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

   Using df.groupby("col1")["col2"].describe(), we can easily generate a breakdown of summary statistics for a specific column within groups defined by another column. This is particularly useful when analyzing data that can be categorized into groups

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.