Data analysis for Biomedical Engineers

An introduction to Pandas

CU@EMBS technical workshop series

Presented by
François Charih

Note: Create a Google account if you do not own one already.

Introduction

About me

- CU@EMBS communications officer
- Ph.D. student in Electrical and Computer Engineering (Carleton, 2023)
- M.A.Sc. in Electrical and Computer Engineering (Carleton, 2018)
- B.A.Sc. in Chemical Engineering/B.Sc. in Biochemistry (Ottawa, 2016)

Research interests: Computational biology, *in silico* drug design, cancer, neurodegeneration
Hobbies: Reading and songwriting

http://charih.ca

Introduction

Data analysis in the 21st century

Real world data can be *really* hard to deal with...

- **Voluminous**
  - Millions of observations not unusual
- **Complex**
  - High-dimensional (thousands of correlates)
  - Polymorphic (images, text, time series, etc.)
  - Concept drift -- evolution of data generating process over time
- **Messy**
  - Unstructured
  - Inconsistent
  - Incomplete
  - Outliers

Introduction

Seeking a unicorn in a zoo

One must choose his/her weapon wisely!

MATLAB

perl
Programming Language

BASH
THE BOURNE-AGAIN SHELL

R

julia

Introduction

The Python data science ecosystem is ripe

- **Free software (unlike MATLAB)**
  - … as in "speech" and "beer"!
- **Production-ready (unlike R and Julia)**
  - Deployed in production in many industries
- **General-purpose programming support**
  - Interface with general software (web applications, servers, databases, GUIs, etc.)
- **Vast ecosystem**
  - Tensorflow, PyTorch, Keras, scikit-learn (machine learning)
  - NumPy (numerical computation)
  - PyMC3, StatsModels, SciPy (statistical software)

Introduction

Fitting a model is easy, cleaning the data, not so much

- Training a statistical/ML model is child's play

```
from CoolestLibraryInTown import NewestShiniestNN

model = NewestShiniestNN()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- "Garbage in, garbage out"

- Preprocessing messy data to a format that is amenable to modeling is the real challenge!

- You can preprocess your data using standard Python constructs: lists, loops and dictionaries
    - Impractical for very large/complex datasets (speed, readability, maintainability)

Introduction

Pandas to the rescue

- **Dataframe data structure optimized**
    - Selecting, slicing, indexing, filtering, reshaping
- **Seamless integration with stats/ML libraries and Jupyter**
- **Optimized for performance**
    - *Much, much* faster than vanilla Python code
    - **Loops are bad!**
    - Mostly written in Cython
- **Extensive documentation and community**
    - *"Yes, someone has solved that problem before."*

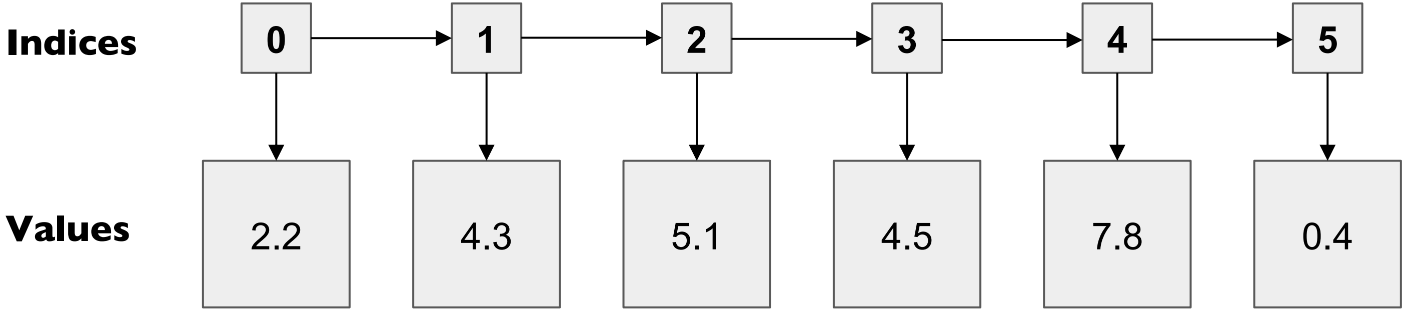
Disclaimer: You can do everything vanilla Python and loops. I can guarantee that it will be slower to write, and slower to run, once you know pandas.

Introduction
Overview

- **Pandas basics**
  - Series and dataframes
  - Data I/O
  - Indexing and querying
  - Quick plotting
- **Intermediary concepts**
  - Reshaping data (melt, stack, pivot)
  - Merging, joining
  - Split-apply-combine
- **Real-life example (if time allows)**
  - Preparing a machine learning dataset

Pandas basics (Series)

Pandas basics
Series



```
import pandas as pd

data = [2.2, 4.3, 5.1, 4.5, 7.8, 0.4]
series = pd.Series(data)

# or

data = ({0: 2.2, 1: 4.3, 2: 5.1,
         3: 4.5, 4: 7.8, 5: 0.4})
series = pd.Series(data)
```

Pandas basics

Series (slicing and selection)

- Much like standard Python lists, you may slice the data (same syntax as Python)

```
series[0:3] # retrieves first three elements
```

- The .iloc[] syntax more flexible however, as the elements do not have to be contiguous in the series

```
series.iloc[0:3] # equivalent to above
series.iloc[[0,1,2]] # equivalent

series.iloc[[0,2,5]]
```

- The .loc[] syntax retrieve elements using the name of the index

```
# retrieve element with index Francois
series.loc[“Francois”]
```

Pandas basics

Series (filtering)

- You may use equalities/inequalities to retrieve elements satisfying it

```
series[series > 5] # gets all entries > 5
```

- You may combine such operations with bitwise operators:
    - "&" (and)
    - "|" (or)
    - "~" (not)

```
series[(series < 5) & ~(series <= 1)]
```

Pandas basics

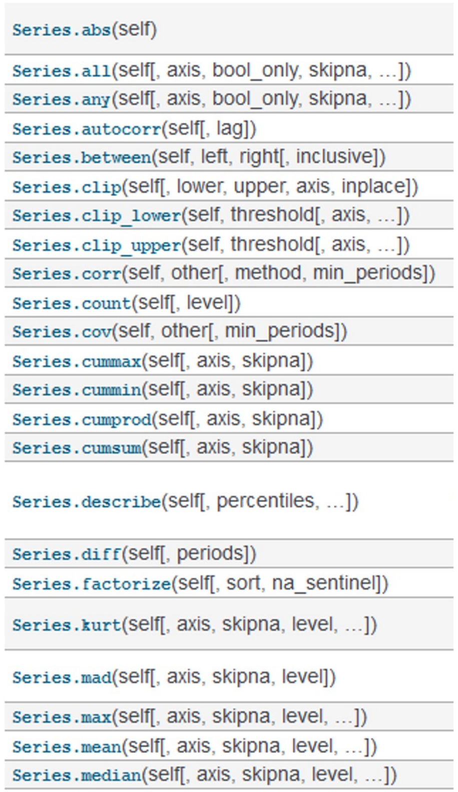
Series (operations on series)

- You can add, subtract, multiply, divide and exponentiate series elements
    - Returns a new series

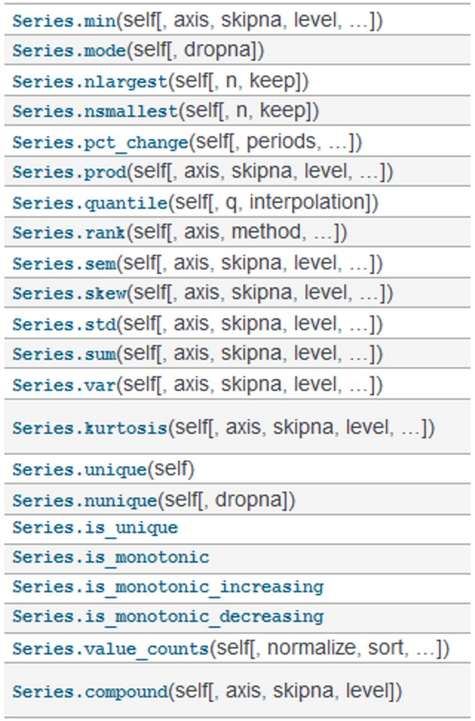
```
# multiply by a constant
series*2

# multiply by another series of same dimension
series * series2
```

Pandas basics

Series (descriptive statistics)



Series.abs(self)
Series.all(self[, axis, bool_only, skipna, …])
Series.any(self[, axis, bool_only, skipna, …])
Series.autocorr(self[, lag])
Series.between(self, left, right[, inclusive])
Series.clip(self[, lower, upper, axis, inplace])
Series.clip_lower(self, threshold[, axis, …])
Series.clip_upper(self, threshold[, axis, …])
Series.corr(self, other[, method, min_periods])
Series.count(self[, level])
Series.cov(self, other[, min_periods])
Series.cummax(self[, axis, skipna])
Series.cummin(self[, axis, skipna])
Series.cumprod(self[, axis, skipna])
Series.cumsum(self[, axis, skipna])
Series.describe(self[, percentiles, …])
Series.diff(self[, periods])
Series.factorize(self[, sort, na_sentinel])
Series.kurt(self[, axis, skipna, level, …])
Series.mad(self[, axis, skipna, level])
Series.max(self[, axis, skipna, level, …])
Series.mean(self[, axis, skipna, level, …])
Series.median(self[, axis, skipna, level, …])



Series.min(self[, axis, skipna, level, …])
Series.mode(self[, dropna])
Series.nlargest(self[, n, keep])
Series.nsmallest(self[, n, keep])
Series.pct_change(self[, periods, …])
Series.prod(self[, axis, skipna, level, …])
Series.quantile(self[, q, interpolation])
Series.rank(self[, axis, method, …])
Series.sem(self[, axis, skipna, level, …])
Series.skew(self[, axis, skipna, level, …])
Series.std(self[, axis, skipna, level, …])
Series.sum(self[, axis, skipna, level, …])
Series.var(self[, axis, skipna, level, …])
Series.kurtosis(self[, axis, skipna, level, …])
Series.unique(self)
Series.nunique(self[, dropna])
Series.is_unique
Series.is_monotonic
Series.is_monotonic_increasing
Series.is_monotonic_decreasing
Series.value_counts(self[, normalize, sort, …])
Series.compound(self[, axis, skipna, level])

Pandas basics

Series (handling missing data)

- You may drop NaN entries

```
series = series.dropna()
```

- You may specify a value

```
series = series.fillna(0)
```

- You may interpolate in-between values

```
# fill all consecutive NaNs with the first
preceding non-NaN value
series = series.interpolate(method=“pad”)

# midpoint between contiguous values
series = series.interpolate(method=“linear”)
```

Pandas basics

Exercise

Scenario 1

Let us imagine that we are pediatricians preparing a yearly report on diabetes. We are reviewing charts and are interested in:

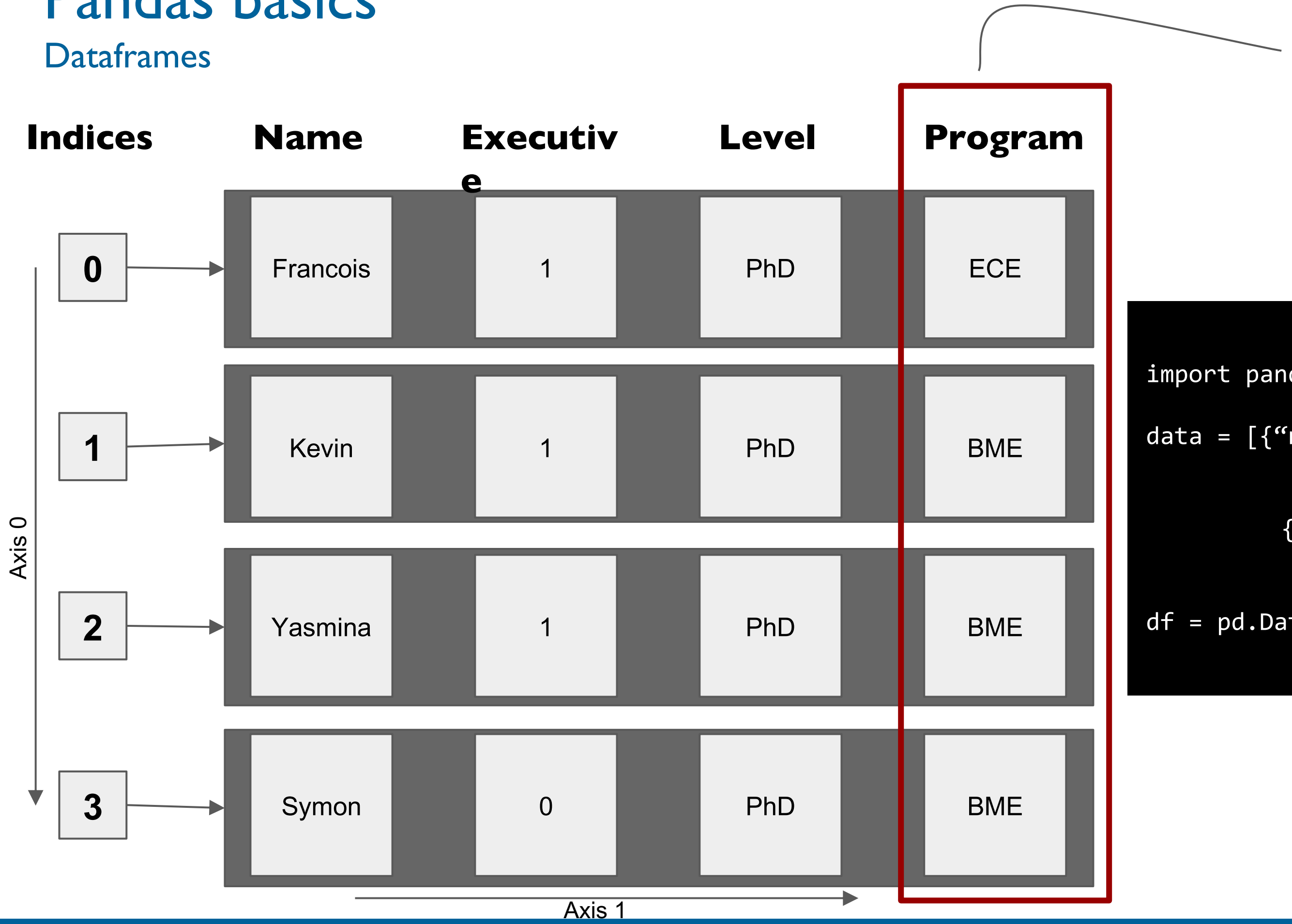
1) Identifying patients with pre-diabetes and diabetes
2) Counting these patients
3) Finding the 5 patients with the highest fasting sugar levels
4) Find the patients with missing values
5) Computing the z-score of the sugar level of each children to give context to parents

Download this and load the contents into Collab.

Pandas basics (Dataframes)

Pandas basics

Dataframes



Each column is... a Pandas series!
And the same operations apply!

```
import pandas as pd

data = [{“name”: “Francois”, “executive”: 1,
         “level”: “PhD”, “program”: “ECE”},
         …,
        {“name”: “Symon”, “executive”: 0,
          “level”: “PhD”, “program”: “BME”}]

df = pd.DataFrame(data)
```

Pandas basics

Dataframes (slicing and selection)

- You can easily get columns using the following syntax:

```
df[“name”] # retrieves the name column (series)
```

- The .iloc[] syntax still applies, except that columns are the second element in the square brackets, first element being rows

```
df.iloc[:,0:3] # by index

df.iloc[:,[“name”, “executive”]] # by name

df[[“name”, “executive”]] # by name (other way)
```

Pro-tip: Give columns meaningful names!

Pandas basics

Dataframes (querying/selecting data)

- A common operation is selecting data satisfying certain conditions.

```
df[(df.name != “Francois”) & (df.executive)]

df.query(“name != ‘Francois’ and executive”) # equivalent
```

- You can sample random rows in your dataset with/without replacement. That can be useful for bootstrapping!

```
df.sample(10, replace=True)
```

Pandas basics

Dataframes (generating new columns from existing columns)

- You can create new columns by applying operations to other columns

```
df[“new_column”] = df[“old_column”] * df[“some_other_column”]
```

- Often, you’ll want to apply some custom function to transform your data. This is where the *apply()* method comes in handy.

```
def age_to_label(age):
            return “adult” if age >= 18 else “child”

df[“description”] = df[“age”].apply(age_to_label)

# or equivalently
df[“description”] = (df[“age”].apply(
                        lambda age: “adult” if age >= 18 else “child”
                    ))
```

Pandas basics

Dataframes (File I/O)

- Often, datasets will be provided to you in a file -- Excel, csv, tsv, etc.

```
df = pd.read_csv(“my_dataset.tsv”, sep=“\t”)
```

- Of course, there is also excellent support to save a Pandas dataframe.

```
df.to_csv(“my_dataset.tsv”, sep=“\t”, index=False)
```

- There is also support for importing from an SQL database, JSON files, etc. Look up the methods here.

Pandas basics

Dataframes (Other useful functions)

- It might happen that you have duplicate rows that are totally or partially duplicated.

```
# drop identical rows
df.drop_duplicates(keep=False) # keep none of the entries that appear >1 time

# drop rows with duplicate values for columns name and sex
df.drop_duplicates([“name”, “sex”], keep=“first”) # keep first instance of duplicate

# identify duplicated entries
df[df.duplicated()]
```

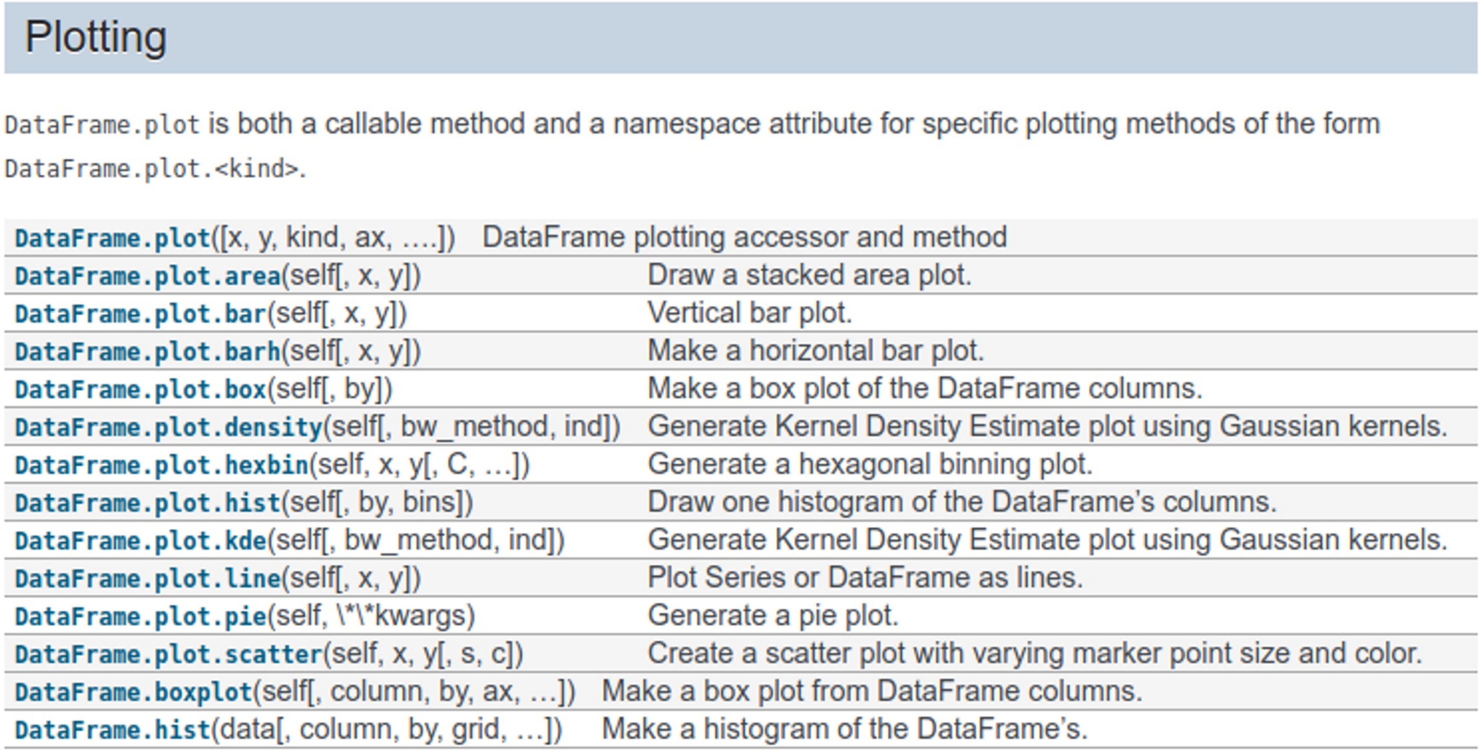
- You can replace a value throughout the entire dataframe (**note**: you can chain methods).

```
df.replace(“positive”, 1).replace(“negative”, 0)
```

- Other useful functions include: *head()*, *shape()*, *sort_values()*, *to_json()*, to_latex(), isin(), etc.
- Quick plotting is also possible (compatibility with matplotlib and seaborn)!

Pandas basics

Plotting



Plotting

`DataFrame.plot` is both a callable method and a namespace attribute for specific plotting methods of the form `DataFrame.plot.<kind>`.

`DataFrame.plot`([x, y, kind, ax, ….])	DataFrame plotting accessor and method
`DataFrame.plot.area`(self[, x, y])	Draw a stacked area plot.
`DataFrame.plot.bar`(self[, x, y])	Vertical bar plot.
`DataFrame.plot.barh`(self[, x, y])	Make a horizontal bar plot.
`DataFrame.plot.box`(self[, by])	Make a box plot of the DataFrame columns.
`DataFrame.plot.density`(self[, bw_method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
`DataFrame.plot.hexbin`(self, x, y[, C, …])	Generate a hexagonal binning plot.
`DataFrame.plot.hist`(self[, by, bins])	Draw one histogram of the DataFrame's columns.
`DataFrame.plot.kde`(self[, bw_method, ind])	Generate Kernel Density Estimate plot using Gaussian kernels.
`DataFrame.plot.line`(self[, x, y])	Plot Series or DataFrame as lines.
`DataFrame.plot.pie`(self, **kwargs)	Generate a pie plot.
`DataFrame.plot.scatter`(self, x, y[, s, c])	Create a scatter plot with varying marker point size and color.
`DataFrame.boxplot`(self[, column, by, ax, …])	Make a box plot from DataFrame columns.
`DataFrame.hist`(data[, column, by, grid, …])	Make a histogram of the DataFrame's.

Pandas basics

Exercise

Scenario 2

We are statisticians interested in studying the prevalence of anxiety. We designed a questionnaire and collected 9905 responses. Examine the data and answer the following questions:

1) Import the dataset.
2) Many participants submitted the questionnaire more than once by double clicking the submit button.
    - How many unique patients filled out the questionnaire?
    - How many responses are duplicates?
    - Drop the duplicates.
3) How many participants failed to answer at least one of the questions?
4) What is the most common form of anxiety?
5) Create a column for the "total anxiety" score by summing (out of 30) the 3 scores (OCD, panic, social anxiety) of each participants.
    - What is the mean score?
6) Create a column specifying whether the patient is a child, teenager, or an adult.
7) What are the counts of children, teenagers and adults with anxiety? (There is a function to find this: see slide 14)
8) What is the correlation coefficient between the total anxiety score and the age? (See slide 14)

Download this and load the contents into Collab.

Pandas basics (Intermediary concepts)

Intermediary concepts

Reshaping - **Melting**

- Sometimes, the data will be provided to you in a format that is inconvenient. A record (row) contains multiple measurements.

patient_id	name	bmi	blood_sugar
223	Jeremy	24.3	97.5
481	Karen	23.5	89.4

```
df.melt(
        id_vars=[“patient_id”, “name”],
        var_name=“value”
)
```

patient_id	name	variable	value
223	Jeremy	bmi	24.3
481	Karen	bmi	23.5
223	Jeremy	blood_sugar	97.5
481	Karen	blood_sugar	89.4

Intermediary concepts

Reshaping - **Pivoting**

- Pivoting allows you to turn certain column values into columns, essentially the opposite of melting, in this situation.

patient_id	name	variable	value
223	Jeremy	bmi	24.3
481	Karen	bmi	23.5
223	Jeremy	blood_sugar	97.5
481	Karen	blood_sugar	89.4

```
df.pivot_table(
      index=[“patient_id”, “name”],
      columns=“variable”,
      values= “value”
)
```

		bmi	blood_sugar
patient_id	name		
223	Jeremy	24.3	97.5
481	Karen	23.5	89.4

Indices, not observations!

You can turn these into observations with *reset_index()*.

Intermediary concepts

Reshaping - **Stacking**

- Let’s consider a scenario where you have two groups: cancer patients and healthy patients. For each group you measure twice the concentration of WBCs and C-reactive protein (CRP).

indices, not observations

		wbc	crp
sex	**group**		
male	cancer	15,986	7.2
	healthy	5,153	2.4
female	cancer	17,357	6.6
	healthy	7,454	1.5

```
df.stack()
```

```
df.unstack()
```

sex	group		
male	cancer	wbc	15,986
	cancer	crp	7.2
	healthy	wbc	5,153
	healthy	crp	2.4
female	cancer	wbc	17,357
	cancer	crp	6.6

...

Intermediary concepts

Reshaping - **Concatenating dataframes**

- You may have separate datasets with the <u>same type of data (columns)</u> that you want to join together. To do so, you may use the pd.concat() function:

```
# concatenate rows
pd.concat([df1, df2], axis=0)

# concatenate columns
pd.concat([df1, df2], axis=1)
```

- Useful, but not always what you need…

Careful!

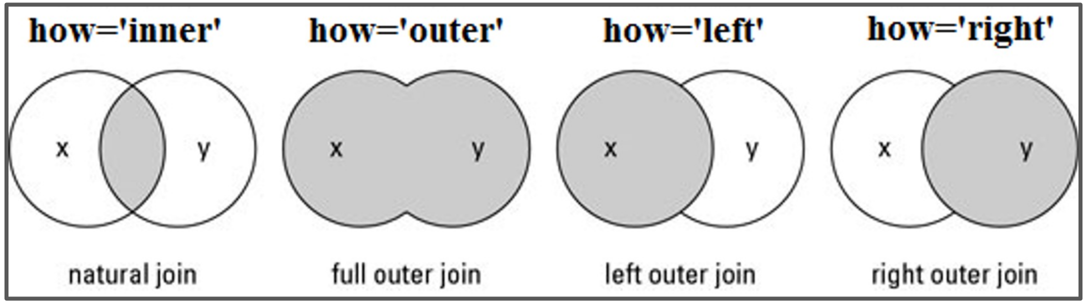
Columns that only appear in one dataframe will be filled with NaNs in the other without warning.

Intermediary concepts

Reshaping - **Merging (or joining) dataframes**

- Merging is an operation similar to SQL’s *join* statement -- allows us to keep compact files.

```
df1.merge(
  df2,
  on=[“patient_id”], # merge rows that share the patient_id value
  how=“left” # keep all rows from df1
)
```

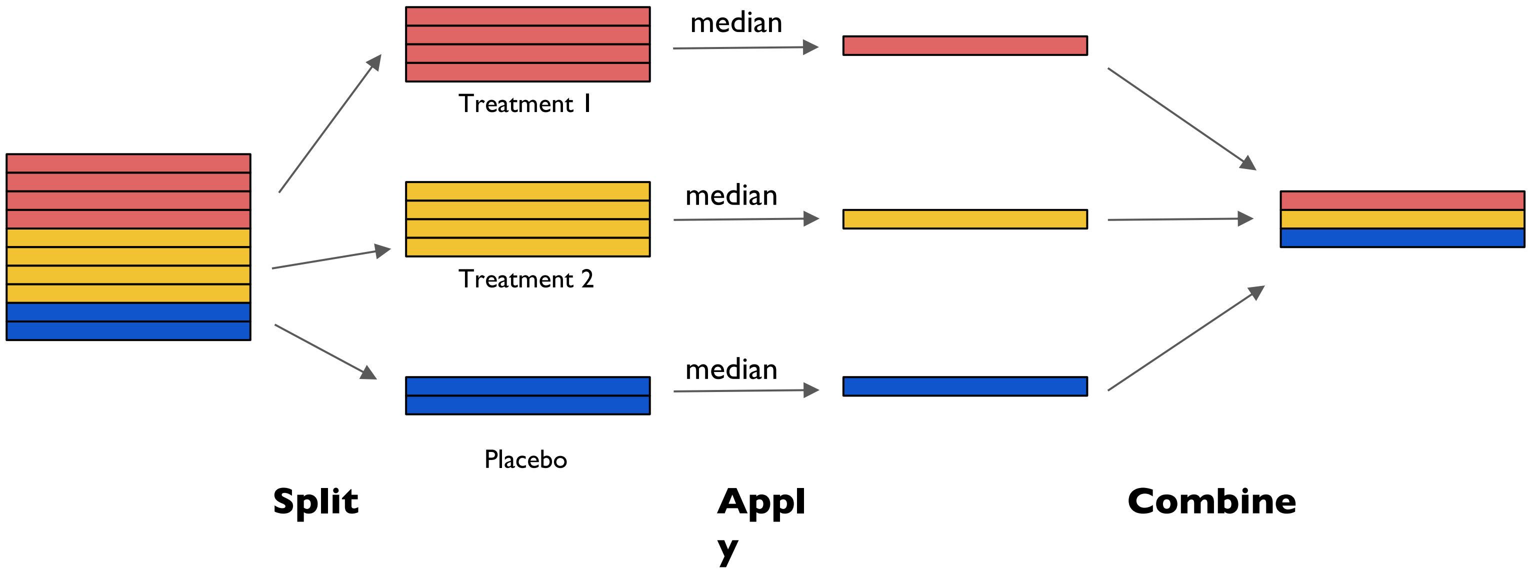


Source: http://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/

Intermediary concepts

Grouping - **Split-apply-combine**

- In many datasets, observations will be "groupable"
  - Interested in computing certain statistics for specific groups of observations
  - e.g. gender, age group, occupation, medical condition, treatment1/treatment2/control, etc.

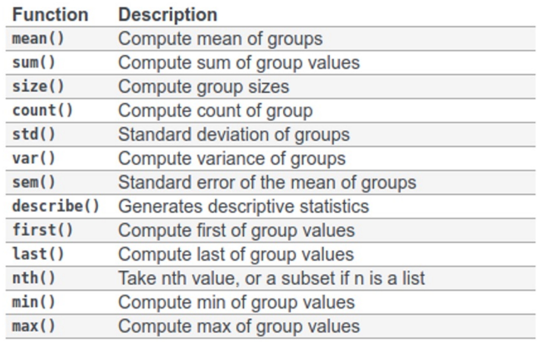


Intermediary concepts

Grouping - **Split-apply-combine**

- The *apply* must transform the observations into a single number (*aggregation*).
    - Possible to do more than one at a time.

```
# generate a statistical summary of the observations grouped by treatment
df.groupby(“treatment”).describe()

# apply multiple aggregation operations
df.groupby(“treatment”).agg([np.sum, np.mean, np.mode])
```



Function	Description
mean()	Compute mean of groups
sum()	Compute sum of group values
size()	Compute group sizes
count()	Compute count of group
std()	Standard deviation of groups
var()	Compute variance of groups
sem()	Standard error of the mean of groups
describe()	Generates descriptive statistics
first()	Compute first of group values
last()	Compute last of group values
nth()	Take nth value, or a subset if n is a list
min()	Compute min of group values
max()	Compute max of group values

Pandas basics

Exercise

Scenario 3

We are running a clinical trial for a disease that affects the elderly. We are testing two treatments: treatment 1 and treatment 2. Of course, this trial is placebo-controlled.

1) Import the datasets, and create a new dataframe by merging the trial data dataframe with the patient information dataframe. We want to associate every patient to his/her age, city and email address.
2) How many participants:
   - Are in the disease and control group?
   - Receive the placebo, treatment 1, treatment 2?
   - Have the disease and get treatment 1, 2 and placebo? Same for controls.
3) What is the mean survival of the participants in the control group? In the disease group?
4) What is the median survival of survival of participants in the disease group for the three treatments? Is any of the treatments clearly effective (assuming we don't need to test for significance)?
5) Plot a bar chart showing the median survival of the 6 different groups. (**Hint:** Use the *.plot.bar()* plotting method.)

Download this and load the contents into Collab.

Take-home message

- Pandas is ***fast***
  - To write: one-liners
  - To run: no naive looping (vectorized operations)

- Dataframe are an intuitive way to work with data
  - Compatibility with relational databases

- Easy to import and export

- Excellent documentation

- Superb compatibility with Jupyter Notebooks

Excellent cheat sheet
here.

Merci!
Thank you!

francoischarih@sce.carleton.ca
http://charih.ca