STAT4710J SU2024 Midterm RC Part 1

Author: Li Li

Data Sampling and Probability (Lec 02)

Sample

Probability/Random Sampling

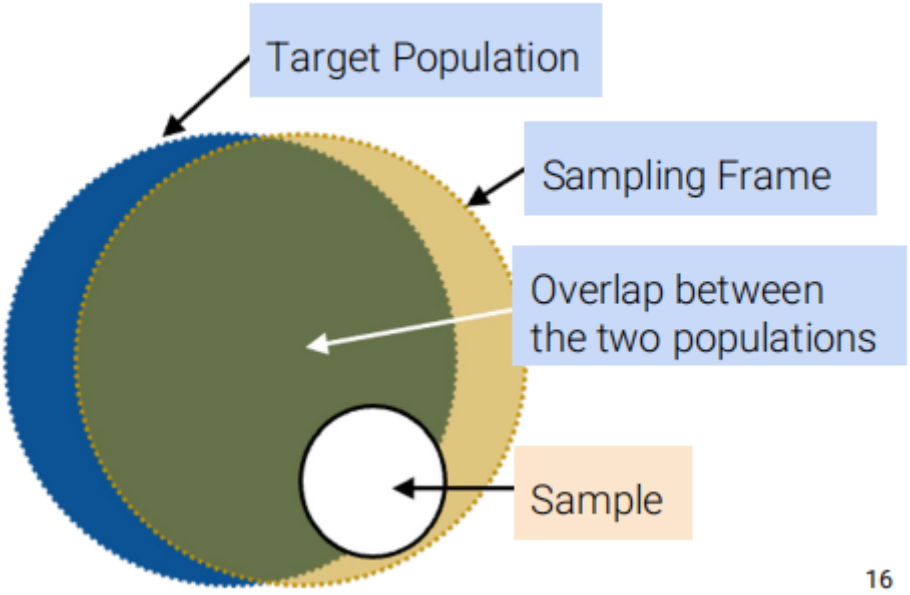
- You **must** be able to provide the **probability** that any specified set of individuals will be in the sample.
- All individuals in the population **do not need to** have the same chance of being selected.
- You will still be able to measure the errors, because you know all the probabilities.

(understand concepts through examples in slides)

probability(random) sample	**Non-random sample**
simple random sample	convenience sample
systematic sample	voluntary response sample
stratified sample	quota sample
cluster sample	snowball sample

- Convenience samples: **Not random**, select the ease samples. Usually used in pilot testing
- Quota samples: **Not random**. First group by one feature, and then do convenience sampling
- Stratified samples: **Random**. First group by one feature, and then do random sampling in each of the subgroups
- Simple random samples: **Random**. Use random number generator to generate a sample. (without replacement)
    - Every individual (and subset of individuals) has the same chance of being selected.
- Systematic samples: **Random**. Choose one candidates every elements
- Cluster samples: **Random**.
- Voluntary response samples and snowball samples: **not random**

Population, sample, and sampling frame



- **Population**: The group that you want to learn something about.
- **Sampling Frame**: The list from which the sample is drawn.
    - If you're sampling people, the sampling frame is the set of all people that could possibly end up in your sample.
- **Sample**: Who you actually end up sampling.
    - A subset of your sampling frame.

Bias vs. Chance Error

- **Bias**: One direction (If your sampling method is biased, those biases will be magnified with a larger sample size)
    - Selection bias: Systematically excluding (or favoring) particular groups.
    - Response bias: People don't always tell the truth
    - Non-response bias: No answers
- **Chance error**: Any direction (get smaller as the sample size get larger, but it is unavoidable)

Sample 1

The City of Feishu wants to hear from its homeowners on issues related to zoning laws.

(For the purposes of this question, homeowners are individuals who own their home, instead of leasing or renting from someone else.)

(a) (1 pt) One method of surveying would be to have city workers come to Feishu Univ's campus and ask passing by students and faculty members for their thoughts. Suppose for now that the question "Are you a homeowner?" is not asked.

What type of sample is this?

- ◯ Convenience sample
- ◯ Probability sample, but not simple random sample
- ◯ Simple random sample
- ◯ Quota sample

(b) (1 pt) Many students and faculty members aren't homeowners, but will be surveyed anyways.

What form of bias or error is this?

- ◯ Response bias
- ◯ Non-response bias
- ◯ Chance error
- ◯ Selection bias

(c) (1 pt) The City of Feishu has a list of all the homeowners' email addresses. Instead of the previous surveying technique, now suppose they take the list of all homeowners' email addresses, shuffle it, and send a survey to every other email address. That is, from the shuffled list, they email the first, third, fifth, seventh, and so on.
(You may assume that the shuffling is done uniformly at random, meaning that each email address has the same probability of landing in any particular position. You may also assume that the City of Feishu has the email address for every single homeowner, and that every single homeowner has a unique email address.)

What type of sample is this?

- ◯ Quota sample
- ◯ Convenience sample
- ◯ Probability sample

(d) (1 pt) Fill in the blank: In this new sampling technique, the sampling frame is _____ the population of interest.

- ◯ equal to
- ◯ greater than
- ◯ smaller than

(e) (1 pt) In this new sampling technique, some homeowners may see the survey and choose not to respond.

True or False: The only form of bias or error in this new surveying technique is non-response bias.

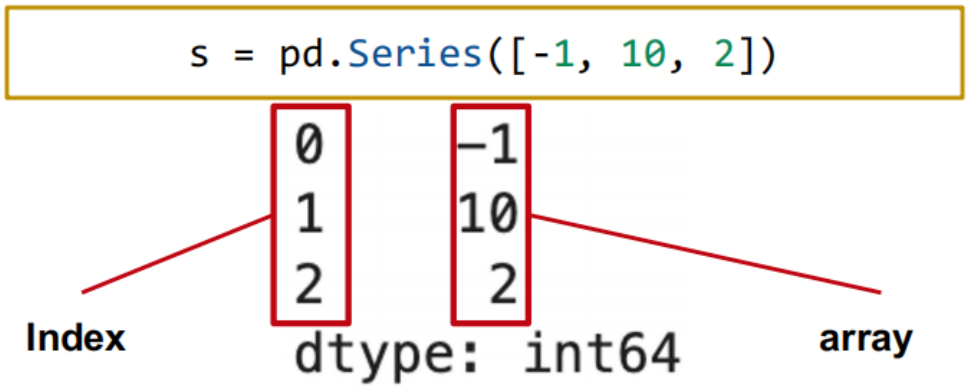
- ◯ True
- ◯ False

Pandas (Lec 03, 04)

Series

- 1D array-like object, value with index

```
s = pd.Series([-1,10,2])
s.array
s.index
```



- custom index (2 ways)

```
s = pd.Series([-1,10,2],index = ["a","b","c"])
s.index = ["first","second","third"]
```

- selection a single value or a set of values in a Series using:

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

  - A single label

```
s["a"]
```

  - A list of labels

```
s[["a","c"]]
```

  - A filtering condition

```
s[s>0]
```

DataFrame

- Creating a DataFrame

```
pandas.DataFrame(data, index, columns)
```

  - Using a list and column names

```
pd.DataFrame([1,2,3],columns=["Numbers"])
pd.DataFrame([[1,"one"],[2,"two"]],columns=["Numbers","Description"])
```

  - From a dictionary

```
pd.DataFrame({"Fruit":["Strawberry","Orange"],"Price":[5.46,3.99]})
pd.DataFrame([{"Fruit":"Strawberry","Price": 5.46},
              {"Fruit":"Orange","Price":3.99}])
```

- From a Series

```
s_a = pd.Series(["a1","b1","c1"],index = ["r1","r2","r3"])
s_b = pd.Series(["a2","b2","c2"],index = ["r1","r2","r3"])
pd.DataFrame({"A-column":s_a,"B-column":s_b})
```

Indices

- Modifying Indices: select a new column and set it as the index of the DataFrame

```
elections.set_index("Candidate", inplace=True)
# Resetting the index
elections.reset_index(inplace=True)
```

Slicing

	loc	**iloc**	**[]**
select by	value	number	one argument (row: number; column: value)
Arguments	a list, a slice (**right inclusive**), a single value, boolean array	a list, a slice (**right exclusive**), a single value	a slice of row numbers (**right exclusive**), a list of column labels, a single column label, boolean array

Conditional Selection

- Create boolean arrays and pass them into slicing.

```
df[df['Party'] == 'Republic']
```

- Use `&` and `|` for boolean operations

```
babynames[(babynames["Sex"] == "F") & (babynames["Year"] < 2000)
```

Handy Utility Functions

- `np.mean()`, `max`
- shape/size

  `df.shape`: (row, col)

  `df.size`: row×col

- `df.describe()`: get statistic information like count, mean, std, ...

  `df['column_name'].describe()`: a different set of statistics will be reported if .describe() is called on a Series
- `df.sample(n, replace = ..., ,,,)`: Sampling without replacement by default. Set replace=True otherwise.
- `df.sort_values(by=..., key=..., ascending=..., ...)`: Sort values in ascending order by default. Set ascending=False otherwise.
- `series.value_counts()`: Counts unique values in descending order
- `series.unique()`: return an **array** of every unique value in a **Series**
- String manipulation
  - `series.str.isin()`
  - `series.str.len()`
  - `series.str.startswith()`

Column Manipulation

- Addition: `df[newcol] = ...`
- Deletion: `df.drop(colname, axis=1)`
- Modification: `df.rename({col1: newcol1, col2: newcol2, ...})`

Groupby and Aggregation Functions

`groupby` creates sub-dataframes and `agg` aggregates all the columns with corresponding data types to one row of output to represent the group.

```
# syntax to aggregate all the columns
# may cause errors
df.groupby(colname).agg(aggfunc)

# select specific columns before aggregation
df.groupby(colname)[[col1, col2, ...]].agg(aggfunc)

# aggregate using lambda functions
df.groupby('Name')[['Count']].agg(lambda x: x.iloc[0])

# aggragate using customized functions
def ratio_to_peak(series):
    return series.iloc[-1]/max(series)
df.groupby('Name')[['Count']].agg(ratio_to_peak)

# can also directly use built-in aggregation functions
# mean(), median(), max(), min(), sum(), count(), size()...
df.groupby(...)[[...]].mean()

# filter subtables that satisfy certain conditions
df.groupby('Name').filter(lambda x: x['num'].sum() > 10)
```

Groupby and Pivot Table

```
# sometimes we want to group by first feature, and then by second feature
df.groupby(['Year', 'Sex']).agg(sum).head()

# use pivot table instead
df_pivot = df.pivot_table(index='Year',
                          columns='Sex',
                          values=['Count', 'Name'],
                          aggfunc=np.max
).head()
```

Join Tables

- Inner join: retains only **matched data** from both tables
- Left join: retains all the rows from the **left table** and sets NaN to the unmatched data in the right table
- Right join: retains all the rows from the **right table** and sets NaN to the unmatched data in the left table
- Outer join: retain all the rows from **both tables** and sets NaN to the unmatched data in the corresponding table

```
# basic syntax
pd.merge(
    left=left_df, # the left table to merge
    right=right_df, # the right table to merge
    how=..., # 'inner' by default
    on=None, # if the column names to merge are the same
    left_on=None, # column name to merge in the left table
    right_on=None, # column name to merge in the right table
    left_index=False, # whether to merge on the index of the left table
    right_index=False # whether to merge on the index of the right table
)
```

Regular Expressions (Lec 05)

operation	order	example	matches	doesn't match
concatenation	3	AABAAB	AABAAB	every other string
or `|`	4	AA|BAAB	AA, BAAB	every other string
closure `*` (zero or more)	2	b AB*A	AA, ABBBA, ...	AB, ...
group `()`	1	(AB)*A	A, ABABA, ...	every other string

Common Meta Characters

`^`: Start of string, or start of line in multi-line pattern

`$` : End of string, or end of line in multi-line pattern

`.` : Any character except new line (\n)

`*` : Match the previous token 0 or more times. *E.g*: ah* matches a and ahhh

`+` : Match the previous token 1 or more times. *E.g*: ah+ matches ah and ahhh , but not a

`?` : Match the previous token zero or one time. *E.g*: bite? matches bite and bit

`.*?` : lazy version of zero or more

`{n}` : Match the previous token exactly n times.

`{n,}` : Match the previous token n or more times.

`{m,n}` : Match the previous token n times, n ∈ [m,n]

`|` : Or.

Groups and Ranges

`(...)` : Group

`(?:...)` : Passive (non-capturing) group

`[abc]` : a or b or c

`[^abc]` : Not a or b or c

`[a-k]` : Lower letters from a to k

`[A-K]` : Upper letter from A to K

`[a-kA-Q]` : Lower letters from a to k , upper letter from A to Q

`[0-8]` : Digits from 0 to 8

Character Classes

`\s` : White space. Equivalent to [\r\n\t\f\v]

`\S` : Not white space. Equivalent to [^\r\n\t\f\v]

`\d` : Digit. Equivalent to [0-9]

`\D` : Not digit. Equivalent to [^0-9]

`\w` : Word character. equivalent to [a-zA-Z0-9_]

`\W` : Not word character. equivalent to [^a-zA-Z0-9_]

`\c` : Control character

String Manipulation Functions

re	pandas Series
`re.sub(pattern, repl, text)`: Returns text with all instances of pattern replaced by repl. (patter is a raw string: r"...")	`ser.str.replace(pattern, repl,regex=True )`: Returns Series with all instances of pattern in Series ser replaced by repl.
`re.findall(pattern, text)`: Return a list of all matches to pattern.	`ser.str.findall(pattern)`: Returns a Series of lists `df["SSN"].str.findall(pattern)` SSN 0 987-65-4321 1 forty 2 123-45-6789 bro or 321-45-6789 3 999-99-9999 0 [987-65-4321] 1 [] 2 [123-45-6789, 321-45-6789] 3 [999-99-9999] Name: SSN, dtype: object
	`ser.str.extract(pattern)`: Returns a DataFrame of first match, one group per column
	`ser.str.extractall(pattern)`: Returns a DataFrame of all matches, one row per match

Raw String

Regular String	Raw String
'ab*'	r'ab*'
`'\\\\section'`	`r'\\section'`
`\\w+\\s+\\1`	`r'\w+\s+\1'`

Capture Group

- Every set of parentheses specifies a **match/capture group**.

```
text = """Observations: 03:04:53 - Horse awakens.
03:05:14 - Horse goes back to sleep."""
pattern = "(\d\d):(\d\d):(\d\d) - (.*)"
matches = re.findall(pattern, text)
```

```
[('03', '04', '53', 'Horse awakens.'),
 ('03', '05', '14', 'Horse goes back to sleep.')]
```


Sample3

For this question, you're given the following code:

```
re.findall(pattern, "godoggogo100")
```

For each possible pattern, list the number of times that the string `"go"` appears as an item in the list returned by the above code. The first two have been completed for you: Pattern 1 returns `["go", "go", "go"]`, so we wrote 3; pattern 2 returns `["godo"]` and does not contain the string `"go"` as an item, so we wrote 0.

Each response is worth 1 point.

1. `pattern = r'go'` ___3___
2. `pattern = r'godo'` ___0___
3. `pattern = r'go.*'` _____
4. `pattern = r'.*go.*'` _____
5. `pattern = r'go{2}'` _____
6. `pattern = r'(go){1}'` _____
7. `pattern = r'(go)[dg1]'` _____
8. `pattern = r'[go](go)'` _____
9. `pattern = r'[go]*(go)'` _____

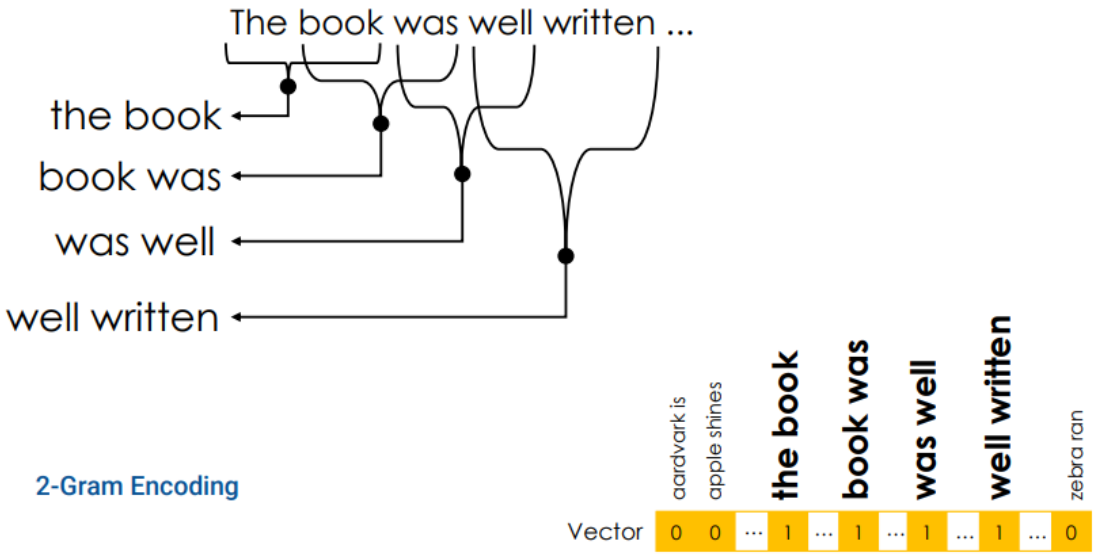
Word Embedding (Lec 06)

Bag of words

- A **bag** is another term for a multiset: an unordered collection which may contain multiple instances of each element
- Represents the text as the frequency with which each term appears in the vocabulary
- **Key idea**: The **more similar** two unit vectors are, the **larger their dot product** is

N-Gram Encoding

- N-Gram: “Bag-of- sequences-of-words”



TF-IDF

Motivation

- The bag of words model doesn't know which words are "important" in a document.
- How do we determine which words are important?
  - The **most common words** ("the", "has") often don't have much meaning!
  - The **very rare words** are also less important!

TF (Term frequency)

- Importance in a single document

$$\text{tf}(t, d) = \frac{\#\text{ of occurrences of } t \text{ in } d}{\text{total } \#\text{ of words in } d}$$

IDF (Inverse document frequency)

- the “rarity factor” of t across documents – **the larger idf(t) is, the more rare t is.**

$$\text{idf}(t) = \log\left(\frac{\text{total } \#\text{ of documents}}{\#\text{ of documents in which } t \text{ appears}}\right)$$

TF-IDF

- Goal: Quantify how well word t summarizes document d

$$\begin{aligned}\text{tfidf}(t, d) &= \text{tf}(t, d) \cdot \text{idf}(t) \\ &= \frac{\#\text{ of occurrences of } t \text{ in } d}{\text{total } \#\text{ of words in } d} \cdot \log\left(\frac{\text{total } \#\text{ of documents}}{\#\text{ of documents in which } t \text{ appears}}\right)\end{aligned}$$

Data on the Internet (Lec 07, 08)

HTTP

- Hypertext Transfer Protocol
- follows the **request-response** model

Request methods

two ways to make HTTP requests outside of a browser:

- From the command line, with `curl`.
- **From Python, with the `requests` package.**
  - `Get`: used to request data **from** a specified resource.

```
get_res = requests.get('https://bing.com')
```

  - `POST`: is used to **send** data to the server.

```
post_res = requests.post('https://httpbin.org/post', data={'name': 'King Triton'})
```

HTTP status codes

- `200` : Successful request
- `400` : Bad request
- `404` : Page not found
- `500` : Internal server error

Data formats

Responses typically come in one of two formats: **HTML or JSON.**

- The response body of a `GET` request is usually either JSON (when using an API) or HTML (when accessing a webpage).
- The response body of a `POST` request is usually JSON.

JSON

- **JavaScript Object Notation**
- JSON objects *resemble* Python dictionaries (but are not the same!).

```
family_tree = {'name': 'Grandma',
 'age': 94,
 'children': [{'name': 'Dad',
   'age': 60,
   'children': [{'name': 'Me', 'age': 24}, {'name': 'Brother', 'age': 22}]},
  {'name': 'My Aunt',
   'children': [{'name': 'Cousin 1', 'age': 34},
    {'name': 'Cousin 2',
     'age': 36,
     'children': [{'name': 'Cousin 2 Jr.', 'age': 2}]}]}]}
```

- Accessing values

```
family_tree['children'][0]['children'][1]['age']
```

- `json.load(f)` loads a JSON file from a file object.
- `json.loads(f)` loads a JSON file from a string.

APIs and scraping

- There are two ways of collecting data through a request:
  - By using a published **API** (application programming interface).
  - By **scraping** a webpage to collect its HTML source code.

HTML

- HTML (HyperText Markup Language) is **the** basic building block of the internet.

Useful tags:

Element	Description
`<html>`	the document
`<head>`	the header
`<body>`	the body
`<div>`	a logical division of the document
`<span>`	an *inline* logical division
`<p>`	a paragraph
`<a>`	an anchor (hyperlink)
`<h1>, <h2>, ...`	header(s)
`<img>`	an image

- tags can have **attributes**, which further specify how to display information on a webpage.

```
<img src="king-selfie.png" alt="A photograph of King Triton." width=500>
<!--`src`: source of image-->
<!--`alt`: text to display when the image fails to display-->
```

```
Click <a href="https://bing.com">this link</a> to search your keywords.
<!--destination of the hyperlink-->
```

```
<input id="username"> <!--usually unique-->
<div class="stat4710j">some text</div> <!--not necessarily unique-->
```

Parsing HTML using Beautiful Soup

- Beautiful Soup 4 is a Python HTML parser.
    - To "parse" means to "extract meaning from a sequence of symbols".

```
import requests
import bs4

html_string = requests.get(url)
soup = bs4.BeautifulSoup(html_string)

# Finding elements in a tree
soup.find(tag, attrs={...}) # finds the first instance of a tag
soup.find_all(tag, attrs={...}) # finds all instances of a tag

# Node Attributes
soup.find('p').text # gets the text between the opening and closing tags
soup.find('div').attrs # lists all attributes of a tag
soup.find('div').get('id') # gets the value of a tag attribute (must be
called directly on the node that contains the attribute you're looking for)
```

Data Erangling and EDA (Lec 09)

Key Data Properties to Consider in EDA

Structure -- the “shape” of a data file

Granularity -- how fine/coarse is each datum

Scope -- how (in)complete is the data

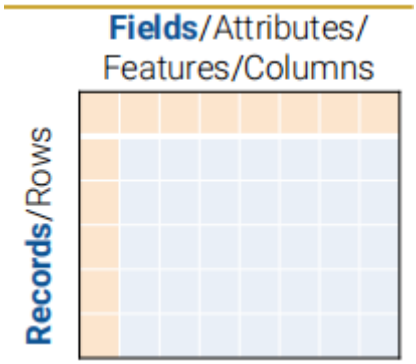
Temporality -- how is the data situated in time

Faithfulness -- how well does the data capture “reality”

Structure

- the “shape” of a data file

Rectangular Data

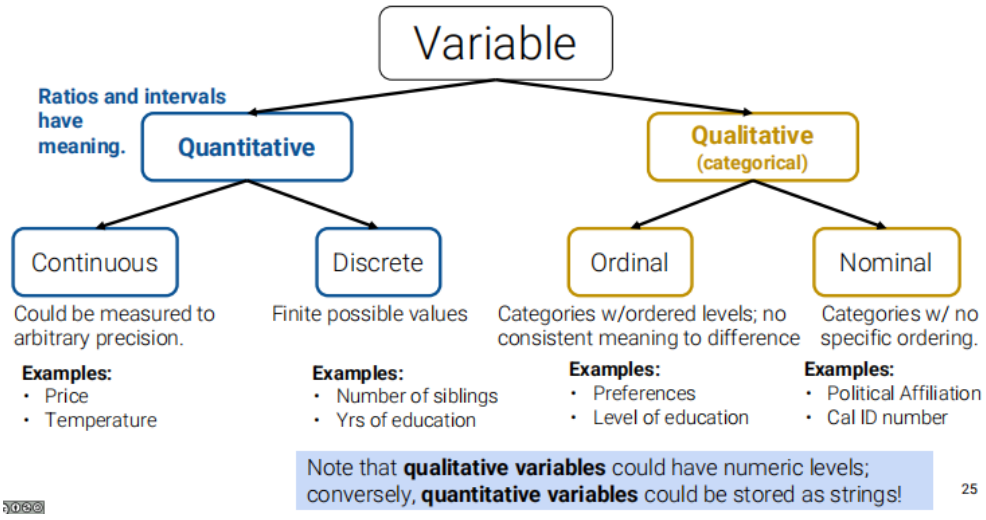


- Two kinds of rectangular data: **Tables** and **Matrices**.
  - Table: dataframes in R/Python and relations in SQL
    - Named columns with **different types**
    - Manipulated using data transformation languages (map, filter, group by, join, ...)
  - Matrics:
    - Numeric data of the **same type** (float, int, etc.)
    - Manipulated using linear algebra

File Format

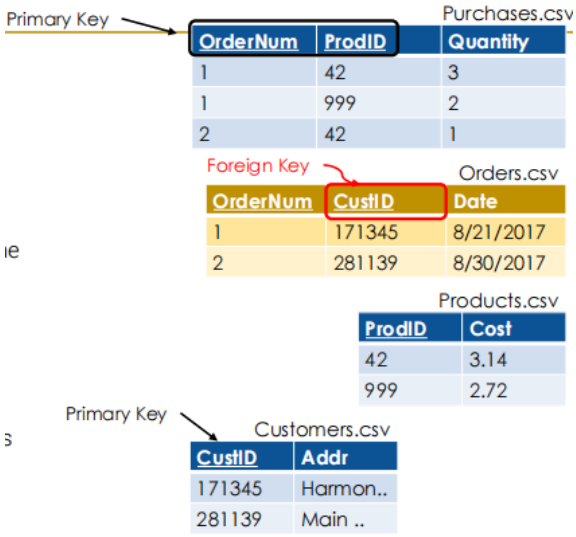
- TSV
- CSV
- JSON (Not rectangular)

Variable Type



Join Multiple files: Keys

- **Primary key**: the column or set of columns in a table that determine the values of the remaining columns
  - Primary keys are unique
- **Foreign keys**: the column or sets of columns that reference primary keys in other tables.



Granularity, Scope, Temporality

Granularity:

- How fine/coarse is each datum?

  What does each record represent?

Scope:

- How (in)complete is the data?
  - Does my data cover my area of interest?
  - Are my data too expensive?
  - Does my data cover the right time frame?

Temporality

- How is the data situated in time?
  - Data changes
  - Periodicity
  - Time zone
  - Null values: January 1st 1970, January 1st 1900, ...

Faithfulness (Missing Values)

- How well does the data capture "reality"?
- Deal with missing data
  - Drop records with missing values
  - Keep as `NaN`
  - Imputation/Interpolation
    - Average value
    - Random value
    - Predicted value

Reference

Boyuan Zhang, midtermRC_Part1, SP2024

Jingye Lin, midtermRC_Part1, SP2023

Zhitong Tang, Mid_RC_Part1, SU2023

STAT4710J Slides, SP2024

It's the annual Monopoly World Championship! The finalists: Shawn, Amanda, Neil, and Annie are playing Monopoly, a board game where players pay a price to buy properties, which can then generate income for them. Each property can be owned by only one player at a time. At the end of the game, the player with the most money wins.

Shawn wants to figure out which properties are most worth buying. He creates a DataFrame `income` with data on the current game state, shown on the **left**. He also finds a DataFrame `properties` with data on Monopoly properties, shown on the **right**.

Both tables have 28 rows. For brevity, only the first few rows of each DataFrame are shown.

	Player	Property	Income Generated
0	Shawn	Boardwalk	$425
1	Amanda	Park Place	$375
2	Neil	Marvin Gardens	$200
3	NaN	Kentucky Ave	NaN
4	Shawn	Pennsylvania Ave	$150
5	Annie	Oriental Ave	$50
6	Amanda	Baltic Ave	$60

`income`

	Property	Property Color	Purchase Price
0	Park Place	Dark Blue	350.0
1	Oriental Ave	Light Blue	100.0
2	Vermont Ave	Light Blue	100.0
3	Pacific Ave	Green	300.0
4	Boardwalk	Dark Blue	400.0
5	Illiniois Ave	Red	240.0
6	Atlantic Ave	Yellow	260.0

`properties`

Left table:

- Player is the name of the player, as a str.
- Property is a property currently owned by the player, as a str.
- Income Generated is the amount of income a player has earned from that property so far, as a str.

Right table:

- Property is the name of the property, as a str. There are 28 unique properties.
- Property Color is a color group that the property belongs to, as a str. There are 10 unique color groups, and each property belongs to a single group.
- Purchase Price is the price to buy the property, as a float.

(a) What is the granularity of the income table?

(b) Which of the following line(s) of code successfully returns a Series with the number of properties each player owns? **Select all that apply.**

- ▫ income.groupby('Player').agg(pd.value_counts)
- ▫ income['Player'].value_counts()
- ▫ income['Player', 'Property'].groupby('Player').size()
- ▫ income.groupby('Player').size()

(c) He now decides to calculate the amount of profit from each property. He wants to store this in a column called Profit in the income DataFrame.

i. To do this, he first has to transform the Income Generated column to be of a float datatype. Write one line of code to replace the old column with a new column, also called Income Generated, with the datatype modification described above. You may assume that each entry in Income Generated consists of a dollar sign ($) followed by a number, except for the NaN values.

ii. [3 Pts] Assuming that the answer to the last sub-part is correct, let's add a `Profit` column to the `income` DataFrame. **Fill in the following blanks to do this**, and please add arguments to function calls as you see appropriate.

Note: Profit is calculated by subtracting the purchase price from generated income.

```
combined_df = income.___A___(_____________B_____________)
income["Profit"] = ____________________C____________________
```

(d) [2 Pts] Regardless of your answer to the previous sub-part, assume we've successfully created the Profit column. Let's help Shawn see how well he's doing by finding the average profit he's made on the properties he owns. Which of the following line(s) of code does this? Select all that apply.

- ▫ income.groupby("Player")["Shawn"].agg(np.average)
- ▫ income[income["Player"]=="Shawn"]["Profit"].mean()
- ▫ income.loc[income["Player"]=="Shawn", "Profit"].mean()
- ▫ income.iloc[income["Player"]=="Shawn", "Profit"].mean()

1
(a) Property
(b) BD
(c) i. income['Income Generated'] = income['Income Generated'] .str[1:].astype(float)
ii. combined_df = income.merge(properties, on = "Property")
income["Profit"] = combined_df["Income Generated"] - combined_df["Purchase Price"]
(d) BC