**The Role of a Data Analyst**

A data analyst uses programming tools to mine large amounts of complex data, and find relevant information from this data.

In short, an analyst is someone who derives meaning from messy data. A data analyst needs to have skills in the following areas, in order to be useful in the workplace:

* **Domain Expertise** — In order to mine data and come up with insights that are relevant to their workplace, an analyst needs to have domain expertise.
* **Programming Skills**—As a data analyst, you will need to know the right libraries to use in order to clean data, mine, and gain insights from it.
* **Statistics** — An analyst might need to use some statistical tools to derive meaning from data.
* **Visualization Skills** — A data analyst needs to have great data visualization skills, in order to summarize and present data to a third party.
* **Storytelling —**Finally, an analyst needs to communicate their findings to a stakeholder or client. This means that they will need to create a data story, and have the ability to narrate it.

In this article, I am going to walk you through the end-to-end data analysis process with Python.

**If you follow along to this tutorial and code everything out the way I did, you can then use these codes and tools for future data analytic projects.**

We will start with downloading and cleaning the dataset, and then move on to the analysis and visualization. Finally, we will tell a story around our data findings.

I will be using a dataset from Kaggle called [Pima Indian Diabetes Database](https://www.kaggle.com/uciml/pima-indians-diabetes-database), which you can download to perform the analysis.

**Pre-Requisites**

For this entire analysis, I will be using a Jupyter Notebook. You can use any Python IDE you like.

You will need to install libraries along the way, and I will provide links that will walk you through the…

**How to Clean Your Data in Python**

A detailed guide on how to clean your data to kickstart your personal projects

When I participated in my college’s directed reading program (a mini-research program where undergrad students get mentored by grad students), *I had only taken 2 statistics in R courses*. While these classes taught me a lot about how to manipulate data, create data visualizations, and extract analyses, working on my first personal project in the program made me **realize I had never worked with “messy data”**. Those courses involved pre-cleaned and processed datasets but **didn’t teach students how to clean datasets** which**creates a barrier to starting on personal projects**. Hence, I hope that this article serves as a starting point for you to **learn how to clean your data efficiently** to **kickstart your personal projects**.

For this article, I’ll be working with the [**Netflix TV Shows and Movies Dataset**](https://www.kaggle.com/datasets/shivamb/netflix-shows) which features many inconsistencies and missing data.

**Table of Contents**

1. Look into your data
2. Look at the proportion of **missing data**
3. Check the **data type** of each column
4. If you have columns of strings, check for **trailing whitespaces**
5. **Dealing with Missing Values** (NaN Values)
6. **Extracting more information from your dataset to get more variables**
7. Check the **unique values** of columns

*All images unless otherwise noted are by the author.*

**Step 1: Look into your data**

Before even performing any cleaning or manipulation of your dataset, you should take a glimpse at your data to understand **what variables you’re working with, how the values are structured based on the column they’re in, and maybe you could have a rough idea of the inconsistencies that you’ll need to address or they’ll be cumbersome in the analysis phase.** Here, you might also be able to eliminate certain columns that you won’t need depending on the analysis you want to do.

**1. Print the first few rows of your dataset**

Here, I printed the first 7 rows of my dataset, but you can print 5 or 10. I recommend keeping it to anything less than 10 or else it’ll be too overwhelming for what you’re currently trying to do–a quick glimpse of the dataset.

|  |
| --- |
| # importing dataset |
|  | netflix\_titles = pd.read\_csv("/Users/huongngo/Desktop/PERSONAL PROJECTS/zuckflix\_meta/Data/netflix\_titles.csv") |
|  | # printing the first 5 rows of dataset |
|  | netflix\_titles.head() |

[view raw](https://gist.github.com/huongngo-8/4a754daaecb2e9a7f80a4f39403aef54/raw/8652f0cfe5ce5f3c5d6ee65f25b4e0952deeefe6/code.py)[code.py](https://gist.github.com/huongngo-8/4a754daaecb2e9a7f80a4f39403aef54#file-code-py)hosted with ❤ by [GitHub](https://github.com/)



Doing this will give you a good idea of what data types you might be dealing with, what columns you need to perform transformations or cleaning, and other data you might be able to extract.

Before we look at this more closely, let’s perform the next step.

**2. Save the variables to a list**

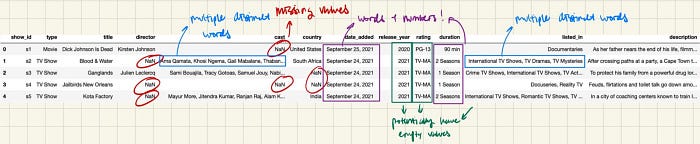
You want to do this to have **easy access to the different columns of the dataset**, especially when you want to perform the same transformations to different subsets of columns.

|  |
| --- |
| # getting the columns of the dataset |
|  | columns = list(netflix\_titles.columns) |
|  | columns |
|  |  |
|  | """ |
|  | Output: |
|  | ['show\_id', |
|  | 'type', |
|  | 'title', |
|  | 'director', |
|  | 'cast', |
|  | 'country', |
|  | 'date\_added', |
|  | 'release\_year', |
|  | 'rating', |
|  | 'duration', |
|  | 'listed\_in', |
|  | 'description'] |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/f8bc85b82ab27733436b6314cf7e6a41/raw/79a6d2c5fc235900a04e08f2330b9118ca4f7140/code_1.py)[code\_1.py](https://gist.github.com/huongngo-8/f8bc85b82ab27733436b6314cf7e6a41#file-code_1-py)hosted with ❤ by [GitHub](https://github.com/)

**3. Note down potential issues you will have to address in each column.**

To stay organized, note the issues you see in your dataset (by taking a glimpse of your dataset like in Step 1).



This picture above represents what I can see just from glimpsing at the dataset and is something that **you should think about when you’re looking at your dataset**. Here are a few things that stand out to me:

* There are some columns with missing values. This could cause a lot of problems for analysis and plotting if not addressed and resolved early in the process.
* There are columns with words and numbers, such as date\_added and duration. This can be a problem if we want to **make time-series graphs** by the date, or **other plots to explore duration’s relationship with other variables.**
* There are 2 columns with multiple distinct words joined together by a comma. This is an issue if we want to make **plots exploring the distribution of listed\_in (genre) or the actors on Netflix.**
* Other columns could potentially have missing values. The next step looks at the way to **check which columns have missing values** and **how much missing data they have.**

**Step 2: Look at the proportion of missing data**

|  |
| --- |
| # examining missing values |
|  | print("Missing values distribution: ") |
|  | print(netflix\_titles.isnull().mean()) |
|  | print("") |
|  |  |
|  | """ |
|  | Output: |
|  | Missing values distribution: |
|  | show\_id 0.000000 |
|  | type 0.000000 |
|  | title 0.000000 |
|  | director 0.299080 |
|  | cast 0.093675 |
|  | country 0.094357 |
|  | date\_added 0.001135 |
|  | release\_year 0.000000 |
|  | rating 0.000454 |
|  | duration 0.000341 |
|  | listed\_in 0.000000 |
|  | description 0.000000 |
|  | dtype: float64 |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/11bf62a377730005486e68e1c252d3a3/raw/510d4e5863b2517a336d4c7221d36543a2c39c03/code_2.py)[code\_2.py](https://gist.github.com/huongngo-8/11bf62a377730005486e68e1c252d3a3#file-code_2-py)hosted with ❤ by [GitHub](https://github.com/)

From this code chunk, you can easily look at the distribution of missing values in the dataset to get a good idea of which columns you’ll need to work with to resolve the missing values issue.

From the output, these are insights you can gather:

* director column has the highest percentage of missing data ~ 30%
* cast and country column also has a considerable percentage of missing data ~ 9%
* date\_added, rating and duration don’t have that much missing data ~ 0% - 0.1%
* Fortunately, most other columns are not empty.

Your next question is probably, **how do I deal with these columns with missing values?**

There are a few ways to deal with it:

1. Drop the column completely. If the column isn’t that important to your analysis, just drop it.
2. Keep the column. In this case, because **the director, cast and country columns are quite important to my analysis**, I will keep them.
3. **Imputation — the process of replacing missing data with substituted values.** Here, it is not possible to do so because most of the data are string values and not numerical values. However, I will be writing an article that talks more about imputation in detail, why and when it should be used, and how you can use it in R and Python with the help of some packages.

Before I continue, I will bring up the issue of missing values **across rows.**

In some cases, you might want to examine the **distribution of missing values across all the rows of your dataset** (given that your dataset doesn’t have a large number of observations/rows). From here, you can **choose from the choices above depending on how important the rows are to your analysis**. For instance, your dataset contains recorded data of something that is changing over time. Even though a row can contain missing values, you might not want to eliminate it because there is important time information you want to retain.

Let’s continue to step 3 before I show you how to deal with the NaN values even after keeping the columns.

**Step 3: Check the data type of each column**

|  |
| --- |
| # check datatype in each column |
|  | print("Column datatypes: ") |
|  | print(netflix\_titles.dtypes) |
|  |  |
|  | """ |
|  | Output: |
|  | Column datatypes: |
|  | show\_id object |
|  | type object |
|  | title object |
|  | director object |
|  | cast object |
|  | country object |
|  | date\_added object |
|  | release\_year int64 |
|  | rating object |
|  | duration object |
|  | listed\_in object |
|  | description object |
|  | dtype: object |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/875e716e62352de7f4584837ad524ceb/raw/b1fe1ddd13a1c53380df95c52fa33912df3237e2/code_3.py)[code\_3.py](https://gist.github.com/huongngo-8/875e716e62352de7f4584837ad524ceb#file-code_3-py)hosted with ❤ by [GitHub](https://github.com/)

Here, you can see that all the columns have object as their datatype aside from release\_year. In pandas, object means either string or mixed type (numerical and non-numerical type mixed). And from our dataset, you’ll be able to tell which columns are strictly string and mixed type.

**Step 4: If you have columns of strings, check for trailing whitespaces**

|  |
| --- |
| # getting all the columns with string/mixed type values |
|  | str\_cols = list(netflix\_titles.columns) |
|  | str\_cols.remove('release\_year') |
|  |  |
|  | # removing leading and trailing characters from columns with str type |
|  | for i in str\_cols: |
|  | netflix\_titles[i] = netflix\_titles[i].str.strip() |

[view raw](https://gist.github.com/huongngo-8/dbcb244e2e95633f6f2551aa9371458a/raw/d83881216c48d19fe92136e3c9c578984c68d73a/code_4.py)

After we know which data types we are dealing with, let’s make sure we remove any trailing characters and whitespace using strip .

**Step 5: Dealing with Missing Values (NaN Values)**

Referring back to the columns of missing values, let’s take a look at the columns: director, cast, country, date\_added, rating, duration. We can segment these columns by whether they are a string or mixed type.

String: director, cast, country, rating (here, it’s a string and not mixed because the numerical values won’t have any meaning if separated)

Mixed: date\_added, duration

NaN means Not a Number in pandas. It is a special floating-point value that is different from NoneType in Python. NaN values can be annoying to work with, especially when you want to filter them out for plots or analysis. To make our lives easier, let’s **replace these NaN values with something else.**

For string type values, we can replace NaN values with “” or “None” or any string that can indicate to you that there isn’t any value in that entry. Here, I chose to replace it with “” using the fillna function. Because it’s not an in-place function, I reassigned the changed values to the column in the dataset.

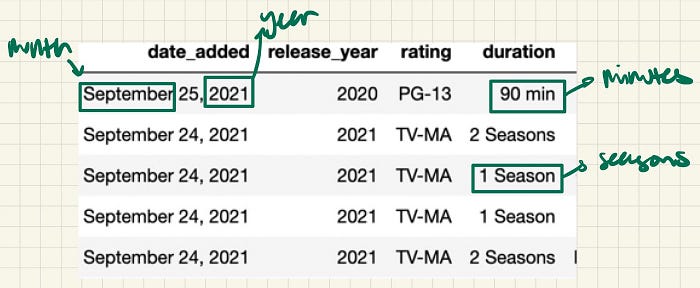
|  |
| --- |
| # names of the columns |
|  | columns = ['director', 'cast', 'country', 'rating', 'date\_added'] |
|  |  |
|  | # looping through the columns to fill the entries with NaN values with "" |
|  | for column in columns: |
|  | netflix\_titles[column] = netflix\_titles[column].fillna("") |

[view raw](https://gist.github.com/huongngo-8/c7d77f9f8a3f67ff693433b63c64eab2/raw/0097d935908b87a7db800835347c6f20d91ad1a8/code_5.py)[code\_5.py](https://gist.github.com/huongngo-8/c7d77f9f8a3f67ff693433b63c64eab2#file-code_5-py)hosted with ❤ by [GitHub](https://github.com/)

Here, you must have noticed that I left out the duration column. This is because we’ll be doing something with that column later down the road.

**Step 6: See if there are any other variables that you can obtain by extracting them from other variables**

For mixed-type values, before we tackle the missing value issue, let’s see if we can extract any data to make our analysis richer or process easier.



Looking at date\_added, we can see that it contains the month, date, and year that the film/show was added. Instead of having all this information in one column, why not try to separate them? That way, we can choose to isolate how month or year interacts with the other variables instead of looking at date\_added where its granularity will make it difficult for any trend to be discovered.

Below, I’ve written code to not only separate the information into 2 other columns but also filtered out the rows with NaN values and replaced them with 0, just like what was done before with “”.

|  |
| --- |
| # examining rows with null values for date\_added column |
|  | rows = [] |
|  | for i in range(len(netflix\_titles)): |
|  | if netflix\_titles['date\_added'].iloc[i] == "": |
|  | rows.append(i) |
|  |  |
|  | # examine those rows to confirm null state |
|  | netflix\_titles.loc[rows, :] |
|  | # extracting months added and years added |
|  | month\_added = [] |
|  | year\_added = [] |
|  | for i in range(len(netflix\_titles)): |
|  | # replacing NaN values with 0 |
|  | if i in rows: |
|  | month\_added.append(0) |
|  | year\_added.append(0) |
|  | else: |
|  | date = netflix\_titles['date\_added'].iloc[i].split(" ") |
|  | month\_added.append(date[0]) |
|  | year\_added.append(int(date[2])) |
|  |  |
|  | # turning month names into month numbers |
|  | for i, month in enumerate(month\_added): |
|  | if month != 0: |
|  | datetime\_obj = datetime.strptime(month, "%B") |
|  | month\_number = datetime\_obj.month |
|  | month\_added[i] = month\_number |
|  |  |
|  | # checking all months |
|  | print(set(month\_added)) |
|  | print(set(year\_added)) |
|  |  |
|  | # inserting the month and year columns into the dataset |
|  | netflix\_titles.insert(7, "month\_added", month\_added, allow\_duplicates = True) |
|  | netflix\_titles.insert(8, "year\_added", year\_added, allow\_duplicates = True)netflix\_titles.head() |

[view raw](https://gist.github.com/huongngo-8/20c8f6a3777ce98e5dee7a8d6a66d3bb/raw/438a0fc8839f0e7590c8af3bb80ff8a371243a4c/code_6.py)[code\_6.py](https://gist.github.com/huongngo-8/20c8f6a3777ce98e5dee7a8d6a66d3bb#file-code_6-py)hosted with ❤ by [GitHub](https://github.com/)

https://miro.medium.com/v2/resize:fit:700/0*9PBmF1ucR08a6UVA

Now, the new dataset contains the month\_added and year\_added columns. This will allow us to do some trend analysis later.

Looking at duration, on top of it being a mixed type, there are also 2 different time units in this column. This is a problem because we are dealing with 2 different types of content that are measured differently for time. Thus, making graphs for durationwill be quite difficult to interpret if we keep them as it is. The good thing is that there are many ways to deal with this issue. The way I’ve chosen to deal with it is by **separating the type of content into 2 different datasets and naturally, the duration column will just be numerical and just have 1 type of time unit.** This way, you can **easily and clearly plot using the values.**

|  |
| --- |
| # separating original dataset to tv show and movie dataset respectively |
|  | shows = [] |
|  | films = [] |
|  |  |
|  | # looping through the dataset to identify rows that are TV shows and films |
|  | for i in range(len(netflix\_titles)): |
|  | if netflix\_titles['type'].iloc[i] == "TV Show": |
|  | shows.append(i) |
|  | else: |
|  | films.append(i) |
|  |  |
|  | # grouping rows that are TV shows |
|  | netflix\_shows = netflix\_titles.loc[shows, :] |
|  |  |
|  | #grouping rows that are films |
|  | netflix\_films = netflix\_titles.loc[films, :] |
|  |  |
|  | # reseting the index of the new datasets |
|  | netflix\_shows = netflix\_shows.set\_index([pd.Index(range(0, len(netflix\_shows)))]) |
|  | netflix\_films = netflix\_films.set\_index([pd.Index(range(0, len(netflix\_films)))]) |

[view raw](https://gist.github.com/huongngo-8/f5ba9c8aca19611dfbfbe9c51207f21b/raw/cf7485b30c6e579a22a65765b6c584134c73674d/code_7.py)[code\_7.py](https://gist.github.com/huongngo-8/f5ba9c8aca19611dfbfbe9c51207f21b#file-code_7-py)hosted with ❤ by [GitHub](https://github.com/)

Because the duration column has both strings and numbers, **I’ll also have to create a function to extract the number from that column so that it can be inserted into the columns of the 2 new datasets.**

|  |
| --- |
| # get length of movie or number of seasons of show |
|  | def getDuration(data): |
|  | count = 0 |
|  | durations = [] |
|  | for value in data: |
|  | # filling in missing values |
|  | if type(value) is float: |
|  | durations.append(0) |
|  | else: |
|  | values = value.split(" ") |
|  | durations.append(int(values[0])) |
|  | return durations |
|  |  |
|  | # inserting new duration type column for shows (renamed column) |
|  | netflix\_shows.insert(11, 'seasons', getDuration(netflix\_shows['duration'])) |
|  | netflix\_shows = netflix\_shows.drop(['duration'], axis = 1) |
|  | netflix\_shows.head() |
|  |  |
|  | # inserting new duration type column for films (renamed column) |
|  | netflix\_films.insert(11, 'length', getDuration(netflix\_films['duration'])) |
|  | netflix\_films = netflix\_films.drop(['duration'], axis = 1) |
|  | netflix\_films.head() |

[view raw](https://gist.github.com/huongngo-8/831a51839b20972b0f800fb45fe9ab63/raw/0ca2ba17f67da6381ecf5fc77cc8fb2fa147c9bf/code_8.py)[code\_8.py](https://gist.github.com/huongngo-8/831a51839b20972b0f800fb45fe9ab63#file-code_8-py)hosted with ❤ by [GitHub](https://github.com/)

**Step 7: Check the unique values of columns**

Beyond potentially missing values, there could be corrupted values that you can run into once you perform analysis. To check this, we can check for unique values for some of the columns. Let’s refer to the first 5 rows of the datasets as our starting point.

https://miro.medium.com/v2/resize:fit:700/0*CLrZpiLukHhnFAaj

https://miro.medium.com/v2/resize:fit:700/0*-NiCLerCzibmjiN9

It might not be strategic to check the unique values of all the columns, especially the title, director, and cast as there could be a large number of unique values to examine. Instead, let’s focus on a list of potential unique values that could be easier and more important to check given that it could be more insightful for future analysis. From a glimpse at the datasets, the columns country, rating, listed\_in are probably the ones of interest. Let’s examine the rating column first as that seems to be the least complicated one to deal with.

You can easily obtain the unique values of a column like rating using Python’s built-in function, unique. Let’s try that!

|  |
| --- |
| # getting the unique ratings for films |
|  | netflix\_films['rating'].unique() |
|  |  |
|  | """ |
|  | Output: |
|  | array(['PG-13', 'PG', 'TV-MA', 'TV-PG', 'TV-14', 'TV-Y', 'R', 'TV-G', |
|  | 'TV-Y7', 'G', 'NC-17', '74 min', '84 min', '66 min', 'NR', '', |
|  | 'TV-Y7-FV', 'UR'], dtype=object) |
|  | """ |
|  |  |
|  | # getting the unique ratings for shows |
|  | netflix\_shows['rating'].unique() |
|  |  |
|  | """ |
|  | Output: |
|  | array(['TV-MA', 'TV-14', 'TV-Y7', 'TV-PG', 'TV-Y', 'TV-G', 'R', 'NR', '', |
|  | 'TV-Y7-FV'], dtype=object) |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/35a08e4af10c76288eb70edb457f2d04/raw/3eb407e83347a58a3f7ce2bfcd4bae5770e8bf6c/code_9.py)[code\_9.py](https://gist.github.com/huongngo-8/35a08e4af10c76288eb70edb457f2d04#file-code_9-py)hosted with ❤ by [GitHub](https://github.com/)

This seems interesting. Why are there 74 min, 84 min, and 66 min in the unique types of rating for films? And why are there UR (Unrated) and NR (Not Rated)? Aren’t they supposed to mean the same thing? Let’s investigate this further by extracting the rows that have these weird entries.

|  |
| --- |
| # printing more details of the rows that have incorrect ratings |
|  | incorrect\_ratings = ['74 min', '84 min', '66 min'] |
|  | for i in range(len(netflix\_films)): |
|  | if netflix\_films['rating'].iloc[i] in incorrect\_ratings: |
|  | print(netflix\_films.iloc[i]) |
|  | print("") |
|  |  |
|  | """ |
|  | Output: |
|  | show\_id s5542 |
|  | type Movie |
|  | title Louis C.K. 2017 |
|  | director Louis C.K. |
|  | cast Louis C.K. |
|  | country United States |
|  | date\_added April 4, 2017 |
|  | month\_added 4 |
|  | year\_added 2017 |
|  | release\_year 2017 |
|  | rating 74 min |
|  | length 0 |
|  | listed\_in Movies |
|  | description Louis C.K. muses on religion, eternal love, gi... |
|  | Name: 3562, dtype: object |
|  |  |
|  | show\_id s5795 |
|  | type Movie |
|  | title Louis C.K.: Hilarious |
|  | director Louis C.K. |
|  | cast Louis C.K. |
|  | country United States |
|  | date\_added September 16, 2016 |
|  | month\_added 9 |
|  | year\_added 2016 |
|  | release\_year 2010 |
|  | rating 84 min |
|  | length 0 |
|  | listed\_in Movies |
|  | description Emmy-winning comedy writer Louis C.K. brings h... |
|  | Name: 3738, dtype: object |
|  |  |
|  | show\_id s5814 |
|  | type Movie |
|  | title Louis C.K.: Live at the Comedy Store |
|  | director Louis C.K. |
|  | cast Louis C.K. |
|  | country United States |
|  | date\_added August 15, 2016 |
|  | month\_added 8 |
|  | year\_added 2016 |
|  | release\_year 2015 |
|  | rating 66 min |
|  | length 0 |
|  | listed\_in Movies |
|  | description The comic puts his trademark hilarious/thought... |
|  | Name: 3747, dtype: object |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/51589b66675ee4f01d6ccd4ac8686350/raw/24daec150ab0d1647d2279bd4909bbf0e400d068/code_10.py)[code\_10.py](https://gist.github.com/huongngo-8/51589b66675ee4f01d6ccd4ac8686350#file-code_10-py)hosted with ❤ by [GitHub](https://github.com/)

Using this code chunk, we can see that 3 distinct rows contain this weird rating and that it actually belongs to the length column. We can also see the row number where the issue is located which will be useful to use for fixing the entries.

After some quick Googling, we can proceed to fix these entries by moving the “wrong ratings” (actually duration) to the length column and entering the right ratings.

|  |
| --- |
| # getting the row indices |
|  | index = [3562, 3738, 3747] |
|  |  |
|  | # fixing the entries |
|  | for i in index: |
|  | split\_value = netflix\_films['rating'].iloc[i].split(" ") |
|  | length = split\_value[0] |
|  | netflix\_films['duration'].iloc[i] = length |
|  | netflix\_films['rating'].iloc[i] = "NR" |
|  |  |
|  | # double checking the entries again |
|  | for i in index: |
|  | print(netflix\_films.iloc[i]) |

[view raw](https://gist.github.com/huongngo-8/74b898a2195341a239a527697e222305/raw/d07f238db1367c38d279065ef4293bb7d05fc077/code_11.py)[code\_11.py](https://gist.github.com/huongngo-8/74b898a2195341a239a527697e222305#file-code_11-py)hosted with ❤ by [GitHub](https://github.com/)

For the UR and NR values in the rating column, we should keep the consistency where NR is used in the netflix\_shows dataset and change UR values to NR.

|  |
| --- |
| # fixing the entries |
|  | for i in range(len(netflix\_films)): |
|  | if netflix\_films['rating'].iloc[i] == "UR": |
|  | netflix\_films['rating'].iloc[i] = "NR" |
|  |  |
|  | # double checking |
|  | netflix\_films['rating'].unique() |
|  |  |
|  | """ |
|  | array(['PG-13', 'PG', 'TV-MA', 'TV-PG', 'TV-14', 'TV-Y', 'R', 'TV-G', |
|  | 'TV-Y7', 'G', 'NC-17', 'NR', '', 'TV-Y7-FV'], dtype=object) |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/fb24f922ad32c7f1f0dc4f1f71f9c01a/raw/5439b87de46b082ca4b17b0e0ef830841bc93973/code_12.py)[code\_12.py](https://gist.github.com/huongngo-8/fb24f922ad32c7f1f0dc4f1f71f9c01a#file-code_12-py)hosted with ❤ by [GitHub](https://github.com/)

Now that we’ve cleaned up the rating column, let’s look at the country and listed\_in column. By now, you must have realized that it’s not as easy as the rating column to extract unique values. This is because the values in those columns are words joined together by commas, making it more difficult to extract the set of words and then find unique words from that set.

How we’re going to get around this issue is by implementing a unique function for this special case.

To start, let’s think about what data structure can give us unique values easily. If you guessed sets, you’re right! Given its ability to **store unique elements of the same type in sorted order,** it’s a fitting data structure for what we want to do.

Then, to extract those words that are joined by commas, we can use the split function to split up the string by the comma.

|  |
| --- |
| # function to get unique values of a column |
|  | def getUnique(data): |
|  | unique\_values = set() |
|  | for value in data: |
|  | if type(value) is float: |
|  | unique\_values.add(None) |
|  | else: |
|  | values = value.split(", ") |
|  | for i in values: |
|  | unique\_values.add(i) |
|  | return list(unique\_values) |

[view raw](https://gist.github.com/huongngo-8/e06965a68ca5d1d24a71ec3ebf9250bd/raw/ba75882b1e1fc60cb2f32ec1d63a10c2a565773e/code_13.py)[code\_13.py](https://gist.github.com/huongngo-8/e06965a68ca5d1d24a71ec3ebf9250bd#file-code_13-py)hosted with ❤ by [GitHub](https://github.com/)

After using the function, we can easily obtain the unique values for the country and listed\_in columns.

|  |
| --- |
| # getting unique country names |
|  | unique\_countries = getUnique(netflix\_titles['country']) |
|  | unique\_countries |
|  |  |
|  | """ |
|  | Output: |
|  | ['', |
|  | 'Czech Republic', |
|  | 'Armenia', |
|  | 'Belgium', |
|  | 'Mozambique', |
|  | 'East Germany'\*, |
|  | 'West Germany'\*, |
|  | 'Soviet Union'\*, |
|  | 'Burkina Faso', etc.] (shortened for article) |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/2ad05383ed680e205bf26aeb7ec7313c/raw/1392f09e47bbc43520ed72e4d7767c89f101ef61/code_14.py)[code\_14.py](https://gist.github.com/huongngo-8/2ad05383ed680e205bf26aeb7ec7313c#file-code_14-py)hosted with ❤ by [GitHub](https://github.com/)

Next, let’s examine the list of unique countries to see if there are any inconsistencies or mistakes. By doing so and with a little bit of Googling, we can see there are some issues with this list:

* There’s both the Soviet Union and Russia
* There’s both the West/East Germany and Germany

We can easily fix this with a few modifications to the dataset.

|  |
| --- |
| # converting soviet union to russia and east/west germany to germany |
|  | for i in range(len(netflix\_titles)): |
|  | if type(netflix\_titles['country'].iloc[i]) is not float: |
|  | countries = netflix\_titles['country'].iloc[i].split(", ") |
|  | for j in range(len(countries)): |
|  | if "Germany" in countries[j]: |
|  | countries[j] = "Germany" |
|  | elif "Soviet Union" in countries[j]: |
|  | countries[j] = "Russia" |
|  | netflix\_titles['country'].iloc[i] = ", ".join(countries) |

[view raw](https://gist.github.com/huongngo-8/c162a6345d854b361c8064a7f7da09ae/raw/bbd353b7bbe7c4b42c98c0a31a1ba7cb395db9a8/code_15.py)[code\_15.py](https://gist.github.com/huongngo-8/c162a6345d854b361c8064a7f7da09ae#file-code_15-py)hosted with ❤ by [GitHub](https://github.com/)

As for the list of genres, we can see that there are some genres we might not want or need to include. Thus, we can easily remove it from the dataset to make our analysis less confounding.

|  |
| --- |
| # getting unique film genres |
|  | unique\_genres\_films = getUnique(netflix\_films['listed\_in']) |
|  | unique\_genres\_films |
|  |  |
|  | """ |
|  | Output: |
|  | ['International Movies', |
|  | 'Children & Family Movies', |
|  | 'LGBTQ Movies', |
|  | 'Classic Movies', |
|  | 'Action & Adventure', |
|  | 'Stand-Up Comedy', |
|  | 'Sports Movies', |
|  | 'Documentaries', |
|  | 'Movies'\*, |
|  | 'Music & Musicals', |
|  | 'Romantic Movies', |
|  | 'Anime Features', |
|  | 'Comedies', |
|  | 'Independent Movies', |
|  | 'Dramas', |
|  | 'Thrillers', |
|  | 'Cult Movies', |
|  | 'Faith & Spirituality', |
|  | 'Horror Movies', |
|  | 'Sci-Fi & Fantasy'] |
|  | """ |
|  |  |
|  | # getting unique show genres |
|  | unique\_genres\_shows = getUnique(netflix\_shows['listed\_in']) |
|  | unique\_genres\_shows |
|  |  |
|  | """ |
|  | Output: |
|  | ['TV Dramas', |
|  | 'TV Comedies', |
|  | 'TV Action & Adventure', |
|  | 'TV Mysteries', |
|  | 'Romantic TV Shows', |
|  | "Kids' TV", |
|  | 'TV Horror', |
|  | 'International TV Shows', |
|  | 'TV Sci-Fi & Fantasy', |
|  | 'Korean TV Shows', |
|  | 'Spanish-Language TV Shows', |
|  | 'Science & Nature TV', |
|  | 'Crime TV Shows', |
|  | 'TV Shows'\*, |
|  | 'Classic & Cult TV', |
|  | 'Teen TV Shows', |
|  | 'TV Thrillers', |
|  | 'Stand-Up Comedy & Talk Shows', |
|  | 'Docuseries', |
|  | 'Reality TV', |
|  | 'British TV Shows', |
|  | 'Anime Series'] |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/42f37874dd89e4594fed28f058379ffa/raw/1da56a42a614587e8dd496fe13ef91f6a6c64225/code_16.py)[code\_16.py](https://gist.github.com/huongngo-8/42f37874dd89e4594fed28f058379ffa#file-code_16-py)hosted with ❤ by [GitHub](https://github.com/)

In both the TV shows and films dataset, there is a “TV Shows” and “Movies” genre. Technically, this isn’t a genre but could be a label of the type of content. To confirm this, we should print out the counts of these “genres” appearing in the respective datasets.

The hypothesis is that if these “genres” appear in all the rows of the datasets, it means that they’re simply labels. Otherwise, we’ll have to investigate further as to what those “genres” represent.

|  |
| --- |
| # checking for TV shows |
|  | # replace netflix\_shows with netflix\_films to check for movies |
|  | count = 0 |
|  | index = [] |
|  | for i, value in enumerate(netflix\_shows['listed\_in']): |
|  | genres = value.split(", ") |
|  | if "TV Shows" in genres: |
|  | count += 1 |
|  | index.append(i) |
|  | print("count %s" %count) |
|  | print("index %s" %index) |
|  |  |
|  | """ |
|  | Output: |
|  | TV shows: |
|  | count 16 |
|  | index [59, 110, 272, 286, 452, 599, 991, 1432, 1548, 1808, 1840, 2107, 2160, 2190, 2465, 2559] |
|  |  |
|  | Movies: |
|  | count 57 |
|  | index [197, 310, 456, 457, 458, 476, 477, 1906, 1938, 1941, 2146, 2165, 2621, 2711, 2758, 2862, 2863, 2867, 3036, 3137, 3138, 3139, 3140, 3141, 3142, 3225, 3226, 3228, 3232, 3517, 3562, 3652, 3694, 3722, 3738, 3747, 3789, 3824, 3883, 4271, 4273, 4543, 4544, 4784, 4910, 4911, 5006, 5178, 5259, 5290, 5292, 5293, 5295, 5476, 5477, 5478, 6092] |
|  | """ |

[view raw](https://gist.github.com/huongngo-8/aebbdccee437b06a6d81e1fcab45e159/raw/ff1a8134a55352c2174fff35b977d0532d4141b9/code_17.py)[code\_17.py](https://gist.github.com/huongngo-8/aebbdccee437b06a6d81e1fcab45e159#file-code_17-py)hosted with ❤ by [GitHub](https://github.com/)

As the count of the “genres” is less than the size of the datasets, let’s use the output of the code to examine the rows.

Since I’ve written the code to specifically output the row indices in a list, we can easily use that list and the iloc function to get a view of the rows.

|  |
| --- |
| # printing the first 5 rows of all rows that have TV Shows as its genre |
|  | netflix\_shows.iloc[index[0:5]] |
|  |  |
|  | # printing the first 5 rows of all rows that have Movies as its genre |
|  | netflix\_films.iloc[index[0:5]] |

[view raw](https://gist.github.com/huongngo-8/63da74d389440e80ebea5cd818b2b9ef/raw/60d1ef20ab5fc055efce348331f6bd4acb796084/code_18.py)[code\_18.py](https://gist.github.com/huongngo-8/63da74d389440e80ebea5cd818b2b9ef#file-code_18-py)hosted with ❤ by [GitHub](https://github.com/)

https://miro.medium.com/v2/resize:fit:700/0*xkf-bz9WvmIlEten



Taking a look at the rows, it is now obvious that the “TV Shows” and “Movies” genre was used to signify that these content didn’t have a genre in the first place. Now that we understand what this meant, we can either choose to exclude or include it in our analysis. Here, I’ve chosen to include it because it doesn’t affect my analysis.

Although this step is tedious, it is also quite important as it allows us to find the issues in our dataset that are hidden away at a first glance.

**Step 8: Join the cleaned datasets together to create another dataset [Optional]**

This step is optional, but in the case that you’d want the cleaned TV shows and movies dataset in one place, you should **concatenate** them.

And that’s it! You’ve successfully cleaned this dataset. Keep in mind that everyone has their methodology of data cleaning, and a lot of it is just from putting in the effort to understand your dataset. However, I hope that this article has helped you understand why data scientists spend 80% of their time cleaning their datasets. **In all seriousness, this article highlights the importance of data cleaning and more importantly, the need for a good data cleaning methodology which will help you keep your work organized which will help if you need to go back to it during the analysis process**. You can check out the full notebook [here](https://github.com/huongngo-8/data_clean_demo).

**Data Cleaning Using Pandas in Python – Complete Guide for Beginners**

Introduction

As we know, Data Science is the discipline of study that involves extracting insights from huge amounts of data by the use of various scientific methods, algorithms, and processes. To extract useful knowledge from data, Data Scientists need raw data. This Raw data is a collection of information from various outlined sources and an essential raw material for Data Scientists. It is also known as primary or source data, which is messy and needs cleaning. This beginner’s guide will tell you all about data cleaning using pandas in Python.

The primary data consists of irregular and inconsistent values, which lead to many difficulties. When using data, the insights and analysis extracted are only as good as the data we use. Essentially, when irregular data is in, then irregular analysis comes out. Here’s where data cleaning comes into play. Data cleansing is an essential part of the data analytics process. Data cleaning removes incorrect, corrupted, garbage, incorrectly formatted, duplicate, or incomplete data within a dataset.

**Learning Objectives**

* Define data cleaning and its importance in the data analytics process.
* Recognize the importance of accurate, complete, and consistent data for effective analysis and decision-making.
* Learn the various techniques and tools available in the Python Pandas library for data cleaning.

Table of Contents

1. [What Is Data Cleaning?](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#What_Is_Data_Cleaning?)
2. [Why Is Data Cleaning Essential?](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Why_Is_Data_Cleaning_Essential?)
3. [Data Cleaning Cycle](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Data_Cleaning_Cycle)
4. [Data Cleaning With Pandas](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Data_Cleaning_With_Pandas)
5. [Conclusion](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Conclusion)
6. [Frequently Asked Questions](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Frequently_Asked_Questions)
   * [Q1. What do you mean by data type casting in the context of data analysis and data cleaning?](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Q1._What_do_you_mean_by_data_type_casting_in_the_context_of_data_analysis_and_data_cleaning?)
   * [Q2. When is it appropriate to drop missing values in data rather than imputing them in the context of data cleaning with Pandas?](https://www.analyticsvidhya.com/blog/2021/06/data-cleaning-using-pandas/#Q2._When_is_it_appropriate_to_drop_missing_values_in_data_rather_than_imputing_them_in_the_context_of_data_cleaning_with_Pandas?)

What Is Data Cleaning?

When working with multiple data sources, there are many chances for data to be incorrect, duplicated, or mislabeled. If data is wrong, outcomes and algorithms are unreliable, even though they may look correct. *Data cleaning* is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset. There’s no such absolute way to describe the precise steps in the data cleaning process because the processes may vary from dataset to dataset. Data cleansing, data cleansing, or data scrub is the general data preparation process initiative. Data cleaning plays an important part in developing reliable answers within the analytical process and is observed to be a basic feature of the info science basics. The motive of data cleaning services is to construct uniform and standardized data sets that enable easy access to data analytics tools and business intelligence and perceive accurate data for each problem.

Why Is Data Cleaning Essential?

Data cleaning is the most important task that should be done by a data science professional. Having wrong or bad-quality data can be detrimental to processes and analysis. Having clean data will ultimately increase overall productivity and permit the very best quality information in your decision-making.



Following are some reasons why data cleaning is essential:

**1. Error-Free Data:** When multiple sources of data are combined, there may be a chance of so much error. Through Data Cleaning, errors can be removed from data. Having clean data which is free from wrong and garbage values can help in performing analysis faster as well as efficiently. By doing this task our considerable amount of time is saved. The results won’t be accurate if we use data containing garbage values. When we don’t use accurate data, we will surely make mistakes. Monitoring errors and good reporting helps to find where errors are coming from and also makes it easier to fix incorrect or corrupt data for future applications.

**2. Data Quality:** The quality of the data is the degree to which it follows the rules of particular requirements. For example, if we have imported phone numbers data of different customers, and in some places, we have added email addresses of customers in the data. But because our needs were straightforward for phone numbers, then the email addresses would be invalid data. Here some pieces of data follow a specific format. Some types of numbers have to be in a specific range. Some data cells might require selected quiet data like numeric, Boolean, etc. In every scenario, there are some mandatory constraints our data should follow. Certain conditions affect multiple fields of data in a particular form. Particular types of data have unique restrictions. It will always be invalid if the data isn’t in the required format. Data cleaning will help us simplify this process and avoid useless data values.

**3. Accurate and Efficient:** Ensuring the data is close to the correct values. We know that most of the data in a dataset are valid, and we should focus on establishing its accuracy. Even if the data is authentic and correct, it doesn’t mean it is accurate. Determining accuracy helps to figure out whether the data entered is accurate or not. For example, a customer’s address is stored in the specified format; maybe it doesn’t need to be in the right one. The email has an additional character or value that makes it incorrect or invalid. Another example is the phone number of a customer. This means that we have to rely on data sources to cross-check the data to figure out if it’s accurate or not. Depending on the kind of data we are using, we might be able to find various resources that could help us in this regard for cleaning.

**4. Complete Data:** Completeness is the degree to which we should know all the required values. Completeness is a little more challenging to achieve than accuracy or quality. Because it’s nearly impossible to have all the info we need, only known facts can be entered. We can try to complete data by redoing the data-gathering activities like approaching the clients again, re-interviewing people, etc. For example, we might need to enter every customer’s contact information. But a number of them might not have email addresses. In this case, we have to leave those columns empty. If we have a system that requires us to fill all columns, we can try to enter missing or unknown there. But entering such values does not mean that the data is complete. It would still be referred to as incomplete.

**5. Maintains Data Consistency:** To ensure the data is consistent within the same dataset or across multiple datasets, we can measure consistency by comparing two similar systems. We can also check the data values within the same dataset to see if they are consistent or not. Consistency can be relational. For example, a customer’s age might be 25, which is a valid value and also accurate, but it is also stated as a senior citizen in the same system. In such cases, we have to cross-check the data, similar to measuring accuracy, and see which value is true. Is the client a 25-year-old? Or is the client a senior citizen? Only one of these values can be true. There are multiple ways to for your data consistent.

* By checking in different systems.
* By checking the source.
* By checking the latest data.

Data Cleaning Cycle

It is the method of analyzing, distinguishing, and correcting untidy, raw data. Data cleaning involves filling in missing values, handling outliers, and distinguishing and fixing errors present in the dataset. Whereas the techniques used for data cleaning might vary in step with different types of datasets. In this tutorial, we will learn how to clean data using pandas. The following are standard steps to map out data cleaning:



Data Cleaning With Pandas

Data scientists spend a huge amount of time cleaning datasets and getting them in the form in which they can work. It is an essential skill of Data Scientists to be able to work with messy data, missing values, and inconsistent, noisy, or nonsensical data. To work smoothly, python provides a built-in module, Pandas. Pandas is the popular Python library that is mainly used for data processing purposes like cleaning, manipulation, and analysis. Pandas stand for “Python Data Analysis Library”. It consists of classes to read, process, and write csv files. There are numerous Data cleaning tools present, but the Pandas library provides a really fast and efficient way to manage and explore data. It does that by providing us with Series and DataFrames, which help us represent data efficiently and manipulate it in various ways.

In this article, we will use the Pandas module to clean our dataset.

We are using a simple dataset for data cleaning, i.e., the iris species dataset. You can download this dataset from [kaggle.com](http://kaggle.com/)[.](https://www.kaggle.com/uciml/iris)

Let’s get started with data cleaning step by step.

To start working with Pandas, we need to first import it. We are using Google Colab as IDE, so we will import Pandas in Google Colab.

#importing module

**import** pandas **as** pd

**Step 1: Import Dataset**

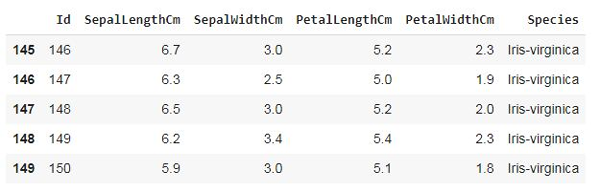
To import the dataset, we use the read\_csv() function of pandas and store it in the pandas DataFrame named as data. As the dataset is in tabular format, when working with tabular data in Pandas, it will be automatically converted into a DataFrame. DataFrame is a two-dimensional, mutable data structure in Python. It is a combination of rows and columns like an excel sheet.

**Python Code:**

The head() function is a built-in function in pandas for the dataframe used to display the rows of the dataset. We can specify the number of rows by giving the number within the parenthesis. By default, it displays the first five rows of the dataset. If we want to see the last five rows of the dataset, we use the tail()function of the dataframe like this:

#displayinf last five rows of dataset

data.tail()



**Step 2: Merge Dataset**

Merging the dataset is the process of combining two datasets in one and lining up rows based on some particular or common property for data analysis. We can do this by using the merge() function of the dataframe. Following is the syntax of the merge function:

DataFrame\_name.merge(**right**, how='inner', **on**=**None**, left\_on=**None**, right\_on=**None**, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'), **copy**=True, **indicator**=False, validate=**None**)

[[source]](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html)

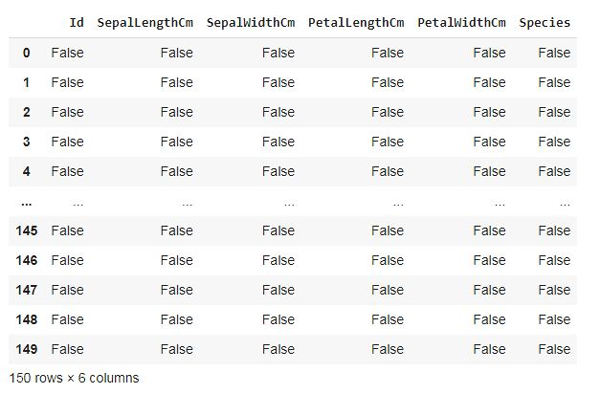
But in this case, we don’t need to merge two datasets. So, we will skip this step.

**Step 3: Rebuild Missing Data**

To find and fill in the missing data in the dataset, we will use another function. There are 4 ways to find the null values if present in the dataset. Let’s see them one by one:

**Using isnull() function:**

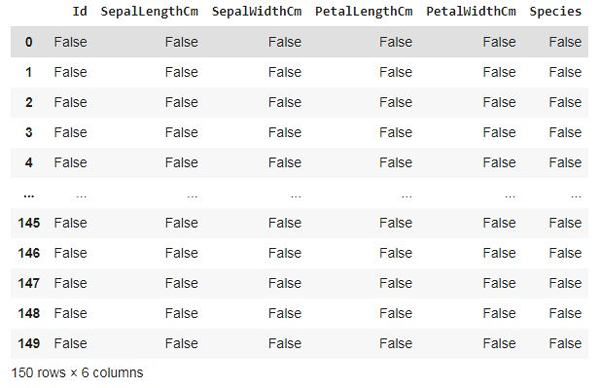
**data**.isnull()



This function provides the boolean value for the complete dataset to know if any null value is present or not.

**Using isna() function:**

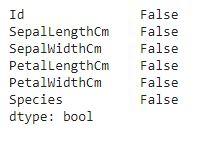
**data**.isna()



This is the same as the isnull() function. Ans provides the same output.

**Using isna().any()**

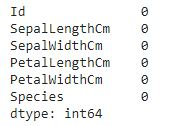
data.isna().any()



This function also gives a boolean value if any null value is present or not, but it gives results column-wise, not in tabular format.

**Using isna(). sum()**

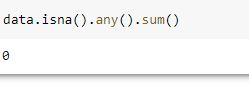
data.isna().sum()



This function gives the sum of the null values preset in the dataset column-wise.

**Using isna().any().sum()**

data.isna().any().sum()



This function gives output in a single value if any null is present or not.

There are no null values present in our dataset. But if there are any null values preset, we can fill those places with any other value using the fillna() function of DataFrame.Following is the syntax of fillna() function:

DataFrame\_name.fillna(**value**=**None**, **method**=**None**, axis=**None**, inplace=False, limit=**None**, downcast=**None**)

[[source]](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html)

This function will fill NA/NaN or 0 values in place of null spaces. You may also drop null values using the dropna method when the amount of missing data is relatively small and unlikely to affect the overall.

**Step 4: Standardization and Normalization**

Data Standardization and Normalization is a common practices in machine learning.

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

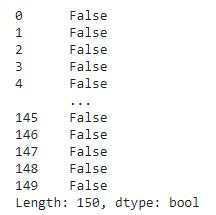
To know more about this, [click here.](https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/)

This step is not needed for the dataset we are using. So, we will skip this step.

**Step 5: De-Duplicate Data**

De-Duplicate means removing all duplicate values. There is no need for duplicate values in data analysis. These values only affect the accuracy and efficiency of the analysis result. To find duplicate values in the dataset, we will use a simple dataframe function, i.e., duplicated(). Let’s see the example:

**data**.duplicated()



This function also provides bool values for duplicate values in the dataset. As we can see, the dataset doesn’t contain any duplicate values. If a dataset contains duplicate values, it can be removed using the drop\_duplicates() function. Following is the syntax of this function:

DataFrame\_name.drop\_duplicates(**subset**=**None**, keep='first', inplace=False, ignore\_index=False)

[[source]](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html)

Step 6: Verify and Enrich the Data

After removing null, duplicate, and incorrect values, we should verify the dataset and validate its accuracy. In this step, we have to check that the data cleaned so far is making any sense. If the data is incomplete, we have to enrich the data again by data gathering activities like approaching the clients again, re-interviewing people, etc. Completeness is a little more challenging to achieve accuracy or quality in the dataset.

Step 7: Export Dataset

This is the last step of the data-cleaning process. After performing all the above operations, the data is transformed into a clean dataset, and it is ready to export for the next process in Data Science or Data Analysis.

Conclusion

Data cleaning is a critical task in data science that helps ensure the accuracy and reliability of analysis and decision-making. Through data cleaning, errors can be removed, data quality can be improved, and the data can be made more accurate and complete. By utilizing the various techniques and tools available for data cleaning in the Python Pandas library, data scientists can gain insights from the raw data and make better informed decisions.

**Key Takeaways**

* Data cleaning is the process of removing incorrect, corrupted, garbage, incorrectly formatted, duplicate, or incomplete data within a dataset.
* Data cleaning is essential for ensuring error-free data, data quality, accuracy, completeness, and efficiency in the analysis and decision-making process.
* Pandas is a popular data manipulation library in Python that provides powerful data-cleaning capabilities. It offers functions and methods to handle missing data, remove duplicate data, and fix data formatting issues.

Frequently Asked Questions

**Q1. What do you mean by data type casting in the context of data analysis and data cleaning?**

A. In the context of data analysis, casting data types means converting data from one type to another. This is often done to ensure consistency and accuracy in data analysis, as well as to enable specific operations or functions that are available for certain data types. For example, casting a string to a numerical data type can enable mathematical operations, while casting a numerical data type to a string can enable string-based operations.

**Q2. When is it appropriate to drop missing values in data rather than imputing them in the context of data cleaning with Pandas?**

A. It is appropriate to drop missing values in data when the amount of missing data is small compared to the overall size of the dataset, and the missing data is randomly distributed or when they would skew the analysis. if the amount of missing data is substantial or the missing data is non-random, it may be more appropriate to impute the missing values rather than drop them, as dropping them may result in a biased or incomplete analysis.