Nerd For Tech · Follow publication

Data Cleaning Process for Beginners

7 min read · May 1, 2023


Mahesh Tiwari Follow

Data Cleaning

Data cleaning is the process of identifying and correcting or removing errors, inconsistencies, and inaccuracies in data sets. It involves checking data for completeness, removing duplicate entries, dealing with missing data, standardizing data formats, and correcting data values that are out of range or invalid.

Importance of Data Cleaning

1. Accurate data is essential for making informed decisions: If the data is incorrect, any insights or conclusions drawn from it may be flawed.
2. Data cleaning can help identify and prevent errors early on: This can save time and resources by avoiding costly mistakes downstream.
3. Data cleaning can improve the quality of data: By removing errors and inconsistencies, data becomes more reliable and trustworthy.
4. Data cleaning can improve the efficiency of data analysis: With clean data, analysts can spend more time analysing data and less time correcting errors.

Step by Step Data Cleaning Process with an Example

Here I have used Jupyter Notebook inside Visual Studio Code to run the Python code. You can find my code in the GitHub repository, https://github.com/mahesh989/Basic-Data-Cleaning.

Step 1: Reading Data

The first step in the data-cleaning process is to read the data. In this example, we will read the data from the URL using pandas. The source of the data is https://github.com/justmarkham/DAT8/blob/master/data/chipotle.tsv. As usual, we import the required libraries.

```
import numpy as np
import pandas as pd

url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.
df = pd.read_csv(url, sep='\t')
```

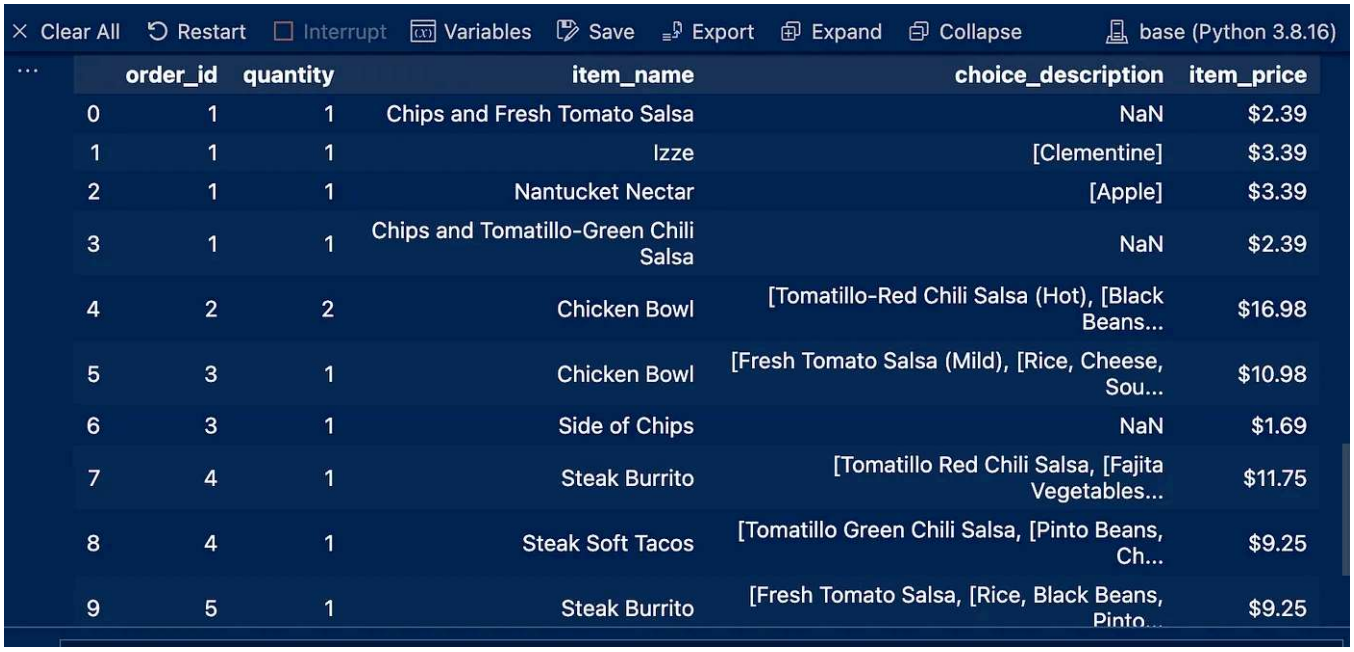
Step 2:

2.1 Observing Data

Once we have the data, we should observe it to understand the structure of the data, data types, and data distribution. This helps us to identify any anomalies or inconsistencies in the data.

```
df.head(10)
df.tail(10)
```

This will print the first and last 10 entries of the dataset, which give the idea of what kind of dataset we are working for. You can choose either of the first or last entries with any number depending on what you are looking for. Then the output using df.head(10) is;



	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

Clearly, we see some NaN entries in the choice_description column and the dollar sign in item_price column.

2.2 Data types of columns


```
for column, dtype in zip(df.columns, df.dtypes):
    print(f"{column}:{dtype}")
```

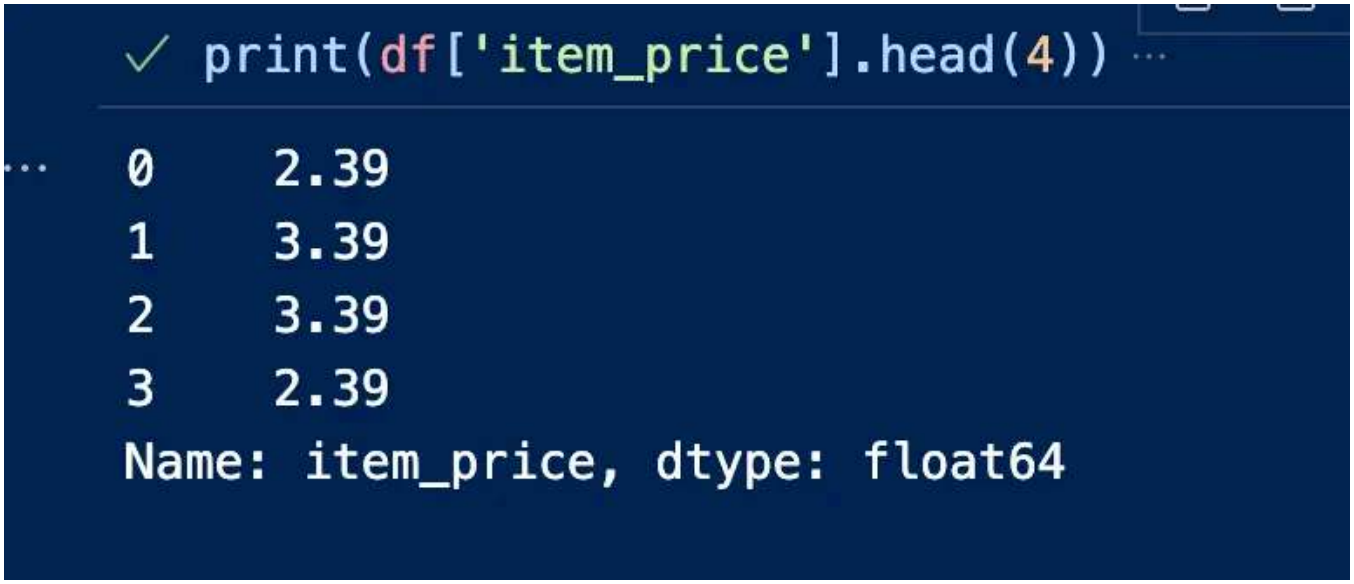
The output is:

Step 3: Data Cleaning

3.1 Changing Datatype

By observing the data, we can convert the data types, if necessary. For example, item_price contains a dollar sign, so we can remove it and replace the data type from an object with float64 as it contains decimal.

```
df['item_price'] = df['item_price'].str.replace('$', '')
df['item_price'] = df['item_price'].astype('float64')
print(df['item_price'].head(4))
```



3.1 Missing Values/ Null Values

First, let us find the null values in the dataset using:

```
dataset_null = df.isnull()
print(dataset_null)
```

The output is:

```
      order_id  quantity  item_name  choice_description  item_price
0        False     False      False                True       False
1        False     False      False               False       False
2        False     False      False               False       False
3        False     False      False                True       False
4        False     False      False               False       False
...        ...       ...        ...                 ...         ...
4617     False     False      False               False       False
4618     False     False      False               False       False
4619     False     False      False               False       False
4620     False     False      False               False       False
4621     False     False      False               False       False

[4622 rows x 5 columns]
```

In the output, True represents the null values and False represents not null values. For a huge dataset, we may not be able to see all or little true values present in the table as in the above figure. So, it's better to find the number of null entries in the table using sum.

```
print(df.isnull().sum())
```

```
order_id               0
quantity               0
item_name              0
choice_description    1246
item_price             0
dtype: int64
```

This gives the idea about columns that contain null values and how many of them are null. From this we can see that column "choice_description" is the only column that has null entries and 1246 of them are null.

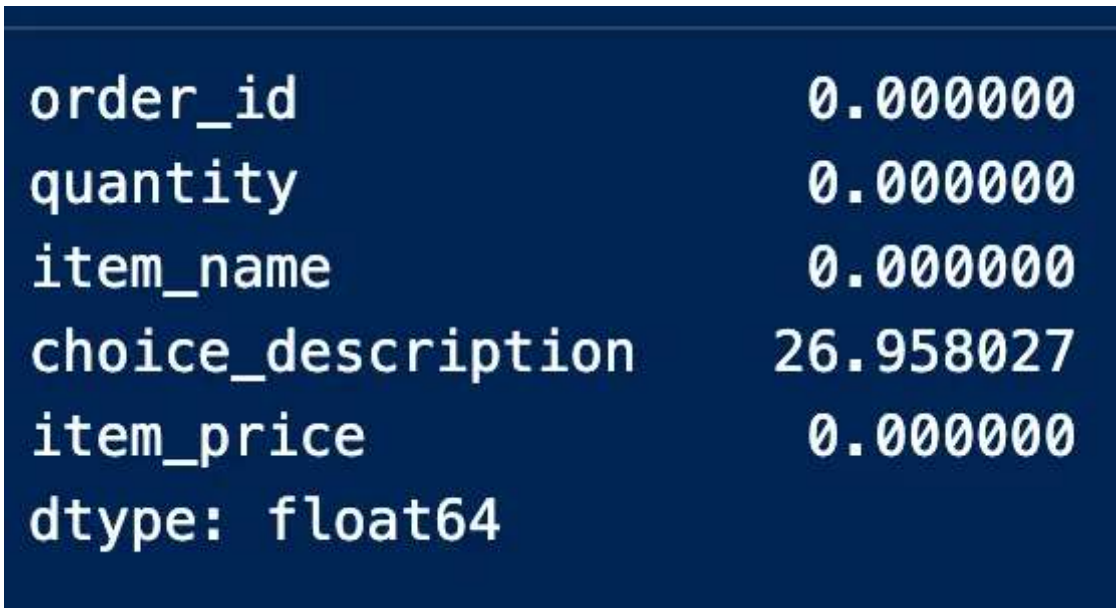
Tips: we can check the null values for each column and find the number as well using:

```
print(df['choice_description'].isnull().sum())
```

Next, check the percentage of missing values in each column. Here, only one column has null values. In reality, for huge data, there can be many columns which can have null values. So, it is essential to calculate the percentage of missing values in each column. Normally, if it exceeds 70% then we can get rid of that column. But still, it depends upon the data set and the information it holds.

```
percent_missing_dataset = df.isnull().mean()*100
print(percent_missing_dataset)
```

The output is:



The description column has 27% of missing values, so it's not necessary to eradicate the whole column. Next, we can find a way to replace the null values.

> *We can use different approaches to handle missing data values while data cleaning, depending on the type of data and the problem at hand. If we have access to a domain expert, always incorporate their expert advice when filling in the missing values. Most*

> *importantly, no matter the imputation method we choose, always run the predictive analytics model to see which one works best from the standpoint of data accuracy.*

Let's understand the null values in the choice_description column, we first check the unique entries in that column to understand what it's about. Let's check the unique item for this description to have more ideas.

```
distinct_entries = df.loc[df['choice_description'].isnull(), 'item_name'].uniqu
print(distinct_entries)
```

```
['Chips and Fresh Tomato Salsa' 'Chips and Tomatillo-Green Chili Salsa'
 'Side of Chips' 'Chips and Guacamole' 'Bottled Water'
 'Chips and Tomatillo Green Chili Salsa' 'Chips'
 'Chips and Tomatillo Red Chili Salsa'
 'Chips and Roasted Chili-Corn Salsa' 'Chips and Roasted Chili Corn Salsa'
 'Chips and Tomatillo-Red Chili Salsa' 'Chips and Mild Fresh Tomato Salsa']
```

Now we check how many unique item_name have null choice_description

```
count_distinct_entries= df[df['choice_description'].isnull()]['item_name'].nuni
print("Number of unique item_name with null description:", count_distinct_entri
```

```
Number of unique item_name with null description: 12
```

These are the categorical missing values. It can be easily replaceable after consulting with the respective department. Since these missing values are for the choice of the customer: For the moment, let's assume that those customers didn't mention their choices. So we can replace these missing values with 'Regular' or "no preferred choice". For the sake of continuity, we choose Regular. Now, let's replace the null values with 'Regular Order'. Use boolean indexing to select the rows where choice_description is "Regular Order".

```
df['choice_description'] = df['choice_description'].fillna('Regular Order')

regular_orders = df[df['choice_description'] == 'Regular Order'].to_string(inde
# Display the selected rows
print(regular_orders)
```

The output:

order_id	quantity	item_name	choice_description	item_price
1	1	Chips and Fresh Tomato Salsa	Regular Order	2.39
1	1	Chips and Tomatillo-Green Chili Salsa	Regular Order	2.39
3	1	Side of Chips	Regular Order	1.69
5	1	Chips and Guacamole	Regular Order	4.45
7	1	Chips and Guacamole	Regular Order	4.45
8	1	Chips and Tomatillo-Green Chili Salsa	Regular Order	2.39
10	1	Chips and Guacamole	Regular Order	4.45
13	1	Chips and Fresh Tomato Salsa	Regular Order	2.39
15	1	Chips and Tomatillo-Green Chili Salsa	Regular Order	2.39
16	1	Side of Chips	Regular Order	1.69

Let's check if we have the null values or not.

```
print(df.isnull().sum())
```

```
order_id              0
quantity              0
item_name             0
choice_description    0
item_price            0
dtype: int64
```

It seems we have improved our data by replacing the null values with respective descriptions, and we have no missing values in any columns.

3.2 Removing Redundancy

Next, we are going to delete duplicate values. First, we will check how many duplicate entries we have.

> Duplicate entries mean that all the rows are completely the same as the other row. We can't delete if at least one of the entries is different.

```
count_duplicates = df[df.duplicated()].shape[0]
print("Number of duplicate rows:", count_duplicates)
```

```
Number of duplicate rows: 59
```

```
duplicates = df[df.duplicated(keep=False)]
duplicates_sorted = duplicates.sort_values(by=['order_id'])
print(duplicates_sorted.to_string(index=False))
```

We can verify by running the above code. The output is:

```
··· Output exceeds the size limit. Open the full output data in a text editor
 order_id  quantity                          item_name
      103         1                     Steak Burrito  [Tomatillo Red Chili Salsa, [Rice, Black Beans, Cheese, Sou
      103         1                     Steak Burrito  [Tomatillo Red Chili Salsa, [Rice, Black Beans, Cheese, Sou
      108         1                       Canned Soda
      108         1                       Canned Soda
      129         1                     Steak Burrito                                [Tomatillo Green Chili Sals
      129         1                     Steak Burrito                                [Tomatillo Green Chili Sals
      165         1                Canned Soft Drink
      165         1                Canned Soft Drink
      205         1                      Chicken Bowl                 [Fresh Tomato Salsa, [Fajita Vegetables
      205         1                      Chicken Bowl                 [Fresh Tomato Salsa, [Fajita Vegetables
      233         1                Canned Soft Drink
      233         1                Canned Soft Drink
      254         1                            Chips
      254         1                            Chips
```

Now, let's delete the duplicate entries.

```
#delete the duplicate entries
df.drop_duplicates(inplace=True)
```

Let's check again if we have the duplicate entries or not.

```
count_duplicates = df[df.duplicated()].shape[0]
print("Number of duplicate rows:", count_duplicates)

duplicates = df[df.duplicated(keep=False)]
duplicates_sorted = duplicates.sort_values(by=['order_id'])
print(duplicates_sorted.to_string(index=False))
```

```
Number of duplicate rows: 0
Empty DataFrame
Columns: [order_id, quantity, item_name, choice_description, item_price]
Index: []
```

3.3 Removing Extra spaces

> *Removing extra spaces involves removing any additional spaces that are not needed between words or characters. This can be done by using various techniques such as regular expressions, string manipulation functions, or dedicated data cleaning tools.*

```
# Iterate through each column in the dataframe
for col in df.columns:
    # Check if the column is a string column
    if df[col].dtype == 'object':
        # Remove extra spaces from each string in the column
        df[col] = df[col].str.strip()
```

Step 4: Exporting data

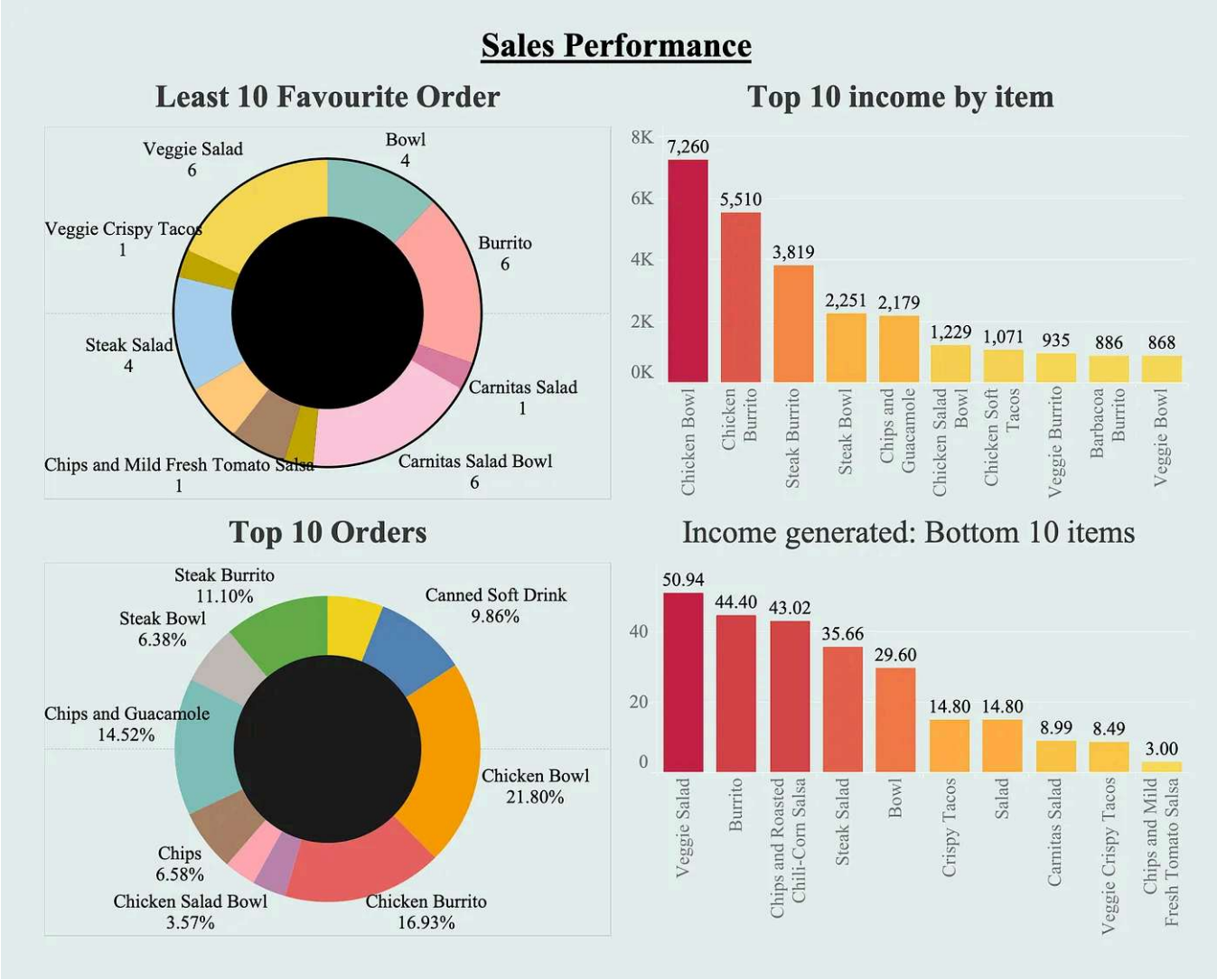
Since the data frame we have used here is not so much complicated, we will stop here for the data cleaning process. The next step is to export the clean data.

```
df.to_csv('cleaned_data.csv', index=False)
```

This code will write the cleaned data to a new CSV file called cleaned_data.csv in the same directory as our Python script. The index=False argument tells Pandas not to include the row index numbers in the exported data. You can modify the file name and path as needed.

Step 5: Data Visualization using Tableau

We can use cleaned data that we obtained by exporting for the visualization. The cleaned data makes it easier to analyse the data. Here is one such visualization. For more data visualization, please visit my tableau profile https://public.tableau.com/app/profile/maheshwor.tiwari4503.



Conclusion:

Data cleaning is an essential step in any data analysis project. It helps to ensure the accuracy and reliability of the results. In this article, we discussed the basic steps involved in the data-cleaning process for beginners. We hope this article helps you to get started with data cleaning.

> ***FOLLOW ME** to be part of my Data Analyst Journey on Medium.*
>
> *Let's get connected on* **Twitter** *or you can* **Email** *me for project collaboration, knowledge sharing or guidance.*