Unit - 4

Statistical Inference

* Frequentist vs. Bayesian approach

Frequentist approach

→ there exists a population, represented by several parameters, from which numerous samples can be obtained

→ Parameters are fixed but are not accessible.

→ Thus parameters of a sample of the population are computed, and statistical inference techniques are used to make propositions about the population parameters.

Bayesian approach

→ produce parameter distributions representing all the knowledge one can extract from a sample & from prior info about the problem.

* Propositions in the Frequentist approach

(i) Point estimates - A point estimate is a particular value that best approximates some parameter. For eg. the mean / variance of the sample.

(ii) Confidence intervals - a range of values that best represents a parameter of interest.

(iii) Accepting or rejecting a hypothesis

* Variability in Estimates

- Assume that one has access only to a sample of the entire population.
- The mean of the population has to be deduced based off of the available sample.

• The sample mean is a point estimate of the population mean.

• However, the sample mean may vary from one sample to another.

• Thus, a sampling distribution of the sampling mean is considered, which can be used to compute a measure of the variability.

• Sampling Distribution: the distribution of the point estimate based on samples of size n drawn from the population.

1. Explain in brief about plotting a sampling distribution of point ostimates.

Procedure

(i) Draw a large no. of independent samples from the population, with each sample having n elements.

(ii) Evaluate the sample mean of each sample.

(iii) Estimate the sampling distribution by the empirical distribution of the sample replications.

Code

① #creating 10 diff samples, wo/ 25 %. of the population

```
import numpy as np
import pandas as pd

df = accidents.to_frame()

m = []

for i in range(10):          → to generate 10 samples
    df['testing'] = False    → assume that initially no value has been chosen
    sampled_ids = np.random.choice(df.index,    → select an index
                                   size=int64(np.ceil(df.index.size*025)),    → to select 93 values
                                   replace=False)    → no replacement
    df.loc[sampled_id, 'testing'] = True    → set chosen indices to True
    samples = df[df['testing']==True]    → accumulate samples labelled as True
    m.append(samples.mean())
    ~~print(accidents-sample[0].mean())~~    → find their mean
```

(2) to generate n samples each w/ a given no. of elements

```
df = accidents.to_frame()
n = 10000
size = 200
mean = [0] * n          # array of all the means
for i in range(n):
    indices = np.random.choice(df.index.values, size)
    samples = df.loc[indices]
    means[i] = samples.mean()

# plotting a histogram of the sample mean
import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.hist(np.array(means), bins=50)
plt.show()
```

* Standard Error

→ Suppose there is only 1 sample of the population

→ The mean computed from this sample may be close, but will not be equal to the actual value.

→ The variability can be measured using the Standard Error.

$$SE = \frac{\sigma}{\sqrt{n}}$$

→ a method to calculate standard deviation, since there is no direct access to the population.

Code:

```
rows = np.random.choice(df.index.values 200)
sample_df = df.loc[rows]
SE = sample_df.std() / math.sqrt(200)
print(se)
```

#1 n = 200

Alternate method of calculating the SE – Bootstrapping

In bootstrapping:

→ N observations are drawn w/ replacement from the original data to make a bootstrap sample.

→ The mean is calculated for this sample.

→ This process is repeated a large no. of times to get a good approximation of the mean sampling distribution.

code:

```
import numpy as np
def Bootstrap(array in):
    mean_list = [0] *n
    for i in range(n):
        sample = [array[i] for i in np.random.randint(len(array),
                                                      size=len(array))]
        mean_list[@j] = np.mean(sample)
    print(np.mean(mean_list))

# drawing a histogram
import matplotlib.pyplot as plt
plt.hist(np.array(m),
         bins=50,
         density=True,
         histtype='stepfilled')
plt.xlabel('  ')
plt.ylabel('  ')
plt.show()
```

```
# bootstrapping of the median
import numpy as np
def medianBootstrap(array, n):
    mean_list = [0] * n
    for i in range(n)
        sample = [mean_list[i] for i in np.random.randint
                  (0, len(array), size = len(array))]
        mean_list[i] = np.median(sample)
    print(np.mean(mean_list))
```

Confidence Intervals

- A point estimate provides a single plausible value for a parameter.
- A point estimate is rarely perfect, there is always some error
- As an alternative, a range of values can be provided for the parameter.
- A possible range of values for the sample parameter is called a confidence interval.

Confidence intervals are based on the following

→ It is built around the point estimate.

→ The plausibility of a range of values is defined from the sampling distribution

Methods of calculation of confidence intervals

(1) using standard errors and the degree of confidence required.

For a 95% confidence interval, the limits would be within 1.96 standard errors of the true mean

$$CI = (\text{mean} - 1.96 \times S.E\ ,\ \text{mean} + 1.96 \times SE)$$

In general, for a confidence level z:

$$C.I = \text{mean} \pm z \times ZE$$

confidence level	90%	95%	99%	99.9%
z	1.65	1.96	2.58	3.291

Code:

```
mean = df.mean()
se = df.std() / math.sqrt(len(df))
ci = [mean - 1.96 * se, mean + 1.96 * se]
print(ci)
```

② Using Bootstrapping

To compute a 95% confidence interval of the sample mean w/ bootstrapping:

1. Do the following process a large no. of times
   (i) Draw n observations w/ replacement
   (ii) Calculate the mean for each sample
2. Calculate the mean of all the samples
3. Calculate the std for the samples
4. Obtain the 2.5th & 97.5th percentile

Code:

```
def meanBootstrap(array, n):
    mean_array = [0] * n
    for i in range(n):
        sample = [arr[j] for i in np.random.randint(len(array),
                                                    size=len(array))]
        mean_array[i] = np.mean(sample)
    return mean_array
```

```
s = mean Bootstrap(array, n)
s-mean = np.mean(s)
s-std = np.std (m)
ci = [np.percentile(s, 2.5), np percentile (s, 97.5)]
```

Note: Calculation of a CI means that in 95% of the cases, the true mean of the population would lie within the interval defined by the bounds $\pm 1.96 \times SE$.

Hypothesis Testing

Hypothesis Testing: A process of determining statistical significance. When there are differences between groups, the evaluation of whether the effect is real or if it may have happened by chance.

It is done by initially setting 2 competing hypotheses:

(i) The Null Hypothesis - the apparent effect occurred by chance

(ii) The Alternative Hypothesis - the effect is real

for eg. consider a dataset w/ info about accidents in Barcelona in 2013, and the other w/ data on the year 2010

Let the null hypothesis be: "The no. of traffic accidents were the same in 2010 & 2013"

Let the alternate hypothesis be: "The no. of traffic accidents were diff. in 2010 & 2013".

Testing Hypotheses using Confidence Intervals

- Initially the mean of the no. of the accidents is computed from the 2 datasets

Code:

```
import pandas as pd

df1 = pd.read_csv('accidents 2010.csv')
data1 = df1['date']
count2010 = data1.value_counts()    #counts the frequency of each value, i.e computes the no. of accidents occurring on a daily basis

print('Mean of 2010 :', count2010.mean())

df2 = pd.read_csv('accidents2013.csv')
data2 = df2['date']
count2013 = data2.value_counts()
print("Mean of 2013 :", count2013.mean())
```

From the computation of the mean from the given data set, it can be seen that a larger no. of accidents occurred in 2013 than 2010. But, the statistical significance of this inference is still unclear.

Since the no. of accidents is higher in 2013, the confidence interval for that year is computed.

If the mean no. of accidents in 2010 falls within that range, then the null hypothesis, that the effect was purely due to chance, can be rejected. Otherwise, we fail to reject the null hypothesis, as there is no strong evidence against it.

Computation of confidence intervals in 2013

Code:

```
n= len(count2013)
mean = count2013.mean()
std = count2013.std()
se = std / np.sqrt(n)
ci = [mean - 1.96 * se , mean + s*1.96 ]
print(ci)
```

In this scenario, the ~~cc~~ mean accident rate in 2010 does not fall within the CI. The null hypothesis cannot completely be ruled out.

This calls for computing the probability that the null hypothesis may actually be true. This is done using p-values

Testing hypotheses using p-values

p-value: The p-value, or probability value is the probability of getting a result that is either the same or more extreme than the actual observation, assuming that the null hypothesis is true.

The p-value is used to support or reject the null hypothesis. The p-value is evidence against a null hypothesis. The smaller the p-value, the stronger the evidence that the null hypothesis can be rejected.

A p-value <0.05, i.e $<5\%$. is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than 5% possibility that it is correct. The alternate hypothesis is accepted.

- p-values are computed as k/N

  k = no. of obs > observed. (~~diff one~~ computed stat $>= 0.05$)

  N = no. of samples

Method of Testing

(i) Quantify the size of the apparent effect, by choosing a test statistic.

In the case of accidents in Barcelona in 2010 & 2013, there is a diff in accident rates, so the test statistic should be a difference in the means.

(ii) Define the null hypothesis: here, the null hypothesis is that there is no. diff between the 2 periods.

(iii) Compute p-value: compute the diff. in means, then find the probability.

(iv) Interpret the result: a lower p-value ⇒ statistically significant, null hypothesis be wrong.

Code to compute the test statistic - i.e the diff. in means

~~mean(count2010)~~

mean_diff = (count2013.mean() - count2010.mean())

print(mean_diff)

Approximating the p-value

(i) pool both distributions, generate samples of size n, find the diff. in the mean

(ii) count how many samples have differences larger than the observed one.