Q6. **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**

1. Count: The number of non-null entries for each variable or column.
2. Mean: The average value of the data for each variable. Can be calculated by dividing the sum of all values into the number of values
3. Std: Std is a standard deviation which is the measure of the amount of variation of the values of a variable about its mean.
4. Min: The smallest value in the dataset for each variable
5. 25%: First Quartule/ The value under 25%.
6. 50%: Median, The middle value of the dataset when it is sorted in ascending order
7. 75%: Third Quartile, The value above 75%
8. Max: The largest value in the dataset for each variable

Q7. **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 peferred over using `del df['col']`

If there is only small number of rows which contain missing values and other columns are complete, it is better to use df.dropna to delete the rows with missing values

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

If most of columns have missing values, it would be better to remove the entire column using del df[‘col’]

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

Deleting a column with a significant amount of missing data before using df.dropna() can prevent unnecessary loss of rows. If you apply df.dropna() first, you might inadvertently lose rows due to missing values in a column you could have discarded. By using del df[‘col’] first, you clean the dataset and preserve as many rows as possible by focusing on meaningful columns.

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.
    1. Identify missing data
    2. Depend on the amount of data that is missing, chose either df.dropna() or del df['col']

       Justification of the approach:

        - Using df.dropna() to remove rows with missing data ensures that the remaining dataset is complete
        - If the missing values are scattered across multiple columns, deleting columns might result in the loss of valuable information

Q8. **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

   The operation df.groupby("col1")["col2"].describe() groups the DataFrame by col1, selects col2, and then provides descriptive statistics (like count, mean, std, min, and percentiles) for each group in col2.

   Using the Titanic dataset example, grouping by Pclass and describing Age:

    - Pclass 1 passengers: Average age is 38.23 years, with a standard deviation of 14.80 years. Ages range from 0.92 to 80 years.
    - Pclass 2 passengers: Average age is 29.88 years.
    - Pclass 3 passengers: Average age is 25.14 years. This analysis shows the distribution of passengers' ages by class.

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()`?

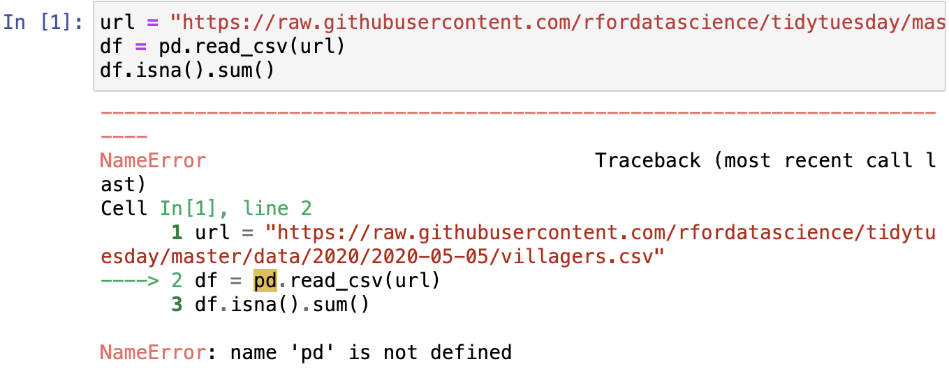
- df.describe(): The count reflects the total number of non-missing values per column across the entire dataset, without considering any groupings.

- df.groupby("col1")["col2"].describe(): The count reflects the number of non-missing values in col2 within each unique group of col1. It captures the completeness of data within each group, rather than the dataset as a whole.

  This distinction is important because groupby allows you to explore how missing data (or any other characteristic) is distributed across categories, while describe() only gives a general overview of the dataset.

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

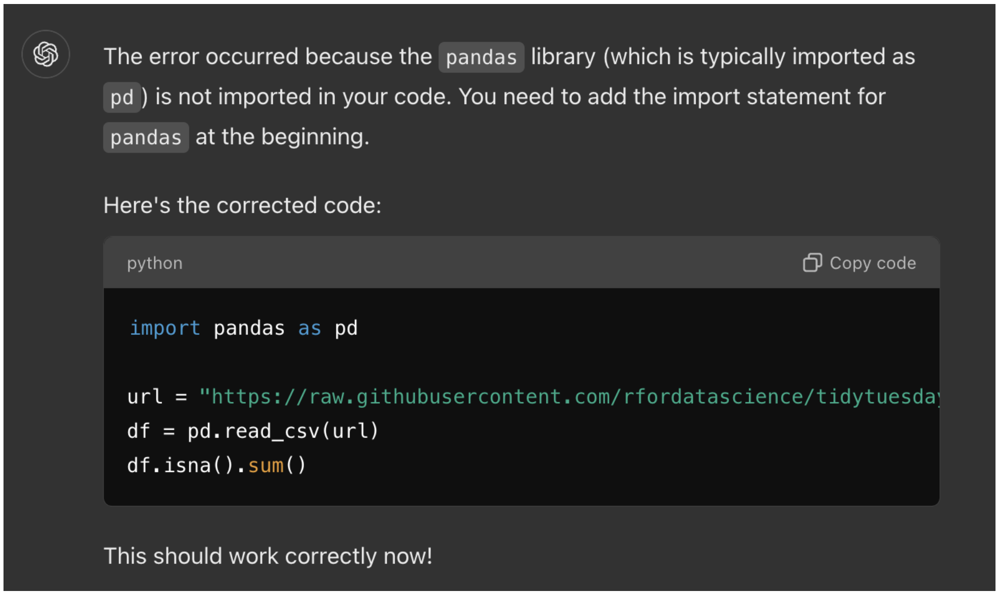
A. Forget to include `import pandas as pd` in your code



```
In [1]: url = "https://raw.githubusercontent.com/rfordatascience/tidytuesday/mas
        df = pd.read_csv(url)
        df.isna().sum()
```

```
-----------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
Cell In[1], line 2
      1 url = "https://raw.githubusercontent.com/rfordatascience/tidytu
esday/master/data/2020/2020-05-05/villagers.csv"
----> 2 df = pd.read_csv(url)
      3 df.isna().sum()

NameError: name 'pd' is not defined
```



The error occurred because the `pandas` library (which is typically imported as `pd`) is not imported in your code. You need to add the import statement for `pandas` at the beginning.

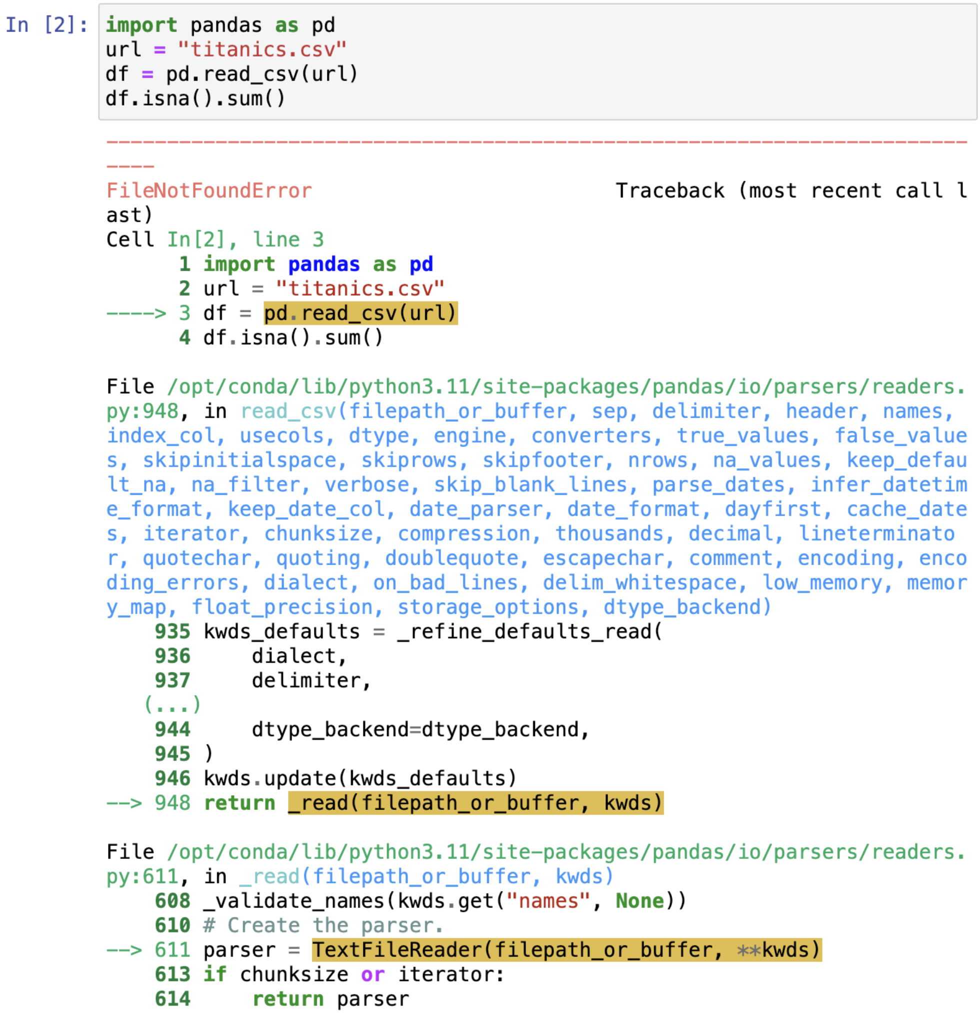
Here's the corrected code:

```python
import pandas as pd

url = "https://raw.githubusercontent.com/rfordatascience/tidytuesda
df = pd.read_csv(url)
df.isna().sum()
```

This should work correctly now!

B. Mistype “titanic.csv” as “titanics.csv”



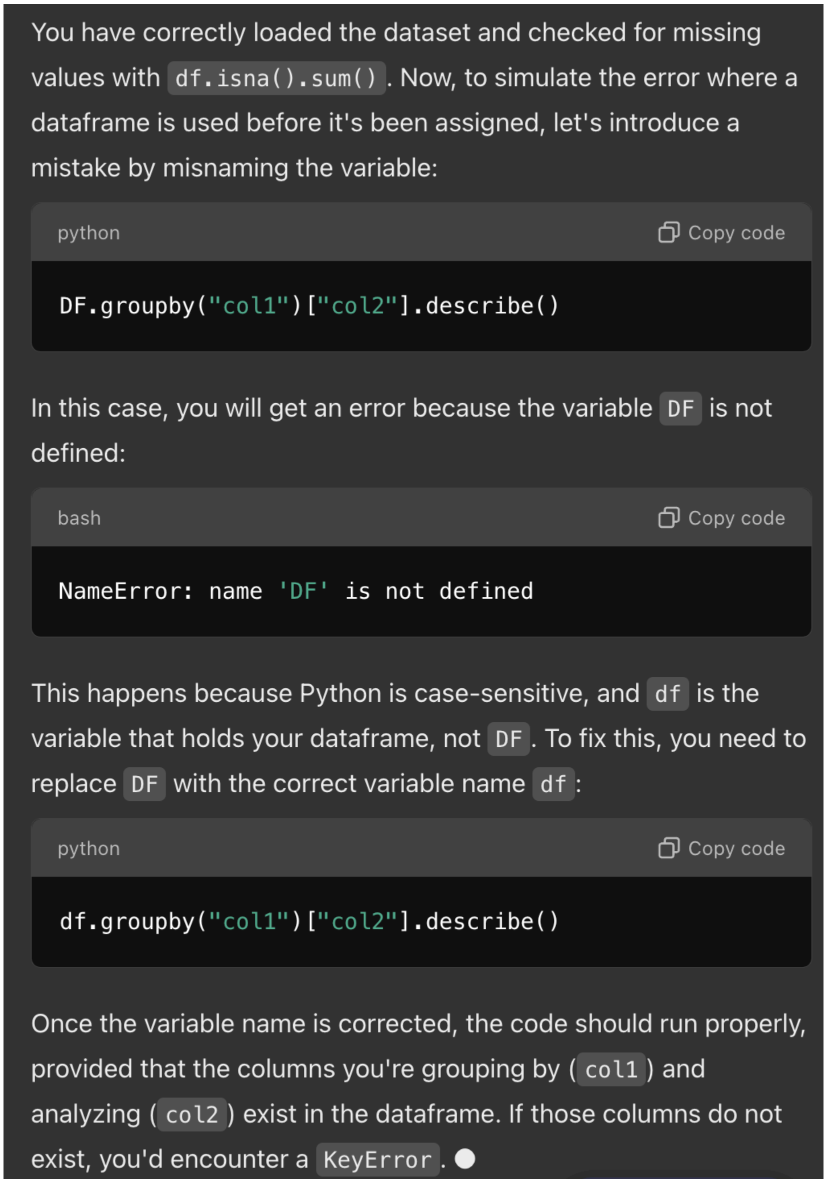
```
In [2]: import pandas as pd
        url = "titanics.csv"
        df = pd.read_csv(url)
        df.isna().sum()
```

```
---------------------------------------------------------------------------
----
FileNotFoundError                         Traceback (most recent call l
ast)
Cell In[2], line 3
      1 import pandas as pd
      2 url = "titanics.csv"
----> 3 df = pd.read_csv(url)
      4 df.isna().sum()

File /opt/conda/lib/python3.11/site-packages/pandas/io/parsers/readers.
py:948, in read_csv(filepath_or_buffer, sep, delimiter, header, names,
index_col, usecols, dtype, engine, converters, true_values, false_value
s, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_defau
lt_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetim
e_format, keep_date_col, date_parser, date_format, dayfirst, cache_date
s, iterator, chunksize, compression, thousands, decimal, lineterminato
r, quotechar, quoting, doublequote, escapechar, comment, encoding, enco
ding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memor
y_map, float_precision, storage_options, dtype_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File /opt/conda/lib/python3.11/site-packages/pandas/io/parsers/readers.
py:611, in _read(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser
```

C. Try to use a dataframe before it’s been assigned into the variable



You have correctly loaded the dataset and checked for missing values with `df.isna().sum()`. Now, to simulate the error where a dataframe is used before it's been assigned, let's introduce a mistake by misnaming the variable:

```python
DF.groupby("col1")["col2"].describe()
```

In this case, you will get an error because the variable `DF` is not defined:

```bash
NameError: name 'DF' is not defined
```

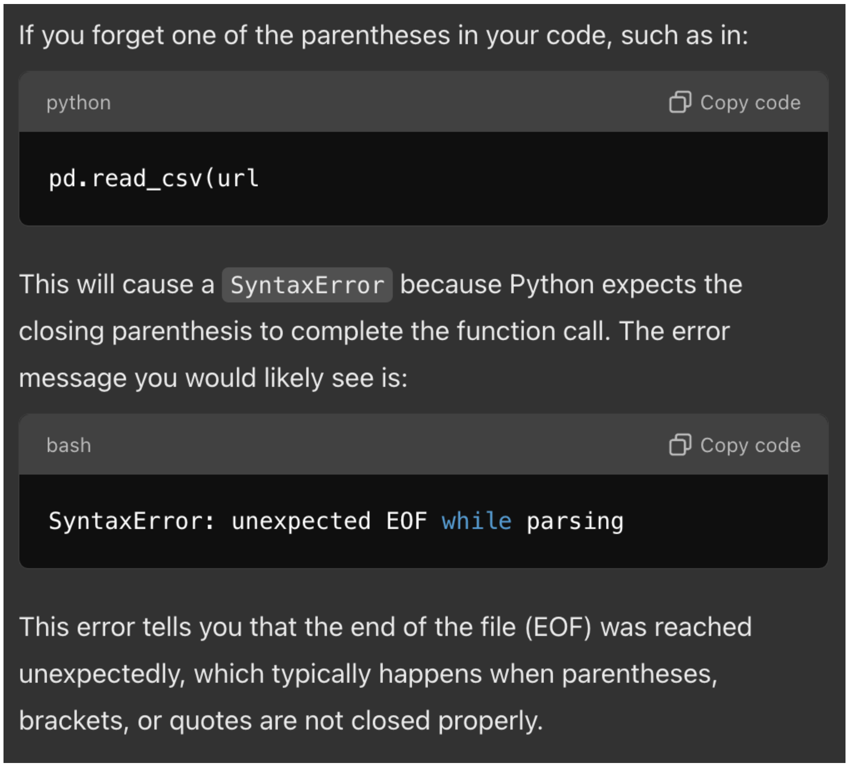
This happens because Python is case-sensitive, and `df` is the variable that holds your dataframe, not `DF`. To fix this, you need to replace `DF` with the correct variable name `df`:

```python
df.groupby("col1")["col2"].describe()
```

Once the variable name is corrected, the code should run properly, provided that the columns you're grouping by (`col1`) and analyzing (`col2`) exist in the dataframe. If those columns do not exist, you'd encounter a `KeyError`.

D. 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)’



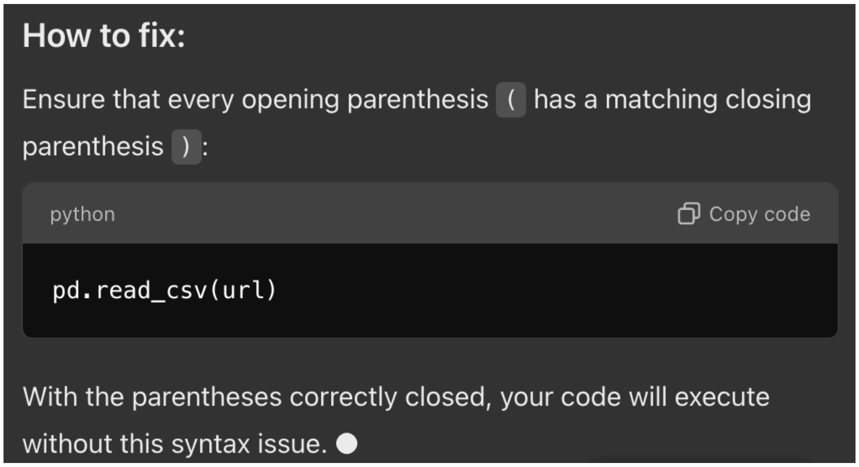
If you forget one of the parentheses in your code, such as in:

```python
pd.read_csv(url
```

This will cause a `SyntaxError` because Python expects the closing parenthesis to complete the function call. The error message you would likely see is:

```bash
SyntaxError: unexpected EOF while parsing
```

This error tells you that the end of the file (EOF) was reached unexpectedly, which typically happens when parentheses, brackets, or quotes are not closed properly.



How to fix:

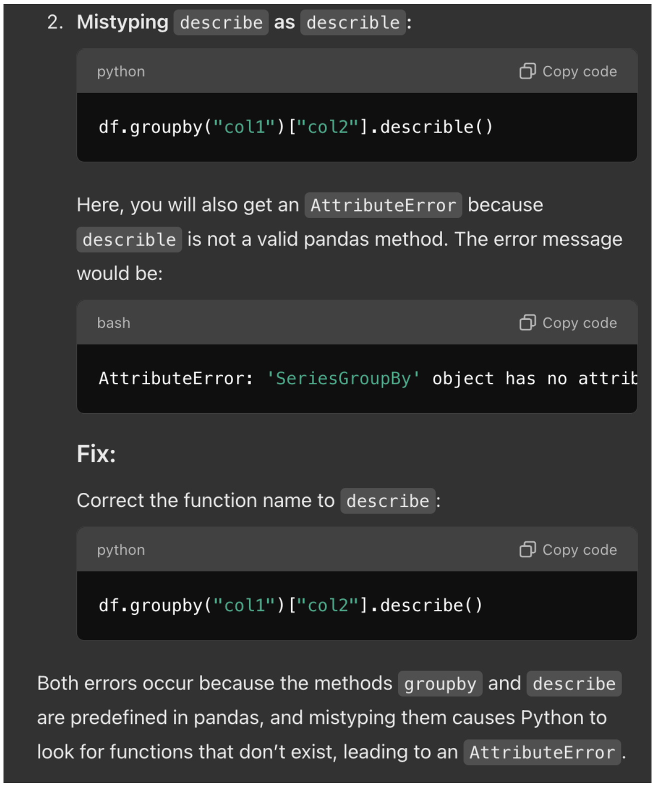
Ensure that every opening parenthesis `(` has a matching closing parenthesis `)`:

```python
pd.read_csv(url)
```

With the parentheses correctly closed, your code will execute without this syntax issue.

E. Mistype one of the names of the chained functions

For example, try something like `df.group_by("col1")["col2"].describe()` and `df.groupby("col1")["col2"].describe()`



2. **Mistyping `describe` as `describible`:**

```python
df.groupby("col1")["col2"].describible()
```

Here, you will also get an `AttributeError` because `describible` is not a valid pandas method. The error message would be:

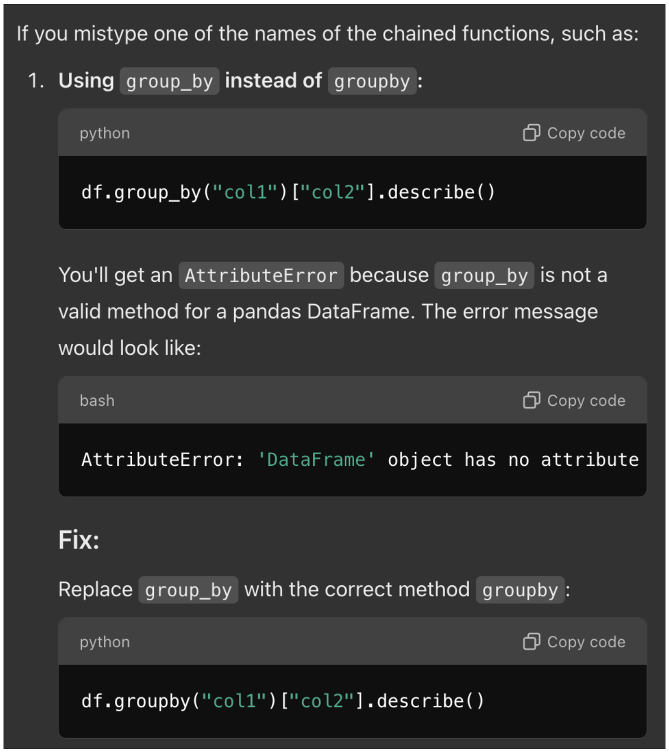
```bash
AttributeError: 'SeriesGroupBy' object has no attrib
```

Fix:

Correct the function name to `describe`:

```python
df.groupby("col1")["col2"].describe()
```

Both errors occur because the methods `groupby` and `describe` are predefined in pandas, and mistyping them causes Python to look for functions that don't exist, leading to an `AttributeError`.



If you mistype one of the names of the chained functions, such as:

1. **Using `group_by` instead of `groupby`:**

```python
df.group_by("col1")["col2"].describe()
```

You'll get an `AttributeError` because `group_by` is not a valid method for a pandas DataFrame. The error message would look like:

```bash
AttributeError: 'DataFrame' object has no attribute
```

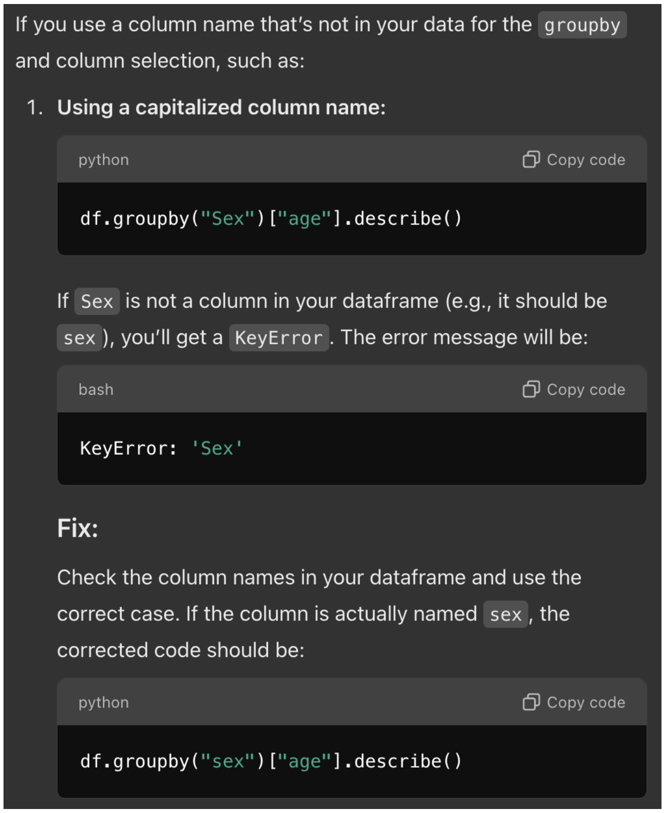
Fix:

Replace `group_by` with the correct method `groupby`:

```python
df.groupby("col1")["col2"].describe()
```

F. 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"



If you use a column name that's not in your data for the `groupby` and column selection, such as:

1. **Using a capitalized column name:**

```python
df.groupby("Sex")["age"].describe()
```

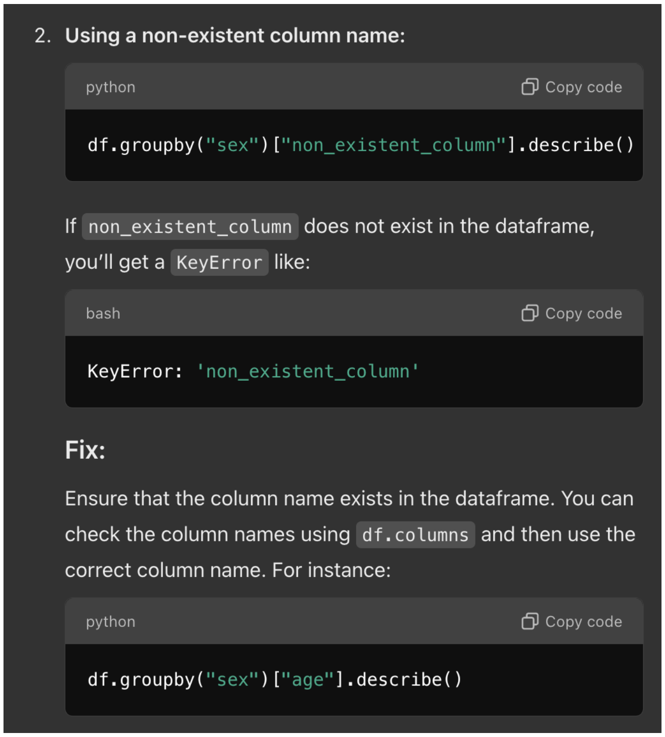
If `Sex` is not a column in your dataframe (e.g., it should be `sex`), you'll get a `KeyError`. The error message will be:

```bash
KeyError: 'Sex'
```

Fix:

Check the column names in your dataframe and use the correct case. If the column is actually named `sex`, the corrected code should be:

```python
df.groupby("sex")["age"].describe()
```



2. **Using a non-existent column name:**

```python
df.groupby("sex")["non_existent_column"].describe()
```

If `non_existent_column` does not exist in the dataframe, you'll get a `KeyError` like:

```bash
KeyError: 'non_existent_column'
```

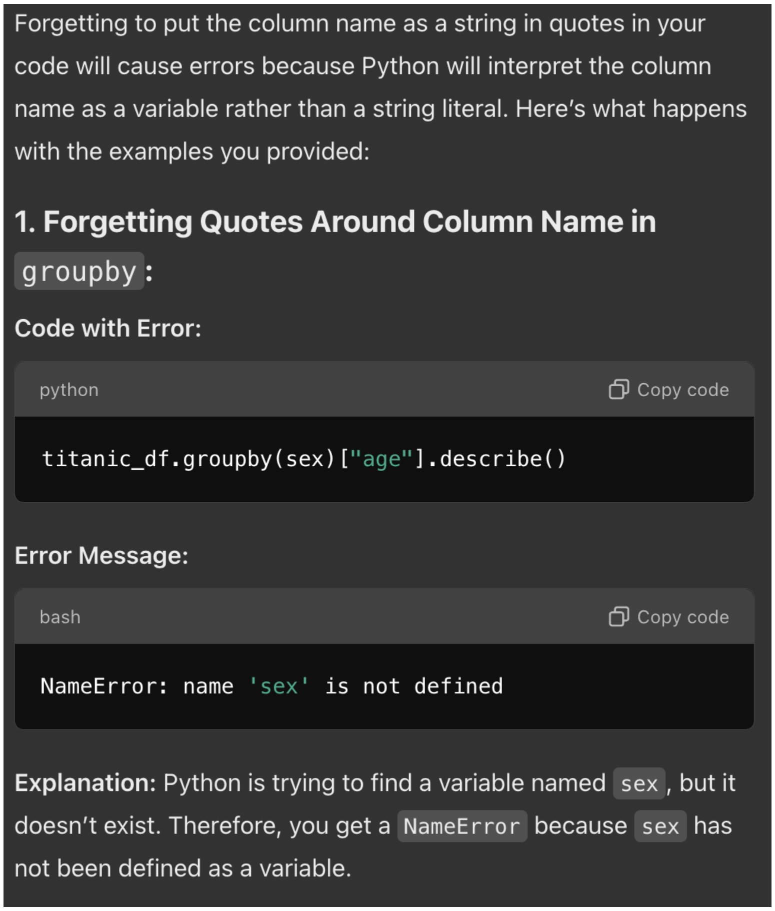
Fix:

Ensure that the column name exists in the dataframe. You can check the column names using `df.columns` and then use the correct column name. For instance:

```python
df.groupby("sex")["age"].describe()
```

G. 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()`



Forgetting to put the column name as a string in quotes in your code will cause errors because Python will interpret the column name as a variable rather than a string literal. Here's what happens with the examples you provided:

1. Forgetting Quotes Around Column Name in `groupby`:

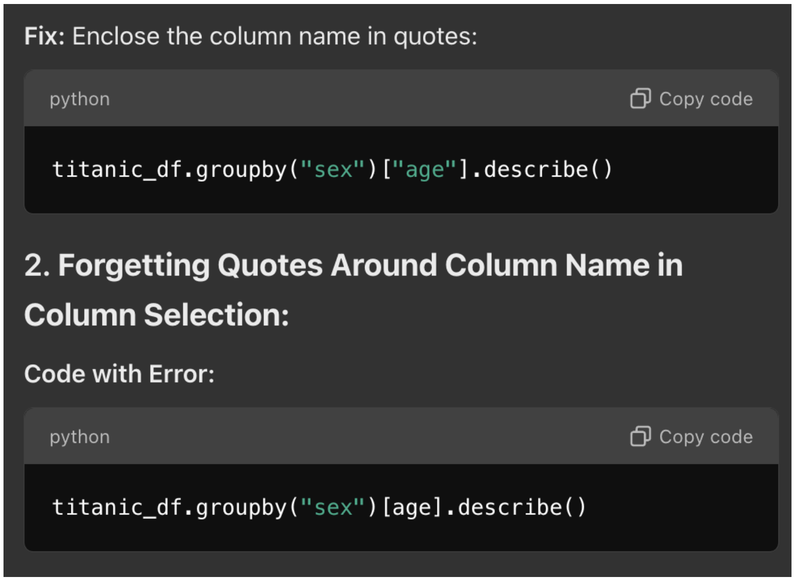
Code with Error:

```python
titanic_df.groupby(sex)["age"].describe()
```

Error Message:

```bash
NameError: name 'sex' is not defined
```

Explanation: Python is trying to find a variable named `sex`, but it doesn't exist. Therefore, you get a `NameError` because `sex` has not been defined as a variable.



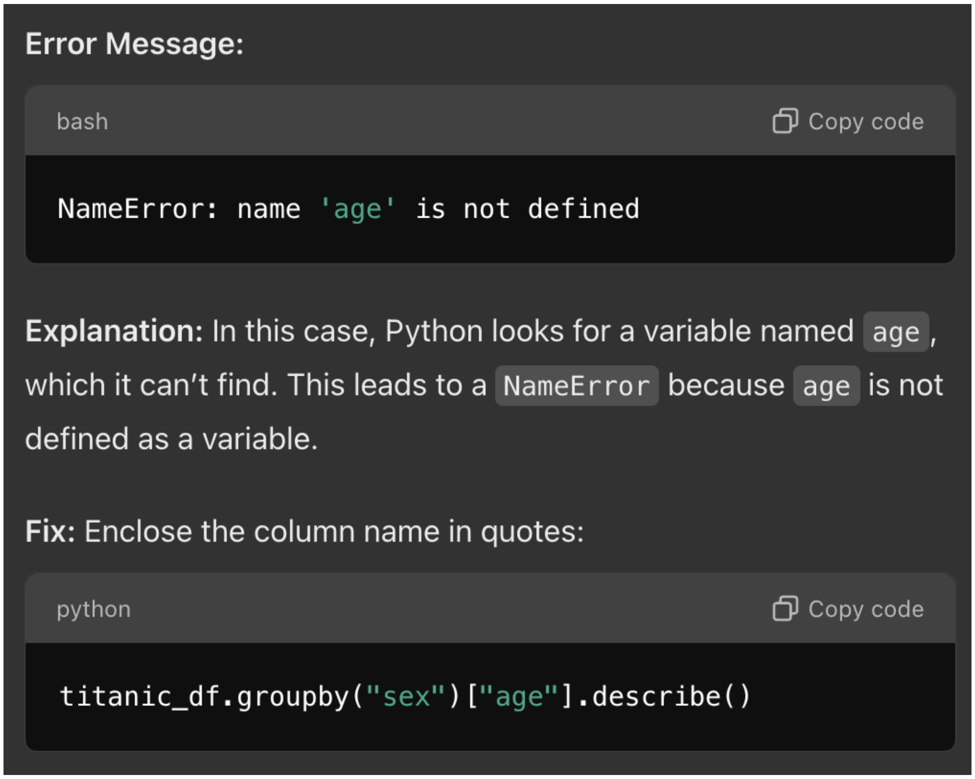
Fix: Enclose the column name in quotes:

```python
titanic_df.groupby("sex")["age"].describe()
```

2. Forgetting Quotes Around Column Name in Column Selection:

Code with Error:

```python
titanic_df.groupby("sex")[age].describe()
```



Error Message:

```bash
NameError: name 'age' is not defined
```

Explanation: In this case, Python looks for a variable named `age`, which it can't find. This leads to a `NameError` because `age` is not defined as a variable.

Fix: Enclose the column name in quotes:

```python
titanic_df.groupby("sex")["age"].describe()
```

Chatbot Troubleshooting Evaluation Compared to Google Search:

A chatbot that I used is ChatGPT and it help by suggesting appropriate code and ways to solve the given problem with quick guidance and immediate correction. However, it may contain unreliable contents since it sometimes provide data whose source is unclear

Google provided solutions from documents, blog or forums where similar topic and issues are discussed which is more reliable than chatbot.