STA130 HW week1 ziang chen 1010885295

September 11, 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]:
```python
# 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]:
```python
2.1
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/
↪master/titanic.csv')
```

```
rows, columns = df.shape

print(f"dataset contains{rows} rows and {columns} columns ")

2.2
Observations are the individual entries in your dataset (each passenger).
Variables are the different pieces of information collected about each␣
↪observation (such as age, sex, or fare).
so the variables are the details in the Observation in the titannic dataset.
```

```
dataset contains891 rows and 12 columns
```

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]:
```
3.
import pandas as pd

# Load the Titanic dataset
df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/
↪master/titanic.csv')

# Generate a simple statistical summary for each column in the dataset
summary = df.describe()

# Print the statistical summary
print("Column summaries for the dataset:")
print(summary)
```

```
Column summaries for the dataset:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
```

```
75%        0.000000   31.000000
max        6.000000  512.329200
```

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

[4]:
```python
4.
import pandas as pd

# Load the Titanic dataset
df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/
↪master/titanic.csv')

# Display the shape of the dataset
shape = df.shape
print(f"The dataset has {shape[0]} rows and {shape[1]} columns.")

# Generate a statistical summary of the dataset
summary = df.describe()
print("\nStatistical summary of the dataset:")
print(summary)
```

```
The dataset has 891 rows and 12 columns.

Statistical summary of the dataset:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

5. Use your ChatBot session to help understand the difference between the following and then provide your own paraphrasing summarization of that difference

[5]:
```
5.
df.shape gives you a tuple with two obvious values: the number of rows and the␣
 ↪number of columns in the Dataset.
df.describe()only analyzes numeric columns in common situations.

An attribute in Python, like df.shape, is a property of an object that stores␣
 ↪and provides data directly, without needing any further action or␣
 ↪computation.
On the other hand, a method, like df.describe(), is an action or function␣
 ↪associated with an object that processes or manipulates the object's data,␣
 ↪often performing calculations or other operations.
```

```
  Cell In[5], line 6
    On the other hand, a method, like df.describe(), is an action or function␣
 ↪associated with an object that processes or manipulates the object's data,␣
 ↪often performing calculations or other operations.
                                                                        ^
 ↪
SyntaxError: unterminated string literal (detected at line 6)
```

[6]:
```
the link of the chatGPT page I have used in this article is
https://chatgpt.com/c/66ddeb8a-1634-8003-b7cb-d2471c4096ef
```

```
  Cell In[6], line 2
    https://chatgpt.com/c/66ddeb8a-1634-8003-b7cb-d2471c4096ef
                            ^
SyntaxError: invalid decimal literal
```

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

[ ]:
```
6.
Count:
Definition: The number of non-null (or non-missing) entries for each column.

Mean:
Definition: The average of the data points in a column. It is calculated by␣
 ↪summing all the non-null values and dividing by the count.

Std (Standard Deviation):
```

```
Definition: A measure of the amount of variation or dispersion in a set of␣
↪values. It shows how much the data deviates from the mean.

Min (Minimum):
Definition: The smallest value in the column.

25% (25th Percentile or First Quartile):
Definition: The value below which 25% of the data points fall. It is also known␣
↪as the first quartile (Q1).

50% (50th Percentile or Median):
Definition: The value that divides the data into two equal halves.
This is the middle value when the data points are sorted in ascending order.

75% (75th Percentile or Third Quartile):
Definition: The value below which 75% of the data points fall. It is also known␣
↪as the third quartile (Q3).

Max (Maximum):
Definition: 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

[ ]:
```
7.1
Imagine you're working with a healthcare dataset that tracks patients' vital␣
↪signs, including heart rate, blood pressure, and temperature, along with␣
↪demographic information like age and gender. In this dataset, each row␣
↪represents a patient, and each column represents a specific measurement or␣
↪demographic detail.

Scenario: Let's say the dataset has some missing data scattered across a few␣
↪rows for the vital signs (e.g., heart rate, blood pressure). These vital␣
↪sign columns are crucial for your analysis, and each one provides unique and␣
↪valuable information. However, some patients might have one or more missing␣
↪measurements.

Why Use df.dropna()?

In this scenario, you wouldn't want to delete any of the vital sign columns␣
↪entirely because that would remove important information for all patients.␣
↪Instead, using df.dropna() to remove only the rows with missing data might␣
↪be more appropriate. This way, you keep all the vital sign columns intact␣
↪and only lose the specific rows where the data is incomplete.
```

```
By using df.dropna(), you ensure that your dataset retains as much of the
 ↪complete data as possible across all vital signs, which is crucial for
 ↪accurate analysis in healthcare research.
--------------------------------------------------------------------------------

7.2
Use Case Example for del df['col'] Over df.dropna():

Imagine you are working with a marketing dataset that includes customer
 ↪information, such as their email address, age, location, and various
 ↪responses to a marketing campaign. One of the columns in this dataset is
 ↪"Phone Number," but it turns out that 90% of the entries in this column are
 ↪missing because not all customers provided their phone numbers.

Scenario: The "Phone Number" column has a very high percentage of missing
 ↪values, making it largely unusable for analysis. If you were to use df.
 ↪dropna() to remove rows with missing phone numbers, you would end up losing
 ↪90% of your dataset, which is not desirable because the other columns (like
 ↪age, location, and campaign responses) are still valuable and mostly
 ↪complete.

Why Use del df['col']?

In this case, it would be more efficient to simply delete the entire "Phone
 ↪Number" column using del df['Phone Number'] rather than dropping rows with
 ↪missing data. This way, you preserve the majority of your dataset and focus
 ↪on the other columns that contain valuable information for your analysis.

By using del df['col'], you avoid losing a significant portion of your dataset,
 ↪allowing you to retain more data for your analysis, which is crucial when
 ↪other columns are complete and relevant to your study.

# Assuming df is your DataFrame and 'Phone Number' is the column with 90%
 ↪missing data

# Delete the 'Phone Number' column
del df['Phone Number']

# Now df contains all rows, minus the 'Phone Number' column
In this scenario, using del df['col'] helps you maintain the integrity of your
 ↪dataset by removing only the problematic column without discarding
 ↪potentially useful rows of data.
--------------------------------------------------------------------------------

7.3
```

```
applying del df['col'] before df.dropna() is important because it allows you to␣
 ↪remove irrelevant or problematic columns first,
thereby minimizing unnecessary data loss and ensuring that the df.dropna()␣
 ↪operation is as efficient and targeted as possible.
This approach helps maintain the quality and usability of your dataset.
----------------------------------------------------------------------------------------------------


7.4
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [24, None, 22, 23, None],
    'City': ['New York', 'Los Angeles', 'Chicago', None, 'Boston'],
    'Salary': [70000, 80000, None, 65000, None]
}

df = pd.DataFrame(data)

Initial DataFrame
      Name   Age         City   Salary
0    Alice  24.0     New York  70000.0
1      Bob   NaN  Los Angeles  80000.0
2  Charlie  22.0      Chicago      NaN
3    David  23.0         None  65000.0
4      Eve   NaN       Boston      NaN

Approach:
Remove columns with excessive missing data:
Use del df['col'] to remove any column that has a significant amount of missing␣
 ↪data. In this example, if a column is more than 50% missing, it's a␣
 ↪candidate for removal.

Remove rows with any missing values:
Use df.dropna() to remove rows that contain any NaN values, ensuring that the␣
 ↪remaining data is complete.

Step 1: Remove Columns with Excessive Missing Data
# Check the percentage of missing data in each column
missing_data = df.isnull().mean()

# Remove columns where more than 50% of the data is missing
cols_to_drop = missing_data[missing_data > 0.5].index
for col in cols_to_drop:
    del df[col]

Justification:
```

```
The Salary column has 2 out of 5 missing values, which is 40%, so it stays.
However, if it were over 50%, it would be removed. Since no columns are over␣
↪50%, none are deleted in this example.

Step 2: Drop Rows with Any Missing Values
# Drop rows with any missing values
df_cleaned = df.dropna()

After Cleaning: Resulting DataFrame
       Name   Age      City  Salary
0  Alice  24.0  New York  70000.0

Justification:
After dropping rows with any missing values, only one row remains.
This approach ensures that the dataset is complete but significantly reduces␣
↪its size, which could be a trade-off depending on the use case.

df.dropna()--This approach is stringent, ensuring the final dataset has no␣
↪missing values,
which is critical in many analysis scenarios. However, the trade-off is a␣
↪potential loss of data.
The method of column deletion based on a threshold and row deletion ensures␣
↪that only the most reliable data is retained.
```

8. Give brief explanations in your own words for any requested answers to the questions below

1. Use your ChatBot session to understand what `df.groupby("col1")["col2"].describe()` does and then demonstrate and explain this using a different example from the "titanic" data set other than what the ChatBot automatically provide for you

   If needed, you can help guide the ChatBot by showing it the code you've used to download the data **AND provide it with the names of the columns** using either a summary of the data with `df.describe()` or just `df.columns` as demonstrated here

2. Assuming you've not yet removed missing values in the manner of question "7" above, `df.describe()` would have different values in the `count` value for different data columns depending on the missingness present in the original data. Why do these capture something fundamentally different from the values in the `count` that result from doing something like `df.groupby("col1")["col2"].describe()`?

   Questions "4" and "6" above address how missing values are handled by `df.describe()` (which is reflected in the `count` output of this method); but, `count` in conjunction with `group_by` has another primary function that's more important than addressing missing values (although missing data could still play a role here).

3. Intentionally introduce the following errors into your code and report your opinion as to whether it's easier to (a) work in a ChatBot session to fix the errors, or (b) use google to search for and fix errors: first share the errors you get in the ChatBot session and see if you

can work with ChatBot to troubleshoot and fix the coding errors, and then see if you think a google search for the error provides the necessary toubleshooting help more quickly than ChatGPT

1. Forget to include `import pandas as pd` in your code Use Kernel->Restart from the notebook menu to restart the jupyter notebook session unload imported libraries and start over so you can create this error When python has an error, it sometimes provides a lot of "stack trace" output, but that's not usually very important for troubleshooting. For this problem for example, all you need to share with ChatGPT or search on google is `"NameError: name 'pd' is not defined"`
2. Mistype "titanic.csv" as "titanics.csv" If ChatBot troubleshooting is based on downloading the file, just replace the whole url with "titanics.csv" and try to troubleshoot the subsequent `FileNotFoundError: [Errno 2] No such file or directory: 'titanics.csv'` (assuming the file is indeed not present) Explore introducing typos into a couple other parts of the url and note the slightly different errors this produces
3. Try to use a dataframe before it's been assigned into the variable You can simulate this by just misnaming the variable. For example, if you should write `df.groupby("col1")["col2"].describe()` based on how you loaded the data, then instead write `DF.groupby("col1")["col2"].describe()` Make sure you've fixed your file name so that's not the error any more
4. Forget one of the parentheses somewhere the code For example, if the code should be `pd.read_csv(url)` the change it to `pd.read_csv(url`
5. Mistype one of the names of the chained functions with the code For example, try something like `df.group_by("col1")["col2"].describe()` and `df.groupby("col1")["col2"].describle()`
6. Use a column name that's not in your data for the `groupby` and column selection For example, try capitalizing the columns for example replacing "sex" with "Sex" in `titanic_df.groupby("sex")["age"].describe()`, and then instead introducing the same error of "age"
7. Forget to put the column name as a string in quotes for the `groupby` and column selection, and see if the ChatBot and google are still as helpful as they were for the previous question For example, something like `titanic_df.groupby(sex)["age"].describe()`, and then `titanic_df.groupby("sex")[age].describe()`

[ ]:
```
8.1
This command in Pandas performs the following operations:

df.groupby("col1"):
It groups the DataFrame df by the unique values in the column col1.
This means that it splits the data into different groups, where each group␣
↪contains rows that share the same value in col1.

["col2"]:
After grouping the data, it selects the column col2 within each group.
```

```
So, you're focusing on the col2 values for each unique group formed by col1.
.describe():

This method generates descriptive statistics for the col2 values within each␣
 ↪group.
It provides summary statistics like count, mean, standard deviation (std),␣
 ↪minimum (min), maximum (max), and percentiles (25%, 50%, 75%).

import seaborn as sns

# Load Titanic dataset
titanic = sns.load_dataset("titanic")

# Group by 'embarked' and describe 'fare'
fare_description = titanic.groupby("embarked")["fare"].describe()

fare_description

Explanation:
Grouping by embarked: We group the Titanic dataset by the embarked column,
    which represents the port where passengers boarded the ship (C, Q, S for␣
 ↪Cherbourg, Queenstown, and Southampton, respectively).

Selecting fare: Within each group (based on the embarkation port), we focus on␣
 ↪the fare column, which shows how much each passenger paid.

Describing fare: The .describe() method provides statistical summaries for the␣
 ↪fare column within each group.
This includes how many passengers boarded at each port (count), the average␣
 ↪fare (mean), and the variability in fares (std),
along with other statistics like minimum fare, maximum fare, and key␣
 ↪percentiles.
----------------------------------------------------------------------------------------------------

8.2
The difference between the count values in df.describe() and those from df.
 ↪groupby("col1")["col2"].
describe() is rooted in what each method is summarizing and how missing data␣
 ↪affects the results.

1. df.describe()
What It Does:

This method provides summary statistics for all numerical columns in the entire␣
 ↪DataFrame df.
```

```
The count value in df.describe() represents the number of non-missing (non-NaN)␣
↪values in each column across the entire DataFrame.
Impact of Missing Data:

If a column has missing data (NaN), the count for that column will be lower␣
↪than the total number of rows in the DataFrame.
This count reflects the number of valid, non-missing entries in that specific␣
↪column.
What It Captures:

The count in df.describe() captures the overall completeness of data in each␣
↪column.
It indicates how much usable data you have in each column when considering the␣
↪entire dataset without any grouping.

2. df.groupby("col1")["col2"].describe()
What It Does:

This method first groups the data by the values in col1, then it computes␣
↪summary statistics for col2 within each group.
The count in this context represents the number of non-missing (non-NaN) values␣
↪in col2 for each group defined by col1.
Impact of Missing Data:

Missing values in col2 will reduce the count for that specific group.
However, the grouping ensures that you see how much data is missing or present␣
↪within each subgroup, not just overall.
What It Captures:

The count here captures the amount of valid data in col2 within each group␣
↪defined by col1.
It provides insights into how complete the data is within these groups,
which can reveal patterns or biases related to specific categories or segments␣
↪of your data.
Fundamental Difference:
Overall Completeness (df.describe()):

The count from df.describe() gives you an overall sense of how complete each␣
↪column is across the entire dataset.
It's a general overview of data availability in each column.
Group-Specific Completeness (df.groupby("col1")["col2"].describe()):

The count from df.groupby("col1")["col2"].describe() provides a more granular␣
↪view.
It shows how much data you have in col2 for each subgroup of col1.
```

```
This is important when analyzing patterns within different segments of your␣
↪data,
as it helps to understand data availability and potential biases or imbalances␣
↪within specific groups.

Example:
If you're analyzing the Titanic dataset and use df.describe(),
the count for the age column might be lower than the total number of passengers␣
↪due to missing age values.

If you then group by pclass and use df.groupby("pclass")["age"].describe(),
the count will tell you how many passengers in each class have valid age data.
This can reveal, for example, whether certain classes have more missing age␣
↪data than others,
which could affect the interpretation of any age-related analysis within those␣
↪groups.

In summary, df.describe() gives you an overall completeness metric for each␣
↪column,
while df.groupby("col1")["col2"].describe() breaks it down by groups,
offering more detailed insights into the distribution and completeness of your␣
↪data within those specific categories.
--------------------------------------------------------------------------------------------

8.3
A.NameError: name 'pd' is not defined
    ChatGPT Experience:
When I encounter the NameError, I can simply describe the issue or share the␣
↪error message with ChatGPT. It will likely recognize that the error is due␣
↪to not importing the Pandas library and suggest adding import pandas as pd␣
↪at the top of the script.
Response Time: Immediate and contextual, ChatGPT provides a direct fix for the␣
↪error.

    Google Search Experience:
Searching for the error message on Google would yield many relevant results,␣
↪often with the correct answer at the top. However, I'd need to sift through␣
↪some results to find one that matches the context of my specific error.
Response Time: Quick, but might involve some extra steps to ensure the solution␣
↪is relevant to my specific problem.

    Conclusion: ChatGPT is slightly faster and more contextual for this simple␣
↪error.

B.FileNotFoundError: [Errno 2] No such file or directory: 'titanics.csv'
        ChatGPT Experience:
```

```
Upon sharing this error, ChatGPT will suggest checking the filename and ensuring it is correct. If the file is indeed not present, it might suggest verifying the path or checking if the file exists in the directory.
Response Time: Immediate and relevant, providing troubleshooting steps directly related to file handling.

        Google Search Experience:
Searching for this error will provide solutions that typically involve checking file paths or filenames. Google may point to resources explaining file handling errors and ways to resolve them.
Response Time: Quick, but again requires some context to apply the solution correctly.

        Conclusion: ChatGPT is more targeted for this specific error, while Google can provide more generalized solutions.

C.NameError: name 'DF' is not defined
    The same as A and B.

D.SyntaxError: unexpected EOF while parsing
        ChatGPT Experience:
ChatGPT will likely recognize that a syntax error such as a missing parenthesis is causing the problem. It will suggest checking for unmatched or missing parentheses in the code.
Response Time: Fast, with a clear explanation.

        Google Search Experience:
Google results will explain SyntaxError, and typically, solutions involve checking for missing or extra parentheses. However, finding the exact spot where the parenthesis is missing could take time.
Response Time: A bit slower, as it may require more manual checking.

        Conclusion: ChatGPT is more efficient at pinpointing the exact issue, while Google provides more general guidance.

E.AttributeError: 'DataFrame' object has no attribute 'group_by'
        ChatGPT Experience:
ChatGPT will identify that the method group_by is incorrect and should be groupby. It might also suggest reviewing the documentation for the correct usage of Pandas methods.
Response Time: Instant and corrective, offering the correct method name.

        Google Search Experience:
Searching this error will return solutions explaining that the method is incorrect and suggest checking the spelling or documentation. However, it may take an extra step to find the exact answer.
```

```
Response Time: Slightly slower, as it requires verifying the correct method name.

        Conclusion: ChatGPT is faster and more direct for fixing function name typos.

F.KeyError: 'Sex'
        ChatGPT Experience:
ChatGPT will likely point out that the column name is case-sensitive and suggest using the correct column name as it appears in the DataFrame.
Response Time: Immediate, with specific advice on column name case-sensitivity.

        Google Search Experience:
Google will provide explanations on KeyError and often suggest checking column names for typos or case sensitivity. Finding the exact cause could take a bit more effort.
Response Time: A bit slower, depending on the relevance of the search results.

        Conclusion: ChatGPT is quicker at resolving case-sensitive issues in column names.

G.NameError: name 'sex' is not defined
        ChatGPT Experience:
ChatGPT will likely recognize that the column name is not in quotes and suggest that it should be treated as a string, i.e., titanic_df.groupby("sex")["age"].describe().
Response Time: Fast, with an immediate solution.

        Google Search Experience:
Google will explain NameError, but it might take more steps to realize that the issue is the missing quotes around the column name.
Response Time: Slower, as it may require more detective work to pinpoint the cause.

        Conclusion: ChatGPT is more direct and efficient in identifying the need for quotes around column names.

 Final Opinion
ChatGPT is generally faster and more context-aware in providing solutions to these coding errors. It offers targeted advice that directly addresses the problem with minimal effort needed from the user. Google is still a powerful tool, especially for more complex issues or when detailed documentation and community discussions are required. However, it often involves more steps to identify the exact issue and find the right solution.
```

```
For simple to moderate coding errors, working in a ChatBot session (a) tends to
    ↪be more efficient and user-friendly.
```

9. Have you reviewed the course wiki-textbook and interacted with a ChatBot (or, if that wasn't sufficient, real people in the course piazza discussion board or TA office hours) to help you understand all the material in the tutorial and lecture that you didn't quite follow when you first saw it?

[ ]:
```
9.
Yes, I have reviewed the wiki-textbook and use the chatbox to assist my
    ↪learning.
I am glad that there are uncountable details are provided,in which I can find a
    ↪lot of pathes to reach the end of a topic or the answer of the question!
```

[ ]:
```
the link of the chatGPT website I have used--
https://chatgpt.com/c/66df58c0-6bc0-8003-9c93-49a6948f5ac4
```