SCALER

Walmart - Business Case
Central Limit Theorem
and
Confidence Interval

Submitted by :

Harsha Srinivas, Tanna
harshasrinivas.tanna@gmail.com
Scaler DSML - Morning TTS Feb 2023

Submitted on September 17, 2023

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

```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/293/original/walmart_data.csv?1641285094'
```

```
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.shape
```

```
(550068, 10)
```

```
df.ndim
```

```
2
```

```
df.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

```
df.head()
```

	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.isnull().sum()
# checking null values
# we can see there are no null values
```

```
User_ID                       0
Product_ID                    0
Gender                        0
Age                           0
Occupation                    0
City_Category                 0
Stay_In_Current_City_Years    0
```

```
Marital_Status                0
Product_Category              0
Purchase                      0
dtype: int64
```

```
df.columns
```

```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

```
df.dtypes
```

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

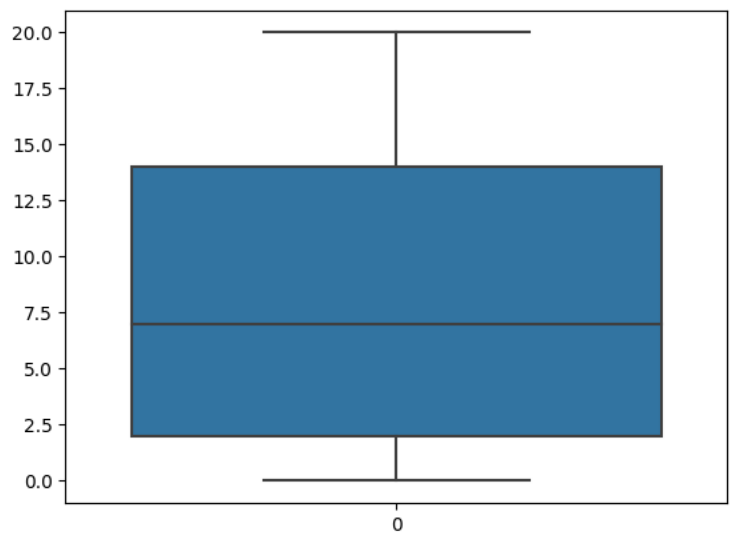
```
#convert occupation to categorical
#convert Stay_In_Current_City_Years to numerical
#convert marital status to categorical
```

```
df[df.duplicated()]
# checking if there are any duplicate rows
# I see none
```

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_

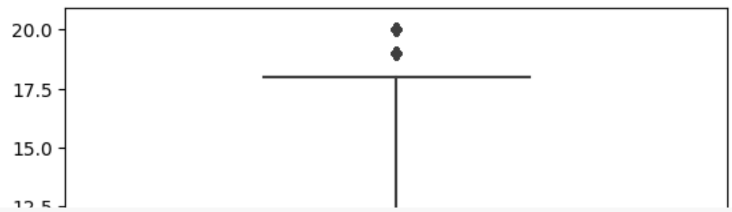
```
sns.boxplot(df['Occupation'])
# no outliers
```

```
<Axes: >
```



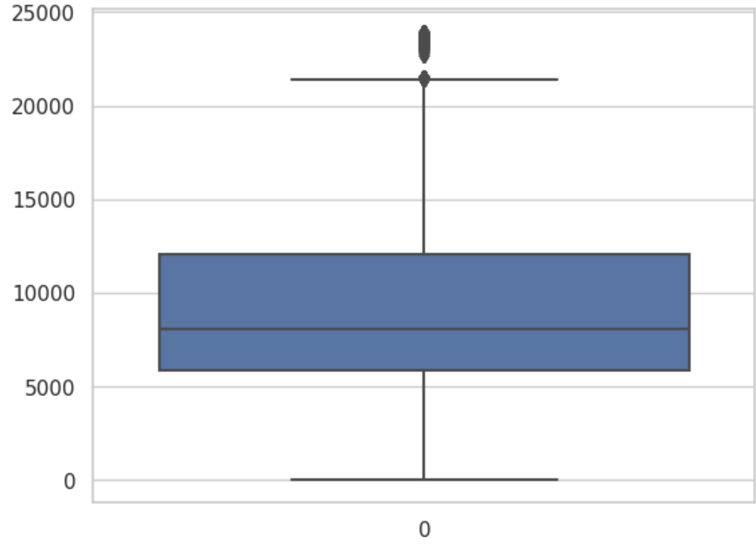

```
sns.boxplot(df['Product_Category'])
# there are some outliers here but this is not a numerical category so I do not remove the outliers
```

```
<Axes: >
```




```
sns.boxplot(df['Purchase'])
# there are some outliers which can impact our analysis.
# so we remove these outliers using Inter Quartile Range(IQR) method
```

```
<Axes: >
```




```
df['Stay_In_Current_City_Years'].value_counts()
```

```
1     193821
2     101838
3      95285
4+     84726
0      74398
Name: Stay_In_Current_City_Years, dtype: int64
```

```
# Removing the outliers
# use Boxplot,IQR to detect outliers
def remove_outliers(df,series_name):
  q1 = np.percentile(df[series_name],25)
  q3 = np.percentile(df[series_name],75)
  iqr = q3-q1

  lower_bound = q1 - 1.5 * iqr
  upper_bound = q3 + 1.5 * iqr

  avg = df['Purchase'].mean()

  df.loc[df['Purchase'] < lower_bound] = avg
  df.loc[df['Purchase'] > upper_bound] = avg
  return df
# Outliers are present in Purchase
# so we remove outliers and get the remaining data

remove_outliers(df, 'Purchase')
df
```

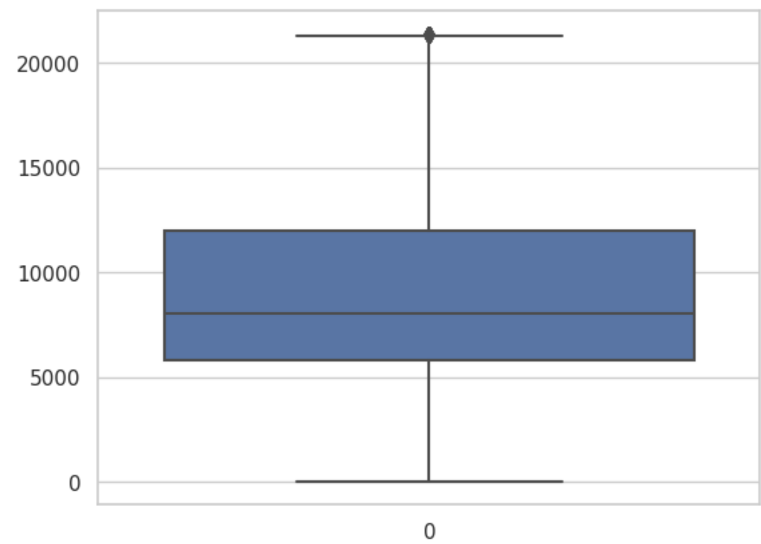
	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Cu
0	1000001.0	P00069042	F	0-17	10.0	A	
1	1000001.0	P00248942	F	0-17	10.0	A	
2	1000001.0	P00087842	F	0-	10.0	A	
550065	1006036.0	P00375436	F	[illegible]	15.0	B	

```
df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current
0	1000001.0	P00069042	F	0-17	10.0	A	
1	1000001.0	P00248942	F	0-17	10.0	A	
2	1000001.0	P00087842	F	0-17	10.0	A	

```
sns.boxplot(df['Purchase'])
# we can see there are no outliers now as we removed them
```

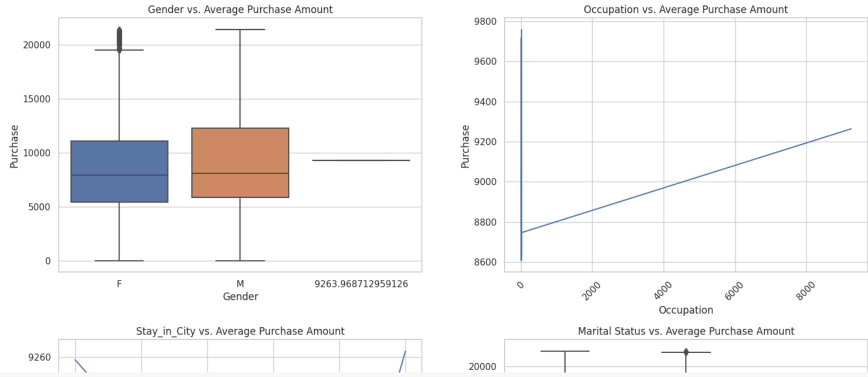
<Axes: >




```
# Heat Map
sns.heatmap(df.corr(),annot = True)
# There is not much of correlation between the columns of the dataframe
```

```
<ipython-input-234-b081d7116fc3>:2: FutureWarning: The default value of numeri
  sns.heatmap(df.corr(),annot = True)
<Axes: >
```


```
# Visualisations
# Setting the style for the plots (optional, for aesthetics)
sns.set(style="whitegrid")

# Creating subplots for each combination
plt.figure(figsize=(15, 12))

# Gender vs. Purchase Amount
plt.subplot(2, 2, 1)
sns.boxplot(x='Gender', y='Purchase', data=df)
plt.title('Gender vs. Average Purchase Amount')

# Occupation vs. Purchase Amount
plt.subplot(2, 2, 2)
sns.lineplot(x='Occupation', y='Purchase', data=df, estimator='mean', errorbar=None)
plt.title('Occupation vs. Average Purchase Amount')
plt.xticks(rotation=45)

# Stay_in_City vs. Purchase Amount
plt.subplot(2, 2, 3)
df['Stay_In_Current_City_Years'] = df['Stay_In_Current_City_Years'].astype(str)
sns.lineplot(x='Stay_In_Current_City_Years', y='Purchase', data=df, estimator='mean', errorbar=None)
plt.title('Stay_in_City vs. Average Purchase Amount')
plt.xticks(rotation=45)

# Marital Status vs. Purchase Amount
plt.subplot(2, 2, 4)
sns.boxplot(x='Marital_Status', y='Purchase', data=df)
plt.title('Marital Status vs. Average Purchase Amount')
plt.xticks([0, 1], ['Single', 'Married'])  # Customize x-axis labels

# Adjusting layout and display the plots
plt.tight_layout()
plt.show()
```

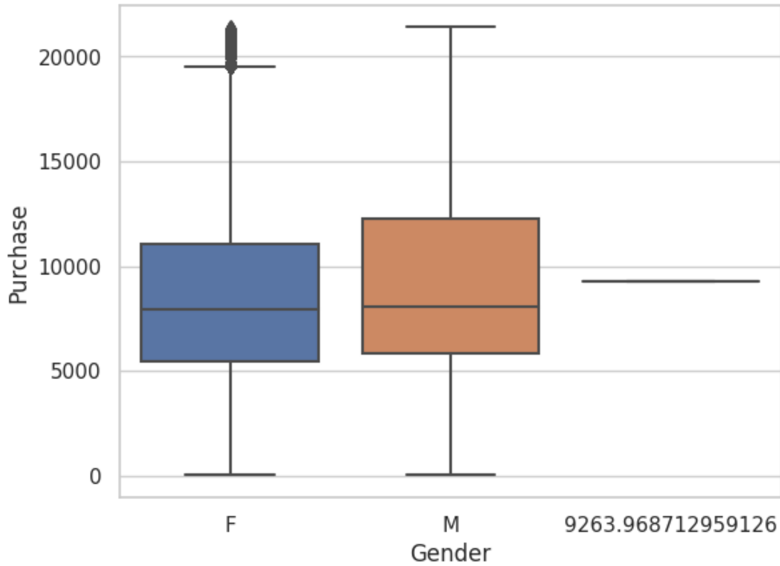



```
# Average spend by male vs female for total population
avg_spend_by_total_population = df['Purchase'].mean()
avg_spend_by_male_population = df[df['Gender'] == 'M']['Purchase'].mean()
avg_spend_by_female_population = df[df['Gender'] == 'F']['Purchase'].mean()
print("Average spend by total population : ",avg_spend_by_total_population)
print("Average spend by males : ",avg_spend_by_male_population)
print("Average spend by females : ",avg_spend_by_female_population)
# we can say for the total population the average spend by males is more than the average spend by females
```

```
Average spend by total population :  9195.959790870567
Average spend by males :  9367.724354697444
Average spend by females :  8671.049038603756
```

```
sns.boxplot(x = 'Gender',y = 'Purchase',data=df)
# from the plot we can see the median purchase amount is almost the same but the median purchase amount is slighly greater for
```

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




```
std_dev_population = df['Purchase'].std()
print("Standard Deviation of Purchase for total Population : ",std_dev_population)
```

```
Standard Deviation of Purchase for total Population :  4926.842634292559
```

```
df.std(axis = 0)
```

```
<ipython-input-269-423c962a8777>:1: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a
  df.std(axis = 0)
User_ID             69179.155442
Occupation            644.165635
Marital_Status        644.666398
Product_Category      644.332290
Purchase             4926.842634
dtype: float64
```

```
# now we take sample sizes of 10,100,1000,10000,100000,250000,500000
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.05
r = 0.95
n = 10
```

```
while n < df.shape[0] :
  sample_size = n
  mu = avg_spend_by_total_population
  std_sample = std_dev_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  print("90 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
90 % CL range for sample size of  10  :  6633.270537326727  -  11758.649044414406
90 % CL range for sample size of  100  :  8385.56629322704  -  10006.353288514094
90 % CL range for sample size of  1000  :  8939.690865516184  -  9452.22871622495
90 % CL range for sample size of  10000  :  9114.920441106215  -  9276.99914063492
90 % CL range for sample size of  100000  :  9170.33289833513  -  9221.586683406005
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.025
r = 0.975
n = 10
while n < df.shape[0] :
  sample_size = n
  mu = avg_spend_by_total_population
  std_sample = std_dev_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  print("95 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
95 % CL range for sample size of  10  :  6142.327201189719  -  12249.592380551414
95 % CL range for sample size of  100  :  8230.316378799582  -  10161.603202941553
95 % CL range for sample size of  1000  :  8890.596531902482  -  9501.323049838653
95 % CL range for sample size of  10000  :  9099.395449663469  -  9292.524132077666
95 % CL range for sample size of  100000  :  9165.42346497376  -  9226.496116767375
```

```
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
n = 10
while n < df.shape[0] :
  sample_size = n
  mu = avg_spend_by_total_population
  std_sample = std_dev_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  print("99 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
99 % CL range for sample size of  10  :  5182.806299880437  -  13209.113281860697
99 % CL range for sample size of  100  :  7926.889227732084  -  10465.03035400905
99 % CL range for sample size of  1000  :  8794.644441771554  -  9597.27513996958
99 % CL range for sample size of  10000  :  9069.052734556719  -  9322.866847184416
99 % CL range for sample size of  100000  :  9155.828255960667  -  9236.091325780468
```

```
avg_spend_by_male_sample = avg_spend_by_male_population
std_dev_avg_spend_by_male_total_population = df[df['Gender'] == 'M']['Purchase'].std()
print("Standard Deviation for purchase by males : ",std_dev_avg_spend_by_male_total_population)
```

```
Standard Deviation for purchase by males :  5009.234087946682
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
```

```
l = 0.05
r = 0.95
n = 10
CI_90_male_sample = []
while n < df[df['Gender'] == 'M'].shape[0] :
  sample_size = n
  mu = avg_spend_by_male_sample
  std_sample = std_dev_avg_spend_by_male_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_90_male_sample.append((l_value,r_value))
  print("90 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
90 % CL range for sample size of  10  :  6762.179319360688  -  11973.269390034198
90 % CL range for sample size of  100  :  8543.778668916639  -  10191.67004047825
90 % CL range for sample size of  1000  :  9107.169851163768  -  9628.27885823112
90 % CL range for sample size of  10000  :  9285.329786119364  -  9450.118923275524
90 % CL range for sample size of  100000  :  9341.668904344077  -  9393.779805050812
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.025
r = 0.975
n = 10
CI_95_male_sample = []
while n < df[df['Gender'] == 'M'].shape[0] :
  sample_size = n
  mu = avg_spend_by_male_sample
  std_sample = std_dev_avg_spend_by_male_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_95_male_sample.append((l_value,r_value))
  print("95 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
95 % CL range for sample size of  10  :  6263.0259513374185  -  12472.42275805747
95 % CL range for sample size of  100  :  8385.93251444686  -  10349.516194948028
95 % CL range for sample size of  1000  :  9057.25451436144  -  9678.194195033446
95 % CL range for sample size of  10000  :  9269.545170672385  -  9465.903538722503
95 % CL range for sample size of  100000  :  9336.677370663843  -  9398.771338731045
```

```
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
n = 10
CI_99_male_sample = []
while n < df[df['Gender'] == 'M'].shape[0] :
  sample_size = n
  mu = avg_spend_by_male_sample
  std_sample = std_dev_avg_spend_by_male_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_99_male_sample.append((l_value,r_value))
  print("99 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
99 % CL range for sample size of  10  :  5287.45900842735  -  13447.989700967537
99 % CL range for sample size of  100  :  8077.431159490532  -  10658.017549904356
99 % CL range for sample size of  1000  :  8959.697820070434  -  9775.750889324454
99 % CL range for sample size of  10000  :  9238.695035176754  -  9496.753674218135
99 % CL range for sample size of  100000  :  9326.921701234744  -  9408.527008160145
```

```
avg_spend_by_female_sample = avg_spend_by_female_population
std_dev_avg_spend_by_female_total_population = df[df['Gender'] == 'F']['Purchase'].std()
print("Standard Deviation for purchase by males : ",std_dev_avg_spend_by_female_total_population)
```

```
Standard Deviation for purchase by males :  4679.058483084379
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.05
r = 0.95
n = 10
CI_90_female_sample = []
while n < df[df['Gender'] == 'F'].shape[0] :
  sample_size = n
  mu = avg_spend_by_female_sample
  std_sample = std_dev_avg_spend_by_female_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_90_female_sample.append((l_value,r_value))
  print("90 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
90 % CL range for sample size of  10  :  6237.244311851964  -  11104.853765355547
90 % CL range for sample size of  100  :  7901.412406941817  -  9440.685670265695
90 % CL range for sample size of  1000  :  8427.668565928578  -  8914.429511278935
90 % CL range for sample size of  10000  :  8594.085375437562  -  8748.01270176995
90 % CL range for sample size of  100000  :  8646.710991336238  -  8695.387085871274
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.025
r = 0.975
n = 10
CI_95_female_sample = []

while n < df[df['Gender'] == 'F'].shape[0] :
  sample_size = n
  mu = avg_spend_by_female_sample
  std_sample = std_dev_avg_spend_by_female_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_95_female_sample.append((l_value,r_value))
  print("95 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

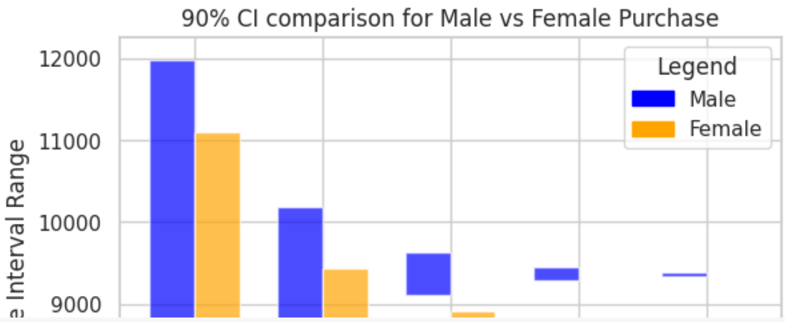
```
95 % CL range for sample size of  10  :  5770.99183492554  -  11571.106242281972
95 % CL range for sample size of  100  :  7753.9704277635565  -  9588.127649443957
95 % CL range for sample size of  1000  :  8381.043318235934  -  8961.054758971579
95 % CL range for sample size of  10000  :  8579.341177519736  -  8762.756899687776
95 % CL range for sample size of  100000  :  8642.048466566974  -  8700.049610640539
```

```
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
n = 10
CI_99_female_sample = []

while n < df[df['Gender'] == 'F'].shape[0] :
  sample_size = n
  mu = avg_spend_by_female_sample
  std_sample = std_dev_avg_spend_by_female_total_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
```

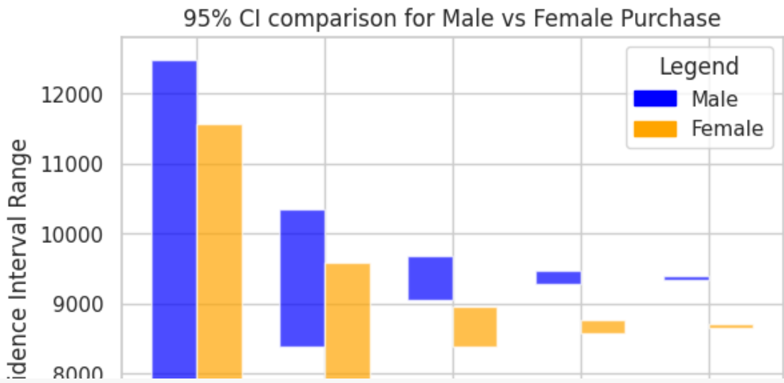
```
  #print("Left value : ",l_value)
  CI_99_female_sample.append((l_value,r_value))
  print("99 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
99 % CL range for sample size of  10  :  4859.727817333747  -  12482.370259873764
99 % CL range for sample size of  100  :  7465.803443228975  -  9876.294633978538
99 % CL range for sample size of  1000  :  8289.916916476755  -  9052.181160730757
99 % CL range for sample size of  10000  :  8550.524479066278  -  8791.573598141234
99 % CL range for sample size of  100000  :  8632.935826391056  -  8709.162250816456
```

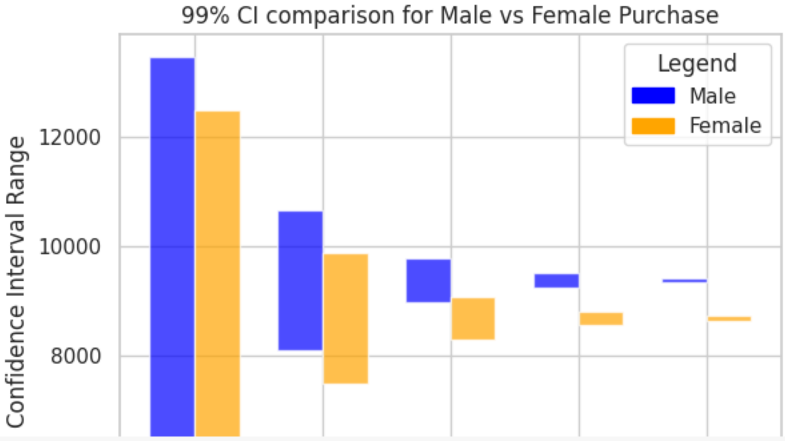
```
# comparing the 90 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_90_male_sample
pair2 = CI_90_female_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
  pair1_lower, pair1_upper = pair1[i]
  pair2_lower, pair2_upper = pair2[i]

  # Create bars for male list
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
  ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('90% CI comparison for Male vs Female Purchase')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Male': 'blue', 'Female': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# when the sample size is small we can see partial overlap of the Confidence Intervals indicating there might be some
# some statistical similarity between the purchase patterns of males and females
# and this overlap wanes out as the sample size increases
```




```
# comparing the 95 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_95_male_sample
pair2 = CI_95_female_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
  pair1_lower, pair1_upper = pair1[i]
  pair2_lower, pair2_upper = pair2[i]

  # Create bars for male list
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
  ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('95% CI comparison for Male vs Female Purchase')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Male': 'blue', 'Female': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# when the sample size is small we can see partial overlap of the Confidence Intervals indicating there might be some
# some statistical similarity between the purchase patterns of males and females
# and this overlap wanes out as the sample size increases
```




```
# comparing the 99 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_99_male_sample
pair2 = CI_99_female_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
  pair1_lower, pair1_upper = pair1[i]
  pair2_lower, pair2_upper = pair2[i]

  # Create bars for male list
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
  ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('99% CI comparison for Male vs Female Purchase')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Male': 'blue', 'Female': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# when the sample size is small we can see partial overlap of the Confidence Intervals indicating there might be some
# some statistical similarity between the purchase patterns of males and females
# and this overlap wanes out as the sample size increases
```




```
df[df['Marital_Status'] == 1].shape[0] + df[df['Marital_Status'] == 0].shape[0]
```

```
547391
```

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

```
0.000000       323242
1.000000       224149
9263.968713      2677
Name: Marital_Status, dtype: int64
```

```
avg_spend_by_unmarried = df[df['Marital_Status'] == 0]['Purchase'].mean()
std_dev_unmarried_population = df[df['Marital_Status'] == 0]['Purchase'].std()
print("Avg spend by unmarried customers : ",avg_spend_by_unmarried)
print("Standard deviation of purchase by unmarried customers : ",std_dev_unmarried_population)
```

```
Avg spend by unmarried customers :  9201.581848893398
Standard deviation of purchase by unmarried customers :  4948.327397459
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.05
r = 0.95
n = 10
CI_90_unmarried_sample = []
while n < df[df['Marital_Status'] == 0].shape[0] :
  sample_size = n
  mu = avg_spend_by_unmarried
  std_sample = std_dev_unmarried_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_90_unmarried_sample.append((l_value,r_value))
  print("90 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
90 % CL range for sample size of  10  :  6627.717330424646  -  11775.44636736215
90 % CL range for sample size of  100  :  8387.654422188021  -  10015.509275598775
90 % CL range for sample size of  1000  :  8944.195397046524  -  9458.968300740273
90 % CL range for sample size of  10000  :  9120.18910622286  -  9282.974591563936
90 % CL range for sample size of  100000  :  9175.843203708711  -  9227.320494078085
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.025
r = 0.975
n = 10
CI_95_unmarried_sample = []
while n < df[df['Marital_Status'] == 0].shape[0] :
  sample_size = n
  mu = avg_spend_by_unmarried
  std_sample = std_dev_unmarried_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
```

```
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_95_unmarried_sample.append((l_value,r_value))
  print("95 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
95 % CL range for sample size of  10  :  6134.633109731749  -  12268.530588055046
95 % CL range for sample size of  100  :  8231.727500620153  -  10171.436197166644
95 % CL range for sample size of  1000  :  8894.886974977233  -  9508.276722809564
95 % CL range for sample size of  10000  :  9104.596414066074  -  9298.567283720722
95 % CL range for sample size of  100000  :  9170.912361501782  -  9232.251336285015
```

```
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
n = 10
CI_99_unmarried_sample = []

while n < df[df['Marital_Status'] == 0].shape[0] :
  sample_size = n
  mu = avg_spend_by_unmarried
  std_sample = std_dev_unmarried_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_99_unmarried_sample.append((l_value,r_value))
  print("99 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
99 % CL range for sample size of  10  :  5170.927971001448  -  13232.235726785348
99 % CL range for sample size of  100  :  7926.977177500522  -  10476.186520286274
99 % CL range for sample size of  1000  :  8798.516461104204  -  9604.647236682593
99 % CL range for sample size of  10000  :  9074.12138175411  -  9329.042316032686
99 % CL range for sample size of  100000  :  9161.275310114479  -  9241.888387672318
```

```
avg_spend_by_married = df[df['Marital_Status'] == 1]['Purchase'].mean()
std_dev_married_population = df[df['Marital_Status'] == 1]['Purchase'].std()
print("Avg spend by married customers : ",avg_spend_by_married)
print("Standard deviation of purchase by married customers : ",std_dev_married_population)
```

```
Avg spend by married customers :  9187.040076020861
Standard deviation of purchase by married customers :  4925.2052318513715
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.05
r = 0.95
n = 10
CI_90_married_sample = []

while n < df[df['Marital_Status'] == 1].shape[0] :
  sample_size = n
  mu = avg_spend_by_married
  std_sample = std_dev_married_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_90_married_sample.append((l_value,r_value))
  print("90 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
90 % CL range for sample size of  10  :  6625.202514717109  -  11748.877637324613
90 % CL range for sample size of  100  :  8376.915907111752  -  9997.16424492997
90 % CL range for sample size of  1000  :  8930.856319890487  -  9443.223832151236
90 % CL range for sample size of  10000  :  9106.02765912995  -  9268.052492911773
90 % CL range for sample size of  100000  :  9161.421700407824  -  9212.658451633899
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
```

```
l = 0.025
r = 0.975
n = 10
CI_95_married_sample = []
while n < df[df['Marital_Status'] == 1].shape[0] :
  sample_size = n
  mu = avg_spend_by_married
  std_sample = std_dev_married_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_95_married_sample.append((l_value,r_value))
  print("95 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

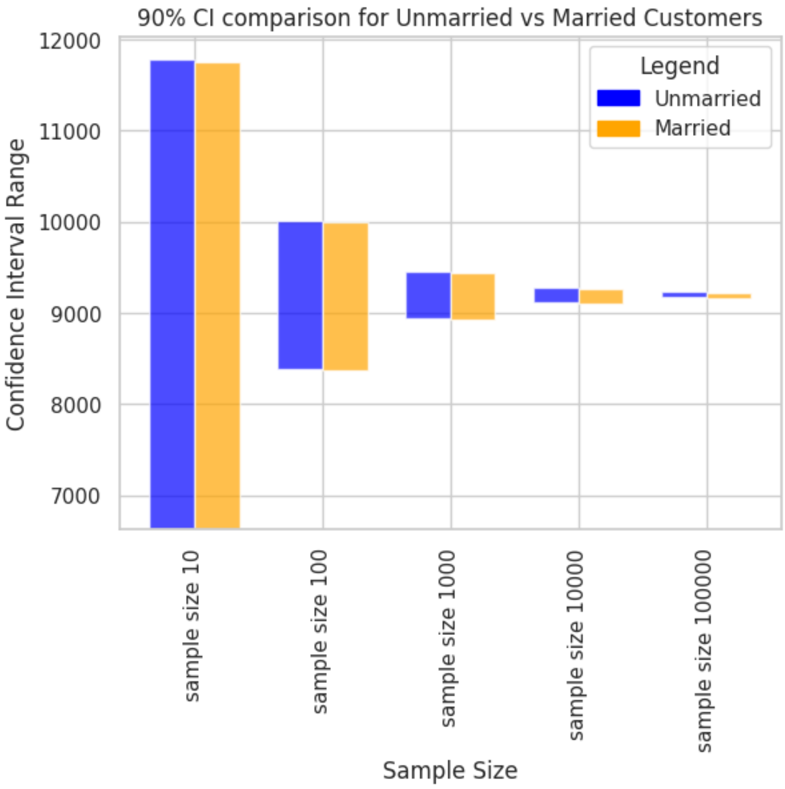
```
95 % CL range for sample size of  10  :  6134.422340238944  -  12239.657811802777
95 % CL range for sample size of  100  :  8221.717588931167  -  10152.362563110555
95 % CL range for sample size of  1000  :  8881.77830244267  -  9492.301849599053
95 % CL range for sample size of  10000  :  9090.507827311892  -  9283.57232472983
95 % CL range for sample size of  100000  :  9156.513898663043  -  9217.56625337868
```

```
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
n = 10
CI_99_married_sample = []
while n < df[df['Marital_Status'] == 1].shape[0] :
  sample_size = n
  mu = avg_spend_by_married
  std_sample = std_dev_married_population/np.sqrt(n)
  #print("mean of sample : ",mu)
  #print("std of sample : ",std_sample)
  z_of_r_value = norm.ppf(r)
  z_of_l_value = norm.ppf(l)
  r_value = mu + z_of_r_value * std_sample
  l_value = mu + z_of_l_value * std_sample
  #print("Right value : ",r_value)
  #print("Left value : ",l_value)
  CI_99_married_sample.append((l_value,r_value))
  print("99 % CL range for sample size of ",n," : ",l_value," - ",r_value)
  n = n * 10
```

```
99 % CL range for sample size of  10  :  5175.220329136391  -  13198.85982290533
99 % CL range for sample size of  100  :  7918.391279801349  -  10455.688872240373
99 % CL range for sample size of  1000  :  8785.858101332415  -  9588.222050709308
99 % CL range for sample size of  10000  :  9060.17519639891  -  9313.904955642813
99 % CL range for sample size of  100000  :  9146.921878552017  -  9227.158273489706
```

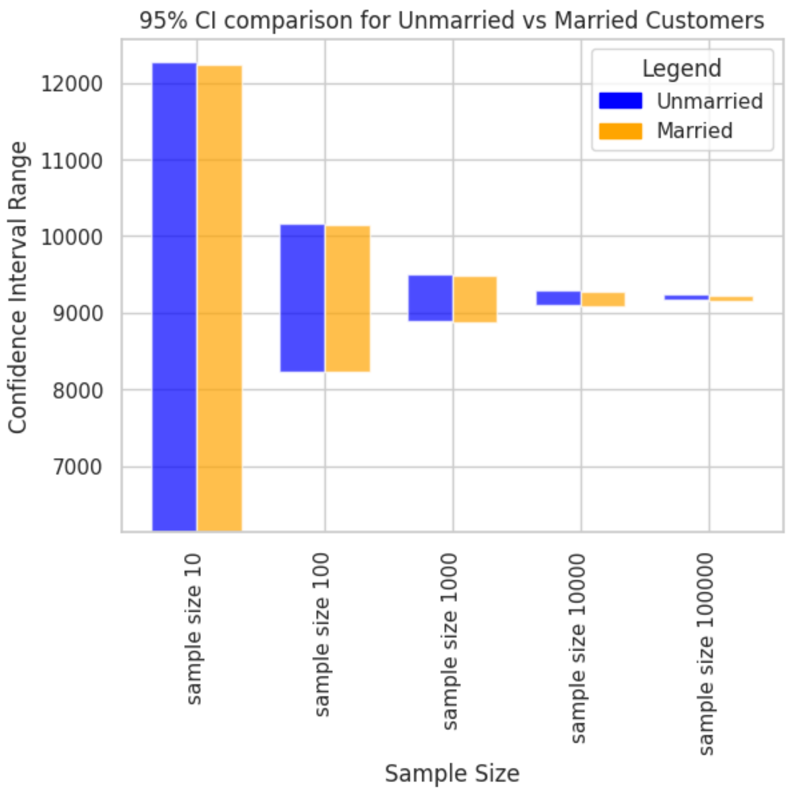
```
# comparing the 90 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_90_unmarried_sample
pair2 = CI_90_married_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
  pair1_lower, pair1_upper = pair1[i]
  pair2_lower, pair2_upper = pair2[i]

  # Create bars for male list
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
  ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
```

```
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('90% CI comparison for Unmarried vs Married Customers')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Unmarried': 'blue', 'Married': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# Irrespective of the sample size we can see there is complete overlap of the confidence intervals for the purchase patterns c
# married and unmarried customers
# so, we can say that there is a statistical similarity between the purchase patterns of married and unmarried customers
```



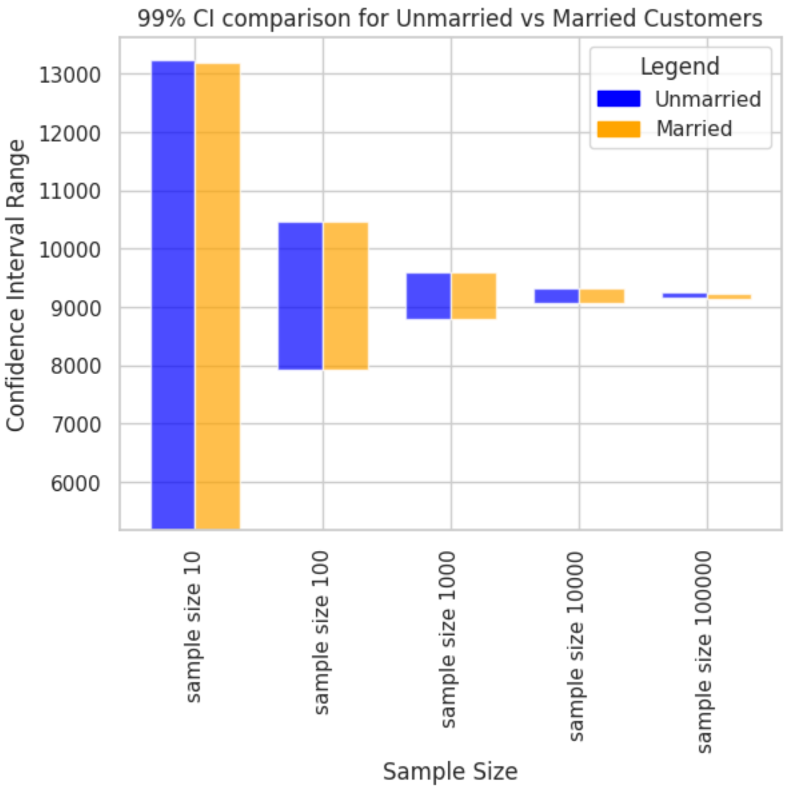

```
# comparing the 95 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_95_unmarried_sample
pair2 = CI_95_married_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
  pair1_lower, pair1_upper = pair1[i]
  pair2_lower, pair2_upper = pair2[i]

  # Create bars for male list
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
```

```
    ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('95% CI comparison for Unmarried vs Married Customers')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Unmarried': 'blue', 'Married': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# Irrespective of the sample size we can see there is complete overlap of the confidence intervals for the purchase patterns c
# married and unmarried customers
# so, we can say that there is a statistical similarity between the purchase patterns of married and unmarried customers
```




```
# comparing the 99 % confidence intervals for married and unmmaried customers for the sample sizes n = 10,100,1000,10000,10000
pair1 = CI_99_unmarried_sample
pair2 = CI_99_married_sample

# Number of pairs
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each bar
width = 0.35

# Iterate through the pairs and create bars for each pair
for i in range(num_pairs):
    pair1_lower, pair1_upper = pair1[i]
    pair2_lower, pair2_upper = pair2[i]

    # Create bars for male list
```

```
  ax.bar(i - width/2, pair1_upper - pair1_lower, bottom=pair1_lower, width=width, label=f'Male', align='center', color='blue',

  # Create bars for female list
  ax.bar(i + width/2, pair2_upper - pair2_lower, bottom=pair2_lower, width=width, label=f'Female', align='center', color='oran

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('99% CI comparison for Unmarried vs Married Customers')

# Set x-axis ticks and labels
ax.set_xticks(indices)
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {'Unmarried': 'blue', 'Married': 'orange'}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend')

# Show the plot
plt.xticks(rotation  = 90)
plt.grid(True)
plt.show()

# Observations
# Irrespective of the sample size we can see there is complete overlap of the confidence intervals for the purchase patterns c
# married and unmarried customers
# so, we can say that there is a statistical similarity between the purchase patterns of married and unmarried customers
```




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

```
26-35                 218661
36-45                 109409
18-25                  99334
46-50                  45442
51-55                  38191
55+                    21322
0-17                   15032
9263.968712959126       2677
Name: Age, dtype: int64
```

```
age_group_list = ['0-17','18-25','26-35','36-45','46-50','51-55','55+']
mean_of_ages_list = []
std_dev_of_ages_list = []

for i in range(0,len(age_group_list)):
  mean_of_ages_list.append(df[df['Age'] == age_group_list[i]]['Purchase'].mean())
  std_dev_of_ages_list.append(df[df['Age'] == age_group_list[i]]['Purchase'].std())
```

```
print("Mean of different age groups : ",mean_of_ages_list)
print("Standard Deviation of different age groups : ",std_dev_of_ages_list)
```

```
Mean of different age groups :  [8867.447046301224, 9124.031731330662, 9193.469923763269, 9254.202213711851, 9128.985079
Standard Deviation of different age groups :  [5030.052845920101, 4978.831061893425, 4937.410901301763, 4927.74443326426
```

```
# for 90 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.05
r = 0.95

age_group_list = ['0-17','18-25','26-35','36-45','46-50','51-55','55+']
CI_90_age_0_17 = []
CI_90_age_18_25 = []
CI_90_age_26_35 = []
CI_90_age_36_45 = []
CI_90_age_46_50 = []
CI_90_age_51_55 = []
CI_90_age_55_plus = []

for i in range(0,len(age_group_list)):
  n = 10
  while n < df[df['Age'] == age_group_list[i]].shape[0] :
    sample_size = n
    mu = mean_of_ages_list[i]
    std_sample = std_dev_of_ages_list[i]/np.sqrt(n)
    #print("mean of sample : ",mu)
    #print("std of sample : ",std_sample)
    z_of_r_value = norm.ppf(r)
    z_of_l_value = norm.ppf(l)
    r_value = mu + z_of_r_value * std_sample
    l_value = mu + z_of_l_value * std_sample

    #print("Right value : ",r_value)
    #print("Left value : ",l_value)
    # Split the string based on '-' or '+'
    #print("Iteration :",age_group_list[i])

    if '-' in age_group_list[i]:
      parts = age_group_list[i].split('-')
    elif '+' in  age_group_list[i]:
      #print("check point 2")
      parts = age_group_list[i].split('+')
    #print("parts list print :",parts)

    #next if else block
    if parts[0] == '0':
      CI_90_age_0_17.append((l_value,r_value))
    elif parts[0] == '18':
      CI_90_age_18_25.append((l_value,r_value))
    elif parts[0] == '26':
      CI_90_age_26_35.append((l_value,r_value))
    elif parts[0] == '36':
      CI_90_age_36_45.append((l_value,r_value))
    elif parts[0] == '46':
      CI_90_age_46_50.append((l_value,r_value))
    elif parts[0] == '51':
      CI_90_age_51_55.append((l_value,r_value))
    elif parts[0] == '55':
      CI_90_age_55_plus.append((l_value,r_value))
      #print("CP 3")
    print("90 % CI range for the age group ",age_group_list[i],"sample size of ",n," : ",l_value," - ",r_value)
    n = n * 10
```

```
90 % CI range for the age group  0-17 sample size of  10  :  6251.073167567023  -  11483.820925035423
90 % CI range for the age group  0-17 sample size of  100  :  8040.076979564298  -  9694.817113038149
90 % CI range for the age group  0-17 sample size of  1000  :  8605.809658427805  -  9129.084434174643
90 % CI range for the age group  0-17 sample size of  10000  :  8784.710039627531  -  8950.184052974917
90 % CI range for the age group  18-25 sample size of  10  :  6534.3007809820465  -  11713.762681679276
90 % CI range for the age group  18-25 sample size of  100  :  8305.086898317257  -  9942.976564344068
90 % CI range for the age group  18-25 sample size of  1000  :  8865.0586362958  -  9383.004826365524
90 % CI range for the age group  18-25 sample size of  10000  :  9042.137248029321  -  9205.926214632003
90 % CI range for the age group  26-35 sample size of  10  :  6625.283603171953  -  11761.656244354584
90 % CI range for the age group  26-35 sample size of  100  :  8381.338100887673  -  10005.601746638862
90 % CI range for the age group  26-35 sample size of  1000  :  8936.651291704136  -  9450.2885558224
90 % CI range for the age group  26-35 sample size of  10000  :  9112.256741475709  -  9274.683106050828
90 % CI range for the age group  26-35 sample size of  100000  :  9167.788060557355  -  9219.151786969182
90 % CI range for the age group  36-45 sample size of  10  :  6691.043890886689  -  11817.360536537011
90 % CI range for the age group  36-45 sample size of  100  :  8443.660383337385  -  10064.744044086317
90 % CI range for the age group  36-45 sample size of  1000  :  8997.886381429335  -  9510.518045994368
```

```
90 % CI range for the age group  36-45 sample size of  10000  :  9173.148030674405  -  9335.256396749297
90 % CI range for the age group  36-45 sample size of  100000  :  9228.5706304836  -  9279.833796940102
90 % CI range for the age group  46-50 sample size of  10  :  6597.207560063129  -  11660.762599700967
90 % CI range for the age group  46-50 sample size of  100  :  8328.36673073806  -  9929.603429026036
90 % CI range for the age group  46-50 sample size of  1000  :  8875.807327900156  -  9382.16283186394
90 % CI range for the age group  46-50 sample size of  10000  :  9048.92324496765  -  9209.046914796447
90 % CI range for the age group  51-55 sample size of  10  :  6846.491425329519  -  11999.751982803287
90 % CI range for the age group  51-55 sample size of  100  :  8608.319667170095  -  10237.92374096271
90 % CI range for the age group  51-55 sample size of  1000  :  9165.458676192715  -  9680.784731940092
90 % CI range for the age group  51-55 sample size of  10000  :  9341.641500376772  -  9504.601907756034
90 % CI range for the age group  55+ sample size of  10  :  6687.882984037911  -  11745.417456821293
90 % CI range for the age group  55+ sample size of  100  :  8416.983806488885  -  10016.31663437032
90 % CI range for the age group  55+ sample size of  1000  :  8963.773496790434  -  9469.526944068772
90 % CI range for the age group  55+ sample size of  10000  :  9136.68357903553  -  9296.616861823675
```

```
# for 95 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.025
r = 0.975
CI_95_age_0_17 = []
CI_95_age_18_25 = []
CI_95_age_26_35 = []
CI_95_age_36_45 = []
CI_95_age_46_50 = []
CI_95_age_51_55 = []
CI_95_age_55_plus = []

age_group_list = ['0-17','18-25','26-35','36-45','46-50','51-55','55+']
for i in range(0,len(age_group_list)):
  n = 10
  while n < df[df['Age'] == age_group_list[i]].shape[0] :
    sample_size = n
    mu = mean_of_ages_list[i]
    std_sample = std_dev_of_ages_list[i]/np.sqrt(n)
    #print("mean of sample : ",mu)
    #print("std of sample : ",std_sample)
    z_of_r_value = norm.ppf(r)
    z_of_l_value = norm.ppf(l)
    r_value = mu + z_of_r_value * std_sample
    l_value = mu + z_of_l_value * std_sample
    #print("Right value : ",r_value)
    #print("Left value : ",l_value)
    if '-' in age_group_list[i]:
      parts = age_group_list[i].split('-')
    elif '+' in  age_group_list[i]:
      #print("check point 2")
      parts = age_group_list[i].split('+')
    #print("parts list print :",parts)

    #next if else block
    if parts[0] == '0':
      CI_95_age_0_17.append((l_value,r_value))
    elif parts[0] == '18':
      CI_95_age_18_25.append((l_value,r_value))
    elif parts[0] == '26':
      CI_95_age_26_35.append((l_value,r_value))
    elif parts[0] == '36':
      CI_95_age_36_45.append((l_value,r_value))
    elif parts[0] == '46':
      CI_95_age_46_50.append((l_value,r_value))
    elif parts[0] == '51':
      CI_95_age_51_55.append((l_value,r_value))
    elif parts[0] == '55':
      CI_95_age_55_plus.append((l_value,r_value))
      #print("CP 3")
    print("95 % CI range for the age group ",age_group_list[i],"sample size of ",n," : ",l_value," - ",r_value)
    n = n * 10
```

```
95 % CI range for the age group  0-17 sample size of  10  :  5749.8452801705225  -  11985.048812431925
95 % CI range for the age group  0-17 sample size of  100  :  7881.574804467564  -  9853.319288134884
95 % CI range for the age group  0-17 sample size of  1000  :  8555.686869688154  -  9179.207222914294
95 % CI range for the age group  0-17 sample size of  10000  :  8768.859822117858  -  8966.03427048459
95 % CI range for the age group  18-25 sample size of  10  :  6038.176972485472  -  12209.886490175852
95 % CI range for the age group  18-25 sample size of  100  :  8148.19877468862  -  10099.864687972706
95 % CI range for the age group  18-25 sample size of  1000  :  8815.446255446142  -  9432.617207215182
95 % CI range for the age group  18-25 sample size of  10000  :  9026.448435666458  -  9221.615026994867
95 % CI range for the age group  26-35 sample size of  10  :  6133.287174690047  -  12253.65267283649
95 % CI range for the age group  26-35 sample size of  100  :  8225.755169420578  -  10161.18467810596
95 % CI range for the age group  26-35 sample size of  1000  :  8887.451648855946  -  9499.488198670591
95 % CI range for the age group  26-35 sample size of  10000  :  9096.698448329  -  9290.241399197537
95 % CI range for the age group  26-35 sample size of  100000  :  9162.868096272536  -  9224.071751254001
95 % CI range for the age group  36-45 sample size of  10  :  6200.0106935082  -  12308.393733915502
95 % CI range for the age group  36-45 sample size of  100  :  8288.38205229028  -  10220.022375133422
95 % CI range for the age group  36-45 sample size of  1000  :  8948.783061691485  -  9559.621365732217
```

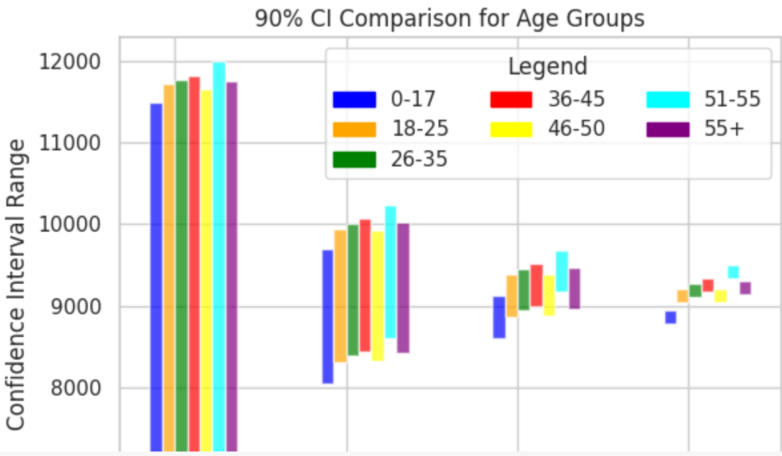
```
95 % CI range for the age group  36-45 sample size of  10000  :  9157.620197569693  -  9350.784229854009
95 % CI range for the age group  36-45 sample size of  100000  :  9223.660298509814  -  9284.744128913888
95 % CI range for the age group  46-50 sample size of  10  :  6112.186092783106  -  12145.78406698099
95 % CI range for the age group  46-50 sample size of  100  :  8174.9894756698895  -  10082.980684094206
95 % CI range for the age group  46-50 sample size of  1000  :  8827.305181172154  -  9430.664978591942
95 % CI range for the age group  46-50 sample size of  10000  :  9033.585519460832  -  9224.384640303264
95 % CI range for the age group  51-55 sample size of  10  :  6352.877358276548  -  12493.366049856257
95 % CI range for the age group  51-55 sample size of  100  :  8452.22519347145  -  10394.018214661357
95 % CI range for the age group  51-55 sample size of  1000  :  9116.097269487418  -  9730.146138645388
95 % CI range for the age group  51-55 sample size of  10000  :  9326.032053006908  -  9520.211355125899
95 % CI range for the age group  55+ sample size of  10  :  6203.438207274086  -  12229.86223358512
95 % CI range for the age group  55+ sample size of  100  :  8263.788916974336  -  10169.51152388487
95 % CI range for the age group  55+ sample size of  1000  :  8915.329019114051  -  9517.971421745155
95 % CI range for the age group  55+ sample size of  10000  :  9121.364090084076  -  9311.93635077513
```

```python
# for 99 % confidence interval
# say initial sample size is 10...so n = 10
l = 0.005
r = 0.995
CI_99_age_0_17 = []
CI_99_age_18_25 = []
CI_99_age_26_35 = []
CI_99_age_36_45 = []
CI_99_age_46_50 = []
CI_99_age_51_55 = []
CI_99_age_55_plus = []

age_group_list = ['0-17','18-25','26-35','36-45','46-50','51-55','55+']
for i in range(0,len(age_group_list)):
  n = 10
  while n < df[df['Age'] == age_group_list[i]].shape[0] :
    sample_size = n
    mu = mean_of_ages_list[i]
    std_sample = std_dev_of_ages_list[i]/np.sqrt(n)
    #print("mean of sample : ",mu)
    #print("std of sample : ",std_sample)
    z_of_r_value = norm.ppf(r)
    z_of_l_value = norm.ppf(l)
    r_value = mu + z_of_r_value * std_sample
    l_value = mu + z_of_l_value * std_sample
    #print("Right value : ",r_value)
    #print("Left value : ",l_value)
    if '-' in age_group_list[i]:
      parts = age_group_list[i].split('-')
    elif '+' in  age_group_list[i]:
      #print("check point 2")
      parts = age_group_list[i].split('+')
    #print("parts list print :",parts)

    #next if else block
    if parts[0] == '0':
      CI_99_age_0_17.append((l_value,r_value))
    elif parts[0] == '18':
      CI_99_age_18_25.append((l_value,r_value))
    elif parts[0] == '26':
      CI_99_age_26_35.append((l_value,r_value))
    elif parts[0] == '36':
      CI_99_age_36_45.append((l_value,r_value))
    elif parts[0] == '46':
      CI_99_age_46_50.append((l_value,r_value))
    elif parts[0] == '51':
      CI_99_age_51_55.append((l_value,r_value))
    elif parts[0] == '55':
      CI_99_age_55_plus.append((l_value,r_value))
      #print("CP 3")
    print("99 % CI range for the age group ",age_group_list[i],"sample size of ",n," : ",l_value," - ",r_value)
    n = n * 10
```

```
99 % CI range for the age group  0-17 sample size of  10  :  4770.223806824317  -  12964.670285778131
99 % CI range for the age group  0-17 sample size of  100  :  7571.7912944091695  -  10163.102798193278
99 % CI range for the age group  0-17 sample size of  1000  :  8457.724722353534  -  9277.169370248914
99 % CI range for the age group  0-17 sample size of  10000  :  8737.88147111202  -  8997.012621490429
99 % CI range for the age group  18-25 sample size of  10  :  5068.531131816035  -  13179.532330845288
99 % CI range for the age group  18-25 sample size of  100  :  7841.5698366662045  -  10406.49362599512
99 % CI range for the age group  18-25 sample size of  1000  :  8718.481671379199  -  9529.581791282126
99 % CI range for the age group  18-25 sample size of  10000  :  8995.785541864216  -  9252.277920797109
99 % CI range for the age group  26-35 sample size of  10  :  5171.7080641302055  -  13215.231783396332
99 % CI range for the age group  26-35 sample size of  100  :  7921.677155439782  -  10465.262692086755
99 % CI range for the age group  26-35 sample size of  1000  :  8791.293737799962  -  9595.646109726575
99 % CI range for the age group  26-35 sample size of  10000  :  9066.29064693092  -  9320.649200595617
99 % CI range for the age group  26-35 sample size of  100000  :  9153.252305166938  -  9233.6875423596
99 % CI range for the age group  36-45 sample size of  10  :  5240.314163499899  -  13268.090263923803
99 % CI range for the age group  36-45 sample size of  100  :  7984.899362551644  -  10523.505064872057
99 % CI range for the age group  36-45 sample size of  1000  :  8852.813408690656  -  9655.591018733046
```

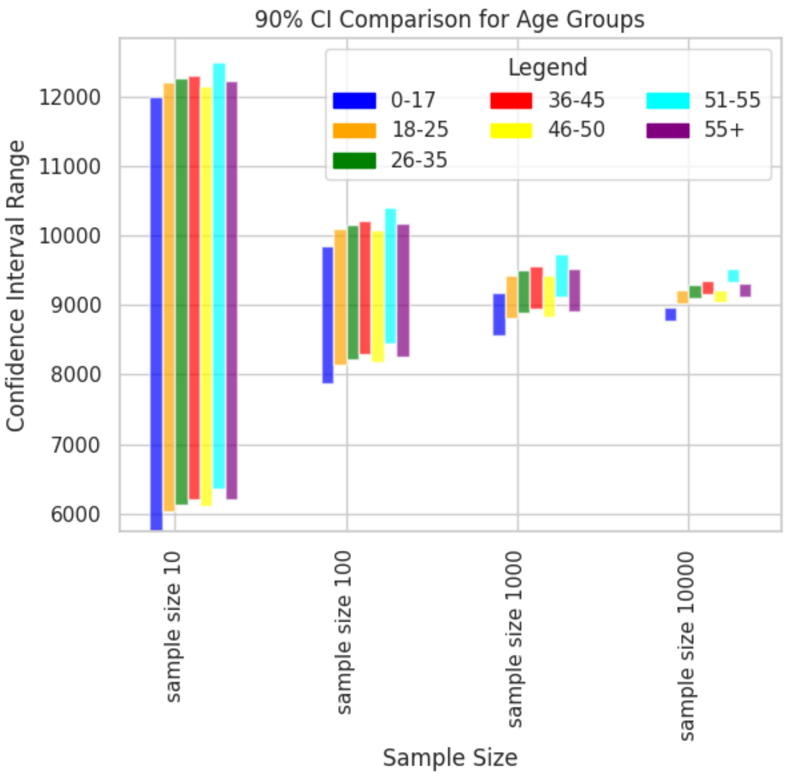
```
99 % CI range for the age group  36-45 sample size of  10000  :  9127.27192859583  -  9381.132498827872
99 % CI range for the age group  36-45 sample size of  100000  :  9214.063333209731  -  9294.34109421397
99 % CI range for the age group  46-50 sample size of  10  :  5164.239148231847  -  13093.731011532249
99 % CI range for the age group  46-50 sample size of  100  :  7875.222331091958  -  10382.747828672138
99 % CI range for the age group  46-50 sample size of  1000  :  8732.510486717028  -  9525.459673047068
99 % CI range for the age group  46-50 sample size of  10000  :  9003.60880500304  -  9254.361354761057
99 % CI range for the age group  51-55 sample size of  10  :  5388.136664891458  -  13458.106743241347
99 % CI range for the age group  51-55 sample size of  100  :  8147.147399216747  -  10699.09600891606
99 % CI range for the age group  51-55 sample size of  1000  :  9019.623200148908  -  9826.620207983899
99 % CI range for the age group  51-55 sample size of  10000  :  9295.524273581437  -  9550.71913455137
99 % CI range for the age group  55+ sample size of  10  :  5256.618371623485  -  13176.68206923572
99 % CI range for the age group  55+ sample size of  100  :  7964.378195526116  -  10468.92224533309
99 % CI range for the age group  55+ sample size of  1000  :  8820.647035548991  -  9612.653405310215
99 % CI range for the age group  55+ sample size of  10000  :  9091.423017939254  -  9341.877422919952
```

```
# Data for 90% confidence intervals for different age groups
pair1 = CI_90_age_0_17
pair2 = CI_90_age_18_25
pair3 = CI_90_age_26_35
pair4 = CI_90_age_36_45
pair5 = CI_90_age_46_50
pair6 = CI_90_age_51_55
pair7 = CI_90_age_55_plus

# Number of pairs (assuming all pairs have the same length)
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each group of bars
width = 0.15  # Adjust this value as needed
space_between_sample_sizes = 1.0  # Adjust the space between sample sizes

# Iterate through the pairs and create grouped bars for each age group
for i in range(num_pairs):
    x = i * (7 * width + space_between_sample_sizes)  # X-coordinate for the current group

    # Create bars for each age group, adjust x-coordinate accordingly
    ax.bar(x - 3*width/2, pair1[i][1] - pair1[i][0], bottom=pair1[i][0], width=width, label=f'0-17', align='center', color='bl
    ax.bar(x - width/2, pair2[i][1] - pair2[i][0], bottom=pair2[i][0], width=width, label=f'18-25', align='center', color='ora
    ax.bar(x + width/2, pair3[i][1] - pair3[i][0], bottom=pair3[i][0], width=width, label=f'26-35', align='center', color='gre
    ax.bar(x + 3*width/2, pair4[i][1] - pair4[i][0], bottom=pair4[i][0], width=width, label=f'36-45', align='center', color='r
    ax.bar(x + 5*width/2, pair5[i][1] - pair5[i][0], bottom=pair5[i][0], width=width, label=f'46-50', align='center', color='y
    ax.bar(x + 7*width/2, pair6[i][1] - pair6[i][0], bottom=pair6[i][0], width=width, label=f'51-55', align='center', color='c
    ax.bar(x + 9*width/2, pair7[i][1] - pair7[i][0], bottom=pair7[i][0], width=width, label=f'55+', align='center', color='pur

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('90% CI Comparison for Age Groups')

# Set x-axis ticks and labels
ax.set_xticks(indices * (7 * width + space_between_sample_sizes))
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {
    '0-17': 'blue',
    '18-25': 'orange',
    '26-35': 'green',
    '36-45': 'red',
    '46-50': 'yellow',
    '51-55': 'cyan',
    '55+': 'purple'
}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend',ncol=3)

# Show the plot
plt.xticks(rotation=90)
plt.grid(True)
plt.show()

# Observations
# when the sample size is small we can see there is overlap between the confidence intervals of the age groups suggesting stat
# their purchase patterns
# but this similarity wanes out as the sample size increases
```



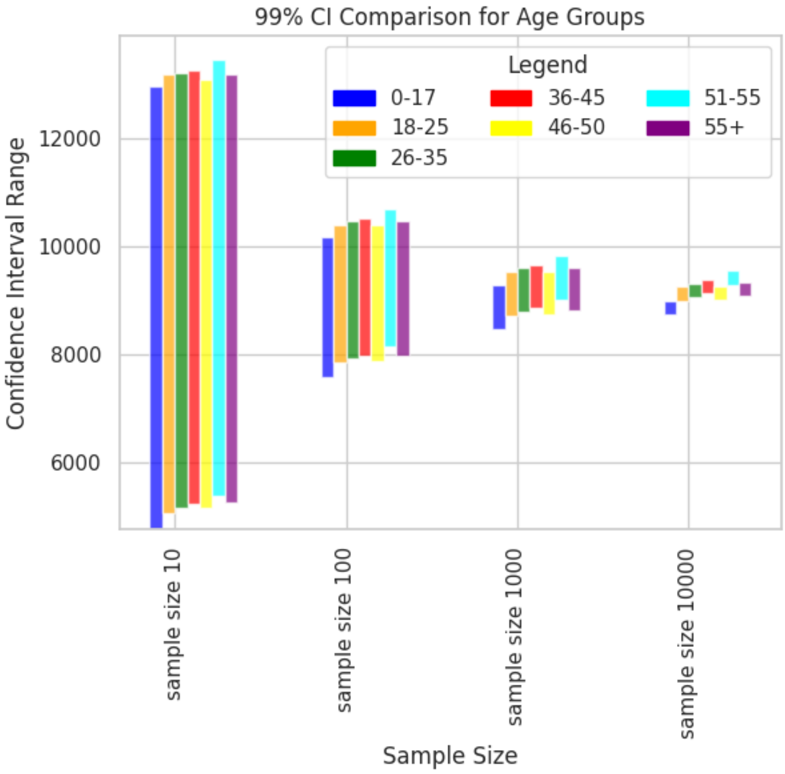

```
# Data for 95% confidence intervals for different age groups
pair1 = CI_95_age_0_17
pair2 = CI_95_age_18_25
pair3 = CI_95_age_26_35
pair4 = CI_95_age_36_45
pair5 = CI_95_age_46_50
pair6 = CI_95_age_51_55
pair7 = CI_95_age_55_plus

# Number of pairs (assuming all pairs have the same length)
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each group of bars
width = 0.15  # Adjust this value as needed
space_between_sample_sizes = 1.0  # Adjust the space between sample sizes

# Iterate through the pairs and create grouped bars for each age group
for i in range(num_pairs):
    x = i * (7 * width + space_between_sample_sizes)  # X-coordinate for the current group

    # Create bars for each age group, adjust x-coordinate accordingly
    ax.bar(x - 3*width/2, pair1[i][1] - pair1[i][0], bottom=pair1[i][0], width=width, label=f'0-17', align='center', color='bl
    ax.bar(x - width/2, pair2[i][1] - pair2[i][0], bottom=pair2[i][0], width=width, label=f'18-25', align='center', color='ora
    ax.bar(x + width/2, pair3[i][1] - pair3[i][0], bottom=pair3[i][0], width=width, label=f'26-35', align='center', color='gre
    ax.bar(x + 3*width/2, pair4[i][1] - pair4[i][0], bottom=pair4[i][0], width=width, label=f'36-45', align='center', color='r
    ax.bar(x + 5*width/2, pair5[i][1] - pair5[i][0], bottom=pair5[i][0], width=width, label=f'46-50', align='center', color='y
    ax.bar(x + 7*width/2, pair6[i][1] - pair6[i][0], bottom=pair6[i][0], width=width, label=f'51-55', align='center', color='c
    ax.bar(x + 9*width/2, pair7[i][1] - pair7[i][0], bottom=pair7[i][0], width=width, label=f'55+', align='center', color='pur

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('95% CI Comparison for Age Groups')

# Set x-axis ticks and labels
ax.set_xticks(indices * (7 * width + space_between_sample_sizes))
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
# Create a custom legend
legend_labels = {
    '0-17': 'blue',
    '18-25': 'orange',
    '26-35': 'green',
    '36-45': 'red',
    '46-50': 'yellow',
    '51-55': 'cyan',
    '55+': 'purple'
}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend',ncol=3)

# Show the plot
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```

```
# Observations
# when the sample size is small we can see there is overlap between the confidence intervals of the age groups suggesting stat
# their purchase patterns
# but this similarity wanes out as the sample size increases
```




```
# Data for 99% confidence intervals for different age groups
pair1 = CI_99_age_0_17
pair2 = CI_99_age_18_25
pair3 = CI_99_age_26_35
pair4 = CI_99_age_36_45
pair5 = CI_99_age_46_50
pair6 = CI_99_age_51_55
pair7 = CI_99_age_55_plus

# Number of pairs (assuming all pairs have the same length)
num_pairs = len(pair1)

# Create a list of indices for the x-axis
indices = np.arange(num_pairs)

# Create a figure and axis
fig, ax = plt.subplots()

# Width of each group of bars
width = 0.15  # Adjust this value as needed
space_between_sample_sizes = 1.0  # Adjust the space between sample sizes

# Iterate through the pairs and create grouped bars for each age group
for i in range(num_pairs):
    x = i * (7 * width + space_between_sample_sizes)  # X-coordinate for the current group

    # Create bars for each age group, adjust x-coordinate accordingly
    ax.bar(x - 3*width/2, pair1[i][1] - pair1[i][0], bottom=pair1[i][0], width=width, label=f'0-17', align='center', color='bl
    ax.bar(x - width/2, pair2[i][1] - pair2[i][0], bottom=pair2[i][0], width=width, label=f'18-25', align='center', color='ora
    ax.bar(x + width/2, pair3[i][1] - pair3[i][0], bottom=pair3[i][0], width=width, label=f'26-35', align='center', color='gre
    ax.bar(x + 3*width/2, pair4[i][1] - pair4[i][0], bottom=pair4[i][0], width=width, label=f'36-45', align='center', color='r
    ax.bar(x + 5*width/2, pair5[i][1] - pair5[i][0], bottom=pair5[i][0], width=width, label=f'46-50', align='center', color='y
    ax.bar(x + 7*width/2, pair6[i][1] - pair6[i][0], bottom=pair6[i][0], width=width, label=f'51-55', align='center', color='c
    ax.bar(x + 9*width/2, pair7[i][1] - pair7[i][0], bottom=pair7[i][0], width=width, label=f'55+', align='center', color='pur

# Adding labels and title
ax.set_xlabel('Sample Size')
ax.set_ylabel('Confidence Interval Range')
ax.set_title('99% CI Comparison for Age Groups')

# Set x-axis ticks and labels
ax.set_xticks(indices * (7 * width + space_between_sample_sizes))
ax.set_xticklabels([f'sample size {10 ** (i+1)}' for i in range(num_pairs)])

# Add a legend
```

```
# Create a custom legend
legend_labels = {
    '0-17': 'blue',
    '18-25': 'orange',
    '26-35': 'green',
    '36-45': 'red',
    '46-50': 'yellow',
    '51-55': 'cyan',
    '55+': 'purple'
}
handles = [plt.Rectangle((0,0),1,1, color=color, label=label) for label, color in legend_labels.items()]
ax.legend(handles=handles, title='Legend',ncol=3)

# Show the plot
plt.xticks(rotation=90)
plt.grid(True)
plt.show()

# Observations
# when the sample size is small we can see there is overlap between the confidence intervals of the age groups suggesting stat
# their purchase patterns
# but this similarity wanes out as the sample size increases
```




```
# Insights
'''
The observations have been mentioned at the plots but again I'm collating them here
1.people with a stay of 2 years in the same city tend to purchase more
2.The Median purchase of married and unmarried customers is almost same
3.The distribution of purchase pattern of married and unmarried customers is alsmost same as it is evident from the boxplot
4.People with a good occupation(more income) tend to purchase more as it is evident from the line plot
5.Men and Women have same purchase median as we can see it from the boxplot
6.when the sample size is small we can see partial overlap of the Confidence Intervals indicating there might be some
  some statistical similarity between the purchase patterns of males and females
  and this overlap wanes out as the sample size increases
7.Irrespective of the sample size we can see there is complete overlap of the confidence intervals for the purchase patterns c
  married and unmarried customers
  so, we can say that there is a statistical similarity between the purchase patterns of married and unmarried customers
8.when the sample size is small we can see there is overlap between the confidence intervals of the age groups suggesting stat
  similarities between their purchase patterns
  but this similarity wanes out as the sample size increases
'''
```

```
'\nThe observations have been mentioned at the plots but again I'm collating them here\n1.people with a stay of 2 years
in the same city tend to purchase more\n2.The Median purchase of married and unmarried customers is almost same\n3.The d
istribution of purchase pattern of married and unmarried customers is alsmost same as it is evident from the boxplot\n4.
People with a good occupation(more income) tend to purchase more as it is evident from the line plot\n5.Men and Women ha
ve same purchase median as we can see it from the boxplot\n6.when the sample size is small we can see partial overlap of
the Confidence Intervals indicating there might be some \n  some statistical similarity between the purchase patterns of
males and females\n  and this overlap wanes out as the sample size increases\n7.Irrespective of the sample size we can s
ee there is complete overlap of the confidence intervals for the purchase patterns of \n  married and unmarried customer
```

```
# Recommendations
'''
Based on the insights you've gathered from the dataset, you can make several recommendations to the company:

1. Target Marketing Based on Stay Duration:
   - Given that customers with a stay of 2 years in the same city tend to purchase more, consider tailoring marketing campaign

2. Marital Status Doesn't Impact Median Purchase:
   - Since the median purchase amount is almost the same for married and unmarried customers, focus on customer segments beyon

3. Similar Purchase Patterns for Married and Unmarried Customers:
   - Acknowledge that there is a statistical similarity between the purchase patterns of married and unmarried customers. This

4. Leverage High-Income Occupations:
   - Customers with higher-income occupations tend to purchase more. Explore strategies to attract and retain these high-incom

5. Gender-Neutral Marketing:
   - Given that there is no significant difference in purchase median between men and women, consider adopting a gender-neutra

6. Sample Size and Confidence Intervals:
   - Recognize that the interpretation of confidence intervals may change with sample size. As the sample size increases, any

7. Age Groups and Purchase Patterns:
   - While there may be some statistical overlap in purchase patterns among age groups, this can vary with sample size. To tar

8. Continuous Monitoring and Adaptation:
   - Continuously monitor customer behavior and preferences. Market dynamics and consumer preferences evolve over time. Regula

9. Customer Engagement and Loyalty:
   - Invest in customer engagement initiatives and loyalty programs to retain existing customers and encourage repeat purchase

10. Data-Driven Decision-Making:
    - Utilize data analytics and machine learning techniques to gain deeper insights into customer behavior. Predictive modeli
'''
```

```
# Dummy Page - when I convert this notebook to PDF it is missing the last two cells
# That is why I created this dummy page
```

Connected to Python 3 Google Compute Engine backend