

# tu13\_DataWrangling

February 28, 2023

## 1 Data Wrangling

Data wrangling generally refers to the process of getting a data set ready for analysis. Why would we need to do that?

Real-world data can be messy. Data sets are recorded and assembled by humans, and humans make mistakes. A single data set might be created and updated by multiple people who may decide to do things in slightly different ways. On a spreadsheet, one person might decide to leave cells with missing data blank, another might enter “NaN”, while a third may enter “missing”. If the data has many many rows, one person might decide to repeat the column headers partway down so they don’t have to scroll up to see them. Any of these things mean that the data set cannot be analyzed “as is” and wrangling will be required.

Even in a tightly controlled laboratory setting in which data are collected via computer and automatically written out to data files, some data wrangling might be required. There might be a separate data file for each subject or experimental session, meaning that these separate files will have to be combined into a single data set before analysis.

Our main wrangling tool is pandas, so we can go ahead and import it.

```
[43]: import pandas as pd
```

### 1.1 Loading

For our wrangling practice today, we’ll look at a data set containing various measurements on breast cancer patients. The file is called `breast_cancer_data.csv`, and you should place it in the “data” folder you should already have in the same directory as this notebook.

Let’s import it as a pandas dataframe.

```
[44]: bcd = pd.read_csv('./data/breast_cancer_data.csv')
bcd
```

```
[44]:
```

|    | patient_id | clump_thickness | cell_size_uniformity | cell_shape_uniformity | \ |
|----|------------|-----------------|----------------------|-----------------------|---|
| 0  | 1000025    | 5.0             | 1.0                  |                       | 1 |
| 1  | 1002945    | 5.0             | 4.0                  |                       | 4 |
| 2  | 1015425    | 3.0             | 1.0                  |                       | 1 |
| 3  | 1016277    | 6.0             | 8.0                  |                       | 8 |
| 4  | 1017023    | 4.0             | 1.0                  |                       | 1 |
| .. | ...        | ...             | ...                  | ...                   |   |

|     |        |     |      |    |
|-----|--------|-----|------|----|
| 694 | 776715 | 3.0 | 1.0  | 1  |
| 695 | 841769 | 2.0 | 1.0  | 1  |
| 696 | 888820 | 5.0 | 10.0 | 10 |
| 697 | 897471 | 4.0 | 8.0  | 6  |
| 698 | 897471 | 4.0 | 8.0  | 8  |

|     | marginal_adhesion | single_ep_cell_size | bare_nuclei | bland_chromatin | \ |
|-----|-------------------|---------------------|-------------|-----------------|---|
| 0   | 1                 | 2                   | 1           | 3.0             |   |
| 1   | 5                 | 7                   | 10          | 3.0             |   |
| 2   | 1                 | 2                   | 2           | 3.0             |   |
| 3   | 1                 | 3                   | 4           | 3.0             |   |
| 4   | 3                 | 2                   | 1           | 3.0             |   |
| ..  | ...               | ...                 | ...         | ...             |   |
| 694 | 1                 | 3                   | 2           | 1.0             |   |
| 695 | 1                 | 2                   | 1           | 1.0             |   |
| 696 | 3                 | 7                   | 3           | 8.0             |   |
| 697 | 4                 | 3                   | 4           | 10.0            |   |
| 698 | 5                 | 4                   | 5           | 10.0            |   |

|     | normal_nucleoli | mitoses | class     | doctor_name |
|-----|-----------------|---------|-----------|-------------|
| 0   | 1.0             | 1       | benign    | Dr. Doe     |
| 1   | 2.0             | 1       | benign    | Dr. Smith   |
| 2   | 1.0             | 1       | benign    | Dr. Lee     |
| 3   | 7.0             | 1       | benign    | Dr. Smith   |
| 4   | 1.0             | 1       | benign    | Dr. Wong    |
| ..  | ...             | ...     | ...       | ...         |
| 694 | 1.0             | 1       | benign    | Dr. Lee     |
| 695 | 1.0             | 1       | benign    | Dr. Smith   |
| 696 | 10.0            | 2       | malignant | Dr. Lee     |
| 697 | 6.0             | 1       | malignant | Dr. Lee     |
| 698 | 4.0             | 1       | malignant | Dr. Wong    |

[699 rows x 12 columns]

Before we do any actual wrangling, let's get familiar with the data frame in its current form.

## 1.2 Exploring the Data Frame

We can explore the data frame by looking at it's attributes, such as its shape, column names, and data types:

```
[45]: bcd.columns
```

```
[45]: Index(['patient_id', 'clump_thickness', 'cell_size_uniformity',
            'cell_shape_uniformity', 'marginal_adhesion', 'single_ep_cell_size',
            'bare_nuclei', 'bland_chromatin', 'normal_nucleoli', 'mitoses', 'class',
            'doctor_name'],
            dtype='object')
```

---

Use the cells below to get the shape and data types (**dtypes**) of our data frame.

```
[46]: bcd.shape
```

```
[46]: (699, 12)
```

```
[47]: bcd.dtypes
```

```
[47]: patient_id          int64
      clump_thickness    float64
      cell_size_uniformity float64
      cell_shape_uniformity int64
      marginal_adhesion    int64
      single_ep_cell_size  int64
      bare_nuclei         object
      bland_chromatin      float64
      normal_nucleoli      float64
      mitoses             int64
      class              object
      doctor_name         object
      dtype: object
```

---

In the cell below, use the **describe()** method to get a summary of the numerical columns.

```
[48]: bcd.describe
```

```
[48]: <bound method NDFrame.describe of      patient_id  clump_thickness
      cell_size_uniformity  cell_shape_uniformity  \
0      1000025          5.0          1.0          1
1      1002945          5.0          4.0          4
2      1015425          3.0          1.0          1
3      1016277          6.0          8.0          8
4      1017023          4.0          1.0          1
..      ...          ...          ...          ...
694      776715          3.0          1.0          1
695      841769          2.0          1.0          1
696      888820          5.0         10.0         10
697      897471          4.0          8.0          6
698      897471          4.0          8.0          8

      marginal_adhesion  single_ep_cell_size  bare_nuclei  bland_chromatin  \
0              1              2              1              3.0
1              5              7             10              3.0
2              1              2              2              3.0
3              1              3              4              3.0
```

```

4          3          2          1          3.0
..      ...          ...      ...      ...
694          1          3          2          1.0
695          1          2          1          1.0
696          3          7          3          8.0
697          4          3          4          10.0
698          5          4          5          10.0

```

```

      normal_nucleoli  mitoses      class  doctor_name
0          1.0          1      benign    Dr. Doe
1          2.0          1      benign    Dr. Smith
2          1.0          1      benign    Dr. Lee
3          7.0          1      benign    Dr. Smith
4          1.0          1      benign    Dr. Wong
..      ...      ...      ...      ...
694          1.0          1      benign    Dr. Lee
695          1.0          1      benign    Dr. Smith
696          10.0          2      malignant    Dr. Lee
697          6.0          1      malignant    Dr. Lee
698          4.0          1      malignant    Dr. Wong

```

```
[699 rows x 12 columns]>
```

---

### 1.3 Modifying a text column

We'll often want to “tune up” columns that contain text. We might encounter, for example, a column containing full names that we need to break up into separate columns for the first and last names.

---

Let's look at the column for the doctors' names. Use the cell below to take a peek.

```
[49]: bcd['doctor_name']
```

```

[49]: 0      Dr. Doe
      1      Dr. Smith
      2      Dr. Lee
      3      Dr. Smith
      4      Dr. Wong
      ...
      694      Dr. Lee
      695      Dr. Smith
      696      Dr. Lee
      697      Dr. Lee
      698      Dr. Wong
      Name: doctor_name, Length: 699, dtype: object

```

---

The doctors' name data are redundant; each one has a “Dr. ” in front of the actual name, but we already know these are doctors by the column name. Further, the entries have white space in them, which can cause us problems down the road. So let's modify this column so it only contains the surnames of the doctors.

One great thing about pandas is that it has versions of many of Python's string methods that operate *element-wise on an entire column of strings*. Here, we want to separate the “Dr. ” from the actual name, which is exactly what Python's `str.split()` function does. So chances are, pandas has a version of this function that operates element-wise on data frames.

---

**String Splitting Review:** Let's briefly remind ourselves of splitting up Python strings and extracting bits of them.

```
[50]: # Here's a string of the form: surname, first initial.
      myStr = 'SirString, A.'
      print(myStr)
```

SirString, A.

Let's say we wanted to get the surname. We could split this string into a Python list at the white space like this:

```
[51]: spltStr = myStr.split()    # split() defaults to splitting at white space
      print(spltStr)
```

```
['SirString,', 'A.']
```

We now have a list in which the items contain the text on either side of the split. This is close to what we want: the first entry in the list has the surname, but it also has an unwanted comma.

Let's split the string at the comma instead:

```
[52]: spltStr = myStr.split(',')  # tell Python to split at commas
      print(spltStr)
```

```
['SirString', ' A.']
```

Now we have isolated the last name, and we can fetch it by indexing:

```
[53]: surname = spltStr[0]
      print(surname)
```

SirString

---

In the cell below, see if you can extract the surname from `myStr` in one line of code:

```
[149]: myStr.split(',')[0]
```

```
[149]: 'SirString'
```

---

Alright, time to replace the `bcd['doctor_name']` column values with just the doctors' last names.

We could do this in one step, but let's break it out for clarity. First, let's copy the name column out into a new series.

```
[54]: dr_names = bcd['doctor_name']
      dr_names
```

```
[54]: 0      Dr. Doe
      1      Dr. Smith
      2      Dr. Lee
      3      Dr. Smith
      4      Dr. Wong
      ...
     694      Dr. Lee
     695      Dr. Smith
     696      Dr. Lee
     697      Dr. Lee
     698      Dr. Wong
      Name: doctor_name, Length: 699, dtype: object
```

---

**Note:** pandas objects behave like ordinary Python objects. So, strictly speaking, we have not created a new object (pandas Series), rather, *we have created a new label that refers to the “doctor\_name” column of bcd.*

In the cell below, use the `id()` function to compare the object IDs of `dr_names` and the corresponding column of `bcd`.

```
[55]: id(bcd['doctor_name'])
```

```
[55]: 140153815318976
```

```
[56]: id(dr_names)
```

```
[56]: 140153815318976
```

---

Now let's split all the names in the `doctor_name` column at the whitespace by using pandas `DataFrame.str.split()` function.

```
[57]: split_dr_names = dr_names.str.split()
      split_dr_names
```

```
[57]: 0      [Dr., Doe]
      1      [Dr., Smith]
      2      [Dr., Lee]
      3      [Dr., Smith]
      4      [Dr., Wong]
      ...
      694     [Dr., Lee]
      695     [Dr., Smith]
      696     [Dr., Lee]
      697     [Dr., Lee]
      698     [Dr., Wong]
      Name: doctor_name, Length: 699, dtype: object
```

`DataFrame.str.split()`, however, *does* create a new object.

---

Use the cell below to confirm that the `split()` spawned a new object.

```
[58]: id(split_dr_names)
```

```
[58]: 140153815348128
```

```
[59]: id(dr_names)
```

```
[59]: 140153815318976
```

---

Now we have a column of lists, each with two elements. The first element of each list is the “Dr. ” bit, and the second consists of the surnames we want.

We can get these by using pandas string indexing, `Series.str[index]`.

```
[60]: surnames = split_dr_names.str[1]
      surnames
```

```
[60]: 0      Doe
      1     Smith
      2      Lee
      3     Smith
      4      Wong
      ...
      694     Lee
      695     Smith
      696     Lee
      697     Lee
      698     Wong
      Name: doctor_name, Length: 699, dtype: object
```

Note that, like the splitting, the string indexing worked on the entire **Series** automatically.

Now we can change the column in our main data frame, **bcd**.

```
[61]: bcd['doctor_name'] = surnames
```

```
[62]: bcd['doctor_name']
```

```
[62]: 0      Doe
      1    Smith
      2      Lee
      3    Smith
      4     Wong
      ...
     694    Lee
     695   Smith
     696    Lee
     697    Lee
     698    Wong
      Name: doctor_name, Length: 699, dtype: object
```

Success!

## 1.4 Converting a column type (and other aggravations)

Let's look at those data types again.

```
[26]: bcd.dtypes
```

```
[26]: patient_id          int64
      clump_thickness    float64
      cell_size_uniformity float64
      cell_shape_uniformity int64
      marginal_adhesion   int64
      single_ep_cell_size  int64
      bare_nuclei         object
      bland_chromatin     float64
      normal_nucleoli     float64
      mitoses             int64
      class              object
      doctor_name        object
      dtype: object
```

Notice that “class” and “doctor\_name” are of dtype “object”, which refers to a general purpose column type, and is how pandas imports text columns by default. Most of the others are numeric (integers or floats), except for “bare\_nuclei”.



In the cell below, take a quick glance at ‘bcd’ again, and see if the “bare\_nuclei” column should be a different data type that, say “marginal\_adhesion”.

[27]: bcd

```
[27]:      patient_id  clump_thickness  cell_size_uniformity  cell_shape_uniformity  \
0      1000025          5.0          1.0          1
1      1002945          5.0          4.0          4
2      1015425          3.0          1.0          1
3      1016277          6.0          8.0          8
4      1017023          4.0          1.0          1
..      ...
694     776715          3.0          1.0          1
695     841769          2.0          1.0          1
696     888820          5.0         10.0         10
697     897471          4.0          8.0          6
698     897471          4.0          8.0          8
```

```
      marginal_adhesion  single_ep_cell_size  bare_nuclei  bland_chromatin  \
0              1              2              1              3.0
1              5              7             10              3.0
2              1              2              2              3.0
3              1              3              4              3.0
4              3              2              1              3.0
..      ...
694              1              3              2              1.0
695              1              2              1              1.0
696              3              7              3              8.0
697              4              3              4             10.0
698              5              4              5             10.0
```

```
      normal_nucleoli  mitoses      class  doctor_name
0              1.0          1    benign      Doe
1              2.0          1    benign      Smith
2              1.0          1    benign      Lee
3              7.0          1    benign      Smith
4              1.0          1    benign      Wong
..      ...
694              1.0          1    benign      Lee
695              1.0          1    benign      Smith
696             10.0          2  malignant      Lee
697              6.0          1  malignant      Lee
698              4.0          1  malignant      Wong
```

[699 rows x 12 columns]

It looks like “bare\_nuclei” was intended to be a numeric column, so let’s try and convert it using the `DataFrame.astype()` converter method.

```
[28]: bcd['bare_nuclei'] = bcd['bare_nuclei'].astype('int64')
```

```
-----
ValueError                                Traceback (most recent call last)
/var/folders/zc/6v283x0929j5f38j6cvlwbwr0000gn/T/ipykernel_713/457653605.py in _
    <module>
----> 1 bcd['bare_nuclei'] = bcd['bare_nuclei'].astype('int64')

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py in _
    <astype(self, dtype, copy, errors)
    5910         else:
    5911             # else, only a single dtype is given
-> 5912         new_data = self._mgr.astype(dtype=dtype, copy=copy, _
    <errors=errors)
    5913         return self._constructor(new_data).__finalize__(self, _
    <method="astype")
    5914

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/managers.py in _
    <astype(self, dtype, copy, errors)
    417
    418     def astype(self: T, dtype, copy: bool = False, errors: str = _
    <"raise") -> T:
-> 419         return self.apply("astype", dtype=dtype, copy=copy, _
    <errors=errors)
    420
    421     def convert(

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/managers.py in _
    <apply(self, f, align_keys, ignore_failures, **kwargs)
    302         applied = b.apply(f, **kwargs)
    303         else:
-> 304         applied = getattr(b, f)(**kwargs)
    305         except (TypeError, NotImplementedError):
    306             if not ignore_failures:

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/internals/blocks.py in _
    <astype(self, dtype, copy, errors)
    578         values = self.values
    579
-> 580         new_values = astype_array_safe(values, dtype, copy=copy, _
    <errors=errors)
    581
    582         new_values = maybe_coerce_values(new_values)
```

```

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/dtypes/cast.py in
↳ astype_array_safe(values, dtype, copy, errors)
    1290
    1291     try:
-> 1292         new_values = astype_array(values, dtype, copy=copy)
    1293     except (ValueError, TypeError):
    1294         # e.g. astype_nansafe can fail on object-dtype of strings

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/dtypes/cast.py in
↳ astype_array(values, dtype, copy)
    1235
    1236     else:
-> 1237         values = astype_nansafe(values, dtype, copy=copy)
    1238
    1239     # in pandas we don't store numpy str dtypes, so convert to object

/opt/anaconda3/lib/python3.9/site-packages/pandas/core/dtypes/cast.py in
↳ astype_nansafe(arr, dtype, copy, skipna)
    1152     # work around NumPy brokenness, #1987
    1153     if np.issubdtype(dtype.type, np.integer):
-> 1154         return lib.astype_intsafe(arr, dtype)
    1155
    1156     # if we have a datetime/timedelta array of objects

/opt/anaconda3/lib/python3.9/site-packages/pandas/_libs/lib.pyx in pandas._libs
↳ lib.astype_intsafe()

ValueError: invalid literal for int() with base 10: '?'

```

And, argh, we get an error! If we look at the bottom of the error message, it seems that the error involves question marks (“?”) in the data, which would also explain why this column imported as text rather than numbers in the first place.

Let’s check.

---

In the cell below, use logical indexing to show the rows of `bcd` in which `bcd[bare_nuclei]` contains a question mark.

```
[66]: bcd[bcd['bare_nuclei']=='?']
```

```
[66]:
```

|     | patient_id | clump_thickness | cell_size_uniformity | cell_shape_uniformity | \ |
|-----|------------|-----------------|----------------------|-----------------------|---|
| 23  | 1057013    | 8.0             | 4.0                  | 5                     |   |
| 40  | 1096800    | 6.0             | 6.0                  | 6                     |   |
| 139 | 1183246    | 1.0             | 1.0                  | 1                     |   |
| 145 | 1184840    | 1.0             | 1.0                  | 3                     |   |
| 158 | 1193683    | 1.0             | 1.0                  | 2                     |   |

|     |         |     |     |   |
|-----|---------|-----|-----|---|
| 164 | 1197510 | 5.0 | 1.0 | 1 |
| 235 | 1241232 | 3.0 | 1.0 | 4 |
| 249 | 169356  | 3.0 | 1.0 | 1 |
| 275 | 432809  | 3.0 | 1.0 | 3 |
| 292 | 563649  | 8.0 | 8.0 | 8 |
| 294 | 606140  | 1.0 | 1.0 | 1 |
| 297 | 61634   | 5.0 | 4.0 | 3 |
| 315 | 704168  | 4.0 | 6.0 | 5 |
| 321 | 733639  | 3.0 | 1.0 | 1 |
| 411 | 1238464 | 1.0 | 1.0 | 1 |
| 617 | 1057067 | 1.0 | 1.0 | 1 |

|     | marginal_adhesion | single_ep_cell_size | bare_nuclei | bland_chromatin | \ |
|-----|-------------------|---------------------|-------------|-----------------|---|
| 23  | 1                 | 2                   | ?           | 7.0             |   |
| 40  | 9                 | 6                   | ?           | 7.0             |   |
| 139 | 1                 | 1                   | ?           | 2.0             |   |
| 145 | 1                 | 2                   | ?           | 2.0             |   |
| 158 | 1                 | 3                   | ?           | 1.0             |   |
| 164 | 1                 | 2                   | ?           | 3.0             |   |
| 235 | 1                 | 2                   | ?           | 3.0             |   |
| 249 | 1                 | 2                   | ?           | 3.0             |   |
| 275 | 1                 | 2                   | ?           | 2.0             |   |
| 292 | 1                 | 2                   | ?           | 6.0             |   |
| 294 | 1                 | 2                   | ?           | 2.0             |   |
| 297 | 1                 | 2                   | ?           | 2.0             |   |
| 315 | 6                 | 7                   | ?           | 4.0             |   |
| 321 | 1                 | 2                   | ?           | 3.0             |   |
| 411 | 1                 | 1                   | ?           | 2.0             |   |
| 617 | 1                 | 1                   | ?           | 1.0             |   |

|     | normal_nucleoli | mitoses | class     | doctor_name |
|-----|-----------------|---------|-----------|-------------|
| 23  | 3.0             | 1       | malignant | Smith       |
| 40  | 8.0             | 1       | benign    | Wong        |
| 139 | 1.0             | 1       | benign    | Lee         |
| 145 | 1.0             | 1       | benign    | Wong        |
| 158 | 1.0             | 1       | benign    | Doe         |
| 164 | 1.0             | 1       | benign    | Smith       |
| 235 | 1.0             | 1       | benign    | Doe         |
| 249 | 1.0             | 1       | benign    | Wong        |
| 275 | 1.0             | 1       | benign    | Lee         |
| 292 | 10.0            | 1       | malignant | Wong        |
| 294 | 1.0             | 1       | benign    | Lee         |
| 297 | 3.0             | 1       | benign    | Smith       |
| 315 | 9.0             | 1       | benign    | Wong        |
| 321 | 1.0             | 1       | benign    | Smith       |
| 411 | 1.0             | 1       | benign    | Lee         |
| 617 | 1.0             | 1       | benign    | Smith       |

---

Sure enough. Rather than leaving the cells of missing values empty, somebody has made the poor decision to enter question marks instead.

When you are dealing with other peoples' data, you'll find that this sort of the happens a LOT. It can be very aggravating, so we need to learn to treat these things as challenging puzzles instead of hassles!

Let's replace the question marks with nothing, so that this column becomes consistent with the rest. Fortunately, `DataFrame` (and `Series`) objects have a `replace()` function built in, so let's use that.

```
[67]: bcd['bare_nuclei'] = bcd['bare_nuclei'].replace('?', '')
```

---

In the cell below, confirm that we no longer have question marks in our “bare\_nuclei” column.

```
[68]: bcd[bcd['bare_nuclei']=='?']
```

```
[68]: Empty DataFrame
      Columns: [patient_id, clump_thickness, cell_size_uniformity,
      cell_shape_uniformity, marginal_adhesion, single_ep_cell_size, bare_nuclei,
      bland_chromatin, normal_nucleoli, mitoses, class, doctor_name]
      Index: []
```

---

**Note:** As mentioned above, extracting columns or other subsets of data from a pandas `DataFrame` or `Series` does not create a new object but rather a new label to the existing object.

So, for example, `the_IDs = bcd['patient_id']` does not make a new object, but rather creates a second label referring to the original object (consistent with the behavior of base Python).

In general, however, pandas methods (functions) *do* create new objects. Thus, the step of assigning the output of `.replace()` back to the original data frame column is necessary.

---

In the cell below, confirm that the output of `.replace()` and `bcd['bare_nuclei']` have different IDs.

```
[70]: id(bcd['bare_nuclei'])
```

```
[70]: 140153815319552
```

---

And now we can convert the column to numeric values.

```
[71]: bcd['bare_nuclei'] = pd.to_numeric(bcd['bare_nuclei'])
```

---

In the cell below, check the data types of columns in `bcd`.

```
[72]: bcd.dtypes
```

```
[72]: patient_id          int64
      clump_thickness    float64
      cell_size_uniformity float64
      cell_shape_uniformity int64
      marginal_adhesion  int64
      single_ep_cell_size int64
      bare_nuclei        float64
      bland_chromatin     float64
      normal_nucleoli     float64
      mitoses            int64
      class              object
      doctor_name        object
      dtype: object
```

---

Okay! We have now have gotten our data somewhat into shape, meaning:

- missing data are actually missing
- columns of numeric data are numeric in type
- the column of doctor names contains only last names

So now we can explore some ways to deal with missing values.

## 1.5 Dealing with missing data

### 1.5.1 Finding missing values

Even though this dataset isn't all that large:

```
[73]: bcd.shape
```

```
[73]: (699, 12)
```

699 rows is lot to look through “by hand” in order to find missing values.

We can test for missing values using the `DataFrame.isna()` method.

```
[74]: bcd.isna()
```

```
[74]:
```

|    | patient_id | clump_thickness | cell_size_uniformity | cell_shape_uniformity | \ |
|----|------------|-----------------|----------------------|-----------------------|---|
| 0  | False      | False           | False                | False                 |   |
| 1  | False      | False           | False                | False                 |   |
| 2  | False      | False           | False                | False                 |   |
| 3  | False      | False           | False                | False                 |   |
| 4  | False      | False           | False                | False                 |   |
| .. | ...        | ...             | ...                  | ...                   |   |

|     |       |       |       |       |
|-----|-------|-------|-------|-------|
| 694 | False | False | False | False |
| 695 | False | False | False | False |
| 696 | False | False | False | False |
| 697 | False | False | False | False |
| 698 | False | False | False | False |

|     | marginal_adhesion | single_ep_cell_size | bare_nuclei | bland_chromatin | \ |
|-----|-------------------|---------------------|-------------|-----------------|---|
| 0   | False             | False               | False       | False           |   |
| 1   | False             | False               | False       | False           |   |
| 2   | False             | False               | False       | False           |   |
| 3   | False             | False               | False       | False           |   |
| 4   | False             | False               | False       | False           |   |
| ..  | ...               | ...                 | ...         | ...             |   |
| 694 | False             | False               | False       | False           |   |
| 695 | False             | False               | False       | False           |   |
| 696 | False             | False               | False       | False           |   |
| 697 | False             | False               | False       | False           |   |
| 698 | False             | False               | False       | False           |   |

|     | normal_nucleoli | mitoses | class | doctor_name |
|-----|-----------------|---------|-------|-------------|
| 0   | False           | False   | False | False       |
| 1   | False           | False   | False | False       |
| 2   | False           | False   | False | False       |
| 3   | False           | False   | False | False       |
| 4   | False           | False   | False | False       |
| ..  | ...             | ...     | ...   | ...         |
| 694 | False           | False   | False | False       |
| 695 | False           | False   | False | False       |
| 696 | False           | False   | False | False       |
| 697 | False           | False   | False | False       |
| 698 | False           | False   | False | False       |

[699 rows x 12 columns]

By itself, that doesn't help us much. But if we combine it with summation (remember that `True` values count as 1 and `False` counts as zero):

```
[75]: bcd.isna().sum()
```

```
[75]: patient_id          0
      clump_thickness     1
      cell_size_uniformity 1
      cell_shape_uniformity 0
      marginal_adhesion    0
      single_ep_cell_size  0
      bare_nuclei         18
      bland_chromatin      4
```

```

normal_nucleoli      1
mitoses              0
class                0
doctor_name          0
dtype: int64

```

Now we have the counts by variable, and can easily see that there are missing values for a few of the variables.

The “bare\_nuclei” variable we dealt with earlier has the most missing values, with “bland\_chromatin” coming in a distant second.

Let’s check some of the rows with missing values and make sure everything else looks normal in those rows. Notice above that the output of `.isna()` is Boolean, so we can use it to do logical indexing.

```
[76]: bcd[bcd['bland_chromatin'].isna()]
```

```
[76]:
```

|     | patient_id | clump_thickness | cell_size_uniformity | cell_shape_uniformity | \ |
|-----|------------|-----------------|----------------------|-----------------------|---|
| 342 | 814265     | 2.0             | 1.0                  | 1                     |   |
| 343 | 814911     | 1.0             | 1.0                  | 1                     |   |
| 359 | 873549     | 10.0            | 3.0                  | 5                     |   |
| 365 | 897172     | 2.0             | 1.0                  | 1                     |   |

  

|     | marginal_adhesion | single_ep_cell_size | bare_nuclei | bland_chromatin | \ |
|-----|-------------------|---------------------|-------------|-----------------|---|
| 342 | 1                 | 2                   | 1.0         | NaN             |   |
| 343 | 1                 | 2                   | 1.0         | NaN             |   |
| 359 | 4                 | 3                   | 7.0         | NaN             |   |
| 365 | 1                 | 2                   | 1.0         | NaN             |   |

  

|     | normal_nucleoli | mitoses | class     | doctor_name |
|-----|-----------------|---------|-----------|-------------|
| 342 | 1.0             | 1       | benign    | Lee         |
| 343 | 1.0             | 1       | benign    | Doe         |
| 359 | 5.0             | 3       | malignant | Doe         |
| 365 | 1.0             | 1       | benign    | Lee         |

In the cell below, check the rows that have missing values for either clump thickness or cell size uniformity. Do this in one go rather than separately (remember about the element-wise or operator, “|”).

```
[90]: bcd[bcd['clump_thickness'].isna() | bcd['cell_size_uniformity'].isna()]
```

```
[90]:
```

|    | patient_id | clump_thickness | cell_size_uniformity | cell_shape_uniformity | \ |
|----|------------|-----------------|----------------------|-----------------------|---|
| 12 | 1041801    | NaN             | 3.0                  | 3                     |   |

  

|    | marginal_adhesion | single_ep_cell_size | bare_nuclei | bland_chromatin | \ |
|----|-------------------|---------------------|-------------|-----------------|---|
| 12 | 3                 | 2                   | 3.0         | 4.0             |   |



|    | normal_nucleoli | mitoses | class     | doctor_name |
|----|-----------------|---------|-----------|-------------|
| 12 | 4.0             | 1       | malignant | Smith       |

```
[79]: bcd[bcd['cell_size_uniformity'].isna()]
```

```
[79]: patient_id  clump_thickness  cell_size_uniformity  cell_shape_uniformity  \
6      1018099              1.0                NaN                1

      marginal_adhesion  single_ep_cell_size  bare_nuclei  bland_chromatin  \
6                   1                2          10.0          3.0

      normal_nucleoli  mitoses  class  doctor_name
6                1.0        1  benign        Doe
```

---

So far so good. It looks like the rows that have missing values just have one missing value, and everything else seems fine. But let's do check that no rows have more than one missing value.

To do this, we can sum the number of missing values across the columns (i.e. within each row), and then see what the maximum number of missing values within a row is.

```
[82]: row_na_totals = bcd.isna().sum(axis = 1)
      row_na_totals.max()
```

```
[82]: 1
```

So we see that no row has more than one missing value.

---

In the cell below, do the above calculation in one line.

```
[84]: bcd.isna().sum(axis = 1).max()
```

```
[84]: 1
```

---

## 1.5.2 Dealing with missing values

Now that we have determined that there are missing values, we have to determine how to deal with them.

**Ignoring missing values elementwise** One way to handle missing values is just to ignore them. Most of the standard math and statistical functions will do that by default.

So this:

```
[85]: bcd['clump_thickness'].mean()
```

```
[85]: 4.416905444126074
```

Computes the mean clump thickness ignoring the one missing value.

We can compute the mean (again ignoring missing values) for all the numeric columns like this:

```
[86]: bcd.mean(numeric_only = True) # the numeric_only refers to columns, not
      ↪missing values
```

```
[86]: patient_id          1.071704e+06
      clump_thickness     4.416905e+00
      cell_size_uniformity 3.137536e+00
      cell_shape_uniformity 3.207439e+00
      marginal_adhesion    2.793991e+00
      single_ep_cell_size  3.216023e+00
      bare_nuclei          3.538913e+00
      bland_chromatin      3.447482e+00
      normal_nucleoli      2.868195e+00
      mitoses              1.589413e+00
      dtype: float64
```

That worked, but the output is a little awkward because the patient ID is being treated as a numeric variable. We can fix that by converting the patient ID variable to a string variable.

```
[87]: bcd['patient_id'] = bcd['patient_id'].astype('string')
```

And now the means should look a little better because we won't have the mean for the ID column in the millions>

---

Recompute the mean for the numeric columns in the cell below.

```
[88]: bcd.mean(numeric_only = True)
```

```
[88]: clump_thickness      4.416905
      cell_size_uniformity 3.137536
      cell_shape_uniformity 3.207439
      marginal_adhesion    2.793991
      single_ep_cell_size  3.216023
      bare_nuclei          3.538913
      bland_chromatin      3.447482
      normal_nucleoli      2.868195
      mitoses              1.589413
      dtype: float64
```

---

**Removing missing values** We are about to start learning how to remove missing values from our data frame, *however...*

Before we start messing around too much with the values in our data frame, let's make sure we can easily “hit the reset button” and get back to a nice starting point. To do this, we'll want to

- reload the data
- modify the column of Dr. names
- set the patient ID to type str
- remove the question marks from the bare nuclei column
- set the bare nuclei column to numeric

This is a perfect job for a function!

---

In the cell below, finish writing the function to reset our data frame to the desired starting point.

```
[160]: def hit_reset():  
        bcd = pd.read_csv('./data/breast_cancer_data.csv')  
        bcd['doctor_name'] = bcd['doctor_name'].str.split().str[1]  
        bcd['patient_id'] = bcd['patient_id'].astype('string')  
        bcd['bare_nuclei'] = bcd['bare_nuclei'].replace('?', '')  
        bcd['bare_nuclei'] = pd.to_numeric(bcd['bare_nuclei'])  
  
        return bcd
```

---

**Removing rows with missing values** Obviously, rows in which all values are missing won't do us any good, so we can drop them with:

```
[162]: bcd = bcd.dropna(how = 'all')
```

This drops rows in which *all* of the values are missing. This code ran without error, but we know it also didn't do anything in this case because we don't have any rows in which all the values are missing!

Sometimes a case can be made for throwing out all observations (rows) that are incomplete, that is, if they contain *any* missing values.

```
[163]: bcd = bcd.dropna(how = 'any')
```

---

In the cell below, check the (new) shape of bcd.

```
[164]: bcd.shape
```

```
[164]: (674, 12)
```

It should have fewer rows now.

---

And now is a perfect time to test our function! In the cell below, hit the reset button on bcd.

```
[165]: bcd = hit_reset()
```

Check the shape.

```
[166]: bcd.shape
```

```
[166]: (699, 12)
```

Check the data types of the columns.

```
[167]: bcd.dtypes
```

```
[167]: patient_id          string
      clump_thickness    float64
      cell_size_uniformity float64
      cell_shape_uniformity int64
      marginal_adhesion  int64
      single_ep_cell_size int64
      bare_nuclei         float64
      bland_chromatin     float64
      normal_nucleoli     float64
      mitoses             int64
      class              object
      doctor_name        object
      dtype: object
```

Check the doctor name column.

```
[168]: bcd['doctor_name']
```

```
[168]: 0      Doe
      1      Smith
      2      Lee
      3      Smith
      4      Wong
      ...
      694    Lee
      695    Smith
      696    Lee
      697    Lee
      698    Wong
      Name: doctor_name, Length: 699, dtype: object
```

---

***Removing columns with missing values*** And we could do the same for columns if we wished, though this is less frequently done. We just need to change the axis (direction) over which

DataFrame.dropna() works.

```
[169]: bcd = bcd.dropna(axis = 1, how = 'any') # drop columns rather than rows
```

This leaves us with only the complete columns.

```
[170]: bcd.shape
```

```
[170]: (699, 7)
```

Let's see which they are.

```
[171]: bcd.columns
```

```
[171]: Index(['patient_id', 'cell_shape_uniformity', 'marginal_adhesion',  
          'single_ep_cell_size', 'mitoses', 'class', 'doctor_name'],  
          dtype='object')
```

**Filling in missing values** Occasionally, we may want to fill in missing values. This isn't very common, but might be useful if some other function you are using doesn't handle missing values gracefully.

Before filling in missing values, we need to restore our data frame so it actually has missing values. Good thing we wrote that function!

```
[172]: bcd = hit_reset()
```

We can fill in missing values with any single value we want, such as a zero.

```
[173]: bcd = bcd.fillna(0)
```

---

In the cell below, check to see that we no longer have missing values.

```
[178]: bcd.isna().sum()
```

```
[178]: patient_id          0  
       clump_thickness   0  
       cell_size_uniformity 0  
       cell_shape_uniformity 0  
       marginal_adhesion  0  
       single_ep_cell_size 0  
       bare_nuclei       0  
       bland_chromatin   0  
       normal_nucleoli   0  
       mitoses           0  
       class             0  
       doctor_name       0  
       dtype: int64
```

In the cell below, reset the data and verify that the missing data are back.

```
[180]: bcd = hit_reset()
bcd.isna().sum()
```

```
[180]: patient_id          0
      clump_thickness      1
      cell_size_uniformity 1
      cell_shape_uniformity 0
      marginal_adhesion    0
      single_ep_cell_size  0
      bare_nuclei          18
      bland_chromatin       4
      normal_nucleoli       1
      mitoses              0
      class                0
      doctor_name          0
      dtype: int64
```

---

In the cell below, fill the missing values in each column with the column mean. (Hint: this is pandas, so this is actually easy!)

```
[181]: bcd = bcd.fillna(bcd.mean)
```

And now verify that there are no more missing values.

```
[182]: bcd.isna().sum()
```

```
[182]: patient_id          0
      clump_thickness      0
      cell_size_uniformity 0
      cell_shape_uniformity 0
      marginal_adhesion    0
      single_ep_cell_size  0
      bare_nuclei          0
      bland_chromatin       0
      normal_nucleoli       0
      mitoses              0
      class                0
      doctor_name          0
      dtype: int64
```

---

## 1.6 Summary

In this tutorial, we learned or remembered how to do some of the foundational data wrangling tasks. These are:

- importing data into pandas from a data file
- cleaning up the data in the columns
- converting columns to the appropriate type
- removing or filling in missing values