Midterm Review 2023 F

Course Outline

20/10/2023 15:31

By Yushu Zou <- Not A Detailed Oriented Person
Not 100% correct, trust lecture note if there exist conflict :)

- **Prepare Tips**
- **Data Wrangling**
- **Distributions**
- **Confidence Intervals**
- **Hypothesis Tests**
- **Python Code**

Prepare Tips

- Lecture Code & Demo Code
- Homework Problem
- Tutorial Quiz
- Practice Exam

Data Wrangling

- Import library
  `import FULL_NAME as SHORT_NAME`
- Axis : column-wise : axis = 1; row-wise: axis = 0
- `.loc` support boolean index, but `.iloc` does not
- `df.col` or `df['col']` or `df[('col')]` or `df[(('col'))]` give series; `df[['col']]` gives dataframe
- compond function: `&`(and), `|`(or), `==`(equal), `!=`(not equal)
  ex. `df.loc[(cond1)&(cond2), ('col1', 'col2')]`
- Possible Error
  - Forget import package before calling function package
  - Calling wrong package name: if `import FULL_NAME`, then use `FULL_NAME.func`; if `import FULL_NAME as SHORT_NAME`, then use `SHORT_NAME.func`

- Calling wrong function name from the package
- Calling Boolean select column when using `.iloc`

Distribution

Characteristics of a distribution

- Location/Center
  - Mean: `n=len(my_samp); my_samp.sum()/n` or `my_samp.mean()`
  - Median: `np.percentile(my_samp, 50)` or `sorted(my_samp)[int(n/2)]` or `np.quantile(my_samp, 0.5)`
  - Mode: `from collections import Counter Counter(my_samp).most_common()`
- Scale/Spread
  - Range: Maximun - Minimum `my_samp.max() - my_samp.min()`
  - IQR (for boxplot) `np.quantile(my_samp, 0.75) -np.quantile(my_samp, 0.25)`
  - Variance: `my_samp.var(ddof=1)`
  - Standard Deviation: `my_samp.std(ddof=1)`
- Skewness:
  - Left-Skewed (Mean < Median <Mode)
  - Right-Skewed (Mean > Median > Mode)
  - Symmetry (Mode = Median = Mean)
- Modality (Unimodal, Bimodal, Multimodal)

Data Type

Quantitative / Numerical

- Continuous: example: (52.4,23.5)
- Discrete: example(30, 50)

Qualitative / Categorical

- Binary : two level categorical example(Yes, No)
- Ordinal: ordered value, example (Monday, Tuesday,..)
- Nominal: factor example(Countries, Hair Color)

Data Visualization

Barplot

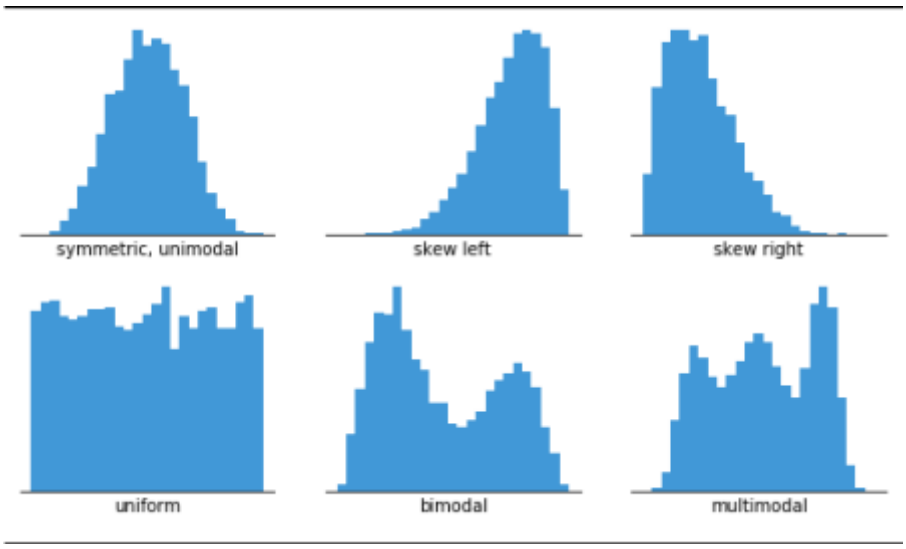
One bar for each **category variable**; order of the bars is arbitrary

- Bar plot can interpret the frequency of the categorical data **[Mode]**

Histogram

Height of each bar counts the number of values of the **numerical variable** in the corresponding bin

- Histogram can interpret **Skewness**, **Modality**



Boxplot

Summarizes the distribution of a numerical variable.

- Boxplot can interpret **Median**; **IQR (**$75^{th} - 25^{th}$**)**; **Outliers**, **Skewness**, etc
- We cannot interpret mean and variance from boxplot

Scatterplot

Each point is determined by the values of 2 **numerical variables**: one on x axis, the other on y axis

Sample Statistics

sample size = n

- Sample Mean : $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- Sample Variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$
- Sample Standard Deviation: $s = \sqrt{s^2}$

Confidence Interval

There's a xxx% chance that this xxx% confidence interval construction procedure **captured the true parameter value**

The purpose of confidence interval is to obtain an estimate the parameter that reflects sampling variability.

A larger confidence level (e.g., instead of 95%, use 98%) would ensure that we capture the population parameter in more samples. This would give a wider confidence interval, extending to more of the bootstrap sampling distribution.

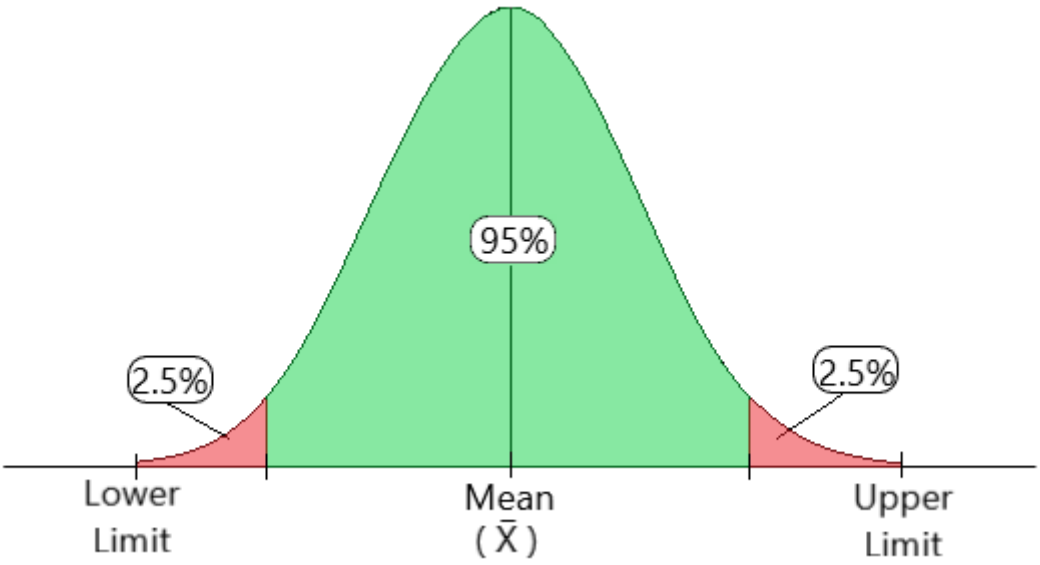
The probability of committing a type I error (rejecting the null when it is actually true) is called α; i.e.,the level of statistical significance. For 99% CI, α=0.01 (confidence level). This means that there is a 1% probability of making a type I error.

Bootstrapping

1. Obtain data sample $x_1, x_2, \ldots, x_n$ drawn from a distribution F.
2. Define u – statistic computed from the sample (mean, median, etc).
3. Sample $x_1^*, x_2^*, \ldots, x_n^*$ with replacement from the original data sample. Let it be F* – the empirical distribution. Repeat N times (N is bootstrap iterations).
4. Compute u* – the statistic calculated from each resample.

if you are still confuse about the concept https://www.youtube.com/watch?v=Xz0x-8-cgaQ

Bootstrapping --> Confidence Interval




```
# Create list variable to store the sample statistics
bootstrapped_means = []
# Start boostrapping, iteration
for i in range(reps):
# Drawing sample from the Sample with replacement <- independent sample
    bootstrap_sample = np.random.choice(x, n, replace = True)
# Calculate the sample statistics from the sample and append to the list
    bootstrapped_means += [bootstrap_sample.mean()]
```

```
# Constructing 95% Confidence Interval based on the sample statistics
np.quantile(bootstrapped_means,(0.025, .975))
```

Hypothesis Testing

A **statistic** is any function of a sample of data
p-value : The probability of a test statistic being as or more extreme than the observed test statistic if H_0 was true

Process

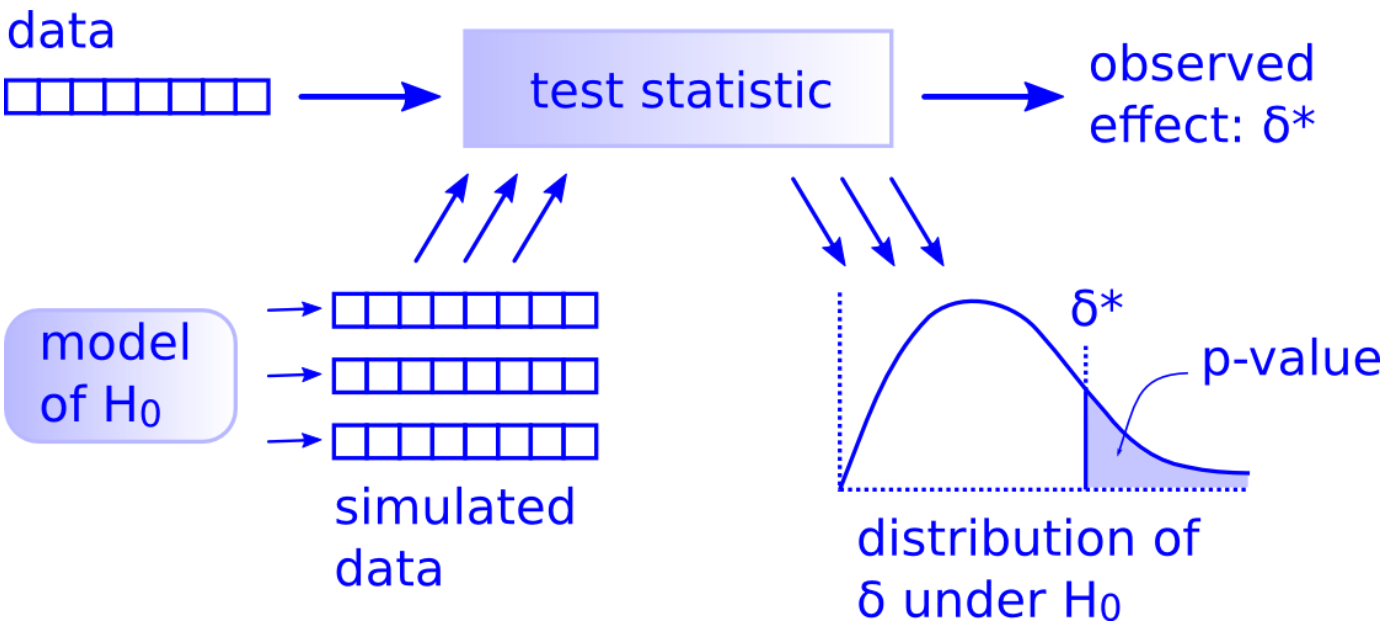
part of resource : [one sample, two sample and paired test]
Step 1. State null hypothesis H_0 and alternative hypothesis H_1: H_0 is FALSE
Step 2. Choose the α-significance level at which H_0 will be rejected (for p-values smaller than α)
Step 3. Calculate the test statistic, from the original data you received:
- ex. *two sample test*: the different between two group's mean/median; *one sample test*: the mean/median of your sample
Step 4. Choose your test type:

	One Sample	**Two Sample *[is different from -->]***	**Paired Sample *[but why?]***
Simulation	[Nonparametric] $H_0 : p = p_0$ Coin Flipping; or, [Parametric] Sampling from H_0 specified distribution	[Nonparametric] Permutation Shuffling H_0 : same population	Take differences then treat like **One Sample** with $H_0 : \mu = 0$
Nonparametric*	`stats.binom` ($H_0 : p = p_0$) `stats.wilcoxon(sample - H0_median)`	`stats.median_test` (equal medians) `stats.mannwhitneyu` (same population)	`stats.wilcoxon` (assumes paired with no difference)
Parametric*	`stats.ttest_1samp` ($H_0 : \mu = \mu_0$ with normality assumption)	`stats.ttest_ind` ($H_0 : \mu_1 = \mu_2$ with normality assumption)	`stats.ttest_rel` ($H_0 : \mu_1 = \mu_2$ with normality and paired assumption)
What about these? -->	`stats.kstest` (this can be "skipped")	`stats.ks_2samp` (this can be "skipped")	<-- These just a challenge to consider...
*Theoretically derived			

Step 5: Conclusion: “Reject H_0 at α-significance level” if the p-value is less than α; Otherwise, “fail to reject H_0 at α-significance level”.

One Sample: We collect a sample, and test whether the sample statistic matches the test statistic

Simulation:



key: construct a simulation based on the H_0

Example Code

```
### Construct the simulation sample under null hypothesis
my_theoretical_population = stats.norm(loc=mu, scale=sd_sigma)
### Create List to store the simulation statistic
simulated_means = []
### collecting the simulated statistic under null hypothesis
for i in range(reps):
    x = normal_population.rvs(size=n)
    simulated_means += [x.mean()]
### p_value: proportion of events that being or more extreme than observed
statistic
p_value = abs(simulated_statistic - null_hypothesis) >= abs(test_statistic -
null_hypothesis)).sum()/num_simulation
```

Parameteric [one sample t-test]

key: the sample must follow normal distribution

Example Code

```
### Key: here you enter the sample data, not the statistic
t_statistic, p_value = stats.ttest_1samp(sample, null_hypothesis)
```

Non-parametric [Binomial Exact Test / Wilcox Sign Test]

```
### Binomial Exact Test
# step1. Calculate the event number (denote as n) in the sample data is
being or more extreme than the oberseved case
# step2. Calculate the probability that is being or more extreme than the
observed case
### two side p value, since we multiply that by 2
```

```
(1-stats.binom(n=n,p=null_hypothesis).cdf(n-1))*2
##### note: stats.binom(n=n,p=p).cdf(n-1) -> The probability of all outcomes
less than or equal to a given value n-1

### one sample wilcoxon-test
z_statistic, p_value = wilcoxon(sample - null_hypothesis)
#### the null hypothesis would be the median value
```

Two Sample: We collect two samples(X_1, X_2), and test whether these two sample statistic are matched, such as $mean(X_1) = mean(X_2)$

$H_0 : mean(X_1) = mean(X_2) \Longrightarrow mean(X_1) - mean(X_2) = 0$

Simulation [Permutation Test]

key: permuting/shuffles labels.
Since the H_0 believes X_1, X_2 has same statistic, when we shuffling the labels in the sample, then there should be no much difference in sample statistic compared to original sample data

```
### STEP1. copy the sample data, because we don't want mess the real label
while shuffling; and create a list to store simulate statistic
df_shuffle = df.copy()
simulate_statistic = []
### STEP2. calculate the observed statistics - the different in statistic
between two groups in sample data
test_statistic = np.diff(df.groupby('label').median().values.flatten())[0]

for i in range(num_simulation):
### STEP2. shuffling the label
#### Take the sample from the df.label and assign to df_shuffle.label, frac
= 1 means 100% of sample will be take, so the distribution of label would
not change after shuffling
    df_shuffle.label = df.label.sample(frac=1,replace = FALSE).value

### STEP3. calculate simulated statistic
    simulated_statistic +=
[np.diff(df_shuffle.groupby('label').median().values.flatten())[0]]


### p_value: proportion of events that being or more extreme than observed
statistic
p_value = abs(simulated_statistic) >=
abs(test_statistic)).sum()/num_simulation
```

Parametric [Two sample t-test]

key: data must follow normal distribution

```
stats.ttest_ind(df[df.label=='group1'],df[df.label=='group2']])
```

Non-parametric [Two-sample wilcoxon test == Mann Whitney U test; median_test]

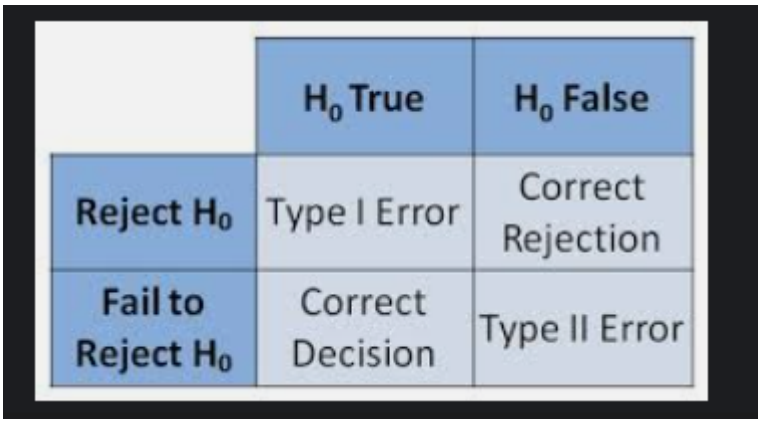
```
### Mann Whitney U test
stats.mannwhitneyu(df[df.label=='group1'],df[df.label=='group2']])

### Median Test
stats.median_test(df[df.label=='group1'],df[df.label=='group2']])
```

Type 1 and Type 2 Error

Type 1: Reject H_0 when H_0 is true

Type 2: Do not reject H_0 when H_0 is false



Python Code (Does not contain all the code Professor show on the lecture!)

```
import Numpy as np
import Pandas as pd
import scipy
from scipy import stats

or
import scipy.stats as stats
```

1. Reading csv file
   `pd.read_csv(.... FILENAME...)`
2. Finding variable type in dataframe
   `df.dtypes` or `type(...VARIABLE_NAME...)`
3. Finding the number of rows and columns in the dataframe
   `df.shape`

4. Summary variable in dataframe
   `df.describe()
5. Checking NULL value in the dataframe
   `df.isnull().sum()
6. Show subset of dataframe
   `df.head()` same as `df.iloc[:5,:]`
   <-- Return first 5 rows
7. Select columns from dataframe
   one column: `df['col']`
   multiple column: `df[['col1', 'col2']]`; `.iloc[,:]`; `df.loc[:, CONDITION]`
   based on condition : `df.loc[boolean_selection_column, ('col1', 'col2')]`
8. Group by and aggregation function
   `df .groupby('one column').aggregation function`
   aggregation function: `.sort_values(ascending=True/False)`; `.size()`; `.mean()`....
9. Drawing histogram/bar plot

```
import plotly.express as px
fig = px.histogram(df, x="col", nbins=n)
fig.show()
```

8. Drawing boxplot

```
fig = px.box(df, y="col") // y is numeric
// if want to create boxplot by categorical variable col2
fig = px.box(df, y="col1", x = "col2")
fig.show()
```