_data, bootstrap_samples, sample_size, alpha)
        print(f"Sample Size: {sample_size}, CI: {ci}")
        cis_data.append({
            'Sample Size': sample_size,
            'Gender': gender,
            'Lower Bound': ci[0],
            'Upper Bound': ci[1]
        })

cis_df = pd.DataFrame(cis_data)
```

```
Confidence Intervals for Gender: Male
Sample Size: 300, CI: (8870.154583333333, 10029.28025)
Sample Size: 3000, CI: (9256.450041666667, 9620.09515)
Sample Size: 30000, CI: (9381.019135833332, 9494.81068)
Confidence Intervals for Gender: Female
Sample Size: 300, CI: (8200.486333333334, 9285.423416666667)
Sample Size: 3000, CI: (8563.1249, 8900.606533333334)
Sample Size: 30000, CI: (8680.1585175, 8788.4276075)
```

Inference: (Randomly 10000 samples drawn for each sample size of 300,3000, 30000 from the dataset)

1. It can be concluded from the above observation of confidence intervals that the gap between intervals gradually got reduced as sample size increased from 300 to 30000. Infact the difference was very low in both females and males regarding their purcahses in highest sample size. With lower difference, we can conclude that the mean calculated from the random 30000 sample size truly represents the population characteristics of the data.

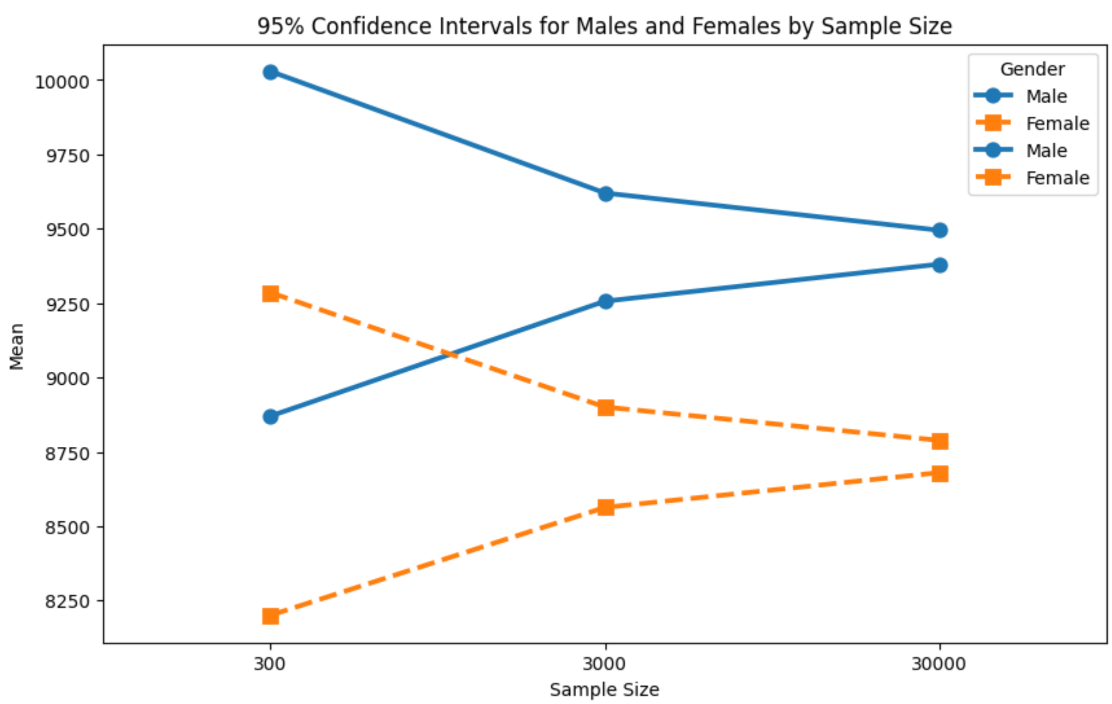
2. There has been evidence of overlapping of male and female samples of mean purchases when the sample size is 300, however when the size increased the overlapping diminished; Thus, we can conclude that there was significant difference of purchasing behaviour between males and females with a reliable sample size.

3. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.

4. With larger samples the variability also decreases, this was because larger samples provide more information about the population, thus gap between the intervals gets reduced.

Visual represenattion for sample sizes

[ ]:
```
plt.figure(figsize=(10, 6))
sns.pointplot(data=cis_df, x='Sample Size', y='Lower Bound', hue='Gender',␣
↪markers=['o', 's'], linestyles=['-', '--'])
sns.pointplot(data=cis_df, x='Sample Size', y='Upper Bound', hue='Gender',␣
↪markers=['o', 's'], linestyles=['-', '--'])
plt.title('95% Confidence Intervals for Males and Females by Sample Size')
plt.xlabel('Sample Size')
plt.ylabel('Mean')
plt.legend(title='Gender')
plt.show()
```



«——————————————————————————» «——————————————————————————»

7. Affect of marital status on Purcahses made

[ ]:
```
marital_data = df[df["Marital_Status"] == 1]['Purchase']
Non_marital_data = df[df["Marital_Status"] == 0]['Purchase']

def bootstrap_CI(data, bootstrap_samples, alpha):
  boot_means = []
  for _ in range(bootstrap_samples):
    sample = np.random.choice(data, size = len(data), replace = True)
    boot_means.append(np.mean(sample))

  lower_bound = np.percentile(boot_means, 100 * alpha/2)
  upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
  return lower_bound, upper_bound

bootstrap_samples= 10000
alpha = 0.05
married_CI = bootstrap_CI(marital_data, bootstrap_samples, alpha)
Non_married_CI = bootstrap_CI(Non_marital_data, bootstrap_samples, alpha)

print("95% Confidence Interval for Married:", married_CI)
print("95% Confidence Interval for Non-married:", Non_married_CI)
```

```
95% Confidence Interval for Married: (9240.931870931094, 9282.811374629999)
95% Confidence Interval for Non-married: (9248.335262494189, 9283.049257308357)
```

Inference: (Random samples drawn 10000 from entire data set considered as sample)

1. It can be concluded from the above observation of confidence intervals that there was no wider gap between intervals and infact the difference are very very low in both married and non-married regarding their purcahses. Here, we can conclude that the mean calculated from the random 10000 samples from the entire dataset truly represents the population characteristics of the data.

2. As the sample size was entire dataset, the width of the intervals is quite very low, but if the smaple size was been lower, we can observe that the width increases gradually to an extent.

3. There has been evidence of overlapping of married and non-married samples of mean purchases; Thus, we can conclude that there was no significant difference of purchasing behaviour between the two groups.

4. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.

5. With larger samples the variability also decreases, this was because larger samples provide more information about the population, thus gap between the intervals gets reduced.

For smaller sample sizes

[ ]:
```
def bootstrap_CI(data, bootstrap_samples, sample_size, alpha):
    boot_means = []
```

```
    for _ in range(bootstrap_samples):
        sample = np.random.choice(data, size=sample_size, replace=True)
        boot_means.append(np.mean(sample))

    lower_bound = np.percentile(boot_means, 100 * alpha / 2)
    upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
    return lower_bound, upper_bound


sample_sizes = [300, 3000, 30000]
bootstrap_samples= 10000
alpha = 0.05

cis_data = []

# Calculate confidence intervals for each marital status and sample size
for status, status_data in {'Married': marital_data, 'Non-married':␣
↪Non_marital_data}.items():
    print(f"Confidence Intervals for Marital Status: {status}")
    for sample_size in sample_sizes:
        ci = bootstrap_CI(status_data, bootstrap_samples, sample_size, alpha)
        print(f"Sample Size: {sample_size}, CI: {ci}")
        cis_data.append({
            'Sample Size': sample_size,
            'Marital Status': status,
            'Lower Bound': ci[0],
            'Upper Bound': ci[1]
        })

cis_df = pd.DataFrame(cis_data)
```

```
Confidence Intervals for Marital Status: Married
Sample Size: 300, CI: (8701.145416666668, 9836.900416666667)
Sample Size: 3000, CI: (9085.670241666667, 9444.045533333334)
Sample Size: 30000, CI: (9204.978958333333, 9318.983489166667)
Confidence Intervals for Marital Status: Non-married
Sample Size: 300, CI: (8692.335333333334, 9837.994083333333)
Sample Size: 3000, CI: (9084.688925, 9444.676258333333)
Sample Size: 30000, CI: (9209.768735833333, 9323.764079999999)
```

Inference: (Randomly 10000 samples drawn for each sample size of 300,3000, 30000 from the dataset)

1. It can be concluded from the above observation of confidence intervals that there was no significant decrement of gap between intervals as sample size increased from 300 to 30000. Infact the difference were stable between both married and non-married for each sample size. But as sample size got to 30000, the interval was been at 9000 range.

2. There has been evidence of overlapping of married and non-married samples of mean purchases

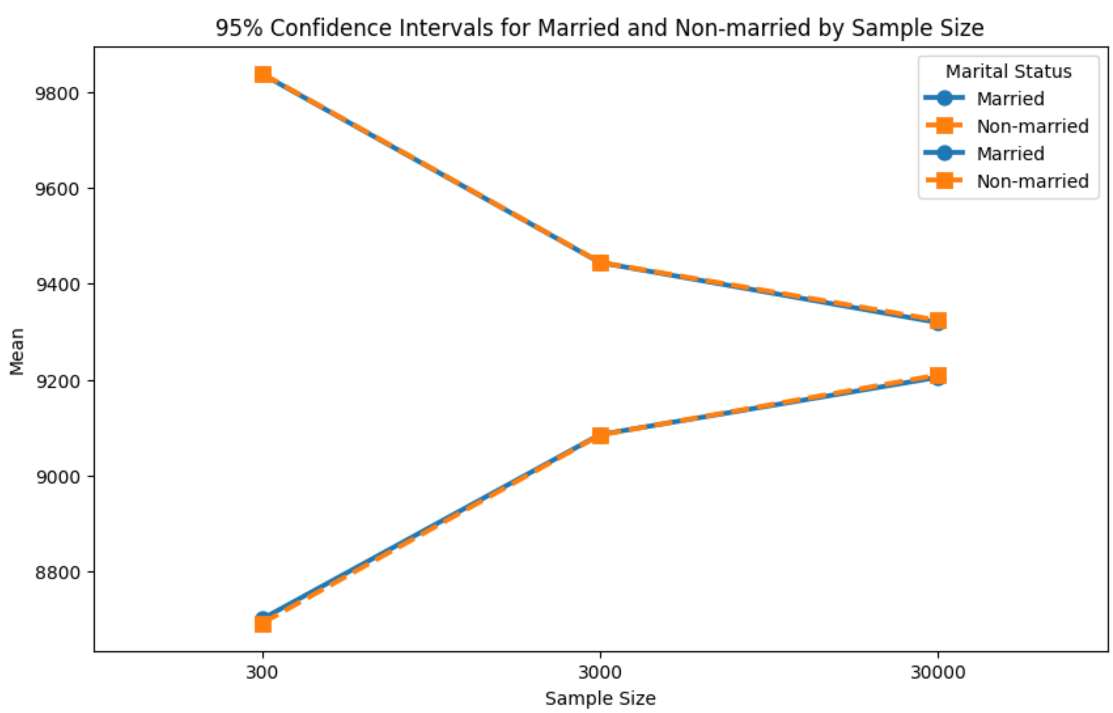
for all sample sizes of 300, 3000 and 30000. Thus, we can conclude that there was no significant difference of purchasing behaviour between the two groups.

3. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.

4. However here even with low sample size we observed very less difference between intervals, does it indicates the strong similarity of purchasing behaviour between married and non married.

Visual represenattion for sample sizes

[ ]:
```
plt.figure(figsize=(10, 6))
sns.pointplot(data=cis_df, x='Sample Size', y='Lower Bound', hue='Marital␣
↪Status', markers=['o', 's'], linestyles=['-', '--'])
sns.pointplot(data=cis_df, x='Sample Size', y='Upper Bound', hue='Marital␣
↪Status', markers=['o', 's'], linestyles=['-', '--'])
plt.title('95% Confidence Intervals for Married and Non-married by Sample Size')
plt.xlabel('Sample Size')
plt.ylabel('Mean')
plt.legend(title='Marital Status')
plt.show()
```



«——» «——

———————————————————————»

8. Affect of Age on Purchases made

```
[ ]: under_18 = df[df["Age"] == '0-17']["Purchase"]
over_18_to_25 = df[df["Age"] == '18-25']["Purchase"]
over_25_to_35 = df[df["Age"] == '26-35']["Purchase"]
over_35_to_45 = df[df["Age"] == '36-45']["Purchase"]
over_45_to_50 = df[df["Age"] == '46-50']["Purchase"]
over_50_to_55 = df[df["Age"] == '51-55']["Purchase"]
over_55 = df[df["Age"] == '55+']["Purchase"]

def bootstrap_CI(data, bootstrap_samples, alpha):
  boot_means = []
  for _ in range(bootstrap_samples):
    sample = np.random.choice(data, size = len(data), replace = True)
    boot_means.append(np.mean(sample))

  lower_bound = np.percentile(boot_means, 100 * alpha/2)
  upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
  return lower_bound, upper_bound

bootstrap_samples= 10000
alpha = 0.05
under_18_CI = bootstrap_CI(under_18, bootstrap_samples, alpha)
over_18_to_25_CI = bootstrap_CI(over_18_to_25, bootstrap_samples, alpha)
over_25_to_35_CI = bootstrap_CI(over_25_to_35, bootstrap_samples, alpha)
over_35_to_45_CI = bootstrap_CI(over_35_to_45, bootstrap_samples, alpha)
over_45_to_50_CI = bootstrap_CI(over_45_to_50, bootstrap_samples, alpha)
over_50_to_55_CI = bootstrap_CI(over_50_to_55, bootstrap_samples, alpha)
over_55_CI =  bootstrap_CI(over_55, bootstrap_samples, alpha)

print("95% Confidence Interval for under_18:", under_18_CI)
print("95% Confidence Interval for over_18_to_25:", over_18_to_25_CI)
print("95% Confidence Interval for over_25_to_35:", over_25_to_35_CI)
print("95% Confidence Interval for over_35_to_45:", over_35_to_45_CI)
print("95% Confidence Interval for over_45_to_50:", over_45_to_50_CI)
print("95% Confidence Interval for over_50_to_55:", over_50_to_55_CI)
print("95% Confidence Interval for over_55:", over_55_CI)
```

```
95% Confidence Interval for under_18: (8853.430565488015, 9017.055244338499)
95% Confidence Interval for over_18_to_25: (9138.419587096127, 9200.6849957355)
95% Confidence Interval for over_25_to_35: (9232.37976382937, 9273.742097209763)
95% Confidence Interval for over_35_to_45: (9301.986292074575,
9361.166050375865)
95% Confidence Interval for over_45_to_50: (9161.848008249273,
9253.808000371982)
95% Confidence Interval for over_50_to_55: (9483.476995402716,
```

```
9585.867744993637)
95% Confidence Interval for over_55: (9269.1715843564, 9403.43980189732)
```

Inference: (Random samples drawn 10000 from entire data set considered as sample)

1. It can be concluded from the above observation of confidence intervals that there was no wider gap between intervals and infact the difference are very low in every age bin category regarding their purcahses. Here, we can conclude that the mean calculated from the random 10000 samples from the entire dataset truly represents the population characteristics of the data.

2. As the sample size is entire dataset, the width of the intervals are quite low, but if the smaple size was been lower, we can observe that the width increases gradually to an extent.

3. There has been evidence of overlapping of age bins between (over_18_to_25) with (over_45_to_50) and (over_25_to_35) with both(over_45_to_50) and (over_55) samples of mean purchases; Thus, we can conclude that there was no significant difference of purchasing behaviour between these bins.

4. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.

5. With larger samples the variability also decreases, this was because larger samples provide more information about the population, thus gap between the intervals gets reduced.

For smaller sample sizes

[ ]:
```python
def bootstrap_CI(data, bootstrap_samples, sample_size, alpha):
    boot_means = []
    for _ in range(bootstrap_samples):
        sample = np.random.choice(data, size=sample_size, replace=True)
        boot_means.append(np.mean(sample))

    lower_bound = np.percentile(boot_means, 100 * alpha / 2)
    upper_bound = np.percentile(boot_means, 100 * (1 - alpha / 2))
    return lower_bound, upper_bound

sample_sizes = [300, 3000, 30000]
bootstrap_samples= 10000
alpha = 0.05



age_groups_data = {
    'under_18': under_18,
    'over_18_to_25': over_18_to_25,
    'over_25_to_35': over_25_to_35,
    'over_35_to_45': over_35_to_45,
    'over_45_to_50': over_45_to_50,
    'over_50_to_55': over_50_to_55,
```

```
    'over_55': over_55
}


cis_df = pd.DataFrame(columns=['Sample Size', 'Age Group', 'Lower Bound',␣
 ↪'Upper Bound'])

for age_group, age_group_data in age_groups_data.items():
    print(f"Confidence Intervals for Age Group: {age_group}")
    cis_age_group = []
    for sample_size in sample_sizes:
        ci = bootstrap_CI(age_group_data, bootstrap_samples, sample_size, alpha)
        print(f"Sample Size: {sample_size}, CI: {ci}")
        cis_df = pd.concat([
            cis_df,
            pd.DataFrame({
                'Sample Size': [sample_size],
                'Age Group': [age_group],
                'Lower Bound': [ci[0]],
                'Upper Bound': [ci[1]]
            })
        ], ignore_index=True)
```

```
Confidence Intervals for Age Group: under_18
Sample Size: 300, CI: (8350.76675, 9511.375166666667)
Sample Size: 3000, CI: (8748.884291666667, 9113.312116666668)
Sample Size: 30000, CI: (8876.187114166667, 8990.469249166666)
Confidence Intervals for Age Group: over_18_to_25
Sample Size: 300, CI: (8611.556416666668, 9753.995499999999)
Sample Size: 3000, CI: (8985.358558333333, 9353.235858333333)
Sample Size: 30000, CI: (9112.7077325, 9226.133791666665)
Confidence Intervals for Age Group: over_25_to_35
Sample Size: 300, CI: (8690.93275, 9825.329333333333)
Sample Size: 3000, CI: (9074.833058333334, 9429.144)
Sample Size: 30000, CI: (9196.752470833333, 9309.900311666666)
Confidence Intervals for Age Group: over_35_to_45
Sample Size: 300, CI: (8766.125416666668, 9903.70975)
Sample Size: 3000, CI: (9156.602483333332, 9511.5147)
Sample Size: 30000, CI: (9274.539663333333, 9388.605986666666)
Confidence Intervals for Age Group: over_45_to_50
Sample Size: 300, CI: (8644.233333333334, 9775.058083333333)
Sample Size: 3000, CI: (9031.189675, 9384.549858333334)
Sample Size: 30000, CI: (9152.6309075, 9265.767978333333)
Confidence Intervals for Age Group: over_50_to_55
Sample Size: 300, CI: (8964.120166666666, 10121.768083333334)
Sample Size: 3000, CI: (9350.308858333334, 9716.131041666667)
Sample Size: 30000, CI: (9476.929204166667, 9591.864839166667)
```

```
Confidence Intervals for Age Group: over_55
Sample Size: 300, CI: (8770.536916666668, 9911.769499999999)
Sample Size: 3000, CI: (9156.086225000001, 9514.072308333332)
Sample Size: 30000, CI: (9280.021446666666, 9393.066071666666)
```

Inference: (Randomly 10000 samples drawn for each sample size of 300,3000, 30000 from the dataset)

1. It can be concluded from the above observation of confidence intervals that the gap between intervals gradually got reduced as sample size increased from 300 to 30000. Infact the difference was very low in all age bins regarding their purcahses in highest sample size. With lower difference, we can conclude that the mean calculated from the random 30000 sample size truly represents the population characteristics of the data.

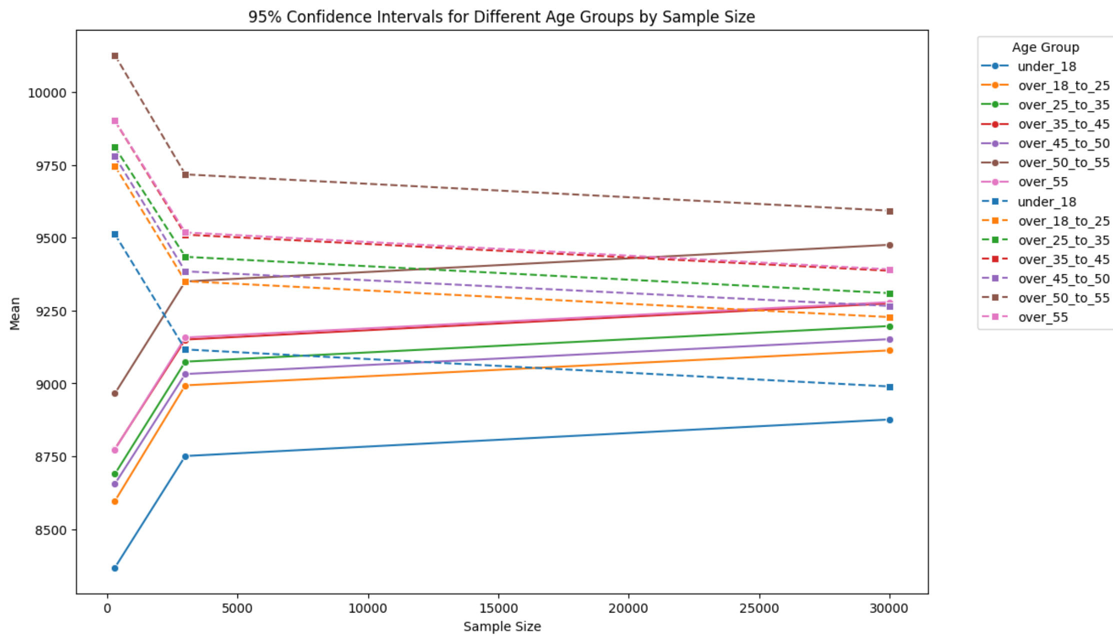
2. There has been evidence of overlapping of certain age bins between over_35_to_45 with over_55 samples of mean purchases at every sample size. Ands Age bins of over_18_to_25, over_25_to_35 and over_45_to_50 interact with under_18, thus the there was less behavioural difference between these age bins.

3. As the sample size increases, the sampling distribution of the sample mean approaches a normal distribution, regardless of the shape of the population distribution, according to the Central Limit Theorem. This means that for sufficiently large sample sizes, the distribution of sample means becomes more symmetric and bell-shaped.

4. With larger samples the variability also decreases, this was because larger samples provide more information about the population, thus gap between the intervals gets reduced.

Visual represenattion for sample sizes

[ ]:
```python
plt.figure(figsize=(12, 8))
sns.lineplot(data=cis_df, x='Sample Size', y='Lower Bound', hue='Age Group',
 ↪marker='o', linestyle='-')
sns.lineplot(data=cis_df, x='Sample Size', y='Upper Bound', hue='Age Group',
 ↪marker='s', linestyle='--')
plt.title('95% Confidence Intervals for Different Age Groups by Sample Size')
plt.xlabel('Sample Size')
plt.ylabel('Mean')
plt.legend(title='Age Group', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



[1]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (6.5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
```