4 Tabular Data and Basic Data Operations

4.1 Introduction

This document demonstrates the use of the `pandas` library in Python to do basic data wrangling and summarization.

> **Note**
>
> If you do not have the `pandas` library installed then you will need to run
>
> `pip install pandas`
>
> in the Jupyter terminal to install. **Remember:** you only need to install once per machine (or Colab session, for packages that don't come pre-installed).

4.2 Reading Tabular Data into Python

We're going to be exploring `pandas` in the context of the famous Titanic dataset. We'll work with a subset of this dataset, but more information about it all can be found here.

We start by loading the `numpy` and `pandas` libraries. Most of our data wrangling work will happen with functions from the `pandas` library, but the `numpy` library will be useful for performing certain mathematical operations should we choose to transform any of our data.

```
import numpy as np
import pandas as pd
```

```
data_dir = "https://dlsun.github.io/pods/data/"
df_titanic = pd.read_csv(data_dir + "titanic.csv")
```

> **Example**
>
> We've already seen `read_csv()` used many times for importing CSV files into Python, but it bears repeating.

Data files of many different types and shapes can be read into Python with similar functions, but we will focus on tabular data.

4.2.1 Tidy Data is Special Tabular Data

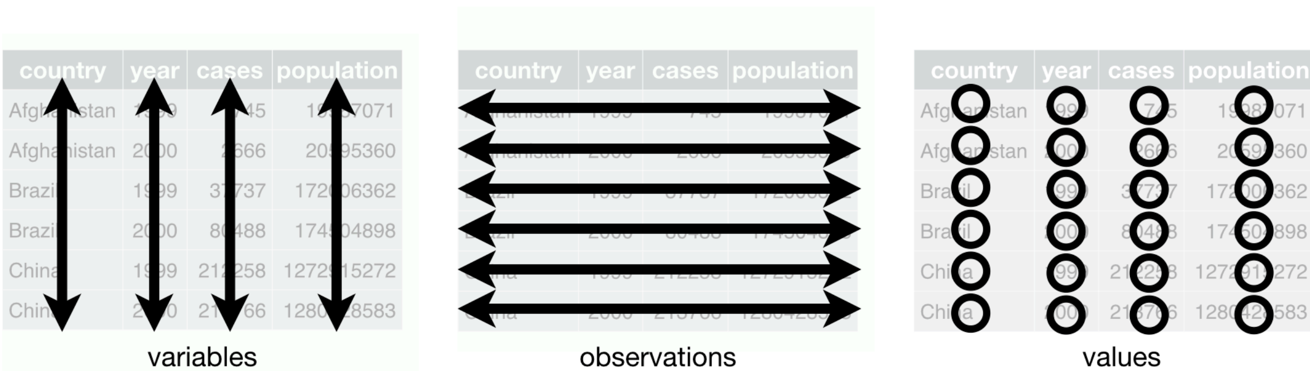
For most people, the image that comes to mind when thinking about data is indeed something tabular or spreadsheet-like in nature. **Which is great!**

Tabular data is a form preferred by MANY different data operations and work. However, we will want to take this one step further. In almost all data science work we want our data to be **tidy**

ⓘ Note

A dataset is **tidy** if it adheres to following three characteristics:

- Every column is a variable
- Every row is an observation
- Every cell is a single value



Check In

With 2-3 people around you, navigate to the [GapMinder Data]() site and download a single CSV file of your choice. Open it up in Excel or your application of choice. Is this dataset *tidy*? If not, then what would have to change to make it *tidy*?

Learn More

The term "tidy data" was first popularized in [this paper]() by R developer Hadley Wickham.

You may have noticed that `plotnine` (`ggplot`) is basically built to take *tidy* data. Variables are specified in the aesthetics function to map them (i.e. columns) in our dataset to plot elements. This type of behavior is **EXTREMELY** common among functions that work with data in all languages, and so the importance of getting our data into a *tidy* format cannot be overstated.

In Python, there are at least two quick ways to view a dataset we've read in:

```
df_titanic
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
0	Abbing, Mr. Anthony	male	42.0	3rd	S	United States	5547.0	7.11	0
1	Abbott, Mr. Eugene Joseph	male	13.0	3rd	S	United States	2673.0	20.05	0

	name	gender	age	class	embarked	country	ticketno	fare	survived
2	Abbott, Mr. Rossmore Edward	male	16.0	3rd	S	United States	2673.0	20.05	0
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.05	1
4	Abelseth, Miss. Karen Marie	female	16.0	3rd	S	Norway	348125.0	7.13	1
...	...	...	...	...	...	...	...	...	...
2202	Wynn, Mr. Walter	male	41.0	deck crew	B	England	NaN	NaN	1
2203	Yearsley, Mr. Harry	male	40.0	victualling crew	S	England	NaN	NaN	1
2204	Young, Mr. Francis James	male	32.0	engineering crew	S	England	NaN	NaN	0
2205	Zanetti, Sig. Minio	male	20.0	restaurant staff	S	England	NaN	NaN	0
2206	Zarracchi, Sig. L.	male	26.0	restaurant staff	S	England	NaN	NaN	0

2207 rows × 9 columns

```
df_titanic.head()
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
0	Abbing, Mr. Anthony	male	42.0	3rd	S	United States	5547.0	7.11	0
1	Abbott, Mr. Eugene Joseph	male	13.0	3rd	S	United States	2673.0	20.05	0
2	Abbott, Mr. Rossmore Edward	male	16.0	3rd	S	United States	2673.0	20.05	0
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.05	1
4	Abelseth, Miss. Karen Marie	female	16.0	3rd	S	Norway	348125.0	7.13	1

The latter (`.head()`) is usually preferred in case the dataset is large.

Check In

Does the titanic dataset appear to be in *tidy* format?

4.3 The "Big Five" Verbs of Data Wrangling

Data wrangling can involve a lot of different steps and operations to get data into a *tidy* format and ready for analysis and visualization. The vast majority of these fall under the umbrella one the following five operations:

1. **Select** columns/variables of interest

2. **Filter** rows/observations of interest

3. **Arrange** the rows of a dataset by column(s) of interest (i.e. order or sort)

4. **Mutate** the columns of a dataset (i.e. create or transform variables)

5. **Summarize** the rows of a dataset for column(s) of interest

4.3.1 Select Columns/Variables

Suppose we want to select the `age` variable from the titanic `DataFrame`. There are three ways to do this.

1. Use `.loc`, specifying both the rows and columns. (The colon `:` is Python shorthand for "all".)

```
df_titanic.loc[:, "age"]
```

2. Access the column as you would a key in a `dict`.

```
df_titanic["age"]
```

3. Access the column as an attribute of the `DataFrame`.

```
df_titanic.age
```

Method 3 (attribute access) is the most concise. However, it does not work if the variable name contains spaces or special characters, begins with a number, or matches an existing attribute of the `DataFrame`. So, methods 1 and 2 are usually safer and preferred.

To select multiple columns, you would pass in a *list* of variable names, instead of a single variable name. For example, to select both `age` and `fare`, either of the two methods below would work (and produce the same result):

```
# Method 1
df_titanic.loc[:, ["age", "fare"]].head()

# Method 2
df_titanic[["age", "fare"]].head()
```

4.3.2 Filter Rows/Observations

4.3.2.1 Selecting Rows/Observations by Location

Before we see how to **filter** (i.e. **subset**) the rows of dataset based on some condition, let's see how to select rows by explicitly identifying them.

We can select a row by its position using the `.iloc` attribute. Keeping in mind that the first row is actually row 0, the fourth row could be extracted as:

```
df_titanic.iloc[3]
```

```
name        Abbott, Mrs. Rhoda Mary 'Rosa'
gender                              female
age                                   39.0
```

```
class                          3rd
embarked                         S
country                    England
ticketno                    2673.0
fare                         20.05
survived                         1
Name: 3, dtype: object
```

Notice that a single row from a `DataFrame` is no longer a `DataFrame` but a different data structure, called a `Series`.

We can also select multiple rows by passing a *list* of positions to `.iloc`.

```
df_titanic.iloc[[1, 3]]
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
1	Abbott, Mr. Eugene Joseph	male	13.0	3rd	S	United States	2673.0	20.05	0
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.05	1

Notice that when we select multiple rows, we get a `DataFrame` back.

So a `Series` is used to store a single observation (across multiple variables), while a `DataFrame` is used to store multiple observations (across multiple variables).

If selecting consecutive rows, we can use Python's `slice` notation. For example, the code below selects all rows from the fourth row, up to (but not including) the tenth row.

```
df_titanic.iloc[3:9]
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.0500	1
4	Abelseth, Miss. Karen Marie	female	16.0	3rd	S	Norway	348125.0	7.1300	1
5	Abelseth, Mr. Olaus Jørgensen	male	25.0	3rd	S	United States	348122.0	7.1300	1
6	Abelson, Mr. Samuel	male	30.0	2nd	C	France	3381.0	24.0000	0
7	Abelson, Mrs. Hannah	female	28.0	2nd	C	France	3381.0	24.0000	1
8	Abī-Al-Munà, Mr. Nāsīf Qāsim	male	27.0	3rd	C	Lebanon	2699.0	18.1509	1

4.3.2.2 Selecting Rows/Observations by Condition

We'll often want to **filter** or **subset** the rows of a dataset based on some condition. To do this we'll take advantage of **vectorization** and **boolean masking**.

Recall that we can compare the values of a variable/column to a particular value in the following way, and observe the result.

```
df_titanic["age"] > 30
```

```
0        True
1       False
2       False
3        True
4       False
        ...
2202     True
2203     True
2204     True
2205    False
2206    False
Name: age, Length: 2207, dtype: bool
```

We can use these `True` and `False` values to filter/subset the dataset! The following subsets the titanic dataset down to only those individuals (rows) with ages over 30.

```
df_titanic[df_titanic["age"] > 30]
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
0	Abbing, Mr. Anthony	male	42.0	3rd	S	United States	5547.0	7.1100	0
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.0500	1
12	Ahlin, Mrs. Johanna Persdotter	female	40.0	3rd	S	Sweden	7546.0	9.0906	0
15	Aldworth, Mr. Augustus Henry	male	35.0	2nd	S	England	248744.0	13.0000	0
21	Allen, Mr. William Henry	male	39.0	3rd	S	England	373450.0	8.0100	0
...	...	...	...	...	...	...	...	...	...
2197	Worthman, Mr. William Henry	male	37.0	engineering crew	S	England	NaN	NaN	0
2200	Wright, Mr. William	male	40.0	victualling crew	S	England	NaN	NaN	1
2202	Wynn, Mr. Walter	male	41.0	deck crew	B	England	NaN	NaN	1
2203	Yearsley, Mr. Harry	male	40.0	victualling crew	S	England	NaN	NaN	1
2204	Young, Mr. Francis James	male	32.0	engineering crew	S	England	NaN	NaN	0

984 rows × 9 columns

We can combine multiple conditions using `&` (and) and `|` (or). The following subsets the titanic dataset down to females over 30 years of age.

```
df_titanic[(df_titanic["age"] > 30) & (df_titanic["gender"] == "female")]
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
3	Abbott, Mrs. Rhoda Mary 'Rosa'	female	39.0	3rd	S	England	2673.0	20.0500	1
12	Ahlin, Mrs. Johanna Persdotter	female	40.0	3rd	S	Sweden	7546.0	9.0906	0
35	Andersson, Miss. Ida Augusta Margareta	female	38.0	3rd	S	Sweden	347091.0	7.1506	0
40	Andersson, Mrs. Alfrida Konstantia Brogren	female	39.0	3rd	S	Sweden	347082.0	31.0506	0
44	Andrews, Miss. Kornelia Theodosia	female	62.0	1st	C	United States	13502.0	77.1902	1
...	...	...	...	...	...	...	...	...	...
1997	Robinson, Mrs. Annie	female	41.0	victualling crew	S	England	NaN	NaN	1
2059	Smith, Miss. Katherine Elizabeth	female	45.0	victualling crew	S	England	NaN	NaN	1
2076	Stap, Miss. Sarah Agnes	female	47.0	victualling crew	S	England	NaN	NaN	1
2143	Wallis, Mrs. Catherine Jane	female	36.0	victualling crew	S	England	NaN	NaN	0
2145	Walsh, Miss. Catherine	female	32.0	victualling crew	S	Ireland	NaN	NaN	0

206 rows × 9 columns

Check In

With the 2-3 people around you, how would you find the just the names of the males under 20 years of age who survived (in the titanic dataset) with a single line of code?

4.3.3 Arrange Rows

As part of exploratory data analysis and some reporting efforts, we will want to sort a dataset or set of results by one or more variables of interest.

We can do this with `.sort_values` in either *ascending* or *descending* order.

The following sorts the titanic dataset by `age` in decreasing order.

```
df_titanic.sort_values(by = ["age"], ascending=False)
```

	name	gender	age	class	embarked	country	ticketno	fare	survived
1176	Svensson, Mr. Johan	male	74.000000	3rd	S	Sweden	347060.0	7.1506	0

	name	gender	age	class	embarked	country	ticketno	fare	survived
820	Mitchell, Mr. Henry Michael	male	72.000000	2nd	S	England	24580.0	10.1000	0
53	Artagaveytia, Mr. Ramon	male	71.000000	1st	C	Argentina	17609.0	49.1001	0
456	Goldschmidt, Mr. George B.	male	71.000000	1st	C	United States	17754.0	34.1301	0
282	Crosby, Captain. Edward Gifford	male	70.000000	1st	S	United States	5735.0	71.0000	0
...	...	...	...	...	...	...	...	...	...
1182	Tannūs, Master. As'ad	male	0.416667	3rd	C	Lebanon	2625.0	8.1004	1
296	Danbom, Master. Gilbert Sigvard Emanuel	male	0.333333	3rd	S	Sweden	347080.0	14.0800	0
316	Dean, Miss. Elizabeth Gladys 'Millvina'	female	0.166667	3rd	S	England	2315.0	20.1106	1
439	Gheorgheff, Mr. Stanio	male	NaN	3rd	C	Bulgaria	349254.0	7.1711	0
677	Kraeff, Mr. Theodor	male	NaN	3rd	C	Bulgaria	349253.0	7.1711	0

2207 rows × 9 columns

Note

Notice that in these last few sections, we have not made any *permanent* changes to the `df_titanic` object. We have only asked python do some selecting/filtering/sorting and then to print out the results, not save them.

If we wanted `df_titanic` to become permanently sorted by age, we would **re-assign** the object:

```
df_titanic = df_titanic.sort_values(by = ["age"], ascending=False)
```

Warning

You will sometimes see object reassignment happen in a different way, using an `inplace = True` argument, like this:

```
df_titanic.sort_values(by = ["age"], ascending=False, inplace=True)
```

We strongly recommend **against** this approach, for two reason:

1. When an object is "overwritten" via reassignment, that's a major decision; you lose the old version of the object. It should be made deliberately and obviously. The `inplace` argument is easy to miss when copying/editing code, so it can lead to accidental overwriting that is hard to keep track of.

2. Not all functions of DataFrames have an `inplace` option. It can be frustrating to get into the habit of using it, only to find out the hard way that it's not available half the time!

4.3.4 Mutate Column(s)

The variables available to us in our original dataset contain all of the information we have access to, but the best insights may instead come from transformations of those variables.

4.3.4.1 Transforming Quantitative Variables

One of the simplest reasons to want to transform a quantitative variable is to change the measurement units.

Here we change the `age` of passengers from a value in years to a value in decades.

```
df_titanic["age"] = df_titanic["age"] / 10
```

If we have a quantitative variable that is particularly skewed, then it might be a good idea to transform the values of that variable…like taking the `log` of the values.

> **Note**
>
> This was a strategy you saw employed with the GapMinder data!

Below is an example of taking the `log` of the `fare` variable. Notice that we're making use of the `numpy` here to take the `log`.

```
df_titanic["fare"] = np.log(df_titanic["fare"])
```

Remember that we can take advantage of **vectorization** here too. The following operation wouldn't really make physical sense, but it's an example of **creating a new variable** out of existing variables.

```
df_titanic["nonsense"] = df_titanic["fare"] / df_titanic["age"]
```

Note that we created the new variable, `nonsense`, by specifying on the left side of the `=` here and populating that column/variable via the expression on the right side of the `=`.

We could want to create a new variable by categorizing (or discretizing) the values of a quantitative variable (i.e. convert a quantitative variable to a categorical variable). We can do so with `cut`.

In the following, we create a new `age_cat` variable which represents whether a person is a child or an adult.

```
df_titanic["age_cat"] = pd.cut(df_titanic["age"],
                              bins = [0, 18, 100],
                              labels = ["child", "adult"])
```

> **Check In**
>
> Consider the four mutations we just performed. In which ones did we **reassign** a column of the dataset, thus *replacing* the old values with new ones? In which ones did we **create** a brand-new column, thus retaining the old column(s) that were involved in the calculation?

4.3.4.2 Transforming Categorical Variables

In some situations, especially later with modeling, we'll need to convert categorical variables (stored as text) into quantitative (often coded) variables. Binary categorical variables can be converted into quantitative variables by coding one category as 1 and the other category as 0. (In fact, the **survived** column in the titanic dataset has already been coded this way.) The easiest way to do this is to create a boolean mask. For example, to convert `gender` to a quantitative variable `female`, which is 1 if the passenger was female and 0 otherwise, we can do the following:

```
df_titanic["female"] = 1 * (df_titanic["gender"] == "female")
```

What do we do about a categorical variable with more than twwo categories, like `embarked`, which has four categories? In general, a categorical variable with **K** categories can be converted into **K** separate 0/1 variables, or **dummy variables**. Each of the **K** dummy variables is an indicator for one of the **K** categories. That is, a dummy variable is 1 if the observation fell into its particular category and 0 otherwise.

Although it is not difficult to create dummy variables manually, the easiest way to create them is the `get_dummies()` function in `pandas`.

```
pd.get_dummies(df_titanic["embarked"])
```

	B	C	Q	S
1176	False	False	False	True
820	False	False	False	True
53	False	True	False	False
456	False	True	False	False
282	False	False	False	True
...	...	...	...	...
1182	False	True	False	False
296	False	False	False	True
316	False	False	False	True
439	False	True	False	False
677	False	True	False	False

2207 rows × 4 columns

We may also want to change the levels of a categorical variable. A categorical variable can be transformed by mapping its levels to new levels. For example, we may only be interested in whether a person on the titanic was a passenger or a crew member. The variable `class` is too detailed. We can create a new variable, `type`, that is derived from the existing variable `class`. Observations with a `class` of "1st", "2nd", or "3rd" get a value of "passenger", while observations with a `class` of "victualling crew", "engineering crew", or "deck crew" get a value of "crew".

```
df_titanic["type"] = df_titanic["class"].map({
    "1st": "passenger",
    "2nd": "passenger",
    "3rd": "passenger",
    "victualling crew": "crew",
    "engineering crew": "crew",
    "deck crew": "crew"
})

df_titanic
```

	name	gender	age	class	embarked	country	ticketno	fare	survived	nonsense	age_cat
1176	Svensson, Mr. Johan	male	7.400000	3rd	S	Sweden	347060.0	1.967196	0	0.265837	child
820	Mitchell, Mr. Henry Michael	male	7.200000	2nd	S	England	24580.0	2.312535	0	0.321185	child
53	Artagaveytia, Mr. Ramon	male	7.100000	1st	C	Argentina	17609.0	3.893861	0	0.548431	child
456	Goldschmidt, Mr. George B.	male	7.100000	1st	C	United States	17754.0	3.530180	0	0.497208	child
282	Crosby, Captain. Edward Gifford	male	7.000000	1st	S	United States	5735.0	4.262680	0	0.608954	child
...	...	...	...	...	...	...	...	...	...	...	...
1182	Tannūs, Master. As'ad	male	0.041667	3rd	C	Lebanon	2625.0	2.091913	1	50.205923	child
296	Danbom, Master. Gilbert Sigvard Emanuel	male	0.033333	3rd	S	Sweden	347080.0	2.644755	0	79.342661	child
316	Dean, Miss. Elizabeth Gladys 'Millvina'	female	0.016667	3rd	S	England	2315.0	3.001247	1	180.074822	child
439	Gheorgheff, Mr. Stanio	male	NaN	3rd	C	Bulgaria	349254.0	1.970059	0	NaN	NaN
677	Kraeff, Mr. Theodor	male	NaN	3rd	C	Bulgaria	349253.0	1.970059	0	NaN	NaN

2207 rows × 13 columns

4.3.5 Summarizing Rows

Summarization of the rows of a dataset for column(s) of interest can take many different forms. This introduction will not be exhaustive, but certainly cover the basics.

4.3.5.1 Summarizing a Quantitative Variable

There are a few descriptive statistics that can be computed directly including, but not limited to, the mean and median.

```
df_titanic["age"].mean()

df_titanic["age"].median()

df_titanic[["age", "fare"]].mean()
```

```
age      3.043673
fare     2.918311
dtype: float64
```

We can ask for a slightly more comprehensive description using `.describe()`

```
df_titanic["age"].describe()

df_titanic.describe()
```

	age	ticketno	fare	survived	nonsense	female
count	2205.000000	1.316000e+03	1291.000000	2207.000000	1289.000000	2207.000000
mean	3.043673	2.842157e+05	2.918311	0.322157	2.147877	0.221568
std	1.215968	6.334726e+05	0.974452	0.467409	7.237694	0.415396
min	0.016667	2.000000e+00	1.108728	0.000000	0.265837	0.000000
25%	2.200000	1.426225e+04	1.971383	0.000000	0.742371	0.000000
50%	2.900000	1.114265e+05	2.645480	0.000000	0.936833	0.000000
75%	3.800000	3.470770e+05	3.435945	1.000000	1.260935	0.000000
max	7.400000	3.101317e+06	6.238443	1.000000	180.074822	1.000000

Note that, by default, `.describe()` provides descriptive statistics for only the quantitative variables in the dataset.

We can enhance numerical summaries with `.groupby()`, which allows us to specify one or more variables that we'd like to **group** our work by.

```
df_titanic[["age", "survived"]].groupby("survived").mean()
```

	age
survived	
0	3.083194
1	2.960631

Check In

With 2-3 people around you, look up how you would compute the correlation between two quantitative variables in Python. Compute the correlation between the `age` and `fare` variables in the titanic dataset.

4.3.5.2 Summarizing a Categorical Variable

When it comes to categorical variables we're most often interested in **frequency distributions** (counts), **relative frequency distributions**, and **cross-tabulations**.

```
df_titanic["class"].unique()

df_titanic["class"].describe()
```

```
count     2207
unique       7
top        3rd
freq       709
Name: class, dtype: object
```

The `.unique()` here allows us to see the unique values of the `class` variable. Notice that the results of `.describe()` on a categorical variable are much different.

To completely summarize a single categorical variable, we report the number of times each level appeared, or its **frequency**.

```
df_titanic["class"].value_counts()
```

```
class
3rd                 709
victualling crew    431
1st                 324
engineering crew    324
2nd                 284
restaurant staff     69
deck crew            66
Name: count, dtype: int64
```

Instead of reporting counts, we can also report proportions or probabilities, or the **relative frequencies**. We can calculate the relative frequencies by specifying `normalize=True` in `.value_counts()`.

```
df_titanic["class"].value_counts(normalize=True)
```

```
class
3rd                  0.321251
victualling crew     0.195288
1st                  0.146806
engineering crew     0.146806
2nd                  0.128681
restaurant staff     0.031264
deck crew            0.029905
Name: proportion, dtype: float64
```

Cross-tabulations are one way we can investigate possible relationships between categorical variables. For example, what can we say about the relationship between `gender` and `survival` on the Titanic?

Check In

Summarize `gender` and `survival` individually by computing the frequency distributions of each.

This does not tell us how `gender` interacts with `survival`. To do that, we need to produce a *cross-tabulation*, or a "cross-tab" for short. (Statisticians tend to call this a *contingency table* or a *two-way table*.)

```
pd.crosstab(df_titanic["survived"], df_titanic["gender"])
```

gender	female	male
survived		
0	130	1366
1	359	352

A cross-tabulation of two categorical variables is a two-dimensional array, with the levels of one variable along the rows and the levels of the other variable along the columns. Each cell in this array contains the number of observations that had a particular combination of levels. So in the Titanic data set, there were 359 females who survived and 1366 males who died. From the cross-tabulation, we can see that there were more females who survived than not, while there were more males who died than not. Clearly, gender had a strong influence on survival because of the Titanic's policy of "women and children first".

To get probabilities instead of counts, we specify `normalize=True`.

```
pd.crosstab(df_titanic["survived"], df_titanic["gender"], normalize=True)
```

gender	female	male
survived		
0	0.058903	0.618940
1	0.162664	0.159493

Check In

What about conditional proportions? With 2-3 people around you, discuss how you would compute *the proportion of females that survived* and *the proportion of males that survived* and then do it.

Note, there are multiple ways to do this.

Practice Activity

Open up this colab notebook and make a copy.

Fill out the sections where indicated, render it to html with Quarto, and push your final notebook and html document to a repository on GitHub (same one as Practice Activity 1.1 is good). Then share this repository link in the quiz question.

Edit this page