walmart

September 18, 2022

[78]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
```

[79]:
```
from google.colab import drive

drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[80]:
```
data=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/walmart_data.txt")
```

[81]:
```
data.head()
```

[81]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969

[82]:
```
data.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
```

1 Conclusion

1.0.1 no null value

2 Need to change data formats required for convinence

[83]:
```
data["City_Category"]=data.City_Category.replace({"A":1,"B":2,"C":3})
# converting city cateogry to 1,2,3 for int format
```

[84]:
```
#converting user id to obj
data["User_ID"]=data.User_ID.astype("object")
```

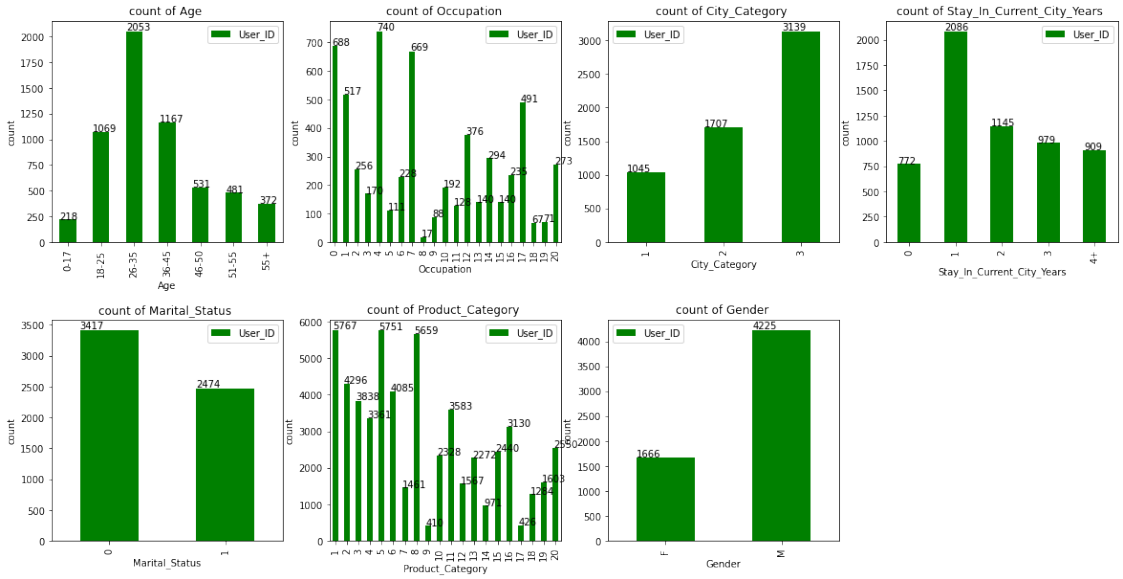
[85]:
```
data.head()
```

[85]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category	Purchase
0	1000001	P00069042	F	0-17	10	1	2	0	3	8370
1	1000001	P00248942	F	0-17	10	1	2	0	1	15200
2	1000001	P00087842	F	0-17	10	1	2	0	12	1422
3	1000001	P00085442	F	0-17	10	1	2	0	12	1057
4	1000002	P00285442	M	55+	16	3	4+	0	8	7969

[86]:
```
column=["Age","Occupation","City_Category","Stay_In_Current_City_Years","Marital_Status","Prod
count=0
for i in column:
  count+=1
```

```
 plt.subplot(2,4,(count))
 column_ratio=data.groupby(i)["User_ID"].nunique()
 plt.subplots_adjust(hspace=0.35)
 ax=column_ratio.plot(kind="bar",ylabel="count",title=f"count of {i}␣
↪",legend=True,figsize=(20,10),color="g")
 for p in ax.patches:
     ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 1.
↪005))
```

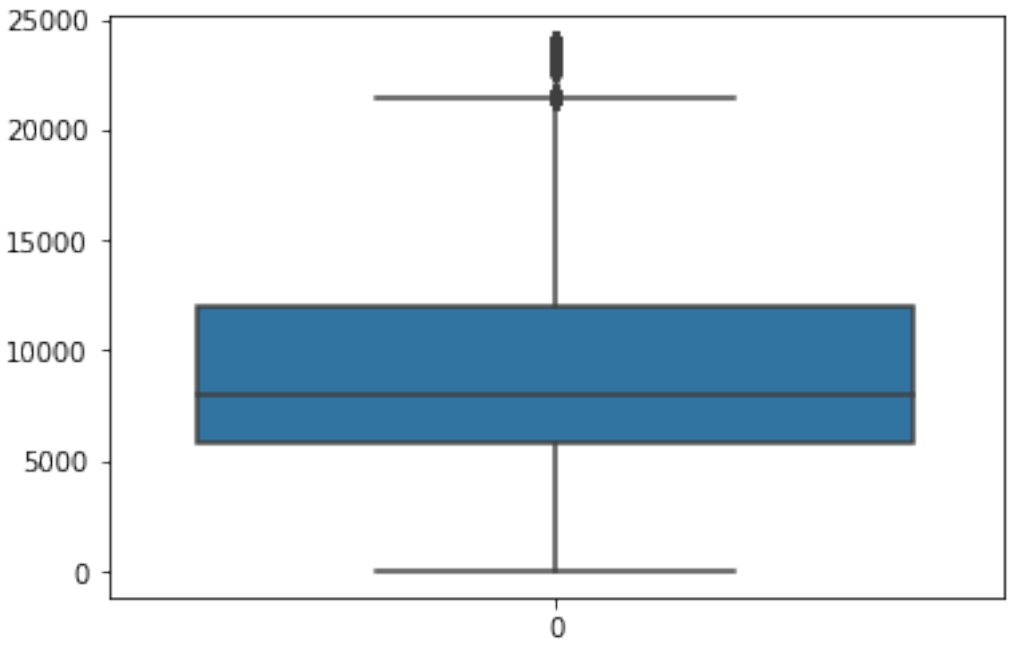


[87]:
```
data.columns
```

[87]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category',
       'Purchase'],
      dtype='object')
```

[88]:
```
#Detecting outliers with box plot
sns.boxplot(data=data.Purchase)
```

[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f358beae8d0>
```



[89]:
```
data.groupby("Gender")["Purchase"].mean()
```

[89]:
```
Gender
F    8734.565765
M    9437.526040
Name: Purchase, dtype: float64
```

By looking at a sample we can see there is a difference in purchase behaviour according to gender, since it is just a sample we will proceed to apply clt to verify behaviour over population

[90]:
```
d=np.array(data[data["Gender"]=="M"].Purchase) #making numpy array of male␣
↪purchase
d
```

[90]:
```
array([ 7969, 15227, 19215, …,   494,   473,   368])
```

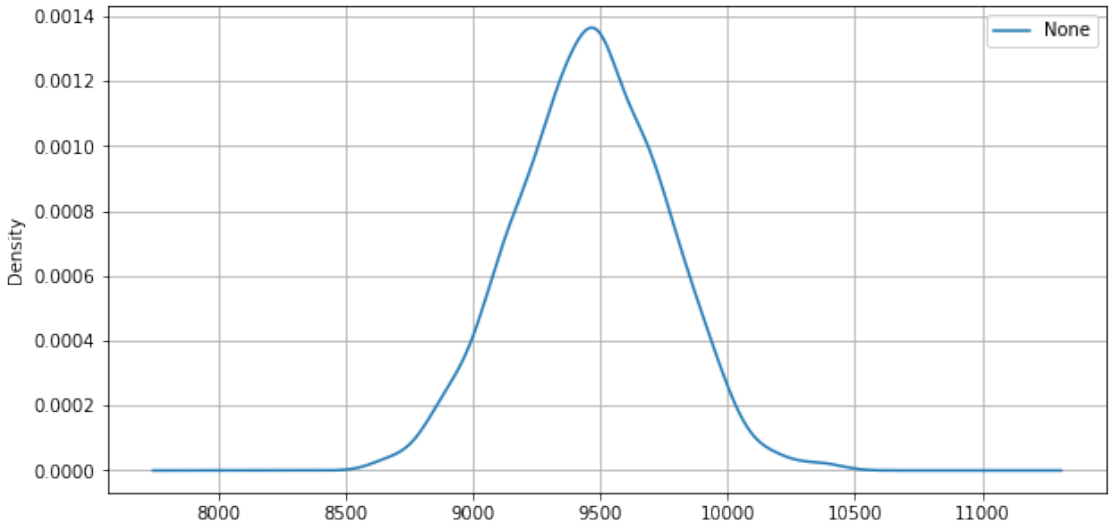
[91]:
```
sample_mean=np.random.choice(d,size=300,replace=True).mean() # finding sample␣
↪mean of male purchase with replacement and sample size 200
sample_mean
```

[91]:
```
9429.736666666666
```

[92]:
```
# making list of various sample_means of 200 in sample size for 1000 times
population_mean_list=[np.random.choice(d,size=300,replace=True).mean() for i in␣
↪range (1000)]
```

[93]:
```
# finding population mean
population_mean_list=[np.random.choice(d,size=300,replace=True).mean() for i in␣
 ↪range (1000)]
pd.Series(population_mean_list).
 ↪plot(kind="kde",figsize=(10,5),legend=True,grid=True)
```

[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f358d2cb550>
```



[93]:
```

```

[94]:
```
d=np.array(data[data["Gender"]=="M"].Purchase) #formulate data to analyze
# making simulation function in order to simulate for various sample size and␣
 ↪iteration to observe its effect
def simulation_iter (data,sample_size,iteration):
  sample_mean=np.random.choice(d,size=sample_size,replace=True).mean()
  population_mean_list=[np.random.choice(d,size=sample_size,replace=True).
 ↪mean() for i in range (iteration)]
  return pd.Series(population_mean_list).
 ↪plot(kind="kde",figsize=(20,5),label=iteration,legend=True,grid=True)
```

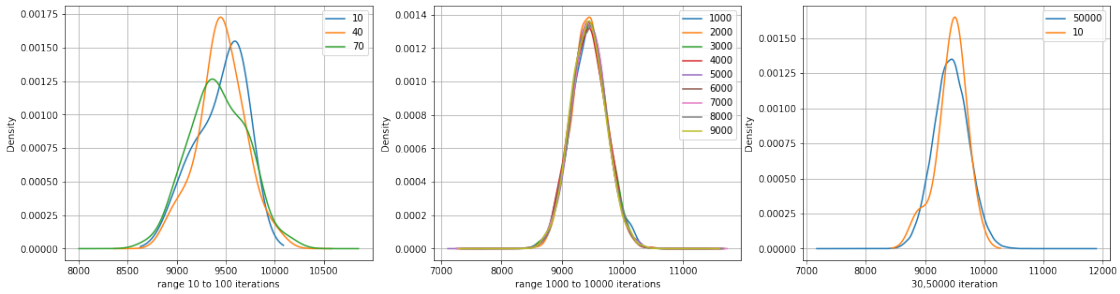
[95]:
```
plt.subplot(1,3,1) #subplot1
for i in range(10,100,30):  #simulation for iterations range 10 to 100
  simulation_iter(d,300,i)
plt.xlabel("range 10 to 100 iterations")

plt.subplot(1,3,2)  #simulation for iteration range 1000 to 10000
for i in range(1000,10000,1000):
  simulation_iter(d,300,i)
plt.xlabel("range 1000 to 10000 iterations")
```

```
plt.subplot(1,3,3)  # simulation for 15000 iteration
simulation_iter(d,300,50000)
simulation_iter(d,300,10)


plt.xlabel("30,50000 iteration ")
```

[95]:
```
Text(0.5, 0, '30,50000 iteration ')
```



As no of times of experiment done is more the distribution becomes closer to **normal distribution**

[96]:
```
# making simulation function in order to simulate for various sample size and␣
 ↪iteration to observe its effect
def simulation_sample (d,sample_size,iteration):
  sample_mean=np.random.choice(d,size=sample_size,replace=True).mean()
  population_mean_list=[np.random.choice(d,size=sample_size,replace=True).
 ↪mean() for i in range (iteration)]
  return pd.Series(population_mean_list).
 ↪plot(kind="kde",figsize=(20,5),label=sample_size,legend=True,grid=True) #␣
 ↪here only i changed label
```

[ ]:
```
plt.subplot(1,3,1) #subplot1
for i in range(10,100,20):  #simulation for frequency range 10 to 1000
  simulation_sample(d,i,10000)
for i in range(100,1000,200):  #simulation for different frequency
  simulation_sample(d,i,10000)
plt.xlabel("range 10 to 1000 samples")

plt.subplot(1,3,2)  #simulation for frequency range 1000 to 10000
for i in range(1000,10000,1000):
  simulation_sample(d,i,10000)
plt.xlabel("range 1000 to 10000 samples")

plt.subplot(1,3,3)  # simulation for 15000 samples
simulation_sample(d,15000,10000)
```

```
plt.xlabel("15000 samples")
```

As sample size increases sample mean becomes closer to population mean

from the two experiments we understands we should take more samples and more no of iteration

[ ]:
```
M=np.array(data[data["Gender"]=="M"].Purchase)
F=np.array(data[data["Gender"]=="F"].Purchase)
def simulation_sample (df1,sample_size,iteration):
  sample_mean=np.random.choice(df1,size=sample_size,replace=True).mean()
  sample_mean_list=[np.random.choice(df1,size=sample_size,replace=True).mean()␣
 ↪for i in range (iteration)]
  return pd.Series(sample_mean_list).
 ↪plot(kind="kde",figsize=(20,5),label=sample_size,legend=True,grid=True) #␣
 ↪here only i changed label
```

[ ]:
```
simulation_sample(M,200,1000)
simulation_sample(F,200,1000)
```

[ ]:
```
# finding confidence intervals
def confidence_interval(df1,cf,sample_size=300,iteration=1000):
  sample_mean=np.random.choice(df1,size=sample_size,replace=True).mean()
  sample_mean_list=[np.random.choice(df1,size=sample_size,replace=True).mean()␣
 ↪for i in range(iteration)]
  population_mean=np.array(sample_mean_list).mean()  # finding population mean
  std_error=np.array(sample_mean_list).std() # finding std error
  lower=population_mean-cf*std_error # finding lower limit
  upper=population_mean+cf*std_error # finding upper limit
  return lower,upper
```

for 95% confidence z score euivalent = 1.96 leave 2.5% from both end

[ ]:
```
M=np.array(data[data["Gender"]=="M"].Purchase) # data of male
F=np.array(data[data["Gender"]=="F"].Purchase) # data of female



# finding overlaps

a,b=confidence_interval(M,1.96)
c,d=confidence_interval(F,1.96)
x = range(int(a),int(b))
y = range(int(c),int(d))

def range_overlapping(x, y):
    if x.start == x.stop or y.start == y.stop:
        return False
    return x.start <= y.stop and y.start <= x.stop
```

```
print(range_overlapping(x, y))
print(f"confidence interval of m is {a},{b}")
print(f"confidence interval of f is {c},{d}")
```

As interval is overlapping we can say there is no difference in purchasing behaviour for male and female

[ ]:
```
data.columns
```

[ ]:
```
data.sample(10)
```

[ ]:
```
check=['Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category']
```

[ ]:
```
l=[]
for i in data["Age"].unique():
  c=np.array(data[data["Age"]==i].Purchase) # data of j,i

  print(f"confidence interval of Age {i} is {int(a)},{int(b)}")
  a,b=confidence_interval(c,1.96)

  e1=(int(a),(b))
  l.append(e1)
for i in l:
  for j in l:
    a=i[0]
    b=i[1]
    c=j[0]
    d=j[1]
    x = range(int(a),int(b))
    y = range(int(c),int(d))

    def range_overlapping(x, y):
        if x.start == x.stop or y.start == y.stop:
            return False
        return x.start <= y.stop and y.start <= x.stop

    if range_overlapping(x, y) == False:
      print(i,j)
```

no false condition satisfied hence

purchasing behaviour across all ages are overlapping

[ ]:
```
l=[]
for i in data["Occupation"].unique():
  c=np.array(data[data["Occupation"]==i].Purchase) # data of j,i
```

```
  print(f"confidence interval of Occupation {i} is {int(a)},{int(b)}")
  a,b=confidence_interval(c,1.96)

  e1=(int(a),(b))
  l.append(e1)
for i in l:
  for j in l:
    a=i[0]
    b=i[1]
    c=j[0]
    d=j[1]
    x = range(int(a),int(b))
    y = range(int(c),int(d))

    def range_overlapping(x, y):
        if x.start == x.stop or y.start == y.stop:
            return False
        return x.start <= y.stop and y.start <= x.stop


    if range_overlapping(x, y) == False:
      print(i,j)
```

occupation 7 and coccupation 1 are not overlapping

occupation 1 and coccupation 17 are not overlapping

occupation 1 and coccupation 0 are not overlapping

[ ]:
```
l=[]
for i in data["City_Category"].unique():
  c=np.array(data[data["City_Category"]==i].Purchase) # data of j,i

  print(f"confidence interval of City_Category {i} is {int(a)},{int(b)}")
  a,b=confidence_interval(c,1.96)

  e1=(int(a),(b))
  l.append(e1)
for i in l:
  for j in l:
    a=i[0]
    b=i[1]
    c=j[0]
    d=j[1]
    x = range(int(a),int(b))
    y = range(int(c),int(d))

    def range_overlapping(x, y):
```

```
        if x.start == x.stop or y.start == y.stop:
            return False
        return x.start <= y.stop and y.start <= x.stop


    if range_overlapping(x, y) == False:
      print(i,j)
```

all across cities purchasing behaviour is same

[ ]:
```
l=[]
for i in data["Marital_Status"].unique():
  c=np.array(data[data["Marital_Status"]==i].Purchase) # data of j,i

  print(f"confidence interval of Marital_Status {i} is {int(a)},{int(b)}")
  a,b=confidence_interval(c,1.96)

  e1=(int(a),(b))
  l.append(e1)
for i in l:
  for j in l:
    a=i[0]
    b=i[1]
    c=j[0]
    d=j[1]
    x = range(int(a),int(b))
    y = range(int(c),int(d))

    def range_overlapping(x, y):
        if x.start == x.stop or y.start == y.stop:
            return False
        return x.start <= y.stop and y.start <= x.stop


    if range_overlapping(x, y) == False:
      print(i,j)
```

Purchasing behaviour is not affected by Marital_Status

[ ]:
```
l=[]
for i in data["Product_Category"].unique():
  c=np.array(data[data["Product_Category"]==i].Purchase) # data of j,i

  print(f"confidence interval of Product_Category {i} is {int(a)},{int(b)}")
  a,b=confidence_interval(c,1.96)

  e1=(int(a),(b))
  l.append(e1)
```

```
for i in l:
  for j in l:
    a=i[0]
    b=i[1]
    c=j[0]
    d=j[1]
    x = range(int(a),int(b))
    y = range(int(c),int(d))

    def range_overlapping(x, y):
        if x.start == x.stop or y.start == y.stop:
            return False
        return x.start <= y.stop and y.start <= x.stop


    if range_overlapping(x, y) == False:
      print(i,j)
```

From above simulations we were able to conclude the purchasing behavious across all caterogry is similar except product wise . we should advertise to all cateogory

occupation 7 and coccupation 1 are not overlapping

occupation 1 and coccupation 17 are not overlapping

occupation 1 and coccupation 0 are not overlapping

hence we concluded occupation 1,7,17 and 0 are behaving differently try to figure out why

since purchasing behaviour is similar in all cateogry advertise to all to get maximum profit and figure out which product is selling less focuss on those like product 3 is having mean value less but consider how much percentage profit is there, now can't proceed due to missing data of profit

Purchasing behaviour is different for different product

As interval is overlapping we can say there is no difference in purchasing behaviour for male and female

from the experiments we understands we should take more samples and more no of iteration

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
!cp "./drive/My Drive/Colab Notebooks/walmart.ipynb" ./
```

[ ]:
```
!jupyter nbconvert --to PDF "walmart.ipynb"
```

[ ]:
```

```