

STA130HW1—YUYANG MA

September 13, 2024

1. Pick one of the datasets from the ChatBot session(s) of the TUT demo (or from your own ChatBot session if you wish) and use the code produced through the ChatBot interactions to import the data and confirm that the dataset has missing values

```
[1]: # feel free to just use the following if you prefer...
import pandas as pd
url = "https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/
      ↪data/2020/2020-05-05/villagers.csv"
df = pd.read_csv(url)
df.isna().sum()
```

```
[1]: row_n      0
      id        1
      name      0
      gender    0
      species   0
      birthday  0
      personality 0
      song      11
      phrase    0
      full_id   0
      url       0
      dtype: int64
```

2. Start a new ChatBot session with an initial prompt introducing the dataset you’re using and request help to determine how many columns and rows of data a pandas DataFrame has, and then

1. use code provided in your ChatBot session to print out the number of rows and columns of the dataset; and,
2. write your own general definitions of the meaning of “observations” and “variables” based on asking the ChatBot to explain these terms in the context of your dataset

```
[2]: df.shape
```

```
[2]: (391, 11)
```

Observations are rows in the dataset and they represent individual cases or records. Each observation

corresponds to a single row in a data table or DataFrame. Variables are columns in the dataset and they represent attributes or characteristics measured for each observation. They can describe different sides of the data and can be used to analyze relationships, patterns, and trends.

3. Ask the ChatBot how you can provide simple summaries of the columns in the dataset and use the suggested code to provide these summaries for your dataset

```
[3]: df.describe()
```

```
[3]:          row_n
count  391.000000
mean   239.902813
std    140.702672
min      2.000000
25%    117.500000
50%    240.000000
75%    363.500000
max    483.000000
```

```
[4]: df.isna().sum()
```

```
[4]: row_n      0
id           1
name         0
gender       0
species      0
birthday     0
personality  0
song        11
phrase       0
full_id      0
url          0
dtype: int64
```

```
[6]: df.describe(include=['object'])
```

```
[6]:          id      name gender species birthday personality      song \
count      390      391     391     391      391          391      380
unique      390      391       2      35      361           8       92
top  admiral  Admiral   male    cat    1-27        lazy  K.K. Country
freq         1         1    204     23         2         60         10

          phrase      full_id \
count      391          391
unique      388          391
top    wee one  villager-admiral
freq         2                  1
```

	url
count	391
unique	391
top	https://villagerdb.com/images/villagers/thumb/...
freq	1

4. If the dataset you're using has (a) non-numeric variables and (b) missing values in numeric variables, explain (perhaps using help from a ChatBot if needed) the discrepancies between size of the dataset given by `df.shape` and what is reported by `df.describe()` with respect to (a) the number of columns it analyzes and (b) the values it reports in the “count” column. `df.shape` is able to provide the overall dimensions of the DataFrame, including all columns and rows. `df.describe()` can provide statistical summaries only for numeric columns and reflects the count of non-null values in those columns.

5. Use your ChatBot session to help understand the difference between the following and then provide your own paraphrasing summarization of that difference. Attributes like `df.shape` can provide direct access to properties of an object and do not involve function calls. Methods like `df.describe()` are able to involve function calls that perform actions or computations on the object.

6. The `df.describe()` method provides the ‘count’, ‘mean’, ‘std’, ‘min’, ‘25%’, ‘50%’, ‘75%’, and ‘max’ summary statistics for each variable it analyzes. Give the definitions (perhaps using help from the ChatBot if needed) of each of these summary statistics: count: The number of non-null entries in the column. This indicates how many values are present for each column in the DataFrame. mean: The average value of the column. It is the sum of all values divided by the count of non-null values. std: A measure of the dispersion or spread of the values around the mean. It quantifies how much the values in the column deviate from the mean. min: The smallest value in the column. 25%: The value below which 25% of the data falls. 50%: The median value of the column. It is the middle value where 50% of the data is below and 50% is above. 75%: The value below which 75% of the data falls. Also known as the third quartile. max: The largest value in the column.

7. Missing data can be considered “across rows” or “down columns”. Consider how `df.dropna()` or `del df['col']` should be applied to most efficiently use the available non-missing data in your dataset and briefly answer the following questions in your own words

1. Provide an example of a “use case” in which using `df.dropna()` might be preferred over using `del df['col']`

```
[ ]: import pandas as pd

# Sample DataFrame
df = pd.DataFrame({
    'age': [25, 30, 35, None, 40],
    'income': [50000, None, 60000, 55000, None],
    'education_level': ['Bachelor', 'Master', 'PhD', 'Bachelor', 'PhD']
})
```

```

})

print("Original DataFrame:")
print(df)

# Drop rows where 'income' is NaN
df_cleaned = df.dropna(subset=['income'])

print("\nDataFrame after dropna():")
print(df_cleaned)

```

`df.dropna()` use is when you need to handle missing values in columns while preserving the rest of the data. It allows you to clean the data selectively based on missing values. `del df['col']` use is when you want to remove an entire column from the DataFrame, irrespective of missing values, often due to the column is not relevant to your analysis.

2. Provide an example of “the opposite use case” in which using `del df['col']` might be preferred over using `df.dropna()`

```

[8]: import pandas as pd

# Sample DataFrame with irrelevant columns
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie', None],
    'email': ['alice@example.com', 'bob@example.com', None, 'charlie@example.
    ↪com'],
    'age': [25, 30, 35, 40],
    'income': [50000, 60000, None, 55000]
})

print("Original DataFrame:")
print(df)

# Remove the 'name' and 'email' columns
del df['name']
del df['email']

print("\nDataFrame after deleting 'name' and 'email' columns:")
print(df)

```

Original DataFrame:

	name	email	age	income
0	Alice	alice@example.com	25	50000.0
1	Bob	bob@example.com	30	60000.0
2	Charlie	None	35	NaN
3	None	charlie@example.com	40	55000.0

DataFrame after deleting 'name' and 'email' columns:

	age	income
0	25	50000.0
1	30	60000.0
2	35	NaN
3	40	55000.0

`del df['col']` use is when you want to completely remove entire columns that are not relevant to your analysis, regardless of their content. `df.dropna()` use is when you need to handle missing values in specific columns and keep the remaining data.

3. Discuss why applying `del df['col']` before `df.dropna()` when both are used together could be important

first of all is Removing Columns: use `del df['col']` to remove irrelevant columns before applying `df.dropna()` ensures that `dropna()` operates only on the relevant columns, improving both efficiency and clarity. next one is efficiency: decrease the size of the DataFrame, making data cleaning operations faster and more manageable. the last is accuracy: ensure that `dropna()` is applied to only the necessary columns, leading to more accurate results.

4. Remove all missing data from one of the datasets you're considering using some combination of `del df['col']` and/or `df.dropna()` and give a justification for your approach, including a "before and after" report of the results of your approach for your dataset.

Initial DataFrame is that contains missing values and potentially irrelevant columns. Approach: Remove Irrelevant Columns is using `del` to remove columns like name that are not needed. Remove Rows with Missing Values is using `dropna()` to ensure all remaining rows have complete data. Outcome is cleaned of missing values and irrelevant columns, resulting in a more focused and complete DataFrame ready for analysis.

8. Give brief explanations in your own words for any requested answers to the questions below 1.rouping: `df.groupby("class")` divides the dataset into subsets based on the class column. Descriptive Statistics: `.describe()` provides a summary of statistics for the age column within each group. Use Case: This approach is useful to compare and contrast statistical measures across different categories (in this case, passenger classes) to understand how the variable of interest (age) varies by category. By applying `df.groupby("class")["age"].describe()`, you can gain insights into how the age distribution differs among the different passenger classes on the Titanic, which can be valuable for understanding patterns and differences within the data.

2. Complete Dataset vs. Grouped Analysis: `df.describe()` gives a broad overview of missing data in the entire dataset, while `df.groupby("col1")["col2"].describe()` provides more granular insights into missing data within specific subsets, which can be crucial for understanding group-specific issues and for targeted data cleaning or analysis.

3. Ease of Use: ChatGPT provides immediate, context-aware help and can guide you through troubleshooting in real-time. This is particularly useful for understanding and fixing errors interactively. Depth of Information: Google searches offer extensive resources and can be very useful for finding detailed solutions and understanding complex issues. It is often better for broader or more specific queries not limited to immediate troubleshooting.

```
[ ]: with chatgpt:
      summary:
```

1. Introducing Errors

Let's intentionally introduce a few common errors into the code:

Syntax Error: Missing a parenthesis or comma.

NameError: Using a variable or method that hasn't been defined.

KeyError: Accessing a column that doesn't exist in the DataFrame.

TypeError: Using a method incorrectly, such as passing the wrong type of argument.

Error-Prone Code:

python

```
import pandas as pd
import seaborn as sns

# Load the Titanic dataset
titanic = sns.load_dataset('titanic')

# Syntax Error: Missing closing parenthesis
print("Initial Titanic DataFrame:")
print(titanic.head())

# NameError: Misspelling the method name
print(titanic.grouby('class')['age'].describe())

# KeyError: Accessing a column that doesn't exist
print(titanic.groupby('class')['nonexistent_column'].describe())

# TypeError: Passing a string instead of a column name
print(titanic.groupby('class')['age'].describe(incorrect_argument="value"))
```

2. Error Reporting and Troubleshooting in ChatGPT Session

Error 1: Syntax Error

arduino

```
File "<stdin>", line 4
    print("Initial Titanic DataFrame:")
    ^
```

SyntaxError: unexpected EOF while parsing

Error 2: **NameError**

csharp

AttributeError: 'DataFrame' object has no attribute 'grouby'

Error 3: **KeyError**

vbnet

KeyError: 'nonexistent_column'

Error 4: **TypeError**

css

TypeError: describe() got an unexpected keyword argument 'incorrect_argument'

ChatGPT Troubleshooting:

Syntax Error:

Fix: Add the missing parenthesis.

python

```
print("Initial Titanic DataFrame:")
```

NameError:

Fix: Correct the method name **from grouby** to **groupby**.

python

```
print(titanic.groupby('class')['age'].describe())
```

KeyError:

Fix: Ensure the column name exists **in** the DataFrame. Check available columns

↳ using `titanic.columns`.

python

```
print(titanic.groupby('class')['age'].describe())
```

TypeError:

Fix: Remove **or** correct the argument based on `describe()` method requirements.

python

```
print(titanic.groupby('class')['age'].describe())
```

3. Google Search for Error Fixes

Searching on Google:

Syntax Error: Search for "SyntaxError unexpected EOF while parsing". This

↳ usually leads to discussions or documentation on common syntax errors.

NameError: Search for "AttributeError: 'DataFrame' object has no attribute

↳ 'grouby'". This directs you to information on spelling mistakes or method

↳ availability.

KeyError: Search for "KeyError: 'nonexistent_column'". This will lead you to

↳ explanations about checking DataFrame columns and handling missing keys.

TypeError: Search for "TypeError: describe() got an unexpected keyword argument

↳ 'incorrect_argument'". This will show documentation or discussions on valid

↳ arguments for the `describe()` method.

Comparing the Approaches:

(a) ChatGPT Session:

Pros:

Interactive: You get immediate feedback **and** suggestions based on the errors you encounter.

Contextual: You can ask follow-up questions **for** clarification on specific issues.

Cons:

Limited Scope: Might **not** provide exhaustive information **or** examples beyond the immediate context of the errors.

(b) Google Search:

Pros:

Extensive Resources: Access to a wide **range** of documentation, forums, **and** examples.

Varied Solutions: Might offer multiple perspectives **and** solutions **from** different sources.

Cons:

Search Precision: Requires precise search queries to find relevant solutions.

Self-Diagnosis: You might need to diagnose the issue yourself based on the information available.

Conclusion

Ease of Use: ChatGPT provides immediate, context-aware help **and** can guide you through troubleshooting **in** real-time. This **is** particularly useful **for** understanding **and** fixing errors interactively.

Depth of Information: Google searches offer extensive resources **and** can be very useful **for** finding detailed solutions **and** understanding **complex** issues. It **is** often better **for** broader **or** more specific queries **not** limited to immediate troubleshooting.

For quick fixes **and** interactive help, working **with** ChatGPT **is** beneficial. For comprehensive, detailed solutions **or** **for** researching less common issues, Google searches might be more effective.

give me summary above

ChatGPT

ChatGPT

Here's a summary of how to handle **and** troubleshoot coding errors using ChatGPT versus Google:

Error-Prone Code Example

Syntax Error: Missing closing parenthesis.

NameError: Misspelled method name (grouby instead of groupby).

KeyError: Accessing a non-existent column.

TypeError: Incorrect argument **in** a method.

Troubleshooting **with** ChatGPT

Error Handling:

Syntax Error: Fix by correcting syntax issues (e.g., adding missing
↳ parentheses).

NameError: Correct method name **from grouby** to groupby.

KeyError: Verify column names exist **in** the DataFrame.

TypeError: Ensure correct arguments are passed to the method.

Pros:

Interactive **and** immediate feedback.

Context-aware suggestions.

Cons:

Limited scope **and** may **not** cover **all** possible fixes **or** examples.

Troubleshooting **with** Google Search

Search Queries:

Syntax Error: Search **for** specific syntax error messages.

NameError: Search **for** method name errors **and** common misspellings.

KeyError: Search **for** handling missing columns **or** keys.

TypeError: Search **for** method argument errors **and** documentation.

Pros:

Extensive resources **and** varied solutions.

Detailed documentation **and** examples.

Cons:

Requires precise search terms.

Can involve more **self**-diagnosis.

Conclusion

ChatGPT: Best **for** immediate, interactive help **and** context-specific guidance.

Google Search: Better **for** comprehensive, detailed solutions **and** broader
↳ research.

Using ChatGPT **is** useful **for** quick, interactive problem-solving, **while** Google
↳ searches provide broader information **and** detailed solutions.