Topics

90 min

- recap of CLT ✓
- Code for CLT & bootstrapping
- Samplings ✓

50-60 min

- Hypothesis Testing
  - intuition
  - terminology
  - Math & code

Assessments:

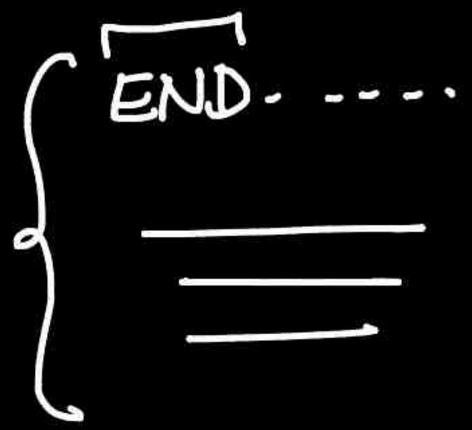
- Instructor assigns based on Concepts covered ↗ .... list all problems ...

{ P&S → 5-6 classes

- Hard to change in-between a module
- next-module onwards

✓ Math for ML

- case-studies in sync with classes.



= Questions =

CLT

$[x_1, x_2 \ldots\ldots x_{75}]$

X : pop mean (μ) & std-dev (σ) are finite

Sample-means

Sample size of n

$\{x_1^1, x_2^1 \ldots\ldots x_n^1\}$ — sample$_1$ → M_1

$\{x_1^2, x_2^2 \ldots\ldots x_n^2\}$ — S2 → M_2

$\{x_1^K \ldots\ldots x_n^K\}$ — SK → MK

Sample means: $m_1 \dots m_k$

$$\text{Normal}\left(\text{mean} = \mu;\ \text{std-dev} = \frac{\sigma}{\sqrt{n}}\right)$$

as $n \to \infty$

Why CLT

X = rec. time of patients medicine 1

$\{x_1, \ldots x_{100}\}$

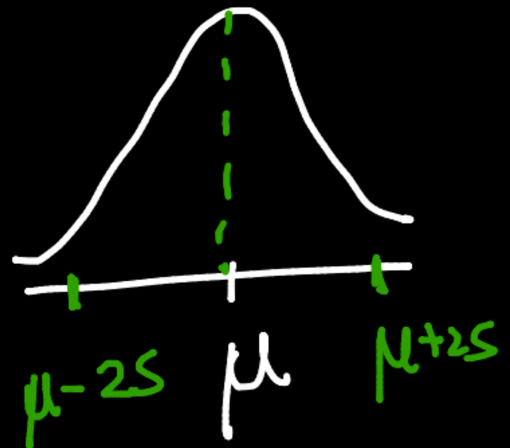
95% C.I on the mean rec. time

S1 — → M1

S2 — → M2

⋮ ⋮

Sk — → Mk



$s = \text{Std dev} \approx \frac{\sigma}{\sqrt{n}}$

95 C.I → bootstrapping : means; medians; P99

→ CLT : means ✓

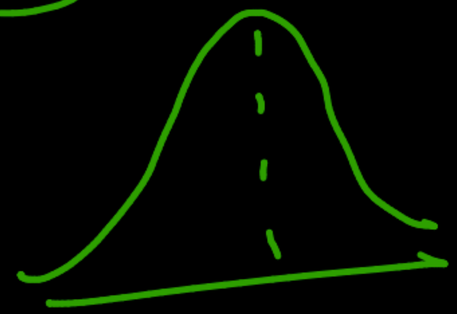
Q

95% C.I of mean

[μ, σ are finite]

pop →

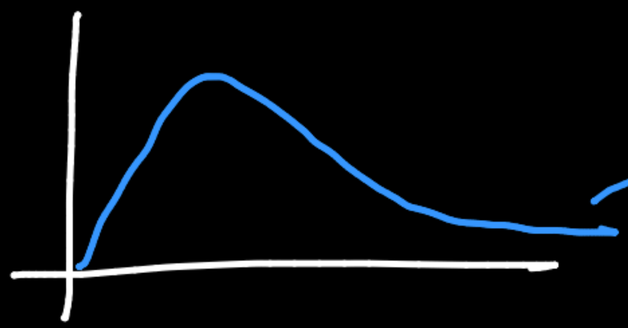
CLT :



disb. of sample-means

{ μ & σ not finite

↳ some real world → finance & trading



→ some long-tailed dist

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

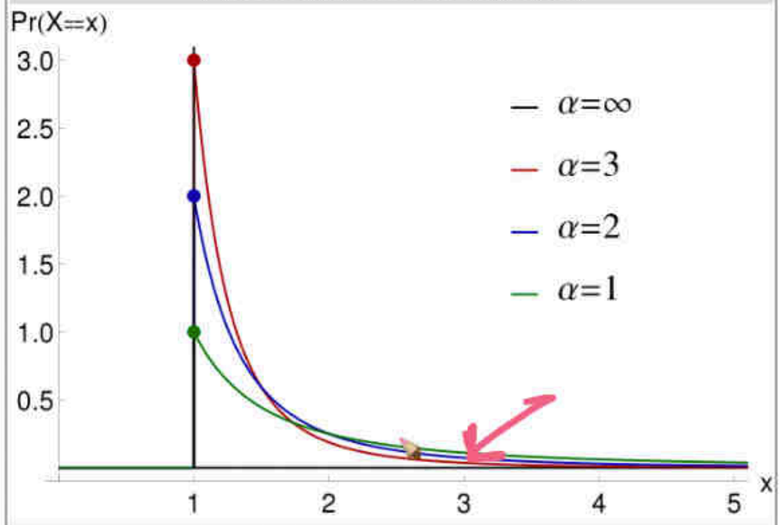
Print/export
Download as PDF
Printable version

The **Pareto distribution**, named after the Italian civil engineer, economist, and sociologist Vilfredo Pareto,[1] (Italian: [paˈreːto] US: /pəˈreɪtoʊ/ *pə-RAY-toh*),[2] is a power-law probability distribution that is used in description of social, quality control, scientific, geophysical, actuarial, and many other types of observable phenomena. Originally applied to describing the distribution of wealth in a society, fitting the trend that a large portion of wealth is held by a small fraction of the population.[3][4] The Pareto principle or "80-20 rule" stating that 80% of outcomes are due to 20% of causes was named in honour of Pareto, but the concepts are distinct, and only Pareto distributions with shape value (α) of $\log_4 5 \approx 1.16$ precisely reflect it. Empirical observation has shown that this 80-20 distribution fits a wide range of cases, including natural phenomena[5] and human activities.[6][7]

Contents [hide]

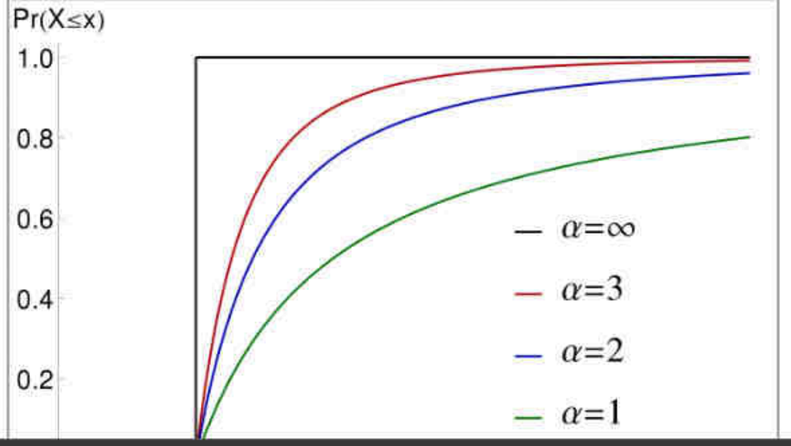
Pareto Type I

Probability density function



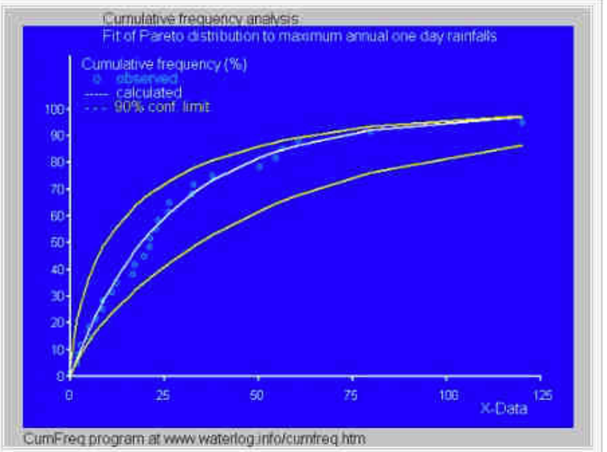
Pareto Type I probability density functions for various α with $x_m = 1$. As $\alpha \to \infty$, the distribution approaches $\delta(x - x_m)$, where δ is the Dirac delta function.

Cumulative distribution function



way that a larger portion of the wealth of any society is owned by a smaller percentage of the peopl o describe distribution of income.[4] This idea is sometimes expressed more simply as the Pareto principle or the "80-20 rule" which says that 20% of the population controls 80% of the wealth.[23] However, the 80-20 rule corresponds to a particular value of α, and in fact, Pareto's data on British income taxes in his *Cours d'économie politique* indicates that about 30% of the population had about 70% of the income. [*citation needed*] The probability density function (PDF) graph at the beginning of this article shows that the "probability" or fraction of the population that owns a small amount of wealth per person is rather high, and then decreases steadily as wealth increases. (The Pareto distribution is not realistic for wealth for the lower end, however. In fact, net worth may even be negative.) This distribution is not limited to describing wealth or income, but to many situations in which an equilibrium is found in the distribution of the "small" to the "large". The following examples are sometimes seen as approximately Pareto-distributed:

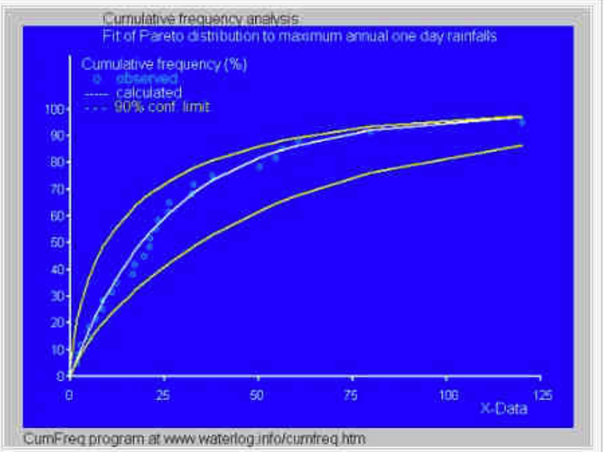
- The sizes of human settlements (few cities, many hamlets/villages)[24][25]
- File size distribution of Internet traffic which uses the TCP protocol (many smaller files, few larger ones)[24]
- Hard disk drive error rates[26]
- Clusters of Bose–Einstein condensate near absolute zero[27]
- The values of oil reserves in oil fields (a few large fields, many small fields)[24]
- The length distribution in jobs assigned to supercomputers (a few large ones, many small ones)[28]
- The standardized price returns on individual stocks [24]
- Sizes of sand particles [24]
- The size of meteorites
- Severity of large casualty losses for certain lines of business such as general liability, commercial auto, and workers compensation.[29][30]
- Amount of time a user on Steam will spend playing different games. (Some games get played a lot, but most get played almost never.) [2][*original research?*]



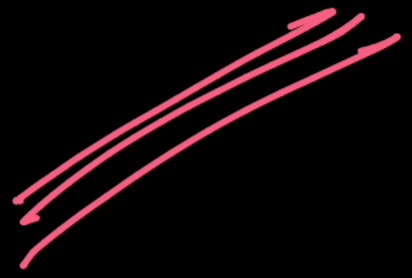
Fitted cumulative Pareto (Lomax)

way that a larger portion of the wealth of any society is owned by a smaller percentage of the peopl... o describe distribution of income.[4] This idea is sometimes expressed more simply as the Pareto principle or the "80-20 rule" which says that 20% of the population controls 80% of the wealth.[23] However, the 80-20 rule corresponds to a particular value of α, and in fact, Pareto's data on British income taxes in his *Cours d'économie politique* indicates that about 30% of the population had about 70% of the income.[*citation needed*] The probability density function (PDF) graph at the beginning of this article shows that the "probability" or fraction of the population that owns a small amount of wealth per person is rather high, and then decreases steadily as wealth increases. (The Pareto distribution is not realistic for wealth for the lower end, however. In fact, net worth may even be negative.) This distribution is not limited to describing wealth or income, but to many situations in which an equilibrium is found in the distribution of the "small" to the "large". The following examples are sometimes seen as approximately Pareto-distributed:

- The sizes of human settlements (few cities, many hamlets/villages)[24][25]
- File size distribution of Internet traffic which uses the TCP protocol (many smaller files, few larger ones)[24]
- Hard disk drive error rates[26]
- Clusters of Bose–Einstein condensate near absolute zero[27]
- The values of oil reserves in oil fields (a few large fields, many small fields)[24]
- The length distribution in jobs assigned to supercomputers (a few large ones, many small ones)[28]
- The standardized price returns on individual stocks [24]
- Sizes of sand particles [24]
- The size of meteorites
- Severity of large casualty losses for certain lines of business such as general liability, commercial auto, and workers compensation.[29][30]
- Amount of time a user on Steam will spend playing different games. (Some games get played a lot, but most get played almost never.) [2][*original research?*]



Fitted cumulative Pareto (Lomax)



Non-parametric stats

Bootstrapping

- we made no assumptions about underlying population

Parametric stats

CLT

- some assumptions about the population

CLT: (μ & σ are finite)

```
-rw-r--r-- 1 root root  18251707 May 12 16:03 uber_dataset.zip
drwxr-xr-x 2 root root      4096 May 12 16:04 __MACOSX
```

```
import pandas as pd

df = pd.read_csv("./uber_travel_data.csv")
df.sample(100).head()
```

	sourceid	source	dstid	destination	travel_time
2936366	187	60, Vijay Vihar Phase II, Pocket C, Sector 1, ...	77	0 Mehrauli - Badarpur Road, Tughlakabad Instit...	4987
982706	63	Jawaharlal Nehru Stadium Marg, CGO Complex, Pr...	172	nullShiva Road, Pocket 14, Sector 8D, Rohini, ...	3093
3711028	234	113, Press Colony, Press Colony, Mayapuri, New...	259	NaN	2328
114690	7	P 13, Baird Place, Delhi Cantonment, New Delhi	131	4400 Gali Bahuji, Pahari Dhiraj, Sadar Bazaar,...	1840
2886810	183	NaN	243	NaN	2715

```
df.shape
```

```
-rw-r--r-- 1 root root  18251707 May 12 16:03 Uber_dataset.zip
drwxr-xr-x 2 root root      4096 May 12 16:04 __MACOSX
```

```
import pandas as pd

df = pd.read_csv("./uber_travel_data.csv")
df.sample(100).head()
```

	sourceid	source	dstid	destination	travel_time
2936366	187	60, Vijay Vihar Phase II, Pocket C, Sector 1, ...	77	0 Mehrauli - Badarpur Road, Tughlakabad Instit...	4987
982706	63	Jawaharlal Nehru Stadium Marg, CGO Complex, Pr...	172	nullShiva Road, Pocket 14, Sector 8D, Rohini, ...	3093
3711028	234	113, Press Colony, Press Colony, Mayapuri, New...	259	NaN	2328
114690	7	P 13, Baird Place, Delhi Cantonment, New Delhi	131	4400 Gali Bahuji, Pahari Dhiraj, Sadar Bazaar,...	1840
2886810	183	NaN	243	NaN	2715

```
[ ] df.shape
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:36

QQ-plot and CLT and Bootstra | Hypothesis Testing.ipynb - Co | QQ_plot_and_CLT_and_Boots | Pareto distribution - Wikipedia

colab.research.google.com/drive/1m1Wa-7i9WknfRE9Y_aoi_5cbm5AAJBjW#scrollTo=M8Qmeb2C8lcH

+ Code + Text RAM Disk

[6] 7s

114690	7	P 13, Baird Place, Delhi Cantonment, New Delhi	131	4400 Gali Bahuji, Pahari Dhiraj, Sadar Bazaar,...		1840
2886810	183	NaN	243	NaN		2715

```
df.shape
```

0s

```
(4542026, 5)
```

```
# histogram of travel_times
df["travel_time"].hist(bins = 100)
```

```
df.value_counts(['sourceid', 'dstid']).sort_values()
```

```
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
data.shape
```

```
data.hist(bins = 100)
```

CLT for C.I on mean of travel_time

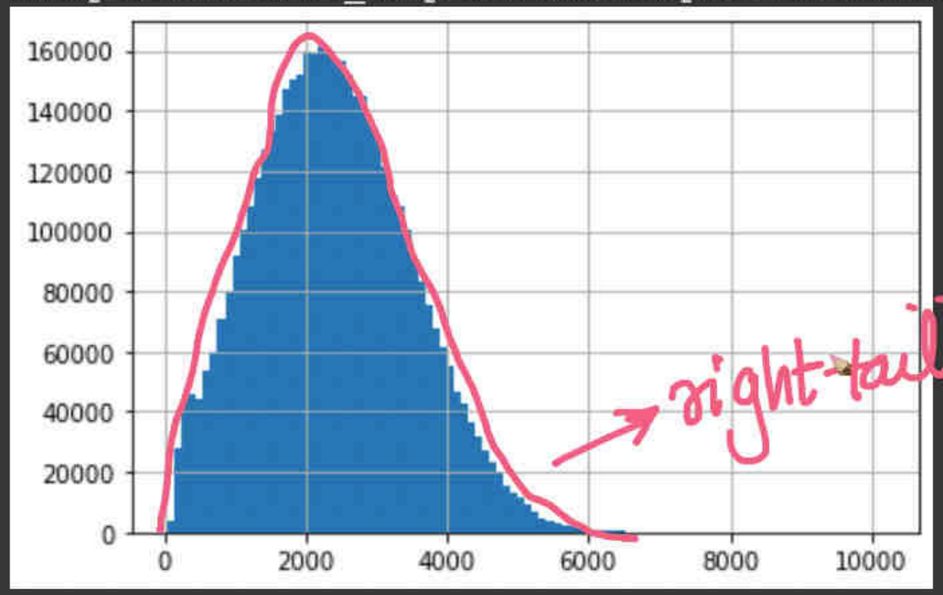
15 / 15

```
[7] df.shape
```

```
(4542026, 5)
```

```
# histogram of travel_times
df["travel_time"].hist(bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05a867c150>
```



4.5 M

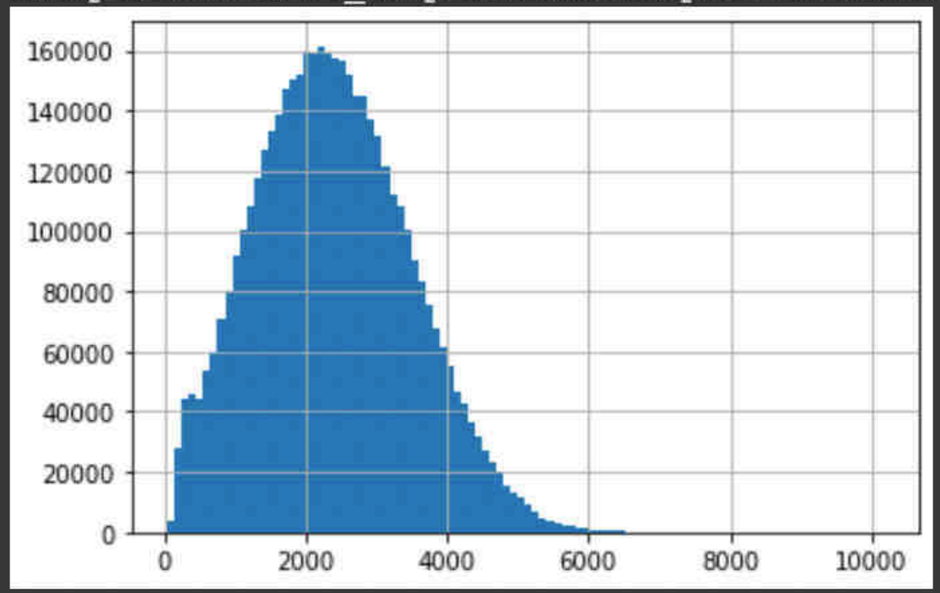
```
[ ] df.value_counts(['sourceid', 'dstid']).sort_values()
```

```
[7] df.shape
```

```
(4542026, 5)
```

```
# histogram of travel_times
df["travel_time"].hist(bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05a867c150>
```



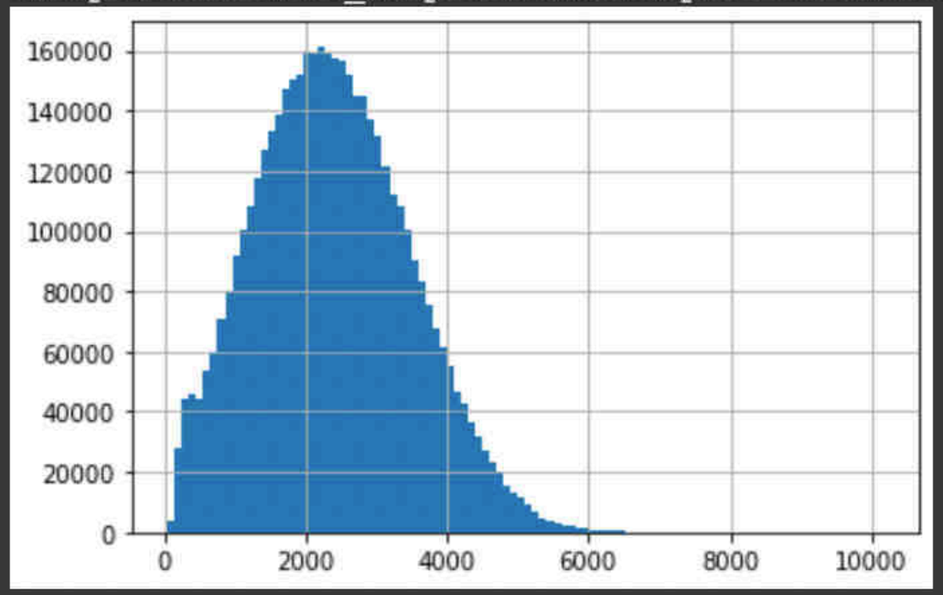
finite μ & σ : population

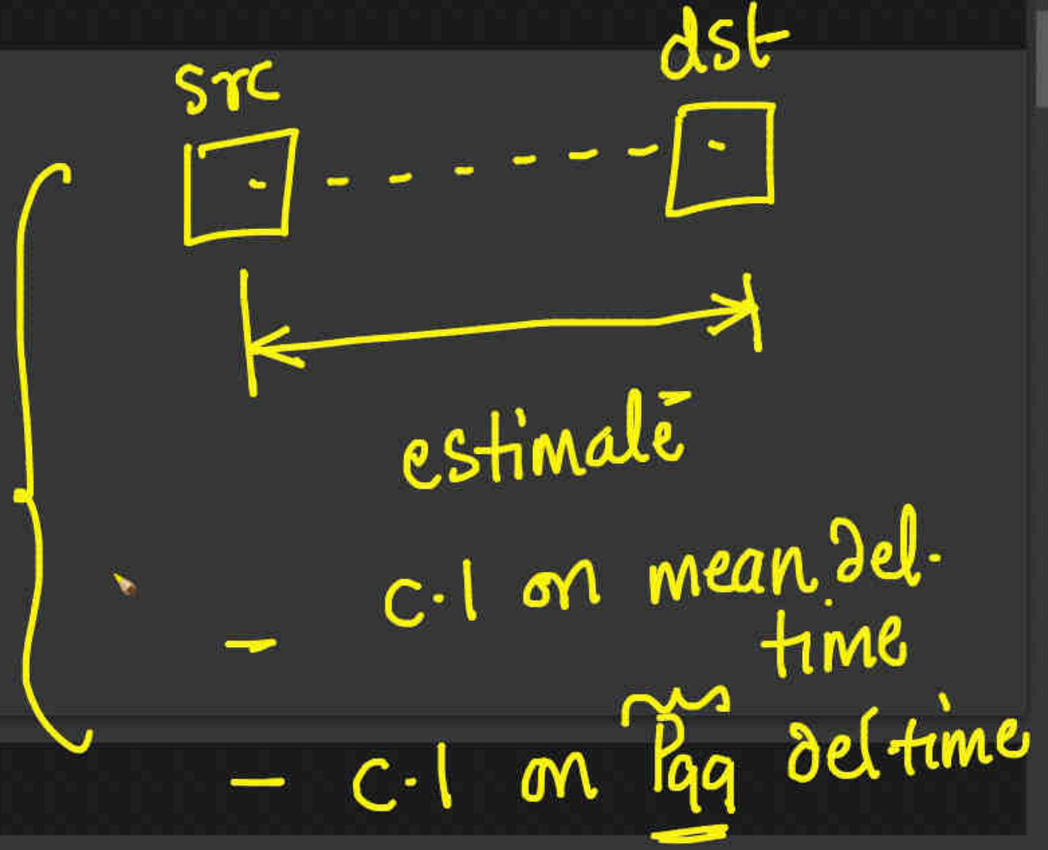
```
[ ] df.value_counts(['sourceid', 'dstid']).sort_values()
```

```
[7] df.shape

(4542026, 5)
```

```
# histogram of travel_times
df["travel_time"].hist(bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05a867c150>
```





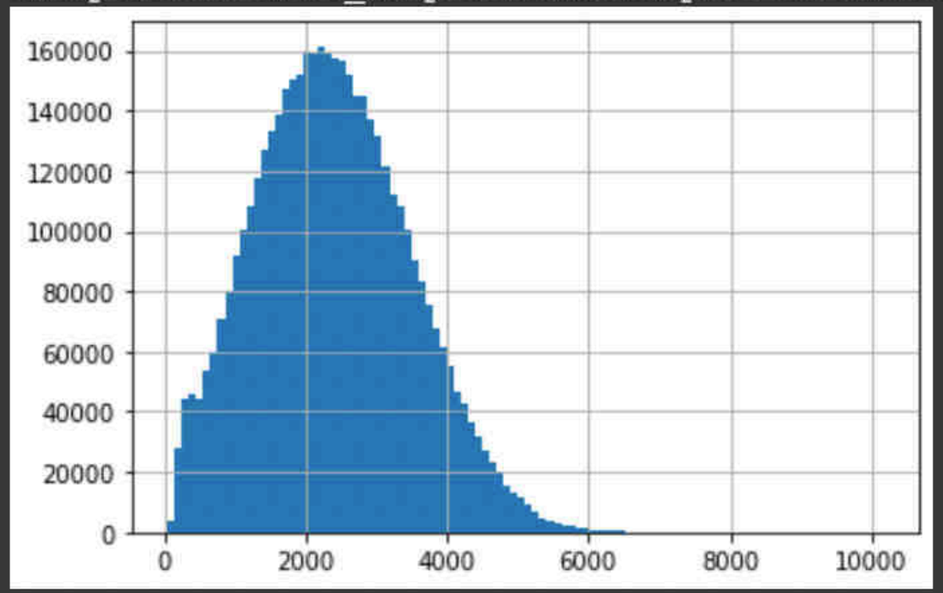

```
[ ] df.value_counts(['sourceid', 'dstid']).sort_values()
```

[7] 0s
```
df.shape
```

```
(4542026, 5)
```

0s
```
# histogram of travel_times
df["travel_time"].hist(bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05a867c150>
```

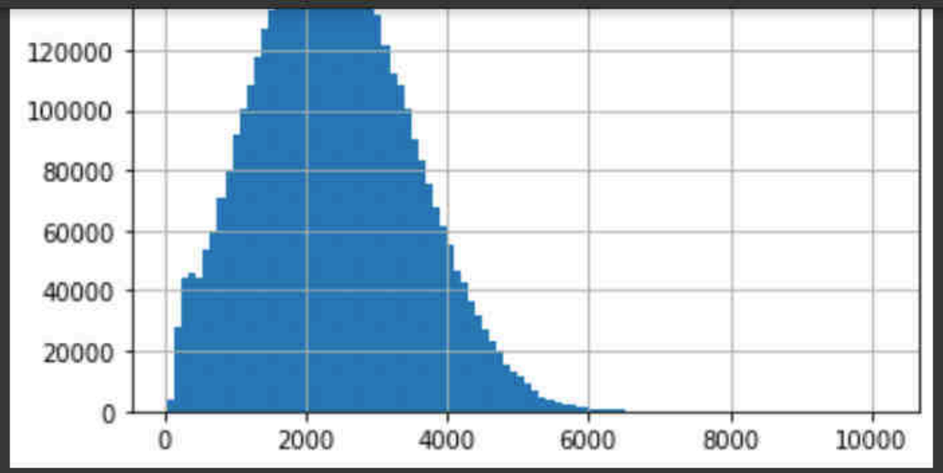


4.5 Million del-times

approx it as pop-dist

[ ]
```
df.value_counts(['sourceid', 'dstid']).sort_values()
```

[8]



4.5 M rows

```
df.value_counts(['sourceid', 'dstid']).sort_values()
```

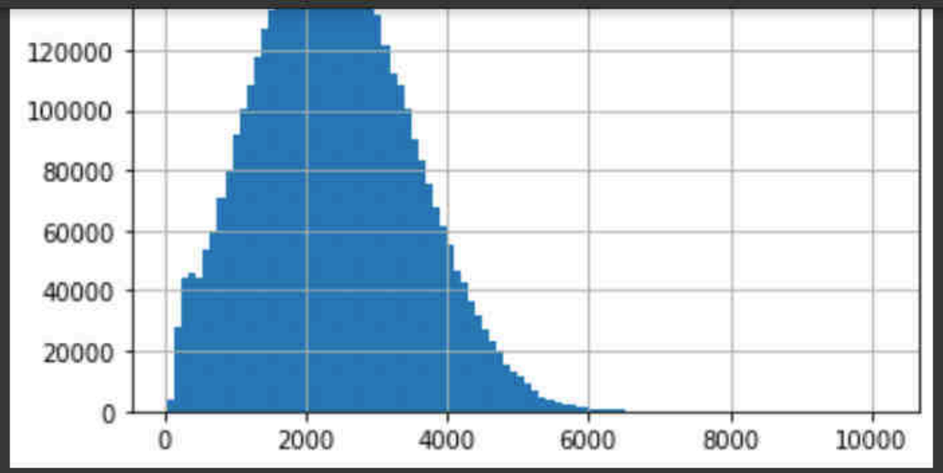
```
sourceid  dstid
69        4        50
167       107      50
          101      50
264       14       50
167       100      50
                   ..
83        88       79
244       32       79
202       201      79
          135      79
45        170      79
Length: 70429, dtype: int64
```

Src

Dest

50 - 79 data points

[8]




```
df.value_counts(['sourceid', 'dstid']).sort_values()
```

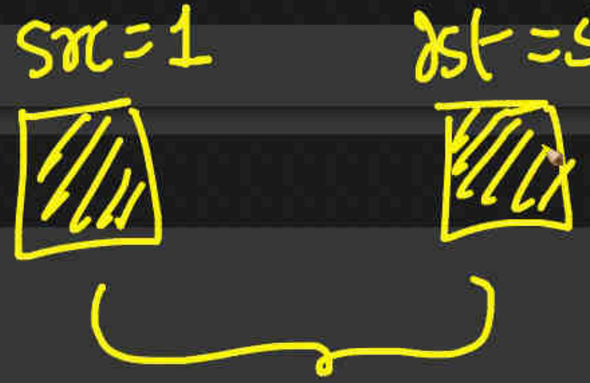
```
sourceid  dstid
69        4        50
167       107      50
          101      50
264       14       50
167       100      50
                   ..
83        88       79
244       32       79
202       201      79
          135      79
45        170      79
Length: 70429, dtype: int64
```

```
45          170       79
Length: 70429, dtype: int64
```

```
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)]["travel_time"]
data.shape
```

```
(75,)
```

```
data.hist(bins = 100)
```



CLT for C.I on mean of travel_time

6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

7 cells hidden

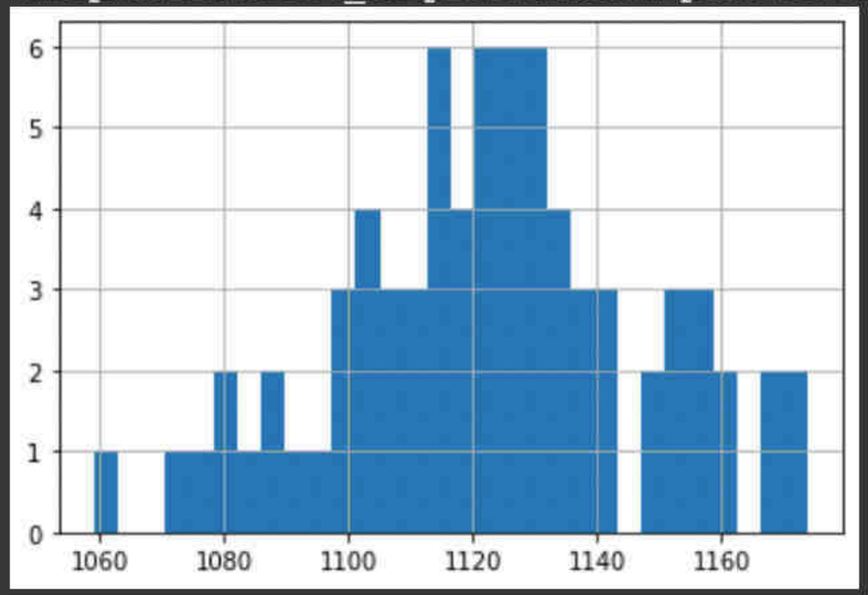
box-cox transform

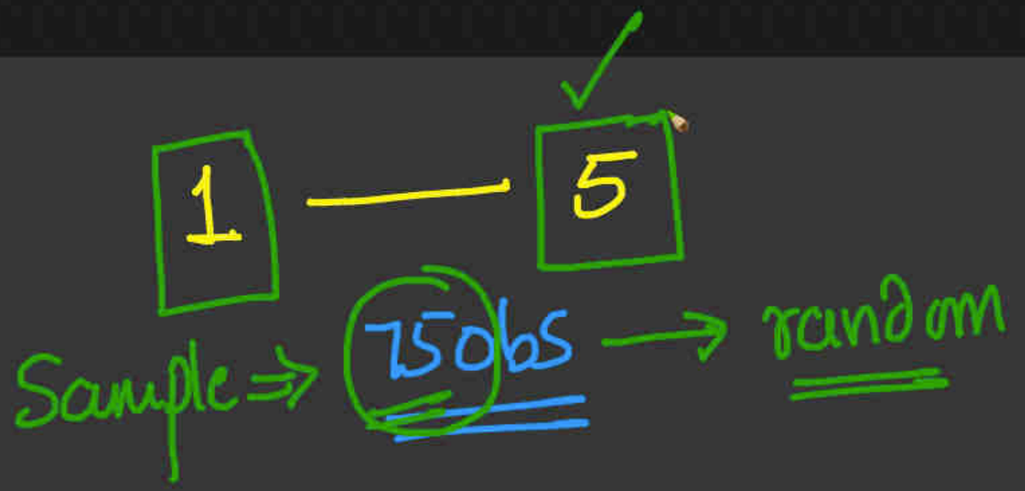
```
[10] data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     data.shape
```

```
(75,)
```

```
data.hist(bins=30)
```

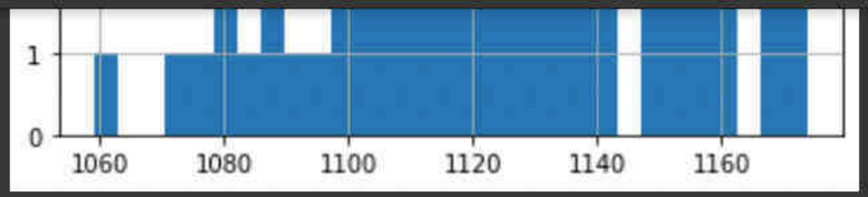
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05a5fd56d0>
```





CLT for C.I on mean of travel_time

[13] 1s



finite μ & σ

CLT for C.I on mean of travel_time

```
[ ]  # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
     # bs_means is a list of 'r' bootstrap sample means
     r = 10000
     data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     size = 50
     bs_means = np.empty(r)

     for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_means[i] = np.mean(bs_sample)
```

```
[ ]  import matplotlib.pyplot as plt
     plt.figure()
     plt.hist(bs_means, bins=100)
     plt.grid()
     plt.show()
```

CLT for C.I on mean of travel_time

```
[ ]  # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
     # bs_means is a list of 'r' bootstrap sample means
     r = 10000
     data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     size = 50
     bs_means = np.empty(r)

     for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

CLT for C.I on mean of travel_time

```
[ ] # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
    # bs_means is a list of 'r' bootstrap sample means
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 50
    bs_means = np.empty(r)

    for i in range(r):
        bs_sample = np.random.choice(data, size=size)
        bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

CLT for C.I on mean of travel_time

75 obs

```
[ ]  # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
     # bs_means is a list of 'r' bootstrap sample means
     r = 10000
     data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     size = 50
     bs_means = np.empty(r)

     for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_means[i] = np.mean(bs_sample)
```

50 = n = sample-size

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

CLT for C.I on mean of travel_time

```
[ ] # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
    # bs_means is a list of 'r' bootstrap sample means
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 50
    bs_means = np.empty(r)

    for i in range(r):
        bs_sample = np.random.choice(data, size=size)
        bs_means[i] = np.mean(bs_sample)
```

M_1

M_2

$M_{10,000}$

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:48

QQ-plot and CLT and Bootstra | Hypothesis Testing.ipynb - Co | QQ_plot_and_CLT_and_Boots | Pareto distribution - Wikipedia

colab.research.google.com/drive/1m1Wa-7i9WknfRE9Y_aoi_5cbm5AAJBjW#scrollTo=IcDaEZiY_Nj0

+ Code + Text

RAM
Disk

CLT for C.I on mean of travel_time

```
[ ]  # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
     # bs_means is a list of 'r' bootstrap sample means
     r = 10000
     data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     size = 50
     bs_means = np.empty(r)

     for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

29 / 29

CLT for C.I on mean of travel_time

sample-means ~ Gaussian

```
[ ]  # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
     # bs_means is a list of 'r' bootstrap sample means
     r = 10000
     data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
     size = 50
     bs_means = np.empty(r)

     for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_means[i] = np.mean(bs_sample)
```

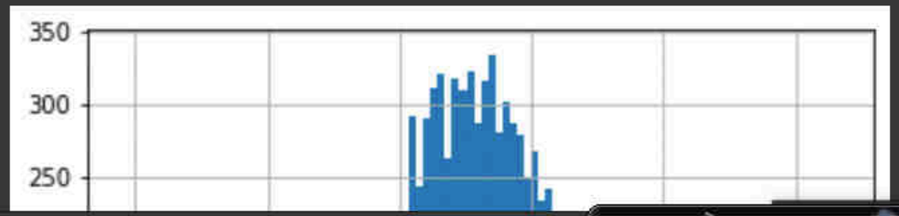
```
[ ]  import matplotlib.pyplot as plt
     plt.figure()
     plt.hist(bs_means, bins=100)
     plt.grid()
     plt.show()
```

CLT for C.I on mean of travel_time

```
[ ] # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
    # bs_means is a list of 'r' bootstrap sample means
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 50
    bs_means = np.empty(r)

    for i in range(r):
        bs_sample = np.random.choice(data, size=size)
        bs_means[i] = np.mean(bs_sample)
```

S_1
S_2
$\vdots$
S_Y

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```



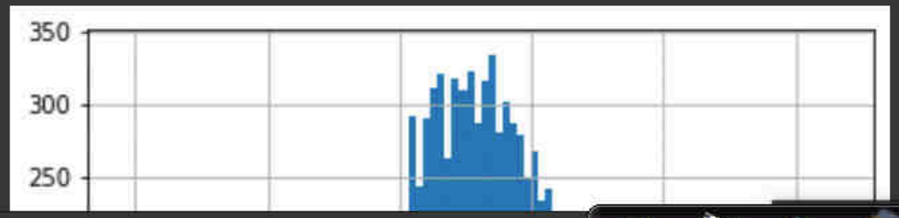
CLT for C.I on mean of travel_time

```
[ ] # Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
    # bs_means is a list of 'r' bootstrap sample means
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 50
    bs_means = np.empty(r)

    for i in range(r):
      bs_sample = np.random.choice(data, size=size)
      bs_means[i] = np.mean(bs_sample)
```

$S_1, S_2, \ldots S_r = 10{,}000$

M_1 $M_2 \ldots$ M_k

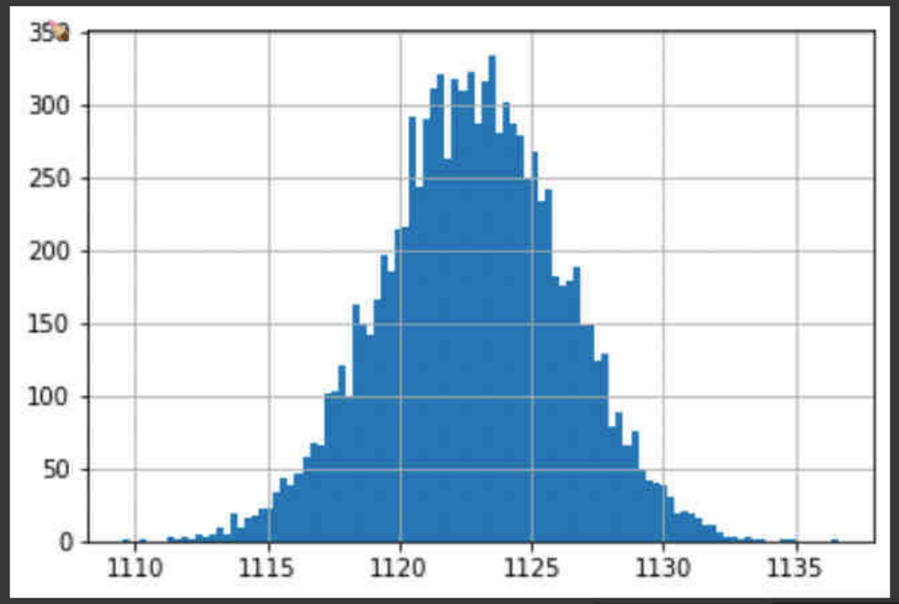
```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```



```
bs_means = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_means[i] = np.mean(bs_sample)
```

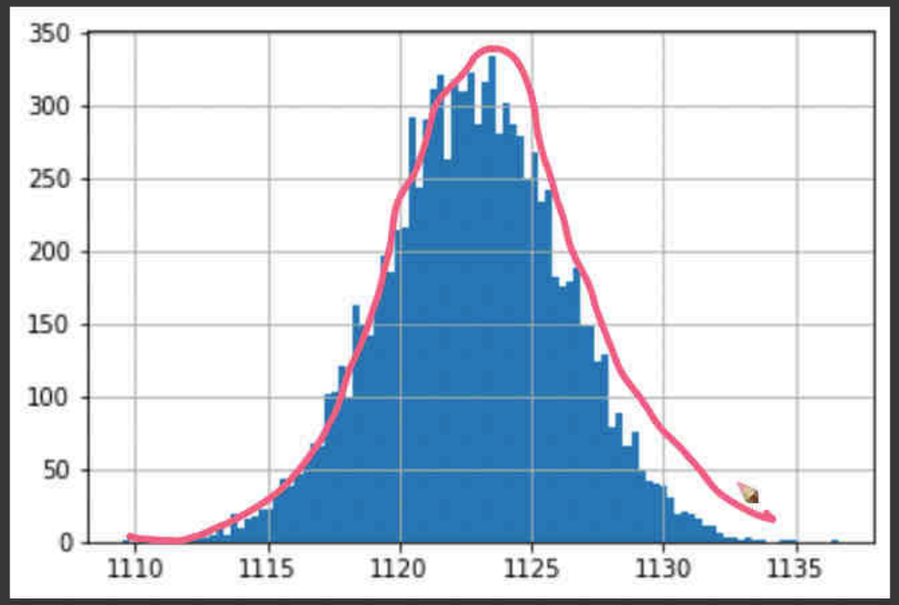
```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

$M_1, M_2 \dots M_k$



```
    bs_sample = np.random.choice(data, size=size)
    bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```




```
# QQ-plot with normal distribution
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:52

QQ-plot and CLT and Bootstra | Hypothesis Testing.ipynb - Co | QQ_plot_and_CLT_and_Boots | Pareto distribution - Wikipedia

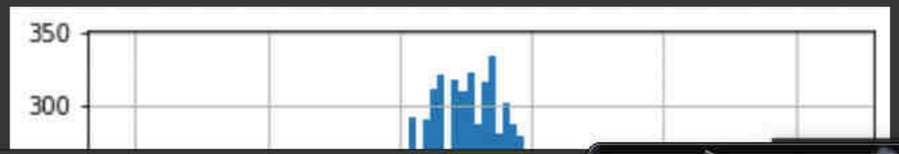
colab.research.google.com/drive/1m1Wa-7i9WknfRE9Y_aoi_5cbm5AAJBjW#scrollTo=-XRxzwtX8h5s

+ Code + Text RAM Disk

CLT for C.I on mean of travel_time

```
# Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
# bs_means is a list of 'r' bootstrap sample means
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 50
bs_means = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

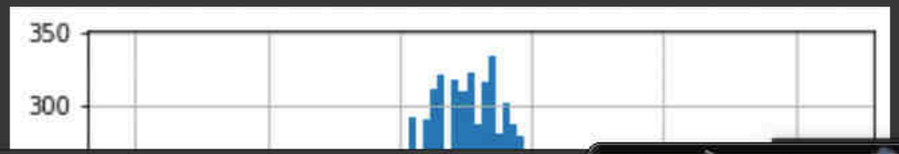


35 / 35

CLT for C.I on mean of travel_time

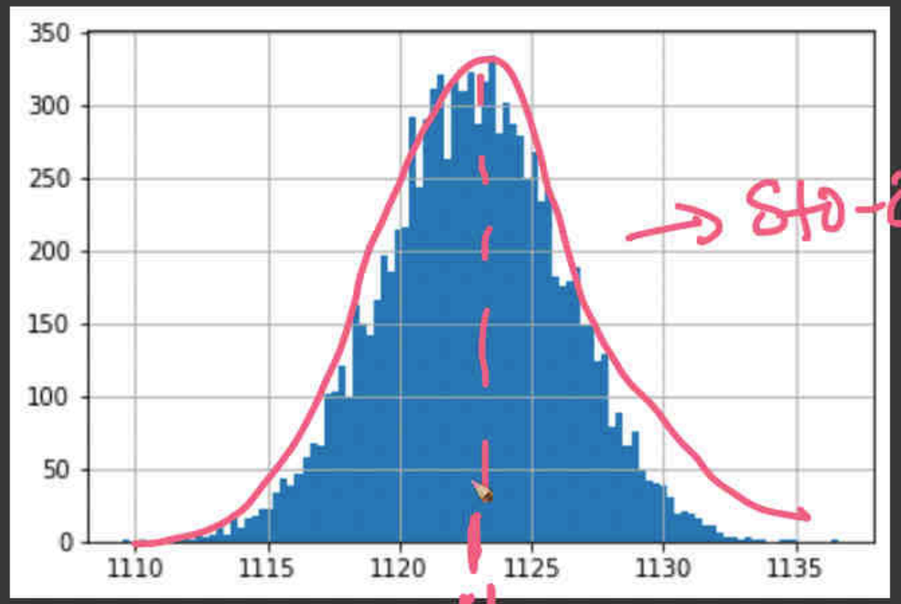
```
# Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=50
# bs_means is a list of 'r' bootstrap sample means
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 50
bs_means = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_means[i] = np.mean(bs_sample)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```



CLT:
Sample-means
$\sim N(\mu; \frac{\sigma}{\sqrt{n}})$
as $n \to \infty$

```
plt.hist(bs_means, bins=100)
plt.grid()
plt.show()
```

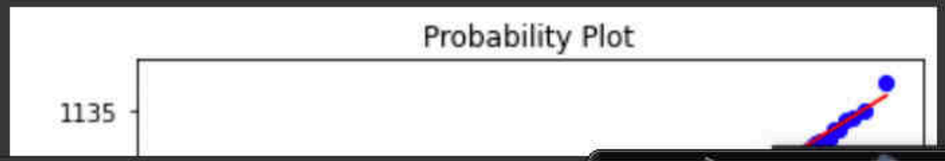


$\rightarrow$ Std-dev $= \frac{\sigma}{\sqrt{n}}$

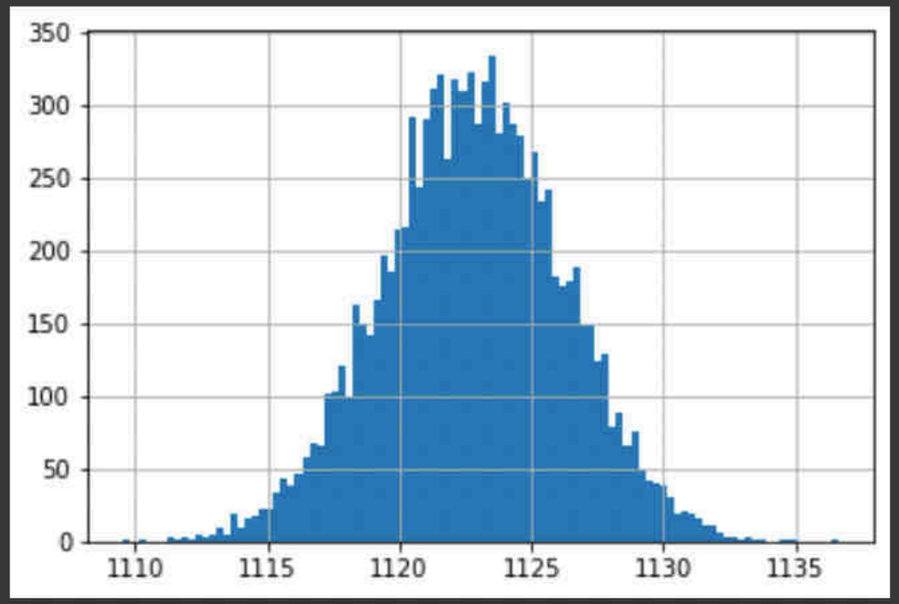
$n = 50$

μ

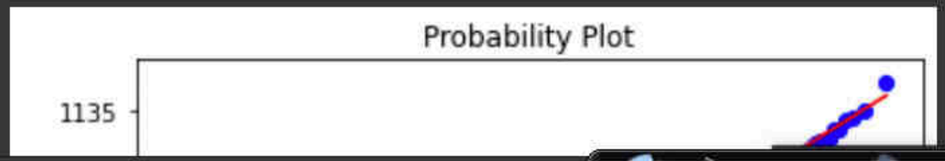
```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_means, dist=stats.norm, plot=ax1)
```



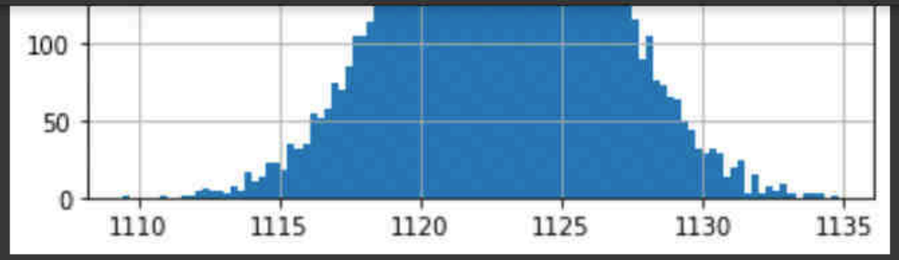
```
[ ] plt.hist(bs_means, bins=100)
    plt.grid()
    plt.show()
```



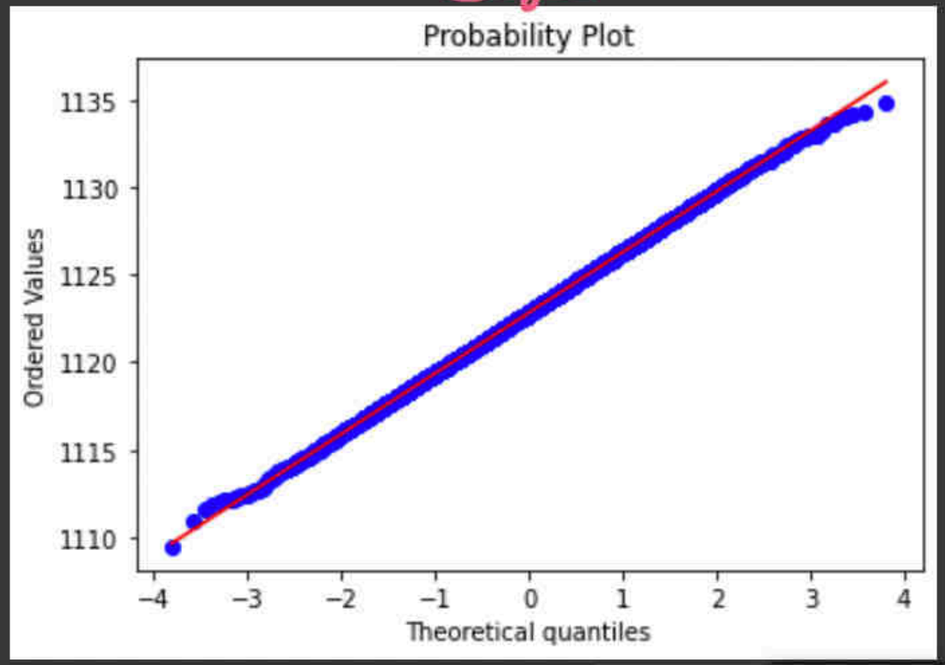

```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_means, dist=stats.norm, plot=ax1)
```



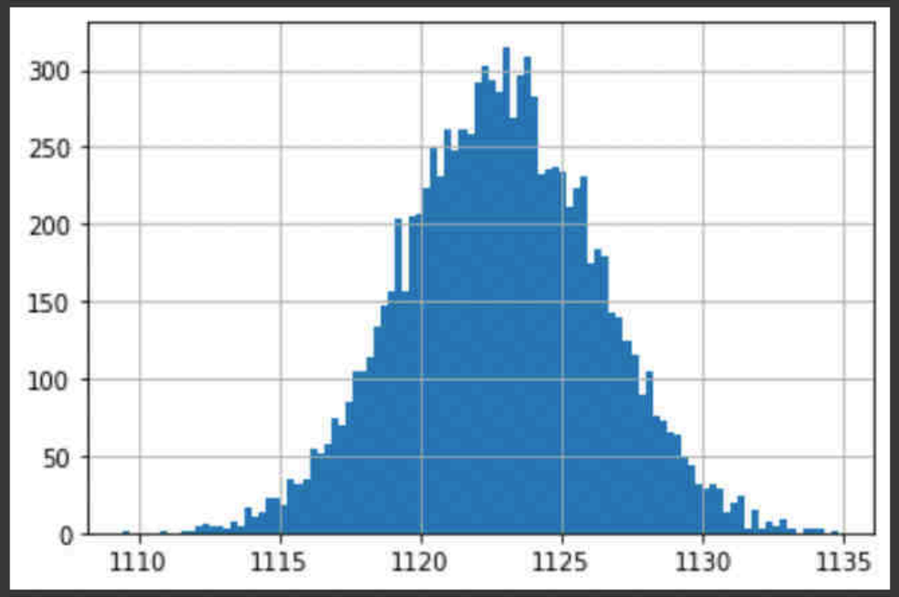
[17]




```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_means, dist=stats.norm, plot=ax1)
```



```
[17] import matplotlib.pyplot as plt
     plt.figure()
     plt.hist(bs_means, bins=100)
     plt.grid()
     plt.show()
```



disb of sample-means

```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_means, dist=stats.norm, plot=ax1)
```

Theoretical quantiles

```
# compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
print(np.mean(bs_means))
print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

95% C.I on mean

std-dev=3

```
print(np.mean(bs_means)-2*np.std(bs_means))
print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

95%

1 5


```
# could we just use the 2.5th percentile and 97.5th percentile value
print(np.percentile(bs_means,2.5))
print(np.percentile(bs_means,97.5))

# what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

Theoretical quantiles

```
# compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
print(np.mean(bs_means))
print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

```
print(np.mean(bs_means)-2*np.std(bs_means))
print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

99% C.I ✓

68-95-99 rule ✓

```
# could we just use the 2.5th percentile and 97.5th percentile value
print(np.percentile(bs_means,2.5))
print(np.percentile(bs_means,97.5))

# what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

Theoretical quantiles

```
# compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
print(np.mean(bs_means))
print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

```
print(np.mean(bs_means)-2*np.std(bs_means))
print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
# could we just use the 2.5th percentile and 97.5th percentile value
print(np.percentile(bs_means,2.5))
print(np.percentile(bs_means,97.5))

# what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

Theoretical quantiles

```
# compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
print(np.mean(bs_means))
print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

95 C.I on the mean (CLT)

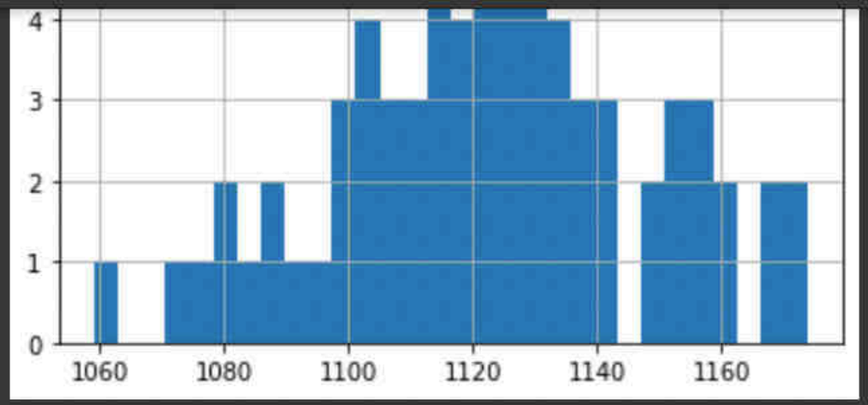
```
print(np.mean(bs_means)-2*np.std(bs_means))
print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
# could we just use the 2.5th percentile and 97.5th percentile value
print(np.percentile(bs_means,2.5))
print(np.percentile(bs_means,97.5))

# what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

[13]



X Normal mean = 0
std = 1

↓

std. normal dist

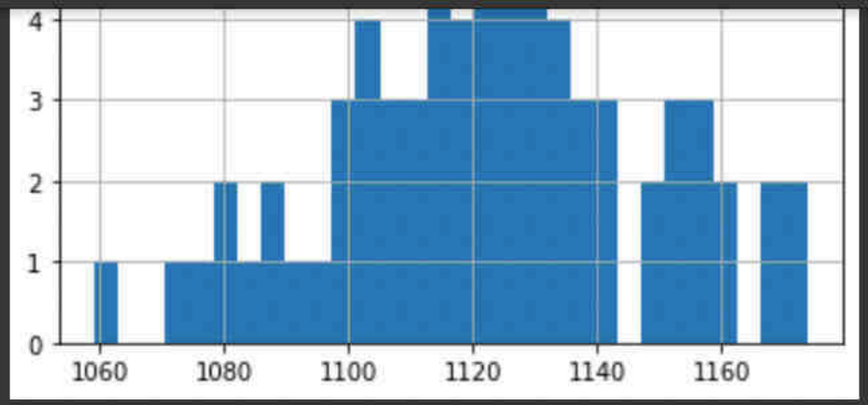
CLT for C.I on mean of travel_time

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

```
[ ] # What if we want a C.I on the 99th precentile?
    #Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
    # bs_99p is a list of 'r' bootstrap sample's 99th percentiles
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 75
```

[13]



$H \sim Normal(150; 20)$ ✓

Z → $Normal(0,1)$ ← std. normal dist

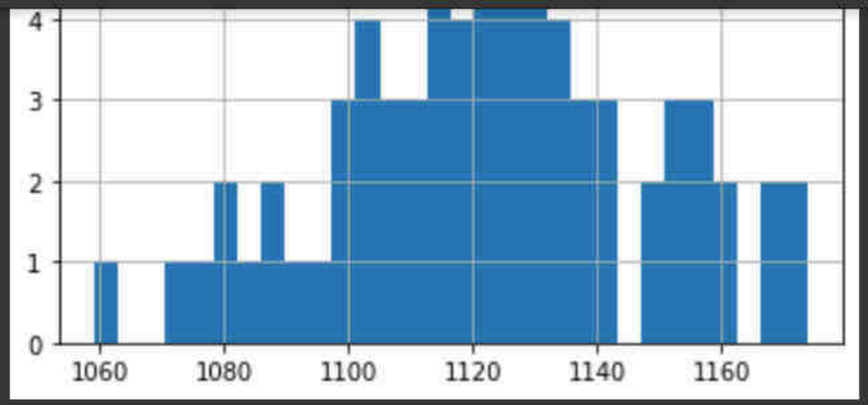
CLT for C.I on mean of travel_time

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

```
[ ] # What if we want a C.I on the 99th precentile?
    #Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
    # bs_99p is a list of 'r' bootstrap sample's 99th percentiles
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 75
```

[13]



CLT for C.I on mean of travel_time

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

```
[ ] # What if we want a C.I on the 99th precentile?
    #Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
    # bs_99p is a list of 'r' bootstrap sample's 99th percentiles
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 75
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

QQ-plot and CLT and Bootstra | Hypothesis Testing.ipynb - Co | QQ_plot_and_CLT_and_Boots | Pareto distribution - Wikipedia

colab.research.google.com/drive/1m1Wa-7i9WknfRE9Y_aoi_5cbm5AAJBjW#scrollTo=IcDaEZiY_Nj0

+ Code + Text

RAM Disk

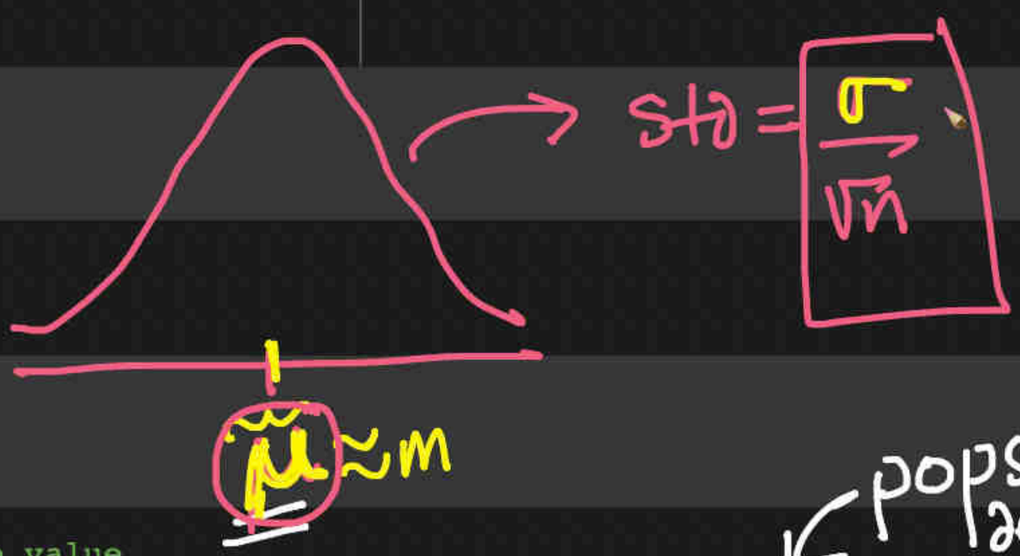
```
[19] # compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
     print(np.mean(bs_means))
     print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

m = 1122.823726
s = 3.460407709638272

10,000

CLT




```
[ ] print(np.mean(bs_means)-2*np.std(bs_means))
    print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
[ ] # could we just use the 2.5th percentile and 97.5th percentile value
    print(np.percentile(bs_means,2.5))
    print(np.percentile(bs_means,97.5))

    # what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

{ σ/√n : sto·error }

s = 3.46 = σ/√n (σ: popsto dev)

s × √n ≈ σ

▸ 95% C.I on 99th percentile value for travel times via bootstrapping

48 / 49

```
[19] # compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
     print(np.mean(bs_means))
     print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

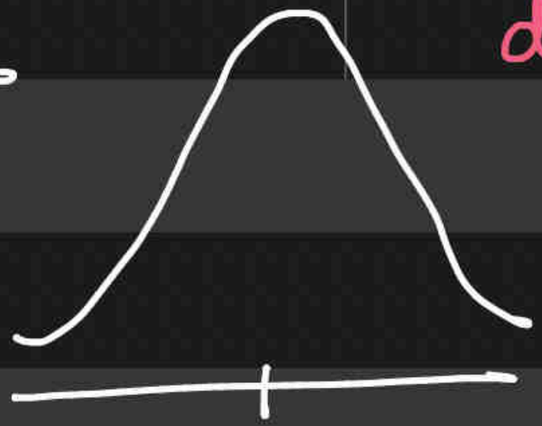
```
[ ] print(np.mean(bs_means)-2*np.std(bs_means))
    print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
[ ] # could we just use the 2.5th percentile and 97.5th percentile value
    print(np.percentile(bs_means,2.5))
    print(np.percentile(bs_means,97.5))

    # what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

n is large



disb of Sample-means

$$\begin{cases} \mu \approx m \\ \sigma = std * \sqrt{n} \end{cases}$$

95% C.I on 99th percentile value for travel time via bootstrapping

```
[19] # compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
     print(np.mean(bs_means))
     print(np.std(bs_means))
```

```
1122.823726
3.460407709638272
```

biased estimates

```
[ ] print(np.mean(bs_means)-2*np.std(bs_means))
    print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
[ ] # could we just use the 2.5th percentile and 97.5th percentile value
    print(np.percentile(bs_means,2.5))
    print(np.percentile(bs_means,97.5))

    # what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

95% C.I on 99th percentile value for travel time via bootstrapping

```
[19] # compute C.I on the mean given that bs_means follows Gaussian distribution: CLT
     print(np.mean(bs_means))
     print(np.std(bs_means))
```

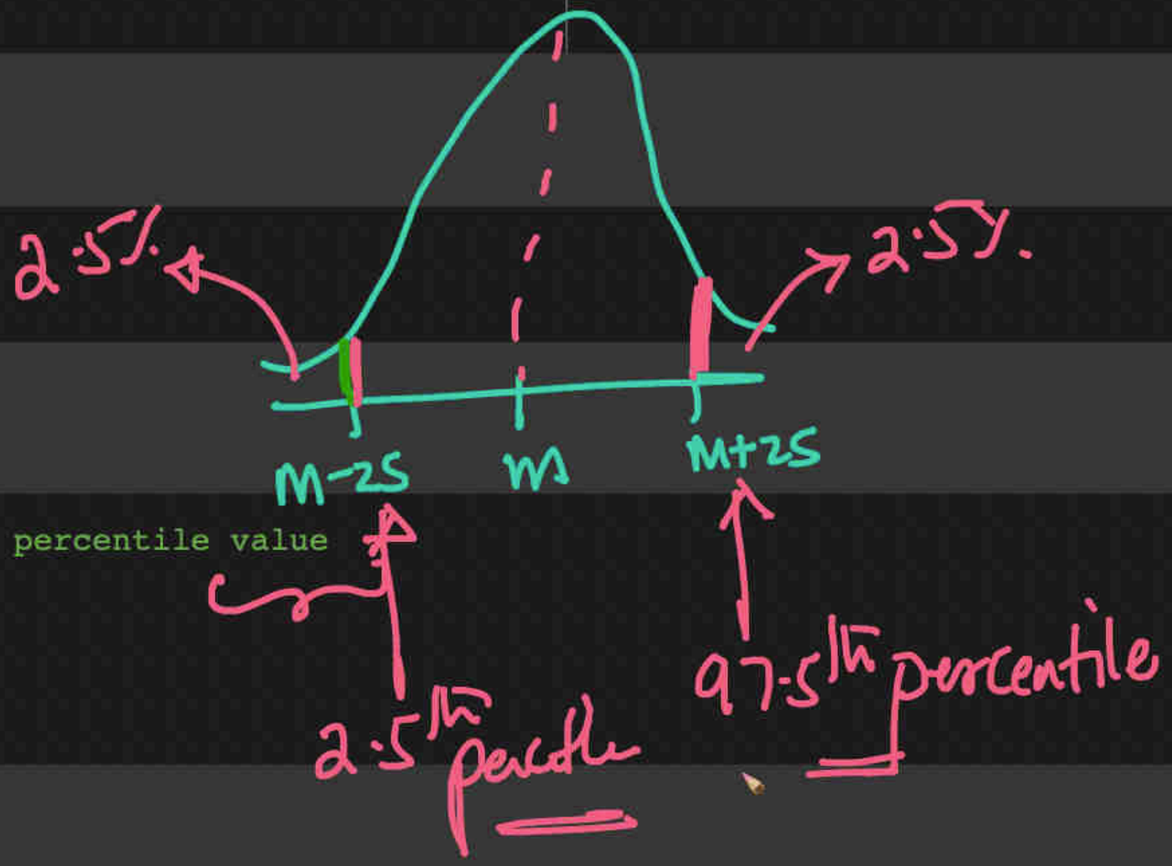
```
1122.823726
3.460407709638272
```

```
[ ] print(np.mean(bs_means)-2*np.std(bs_means))
    print(np.mean(bs_means)+2*np.std(bs_means))
```

```
1115.9224421865092
1129.6414778134913
```

```
[ ] # could we just use the 2.5th percentile and 97.5th percentile value
    print(np.percentile(bs_means,2.5))
    print(np.percentile(bs_means,97.5))

    # what if r is say 100 and not 10,000?
```

```
1115.98
1129.4205000000002
```

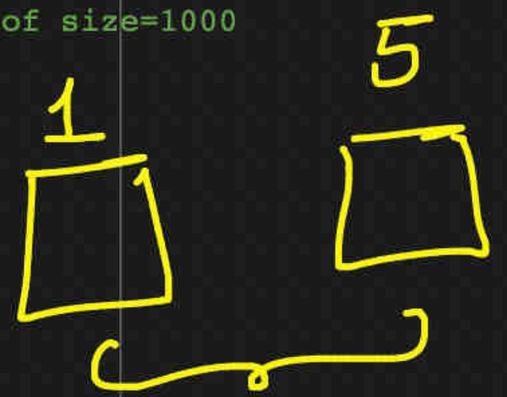


95% C.I on 99th percentile value for travel time via bootstrapping

6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

```
[ ] # What if we want a C.I on the 99th precentile?
    #Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
    # bs_99p is a list of 'r' bootstrap sample's 99th percentiles
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 75
    bs_99p = np.empty(r)

    for i in range(r):
        bs_sample = np.random.choice(data, size=size)
        bs_99p[i] = np.percentile(bs_sample,99)
```




```
[ ] len(bs_99p)

    10000
```

95% C.I on P_{99} [l, u]

```
[ ] bs_99p
```

6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

P_{99} → CLT ✗

```
[ ] # What if we want a C.I on the 99th precentile?
    #Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
    # bs_99p is a list of 'r' bootstrap sample's 99th percentiles
    r = 10000
    data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
    size = 75
    bs_99p = np.empty(r)

    for i in range(r):
        bs_sample = np.random.choice(data, size=size)
        bs_99p[i] = np.percentile(bs_sample,99)
```

```
[ ] len(bs_99p)
```

```
10000
```

```
[ ] bs_99p
```

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

```
# What if we want a C.I on the 99th precentile?
#Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
# bs_99p is a list of 'r' bootstrap sample's 99th percentiles
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)]["travel_time"]
size = 75
bs_99p = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_99p[i] = np.percentile(bs_sample,99)
```

```
[ ] len(bs_99p)
```

```
10000
```

```
[ ] bs_99p
```

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

1s

```
# What if we want a C.I on the 99th precentile?
#Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
# bs_99p is a list of 'r' bootstrap sample's 99th percentiles
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 75
bs_99p = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_99p[i] = np.percentile(bs_sample,99)
```

[ ]
```
len(bs_99p)
```

```
10000
```

[ ]
```
bs_99p
```

[ ] ↳ 6 cells hidden

95% C.I on 99th percentile value for travel_time via bootsrapping

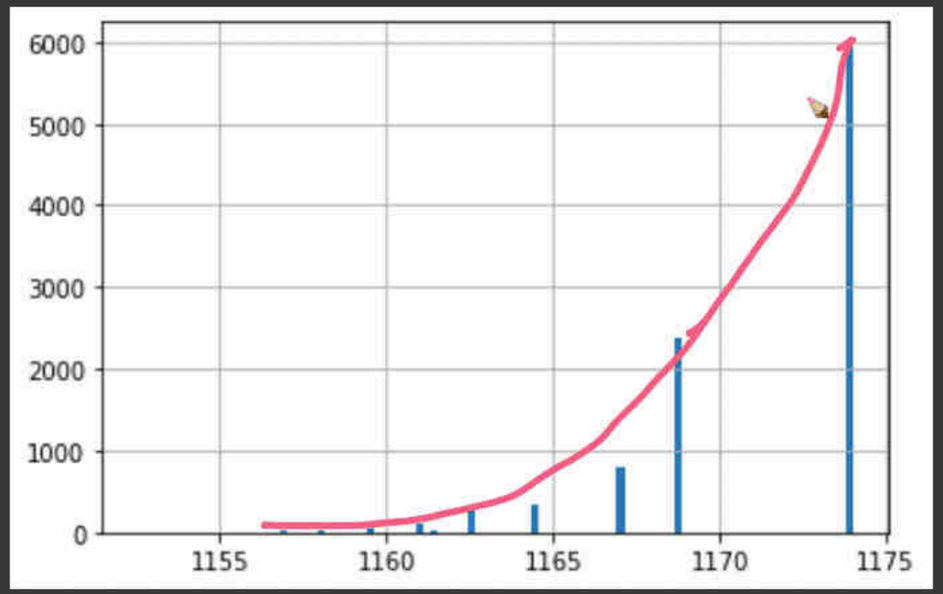
```
# What if we want a C.I on the 99th precentile?
#Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
# bs_99p is a list of 'r' bootstrap sample's 99th percentiles
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 75
bs_99p = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_99p[i] = np.percentile(bs_sample,99)
```

```
[ ] len(bs_99p)
```

```
10000
```

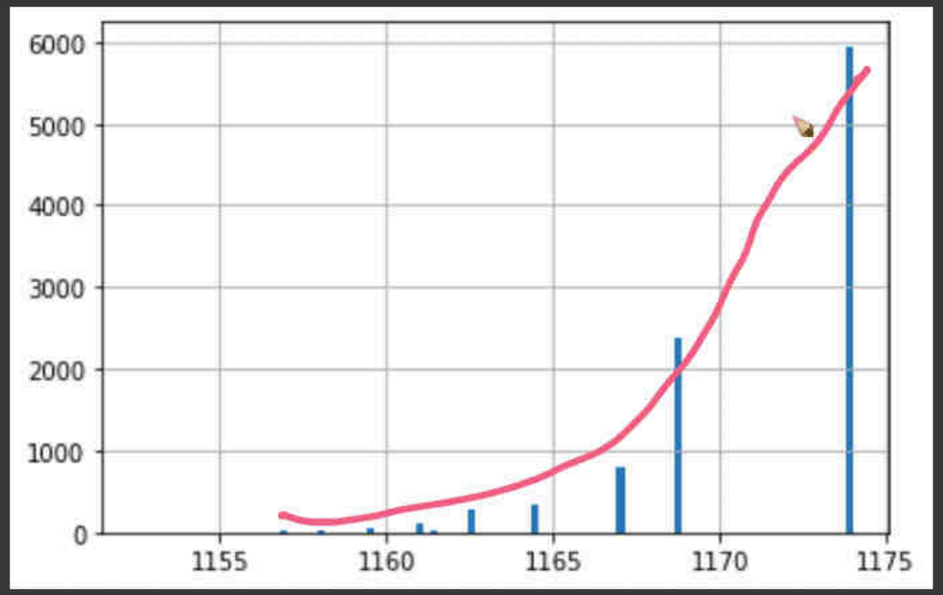
```
[ ] bs_99p
```

```
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```



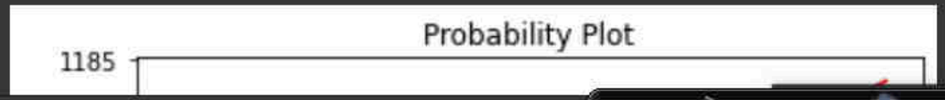

```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_99p, dist=stats.norm, plot=ax1)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```



disb of P99 != Gaussian

```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_99p, dist=stats.norm, plot=ax1)
```



```
[20] for i in range(r):
         bs_sample = np.random.choice(data, size=size)
         bs_99p[i] = np.percentile(bs_sample,99)
```

$x_1, x_2 \cdots x_{75}$

```
[21] len(bs_99p)
```

```
10000
```

↓ S1 with replacement

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

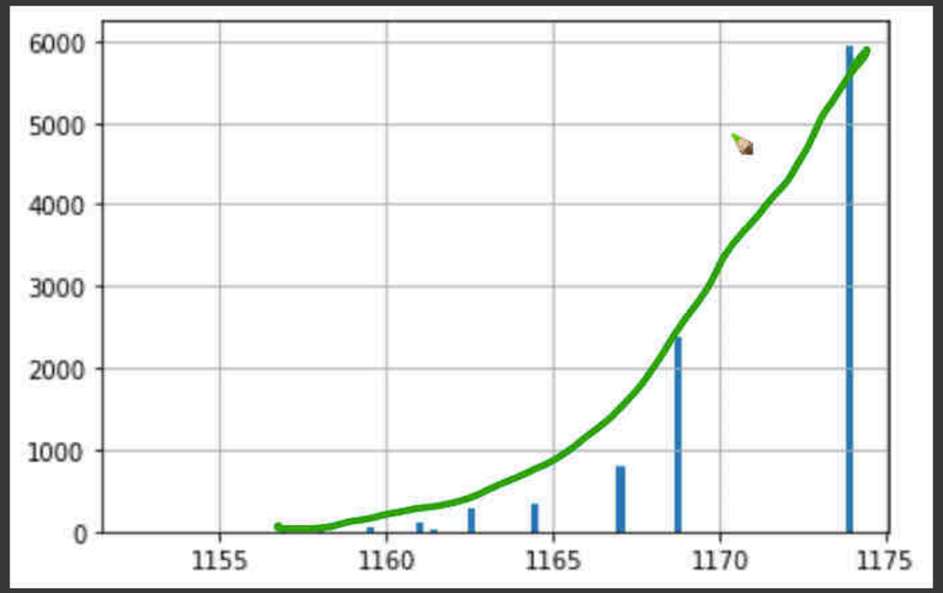
$x'_1 \cdots$

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
    print(np.percentile(data,99))
```

[22]
```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```




```
# QQ-plot with normal distribution
fig, ax1 = plt.subplots()
prob = stats.probplot(bs_99p, dist=stats.norm, plot=ax1)
```

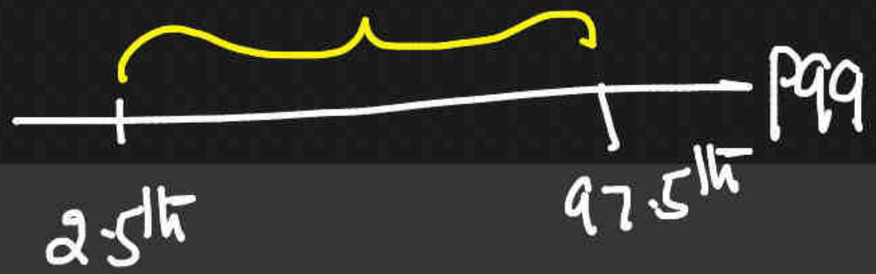
```
10000
```

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```




```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

P_{99}^{1-5} : [1162.56, 1174.0]

```
[22] # plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
```

```
10000
```

[23]
```
bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

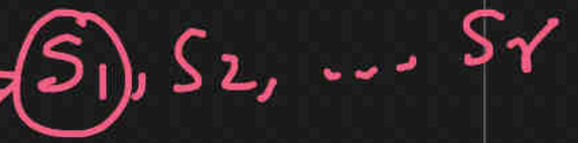
[ ]
```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

75

[22]
```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
```

```
# What if we want a C.I on the 99th precentile?
#Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
# bs_99p is a list of 'r' bootstrap sample's 99th percentiles
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 75
bs_99p = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_99p[i] = np.percentile(bs_sample,99)
```




```
[21] len(bs_99p)
```

```
10000
```

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
[ ] #bs_99p may or maynot be normally distributed.
    print(np.percentile(bs_99p,2.5))
    print(np.percentile(bs_99p,97.5))
```

```
# What if we want a C.I on the 99th precentile?
#Let's create r=10000 bootstrap samples, and let each bootstrap sample be of size=1000
# bs_99p is a list of 'r' bootstrap sample's 99th percentiles
r = 10000
data = df[(df["sourceid"] == 1) & (df["dstid"] == 5)] ["travel_time"]
size = 75
bs_99p = np.empty(r)

for i in range(r):
    bs_sample = np.random.choice(data, size=size)
    bs_99p[i] = np.percentile(bs_sample,99)
```

→ S1: - - - - - -

```
[21] len(bs_99p)
```

```
10000
```

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
[ ] #bs_99p may or maynot be normally distributed.
    print(np.percentile(bs_99p,2.5))
    print(np.percentile(bs_99p,97.5))
```

[23]
```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

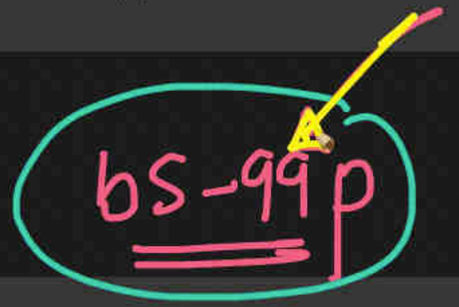
```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

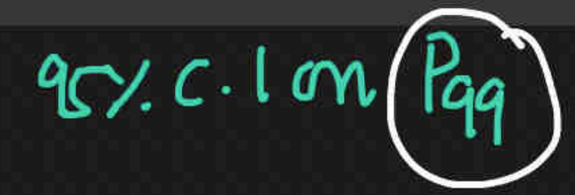
```
1162.56
1174.0
```

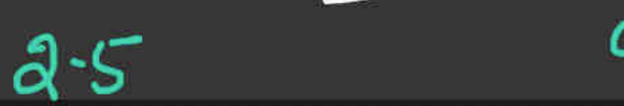
```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

[22]
```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```







[23] bs_99p

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```

[23]

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

95% C.I [l, u] on P_{99}

```
1162.56
1174.0
```

75

```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```

[23]

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

point estimate : Mean 100.2

95% C.I: [98, 102.1]

```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```

[23]

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

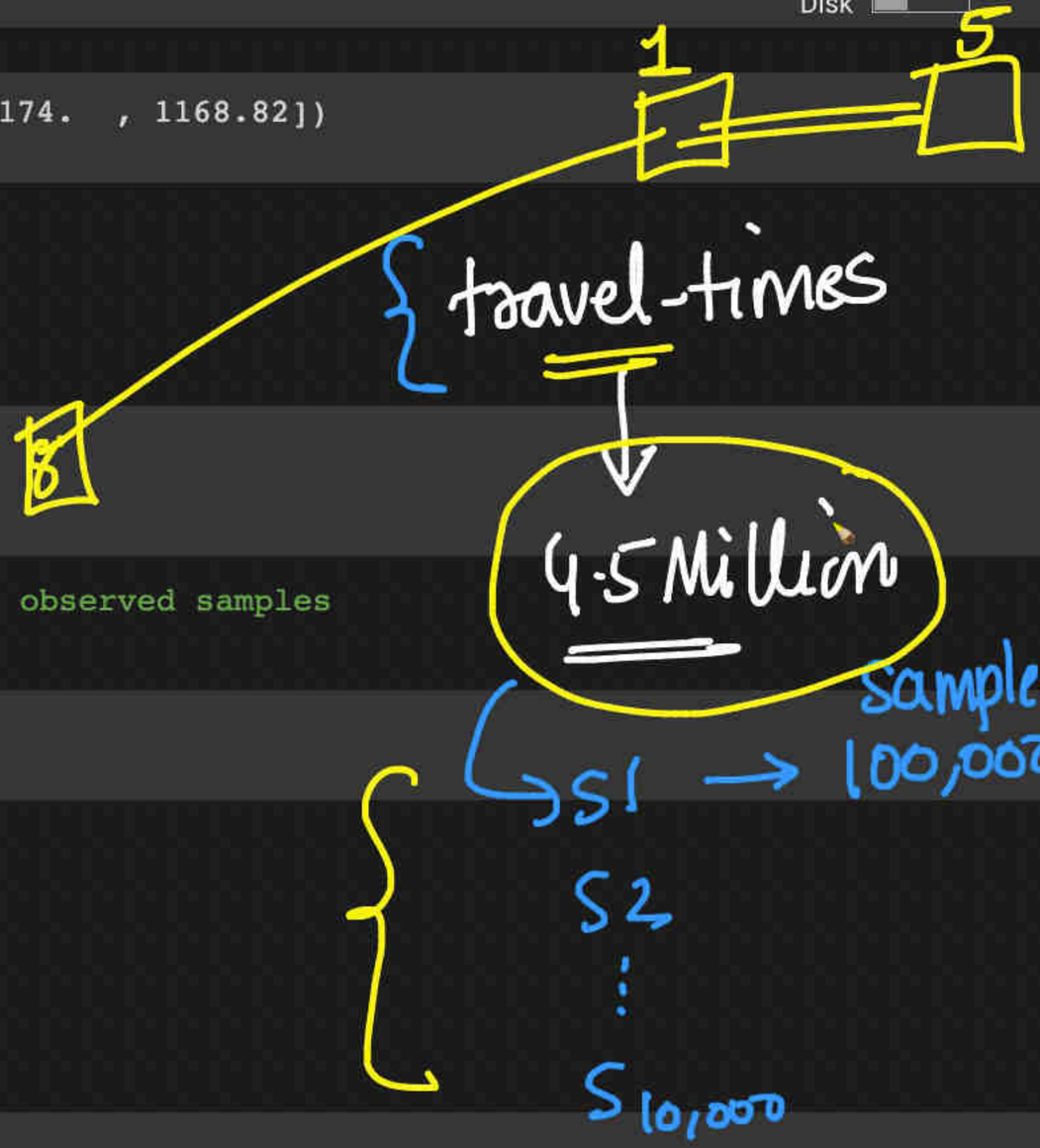
```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

```
# Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```



[23]

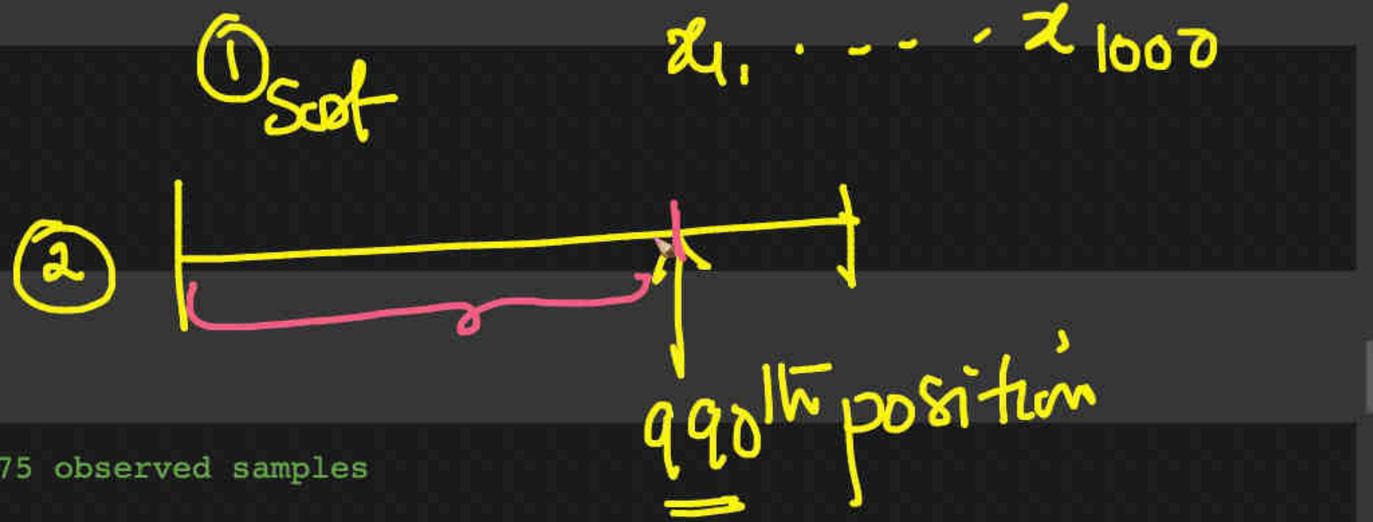
```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
[ ] #bs_99p may or maynot be normally distributed.
    print(np.percentile(bs_99p,2.5))
    print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
    print(np.percentile(data,99))
```

```
1174.0
```




```
# plot the pdf of bs_99p
import matplotlib.pyplot as plt
plt.figure()
plt.hist(bs_99p, bins=100)
plt.grid()
plt.show()
```

```
[21] len(bs_99p)
```

```
10000
```

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

95% C.I of P_{99}

```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

point estimate of P_{99}

```
[22] # plot the pdf of bs_99p
import matplotlib.pyplot as plt
```

prices of items sold

99 lk 100 lk → Max

zolex...

```
[21] len(bs_99p)
```

```
10000
```

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

```
1162.56
1174.0
```

```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

```
[22] # plot the pdf of bs_99p
import matplotlib.pyplot as plt
```

+ Code + Text

```
[21] len(bs_99p)
```

```
10000
```

99.9th   0.1%

```
[23] bs_99p
```

```
array([1174.  , 1174.  , 1174.  , ..., 1167.  , 1174.  , 1168.82])
```

```
#bs_99p may or maynot be normally distributed.
print(np.percentile(bs_99p,2.5))
print(np.percentile(bs_99p,97.5))
```

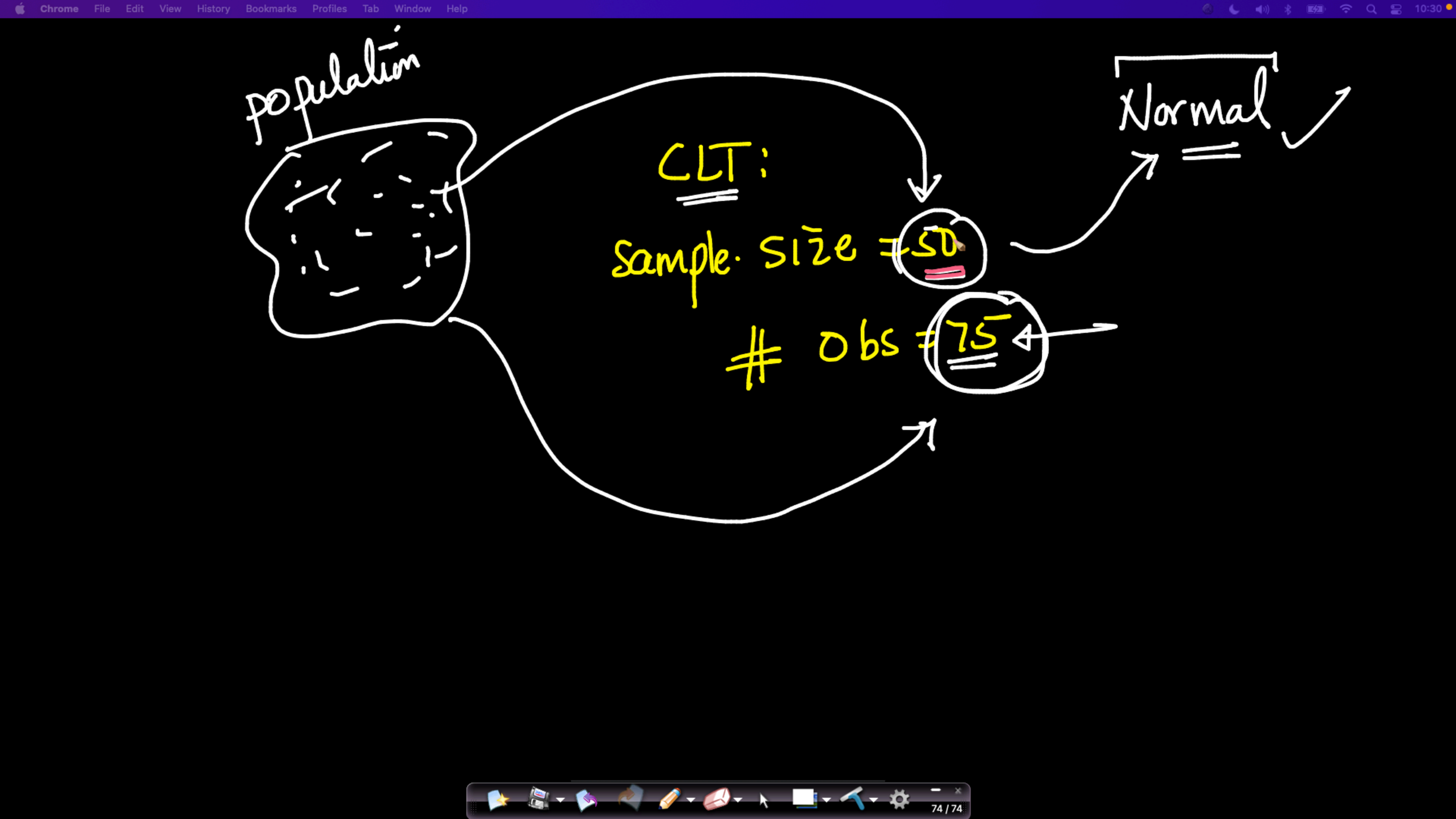
```
1162.56
1174.0
```

```
[ ] # Point estimate of the 99th percenitle of the 75 observed samples
print(np.percentile(data,99))
```

```
1174.0
```

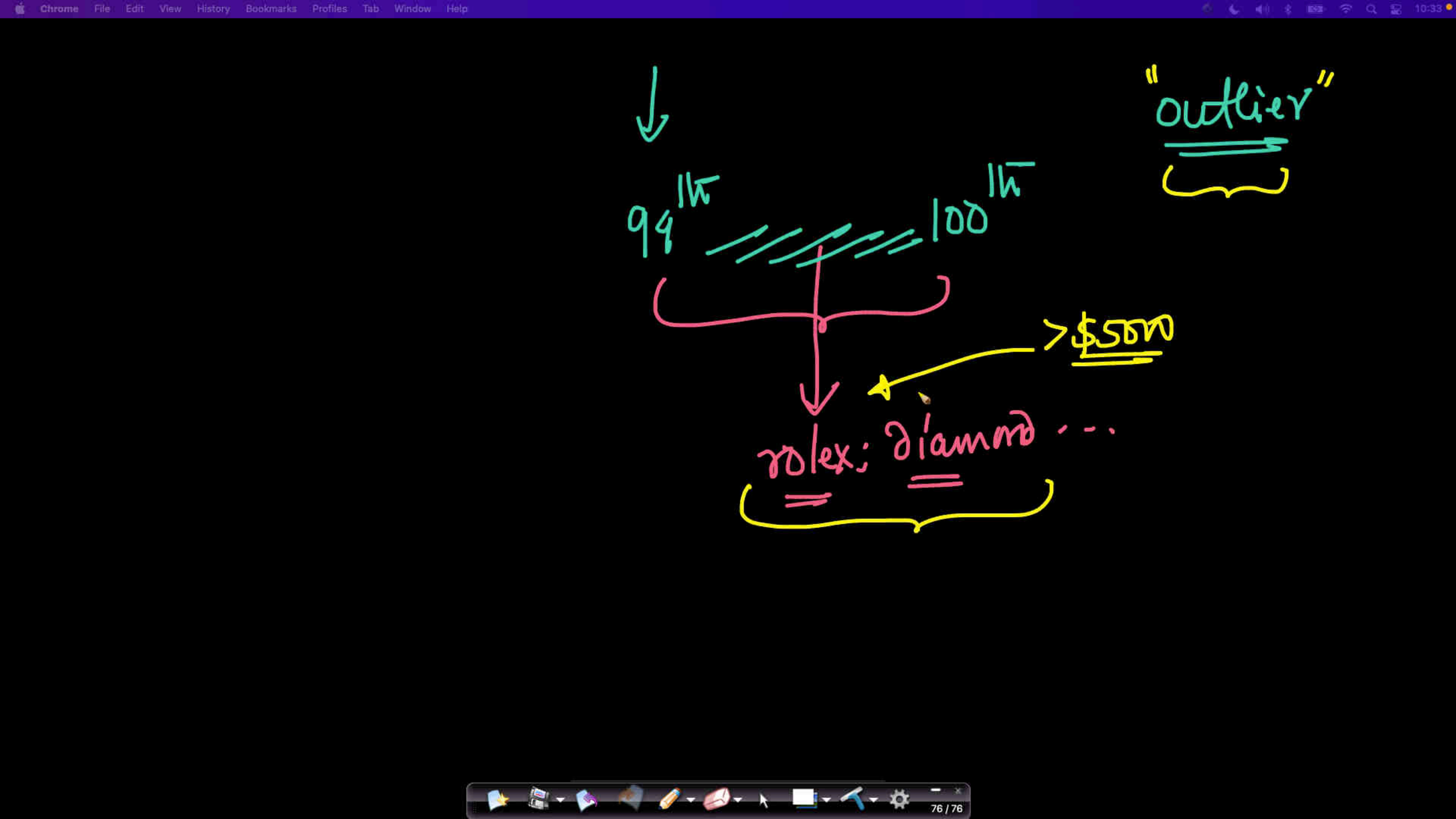
```
[22] # plot the pdf of bs_99p
import matplotlib.pyplot as plt
```

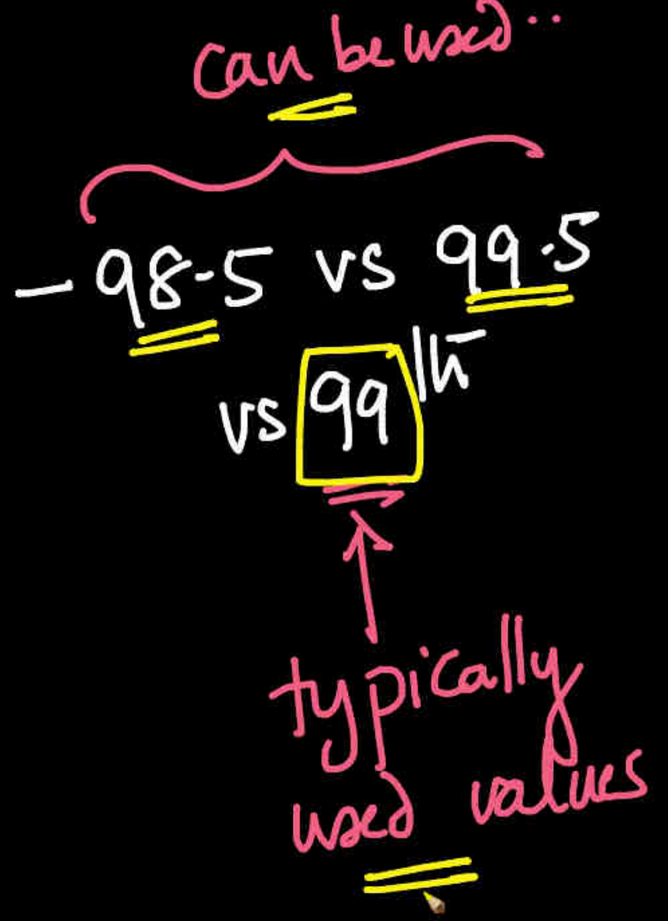
73 / 73


population
CLT:
Sample size = 50
obs = 75
Normal

$r = 10{,}000$ ← computational reasonable

n = sample size $\leq$ #obs


"outlier"
94 lb
100 lb
>$500
rolex; diamond ...

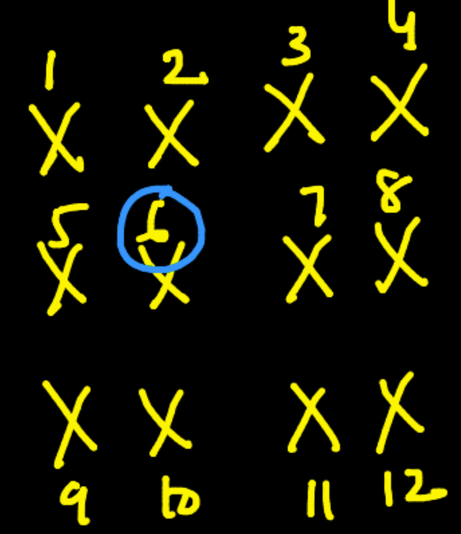


Samplings:

Random Sampling ✓

Uniform dist

[1, 12]



with repl

6 8 . . .

random - sampling

[1 ... 12]

1 2 3 4
5 6
9 10 11

without repl →

x^7, x^8, x^{13}

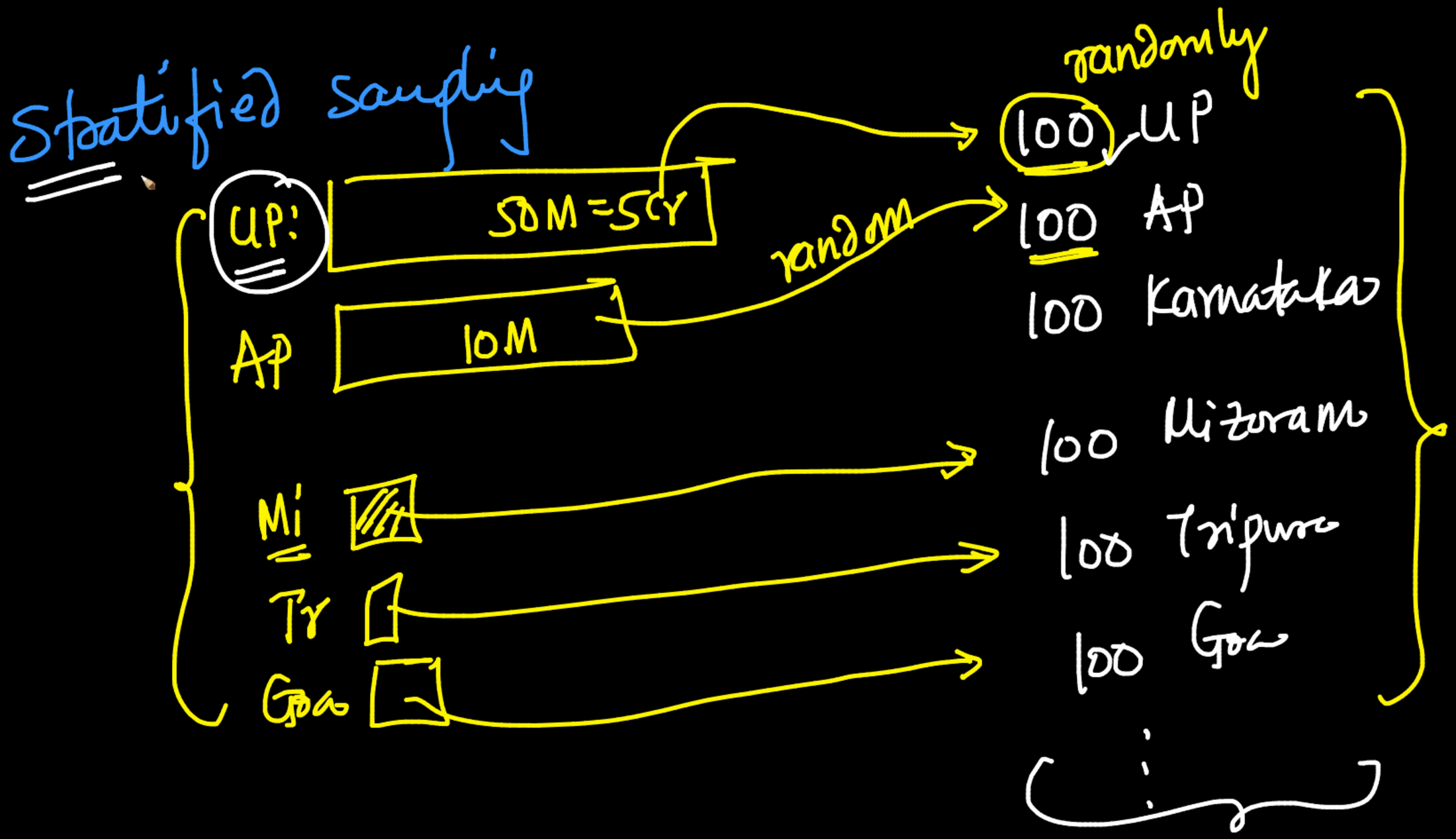
8-pls →

✓ {family incomes in India across states} ✓

Stratified sampling

✓ random-sample → more obs from more populous states

DO NOT WANT


Stratified sampling
UP:
50M = 5Cr
AP
10M
random
randomly
100 UP
100 AP
100 Karnataka
100 Mizoram
100 Tripura
100 Goa
Mi
Tr
Goa

Systematic Sampling:

4 → 330M

Salaries of each family in Income

1. Sort all the families by Salary

330M → inc

F_1^1 F_2^2 ---- F_{330M} ... F_{1M}

1M families

2. F_2^1 F_{330}^3 $F^{(1)}$ ....

picking @ regular intervals

Hypothesis testing

- most useful & fundamental
- Tricky to grasp
  - counter intuitive
- takes time to grasp → revise after Class
- Terminology ...

- C.I → ✓
- p-value
- → dist ✓

Toy-problem:

Coin is fair (or) biased towards heads

↓

$P(H) = 0.5$

↓

$P(H) > 0.5$

→ toss n times

Probabilistic

6/10 heads → biased towards heads or fair → No

INTUITION

65/100 heads → biased towards heads

52/100 → fair (very confident)

651/1000 → biased (V.V.confident)

Task: coin is fair or ~~Unbiased~~ towards heads

Experiment: Toss 100 times

Test-Statistic (T): Sensible way to measure the outcome of the expt which can help us answer & solve my task

→ #heads in 100 tosses (expt)

let 65 ← T_{obs} observed value of T after conducting the expt

default statement / assumption

✓ Null-hypothesis (Ho): coin is fair ✓

✓ Alt. hyp (Ha): coin is biased towards heads

(H1)

$$\left\{ P\left(\text{observing } \underbrace{\text{Test-statistic}}_{\text{as extreme or more than } T_{obs}=65} \;\middle|\; H_0 \right) \right.$$

$H_0 \rightarrow$ coin is fair

$= ?$

$$P(T \geq T_{obs}=65 \mid H_0) = ? \quad \cdots$$

↳ how to compute it

H_0 is true

$T \sim Binomial(n=100, p=0.5)$

CDF

$P(H) = 0.5$

T = # heads in 100 tosses

$P($ 65 or more heads in 100 tosses $| H_0)$

$= P(T \geq 65 | H_0) = 1 - CDF(T=64)$

$P(T \leq 64)$

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

```
# Expt: Toss the coin 100 times
# Test statistic: Count the number of heads
# H0: Coin is fair
# Ha: Coin is biased towards heads
# T ~ Binomial(n=100, p=0.5) under Null Hypothesis(=H0)

prob = stats.binom.cdf(k=64, n=100, p=0.5)
print(1-prob) # P(T >= 65 | H0 )
```

```
0.0017588208614850442
```

```
# Plot PDF of Binomial(100, 0.5)
X = stats.binom.rvs(n=100, p=0.5, size=100000)
plt.grid()
sns.distplot(X)
```

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

```
# Expt: Toss the coin 100 times
# Test statistic: Count the number of heads
# H0: Coin is fair
# Ha: Coin is biased towards heads
# T ~ Binomial(n=100, p=0.5) under Null Hypothesis(=H0)

prob = stats.binom.cdf(k=64, n=100, p=0.5)
print(1-prob) # P(T >= 65 | H0 )
```

1 − CDF(64) T=

0.00175....

```
0.0017588208614850442
```

0.175%

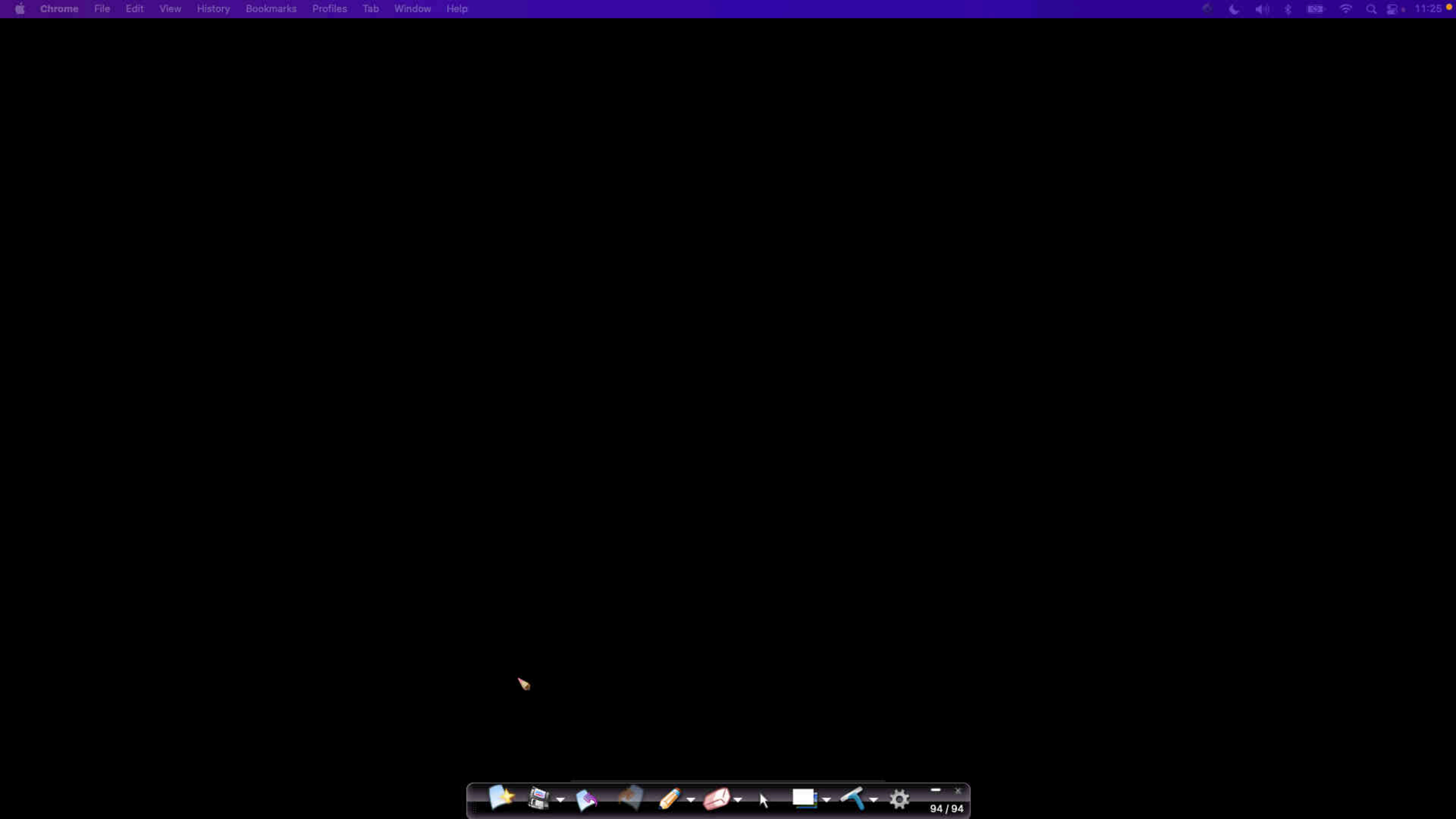
```
# Plot PDF of Binomial(100, 0.5)
X = stats.binom.rvs(n=100, p=0.5, size=100000)
plt.grid()
sns.distplot(X)
```

$$P(T \geq 65) = 1 - P(T < 65)$$

(T: #heads)

$$= 1 - P(T \leq 64)$$

$P(T \leq 64)$ = CDF $(T = 64)$


Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:25
94 / 94

$$P(T \geq T_{obs} = 65 \mid H_0) = 0.17\%$$

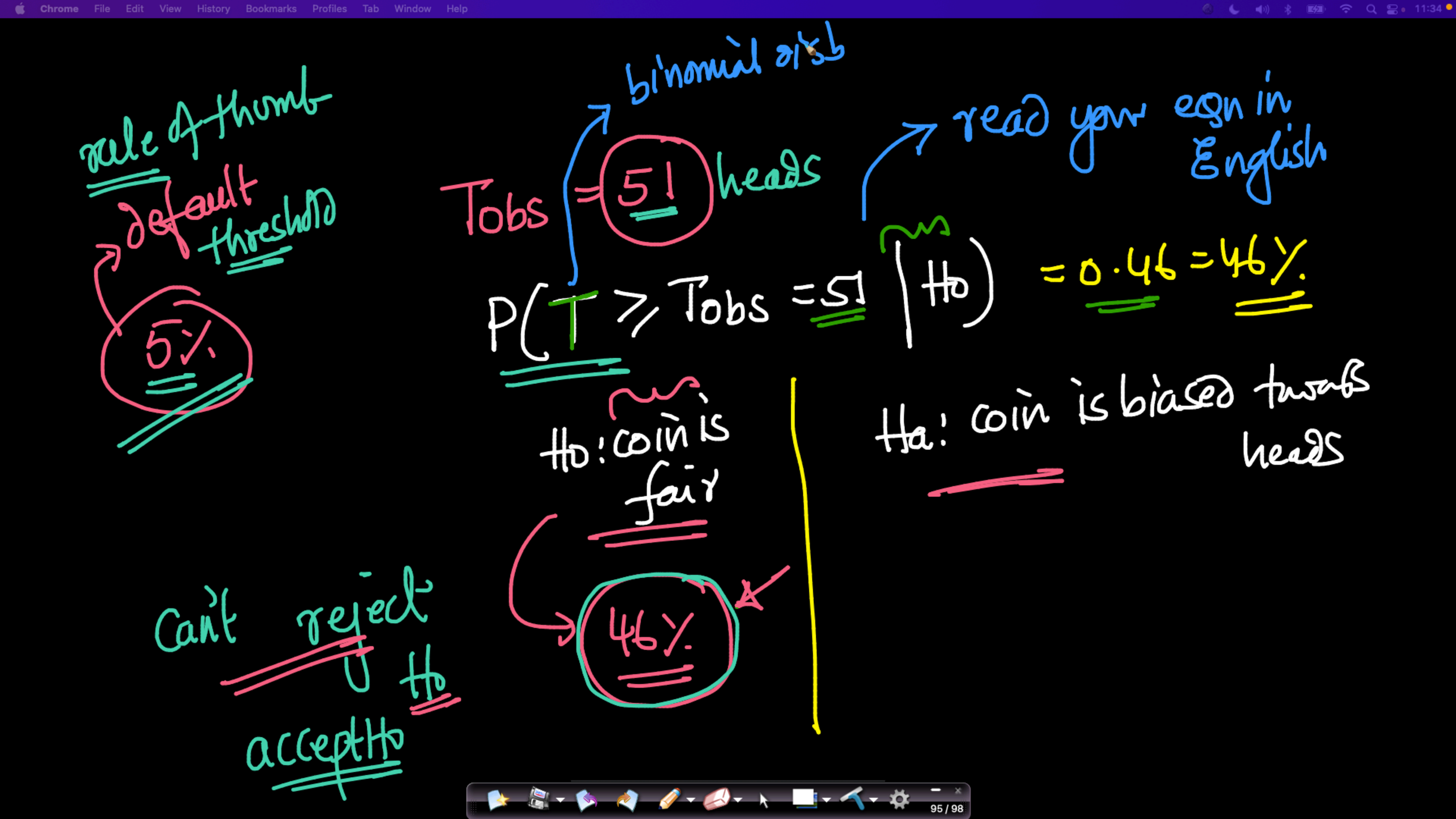
H_0: coin is fair

H_a: coin is biased towards heads

0.17% : V. low

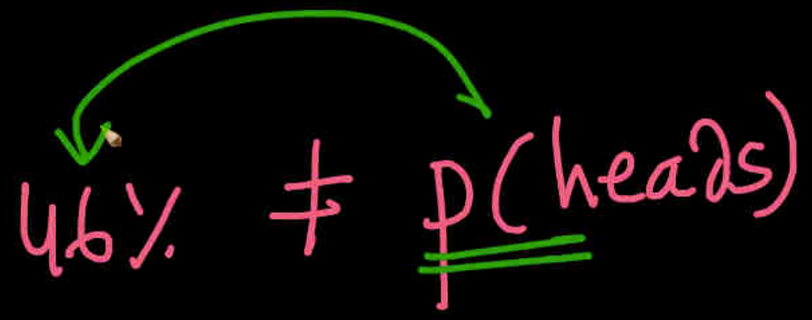
reject H_0

accept H_a


rule of thumb
default threshold
5%
Tobs = 51 heads
binomial dist
read your eqn in English
P(T ≥ Tobs = 51 | Ho) = 0.46 = 46%
Ho: coin is fair
Ha: coin is biased towards heads
46%
Can't reject Ho
accept Ho

Chrome File Edit View History Bookmarks Profiles Tab Window Help

rewatch — didnot click



$$\hookrightarrow P(T \geq 51 \text{ (Tobs)} \mid H_0) = 46\%.$$

[1]
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
```

```
# Expt: Toss the coin 100 times
# Test statistic: Count the number of heads
# H0: Coin is fair
# Ha: Coin is biased towards heads
# T ~ Binomial(n=100, p=0.5) under Null Hypothesis(=H0)

prob = stats.binom.cdf(k=50, n=100, p=0.5)
print(1-prob) # P(T >= 65 | H0 )
```

$P(T \geq 51 \mid H_0)$

```
0.4602053813064103
```

```
# Plot PDF of Binomial(100, 0.5)
X = stats.binom.rvs(n=100, p=0.5, size=100000)
plt.grid()
sns.distplot(X)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated func
```

assessment → off sync

5-6 classes

→ back to sync → next module

rewatch → Hyp. testing

convention

Small <5%.

has some dist

$T \sim Bin(n, p)$ → $P(T \geq T_{obs} \mid H_0)$

Under H_0

↳ future classes ...

(permutation testing)

↓

videos as post read

5-6 hyp. testing

A/B tests

T under Ho

→ t-dist

→ χ^2-dist

F-dist

Various tests

→ ANOVA

Netflix

Z-dist N(0,1)

Cont'd expts

rec. times

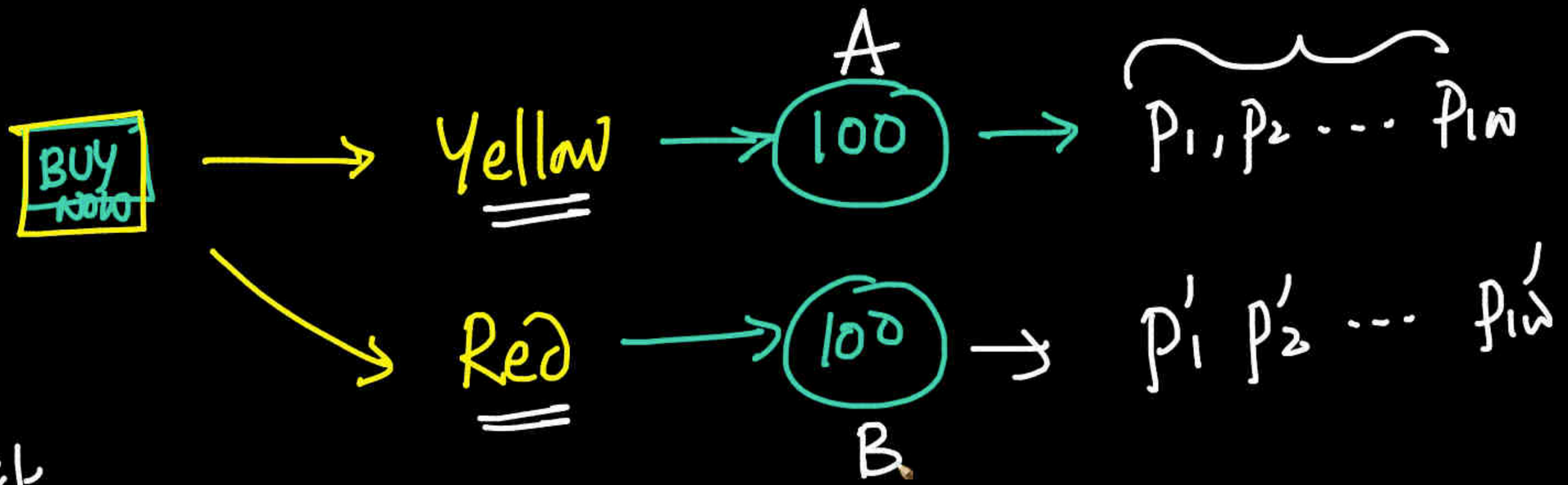
m_1: $x_1^1, x_2^1 \ldots x_{50}^1$ ✓

m_2: $x_1^2, x_2^2 \ldots x_{60}^2$

Z-Test

H_0: mean rec. time of M_1 = mean rec. time of M_2

H_a: " $\neq$ "



Z-test

conversations about popular titles. Second, we can help members choose some great content to watch by fulfilling the intrinsic human desire to be part of a shared conversation.

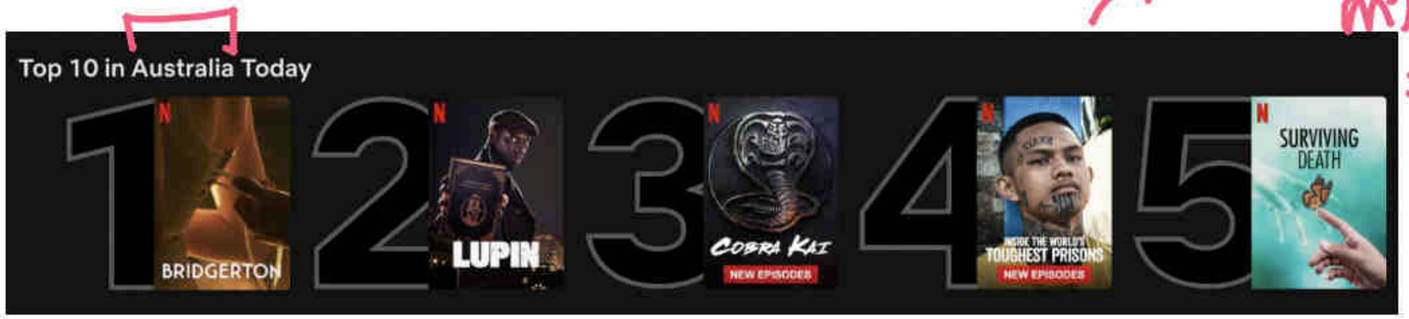


Figure 5: An example of the Top 10 experience on the Web UI.

We next turn this idea into a testable hypothesis, a statement of the form "If we make change X, it will improve the member experience in a way that makes metric Y improve." With the Top 10 example, the hypothesis read: "*Showing members the Top 10 experience will help them find something to watch, increasing member joy and satisfaction.*" The primary decision metric for this test (and many others) is a measure of member engagement with Netflix: are the ideas we are testing helping our members to choose Netflix as their entertainment destination on any given night? Our research shows that this metric (details omitted) is correlated, in the long term, wi

Task:

A: loo → no top lo ⟶ $t_1, t_2 \ldots t_w$ how much time Spent

B: loo → top lo  $t_1', t_2' \ldots t_w'$

H_0: Mean watchtime A = mean WT_B

H_a: " ≠ "

Bin(n=100; p=0.5) → P(H)=0.5

p-value ← P(T≥Tobs | H0) ← H0: coin is fair

P(T≥Tobs | H0)

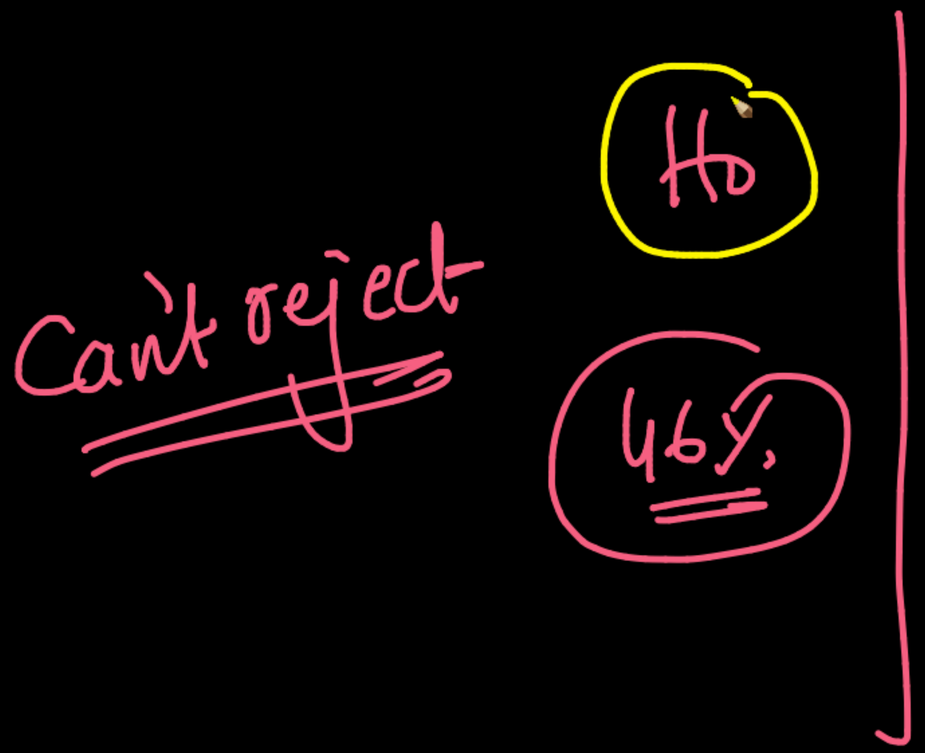
Bin(n=100; p=?)

→ is impossible

H0: coin is biased towards heads

{P(H)=0.51, 0.52, 0.53 ...} ∞-many

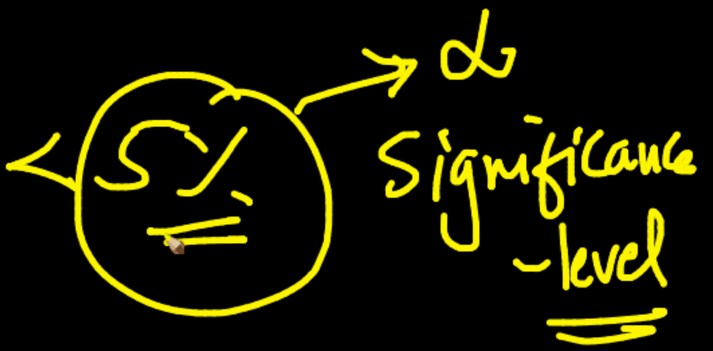
$P(T \geq \approx 51 \mid H_0) = 0.46 = 46\%$ $> 5\%$



reject H_0 ⟶ accept H_a

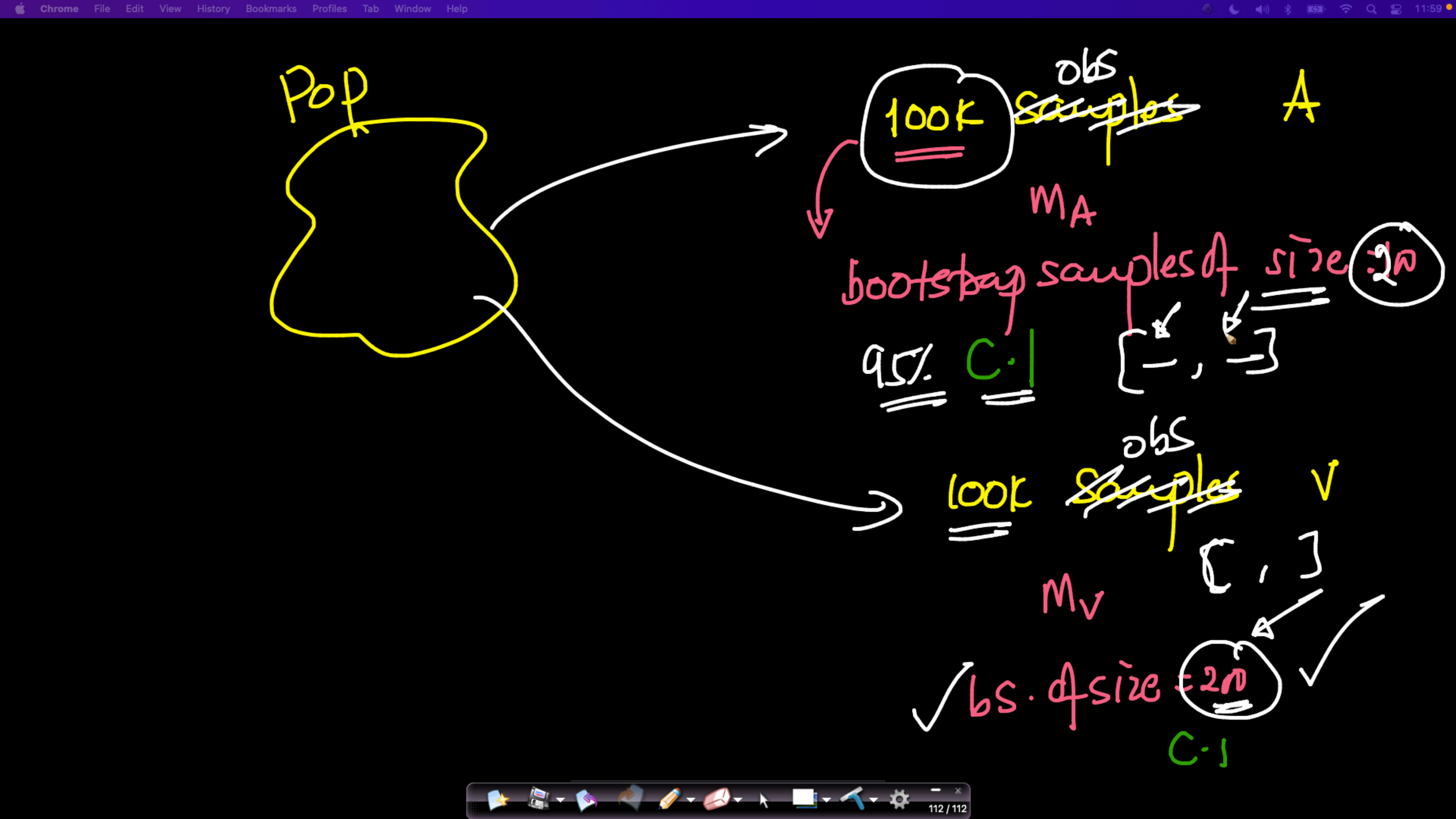
Can't reject H_0 ⇒ accept H_0 & reject H_a

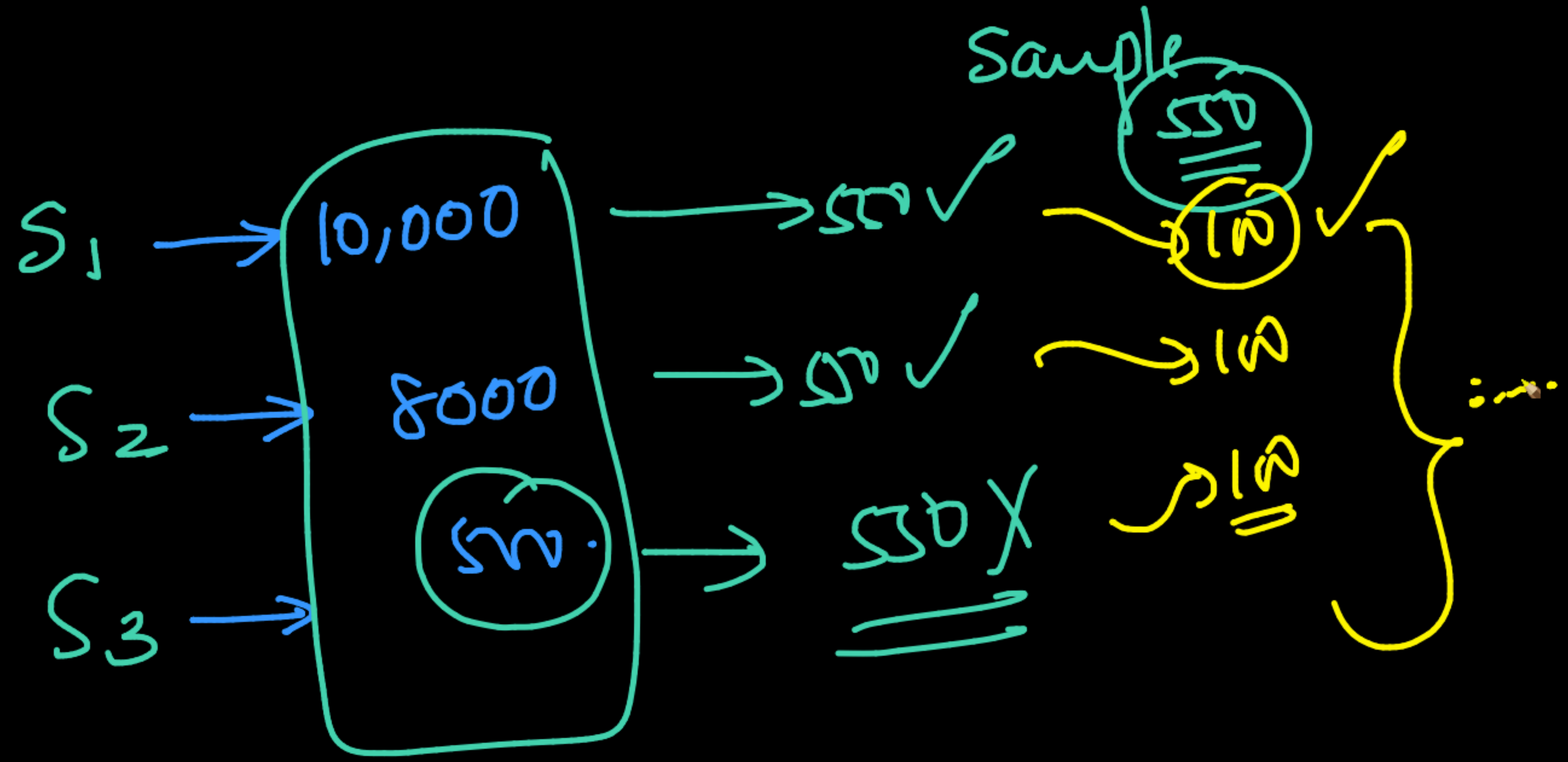
$P(T \geq T_{obs} \mid H_0) = \overline{p\text{-val}}$



reject H_0 $\Rightarrow$ accept H_0

o/w:- accept H_0


Pop
100k
obs
samples
A
m_A
bootstrap samples of size :2n
95% C.I
[,]
100k
obs
samples
V
[,]
m_V
✓bs.dsize =2n
C.I
✓



conversations about popular titles. Second, we can help members choose some great content to watch by fulfilling the intrinsic human desire to be part of a shared conversation.

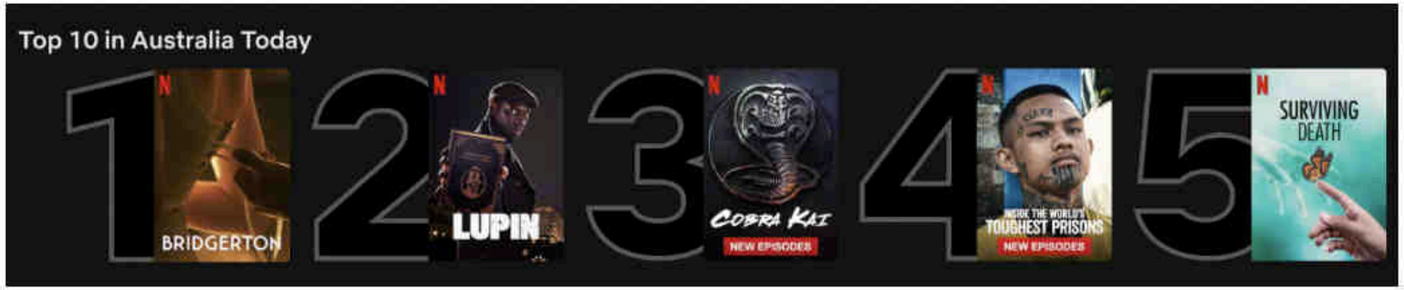


Figure 5: An example of the Top 10 experience on the Web UI.

We next turn this idea into a testable hypothesis, a statement of the form "If we make change X, it will improve the member experience in a way that makes metric Y improve." With the Top 10 example, the hypothesis read: "*Showing members the Top 10 experience will help them find something to watch, increasing member joy and satisfaction.*" The primary decision metric for this test (and many others) is a measure of member engagement with Netflix: are the ideas we are testing helping our members to choose Netflix as their entertainment destination on any given night? Our research shows that this metric (details omitted) is correlated, in the long term, with whether members will retain their subscriptions. Other areas of the business in which we run tests, such as the

1.6K 15

114 / 115

Netflix Technology Blog
265K Followers

Learn more about how Netflix designs, builds, and operates our systems and engineering organizations

Follow

More from Medium

Netflix Technology ... in Netflix Tech...
Experimentation is a major focus of Data Science across Netflix

Diana Jerman in disney-streaming
Disney Streaming Embraces 3 Key Tenets of Experimentation

Clare La... in Expedia Group Technol...
2021's Top 10 Posts on Expedia Group Technology

Zhiheng Xu in The Airbnb Tech Blog
Intelligent Automation Platform: Empowering Conversational AI and Beyond at Airbnb