Python implementation and result for the proposed Dataset



This code snippet demonstrates the process of loading a dataset from a CSV file, performing basic data cleaning operations, and saving the cleaned dataset back to a CSV file. Here's a summary of each step:

1. Load the dataset: The `pd.read\_csv('data.csv')` function reads the CSV file named `data.csv` and creates a pandas DataFrame called `df` to store the data.

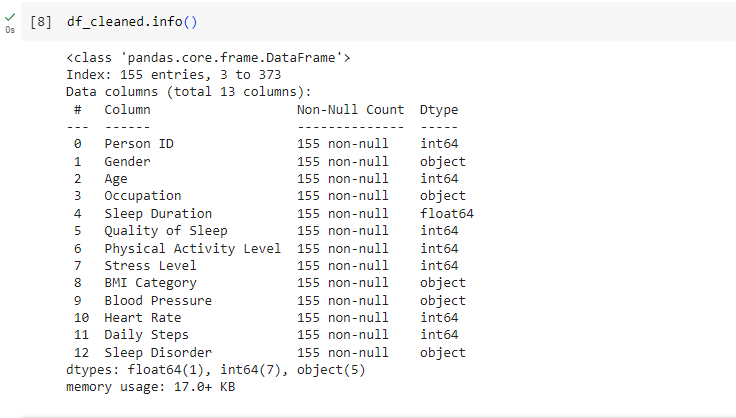
2. Check for missing values: The `df.isnull().sum()` method calculates the number of missing values in each column of the DataFrame `df` and stores the result in `missing\_values`.

3. Drop rows with missing values: The `df.dropna()` method removes rows from the DataFrame `df` that contain missing values (NaN values) and creates a new DataFrame called `df\_cleaned`.

4. Drop duplicate rows: The `df\_cleaned.drop\_duplicates()` method removes duplicate rows from the DataFrame `df\_cleaned` and updates `df\_cleaned` with the result.

5. Convert columns to appropriate data types: This step is optional and is used to convert columns to the appropriate data types if needed. For example, you can convert a column containing numerical values stored as strings to integers using `astype('int')`.

6. Save the cleaned dataset: Finally, the `df\_cleaned.to\_csv('cleaned\_data.csv', index=False)` method saves the cleaned DataFrame `df\_cleaned` to a new CSV file called `cleaned\_data.csv` without including the index column in the output file (`index=False`).



This code snippet uses the `info()` method in pandas to provide a concise summary of the DataFrame `df\_cleaned`. Here's a summary of what it shows:

- The total number of rows in the DataFrame (`RangeIndex`).

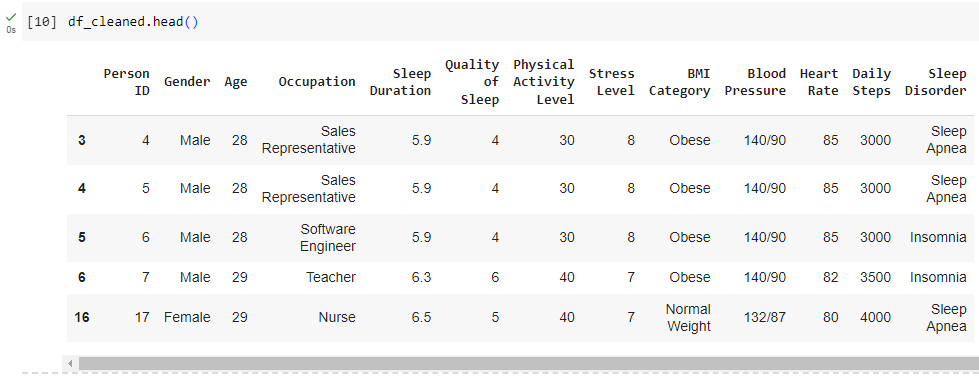
- The number of columns in the DataFrame.

- The name and data type of each column.

- The number of non-null values in each column.

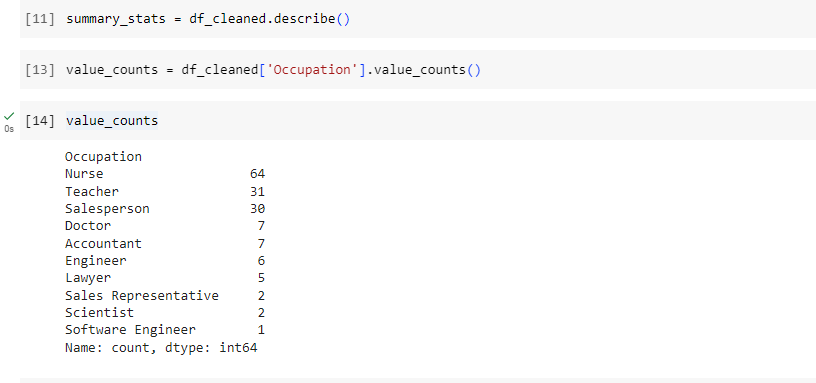
- The memory usage of the DataFrame.

This information is useful for understanding the structure of your DataFrame, including the number of columns, their names, and data types, as well as checking for missing values (non-null counts).



The `head()` method in pandas is used to display the first few rows of a DataFrame. By default, it shows the first five rows, but you can specify the number of rows to display by passing an integer argument to the method (e.g., `df\_cleaned.head(10)` to display the first ten rows).

This method is helpful for quickly inspecting the contents of a DataFrame, including the column names and the actual data values. It's often used at the beginning of the data analysis process to get a sense of what the dataset looks like and to check if the data has been loaded correctly.



This code snippet demonstrates how to perform basic data analysis using pandas in Python. Here's a summary of each step:

1. Summary statistics: The `df\_cleaned.describe()` method calculates summary statistics for numerical columns in the DataFrame `df\_cleaned`, such as count, mean, standard deviation, minimum, maximum, and quartile values. The result is stored in the `summary\_stats` variable.

2. Value counts for categorical variables: The `df\_cleaned['Occupation'].value\_counts()` method calculates the frequency of each unique value in the specified column (`Occupation`) of the DataFrame `df\_cleaned`. This is useful for understanding the distribution of categorical variables. The result is stored in the `value\_counts` variable.

3. Correlation matrix: The `df\_cleaned.corr()` method calculates the correlation coefficients between pairs of columns in the DataFrame `df\_cleaned`. This is useful for identifying relationships between numerical variables. The result is stored in the `correlation\_matrix` variable, which is a square matrix where each cell represents the correlation between two columns.

4. Hypothesis testing, regression, or other statistical analysis: This comment suggests that additional statistical analysis can be performed as needed for your specific analysis goals. This could include hypothesis testing to compare groups, regression analysis to model relationships between variables, or other types of statistical tests depending on your data and research questions.



This code snippet demonstrates how to create three different types of plots using matplotlib and seaborn libraries in Python:

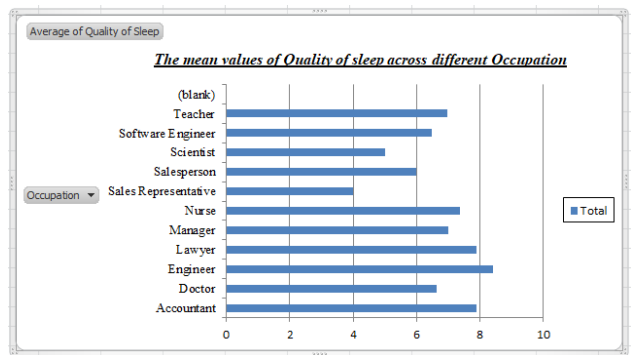
1. Histogram:

- `plt.hist(df\_cleaned['Occupation])` creates a histogram of the values in the specified column (`Occupation`).

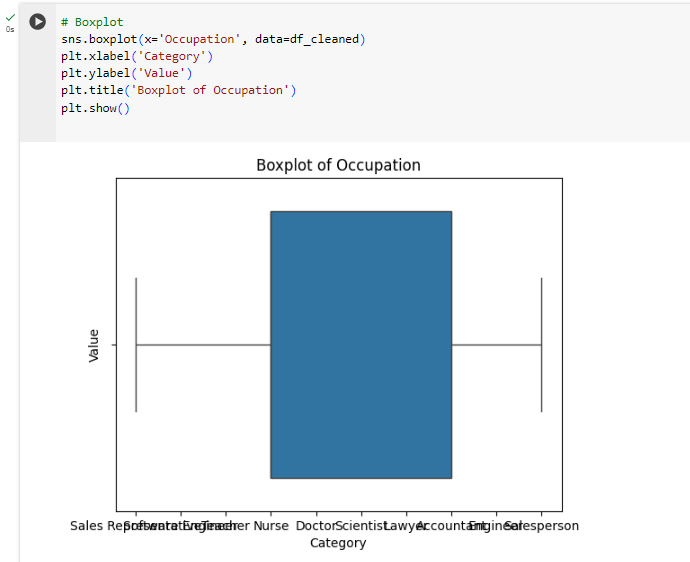
- `plt.xlabel('Value')` and `plt.ylabel('Frequency')` set the labels for the x-axis and y-axis, respectively.

- `plt.title('Histogram of Occupation')` sets the title of the plot.

- `plt.show()` displays the histogram plot.



2. Boxplot:



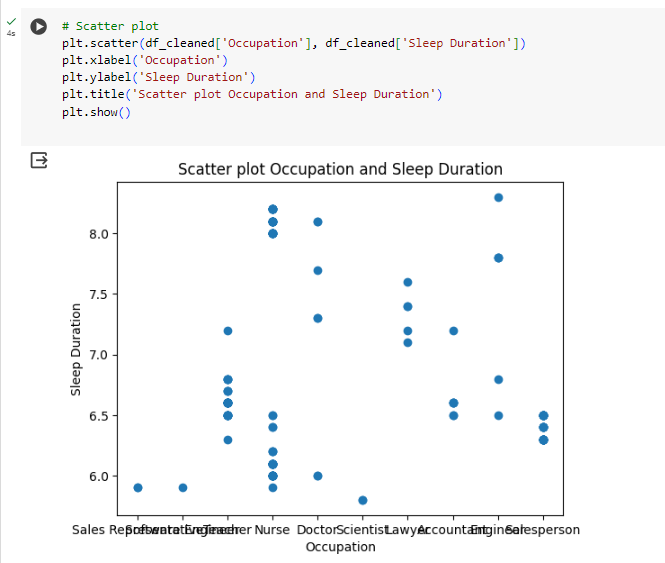
sns.boxplot(x=Occupation', data=df\_cleaned) creates a boxplot of the values in the specified column (Occupaton).

plt.xlabel('Category') and plt.ylabel('Value') set the labels for the x-axis and y-axis, respectively.

plt.title('Boxplot of Occupation) sets the title of the plot.

plt.show() displays the boxplot.

3. Scatter plot:



plt.scatter(df\_cleaned[Occupation], df\_cleaned[Sleep Duration]) creates a scatter plot between two columns (Occuaptaion and Sleep Duration).

plt.xlabel(Occupation) and plt.ylabel(Sleep Duration) set the labels for the x-axis and y-axis, respectively.

plt.title('Scatter plot between Occupation and Sleep Duration) sets the title of the plot.

plt.show() displays the scatter plot.

These plots are useful for visualizing the distribution of values in a single column (histogram), comparing the distribution of a numerical variable across different categories (boxplot), and exploring the relationship between two numerical variables (scatter plot).