Week 6

pandas dataframes

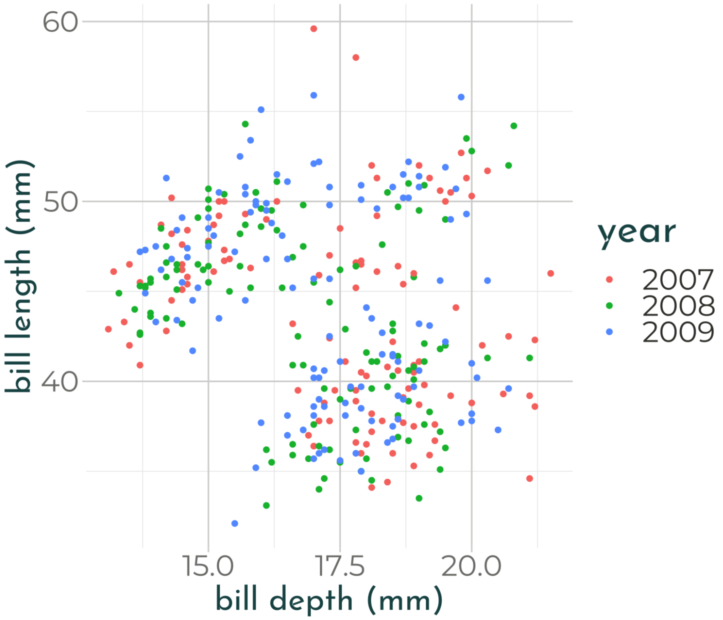
STAT 198/298 Fall 2020

Agenda

1. Review: Penguin Arrays
2. Pandas Dataframes

From the lab

```r
library(palmerpenguins)
library(ggplot2)
penguins <- as.data.frame(unclass
ggplot(penguins,
       aes(x = bill_depth_mm,
           y = bill_length_mm,
           color = factor(year))) +
  geom_point() +
  labs(x = "bill depth (mm)",
       y = "bill length (mm)",
       color = "year") +
  theme_xaringan()
```



R Dataframe

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	bod
Adelie	Torgersen	39.1	18.7	181	3750
Adelie	Torgersen	39.5	17.4	186	3800
Adelie	Torgersen	40.3	18.0	195	3250
Adelie	Torgersen	NA	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193	3450

Properties

1. A *list* of atomic vectors (columns), where the *keys* are called the *names*.
2. Each atomic vector can be a different type.
3. Each atomic vector can be the same length.
4. Can be indexed like a matrix (`penguins[3, 2]`) or a list (`penguins[[2]][3]` or `penguins$island[3]`).
5. Can add row names.

Rownames in R

```r
small_penguins <- slice(penguins, 1:5)
rownames(small_penguins) <- c("janet", "phyllis", "jose",
                              "benny", "marty")
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_m
janet	Adelie	Torgersen	39.1	18.7	181
phyllis	Adelie	Torgersen	39.5	17.4	186
jose	Adelie	Torgersen	40.3	18.0	195
benny	Adelie	Torgersen	NA	NA	NA
marty	Adelie	Torgersen	36.7	19.3	193

Indexing by rowname

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	bod
Adelie	Torgersen	39.1	18.7	181	375
Adelie	Torgersen	39.5	17.4	186	380
Adelie	Torgersen	40.3	18.0	195	325
Adelie	Torgersen	NA	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193	345

```r
small_penguins["phyllis", "bill_length_mm"]
```

```
## [1] 39.5
```

Colnames in R

```
colnames(small_penguins)
```

```
## [1] "species"           "island"
## [3] "bill_length_mm"    "bill_depth_mm"
## [5] "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

```
names(small_penguins)
```

```
## [1] "species"           "island"
## [3] "bill_length_mm"    "bill_depth_mm"
## [5] "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

What's the difference?

CODE

Advice on rownames

"Generally, it is best to avoid row names, because they are basically a character column with different semantics than every other column."

> Treat rownames like an ordinary column of strings.

```
small_penguins <- rownames_to_column(small_penguins,
                                     var = "given_name")
small_penguins[small_penguins$given_name == "phyllis",
               "bill_length_mm"]
```

```
## # A tibble: 1 x 1
##   bill_length_mm
##            <dbl>
## 1           39.5
```

Into Python

Step one: remove missing values (in R).

```
small_penguins <- small_penguins %>%
  select(-given_name) %>%
  tidyr::drop_na()
```

Step two: make type homogeneous

```
import numpy as np
pypenguins = r.small_penguins
pypenguins = {k:v for (k,v) in pypenguins.items() if k not in ["species'
pg_array = np.array(list(pypenguins.values()), dtype = "float64").transp
pg_array
```

```
## array([[  39.1,   18.7,  181. , 3750. , 2007. ],
##        [  39.5,   17.4,  186. , 3800. , 2007. ],
##        [  40.3,   18. ,  195. , 3250. , 2007. ],
##        [  36.7,   19.3,  193. , 3450. , 2007. ]])
```

```
pg_array[1, 0]
```

```
## 39.5
```

Limitations of Numpy Array

1. Type homogenous
2. Can only subset by index
   - Loses context of data

pandas

A package built on top of Numpy to provide data structures and operations needed by modern data science workflows.

New data structures

1. *Series*: roughly, a named atomic vector in R
2. *Dataframe*: roughly, an R dataframe

Pandas

Install pandas

```
reticulate::py_install("pandas")
```

Load pandas

```
import pandas as pd
```

Pandas Series

```
s = pd.Series([1, 2, 3, 4])
s
```

```
## 0    1
## 1    2
## 2    3
## 3    4
## dtype: int64
```

Series are like one dimensional numpy arrays but with an *explicit* index.

```
s.index
```

```
## RangeIndex(start=0, stop=4, step=1)
```

```
s.values
```

```
## array([1, 2, 3, 4])
```

Series Indexing

We can subset a series just like a list or an array.

```
s[0:2]
```

```
## 0    1
## 1    2
## dtype: int64
```

Or we can use an alternative, explicit index (or name).

```
s = pd.Series([1, 2, 3, 4],
              index = ["one", "two", "three", "four"])
s
```

```
## one      1
## two      2
## three    3
## four     4
## dtype: int64
```

Series indexing

```
## one      1
## two      2
## three    3
## four     4
## dtype: int64
```

Subset by new explicit index:

```
s["two"]
```

```
## 2
```

Or continue to use the *implicit* integer index.

```
s[1]
```

```
## 2
```

These kinda look like...

Dictionaries!

```
d = {"one":1, "two":2, "three":3, "four":4}
d["two"]
```

```
## 2
```

```
d[2]
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): KeyError: 1
```

```
pd.Series(d)[1]
```

```
## 2
```

```
pd.Series(d)["one":"two"]
```

```
## one      1
## two      2
## dtype: int64
```

Pandas Dataframes

A two-dimensional generalization of a series. Let's build one.

```
d_pop = {"California": 38332521,
         "Texas": 26448193,
         "New York": 19651127,
         "Florida": 19552860,
         "Illinois": 12882135}
s_pop = pd.Series(d_pop)
s_pop
```

```
## California    38332521
## Texas         26448193
## New York      19651127
## Florida       19552860
## Illinois      12882135
## dtype: int64
```

```
s_area = pd.Series({"California": 423967,
                    "Florida": 170312,
                    "Illinois": 149995,
                    "New York": 141297,
                    "Texas": 695662})
```

Pandas Dataframes

`CODE`

Pandas Dataframes

```
states = pd.DataFrame({"population": s_pop, "area": s_area})
states
```

```
##             population    area
## California    38332521  423967
## Florida       19552860  170312
## Illinois      12882135  149995
## New York      19651127  141297
## Texas         26448193  695662
```

- A dataframe can also be built from a numpy array.
- If column or row names are omitted, integers indices are used.

Penguins Dataframe

```
pypenguins = pd.DataFrame(r.small_penguins)
pypenguins
```

```
##   species     island  bill_length_mm  ...  body_mass_g     sex  year
## 0  Adelie  Torgersen            39.1  ...         3750    male  2007
## 1  Adelie  Torgersen            39.5  ...         3800  female  2007
## 2  Adelie  Torgersen            40.3  ...         3250  female  2007
## 3  Adelie  Torgersen            36.7  ...         3450  female  2007
## 
## [4 rows x 8 columns]
```

Subsetting Penguins

Question: How do I extract a dataframe of only the female penguins with bill lengths greater than 40 mm?

```
female_mask = pypenguins["sex"] == "female"
short_bill_mask = pypenguins["bill_length_mm"] > 40
pypenguins.loc[female_mask & short_bill_mask, :]
```

```
##   species     island  bill_length_mm  ...  body_mass_g     sex  year
## 2  Adelie  Torgersen            40.3  ...         3250  female  2007
##
## [1 rows x 8 columns]
```