**Data Cleaning: Removing NaN Values and Duplicates**

The first step as always is getting a better idea about what we're dealing with.

**Preliminary Data Exploration**

**Challenge**: How many rows and columns does df\_apps have? What are the column names? What does the data look like? Look at a random sample of 5 different rows with [.sample()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html)

.

.

..

...

..

.

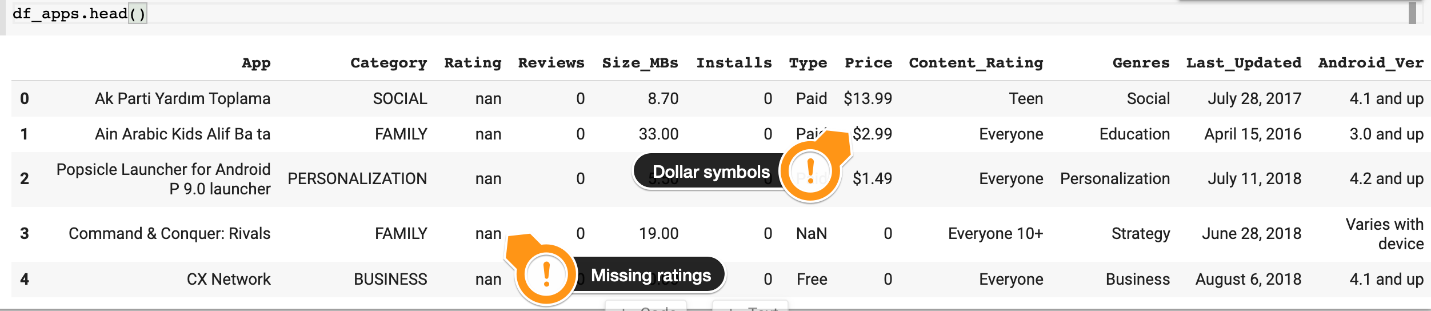
.

**Solution**

Compared to the previous projects we are working with a fairly large DataFrame this time.

1. df\_apps.shape

tells us we have 10841 rows and 12 columns.



We can already see that there are some data issues that we need to fix. In the Ratings and Type columns there are NaN (Not a number values) and in the Price column we have dollar signs that will cause problems.

The .sample(n) method will give us n random rows. This is another handy way to inspect our DataFrame.



**Challenge**: Remove the columns called Last\_Updated and Android\_Version from the DataFrame. We will not use these columns.

**Challenge**: How many rows have a NaN value (not-a-number) in the Rating column? Create DataFrame called df\_apps\_clean that does not include these rows.

.

.

..

...

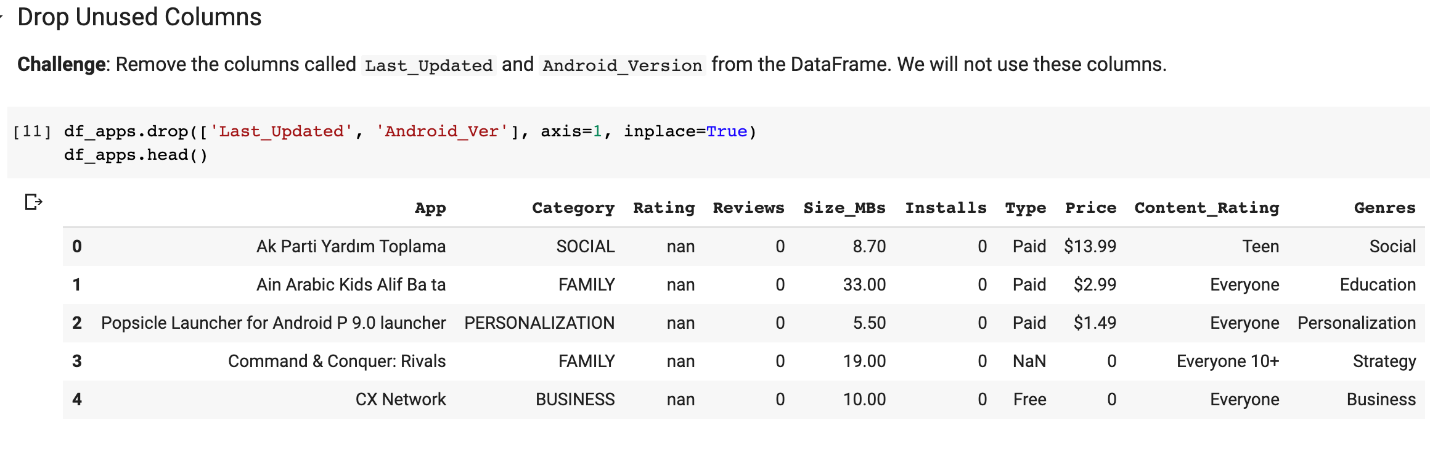
..

.

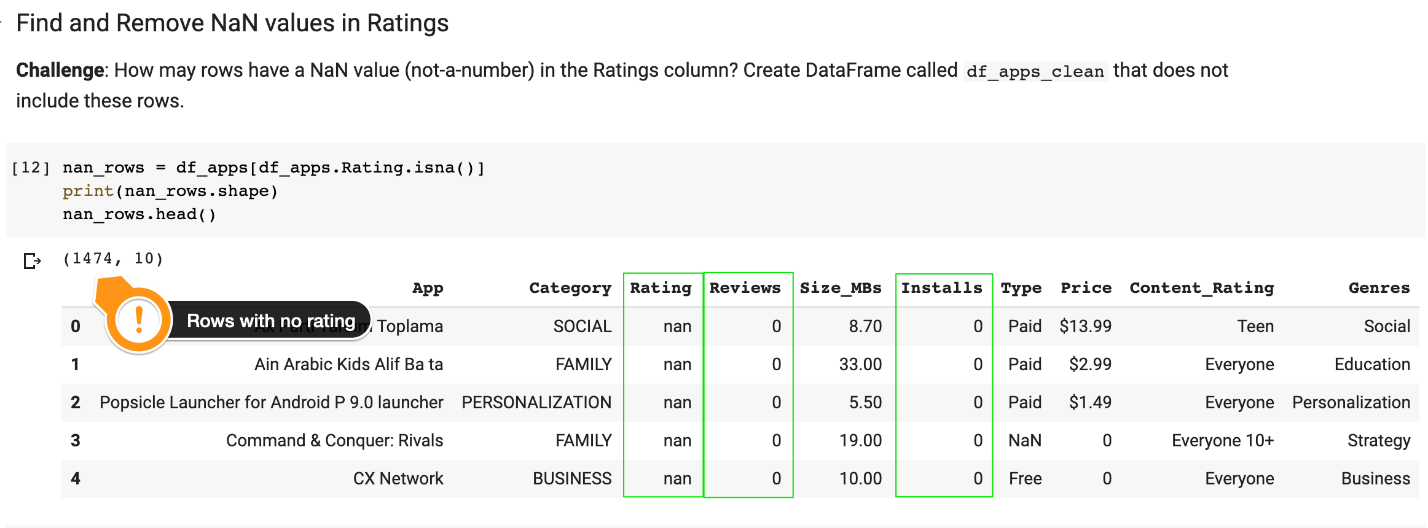
.

**Solution: Dropping Unused Columns and Removing NaN Values**

To remove the unwanted columns, we simply provide a list of the column names ['Last\_Updated', ‘Android\_Ver'] to the .drop() method. By setting axis=1 we are specifying that we want to drop certain columns.



To find and remove the rows with the NaN values we can create a subset of the DataFrame based on where .isna() evaluates to True. We see that NaN values in ratings are associated with no reviews (and no installs). That makes sense.



We can drop the NaN values with .dropna():

1. df\_apps\_clean = df\_apps.dropna()
2. df\_apps\_clean.shape

This leaves us with 9,367 entries in our DataFrame. But there may be other problems with the data too:

**Challenge**: Are there any duplicates in data? Check for duplicates using the [.duplicated()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.duplicated.html) function. How many entries can you find for the "Instagram" app? Use [.drop\_duplicates()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop_duplicates.html) to remove any duplicates from df\_apps\_clean.

.

.

..

…

..

.

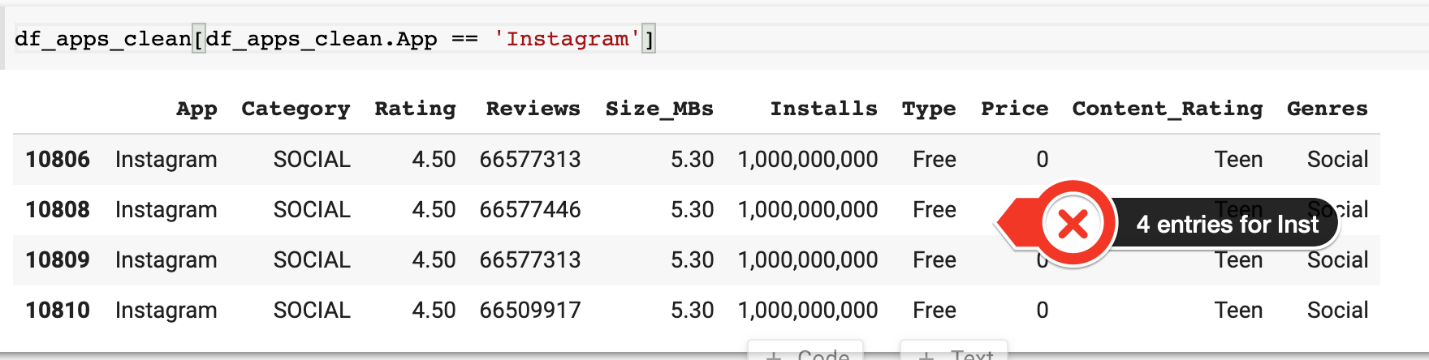
.

**Solution: Finding and Removing Duplicates**

There are indeed duplicates in the data. We can show them using the .duplicated() method, which brings up 476 rows:

1. duplicated\_rows = df\_apps\_clean[df\_apps\_clean.duplicated()]
2. print(duplicated\_rows.shape)
3. duplicated\_rows.head()

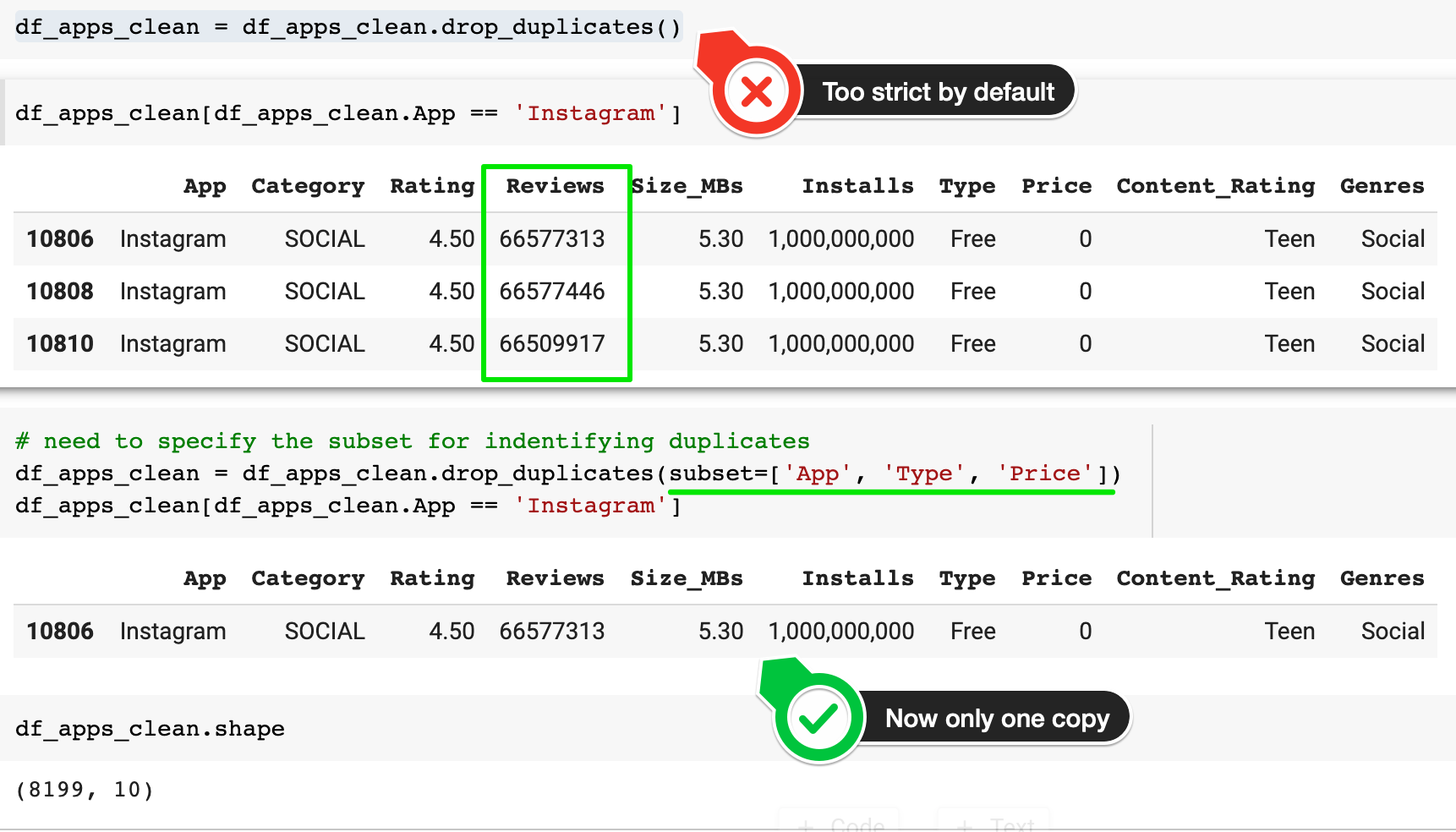
We can actually check for an individual app like ‘Instagram’ by looking up all the entries with that name in the App column.



So how do we get rid of duplicates? Can we simply call .drop\_duplicates()?

1. df\_apps\_clean = df\_apps\_clean.drop\_duplicates()

Not really. If we do this without specifying how to identify duplicates, we see that 3 copies of Instagram are retained because they have a different number of reviews. We need to provide the column names that should be used in the comparison to identify duplicates. For example:



This leaves us with 8,199 entries after removing duplicates. Huzzah! 💪

**What else should I know about the data?**

So we can see that 13 different features were originally scraped from the Google Play Store.

* Obviously, the data is just a sample out of all the Android apps. It doesn't include all Android apps of which there are millions.
* I’ll assume that the sample is representative of the App Store as a whole. This is not necessarily the case as, during the web scraping process, this sample was served up based on geographical location and user behaviour of the person who scraped it - in our case Lavanya Gupta.
* The data was compiled around 2017/2018. The pricing data reflect the price in USD Dollars at the time of scraping. (developers can offer promotions and change their app’s pricing).
* I’ve converted the app’s size to a floating-point number in MBs. If data was missing, it has been replaced by the average size for that category.
* The installs are not the exact number of installs. If an app has 245,239 installs then Google will simply report an order of magnitude like 100,000+. I’ve removed the '+' and we’ll assume the exact number of installs in that column for simplicity.

Here’s what you would see under an Android app listing if you go to a listing on the Google Play Store:

