```
!gdown 1IsD_gWZ7XAj9tNK73R6KQWJaIhOHTuoi -O 'Walmart_data.csv'
```

```
Downloading...
From: https://drive.google.com/uc?id=1IsD_gWZ7XAj9tNK73R6KQWJaIhOHTuoi
To: /content/Walmart_data.csv
100% 23.0M/23.0M [00:00<00:00, 148MB/s]
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
```

```
wmt = pd.read_csv('Walmart_data.csv')
wmt.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purcha
0	1000001	P00069042	F	0-17	10	A	2	0	3	83
1	1000001	P00248942	F	0-17	10	A	2	0	1	152
2	1000001	P00087842	F	0-17	10	A	2	0	12	14

```
wmt.shape
```

```
(550068, 10)
```

```
wmt.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
                                 550068 non-null  object
                                 550068 non-null  object
                                 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
```



```
wmt.describe()
```

	User_ID	Occupation	Marital_Status	Product_Category	Purchase
count	5.500680e+05	550068.000000	550068.000000	550068.000000	550068.000000
mean	1.003029e+06	8.076707	0.409653	5.404270	9263.968713
std	1.727592e+03	6.522660	0.491770	3.936211	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	12.000000
25%	1.001516e+06	2.000000	0.000000	1.000000	5823.000000
50%	1.003077e+06	7.000000	0.000000	5.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	23961.000000

```
wmt.describe(include=object)
```

```
wmt.shape
```

```
(550068, 10)
```

```
wmt.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
```

```
np.any(wmt.isna().any(axis=1))
```

```
False
```

```
wmt['Marital_Status'].unique()
```

```
array([0, 1])
```

```
wmt['Age'].unique()
```

```
array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
      dtype=object)
```

```
wmt['Gender'].unique()
```

```
array(['F', 'M'], dtype=object)
```

```
np.sort(wmt['Product_Category'].unique())
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

```
np.sort(wmt['Stay_In_Current_City_Years'].unique())
```

Saved successfully!

```
], dtype=object)
```

```
wmt['Gender'].value_counts()
```

```
M    414259
F    135809
Name: Gender, dtype: int64
```

```
wmt['Marital_Status'].value_counts()
```

```
0    324731
1    225337
Name: Marital_Status, dtype: int64
```

```
wmt['Age'].value_counts()
```

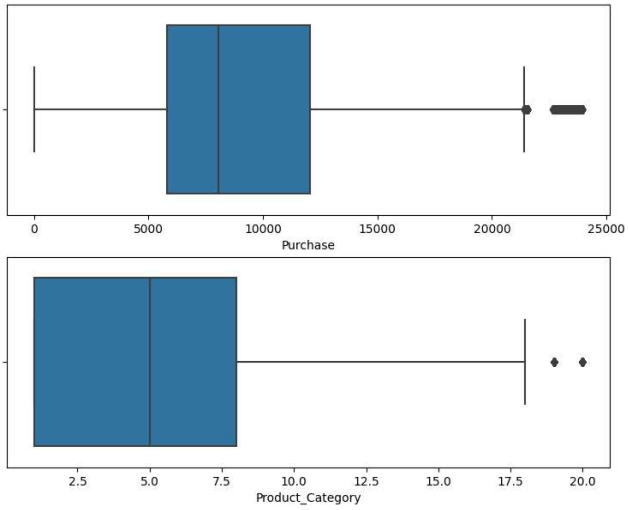
```
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: Age, dtype: int64
```

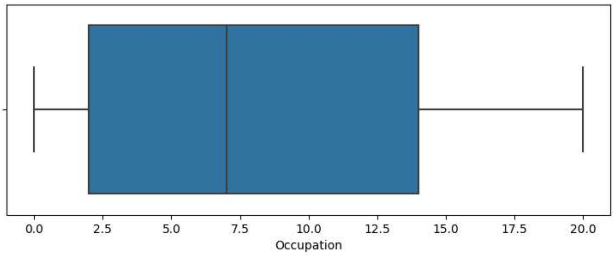
```
wmt['Product_Category'] = wmt['Product_Category'].astype(int)
```

```
#Outlier

fig = plt.figure(figsize=(20,7))

plt.subplot(2,2, 1)
sns.boxplot(data = wmt , x ='Purchase' )
```

```
plt.subplot(2,2, 2)
sns.boxplot(data = wmt , x = 'Occupation' )

plt.subplot(2,2, 3)
sns.boxplot(data = wmt , x = 'Product_Category' )
```

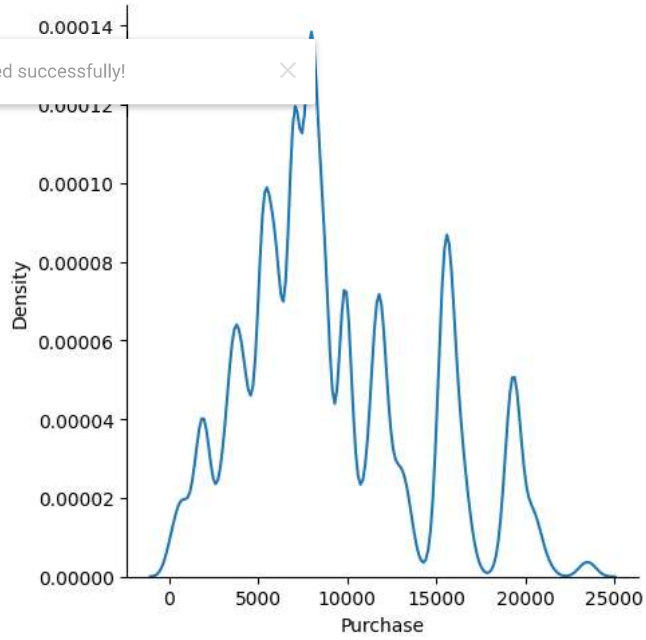
```
<Axes: xlabel='Product_Category'>
```





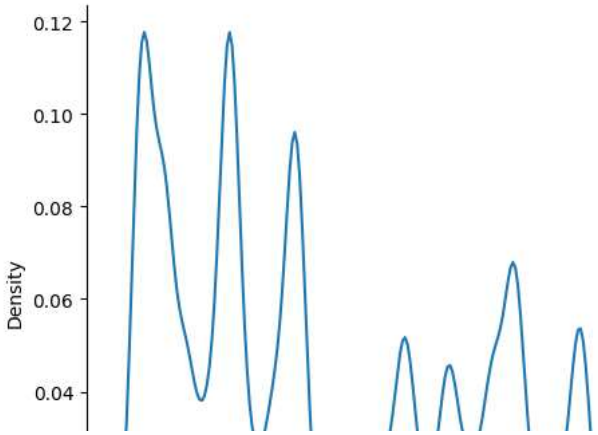

```
sns.displot(wmt['Purchase'],kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f1396498520>
```

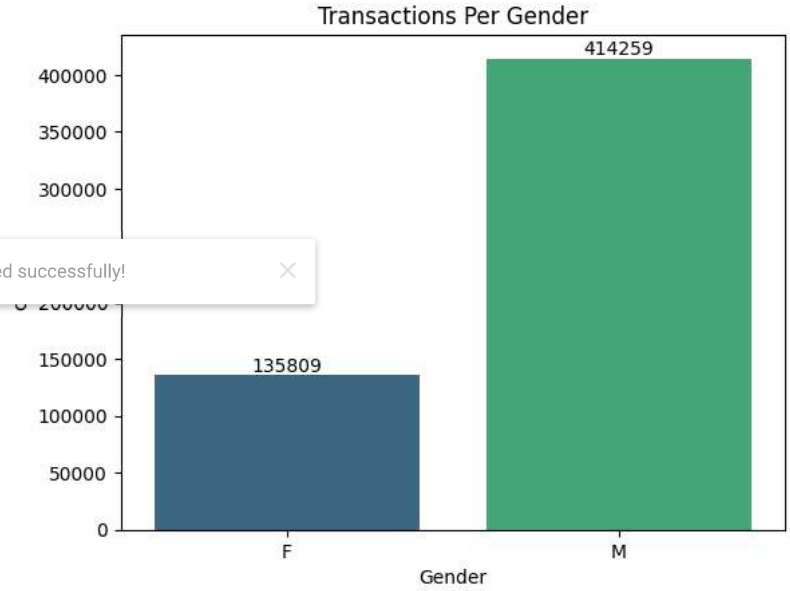



```
sns.displot(wmt['Occupation'],kind='kde')
```

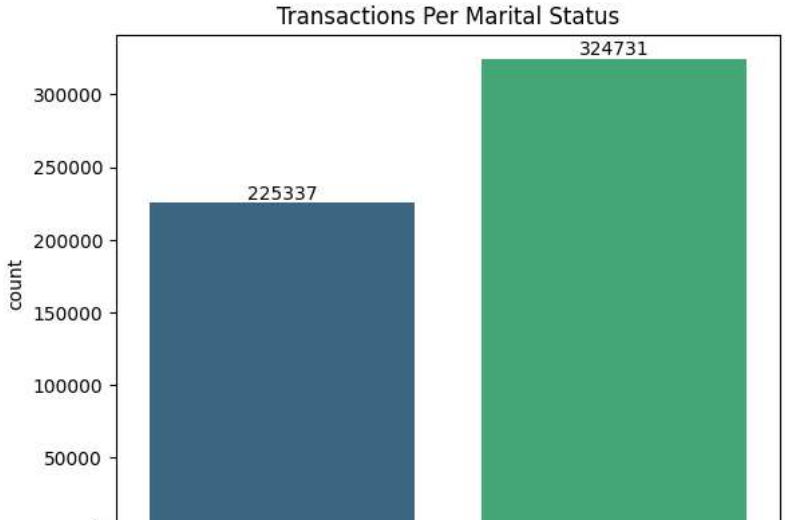
```
<seaborn.axisgrid.FacetGrid at 0x7f13950ff880>
```



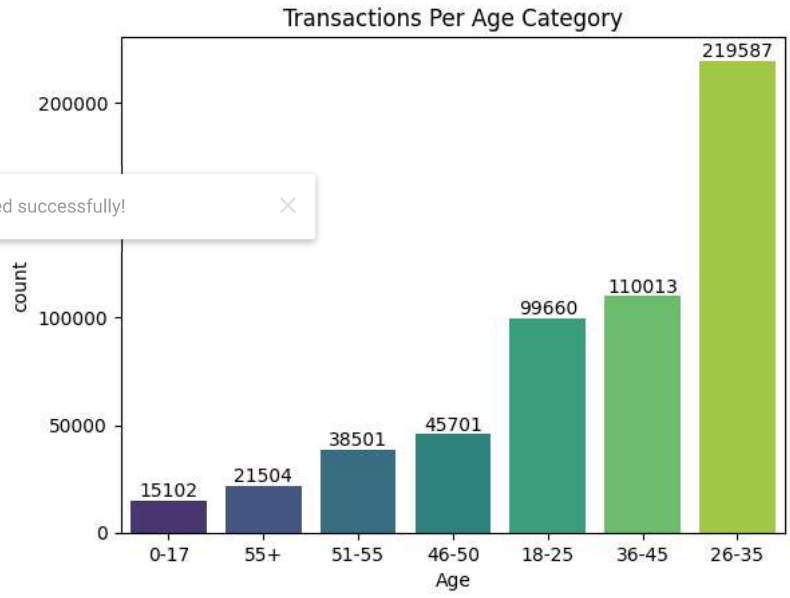

```
ax = sns.countplot(data = wmt,
              x = "Gender",
             order = list(wmt["Gender"].value_counts().index)[::-1],
             palette= "viridis")

plt.xticks(rotation = 0, fontsize = 10)


for i in ax.containers:
    ax.bar_label(i, fontsize = 10)


plt.title("Transactions Per Gender")

plt.show()
```




```
ax = sns.countplot(data = wmt,
              x = "Marital_Status",
             order = list(wmt["Marital_Status"].value_counts().index)[::-1],
             palette= "viridis")

plt.xticks(rotation = 0, fontsize = 10)


for i in ax.containers:
    ax.bar_label(i, fontsize = 10)


plt.title("Transactions Per Marital Status")

plt.show()
```

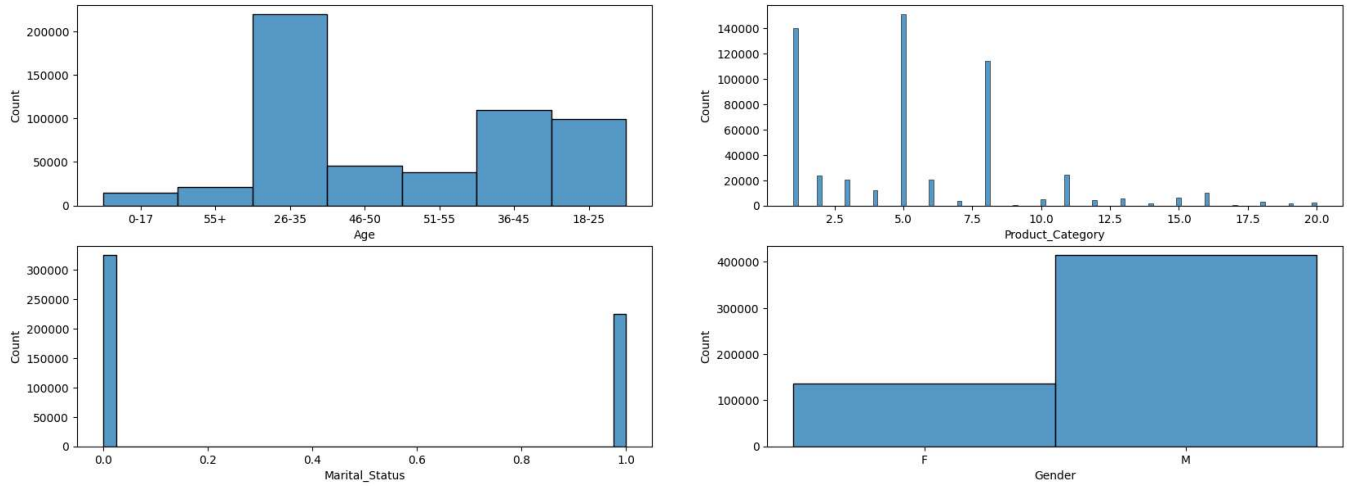



```
ax = sns.countplot(data = wmt,
                   x = "Age",
                  order = list(wmt["Age"].value_counts().index)[::-1],
                  palette= "viridis")

plt.xticks(rotation = 0, fontsize = 10)


for i in ax.containers:
    ax.bar_label(i, fontsize = 10)


plt.title("Transactions Per Age Category")

plt.show()
```



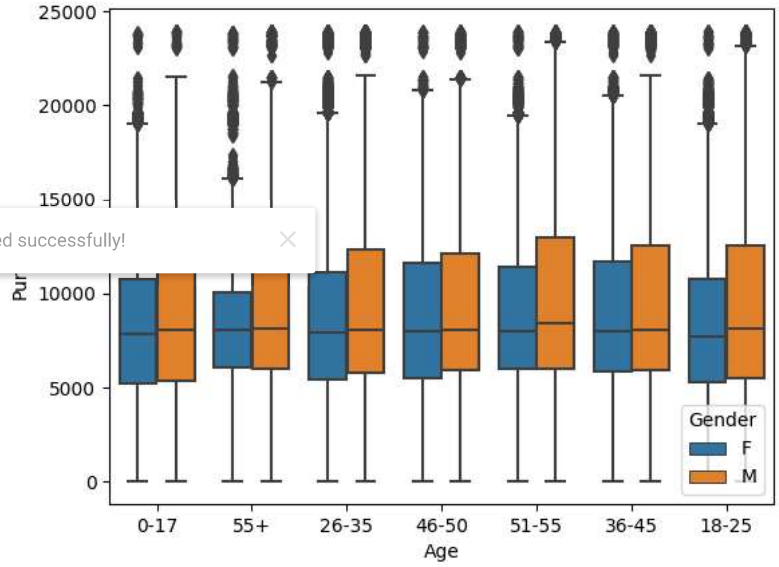

```
fig = plt.figure(figsize=(20,7))

plt.subplot(2,2, 1)
sns.histplot(data = wmt , x ='Age' )

plt.subplot(2,2, 2)
sns.histplot(data = wmt , x = 'Product_Category' )

plt.subplot(2,2, 3)
sns.histplot(data = wmt , x = 'Marital_Status' )

plt.subplot(2,2, 4)
sns.histplot(data = wmt , x = 'Gender' )
```

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




```
sns.boxplot(data=wmt,
            x="Age",
            y="Purchase",
             hue = "Gender")
```

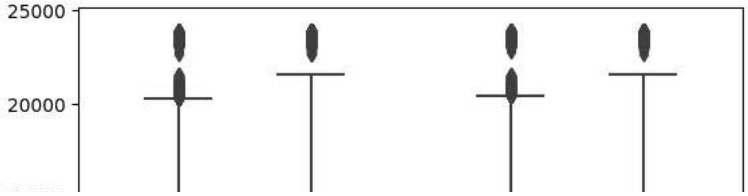
```
<Axes: xlabel='Age', ylabel='Purchase'>
```



Saved successfully!

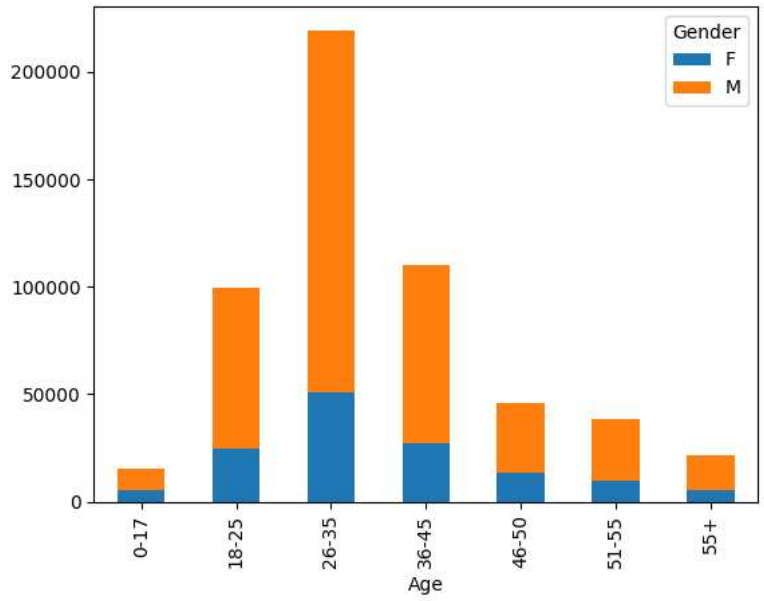
```
sns.boxplot(data=wmt,
            x="Marital_Status",
            y="Purchase",
             hue = "Gender")
```

```
<Axes: xlabel='Marital_Status', ylabel='Purchase'>
```



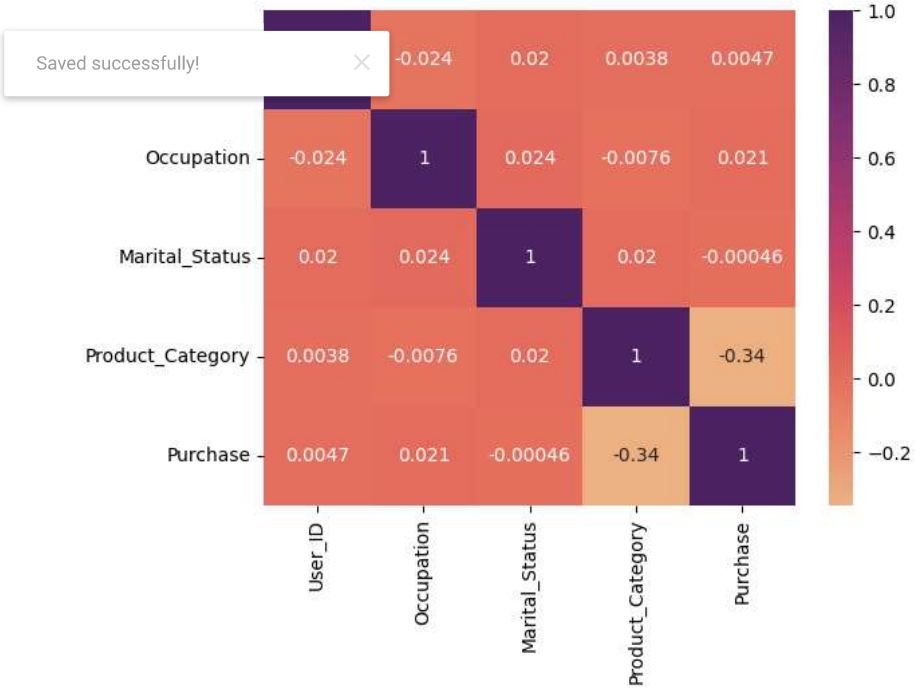

```
df_plot = pd.crosstab(index = wmt["Age"], columns=wmt["Gender"])
df_plot.plot(kind="bar",stacked=True)
```

```
<Axes: xlabel='Age'>
```




```
sns.heatmap(wmt.corr(),annot=True, cmap="flare")
```

```
<ipython-input-138-b1cff4e48085>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  sns.heatmap(wmt.corr(),annot=True, cmap="flare")
<Axes: >
```



CONFIDENCE INTERVAL using CLT

Male Population Transaction Purchase Mean **Confidence Interval using CLT**

```
malesTransactionAmount = wmt.loc[wmt['Gender'] =='M', 'Purchase' ]
malesSampleMean = malesTransactionAmount.mean()
malesSampleStandardDeviation = malesTransactionAmount.std()
# Standard Error Given by Central Limit Theorm
stdError = malesSampleStandardDeviation/ np.sqrt(len(malesTransactionAmount))


# 90% Confidence Interal for Male Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = malesSampleMean + z1 * stdError
x1
z2 = norm.ppf(0.95)
x2 = malesSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9424.512497305488 9450.539583639042
```

```
# 95% Confidence Interal for Male Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = malesSampleMean + z1 * stdError
x1
z2 = norm.ppf(0.975)
x2 = malesSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9422.01944736257 9453.032633581959
```

```
# 99% Confidence Interal for Male Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = malesSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = malesSampleMean + z2 * stdError
x2

print(x1,x2)
```

Saved successfully!

```
505
```

Female Population Transaction Purchase Mean **Confidence Interval using CLT**

```
femalesTransactionAmount = wmt.loc[wmt['Gender'] =='F', 'Purchase' ]
femalesSampleMean = femalesTransactionAmount.mean()
femalesSampleStandardDeviation = femalesTransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = femalesSampleStandardDeviation/ np.sqrt(len(femalesTransactionAmount))


# 90% Confidence Interal for Female Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = femalesSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = femalesSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
8713.287834648021 8755.84369566293
```

```
# 95% Confidence Interal for Female Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = femalesSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
```

```
x2 = femalesSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
8709.21154714068 8759.919983170272
```

```
# 99% Confidence Interal for Female Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = femalesSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = femalesSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
8701.244674438389 8767.886855872563
```

Married Population Transaction Purchase Mean **Confidence Interval using CLT**

```
marriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 1, 'Purchase' ]
marriedSampleMean = marriedTransactionAmount.mean()
marriedSampleStandardDeviation = marriedTransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = marriedSampleStandardDeviation/ np.sqrt(len(marriedTransactionAmount))


# 90% Confidence Interal for Married Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = marriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = marriedSampleMean + z2 * stdError
x2

print(x1,x2)
```

Saved successfully!

```
1702

# 95% Confidence Interal for Married Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = marriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = marriedSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9240.460427057078 9281.888721107669
```

```
# 99% Confidence Interal for Married Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = marriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = marriedSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9233.951570329937 9288.39757783481
```

UnMarried Population Transaction Purchase Mean **Confidence Interval using CLT**

```
unmarriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 0, 'Purchase' ]
unmarriedSampleMean = unmarriedTransactionAmount.mean()
unmarriedSampleStandardDeviation = unmarriedTransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = unmarriedSampleStandardDeviation/ np.sqrt(len(unmarriedTransactionAmount))


# 90% Confidence Interal for UnMarried Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = unmarriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = unmarriedSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9251.396385823671 9280.418852019342
```

```
# 95% Confidence Interal for UnMarried Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = unmarriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = unmarriedSampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9248.61641818668 9283.198819656332
```

```
# 99% Confidence Interal for UnMarried Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = unmarriedSampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = unmarriedSampleMean + z2 * stdError
x2
```

Saved successfully!

```
9243.183129136169 9288.632108706845
```

0-17 Age Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat1TransactionAmount = wmt.loc[wmt['Age'] == '0-17', 'Purchase' ]
ageCat1SampleMean = ageCat1TransactionAmount.mean()
ageCat1SampleStandardDeviation = ageCat1TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat1SampleStandardDeviation/ np.sqrt(len(ageCat1TransactionAmount))


# 90% Confidence Interal for 0-17 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat1SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat1SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
8865.053694527898 9001.87558636205
```

```
# 95% Confidence Interal for 0-17 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat1SampleMean + z1 * stdError
```

```
x1

z2 = norm.ppf(0.975)
x2 = ageCat1SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
     8851.947970542686 9014.981310347262
```

```
# 99% Confidence Interal for 0-17 age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat1SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = ageCat1SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
     8826.333576446717 9040.59570444323
```

18-25 Age Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat2TransactionAmount = wmt.loc[wmt['Age'] == '18-25', 'Purchase' ]
ageCat2SampleMean = ageCat2TransactionAmount.mean()
ageCat2SampleStandardDeviation = ageCat2TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat2SampleStandardDeviation/ np.sqrt(len(ageCat2TransactionAmount))

# 90% Confidence Interal for 18-25 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat2SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat2SampleMean + z2 * stdError
x2
```

Saved successfully!

```
     9143.433031607847 9195.89418091473
```

```
# 95% Confidence Interal for 18-25 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat2SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat2SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
     9138.407948753442 9200.919263769136
```

```
# 99% Confidence Interal for 18 -25 age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat2SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = ageCat2SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
     9128.586709366526 9210.740503156052
```

26-35 Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat3TransactionAmount = wmt.loc[wmt['Age'] == '26-35', 'Purchase' ]
ageCat3SampleMean = ageCat3TransactionAmount.mean()
ageCat3SampleStandardDeviation = ageCat3TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat3SampleStandardDeviation/ np.sqrt(len(ageCat3TransactionAmount))


# 90% Confidence Interal for 26-35 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat3SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat3SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9235.103000581124 9270.278265158651
```

```
# 95% Confidence Interal for 26-35 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat3SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat3SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9231.733676400028 9273.647589339747
```

```
# 99% Confidence Interal for 26-35 age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat3SampleMean + z1 * stdError
```

Saved successfully!

```
x2 = ageCat3SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9225.148523415806 9280.23274232397
```

36-45 Age Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat4TransactionAmount = wmt.loc[wmt['Age'] == '36-45', 'Purchase' ]
ageCat4SampleMean = ageCat4TransactionAmount.mean()
ageCat4SampleStandardDeviation = ageCat4TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat4SampleStandardDeviation/ np.sqrt(len(ageCat4TransactionAmount))


# 90% Confidence Interal for 36-45 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat4SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat4SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9306.441376202305 9356.260013633442
```

```
# 95% Confidence Interal for 36-45 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat4SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat4SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9301.669410965314 9361.031978870433
```

```
# 99% Confidence Interal for 36-45 age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat4SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = ageCat4SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9292.342875603326 9370.358514232421
```

46-50 Age Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat5TransactionAmount = wmt.loc[wmt['Age'] == '46-50', 'Purchase' ]
ageCat5SampleMean = ageCat5TransactionAmount.mean()
ageCat5SampleStandardDeviation = ageCat5TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat5SampleStandardDeviation/ np.sqrt(len(ageCat5TransactionAmount))

# 90% Confidence Interal for 46-50 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat5SampleMean + z1 * stdError
```

Saved successfully!

```
x2 = ageCat5SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9170.406859081895 9246.84453585476
```

```
# 95% Confidence Interal for 46-50 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat5SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat5SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9163.085142648752 9254.166252287903
```

```
# 99% Confidence Interal for 46-50 age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat5SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = ageCat5SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9148.775263210646 9268.476131726009
```

51-55 Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat6TransactionAmount = wmt.loc[wmt['Age'] == '51-55', 'Purchase' ]
ageCat6SampleMean = ageCat6TransactionAmount.mean()
ageCat6SampleStandardDeviation = ageCat6TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat6SampleStandardDeviation/ np.sqrt(len(ageCat6TransactionAmount))


# 90% Confidence Interal for 51-55 age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat6SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat6SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9492.161430973249 9577.454630947223
```

```
# 95% Confidence Interal for 51-55 age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat6SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat6SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9483.991472776577 9585.624589143894
```

```
# 99% Confidence Interal for 51-55 age Population Transaction Purchase Mean
```

Saved successfully!

```
or
x1

z2 = norm.ppf(0.995)
x2 = ageCat6SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9468.02375292888 9601.59230899159
```

55+ Age Population Transaction Purchase Mean **Confidence Interval using CLT**

```
ageCat7TransactionAmount = wmt.loc[wmt['Age'] == '55+', 'Purchase' ]
ageCat7SampleMean = ageCat7TransactionAmount.mean()
ageCat7SampleStandardDeviation = ageCat7TransactionAmount.std()

# Standard Error Given by Central Limit Theorm
stdError = ageCat7SampleStandardDeviation/ np.sqrt(len(ageCat7TransactionAmount))


# 90% Confidence Interal for 55+ age Population Transaction Purchase Mean

z1 = norm.ppf(0.05)
x1 = ageCat7SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.95)
x2 = ageCat7SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9280.067707714425 9392.493211184385
```

```
# 95% Confidence Interal for 55+ age Population Transaction Purchase Mean

z1 = norm.ppf(0.025)
x1 = ageCat7SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.975)
x2 = ageCat7SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9269.29883441773 9403.262084481079
```

```
# 99% Confidence Interal for 55+ age Population Transaction Purchase Mean

z1 = norm.ppf(0.005)
x1 = ageCat7SampleMean + z1 * stdError
x1

z2 = norm.ppf(0.995)
x2 = ageCat7SampleMean + z2 * stdError
x2

print(x1,x2)
```

```
9248.251682432667 9424.309236466142
```

CONFIDENCE INTERVAL using **BOOTSTRAPPING**

Male Population Transaction Purchase Mean Confidence Interval using BOOTSTRAPPING

```
malesTransactionAmount = wmt.loc[wmt['Gender'] =='M', 'Purchase' ]

malesTransactionAmount_means = []
```

Saved successfully!

```
for reps in range(100000):
    bootstrapped_samples = np.random.choice(malesTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    malesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(malesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(malesTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(malesTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 6872.4   12164.305] [ 6420.9975 12704.1075] [ 5629.0965 13822.4055]
```

```
malesTransactionAmount = wmt.loc[wmt['Gender'] =='M', 'Purchase' ]

malesTransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(malesTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    malesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(malesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(malesTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(malesTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 7940.03333333 10992.26833333] [ 7672.9325 11289.2675] [ 7147.9665     11877.00383333]
```

```
malesTransactionAmount = wmt.loc[wmt['Gender'] =='M', 'Purchase' ]

malesTransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(malesTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    malesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(malesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(malesTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(malesTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 8606.6895 10283.121 ] [ 8452.70925 10447.95225] [ 8154.635   10781.59135]
```

Female Population Transaction Purchase Mean Confidence Interval using BOOTSTRAPPING

```
femalesTransactionAmount = wmt.loc[wmt['Gender'] =='F', 'Purchase' ]

femalesTransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(femalesTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    femalesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(femalesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(femalesTransactionAmount_means, [2.5, 97.5])
```

Saved successfully!

```
ctionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 6351.69 11339.5 ] [ 5963.   11871.81] [ 5229.0995 12948.7175]
```

```
femalesTransactionAmount = wmt.loc[wmt['Gender'] =='F', 'Purchase' ]

femalesTransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(femalesTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    femalesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(femalesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(femalesTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(femalesTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 7338.6        10205.17333333] [ 7085.89       10504.80916667] [ 6601.1325     11075.73383333]
```

```
femalesTransactionAmount = wmt.loc[wmt['Gender'] =='F', 'Purchase' ]

femalesTransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(femalesTransactionAmount, size=n)
```

```
    bootstrapped_mean = np.mean(bootstrapped_samples)
    femalesTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(femalesTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(femalesTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(femalesTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [7961.5185 9526.711 ] [7818.0185  9686.39025] [ 7539.0079 10001.513 ]
```

Married Population Transaction Purchase Mean Confidence Interval using BOOTSTRAPPING

```
marriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 1, 'Purchase' ]

marriedTransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(marriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    marriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(marriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(marriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(marriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6738.095 11955.71 ] [ 6317.0875 12500.815 ] [ 5509.079 13604.41 ]
```

```
marriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 1, 'Purchase' ]

marriedTransactionAmount_means = []
```

Saved successfully!

```
    bootstrapped_samples = np.random.choice(marriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    marriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(marriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(marriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(marriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7778.5        10792.10833333] [ 7514.26583333 11098.97416667] [ 7012.39966667 11702.204     ]
```

```
marriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 1, 'Purchase' ]

marriedTransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(marriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    marriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(marriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(marriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(marriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 8443.7185 10094.6725] [ 8293.6135  10253.97125] [ 7992.66765 10584.76395]
```

UnMarried Population Transaction Purchase Mean Confidence Interval using BOOTSTRAPPING

```
unmarriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 0, 'Purchase' ]

unmarriedTransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(unmarriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    unmarriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(unmarriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(unmarriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(unmarriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

Saved successfully!

```
25 12502.1   ] [ 5515.582 13578.503]
```

```
unmarriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 0, 'Purchase' ]

unmarriedTransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(unmarriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    unmarriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(unmarriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(unmarriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(unmarriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7770.2        10801.60166667] [ 7508.895  11112.5675] [ 6993.732      11718.36883333]
```

```
unmarriedTransactionAmount = wmt.loc[wmt['Marital_Status'] == 0, 'Purchase' ]

unmarriedTransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(unmarriedTransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    unmarriedTransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
```

```
boot90 = np.percentile(unmarriedTransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(unmarriedTransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(unmarriedTransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 8446.257  10101.2705] [ 8294.04925 10267.89725] [ 7994.20885 10588.46245]
```

0-17 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat1TransactionAmount = wmt.loc[wmt['Age'] == '0-17', 'Purchase' ]

ageCat1TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat1TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat1TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat1TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat1TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat1TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6379.9   11678.415] [ 5936.4975 12229.4125] [ 5109.595 13331.303]
```

```
ageCat1TransactionAmount = wmt.loc[wmt['Age'] == '0-17', 'Purchase' ]

ageCat1TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat1TransactionAmount, size=n)
                                            strapped_samples)
                                            ppend(bootstrapped_mean)
```

Saved successfully!

```
## 90 % CI
boot90 = np.percentile(ageCat1TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat1TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat1TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7417.49833333 10502.47166667] [ 7147.43       10812.23416667] [ 6645.29866667 11413.23616667]
```

```
ageCat1TransactionAmount = wmt.loc[wmt['Age'] == '0-17', 'Purchase' ]

ageCat1TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat1TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat1TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat1TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat1TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat1TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [8097.267  9778.7135] [7943.2615  9945.95175] [ 7647.0899  10280.46095]
```

18-25 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat2TransactionAmount = wmt.loc[wmt['Age'] == '18-25', 'Purchase' ]

ageCat2TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat2TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat2TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat2TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat2TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat2TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6649.99  11868.815] [ 6213.4975 12421.41  ] [ 5391.296  13545.8005]
```

```
ageCat2TransactionAmount = wmt.loc[wmt['Age'] == '18-25', 'Purchase' ]

ageCat2TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat2TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat2TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat2TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat2TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat2TransactionAmount_means, [0.5, 99.5])

                                                       n' )
```

Saved successfully!

```
     [ 7415.19416667 11013.74166667] [ 6907.766 11633.568]
```

```
ageCat2TransactionAmount = wmt.loc[wmt['Age'] == '18-25', 'Purchase' ]

ageCat2TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat2TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat2TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat2TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat2TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat2TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 8349.799 10008.021] [ 8193.775   10169.19225] [ 7888.88945 10494.2623 ]
```

26-35 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat3TransactionAmount = wmt.loc[wmt['Age'] == '26-35', 'Purchase' ]

ageCat3TransactionAmount_means = []
n = 10
```

```
for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat3TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat3TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat3TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat3TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat3TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6743.295 11958.43 ] [ 6312.0925 12521.905 ] [ 5491.1995 13607.701 ]
```

```
ageCat3TransactionAmount = wmt.loc[wmt['Age'] == '26-35', 'Purchase' ]

ageCat3TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat3TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat3TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat3TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat3TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat3TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7779.06333333 10787.74      ] [ 7504.16666667 11089.33833333] [ 7002.166      11681.73566667]
```

```
ageCat3TransactionAmount = wmt.loc[wmt['Age'] == '26-35', 'Purchase' ]

ageCat3TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat3TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat3TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat3TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat3TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat3TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

Saved successfully!

```
     [ 8440.2    10080.5405] [ 8288.4855 10236.89  ] [ 7985.86335 10565.1032 ]
```

36-45 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat4TransactionAmount = wmt.loc[wmt['Age'] == '36-45', 'Purchase' ]

ageCat4TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat4TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat4TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat4TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat4TransactionAmount_means, [2.5, 97.5])
## 99 % CI
```

```
boot99 = np.percentile(ageCat4TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6811.085 12042.205] [ 6369.595  12573.2125] [ 5600.399 13680.601]
```

```
ageCat4TransactionAmount = wmt.loc[wmt['Age'] == '36-45', 'Purchase' ]

ageCat4TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat4TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat4TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat4TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat4TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat4TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7858.76666667 10877.00666667] [ 7581.36583333 11177.83666667] [ 7069.665      11769.28016667]
```

```
ageCat4TransactionAmount = wmt.loc[wmt['Age'] == '36-45', 'Purchase' ]

ageCat4TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat4TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat4TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat4TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat4TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat4TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

Saved successfully!

```
567   10337.76175] [ 8063.68165 10668.9427 ]
```

46-50 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat5TransactionAmount = wmt.loc[wmt['Age'] == '46-50', 'Purchase' ]

ageCat5TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat5TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat5TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat5TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat5TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat5TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6726.695 11894.405] [ 6284.9975 12447.4   ] [ 5485.897 13464.904]
```

```
ageCat5TransactionAmount = wmt.loc[wmt['Age'] == '46-50', 'Purchase' ]

ageCat5TransactionAmount_means = []
```

```
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat5TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat5TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat5TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat5TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat5TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 7746.06166667 10731.345     ] [ 7479.59666667 11038.07083333] [ 6991.53033333 11620.60333333]
```

```
ageCat5TransactionAmount = wmt.loc[wmt['Age'] == '46-50', 'Purchase' ]

ageCat5TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat5TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat5TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat5TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat5TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat5TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 8401.7885 10033.701 ] [ 8247.96775 10198.76425] [ 7955.2597 10522.6743]
```

51-55 Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**


```
ageCat6TransactionAmount = wmt.loc[wmt['Age'] == '51-55', 'Purchase' ]

ageCat6TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat6TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat6TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat6TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat6TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat6TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 6968.89 12284.3 ] [ 6535.     12843.3025] [ 5720.0965 13885.704 ]
```

```
ageCat6TransactionAmount = wmt.loc[wmt['Age'] == '51-55', 'Purchase' ]

ageCat6TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat6TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat6TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat6TransactionAmount_means, [5, 95])
```

```
## 95 % CI
boot95 = np.percentile(ageCat6TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat6TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 8037.53333333 11085.27666667] [ 7770.69916667 11398.20416667] [ 7254.733      12010.46816667]
```

```
ageCat6TransactionAmount = wmt.loc[wmt['Age'] == '51-55', 'Purchase' ]

ageCat6TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat6TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat6TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat6TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat6TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat6TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 8709.6795 10378.801 ] [ 8548.569  10540.9205] [ 8241.71805 10865.69105]
```

55+ Age Population Transaction Purchase Mean **Confidence Interval using BOOTSTRAPPING**

```
ageCat7TransactionAmount = wmt.loc[wmt['Age'] == '55+', 'Purchase' ]

ageCat7TransactionAmount_means = []
n = 10

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat7TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
                                ppend(bootstrapped_mean)
```

Saved successfully!

```
boot90 = np.percentile(ageCat7TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat7TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat7TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 6827.27 12045.81] [ 6382.295  12594.5025] [ 5588.69   13652.8025]
```

```
ageCat7TransactionAmount = wmt.loc[wmt['Age'] == '55+', 'Purchase' ]

ageCat7TransactionAmount_means = []
n = 30

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat7TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat7TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat7TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat7TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat7TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
[ 7851.89833333 10873.00166667] [ 7576.06583333 11171.30083333] [ 7072.96483333 11765.96966667]
```

```
ageCat7TransactionAmount = wmt.loc[wmt['Age'] == '55+', 'Purchase' ]
```

```
ageCat7TransactionAmount_means = []
n = 100

for reps in range(100000):
    bootstrapped_samples = np.random.choice(ageCat7TransactionAmount, size=n)
    bootstrapped_mean = np.mean(bootstrapped_samples)
    ageCat7TransactionAmount_means.append(bootstrapped_mean)

## 90 % CI
boot90 = np.percentile(ageCat7TransactionAmount_means, [5, 95])
## 95 % CI
boot95 = np.percentile(ageCat7TransactionAmount_means, [2.5, 97.5])
## 99 % CI
boot99 = np.percentile(ageCat7TransactionAmount_means, [0.5, 99.5])

print(boot90,boot95,boot99, end = '\n' )
```

```
     [ 8524.1365 10168.2315] [ 8369.407   10332.47025] [ 8069.7015 10660.8905]
```

```
plt.figure(figsize=(8,8))
sns.pairplot(data=wmt)
```