Data type constraints

CLEANING DATA IN PYTHON

Adel Nehme
Content Developer @ DataCamp

Course outline

Diagnose dirty data

Course outline

Diagnose dirty data

Side effects of dirty data

Course outline

Diagnose dirty data

Side effects of dirty data

Clean data

Course outline

Diagnose dirty data

Side effects of dirty data

Clean data

Chapter 1 - Common data problems

Why do we need to clean data?

6. Why do we need to clean data?
To understand why we need to clean data, let's remind ourselves of the data science workflow. In a typical data science workflow, we usually access our raw data, explore and process it, develop insights using visualizations or predictive models, and finally report these insights with dashboards or reports.

Why do we need to clean data?

Why do we need to clean data?

Garbage in Garbage out

7. Why do we need to clean data?
Dirty data can appear because of duplicate values, mis-spellings, data type parsing errors and legacy systems.

Data type constraints

Datatype	Example
Text data	First name, last name, address ...
Integers	# Subscribers, # products sold ...
Decimals	Temperature, $ exchange rates ...
Binary	Is married, new customer, yes/no, ...
Dates	Order dates, ship dates ...
Categories	Marriage status, gender ...

Python data type
`str`
`int`
`float`
`bool`
`datetime`
`category`

9. Data type constraints
When working with data, there are various types that we may encounter along the way. We could be working with text data, integers, decimals, dates, zip codes, and others. Luckily, Python has specific data type objects for various data types that you're probably familiar with by now. This makes it much easier to manipulate these various data types in Python. As such, before preparing to analyze and extract insights from our data, we need to make sure our variables have the correct data types, other wise we risk compromising our analysis.

Strings to integers

```
# Import CSV file and output header
sales = pd.read_csv('sales.csv')
sales.head(2)
```

```
   SalesOrderID    Revenue    Quantity
0         43659     23153$          12
1         43660      1457$           2
```

10. Strings to integers
Let's take a look at the following example. Here's the head of a DataFrame containing revenue generated and quantity of items sold for a sales order. We want to calculate the total revenue generated by all sales orders. As you can see, the Revenue column has the dollar sign on the right hand side. A close inspection of the DataFrame column's data types using the dot-dtypes attribute returns object for the Revenue column, which is what pandas uses to store strings.

```
# Get data types of columns
sales.dtypes
```

```
SalesOrderID    int64
Revenue         object
Quantity        int64
dtype: object
```

String to integers

```
# Get DataFrame information
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31465 entries, 0 to 31464
Data columns (total 3 columns):
SalesOrderID      31465 non-null int64
Revenue           31465 non-null object
Quantity          31465 non-null int64
dtypes: int64(2), object(1)
memory usage: 737.5+ KB
```

11. String to integers
We can also check the data types as well as the number of missing values per column in a DataFrame, by using the dot-info() method.

String to integers

```python
# Print sum of all Revenue column
sales['Revenue'].sum()
```

```
'23153$1457$36865$32474$472$27510$16158$5694$6876$40487$807$6893$9153$6895$4216..
```

```python
# Remove $ from Revenue column
sales['Revenue'] = sales['Revenue'].str.strip('$')
sales['Revenue'] = sales['Revenue'].astype('int')
```

```python
# Verify that Revenue is now an integer
assert sales['Revenue'].dtype == 'int'
```

12. String to integers

We need to first remove the $ sign from the string so that pandas is able to convert the strings into numbers without error. We do this with the dot-str-dot-strip() method, while specifying the string we want to strip as an argument, which is in this case the dollar sign. Since our dollar values do not contain decimals, we then convert the Revenue column to an integer by using the dot-astype() method, specifying the desired data type as argument. Had our revenue values been decimal, we would have converted the Revenue column to float. We can make sure that the Revenue column is now an integer by using the assert statement, which takes in a condition as input, as returns nothing if that condition is met, and an error if it is not.

The assert statement

```
# This will pass
assert 1+1 == 2
```

13. The assert statement
For example, here we are testing the equality that 1+1 equals 2. Since it is the case, the assert statement returns nothing. However, when testing the equality 1+1 equals 3, we receive an assertionerror. You can test almost anything you can imagine of by using assert,

```
# This will not pass
assert 1+1 == 3
```

```
AssertionError                            Traceback (most recent call last)
        assert 1+1 == 3
AssertionError:
```

Numeric or categorical?

```
...   marriage_status    ...
...                 3    ...
...                 1    ...
...                 2    ...
```

`0` = Never married   `1` = Married   `2` = Separated   `3` = Divorced

```
df['marriage_status'].describe()
```

```
       marriage_status
...
mean               1.4
std               0.20
min               0.00
50%                1.8 ...
```

14. Numeric or categorical?
A common type of data seems numeric but actually represents categories with a finite set of possible categories. This is called categorical data. We will look more closely at categorical data in Chapter 2, but let's take a look at this example. Here we have a marriage status column, which is represented by 0 for never married, 1 for married, 2 for separated, and 3 for divorced. However it will be imported of type integer, which could lead to misleading results when trying to extract some statistical summaries.

Numeric or categorical?

```
# Convert to categorical
df["marriage_status"] = df["marriage_status"].astype('category')
df.describe()
```

```
        marriage_status
count               241
unique              4
top                 1
freq                120
```

15. Numeric or categorical?
We can solve this by using the same dot-astype() method seen earlier, but this time specifying the category data type. When applying the describe again, we see that the summary statistics are much more aligned with that of a categorical variable, discussing the number of observations, number of unique values, most frequent category instead of mean and standard deviation.

Let's practice!

CLEANING DATA IN PYTHON

Data range constraints

CLEANING DATA IN PYTHON

Adel Nehme
Content Developer @ DataCamp

Motivation

```
movies.head()
```

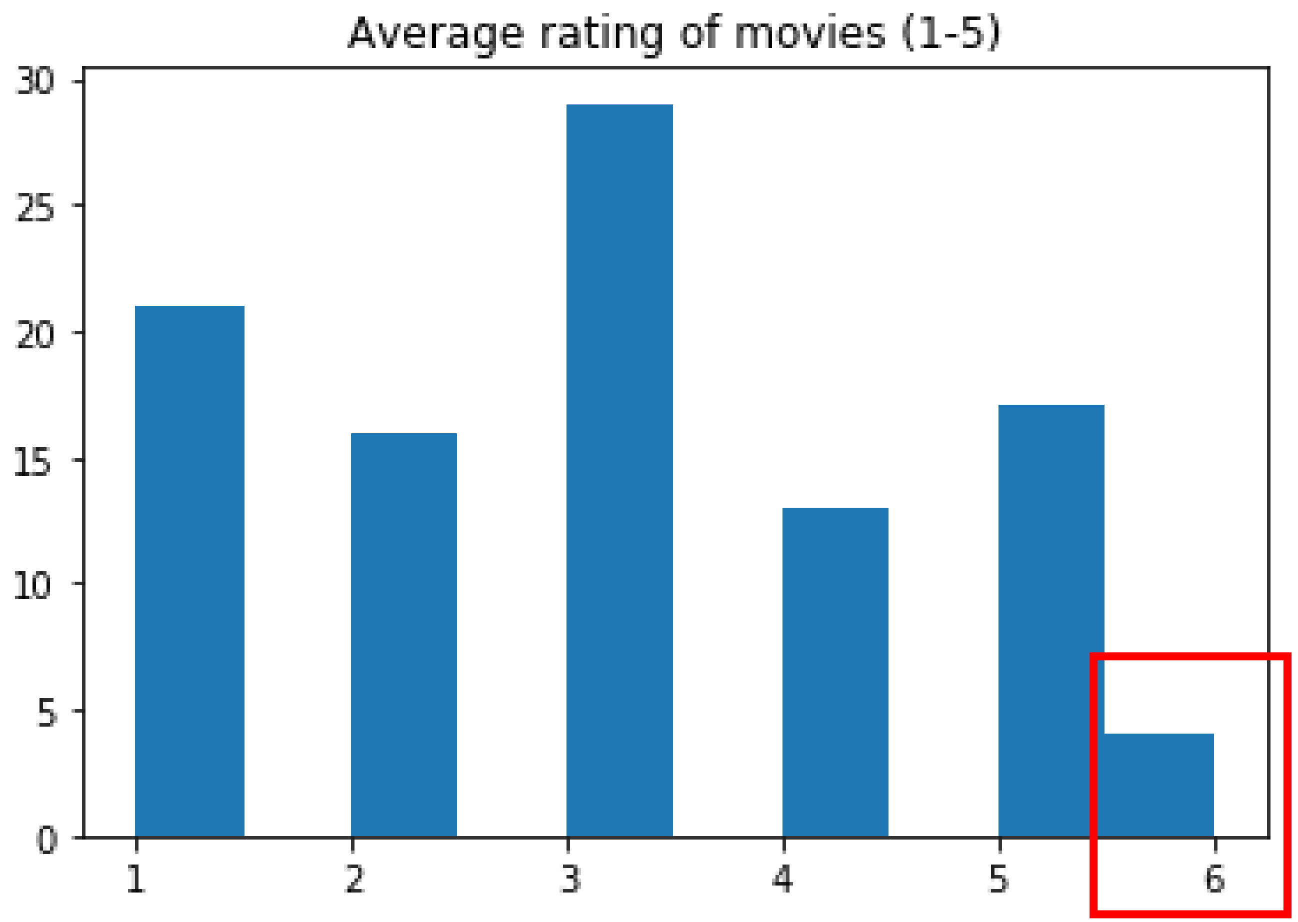
```
      movie_name     avg_rating
0     The Godfather           5
1     Frozen 2                3
2     Shrek                   4
...
```

Let's first start off with some motivation. Imagine we have a dataset of movies with their respective average rating from a streaming service. The rating can be any integer between 1 an 5.

Motivation

```python
import matplotlib.pyplot as plt
plt.hist(movies['avg_rating'])
plt.title('Average rating of movies (1-5)')
```

3. Motivation
After creating a histogram with maptlotlib, we see that there are a few movies with an average rating of 6, which is well above the allowable range. This is most likely an error in data collection or parsing, where a variable is well beyond its range and treating it is essential to have accurate analysis.



Motivation

Can future sign-ups exist?

4. Motivation
Here's another example, where we see subscription dates in the future for a service. Inherently this doesn't make any sense, as we cannot sign up for a service in the future, but these errors exist either due to technical or human error. We use the datetime package's dot-date-dot-today() function to get today's date, and we filter the dataset by any subscription date higher than today's date. **We need to pay attention to the range of our data**.

```python
# Import date time
import datetime as dt
today_date = dt.date.today()
user_signups[user_signups['subscription_date'] > dt.date.today()]
```

```
   subscription_date  user_name         ...             Country
0         01/05/2021      Marah         ...             Nauru
1         09/08/2020     Joshua         ...             Austria
2         04/01/2020      Heidi         ...             Guinea
3         11/10/2020       Rina         ...             Turkmenistan
4         11/07/2020  Christine         ...             Marshall Islands
5         07/07/2020     Ayanna         ...             Gabon
```

How to deal with out of range data?

- Dropping data
- Setting custom minimums and maximums
- Treat as missing and impute
- Setting custom value depending on business assumptions

5. How to deal with out of range data?
There's a variety of options to deal with out of range data. The simplest option is to drop the data. However, depending on the size of your out of range data, you could be losing out on essential information. As a rule of thumb, only drop data when a small proportion of your dataset is affected by out of range values, however you really need to understand your dataset before deciding to drop values. Another option would be setting custom minimums or maximums to your columns. We could also set the data to missing, and impute it, but we'll take a look at how to deal with missing data in Chapter 3. We could also, dependent on the business assumptions behind our data, **assign a custom value for any values of our data that go beyond a certain range.**

Movie example

```python
import pandas as pd
# Output Movies with rating > 5
movies[movies['avg_rating'] > 5]
```

```
         movie_name  avg_rating
23  A Beautiful Mind           6
65   La Vita e Bella           6
77            Amelie           6
```

```python
# Drop values using filtering
movies = movies[movies['avg_rating'] <= 5]
# Drop values using .drop()
movies.drop(movies[movies['avg_rating'] > 5].index, inplace = True)
# Assert results
assert movies['avg_rating'].max() <= 5
```

6. Movie example
Let's take a look at the movies example mentioned earlier. We first isolate the movies with ratings higher than 5. Now if these values are affect a small set of our data, we can drop them. We can drop them in two ways - we can either create a new filtered movies DataFrame where we only keep values of avg_rating lower or equal than to 5. Or drop the values by using the drop method. The drop method takes in as argument the row indices of movies for which the avg_rating is higher than 5. We set the inplace argument to True so that values are dropped in place and we don't have to create a new column. We can make sure this is set in place using an assert statement that checks if the maximum of avg_rating is lower or equal than to 5.

Movie example

```
# Convert avg_rating > 5 to 5
movies.loc[movies['avg_rating'] > 5, 'avg_rating'] = 5
```

```
# Assert statement
assert movies['avg_rating'].max() <= 5
```

Remember, no output means it passed

7. Movie example

Depending on the assumptions behind our data, we can also change the out of range values to a hard limit. For example, here we're setting any value of the avg_rating column in to 5 if it goes beyond it. We can do this using the dot-loc method, which returns all cells that fit a custom row and column index. It takes as first argument the row index, or here all instances of avg_rating above 5 and as second argument the column index, which is here the avg_rating column. Again, we can make sure that this change was done using an assert statement.

Date range example

8. Date range example

Let's take another at the date range example mentioned earlier, where we had subscriptions happening in the future. We first look at the datatypes of the column with the dot-dtypes attribute. We can confirm that the subscription_date column is an object and not a datetime object. Datetime objects allow much easier manipulation of date data, so let's convert it to that. We do so with the to_datetime function from pandas, which takes in as argument the column we want to convert. We can then test the data type conversion by asserting that the subscription date's column is equal to datetime64[ns], which is how the data type is represented in pandas.

```
import datetime as dt
import pandas as pd
# Output data types
user_signups.dtypes
```

```
subscription_date     object
user_name             object
Country               object
dtype: object
```

```
# Convert to DateTime
user_signups['subscription_date'] = pd.to_datetime(user_signups['subscription_date'])
```

```
# Assert that conversion happened
assert user_signups['subscription_date'].dtype == 'datetime64[ns]'
```

Date range example

9. Date range example
Now that the column is in datetime, we can treat it in a variety of ways. We first create a today_date variable using the datetime function date.today, which allows us to store today's date. We can then either drop the rows with exceeding dates similar to how we did in the average rating example, or replace exceeding values with today's date. In both cases we can use the assert statement to verify our treatment went well, by comparing the maximum value in the subscription_date column. However, make sure to chain it with the dot-date() method to return a datetime object instead of a timestamp.

```
today_date = dt.date.today()
```

Drop the data

```
# Drop values using filtering
user_signups = user_signups[user_signups['subscription_date'] < today_date]
# Drop values using .drop()
user_signups.drop(user_signups[user_signups['subscription_date'] > today_date].index, inplace = True)
```

Hardcode dates with upper limit

```
# Drop values using filtering
user_signups.loc[user_signups['subscription_date'] > today_date, 'subscription_date'] = today_date
# Assert is true
assert user_signups.subscription_date.max().date() <= today_date
```

Let's practice!

CLEANING DATA IN PYTHON

Uniqueness constraints

CLEANING DATA IN PYTHON

Adel Nehme
Content Developer @ DataCamp

What are duplicate values?

All columns have the same values

first_name	last_name	address	height	weight
Justin	Saddlemyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg

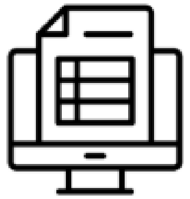
What are duplicate values?

Most columns have the same values

first_name	last_name	address	height	weight
Justin	Saddlemyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemyer	Boulevard du Jardin Botanique 3, Bruxelles	***194 cm***	87 kg

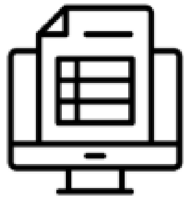
3. What are duplicate values?
In this one, there are duplicate values for all columns except the height column -- which leads us to think it's more likely a data entry error than an actual other person.

Why do they happen?



Data Entry & Human Error

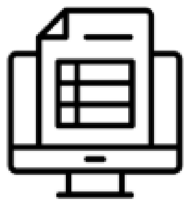
Why do they happen?



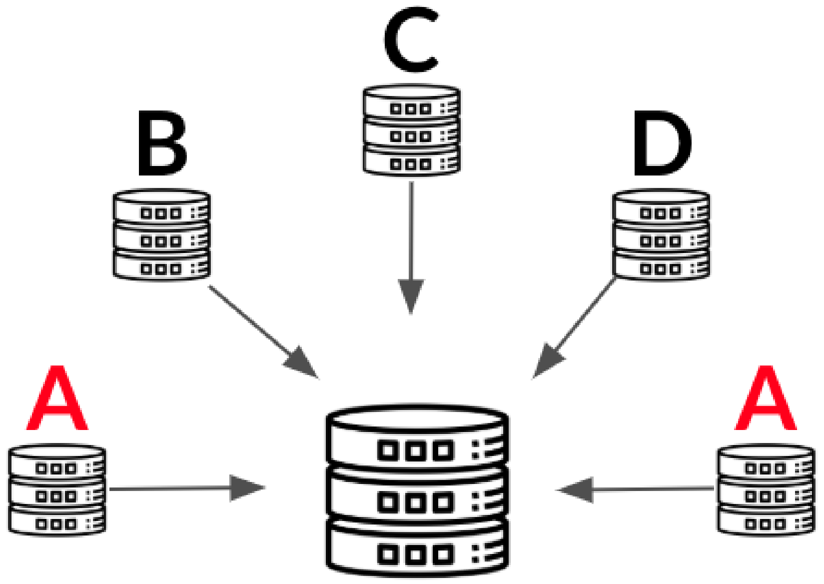
Data Entry &
Human Error

Bugs and design
errors

Why do they happen?



Data Entry & Human Error



Join or merge Errors

Bugs and design errors

6. Why do they happen?
However they oftenmost arise from the necessary act of joining and consolidating data from various resources, which could retain duplicate values.

How to find duplicate values?

```
# Print the header
height_weight.head()
```

```
  first_name last_name                         address  height  weight
0       Lane     Reese               534-1559 Nam St.     181      64
1       Ivor    Pierce              102-3364 Non Road     168      66
2      Roary    Gibson    P.O. Box 344, 7785 Nisi Ave     191      99
3    Shannon    Little  691-2550 Consectetuer Street      185      65
4      Abdul       Fry                4565 Risus St.      169      65
```

How to find duplicate values?

```python
# Get duplicates across all columns
duplicates = height_weight.duplicated()
print(duplicates)
```

```
1       False
...     ....
22      True
23      False
...     ...
```

8. How to find duplicate values?
We can find duplicates in a DataFrame by using the dot-duplicated() method. It returns a Series of boolean values that are True for duplicate values, and False for non-duplicated values.

How to find duplicate values?

```
# Get duplicate rows
duplicates = height_weight.duplicated()
height_weight[duplicates]
```

```
    first_name last_name                                 address  height  weight
100       Mary     Colon                             4674 Ut Rd.     179      75
101       Ivor    Pierce                       102-3364 Non Road     168      88
102       Cole    Palmer                         8366 At, Street     178      91
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     196      83
```

9. How to find duplicate values?
We can see exactly which rows are affected by using brackets as such. However, using dot-duplicated() without playing around with the arguments of the method can lead to misleading results, as all the columns are required to have duplicate values by default, with all duplicate values being marked as True except for the first occurrence. This limits our ability to properly diagnose what type of duplication we have, and how to effectively treat it.

How to find duplicate rows?

The `.duplicated()` method

`subset` : List of column names to check for duplication.

`keep` : Whether to keep **first** (`'first'`), **last** (`'last'`) or **all** (`False`) duplicate values.

```
# Column names to check for duplication
column_names = ['first_name','last_name','address']
duplicates = height_weight.duplicated(subset = column_names, keep = False)
```

10. How to find duplicate rows?
To properly calibrate how we go about finding duplicates, we will use 2 arguments from the dot-duplicated() method. The subset argument lets us set a list of column names to check for duplication. For example, it allows us to find duplicates for the first and last name columns only. The keep argument lets us keep the first occurrence of a duplicate value by setting it to the string first, the last occurrence of a duplicate value by setting it the string last, or keep all occurrences of duplicate values by setting it to False. In this example, we're checking for duplicates across the first name, last name, and address variables, and we're choosing to keep all duplicates.

How to find duplicate rows?

```
# Output duplicate values
height_weight[duplicates]
```

```
    first_name last_name                                 address  height  weight
1         Ivor    Pierce                      102-3364 Non Road     168      66
22        Cole    Palmer                        8366 At, Street     178      91
28     Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     195      83
37        Mary     Colon                            4674 Ut Rd.     179      75
100       Mary     Colon                            4674 Ut Rd.     179      75
101       Ivor    Pierce                      102-3364 Non Road     168      88
102       Cole    Palmer                        8366 At, Street     178      91
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     196      83
```

How to find duplicate rows?

```
# Output duplicate values
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name	address	height	weight
22	Cole	Palmer	8366 At, Street	178	91
102	Cole	Palmer	8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83
1	Ivor	Pierce	102-3364 Non Road	168	66
101	Ivor	Pierce	102-3364 Non Road	168	88
37	Mary	Colon	4674 Ut Rd.	179	75
100	Mary	Colon	4674 Ut Rd.	179	75

12. How to find duplicate rows?
We sort the duplicate rows using the dot-sort_values method, choosing first_name to sort by.

How to find duplicate rows?

```
# Output duplicate values
height_weight[duplicates].sort_values(by = 'first_name')
```

```
    first_name last_name                             address  height  weight
22        Cole    Palmer                     8366 At, Street     178      91
102       Cole    Palmer                     8366 At, Street     178      91
28     Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     195      83
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     196      83
1         Ivor    Pierce                   102-3364 Non Road     168      66
101       Ivor    Pierce                   102-3364 Non Road     168      88
37        Mary     Colon                       4674 Ut Rd.     179      75
100       Mary     Colon                       4674 Ut Rd.     179      75
```

13. How to find duplicate rows?
We find that there are four sets of duplicated rows, the first 2 being complete duplicates of each other across all columns, highlighted here in red.

How to find duplicate rows?

```
# Output duplicate values
height_weight[duplicates].sort_values(by = 'first_name')
```

```
    first_name last_name                             address  height  weight
22        Cole    Palmer                     8366 At, Street     178      91
102       Cole    Palmer                     8366 At, Street     178      91
28     Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     195      83
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     196      83
1         Ivor    Pierce                  102-3364 Non Road     168      66
101       Ivor    Pierce                  102-3364 Non Road     168      88
37        Mary     Colon                       4674 Ut Rd.     179      75
100       Mary     Colon                       4674 Ut Rd.     179      75
```

14. How to find duplicate rows?
The other 2 being incomplete duplicates of each other highlighted here in blue with discrepancies across height and weight respectively.

How to treat duplicate values?

```
# Output duplicate values
height_weight[duplicates].sort_values(by = 'first_name')
```

```
    first_name last_name                          address  height  weight
22        Cole    Palmer                  8366 At, Street     178      91
102       Cole    Palmer                  8366 At, Street     178      91
28     Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.  195      83
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.  196      83
1         Ivor    Pierce              102-3364 Non Road     168      66
101       Ivor    Pierce              102-3364 Non Road     168      88
37        Mary     Colon                    4674 Ut Rd.     179      75
100       Mary     Colon                    4674 Ut Rd.     179      75
```

15. How to treat duplicate values?
The complete duplicates can be treated easily. All that is required is to keep one of them only and discard the others.

How to treat duplicate values?

The `.drop_duplicates()` method

`subset` : List of column names to check for duplication.

`keep` : Whether to keep **first** (`'first'`), **last** (`'last'`) or **all** (`False`) duplicate values.

`inplace` : Drop duplicated rows directly inside DataFrame without creating new object (`True`).

```
# Drop duplicates
height_weight.drop_duplicates(inplace = True)
```

16. How to treat duplicate values?
This can be done with the dot-drop_duplicates() method, which also takes in the same subset and keep arguments as in the dot-duplicated() method, as well as the inplace argument which drops the duplicated values directly inside the height_weight DataFrame. Here we are dropping complete duplicates only, so it's not necessary nor advisable to set a subset, and since the keep argument takes in first as default, we can keep it as such. Note that we can also set it as last, but not as False as it would keep all duplicates.

How to treat duplicate values?

```python
# Output duplicate values
column_names = ['first_name','last_name','address']
duplicates = height_weight.duplicated(subset = column_names, keep = False)
height_weight[duplicates].sort_values(by = 'first_name')
```

```
    first_name last_name                          address  height  weight
28     Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     195      83
103    Desirae   Shannon  P.O. Box 643, 5251 Consectetuer, Rd.     196      83
1         Ivor    Pierce                    102-3364 Non Road     168      66
101       Ivor    Pierce                    102-3364 Non Road     168      88
```

17. How to treat duplicate values?
This leaves us with the other 2 sets of duplicates discussed earlier, which are the same for first_name, last_name and address, but contain discrepancies in height and weight. Apart from dropping rows with really small discrepancies, we can use a statistical measure to combine each set of duplicated values.

How to treat duplicate values?

```
# Output duplicate values
column_names = ['first_name','last_name','address']
duplicates = height_weight.duplicated(subset = column_names, keep = False)
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name	address	height	weight
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83
1	Ivor	Pierce	102-3364 Non Road	168	66
101	Ivor	Pierce	102-3364 Non Road	168	88

18. How to treat duplicate values?
For example, we can combine these two rows into one by computing the average mean between them, or the maximum, or other statistical measures, this is **highly dependent on a common sense understanding of our data,** and what type of data we have.

How to treat duplicate values?

The `.groupby()` and `.agg()` methods

```
# Group by column names and produce statistical summaries
column_names = ['first_name','last_name','address']
summaries = {'height': 'max', 'weight': 'mean'}
height_weight = height_weight.groupby(by = column_names).agg(summaries).reset_index()
# Make sure aggregation is done
duplicates = height_weight.duplicated(subset = column_names, keep = False)
height_weight[duplicates].sort_values(by = 'first_name')
```

```
first_name    last_name    address    height    weight
```

19. How to treat duplicate values?

We can do this easily using the groupby method, which when chained with the agg method, lets you group by a set of common columns and return statistical values for specific columns when the aggregation is being performed. For example here, we created a dictionary called summaries, which instructs groupby to return the maximum of duplicated rows for the height column, and the mean duplicated rows for the weight column. We then group height_weight by the column names defined earlier, and chained it with the agg method, which takes in the summaries dictionary we created. We chain this entire line with the dot-reset_index() method, so that we can have numbered indices in the final output. We can verify that there are no more duplicate values by running the duplicated method again, and use brackets to output duplicate rows.

Let's practice!

CLEANING DATA IN PYTHON