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


# df = pd.read_csv("original_walmart_data.csv")
# df.head()

from google.colab import drive
drive.mount('/content/drive')

# !ls /content/drive/MyDrive/'Colab Notebooks'/neoversity/

df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/neoversity/original_walmart_data.csv")
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	

```
df.shape
```

```
(550068, 10)
```

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

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

```
df.info()
```

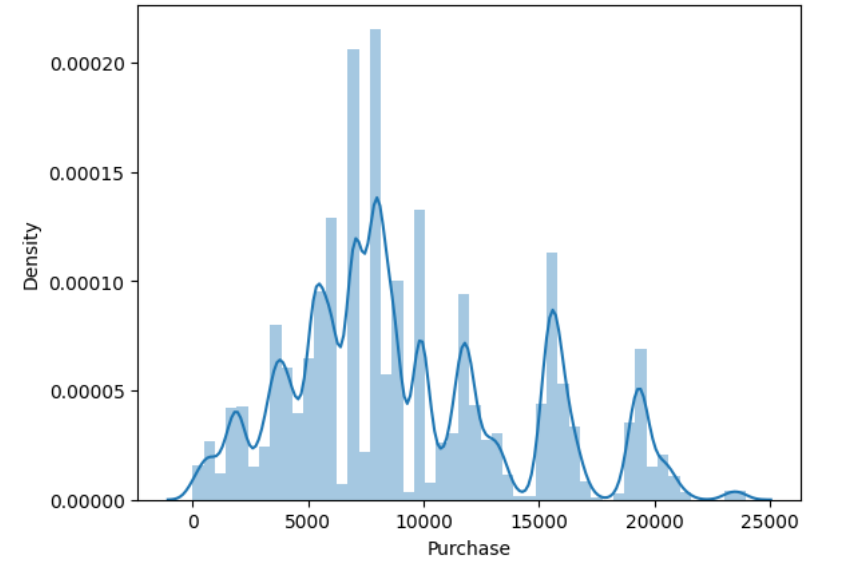
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     550068 non-null  int64
 1   Product_ID  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

```
df.isnull().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
```

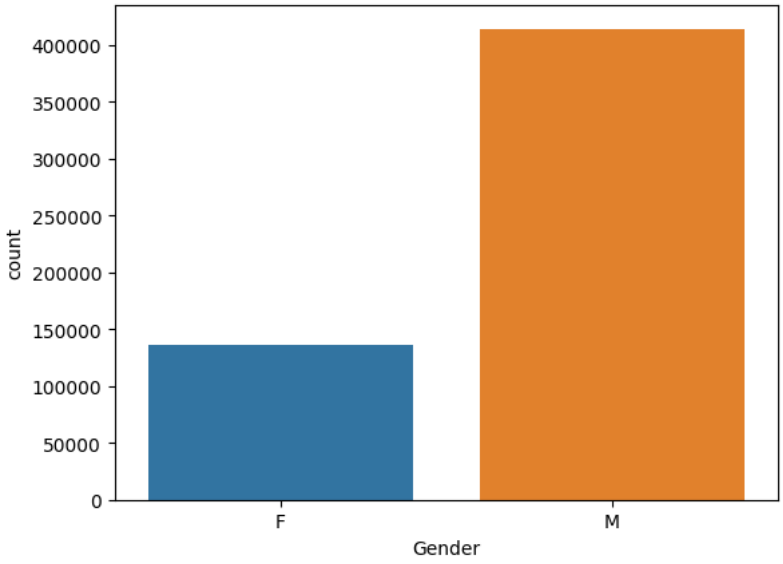
```
# Plot the distribution
sns.distplot(df["Purchase"])
```

```
<ipython-input-7-22c6b7339847>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function wit
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["Purchase"])
<Axes: xlabel='Purchase', ylabel='Density'>
```



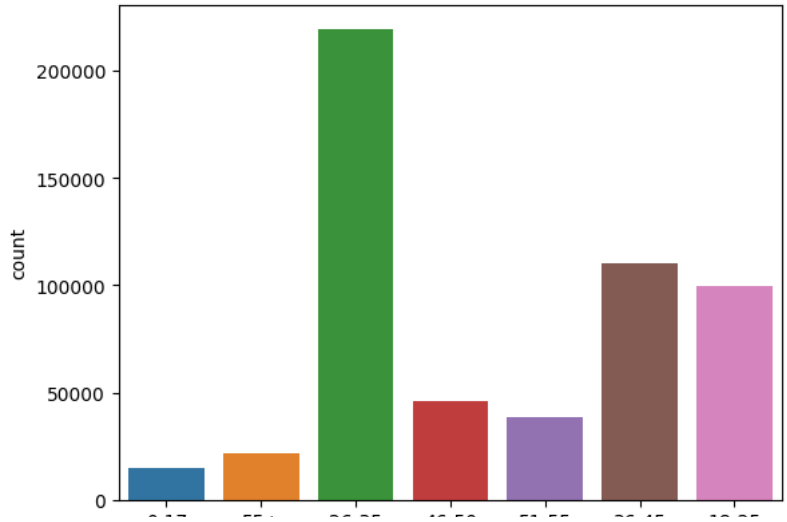

```
sns.countplot(x="Gender", data=df)
```

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



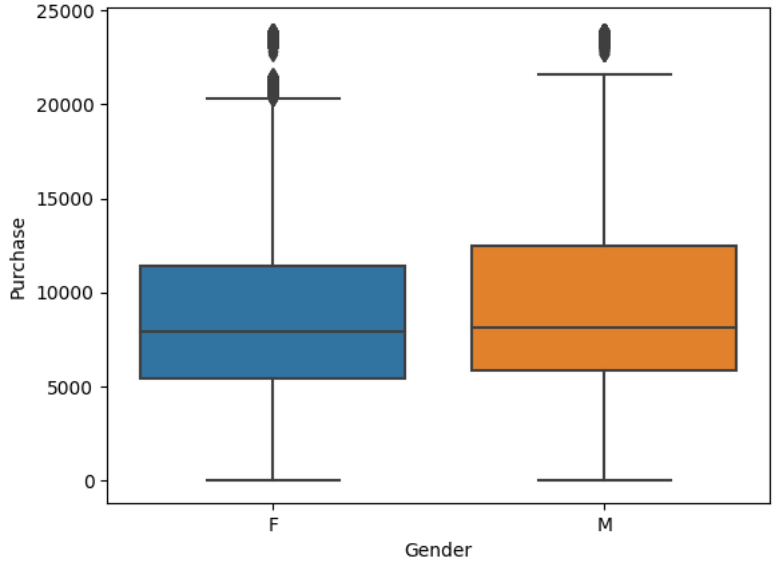

```
sns.countplot(x="Age", data=df)
```

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



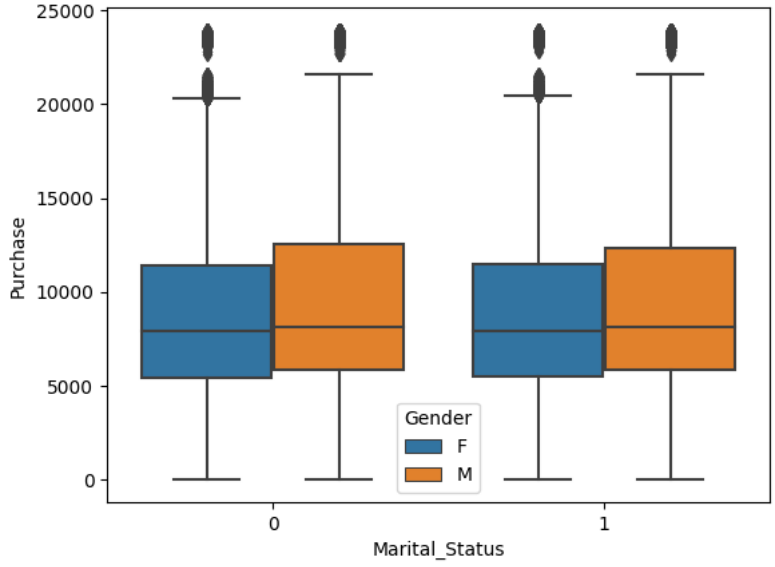

```
sns.boxplot(x="Gender", y="Purchase", data=df)

# Not much signifiance in distribution but show a little high purchase distribution from males.
```

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



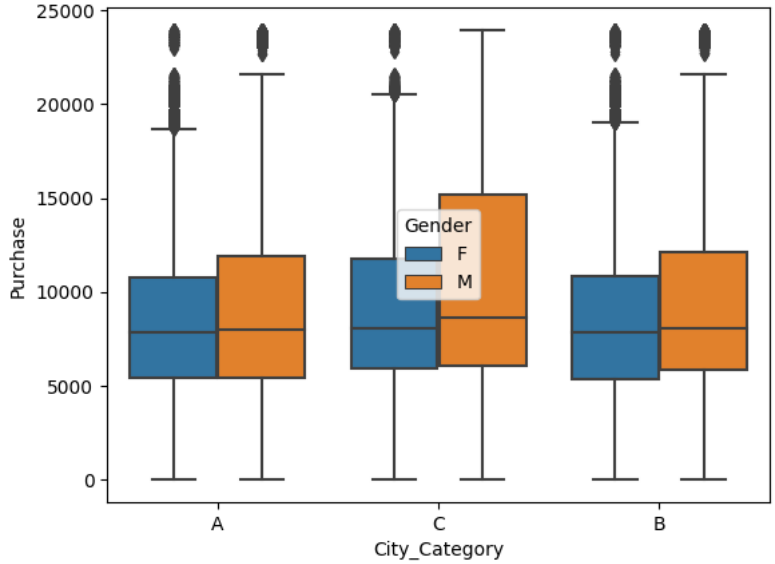

```
sns.boxplot(x="Marital_Status", y="Purchase", data=df, hue="Gender", )
```

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



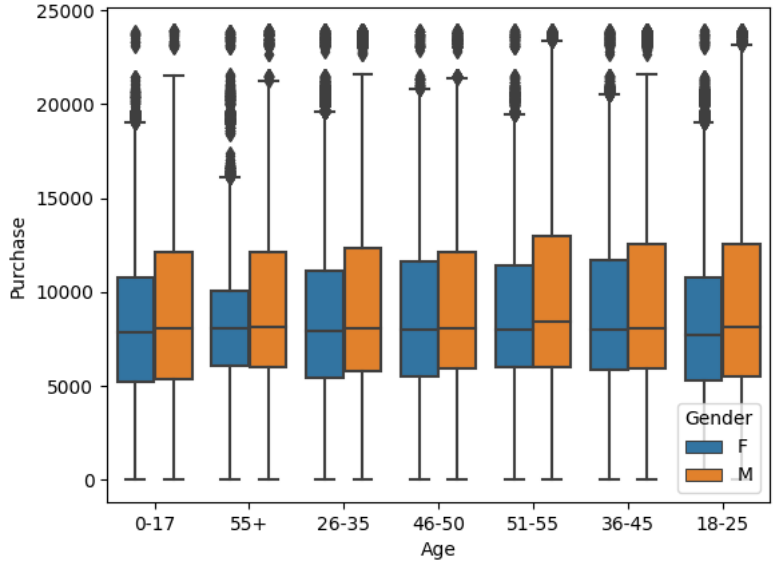

```
sns.boxplot(x="City_Category", y="Purchase", data=df, hue="Gender", )
```

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



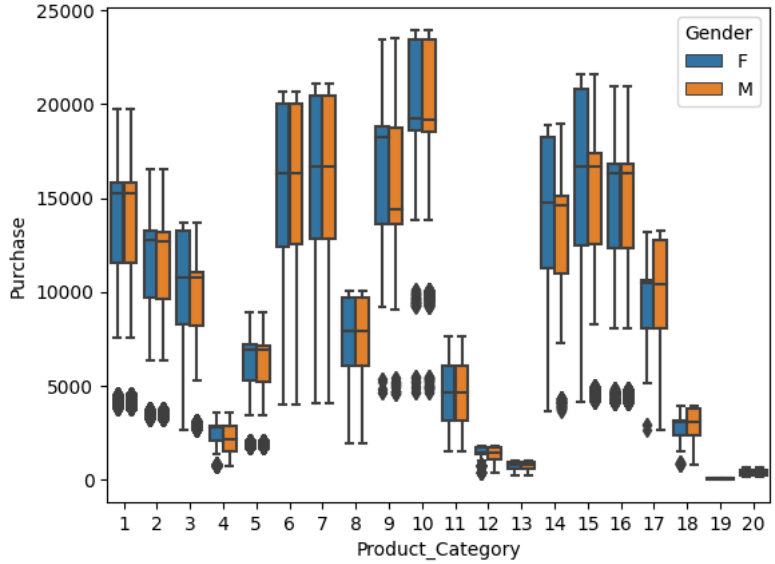

```
sns.boxplot(x=df["Age"], y=df["Purchase"],hue=df["Gender"])
```

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



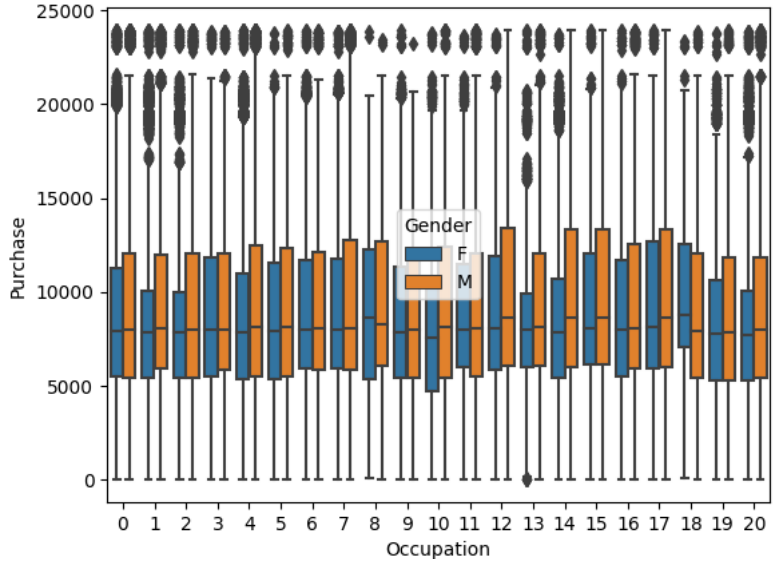

```
sns.boxplot(x=df["Product_Category"], y=df["Purchase"],hue=df["Gender"])
```

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




```
sns.boxplot(x=df["Occupation"], y=df["Purchase"], hue=df["Gender"])
```

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

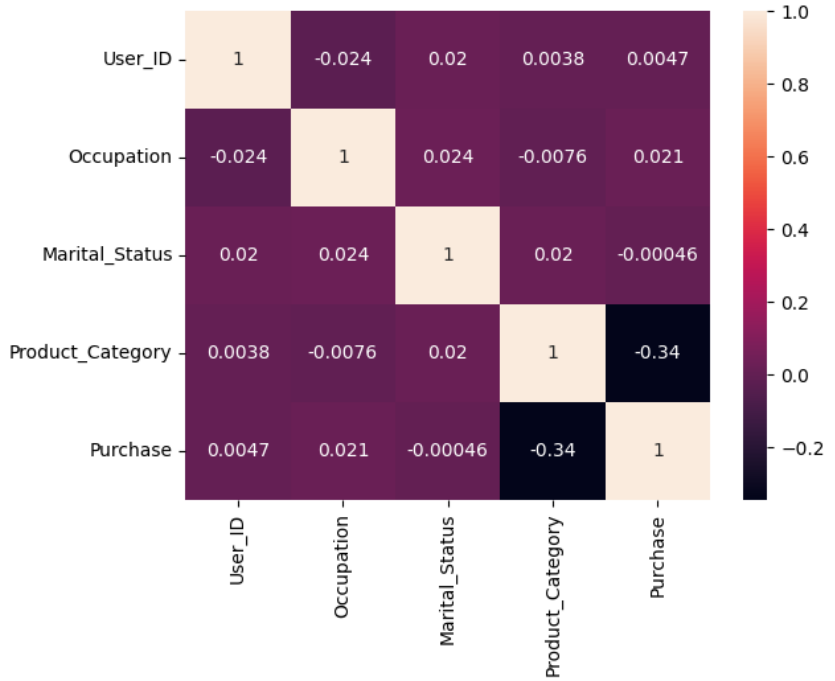



```
df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Curr
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	

```
sns.heatmap(df.corr(), annot=True)
```

```
<ipython-input-17-6dc1c4c1753e>:1: FutureWarning: The default value of nume
  sns.heatmap(df.corr(), annot=True)
<Axes: >
```




```
df.groupby(["Gender"])["Purchase"].describe()
```

Gender	count	mean	std	min	25%	50%	75%	max
F	135809.0	8734.565765	4767.233289	12.0	5433.0	7914.0	11400.0	23959.0
M	414259.0	9437.526040	5092.186210	12.0	5863.0	8098.0	12454.0	23961.0

```
df.groupby(["Gender","Marital_Status"])["Purchase"].describe()
```

Gender	Marital_Status	count	mean	std	min	25%	50%
F	0	78821.0	8679.845815	4740.048367	12.0	5417.00	7895.0
	1	56988.0	8810.249789	4803.594163	12.0	5456.75	7939.0
M	0	245910.0	9453.756740	5101.803346	12.0	5854.00	8101.0
	1	168349.0	9413.817605	5078.027482	12.0	5874.00	8094.0

```
df.groupby(["Gender","Age"])["Purchase"].describe()
```

Gender	Age	count	mean	std	min	25%	50%	75%	
F	0-17	5083.0	8338.771985	4850.032944	12.0	5243.0	7824.0	10755.00	238
	18-25	24628.0	8343.180201	4688.707126	12.0	5274.0	7731.0	10779.25	239
	26-35	50752.0	8728.251754	4718.826059	12.0	5442.0	7886.0	11101.25	239
	36-45	27170.0	8959.844056	4833.296586	12.0	5821.0	7984.0	11697.75	239
	46-50	13199.0	8842.098947	4795.838799	12.0	5472.5	7957.0	11601.00	239
	51-55	9894.0	9042.449666	4848.718221	12.0	5989.0	8002.0	11375.75	239
	55+	5083.0	9007.036199	4801.556874	12.0	6039.5	8084.0	10067.00	238
M	0-17	10019.0	9235.173670	5212.954953	12.0	5372.0	8080.0	12121.50	239
	18-25	75032.0	9440.942971	5113.697699	12.0	5475.0	8119.0	12561.00	239

```
pd.crosstab(index=df["Product_Category"], columns=df["Gender"], margins=True)
```

Gender	F	M	All
Product_Category			

```
pd.crosstab(index=df["Product_Category"], columns=df["Gender"], margins=True, normalize="index")
```

Gender	F	M
Product_Category		
1	0.176887	0.823113
2	0.237094	0.762906
3	0.297136	0.702864
4	0.309623	0.690377
5	0.278011	0.721989
6	0.222760	0.777240
7	0.253426	0.746574
8	0.294562	0.705438
9	0.170732	0.829268
10	0.226732	0.773268
11	0.195125	0.804875
12	0.388143	0.611857
13	0.263471	0.736529
14	0.409061	0.590939
15	0.166296	0.833704
16	0.244404	0.755596
17	0.107266	0.892734
18	0.122240	0.877760
19	0.281347	0.718653
20	0.283529	0.716471
All	0.246895	0.753105

```
pd.crosstab(index=df["Product_Category"], columns=df["Gender"], margins=True, normalize="columns")
```

Gender	F	M	All
1	0.182838	0.278925	0.255201

```
df["Gender"].value_counts()
```

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

5	0.208971	0.263053	0.274390

```
df["Marital_Status"].value_counts()
```

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

```
df["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
```

```
np.mean(df["Purchase"])
```

```
9263.968712959126
```

18	0.002813	0.006621	0.005681

```
np.std(df["Purchase"])
```

```
5023.060827959928
```

```
## Female average spending
np.mean(df.loc[df["Gender"]=="F"]["Purchase"])
```

```
8734.565765155476
```

```
## Male average spending
np.mean(df.loc[df["Gender"]=="M"]["Purchase"])
```

```
9437.526040472265
```

```
# Define the sample sizes for each gender
female_sample_size = 5000  # Adjust as per your desired sample size
male_sample_size = 5000  # Adjust as per your desired sample size

# Stratified sampling
female_data = df[df['Gender'] == 'F'].sample(n=female_sample_size, random_state=4)
male_data = df[df['Gender'] == 'M'].sample(n=male_sample_size, random_state=4)

# Combine the sampled data
sampled_data = pd.concat([female_data, male_data])
```

```
sampled_data.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_
199151	1000759	P00052642	F	55+	3	C	
153444	1005718	P00057742	F	36-45	14	C	
170452	1002264	P00050742	F	46-50	7	C	
428184	1001693	P00286942	F	26-	7	C	

```
def calc_CI(mean, std, N, prob):
    std_err = std / np.sqrt(N)
    # print("SE ", std_err)
    slice = (1 - (prob/100))/2
    # print("slice ", slice)
```

```
    z1 = norm.ppf(slice)
    # print("z1 ", z1)
    z2 = norm.ppf(1-slice)
    # print("z2 ", z2)
    x1 = mean + ( z1 * std_err)
    x2 = mean + ( z2 * std_err)
    return x1, x2


male_mean = np.mean(sampled_data.loc[sampled_data["Gender"]=="M"]["Purchase"])
female_mean = np.mean(sampled_data.loc[sampled_data["Gender"]=="F"]["Purchase"])
male_std = np.std(sampled_data.loc[sampled_data["Gender"]=="M"]["Purchase"])
female_std = np.std(sampled_data.loc[sampled_data["Gender"]=="F"]["Purchase"])


confidence_interval = [90,95,99]
for intervals in confidence_interval:
  print("-----",intervals,"%-------\n", "Male confidence interval:", calc_CI(male_mean, male_std, 1000, intervals), "\n Femal
```

```
 ----- 90 %-------
  Male confidence interval: (9125.58195554966, 9649.28644445034)
  Female confidence interval: (8492.580271941353, 8991.467728058646)
 ----- 95 %-------
  Male confidence interval: (9075.41800598471, 9699.45039401529)
  Female confidence interval: (8444.793465225628, 9039.25453477437)
 ----- 99 %-------
  Male confidence interval: (8977.375412151921, 9797.492987848078)
  Female confidence interval: (8351.396862246238, 9132.65113775376)
```

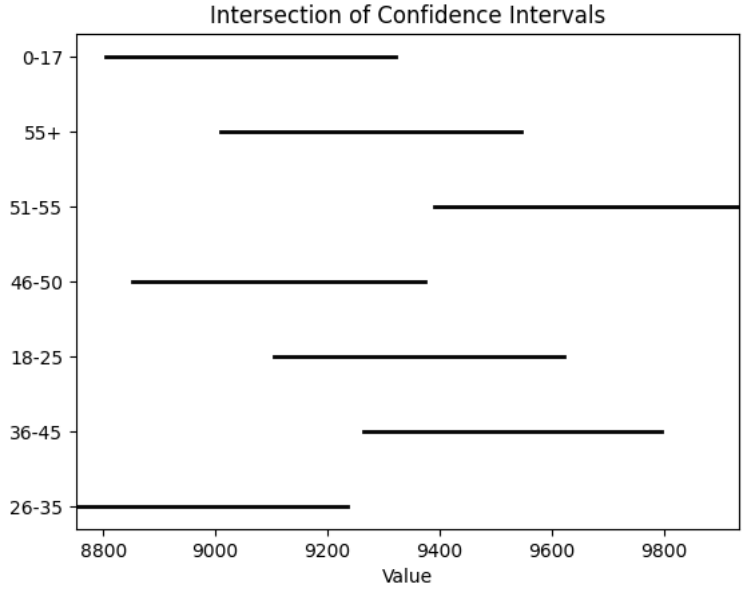
```
# Define the sample sizes for each gender
sample_size = 5000  # Adjust as per your desired sample size

# Stratified sampling
single_data = df[df['Marital_Status'] == 0].sample(n=sample_size, random_state=4)
married_data = df[df['Marital_Status'] == 1].sample(n=sample_size, random_state=4)

# Combine the sampled data
sampled_data = pd.concat([single_data, married_data])

single_mean = np.mean(sampled_data.loc[sampled_data["Marital_Status"]==0]["Purchase"])
married_mean = np.mean(sampled_data.loc[sampled_data["Marital_Status"]==1]["Purchase"])
single_std = np.std(sampled_data.loc[sampled_data["Marital_Status"]==0]["Purchase"])
married_std = np.std(sampled_data.loc[sampled_data["Marital_Status"]==1]["Purchase"])

confidence_interval = [90,95,99]
for intervals in confidence_interval:
  print("-----",intervals,"%-------\n", "Single confidence interval:", calc_CI(single_mean, single_std, 1000, intervals), "\n
```

```
 ----- 90 %-------
  Single confidence interval: (9004.40605056871, 9526.398749431291)
  Married confidence interval: (9126.084208461514, 9654.566191538484)
 ----- 95 %-------
  Single confidence interval: (8954.406067803659, 9576.398732196343)
  Married confidence interval: (9075.462638268928, 9705.187761731071)
 ----- 99 %-------
  Single confidence interval: (8856.683937780117, 9674.120862219885)
  Married confidence interval: (8976.525650878339, 9804.12474912166)
```

```
age_bins = df["Age"].value_counts().reset_index()["index"]


sample_size = 1000
age_calculations={}
for age_group in age_bins:
  age_calculations[age_group] = {}
  age_calculations[age_group]["sample"] = df[df['Age'] == age_group].sample(n=sample_size, random_state=4)
  age_calculations[age_group]["mean"] = np.mean(age_calculations[age_group]["sample"]["Purchase"])
  age_calculations[age_group]["std"] = np.std(age_calculations[age_group]["sample"]["Purchase"])


def plot_visualization(age_groups, intervals):
  # Plotting the intersection graph
  fig, ax = plt.subplots()

  # Plotting the intervals
  for i, interval in enumerate(intervals):
      ax.plot(interval, [i, i], 'k-', lw=2)

  # Labeling the age groups
  ax.set_yticks(range(len(age_groups)))
  ax.set_yticklabels(age_groups)

  # Setting the x-axis limits
  min_val = min(interval[0] for interval in intervals)
```

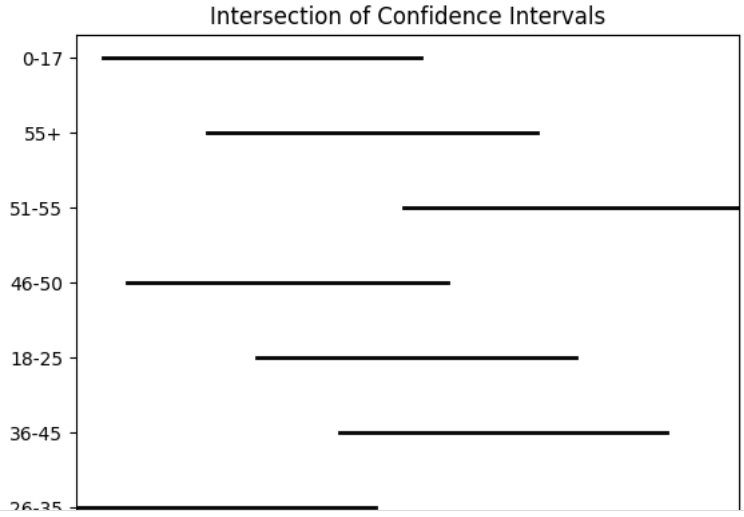
```
    max_val = max(interval[1] for interval in intervals)
    ax.set_xlim(min_val, max_val)

    # Adding labels and title
    ax.set_xlabel('Value')
    ax.set_title('Intersection of Confidence Intervals')

    # Display the plot
    plt.show()


confidence_interval = [90,95,99]
for intervals in confidence_interval:
  age_grp_val = []
  confidence_inter_val = []
  print("\n-----",intervals,"%-------")
  for age_group in age_bins:
    x1, x2 = calc_CI(age_calculations[age_group]["mean"], age_calculations[age_group]["std"], 1000, intervals)
    age_grp_val.append(age_group)
    confidence_inter_val.append((x1, x2))
    print(age_group, "confidence interval:", x1, x2)
  plot_visualization(age_grp_val, confidence_inter_val)
```

```
----- 90 %-------
26-35 confidence interval: 8751.521389002177 9236.452610997821
36-45 confidence interval: 9264.992229647985 9795.049770352016
18-25 confidence interval: 9104.379141438014 9621.422858561986
46-50 confidence interval: 8852.999383766888 9374.08461623311
51-55 confidence interval: 9390.18689980946 9933.16710019054
55+ confidence interval: 9009.171814696234 9544.888185303767
0-17 confidence interval: 8805.334479961088 9320.763520038914
```




```
----- 95 %-------
26-35 confidence interval: 8705.071404558234 9282.902595441765
36-45 confidence interval: 9214.219741915038 9845.822258084963
18-25 confidence interval: 9054.853205536403 9670.948794463597
46-50 confidence interval: 8803.086324256272 9423.997675743727
```

```
0-17 confidence interval: 8755.963208721749 9370.134791278253
```



✓ 0s completed at 11:38 PM