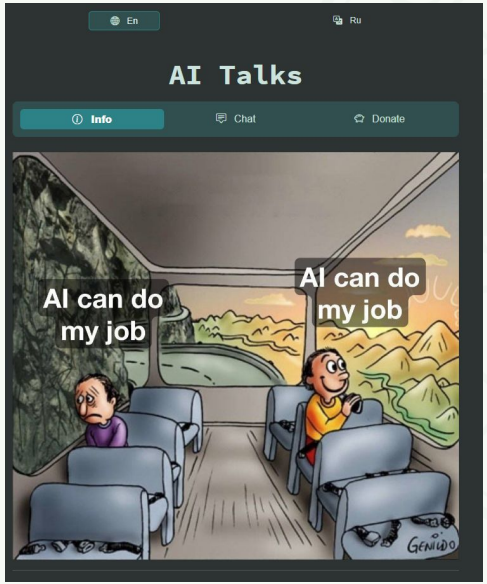
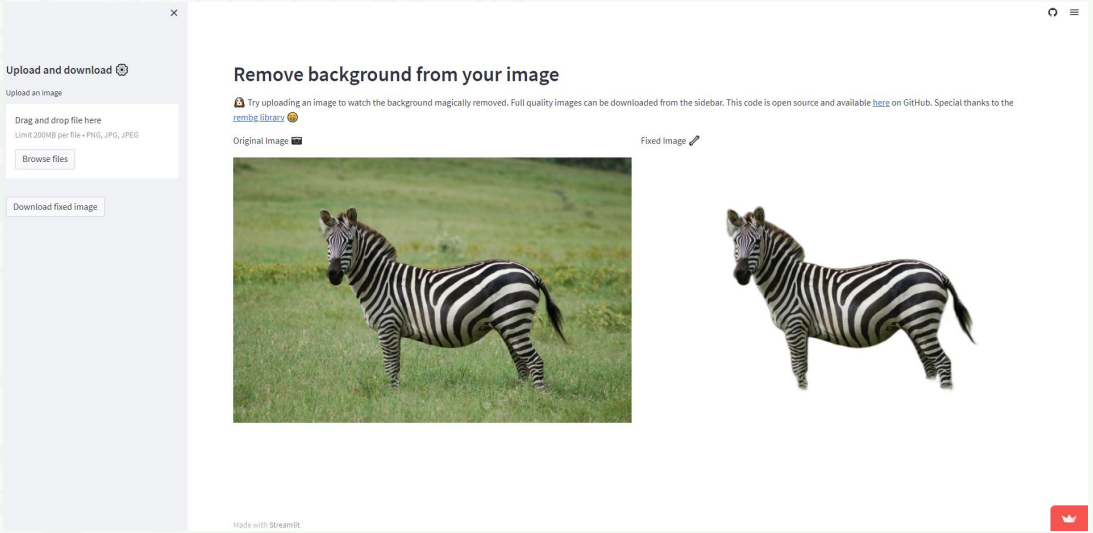
Introduction to Pandas

Prof. Murillo
Computational Mathematics, Science and Engineering
Michigan State University

Streamlit

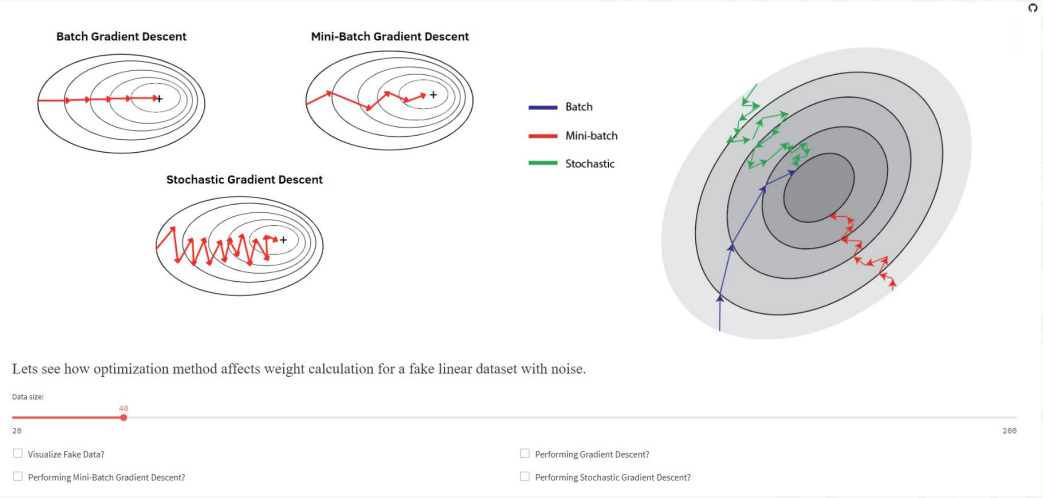
- Open-source python library for developing web applications

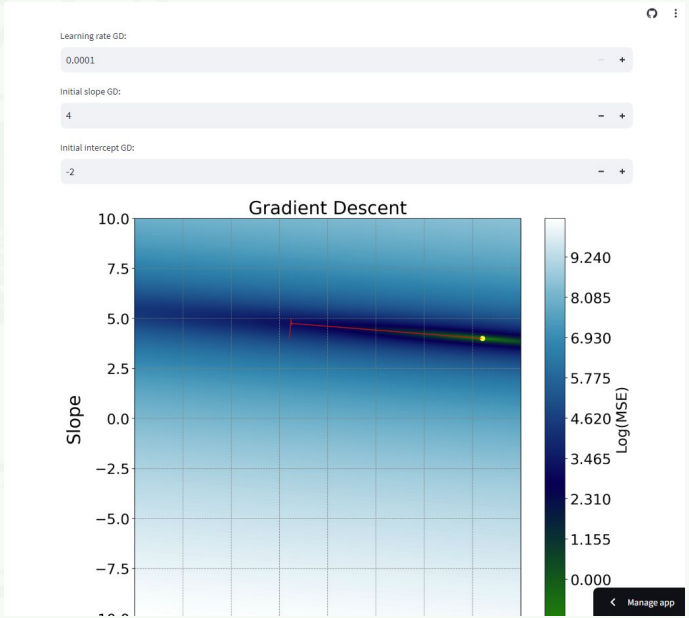




Streamlit

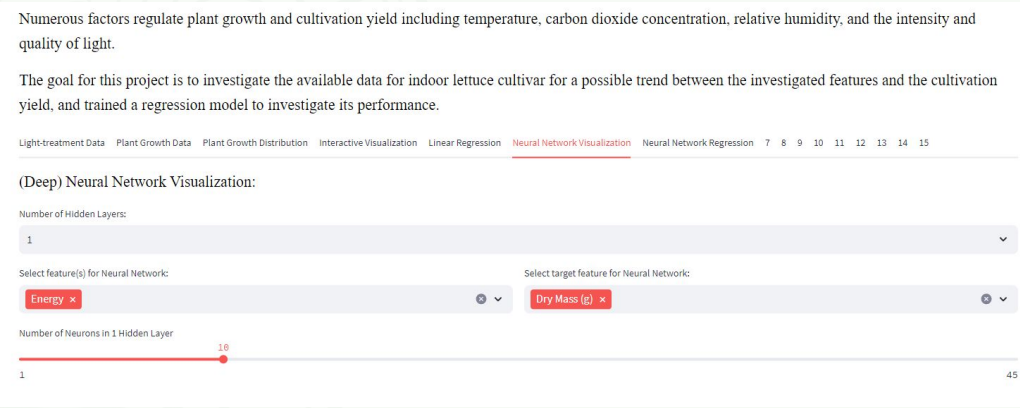
- Event-based program which is ideal for user interaction





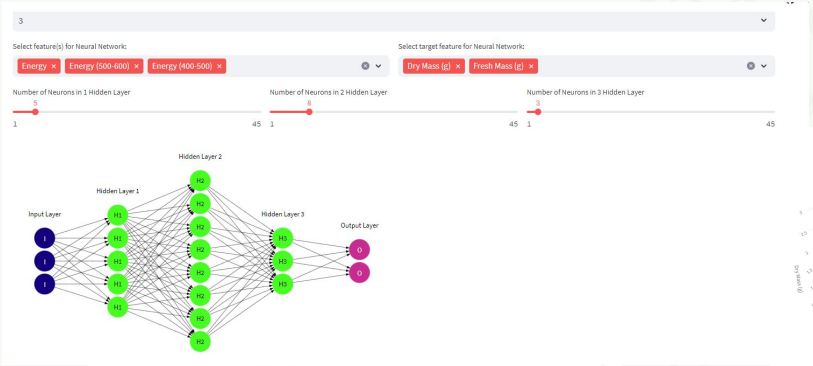
Streamlit

- Powerful customization capabilities to tailor the appearance of webapp



Streamlit

- Integration with other visualization libraries



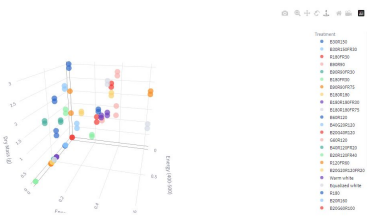
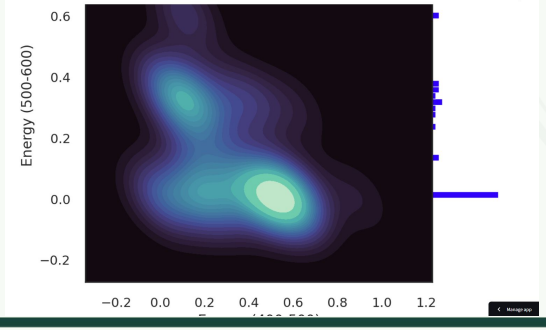
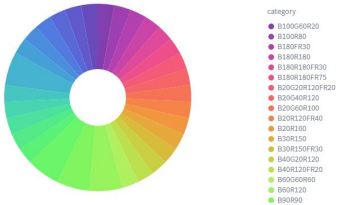


Fig. 5: Heatmap of lettuce mass with respect to a feature.



Plant Growth Distribution:

Next, distribution of plant growth data are presented based on the species or the light-treatments used for indoor-cultivation of lettuce.



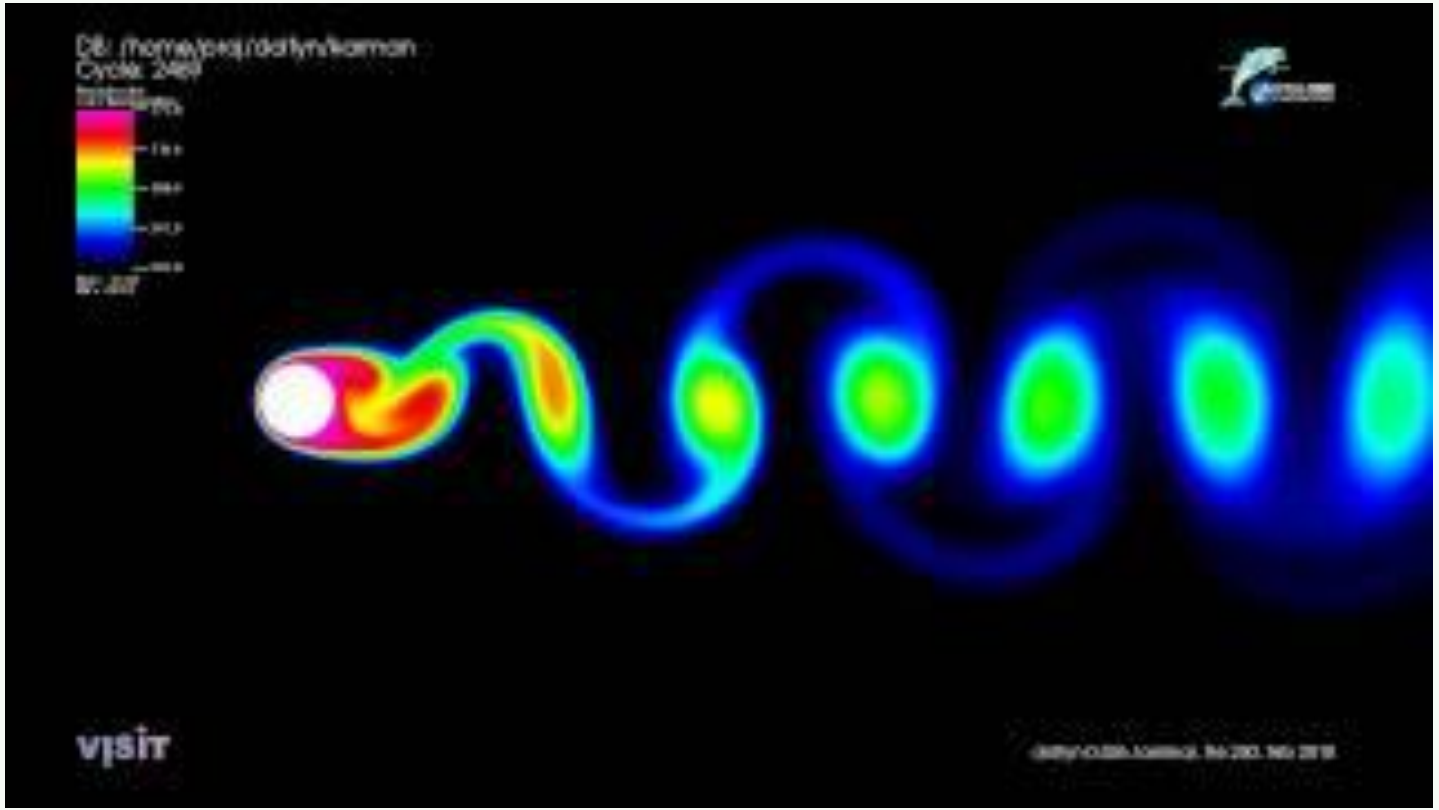
Streamlit Examples

- https://second-part---data-science-project-bj6pdddikzwtrr9rdgys6r.streamlit.app/
- https://decisiontreerandomforest-gqvgc5eavkrekfodkmehwr.streamlit.app/
- https://mahyarabedi93-classification-webapp-rmlrag.streamlit.app/

Streamlit Examples

- https://training-models-hysvxngrybeuauokkakekd.streamlit.app/
- https://training-models-fvha5acdjpowm4cfvid7qr.streamlit.app/
- https://trainingmodelii-esvduwbpts6tnbkf7yyjsq.streamlit.app/

Karman Vortex Street

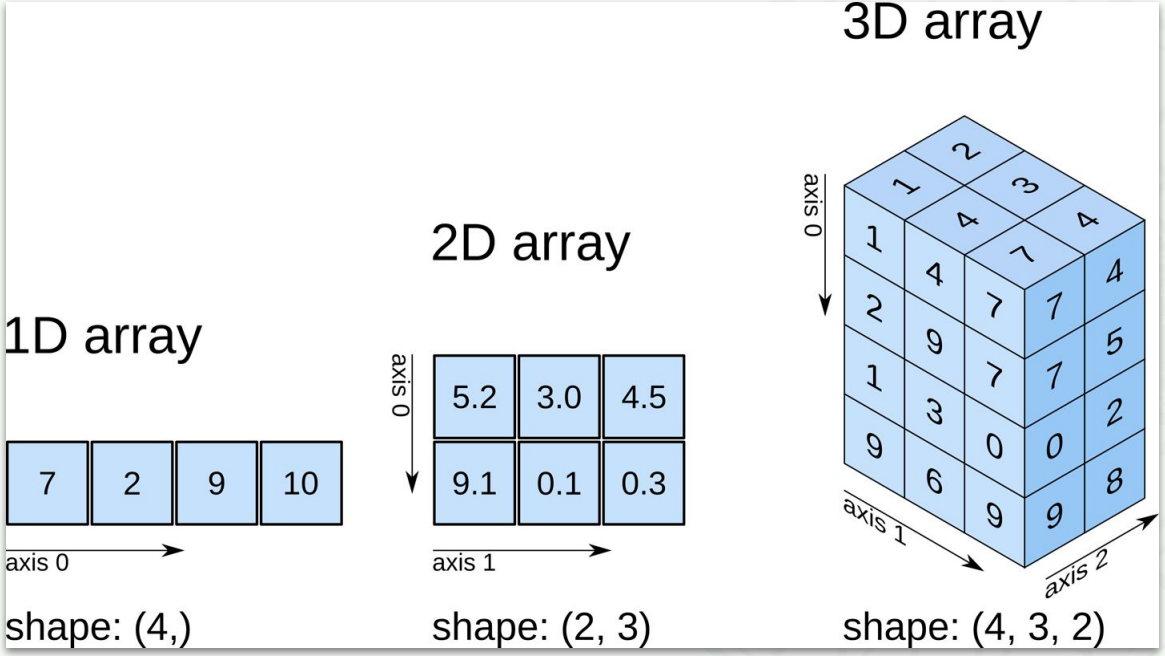


Karman Vortex Street: Reality

Overview

- data models: how is our data organized?
- organizing our data in Python: arrays, dictionaries and Pandas
- structure/logic of `Series` and `DataFrame` objects and `Index`
- Wed: using Pandas for wrangling, IDA, EDA

From Last Week: NumPy Arrays



NumPy provides the `ndarray` object, which allows you to create arrays of any dimension.

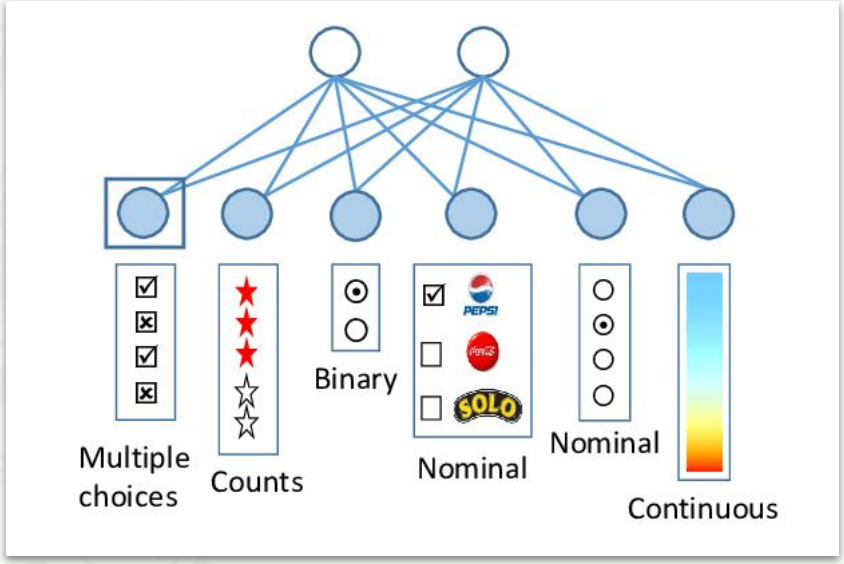
For many tasks, this is a very convenient way to organize data. Many libraries will assume your data is organized this way.

Are there other ways to organize data?

NumPy Won't Always Work

Data is a collection of discrete objects:

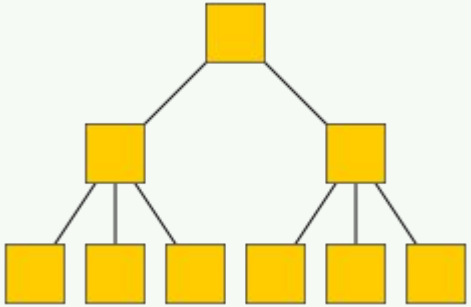
- numbers
- words
- facts
- objects
- measurements
- observations
- descriptions
- and so on....



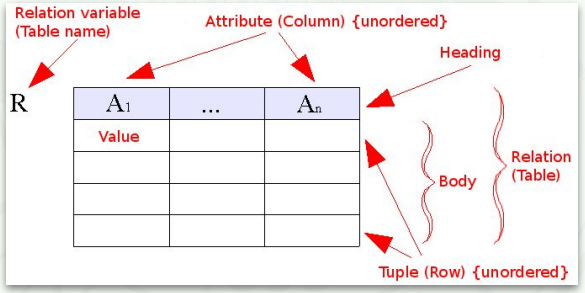
Data Model

A **data model** is an organization principle for the elements of a dataset.

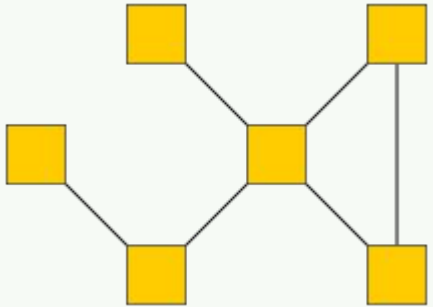
Some examples:



hierarchical



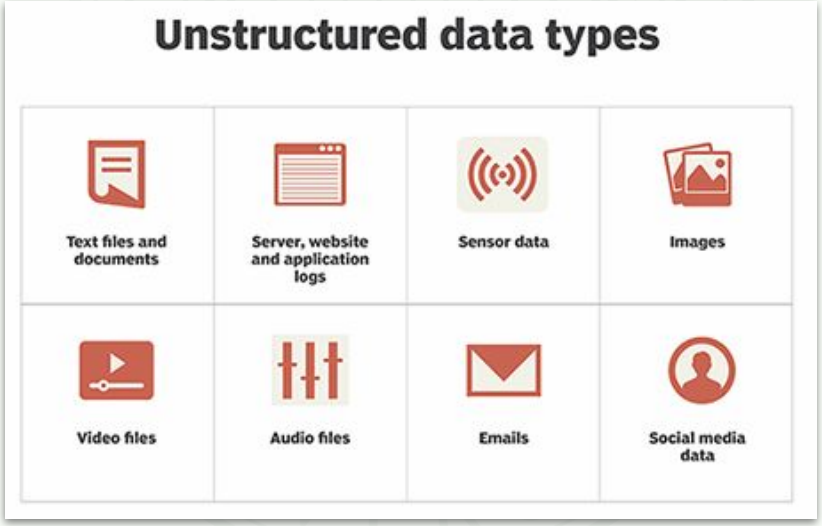
relational



network

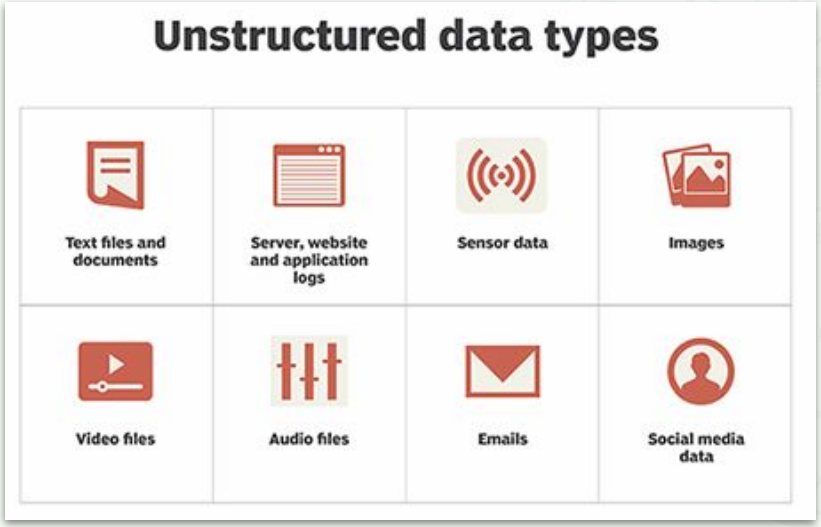
Structured and Unstructured Data

Unstructured data does not have a data model.



Structured and Unstructured Data

Unstructured data does not have a data model.



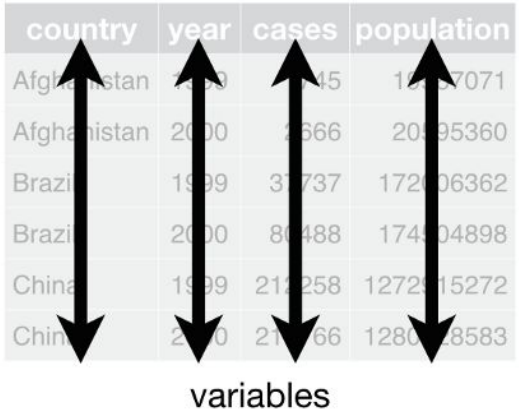
Structured data does have a data model.

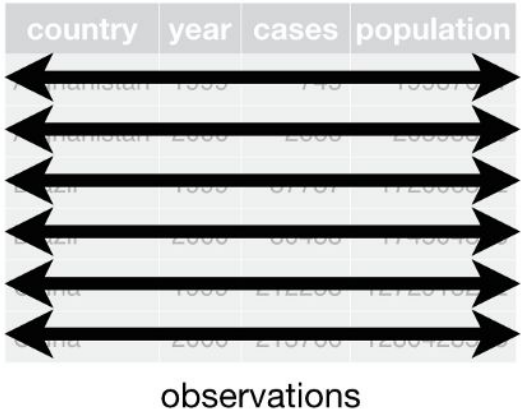
Structured data

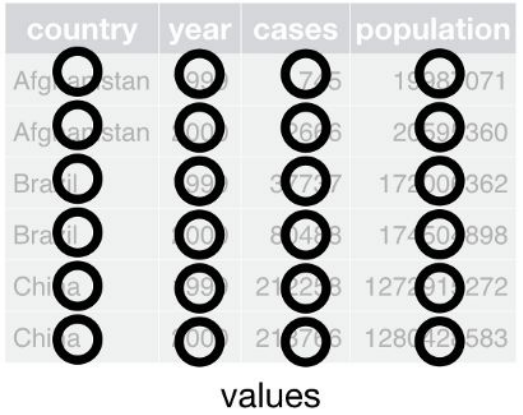
ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

Tidy Data

Tidy data is defined as an organization or display of data such that columns are features and rows are samples.



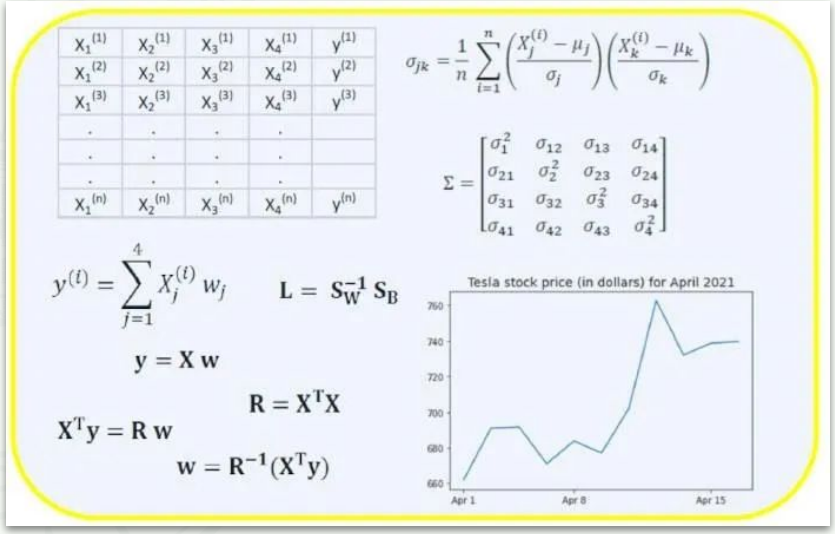




Peek Into Our Future: Data Matrix (Linear Algebra)

The **data matrix** is defined as a mathematical (i.e., linear algebra) matrix containing data organized in a tidy way.

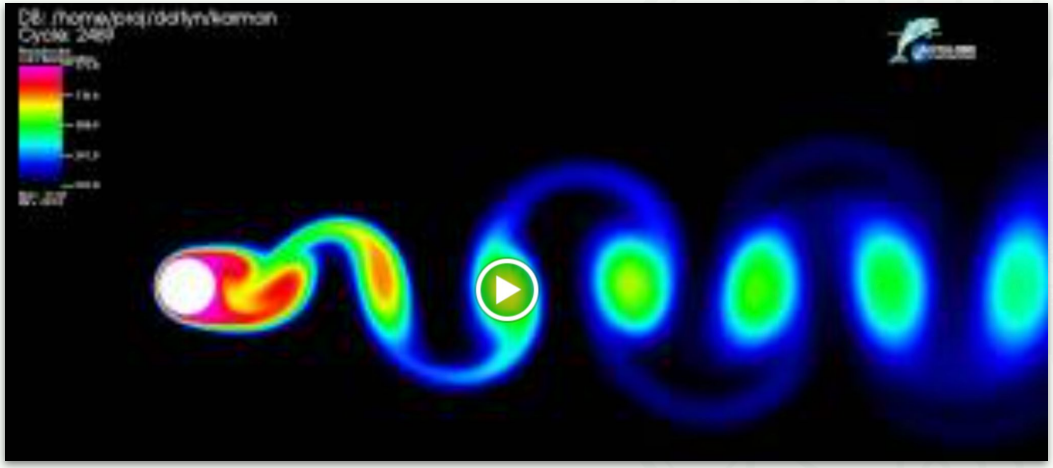
$X_1^{(1)}$	$X_2^{(1)}$	$X_3^{(1)}$	$X_4^{(1)}$
$X_1^{(2)}$	$X_2^{(2)}$	$X_3^{(2)}$	$X_4^{(2)}$
$X_1^{(3)}$	$X_2^{(3)}$	$X_3^{(3)}$	$X_4^{(3)}$
.	.	.	.
.	.	.	.
.	.	.	.
$X_1^{(n)}$	$X_2^{(n)}$	$X_3^{(n)}$	$X_4^{(n)}$



The data matrix will be covered in great detail later in the semester!

Spatiotemporal Data

Suppose we have data that varies in **space** and **time**.



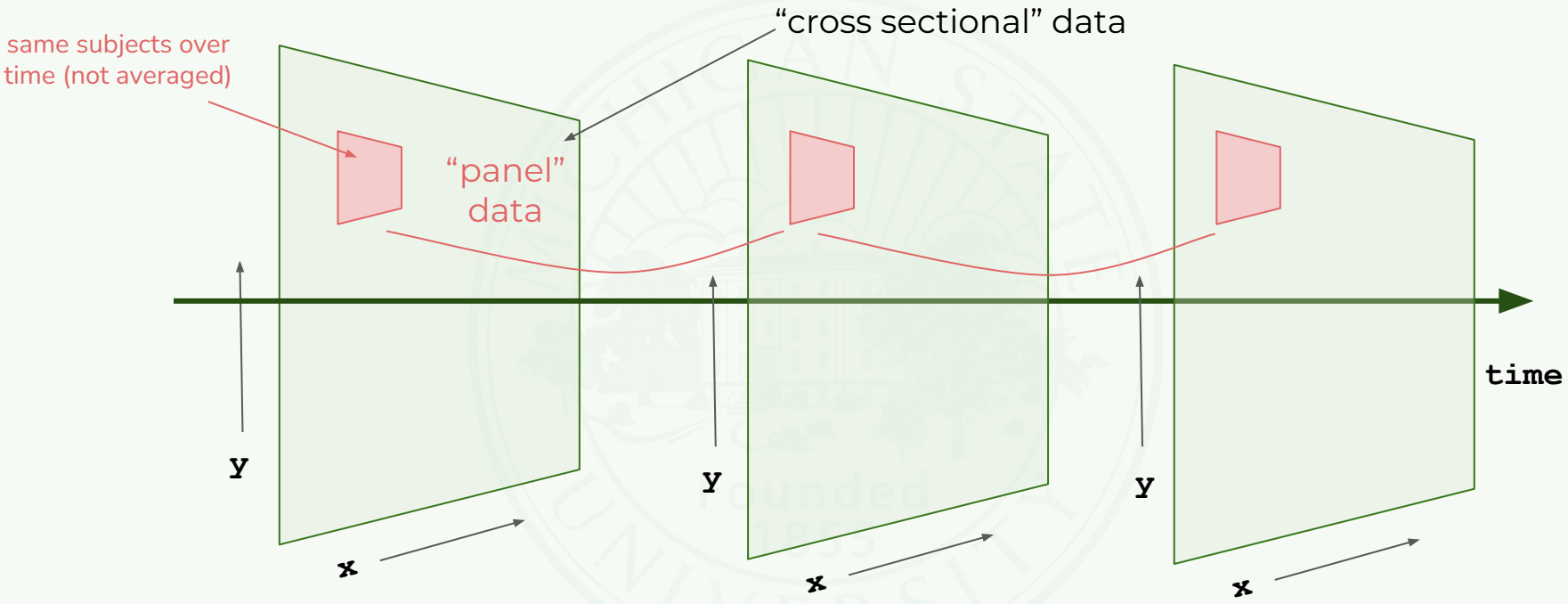
Here, x is the spatial data. Each column is a snapshot in time.

$$X = \begin{bmatrix} | & | & | \\ x(t_1) & x(t_2) & x(t_3) \\ | & | & | \end{bmatrix}$$

$$XX^T = \begin{bmatrix} | & | & | \\ x(t_1) & x(t_2) & x(t_3) \\ | & | & | \end{bmatrix} \begin{bmatrix} -x(t_1)- \\ -x(t_2)- \\ -x(t_3)- \end{bmatrix}$$

This yields the spatial covariance averaged over time.

Panel Data



"longitudinal" (time series) data

How would we invent a simple “data matrix” structure in Python?

Probably: *with a dictionary*.

state	color	food	age	height	score
NY	blue	Steak	30	165	4.6
TX	green	Lamb	2	70	8.3
FL	red	Mango	12	120	9.0
AL	white	Apple	4	80	3.3
AK	gray	Cheese	32	180	1.8
TX	black	Melon	33	172	9.5
TX	red	Beans	69	150	2.2

```
data_dict = {'state': ['NY', 'TX', 'FL', 'AL', 'AK', 'TX', 'TX'],
             'color': ["blue", "green", "red", "white", "gray", "black", "red"],
             'food': ["Steak", "Lamb", "Mango", "Apple", "Cheese", "Melon", "Beans"],
             "age": [30, 2, 12, 4, 32, 33, 69],
             "height": [165, 70, 120, 80, 180, 172, 150],
             "score": [4.6, 8.3, 9.0, 3.3, 1.8, 9.5, 2.2]}
```

```
data_dict
```

```
{'state': ['NY', 'TX', 'FL', 'AL', 'AK', 'TX', 'TX'],
 'color': ['blue', 'green', 'red', 'white', 'gray', 'black', 'red'],
 'food': ['Steak', 'Lamb', 'Mango', 'Apple', 'Cheese', 'Melon', 'Beans'],
 'age': [30, 2, 12, 4, 32, 33, 69],
 'height': [165, 70, 120, 80, 180, 172, 150],
 'score': [4.6, 8.3, 9.0, 3.3, 1.8, 9.5, 2.2]}
```

We Can Go Far Beyond Putting Data Into Dictionaries

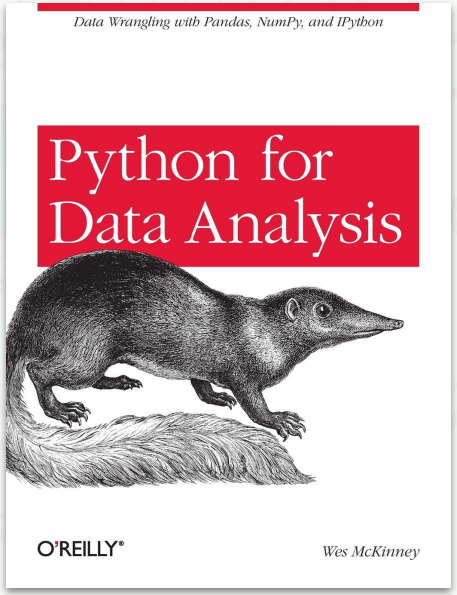
AQR CAPITAL MANAGEMENT

hedge fund

Wes McKinney

python™

Panel **Da**ta



pandas

Pandas: What is it?

Pandas is a software library for data science.

In particular, Pandas allows for data analysis and manipulations (e.g., munging and wrangling).

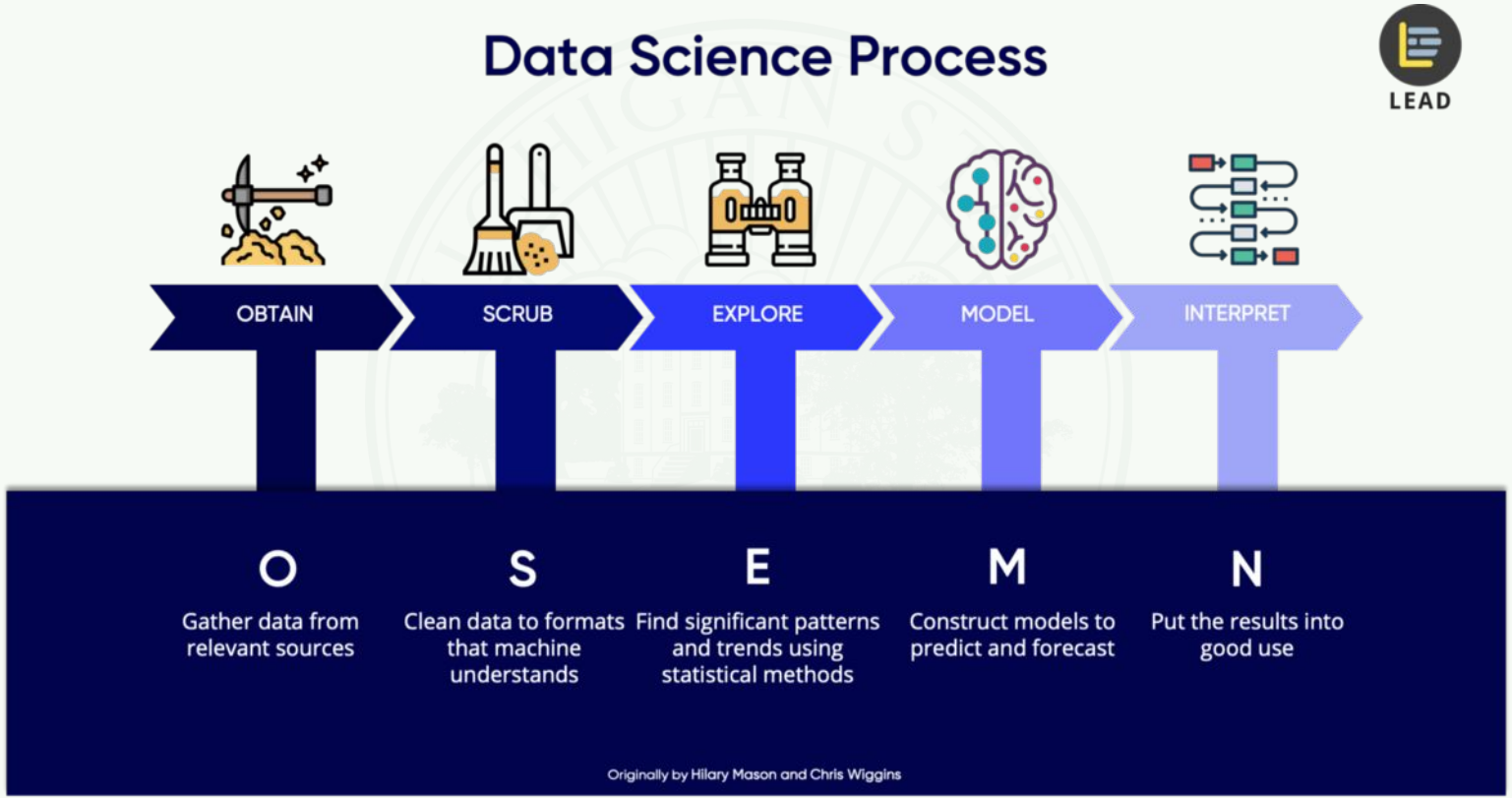
	state	color	food	age	height	score
Jane	NY	blue	Steak	30	165	4.6
Niko	TX	green	Lamb	2	70	8.3
Aaron	FL	red	Mango	12	120	9.0
Penelope	AL	white	Apple	4	80	3.3
Dean	AK	gray	Cheese	32	180	1.8
Christina	TX	black	Melon	33	172	9.5
Cornelia	TX	red	Beans	69	150	2.2

Just as NumPy brings the array object into Python, Pandas brings in its own object, the

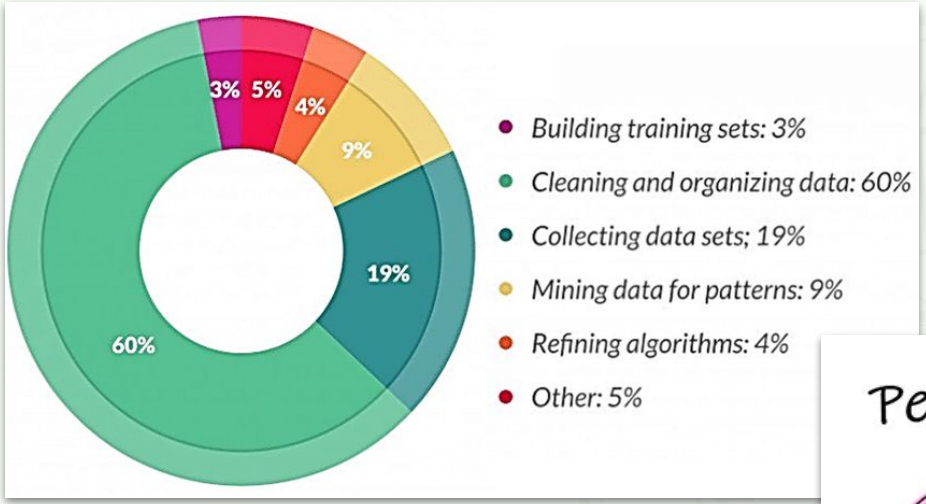
dataframe.

And, as with all other objects, there are a vast supply of methods.

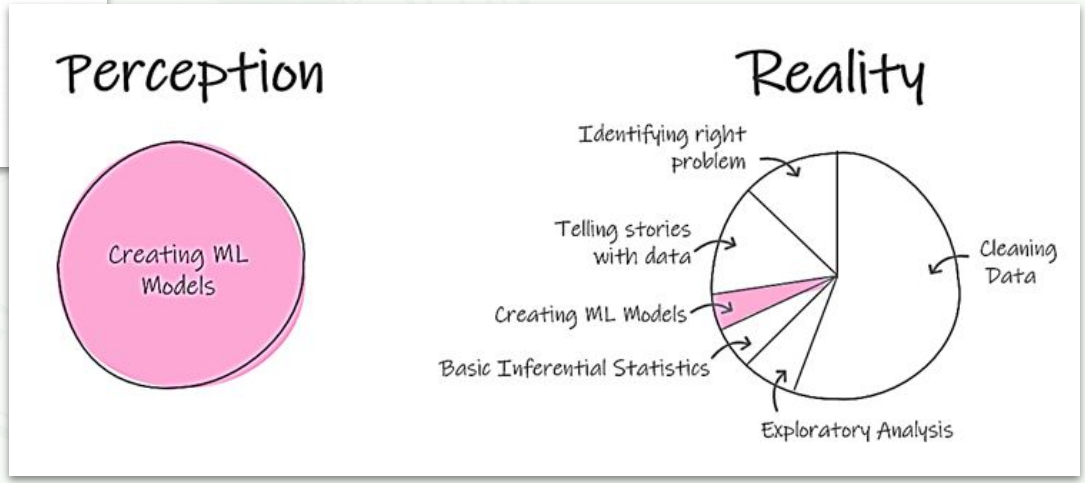
What is the Data Science Process/Workflow?



Where is the time spent?



pandas lives in the most time-intensive part of the data science workflow.



Pandas Basics

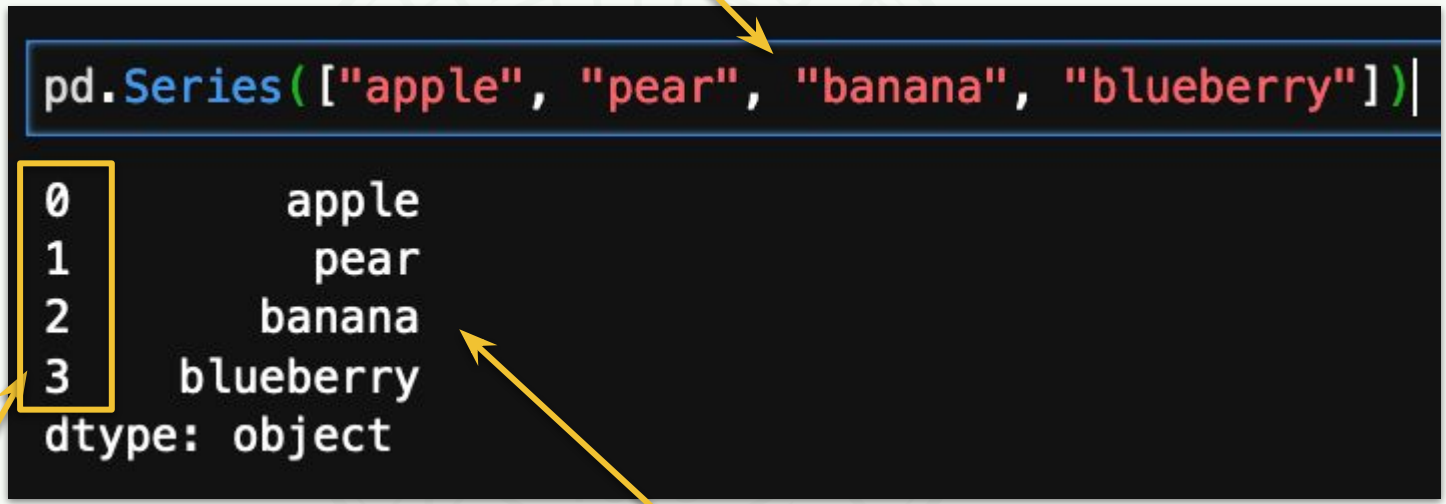
- Pandas is built from NumPy.
  - you don’t need to import NumPy,
  - but, you need NumPy on your computer (a good reason to use a distro like Anaconda)
- The standard way to import is:
  - `import pandas as pd`

Fundamental Pandas Objects

1. Index
2. Series
3. DataFrame
4. Panel Deprecated as of Pandas 0.25.0

Series

here, making the `Series` from a normal Python list



`Series` has added an `Index`

`Series` displays as a column

Series Attributes (Have No Parentheses)

```
my_series.shape
```

```
(4,)
```

```
my_series.head
```

```
<bound method NDFrame.head of 0        apple
1         pear
2       banana
3    blueberry
dtype: object>
```

```
my_series.values
```

```
array(['apple', 'pear', 'banana', 'blueberry'], dtype=object)
```

```
my_series.index
```

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

Attributes

`T`	Return the transpose, which is by definition self.
`array`	The ExtensionArray of the data backing this Series or Index.
`at`	Access a single value for a row/column label pair.
`attrs`	Dictionary of global attributes of this dataset.
`axes`	Return a list of the row axis labels.
`dtype`	Return the dtype object of the underlying data.
`dtypes`	Return the dtype object of the underlying data.
`flags`	Get the properties associated with this pandas object.
`hasnans`	Return True if there are any NaNs.
`iat`	Access a single value for a row/column pair by integer position.
`iloc`	Purely integer-location based indexing for selection by position.
`index`	The index (axis labels) of the Series.

Series Methods (Have Parentheses)

```
[34]: my_series.describe()

[34]: count          4
      unique         4
      top        apple
      freq           1
      dtype: object
```

Note: these are descriptive statistics of the *columns*.

Methods

`abs`()	Return a Series/DataFrame with absolute numeric value of each element.
`add`(other[, level, fill_value, axis])	Return Addition of series and other, element-wise (binary operator *add*).
`add_prefix`(prefix)	Prefix labels with string *prefix*.
`add_suffix`(suffix)	Suffix labels with string *suffix*.
`agg`([func, axis])	Aggregate using one or more operations over the specified axis.
`aggregate`([func, axis])	Aggregate using one or more operations over the specified axis.
`align`(other[, join, axis, level, copy, ...])	Align two objects on their axes with the specified join method.
`all`([axis, bool_only, skipna, level])	Return whether all elements are True, potentially over an axis.
`any`([axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
`append`(to_append[, ignore_index, ...])	(DEPRECATED) Concatenate two or more Series.
`apply`(func[, convert_dtype, args])	Invoke function on values of Series.
`argmax`([axis, skipna])	Return int position of the largest value in the Series.
`argmin`([axis, skipna])	Return int position of the smallest value in the Series.
`argsort`([axis, kind, order])	Return the integer indices that would sort the Series values.

The `Index` (noun!) allows you to index (verb!)

```
my_series

0         apple
1          pear
2        banana
3     blueberry
dtype: object
```

```
my_series.iloc[1]
```

```
'pear'
```

```
my_series.iloc[2]
```

```
'banana'
```

`loc` and `iloc`

access by name/label

access by integer/position

Adding Labels To `Index`, Using `loc`

```
my_series = pd.Series(["apple", "pear", "banana", "blueberry"], index=["1st", "2nd", "3rd", "4th"])
```

```
my_series
```

```
1st        apple
2nd         pear
3rd       banana
4th    blueberry
dtype: object
```

```
my_series.loc["4th"]
```

```
'blueberry'
```

Multi-Index

You can have multiple indices.

```
my_series = pd.Series(["apple", "pear", "banana", "blueberry"],
                      index=[["1st", "2nd", "3rd", "4th"],[0,1,2,3]])
```

```
my_series
```

```
1st  0         apple
2nd  1          pear
3rd  2        banana
4th  3     blueberry
dtype: object
```

Hierarchical Indexing

You can have *hierarchical* indices.

```
my_series = pd.Series(["apple", "pear", "banana", "blueberry"],
                      index=[["1st", "1st", "2nd", "2nd"],[0,1,0,1]])
```

```
my_series
```

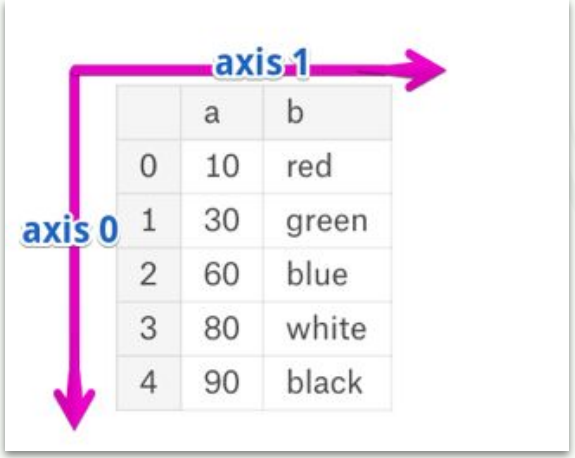
```
1st  0        apple
     1         pear
2nd  0       banana
     1    blueberry
dtype: object
```

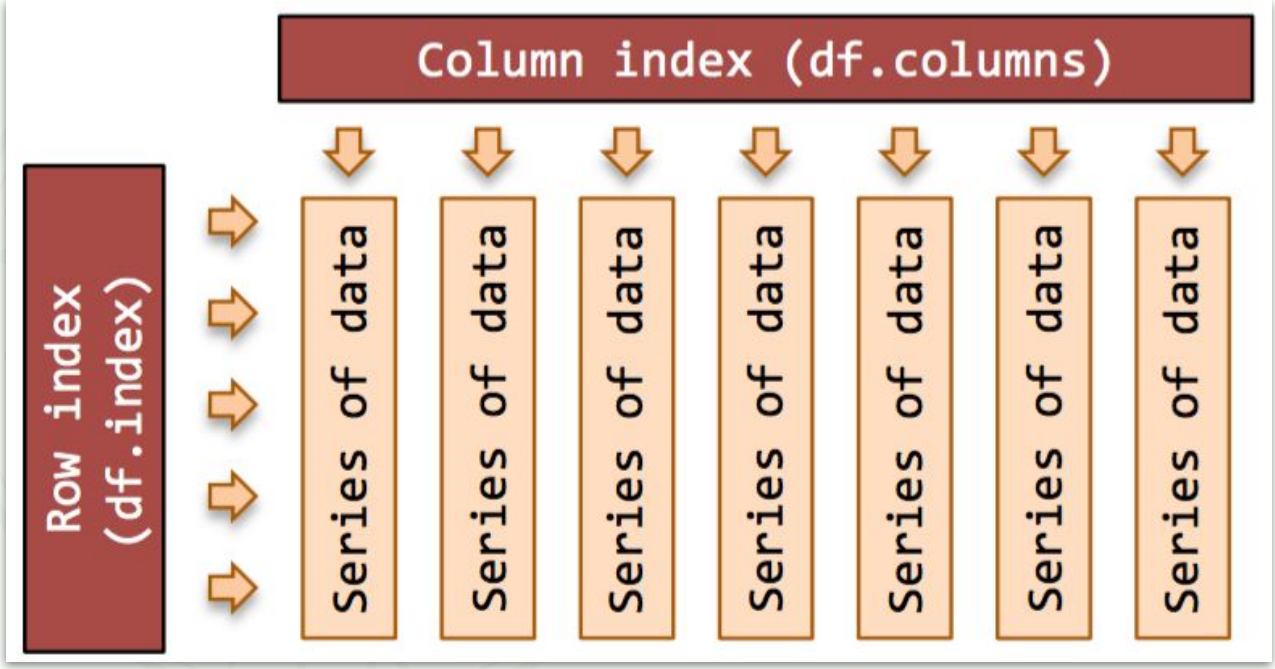
```
my_series.loc["1st"]
```

```
0    apple
1     pear
dtype: object
```

DataFrames

`DataFrames`, which are far more commonly used, can be thought of as a stacked `Series` with a column index.





DataFrame from Series

```
series_1 = pd.Series(range(5))
series_2 = pd.Series(["a", "b", "car", "cat", "cab"])
```

```
series_1
```

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

```
series_2
```

```
0      a
1      b
2    car
3    cat
4    cab
dtype: object
```

```
my_frame = {"title 1": series_1, "title 2": series_2}
```

```
my_first_df = pd.DataFrame(my_frame)
```

```
my_first_df
```

	title 1	title 2
0	0	a
1	1	b
2	2	car
3	3	cat
4	4	cab

`DataFrame` from a dictionary

pandas.DataFrame.from_dict

classmethod DataFrame.**from_dict**(*data*, *orient='columns'*, *dtype=None*, *columns=None*)

Construct DataFrame from dict of array-like or dicts.

Creates DataFrame object from dictionary by columns or by index allowing dtype specificatio

Parameters: **data** : *dict*

Of the form {field : array-like} or {field : dict}.

orient : *{'columns', 'index', 'tight'}, default 'columns'*

The "orientation" of the data. If the keys of the passed dict should be the the resulting DataFrame, pass 'columns' (default). Otherwise if the keys rows, pass 'index'. If 'tight', assume a dict with keys ['index', 'columns', 'd 'index_names', 'column_names'].

New in version 1.4.0: 'tight' as an allowed value for the `orient` arg

dtype : *dtype, default None*

Data type to force, otherwise infer.

columns : *list, default None*

Column labels to use when `orient='index'`. Raises a ValueError if used with `orient='columns'` or `orient='tight'`.

Returns: **DataFrame**

```
>>> import pandas as pd
>>> data = {'col_1': [3, 2, 1, 0], 'col_2': ['a', 'b', 'c', 'd']}
>>> df = pd.DataFrame.from_dict(data)
>>> df
   col_1 col_2
0      3     a
1      2     b
2      1     c
3      0     d
>>> df['col_1']
0    3
1    2
2    1
3    0
Name: col_1, dtype: int64
```

Note that a `Series` was returned.

`DataFrame` from a file

pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None,
header='infer', names=NoDefault.no_default, index_col=None, usecols=None,
squeeze=None, prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None,
engine=None, converters=None, true_values=None, false_values=None,
skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True,
parse_dates=None, infer_datetime_format=False, keep_date_col=False,
date_parser=None, dayfirst=False, cache_dates=True, iterator=False,
chunksize=None, compression='infer', thousands=None, decimal='.',
lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None,
comment=None, encoding=None, encoding_errors='strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None,
delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None,
storage_options=None)                                                    [sourc
```

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

Parameters: **filepath_or_buffer** : *str, path object or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.csv.
If you want to pass in a path object, pandas accepts any `os.PathLike`.
By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sep : *str, default ','*

Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and

pandas.read_json

```
pandas.read_json(path_or_buf=None, orient=None, typ='frame', dtype=None,
convert_axes=None, convert_dates=True, keep_default_dates=True, numpy=False,
precise_float=False, date_unit=None, encoding=None, encoding_errors='strict',
lines=False, chunksize=None, compression='infer', nrows=None,
storage_options=None)                                                    [source]
```

Convert a JSON string to pandas object.

Parameters: **path_or_buf** : *a valid JSON str, path object or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.json.
If you want to pass in a path object, pandas accepts any `os.PathLike`.
By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

orient : *str*

Indication of expected JSON string format. Compatible JSON strings can be produced by `to_json()` with a corresponding orient value. The set of possible orients is:

- `'split'` : dict like `{index -> [index], columns -> [columns], data -> [values]}`
- `'records'` : list like `[{column -> value}, ... , {column -> value}]`
- `'index'` : dict like `{index -> {column -> value}}`
- `'columns'` : dict like `{column -> {index -> value}}`
- `'values'` : just the values array

pandas.read_excel

```
pandas.read_excel(io, sheet_name=0, header=0, names=None, index_col=None,
usecols=None, squeeze=None, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, parse_dates=False,
date_parser=None, thousands=None, decimal='.', comment=None, skipfooter=0,
convert_float=None, mangle_dupe_cols=True, storage_options=None)          [source]
```

Read an Excel file into a pandas DataFrame.

Supports *xls*, *xlsx*, *xlsm*, *xlsb*, *odf*, *ods* and *odt* file extensions read from a local filesystem or URL. Supports an option to read a single sheet or a list of sheets.

Parameters: **io** : *str, bytes, ExcelFile, xlrd.Book, path object, or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.xlsx.
If you want to pass in a path object, pandas accepts any `os.PathLike`.
By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sheet_name : *str, int, list, or None, default 0*

Strings are used for sheet names. Integers are used in zero-indexed sheet positions (chart sheets do not count as a sheet position). Lists of strings/integers are used to request multiple sheets. Specify None to get all worksheets.
Available cases:

Mixing Pandas and NumPy

pandas.Series.to_numpy

`Series.to_numpy(dtype=None, copy=False, na_value=NoDefault.no_default, **kwargs)` [source]

A NumPy ndarray representing the values in this Series or Index.

Parameters:

dtype : *str or numpy.dtype, optional*

The dtype to pass to `numpy.asarray()`.

copy : *bool, default False*

Whether to ensure that the returned value is not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

na_value : *Any, optional*

The value to use for missing values. The default value depends on *dtype* and the type of the array.

> *New in version 1.0.0.*

****kwargs**

Additional keywords passed through to the `to_numpy` method of the underlying array (for extension arrays).

> *New in version 1.0.0.*

pandas.DataFrame.to_numpy

`DataFrame.to_numpy(dtype=None, copy=False, na_value=NoDefault.no_default)` [source]

Convert the DataFrame to a NumPy array.

By default, the dtype of the returned array will be the common NumPy dtype of all types in the DataFrame. For example, if the dtypes are `float16` and `float32`, the results dtype will be `float32`. This may require copying data and coercing values, which may be expensive.

Parameters:

dtype : *str or numpy.dtype, optional*

The dtype to pass to `numpy.asarray()`.

copy : *bool, default False*

Whether to ensure that the returned value is not a view on another array. Note that `copy=False` does not *ensure* that `to_numpy()` is no-copy. Rather, `copy=True` ensure that a copy is made, even if not strictly necessary.

na_value : *Any, optional*

The value to use for missing values. The default value depends on *dtype* and the dtypes of the DataFrame columns.

> *New in version 1.1.0.*

Returns: numpy.ndarray

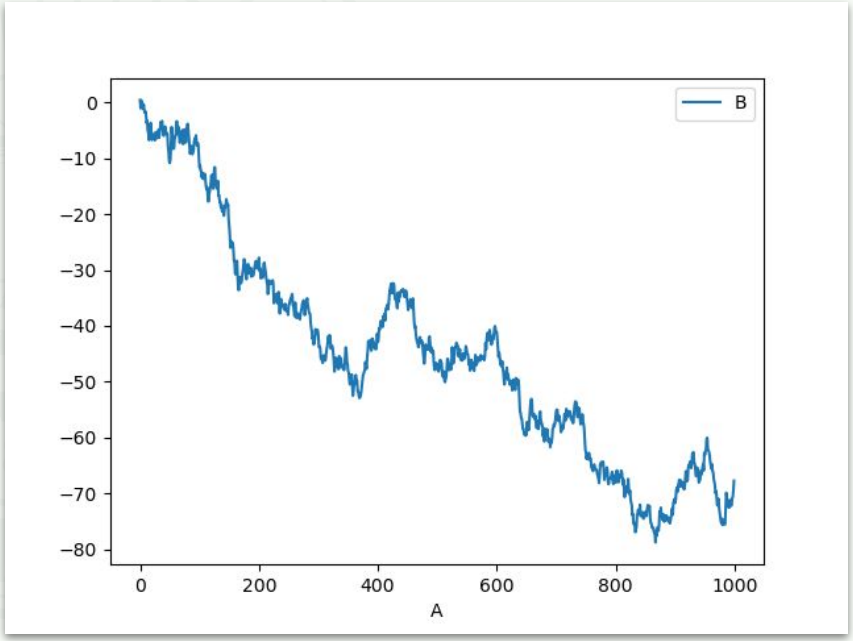
Visualization in Pandas

```
In [11]: df3 = pd.DataFrame(np.random.randn(1000, 2), columns=["B", "C"]).cumsum()

In [12]: df3["A"] = pd.Series(list(range(len(df))))

In [13]: df3.plot(x="A", y="B");
```

There are some advantages to plotting directly in Pandas, despite the fact that it uses Matplotlib.

Pandas exploits the fact that your data is in a dataframe.

You might be better off using Seaborn?



Wednesday