HW1 submission

September 23, 2024

[1]:
```
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
```

[4]:
```
# Let's create a synthetic dataset with missing values to simulate the pandas␣
 ↪missing values dataset

import numpy as np

# Creating a dictionary with some missing values
data = {
    'Name': ['Alice', 'Bob', np.nan, 'David', 'Eva'],
    'Age': [25, 30, np.nan, 22, 29],
    'Score': [88, 92, 85, np.nan, 95],
    'Country': ['USA', 'Canada', np.nan, 'USA', 'Germany']
}

# Create DataFrame
df = pd.DataFrame(data)

# Get the shape of the DataFrame (rows, columns)
df_shape = df.shape
```

```
df_shape
```

[4]: (5, 4)

[ ]:
```
My own defition on "observations" and "varaibles": "observations", which is␣
 ↪rows according
to the dateset in statistics, is a collection of measurements obtained by an␣
 ↪individual.
"variabels", which is columns according to the dateset in statistics, is a␣
 ↪feature of
something or someone that differs from other individuals that is measured for␣
 ↪each
observations.
```

[ ]:
```
df.shape: This gives the total number of rows and columns in the DataFrame. It␣
 ↪reports the full structure of the dataset, regardless of the type of data in␣
 ↪each column.
df.describe(): By default, df.describe() only provides summary statistics for␣
 ↪numerical columns (e.g., integers and floats). This means it excludes␣
 ↪non-numerical columns like strings (object type) unless specified otherwise␣
 ↪(e.g., df.describe(include='all')).

df.shape[0]: This represents the total number of rows (observations) in the␣
 ↪dataset, whether or not there are missing values in those rows.
df.describe() "count": The "count" column in df.describe() reports the number␣
 ↪of non-missing (non-null) values for each variable. It excludes rows with␣
 ↪missing values in the corresponding column.


Number of columns analyzed: df.describe() by default only includes numerical␣
 ↪columns, while df.shape counts all columns.
"Count" values in df.describe(): These represent the number of non-missing␣
 ↪values in each numerical column, whereas df.shape counts the total number of␣
 ↪rows, even if some contain missing data.
```

[ ]:
```
Attributes:
Represent static properties or values.
Accessed without parentheses (df.shape).
Do not change the DataFrame or perform any computation.
Methods:
Perform actions, calculations, or transformations.
Accessed with parentheses (df.describe()).
May take arguments and often return a modified DataFrame or computed result.
```

[ ]:
```
https://chatgpt.com/share/66f1ee10-3744-8000-9eae-8fc66383b7be
```

[ ]:
```
Count:
The number of non-null (non-missing) entries for each variable/column.
Mean:
The arithmetic average of the data in the column. It's calculated as the sum of␣
 ↪all values divided by the count (number of values).
Std (Standard Deviation):
A measure of the dispersion or spread of data from the mean. It indicates how␣
 ↪much the data points typically deviate from the mean.
Min (Minimum):
The smallest value in the data for each variable/column.
25% (1st Quartile):
The 25th percentile, also known as the first quartile (Q1). It represents the␣
 ↪value below which 25% of the data fall. It gives insight into the lower␣
 ↪range of the data.
50% (Median or 2nd Quartile):
The median or 50th percentile, representing the middle value in the dataset␣
 ↪when sorted. It divides the data into two equal halves and is less affected␣
 ↪by outliers compared to the mean.
75% (3rd Quartile):
The 75th percentile, also known as the third quartile (Q3). It represents the␣
 ↪value below which 75% of the data fall, providing insight into the upper␣
 ↪range of the data.
Max (Maximum):
The largest value in the dataset for each variable/column.
```

[ ]:
```
When missing data is scattered across rows, dropping rows might be a better␣
 ↪choice.
When missing data is concentrated within certain columns, dropping columns may␣
 ↪be a more efficient option.
In many cases, a combination of both methods may provide the most efficient use␣
 ↪of non-missing data.
First, use del df['col'] to remove columns that have a large proportion of␣
 ↪missing data, reducing noise and potential bias.
Then, use df.dropna() to remove any remaining rows with missing data, as the␣
 ↪overall missingness would have reduced after removing the problematic␣
 ↪columns.
```

[ ]:
```
Scenario: You have a dataset containing customer feedback with multiple␣
 ↪features, including ratings, comments, and demographics.
Use Case:
Suppose the dataset has 1,000 entries, but only 10 rows have missing values in␣
 ↪the Rating column.
Since the Rating is crucial for your analysis (e.g., customer satisfaction),␣
 ↪and only a small number of entries are missing, you would use df.dropna() to␣
 ↪remove just those 10 rows.
```

```
This preserves the integrity of the dataset, allowing you to retain a majority␣
 ↪of the data for analysis without losing important columns.

Scenario: You have a dataset of student grades with columns for Student ID,␣
 ↪Name, Assignment Scores, and Extracurricular Activities.
Use Case:
Imagine that the Extracurricular Activities column has 80% missing values while␣
 ↪the other columns are complete.
In this case, using del df['Extracurricular Activities'] would be preferred.
Since this column contributes little information due to the high percentage of␣
 ↪missing data, deleting it allows you to keep all the rows intact and focus␣
 ↪on the more reliable columns for your analysis.

When using both methods, applying del df['col'] before df.dropna() can help in␣
 ↪the following ways:
Reduce Missingness: Removing columns with high missing values can decrease the␣
 ↪overall missingness across the dataset, meaning that fewer rows will be␣
 ↪dropped when df.dropna() is applied afterward.
Maintain Row Integrity: If you have several columns with scattered missing␣
 ↪values, dropping the ones with excessive missing data first may allow you to␣
 ↪keep more rows that contain valuable information.
Streamlined Analysis: It simplifies the dataset for analysis by eliminating␣
 ↪unreliable data early on, making the subsequent row removal process more␣
 ↪effective.

Justification for Approach
By removing the Extracurricular Activities column first, which had a high␣
 ↪percentage of missing values, I ensured that I retained the maximum number␣
 ↪of rows with valid data in the Score column. After removing this column, I␣
 ↪used df.dropna() to clean up any remaining missing values, allowing me to␣
 ↪maintain a clean and effective dataset for further analysis. This strategy␣
 ↪minimizes data loss while focusing on retaining the most relevant␣
 ↪information.

The remove action using some combination of del df['col'] and/or df.dropna() is␣
 ↪as follows in the next In.
```

[ ]:
```
import pandas as pd
import numpy as np

# Sample DataFrame
data = {
    'ID': [1, 2, 3, 4, 5, 6],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],
    'Score': [85, np.nan, 90, 78, np.nan, 92],
```

```
    'Extracurricular Activities': ['Football', 'Basketball', np.nan, 'Soccer',␣
 ↪np.nan, 'Chess']
}

df = pd.DataFrame(data)

# Step 1: Remove the column with high missingness
del df['Extracurricular Activities']

# Step 2: Drop rows with missing values
df = df.dropna()

# Final DataFrame
print(df)
```

[ ]:
```
df.groupby("col1"): This part of the command groups the DataFrame by the unique␣
 ↪values in col1. Each unique value in col1 forms a separate group.
["col2"]: This specifies that we are interested in the col2 column for our␣
 ↪analysis.
.describe(): This function generates descriptive statistics for col2 for each␣
 ↪group created by col1. The output will typically include:
count: Number of non-null entries in col2.
mean: Average of col2 for the group.
std: Standard deviation of col2.
min: Minimum value of col2.
25%: 25th percentile (1st quartile) of col2.
50%: Median (2nd quartile) of col2.
75%: 75th percentile (3rd quartile) of col2.
max: Maximum value of col2.
```

[ ]:
```
import pandas as pd

# Create a sample DataFrame
data = {
    'Category': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C', 'C'],
    'Values': [10, 20, 30, 15, 25, 35, 5, 15, 25, None]
}

df = pd.DataFrame(data)
```

[ ]:
```
# Group by 'Category' and describe 'Values'
result = df.groupby("Category")["Values"].describe()
print(result)
```

[ ]:
```
Count: The number of valid (non-null) entries for Values in each category:
Category A: 3
Category B: 3
```

```
Category C: 3
Mean: The average of Values for each category:
Category A: (10 + 20 + 30) / 3 = 20.0
Category B: (15 + 25 + 35) / 3 = 25.0
Category C: (5 + 15 + 25) / 3 = 15.0
Standard Deviation (std): Measures the spread of Values for each category:
All categories have a standard deviation of 10.0, indicating how much the␣
 ↪values vary around the mean.
Min, 25%, 50%, 75%, Max: These values provide insights into the distribution of␣
 ↪Values:
Min: The smallest value in each category.
25% (1st quartile): The value below which 25% of the data fall.
50% (Median): The middle value of Values.
75% (3rd quartile): The value below which 75% of the data fall.
Max: The largest value in each category.
```

[ ]:
```
While df.describe() provides an overall view of the dataset, including how many␣
 ↪entries are
missing across all columns, df.groupby("col1")["col2"].describe() gives a␣
 ↪detailed
breakdown that highlights the distribution of valid data across specific␣
 ↪categories.
This distinction is crucial for understanding the context and reliability of␣
 ↪the statistics,
particularly when considering how missing data impacts analysis and␣
 ↪interpretation.
```

[ ]:
```
# No import statement
df = pd.read_csv("titanic.csv")
```

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

FileNotFoundError: [Errno 2] No such file or directory: 'titanics.csv'

NameError: name 'df' is not defined

SyntaxError: unexpected EOF while parsing

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

KeyError: 'Sex'

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

[ ]:
```
question 9:Yes
```

[ ]: https://chatgpt.com/share/66f1fa7e-38d0-8000-bcc9-950d8fc187a4