In [1]:

**from**

**scipy.stats**

**import**

norm

,

t

**import**

**matplotlib.pyplot**

**as**

**plt**

**import**

**numpy**

**as**

**np**

**import**

**pandas**

**as**

**pd**

pd

.

options

.

display

.

max\_columns

=

20

pd

.

options

.

display

.

max\_rows

=

500

**from**

**statistics**

**import**

stdev

,

mean

**import**

**warnings**

warnings

.

filterwarnings

(

"ignore"

)

**import**

**math**

# t- distribution Confidence Interval

The below method performs the following

In [2]:

**def**

t\_ci

(

confidence\_interval

:

float

=

0.90

,

sample\_std\_dev

:

float

=

1

,

number\_of\_samples

:

int

=

2

,

mean

:

float

=

0

,

two\_sided

:

bool

=

**True**

):

*'''*

*Parameters*

*----------*

*confidence\_interval : float, optional*

*DESCRIPTION. The default is 0.90.*

*sample\_std\_dev : float, optional*

*DESCRIPTION. The default is 1.*

*number\_of\_samples : int, optional*

*DESCRIPTION. The default is 100.*

*mean : float, optional*

*DESCRIPTION. The default is 0.*

*two\_sided : bool, optional*

*DESCRIPTION. The default is True.*

*Returns*

*-------*

*TYPE*

*DESCRIPTION.*

*'''*

*# declaring problem constants*

x\_bar

=

mean

n

=

number\_of\_samples

*# number of samples taken*

sigma

=

sample\_std\_dev

*# population standard deviation*

sigma\_x\_bar

=

sigma

/

n

\*\*

0.5

*# std dev of sample means*

df

=

n

-

1

*# degrees of freedom*

*# get the probabilities of the tail areas*

**if**

two\_sided

:

alpha\_high

=

(

1

+

confidence\_interval

)

/

2

alpha\_low

=

(

1

-

confidence\_interval

)

/

2

**else**

:

alpha\_high

=

confidence\_interval

alpha\_low

=

0

*# compute the value of x\_lower and x\_higher*

x\_lower = t.ppf(alpha\_low,df,loc=x\_bar,scale=sigma\_x\_bar) x\_higher = t.ppf(alpha\_high,df,loc=x\_bar,scale=sigma\_x\_bar) **return** round(x\_lower,3), round(x\_higher,3)

# Read the dataset

In [3]:

df

=

pd

.

read\_csv

(

r

'SA1\_Group\_17.csv'

,

index\_col

=

'Index'

)

# Get the numerical and non-numerical columns

A separate dataset has been prepared to list out the numerical and the categorical columns.

In [4]: column\_desc = pd.read\_csv(r'Data description.csv') categorical\_cols = column\_desc.non\_numeric\_columns.dropna().tolist() numeric\_cols = column\_desc.numeric\_columns.dropna().tolist()

# View the dataset descriptive statistics of the numerical columns

Some Important points

Though OPER\_DUR\_DD is expected to be a continuous variable it looks like the rows are missing various data eventhough unit was operational. It just don't make sense that OPER\_DUR\_MM is filled (meaning the unit was operational for given number of months) but don't have the data OPER\_DUR\_DD (duration of operation in days). So for the descriptive statistics we will omit this as of now.

Other columns which are dropped because they are categorical in nature but were encoded. Getting descriptive statistics for categorical columns doesn't make sense. We will get distinct counts of them later.

In [5]:

print

(

'Descriptive Statistics for the numerical columns'

)

display

(

df

[

numeric\_cols

]

.

describe

())

Descriptive Statistics for the numerical columns

**OPER\_DUR\_MM MKT\_VAL\_FA ORI\_PURC\_VAL\_PM EMP\_TOTAL GOP\_Year3 VOE\_Year3 NET\_Year3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000.000000 | 1.000000e+04 | 1.000000e+04 | 10000.000000 | 1.000000e+04 | 1.000000e+04 | 1.000000e+04 |
| **mean** | 10.559000 | 8.547566e+05 | 3.245659e+05 | 5.885900 | 9.259034e+07 | 2.855352e+04 | 1.200584e+06 |
| **std** | 2.111437 | 5.466470e+06 | 1.791879e+06 | 10.858502 | 9.081048e+09 | 1.225493e+06 | 8.056149e+06 |
| **min** | 0.000000 | 0.000000e+00 | 0.000000e+00 | 1.000000 | 0.000000e+00 | 0.000000e+00 | -3.500000e+06 |
| **25%** | 10.000000 | 5.000000e+04 | 2.000000e+04 | 2.000000 | 4.666250e+04 | 0.000000e+00 | 5.700000e+04 |
| **50%** | 12.000000 | 1.500000e+05 | 5.045000e+04 | 3.000000 | 1.000000e+05 | 0.000000e+00 | 1.590000e+05 |
| **75%** | 12.000000 | 5.000000e+05 | 1.900000e+05 | 6.000000 | 3.800000e+05 | 0.000000e+00 | 5.112225e+05 |
| **max** | 12.000000 | 3.653958e+08 | 8.961474e+07 | 350.000000 | 9.081050e+11 | 9.430860e+07 | 4.269214e+08 |

**Inference:**Two majorly important columns GOP\_Year3 and VOE\_Year3 contains outliers. This is evident from their difference between mean and median. Let's try to remove the data which is the outlier.

# A quick view about the two columns containing outliers

Below we plot a scatter plot to visualize the presence of outliers in both the columns

In [6]:

df

.

plot

.

scatter

(

'GOP\_Year3'

,

'VOE\_Year3'

)

Out[6]: <matplotlib.axes.\_subplots.AxesSubplot at 0x28807d6a148>



**Inference:**The single point on GOP\_Year3 might going to be messing up with the whole statistical analysis. Let's remove them from the dataset

# Find out the data which is an outlier

In [7]: **def** check\_outlier(value,mean,sd): z = abs((value-mean)/sd) **return** z > 3.5

mean = df.GOP\_Year3.mean() sd = df.GOP\_Year3.std()

df['GOP\_Year3\_is\_outlier'] = df.GOP\_Year3.apply(**lambda** row: check\_outlier(row,mean,sd))

mean = df.VOE\_Year3.mean() sd = df.VOE\_Year3.std()

df['VOE\_Year3\_is\_outlier'] = df.VOE\_Year3.apply(**lambda** row: check\_outlier(row,mean,sd)) print('Number of outliers for GOP\_Year3 outlier: **{}**'.format(len(df[df.GOP\_Year3\_is\_outlier]))) print('Number of outliers for VOE\_Year3 outlier: **{}**'.format(len(df[df.VOE\_Year3\_is\_outlier])))

Number of outliers for GOP\_Year3 outlier: 1

Number of outliers for VOE\_Year3 outlier: 9

In [8]:

df\_without\_outliers

=

df

[

df

.

GOP\_Year3\_is\_outlier

==

**False**

]

df\_without\_outliers

.

GOP\_Year3

.

describe

()

Out[8]: count 9.999000e+03 mean 1.780014e+06 std 1.715855e+07 min 0.000000e+00 25% 4.662500e+04

50% 1.000000e+05 75% 3.800000e+05 max 1.193961e+09

Name: GOP\_Year3, dtype: float64

**Inference:** As expected from the graph above 1 data point is an outlier in GOP\_Year3. We shall elliminate that. But there are 9 data points as outliers for VOE\_Year3. Losing out 9 more data can be problematic. We shall keep them and move forward.

# Question 1. The 95 percent confidence interval for the “Gross output – Year 3 (Rs)

To perform this we will construct a 2 sided 95% confidence interval by t-test

In [9]: confidence\_interval = 0.95 sample\_std\_dev = df\_without\_outliers.GOP\_Year3.std() number\_of\_samples = len(df\_without\_outliers.GOP\_Year3) sample\_mean = df\_without\_outliers.GOP\_Year3.mean()

lower, higher = t\_ci(confidence\_interval,sample\_std\_dev,number\_of\_samples,sample\_mean)

print('Mean of Gross output – Year 3 of population is expected to lie between Rs. **{}** and Rs. **{}**'.format(lower ,higher))

Mean of Gross output – Year 3 of population is expected to lie between Rs. 1443654.578 and Rs. 2116372.612

# Question 2: Defining metrics for performance of the units

We define the performance of the units as follows:

1. **op\_per\_asset = GOP\_Year3/MKT\_VAL\_FA.**

This metric is useful in determining how the units are performing on the basis of utilization of the fixed assets. As a basic understanding more the MKT\_VAL\_FA more should be GOP\_Year3. If the ratio is low for any unit it means there might be a problem of under utilization of resources happening in that given unit. Also if the ratio is too high denotes the units are working with highly deprecated assets which can be a great risk sooner or later.

1. **op\_per\_employee: GOP\_Year3/EMP\_TOTAL**

In these world of automation initiatives to increase productivitiy of business this metric is very useful. If the ratio is too low it means those units might potentially show redundancies in job roles. Employees of those units might be available to take up newer challenging roles which in turn will be increasing the business. Units showing too high value might be facing employee shortage problems.

In [10]: df\_without\_outliers['op\_per\_asset'] = df\_without\_outliers['GOP\_Year3']/df\_without\_outliers['MKT\_VAL\_FA'] df\_without\_outliers['op\_per\_employee'] = df\_without\_outliers['GOP\_Year3']/df\_without\_outliers['EMP\_TOTAL'] print('Description of the two metrics') display(df\_without\_outliers[['op\_per\_asset','op\_per\_employee']].describe())

Description of the two metrics

**op\_per\_asset op\_per\_employee**

|  |  |  |
| --- | --- | --- |
| **count** | 9997.000000 | 9.999000e+03 |
| **mean** | inf | 1.479189e+05 |
| **std** | NaN | 9.660946e+05 |
| **min** | 0.000000 | 0.000000e+00 |
| **25%** | 0.450000 | 1.760000e+04 |
| **50%** | 0.900000 | 3.421429e+04 |
| **75%** | 1.790667 | 7.575000e+04 |
| **max** | inf | 7.595800e+07 |

**Inference:**It seems there are some rows in the data where the MKT\_VAL\_FA = 0 due to which we are getting inf values. This seems to be some sort of a data collection issue. To avoid this we filter the data to remove the inf values. Also there seems to be some missing values too.

In [11]: **import** **math** filtered\_df = df\_without\_outliers[~df\_without\_outliers.op\_per\_asset.isna()] filtered\_df = df\_without\_outliers[df\_without\_outliers.op\_per\_asset != math.inf] filtered\_df[['op\_per\_asset','op\_per\_employee']].describe()

Out[11]:

**op\_per\_asset op\_per\_employee**

|  |  |  |
| --- | --- | --- |
| **count** | 9952.000000 | 9.954000e+03 |
| **mean** | 2.395023 | 1.476742e+05 |
| **std** | 12.030814 | 9.679534e+05 |
| **min** | 0.000000 | 0.000000e+00 |
| **25%** | 0.450000 | 1.750075e+04 |
| **50%** | 0.891083 | 3.406559e+04 |
| **75%** | 1.762542 | 7.526600e+04 |
| **max** | 601.500000 | 7.595800e+07 |

**Inference:** This resulted in losing out around 40 data points from our sample. There can be a separate analysis how those 40 data points had MKT\_VAL\_FA = 0. But this is out of scope for this exercise.

## Question 3: 99% confidence interval for the population mean of the above metrics

**99% confidence interval for op\_per\_asset:**

Since we have described that both low op\_per\_asset and high op\_per\_asset is a problem here, we will define two sided confidence interval for the given metric

In [12]:

confidence\_interval

=

0.99

sample\_std\_dev

=

filtered\_df

.

op\_per\_asset

.

std

()

number\_of\_samples

=

len

(

filtered\_df

.

op\_per\_asset

)

sample\_mean

=

filtered\_df

.

op\_per\_asset

.

mean

()

two\_sided

=

**True**

lower

,

higher

=

t\_ci

(

confidence\_interval

,

sample\_std\_dev

,

number\_of\_samples

,

sample\_mean

,

two\_sided

)

print

(

'Mean of Output Per Asset as of Year 3 of population is expected to lie between

**{}**

and

**{}**

'

.

format

(

lower

,

higher

))

Mean of Output Per Asset as of Year 3 of population is expected to lie between 2.084 and 2.706

**99% confidence interval for op\_per\_employee:**

Since we have described that both low op\_per\_employee and high op\_per\_asset is a problem here, we will define two sided confidence interval for the given metric

In [13]:

confidence\_interval

=

0.99

sample\_std\_dev

=

filtered\_df

.

op\_per\_employee

.

std

()

number\_of\_samples

=

len

(

filtered\_df

.

op\_per\_employee

)

sample\_mean

=

filtered\_df

.

op\_per\_employee

.

mean

()

two\_sided

=

**True**

lower

,

higher

=

t\_ci

(

confidence\_interval

,

sample\_std\_dev

,

number\_of\_samples

,

sample\_mean

,

two\_sided

)

print

(

'Mean of Gross Output Per Employee as of Year 3 of population is expected to lie between Rs.

**{}**

and Rs.

**{}**

'

.

format

(

lower

,

higher

))

Mean of Gross Output Per Employee as of Year 3 of population is expected to lie between Rs.122679.005 and Rs. 172669.334

## Question 4

### a. Probability that a firm selected at random is a SSSBE unit

In [14]:

*# filter only SSSBE units*

p

=

len

(

filtered\_df

[

filtered\_df

.

UNIT\_TYPE

==

2

])

/

len

(

filtered\_df

)

print

(

f

'Probability = {round(p,3)}'

)

Probability = 0.216

### b. Probability that a firm selected at random is GOOD in performance

We calculate this by checking if the values of the column op\_per\_asset > mean of the column op\_per\_asset

In [15]: mean\_op\_per\_asset = filtered\_df.op\_per\_asset.mean() filtered\_df['good\_in\_performance'] = filtered\_df.op\_per\_asset.apply(**lambda** row: row > mean\_op\_per\_asset) p = len(filtered\_df[filtered\_df.good\_in\_performance==**True**])/len(filtered\_df) print(f'Probability = {round(p,10)}') print('Number of units performing good = **{}**'.format(len(filtered\_df[filtered\_df.good\_in\_performance==**True**])))

Probability = 0.1787221218

Number of units performing good = 1779

### c. Probability that a firm selected is a SSSBE Unit and ALSO GOOD in performance

In [16]: n\_sssbe\_good\_performance = len(filtered\_df[(filtered\_df.UNIT\_TYPE==2) \

&(filtered\_df.good\_in\_performance==**True**) \

]) print('Probability that firm is SSSBE Unit and also a good performer = **{0:.3f}**'.format(n\_sssbe\_good\_performan ce/len(filtered\_df)))

p\_good\_given\_sssbe = n\_sssbe\_good\_performance/len(filtered\_df[(filtered\_df.UNIT\_TYPE==2)]) print('Conditional probability that a firm is Good given that its SSSBE:**{}**'.format(p\_good\_given\_sssbe))

Probability that firm is SSSBE Unit and also a good performer = 0.035

Conditional probability that a firm is Good given that its SSSBE:0.16194895591647332

### d. Conclusion about performance of SSSBE units

From calculation in 4c. we can see that only a mere 3.5% of our sample data comprise of performances from SSSBE units which are performing good.

But to conclude whether SSSBE units performance are good or bad in compared to SSI we have to do a comparative study.

In [17]: n\_ssi\_good\_performance = len(filtered\_df[(filtered\_df.UNIT\_TYPE==1) \

&(filtered\_df.good\_in\_performance==**True**) \

]) print('Probability that firm is SSI Unit and also a good performer = **{0:.3f}**'.format(n\_ssi\_good\_performance/l en(filtered\_df)))

p\_good\_given\_ssi = n\_ssi\_good\_performance/len(filtered\_df[(filtered\_df.UNIT\_TYPE==1)]) print('Conditional probability that a firm is Good given that its SSI:**{}**'.format(p\_good\_given\_ssi))

Probability that firm is SSI Unit and also a good performer = 0.144

Conditional probability that a firm is Good given that its SSI:0.18335684062059238

**Inference:** From the above calculations it is clear that:

1. A majority of good performer is SSI units and not SSSBE units in our sample.
2. If we see the conditional probability to understand if given that a firm is an SSSBE unit what is the probability that it will perform good < if given that a firm is SSI Unit what is the probability of being good performer.

Based on these above caclculations it is evident that performance of SSSBE unit is not good as compared to SSI Units.

## 5. Null Hypothesis test

Null hypothesis H0: Population mean of VOE\_Year3 = 87,300

Alternate Hypothesis H1: Population mean of VOE\_Year3 ≠ 87,300

We will setup a one sided confidence interval of 0.95

Since population standard deviation is not given to us we will use sample standard deviation and use t test

In [18]:

df\_without\_outliers

.

VOE\_Year3

.

describe

()

Out[18]: count 9.999000e+03 mean 2.855638e+04 std 1.225555e+06 min 0.000000e+00 25% 0.000000e+00

50% 0.000000e+00 75% 0.000000e+00 max 9.430860e+07

Name: VOE\_Year3, dtype: float64

In [19]: confidence\_interval = 0.95 sample\_std\_dev = df\_without\_outliers.VOE\_Year3.std() number\_of\_samples = len(df\_without\_outliers.VOE\_Year3) mean = 87300 two\_sided = **False**

lower, higher = t\_ci(confidence\_interval,sample\_std\_dev,number\_of\_samples,mean,two\_sided) sample\_mean = df\_without\_outliers.VOE\_Year3.mean()

print('Sample Mean for Value of Exports for Year 3 is expected to lie between Rs. **{}** and Rs. **{}**'.format(lower

,higher))

print(f'Does sample mean falls within above range? Ans: {lower<=sample\_mean<=higher} and the value **{sample\_me an}**')

print(f'The t value for sample mean:{t.cdf(sample\_mean,df=number\_of\_samples-1,loc=mean,scale=sample\_std\_dev/n umber\_of\_samples\*\*0.5)}')

Sample Mean for Value of Exports for Year 3 is expected to lie between Rs. -inf and Rs. 107461.457

Does sample mean falls within above range? Ans: True and the value 28556.37683768377 The t value for sample mean:8.334157737807495e-07

**Inference:** It is evident that from the one sided t-test though the sample mean lies between the given ranges the P value is << 0.05. Hence we can surely reject the Null hypothesis that the population mean of VOE\_Year3 is 87300.

## 6. Special incentives for SSSBE or SSI or both

**Explanation:** Below we will define the success criteria as follows:

1. If unit is an SSSBE Unit its a success. We calculate the population proportion of its success rate.
2. If unit is an SSI Unit its a success. We calculate the population proportion of its success rate.

For the unit to get incentives the population proportion of it should be < 0.25

In [20]:

**import**

**statsmodels.api**

**as**

**sm**

**from**

**statsmodels.stats.proportion**

**import**

proportion\_confint

In [21]: confidence\_interval = 0.99 sssbe\_count = len(filtered\_df[filtered\_df.UNIT\_TYPE==2]) total\_count = len(filtered\_df)

sssbe\_pop\_prop = proportion\_confint(count=sssbe\_count, *# Number of "successes"*

nobs=total\_count, *# Number of trials* alpha=(1 - confidence\_interval))

print('Population proportion of SSSBE units is expected to lie within **{}** by confidence interval of **{}**'.format (sssbe\_pop\_prop, confidence\_interval))

Population proportion of SSSBE units is expected to lie within (0.2058626874239812, 0.22712907468170493) by c onfidence interval of 0.99

In [22]: confidence\_interval = 0.99 sssbe\_count = len(filtered\_df[filtered\_df.UNIT\_TYPE==1]) total\_count = len(filtered\_df)

sssbe\_pop\_prop = proportion\_confint(count=sssbe\_count, *# Number of "successes"*

nobs=total\_count, *# Number of trials* alpha=(1 - confidence\_interval))

print('Population proportion of SSI units is expected to lie within **{}** by confidence interval of **{}**'.format(s ssbe\_pop\_prop, confidence\_interval))

Population proportion of SSI units is expected to lie within (0.772870925318295, 0.7941373125760187) by confi dence interval of 0.99

**Inference:** Since SSSBE Unit's population proportion is expected to be lying below 25% we would recommend these special incentives for SSSBE.

## 7. Contention that a larger proportion of SSSBEs are managed by men as compared to women

**Explanation:** For this we will estimate population proportion of SSSBE Units managed by Male. The column MAN\_BY will be beneficial for this case.

1. We define success if a unit is managed by man.
2. We estimate the population proportion of SSSBE units to be managed by men from our sample by a set confidence interval.
3. If the estimated population proportion > 0.5 this contention will hold true.

In [23]:

*# filter out only SSSBE Units*

sssbe\_df

=

df

[

df

.

UNIT\_TYPE

==

2

]

sssbe\_df

.

MAN\_BY

.

value\_counts

()

Out[23]: 1 2102 2 55

Name: MAN\_BY, dtype: int64

In [24]: confidence\_interval = 0.99 no\_of\_sssbe\_units\_managed\_by\_men = len(sssbe\_df[sssbe\_df.MAN\_BY == 1]) number\_of\_sssbe\_units = len(sssbe\_df)

male\_employee\_pop\_prop = proportion\_confint(count=no\_of\_sssbe\_units\_managed\_by\_men, *# Number of "successe s"*

nobs=number\_of\_sssbe\_units, *# Number of trials* alpha=(1 - confidence\_interval))

print('Population proportion of SSSBE units being managed by men is expected to lie within **{}** by confidence i nterval of **{}**'.format(sssbe\_pop\_prop, confidence\_interval))

Population proportion of SSSBE units being managed by men is expected to lie within (0.772870925318295, 0.794 1373125760187) by confidence interval of 0.99

**Inference:** Thus we are 99% confident that the population proportion of SSSBE Units being managed by men lies much above 50%. Hence we are accepting the above contention.

## 8. Distribution of defined metrics

In [25]:

filtered\_df

[[

'op\_per\_asset'

,

'op\_per\_employee'

]]

.

describe

()

Out[25]:

**op\_per\_asset op\_per\_employee**

|  |  |  |
| --- | --- | --- |
| **count** | 9952.000000 | 9.954000e+03 |
| **mean** | 2.395023 | 1.476742e+05 |
| **std** | 12.030814 | 9.679534e+05 |
| **min** | 0.000000 | 0.000000e+00 |
| **25%** | 0.450000 | 1.750075e+04 |
| **50%** | 0.891083 | 3.406559e+04 |
| **75%** | 1.762542 | 7.526600e+04 |
| **max** | 601.500000 | 7.595800e+07 |

In [26]:

filtered\_df

[

'op\_per\_asset'

]

.

plot

.

hist

(

bins

=

35

)

Out[26]: <matplotlib.axes.\_subplots.AxesSubplot at 0x2880bb06f88>

In [27]:

filtered\_df

[

'op\_per\_employee'

]

.

plot

.

hist

(

bins

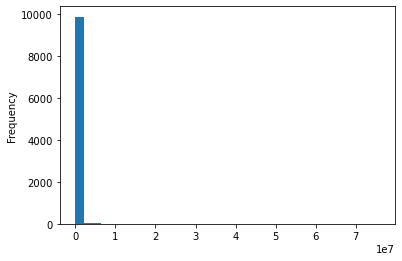
=

35

)



Out[27]: <matplotlib.axes.\_subplots.AxesSubplot at 0x2880bbd9a48>



**Inference:** The distributions of the metrics 'op\_per\_asset' and 'op\_per\_employee' are right skewed in nature. We can find the evidence from the above histograms and also the .describe() method here where it is seen that median << mean