Workshop #1: Introduction to Python for Data Science

TRiCAM 2017

W. Pan

Schedule of Workshops

Workshop series covering computational, mathematical/statistical skills:

- **June 6th**, Workshop 0: setting up computational and collaborative environments
- **June 7th**, Workshop 1: introduction to python for data science
- **June 8th**, Workshop 2: introduction to statistical modeling with python
- **TBA**, Workshop 3: getting serious with GIT
- **TBA**, Workshop 4: advanced data manipulation and visualization in python
- **TBA**, Workshop 5: advanced statistical models
- **TBA**, Workshop 6: air quality exploration app (language/platform TBA)

Schedule of Workshops

Workshop structure:

- **Morning, 9am - 12pm**: interactive lecture
- **Afternoon, 2pm - 5pm**: applying techniques/concepts form the morning to project datasets

Lecture Outline

Introduction to Data Science

What is Jupyter/IPtyhon

Quick Review of Python Basics

Data Gathering & Cleaning

Data Exploration: Descriptive Statistics

Data Exploration: Data Visualization

Introduction to Data Science

What is Data Science

Our working definition: Data science is any set of principled methodology by which we gather data, process it, extract value from it and to communicate our understanding from it.

Glib but insightful alternatives:

> *Data scientist is someone who knows more statistics than a computer scientist and more computer science than a statistician. - Josh Blumenstock*

> *Data scientist = statistician + programmer + coach + storyteller + artist. - Shlomo Aragm*

The Data Science Process

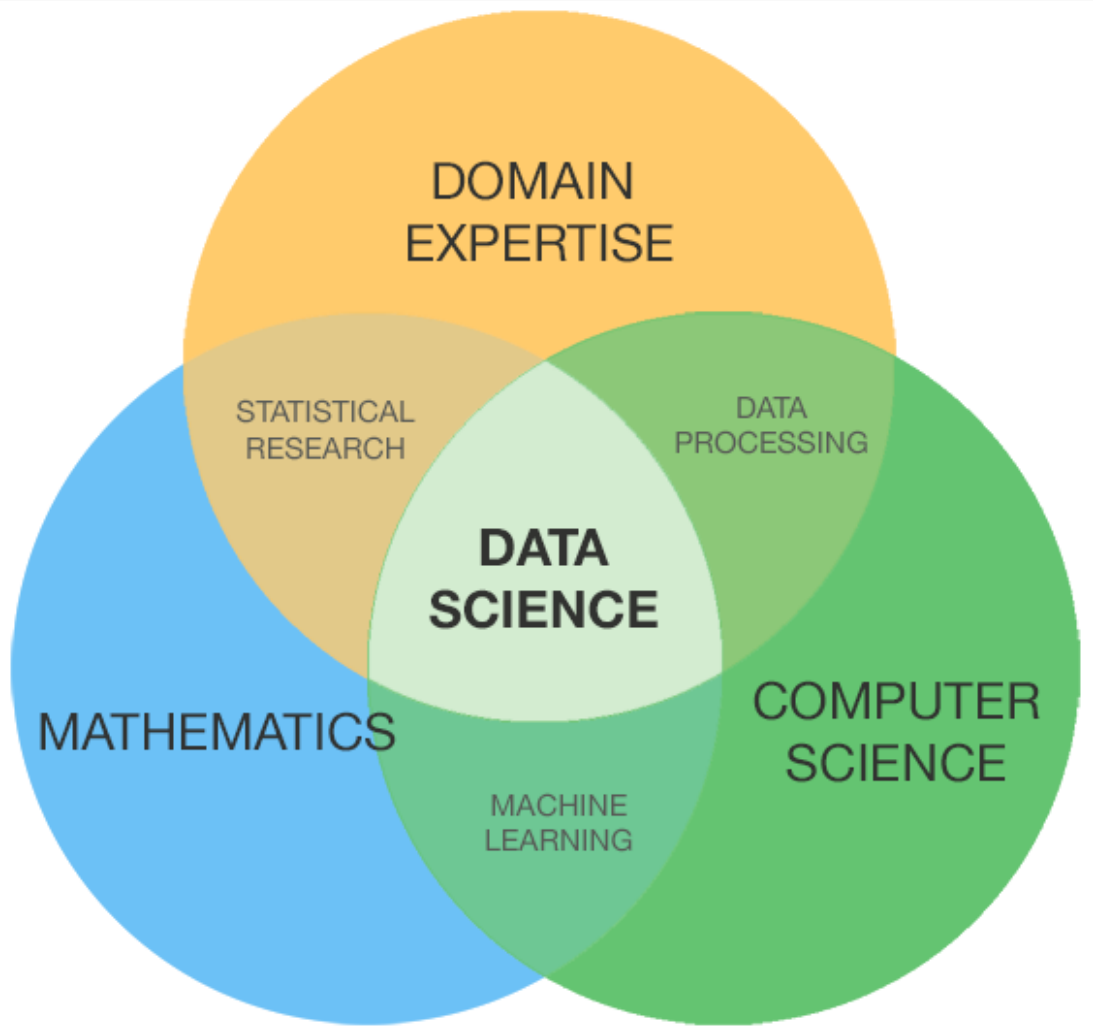
The Data Science Process is similar to the scientific process - one of observation, model building, analysis and conclusion:

- Ask questions
- Data Collection
- Data Exploration
- Data Modeling
- Data Analysis
- Visualization and Presentation of Results

Note: This process is by no means linear!

What Are the Tools of Data Science

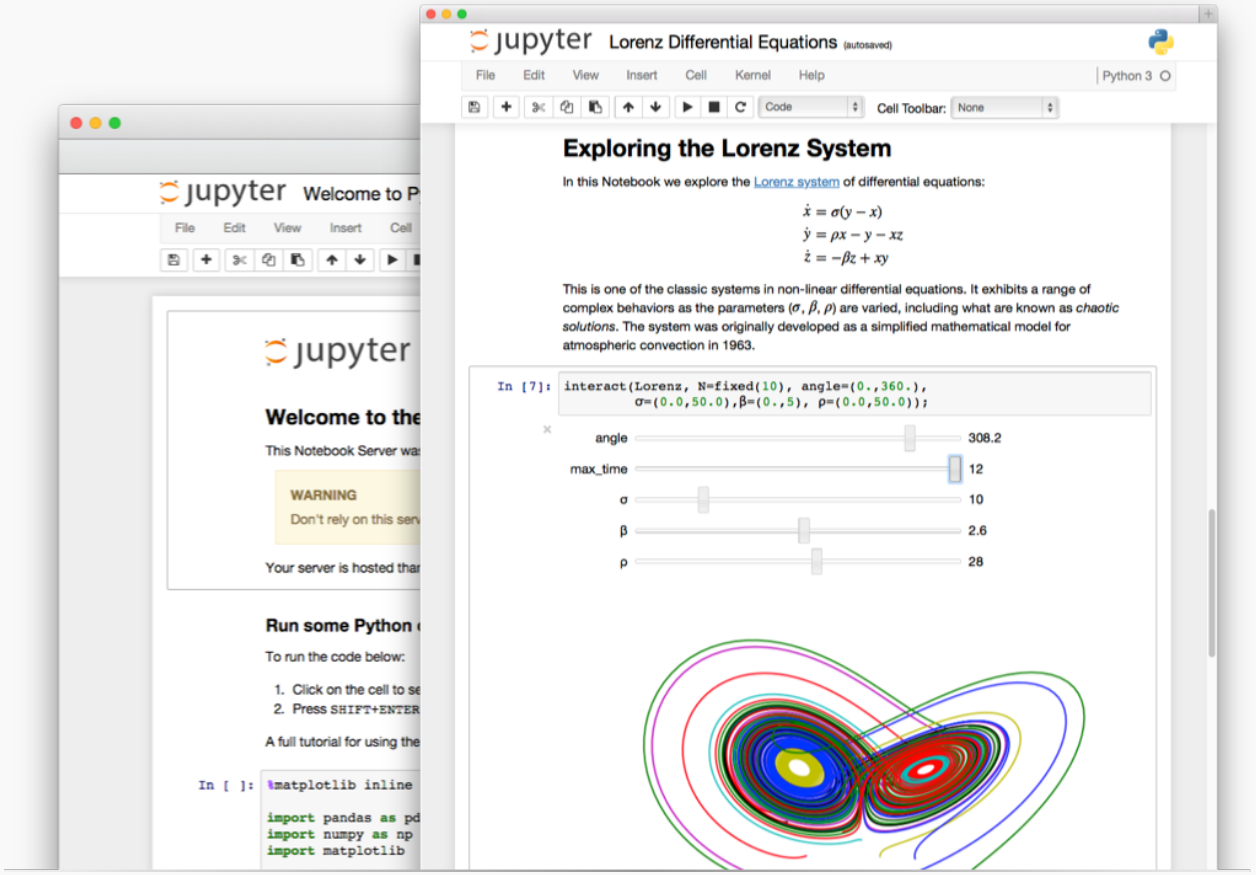
Data science is an interdisciplinary practice require a diverse set of skills, or a diverse team of experts



What is Jupyter/IPtyhon

The Jupyter Notebook App

‘The Jupyter Notebook is a web application that allows you to create interactive documents that contain live code, equations, visualizations and explanatory text.’

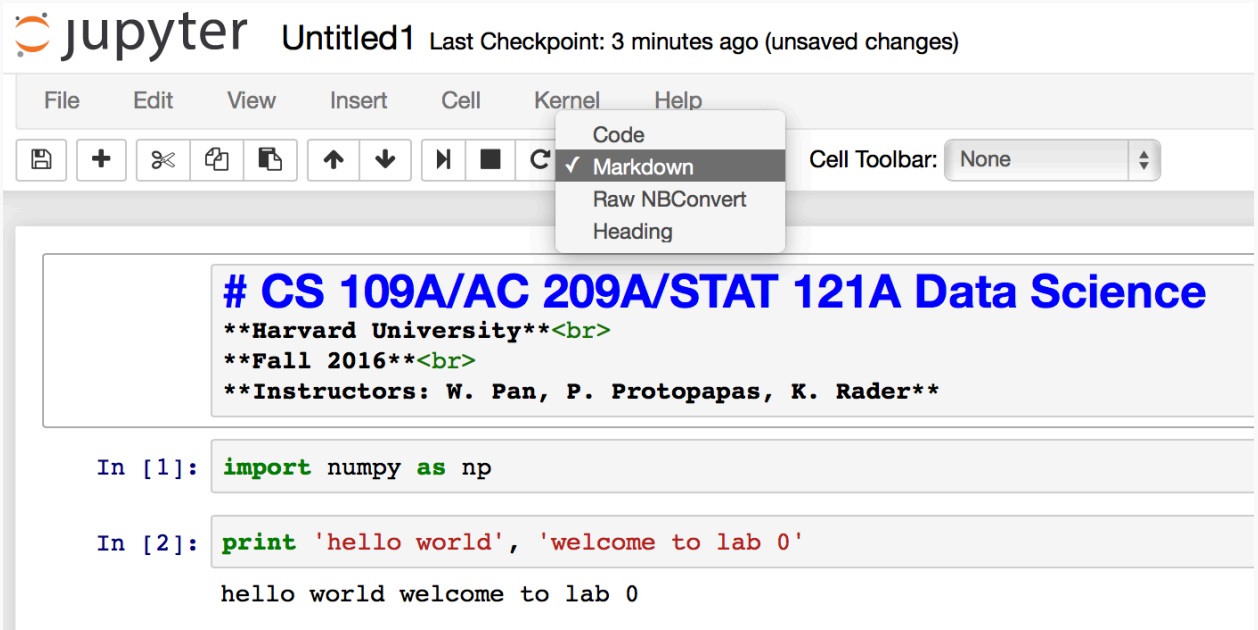


The Jupyter Notebook App

When Jupyter app loads, you see a dashboard displaying files in the Jupyter home directory (you can reset this)

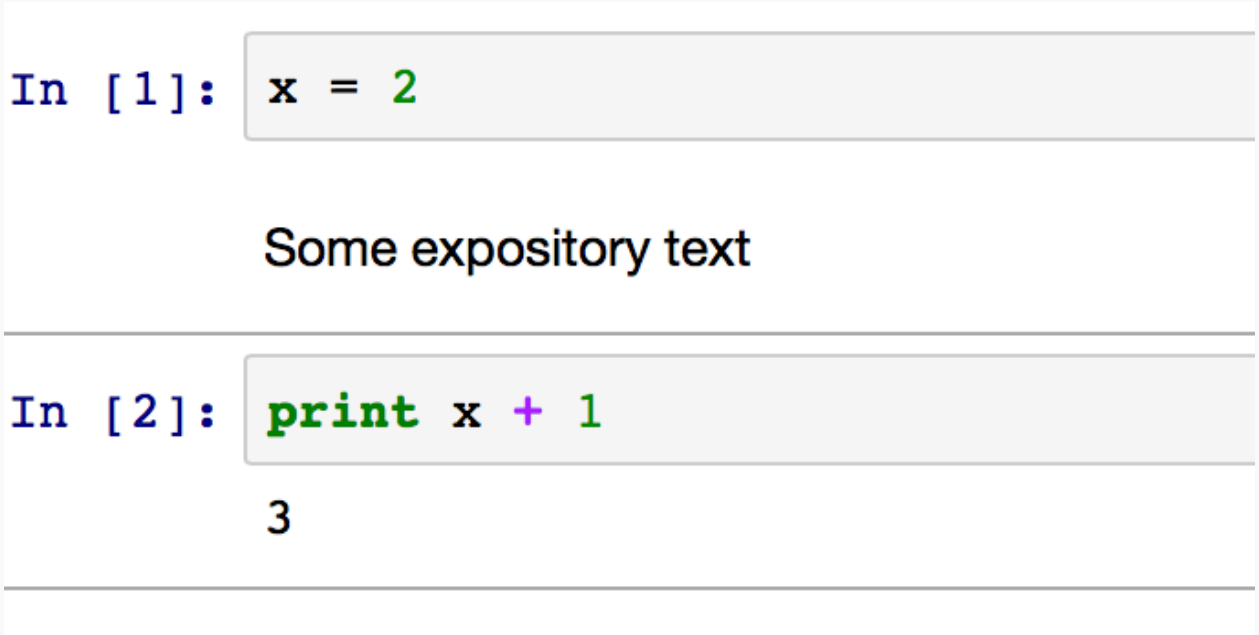
The Jupyter Notebook App

Each notebook consists of blocks of cells. Each cell can display rich text elements (Markdown) or code. Code is executed by an 'computational engine' called the ***kernel*** (IPython). The output of the code is displayed directly below.



The Jupyter Notebook App

Each cell can be executed independently, but once a block of code is executed, it lives in the memory of the kernel.



Quick Review of Python Basics

Python Best Practices

Code readability is key, Python syntax itself is close to plain english.

- Your variables should be given **descriptive identifiers**!

  Identifiers for variable should be descriptive words separated by underscore (not spaces) and in all lower case

BAD	GOOD
`var6 = 25`	`age_of_mother = 25`
`AG3ofMoTh3R = 25`	`age_of_mother = 25`

Python Best Practices

Code readability is key, Python syntax itself is close to plain english.

- Your variables should be given **descriptive identifiers**!
- You should use **white space** to increase readability.

BAD	GOOD
`x=[2,3,4]`	`num_list = [2, 3, 4]`
`v=1/3*(pi*r**2*h)`	`v = 1/3 * (pi * r**2 * h)`

Python Best Practices

Code readability is key, Python syntax itself is close to plain english.

- Your variables should be given **descriptive identifiers**!
- You should use **white space** to increase readability.
- You should liberally intersperse your code with **comments**!

BAD

```
v = 1/3 * (pi * r**2 * h)
```

GOOD

```
#volume of cone
v = 1/3 * (pi * r**2 * h)
```

A line of text following by # is treated as a comment.

Python Best Practices

Code readability is key, Python syntax itself is close to plain english.

- Your variables should be given **descriptive identifiers**!
- You should use **white space** to increase readability.
- You should liberally intersperse your code with **comments**!
- **Proper indentation is non-negotiable!**

BAD

```
for i in range(5):
print i
```

GOOD

```
for i in range(5):
    print i
```

Code blocks are not indicated by delimiters (e.g. `{`, `}`) only by indentation!

Built-in Python Data Types

The basic built-in Python data types we'll be using today are:

1. **integers, floating points**: `7`, `7.0`
2. **booleans**: `True`, `False` with logical operations, `and`, `or`, `not`
3. **strings**: `'hi'`, `"7.0"`
4. **lists**: sequence of data (of various types)

Variables and Types

In Python, you do not need to ***declare*** the types of your variables. The type is inferred based on the valued assigned to the variable.

For example: The assignment

```
my_var = 7
```

types `my_var` as an integer. Later, the assignment

```
my_var = 'hello'
```

will cause `my_var` to be typed as a string.

Functions

Function definition follows the `def` keyword.

The first line, the heading, contains the function name and the list of parameters names (you need not specify the type of each parameter).

If your function returns a value, you can do so with the `return` keyword.

```
def add(x, y):
  return x + y
```

You call a function by its name with values for each parameter:

```
answer = add(1, 2)
```

Functions

We can call a function belonging to an instance of a class:

```
returned_val = object.method(param_1, param_2, ...)
```

or call a function imported from a library or package:

```
returned_val = library.function(param_1, param_2, ...)
```

Numerical Operators

Python has a variety of built-in **arithmetic operators** that allows you to combine numbers.

Operator	Description	Example
`+`	adds values on either side	`1.2 + 2 = 3.2`
`-`	subtracts the right value from the left	`1.2 - 0.2 = 1.0`
`*`	multiplies values on either side	`1.2 * 2 = 2.4`
`/`	divides the left value by the right	`4 / 2 = 2.0`
`%`	divides the left value by the right and returns the remainder	`4 % 3 = 1`
`**`	exponentiate the left value by the right	`3**2 = 9`
`//`	divides the left value by the right and removes the decimal part	`3//2 = 1`

Numerical Operators

Python also has a variety of built-in **comparison operators** for numbers.

Operator	Description	Example
`==`	checks if values on either side are equal	`1 == 2` is `False`
`!=`	checks if values on either side are unequal	`1 != 2` is `True`
`>`	checks if left value is greater	`1 > 2` is `False`
`<`	checks if left value is smaller	`1 < 2` is `True`
`>=`	checks if left value is greater or equal	`2 >= 2` is `True`
`<=`	checks if left value is smaller or equal	`1 <= 2` is `True`

String Data Type

String literals in Python are a set of characters enclosed by either single or double quotation marks.

For example: The following are two equivalent assignments

```
my_str = "Hello World!"
my_str = 'Hello World!'
```

String Operators

Python has a variety of built-in operator for string manipulation.

Let's set `s = 'Hi!'`.

Operator	Description	Example
`==`	checks if strings on either side are equal	`s == 'hi!'` is `False`
`!=`	checks if strings on either side are unequal	`s != 'hi!'` is `True`
`+`	appends right string to end of left	`'Hi' + '!'` is `'Hi!'`
`[n]`	returns the n-th character	`s[0]` is `'H'`
`[n:m]`	returns the substring from n up to m	`s[0:1]` is `'H'`
`[n:]`	returns the substring from n on	`s[1:]` is `'i!'`
`[:n]`	returns the substring up to n	`s[:2]` is `'Hi'`

Note: Python enumerates starting from **zero**!

List Data Type

Lists in Python are collections of items of possibly **different types**. Lists are created and displayed with items separated by commas and enclosed by square brackets. The empty list is denoted by `[]`.

For example: The following list contains both numerical and string data types.

```
['hi', 70, 2.1, ':(', '<3']
```

List Operators

Python has a variety of built-in operator for list manipulation (they look just like the string operators).

Let's set `lst = ['hi', 7, 'c']`.

Operator	Description	Example
`+`	appends right list to end of left	`['H'] + [2]` is `['H', 2]`
`[n]`	returns the n-th item	`lst[0]` is `'hi'`
`[n:m]`	returns items from n up to m	`lst[0:1]` is `['hi']`
`[n:]`	returns items from n on	`lst[1:]` is `[7, 'c']`
`[:n]`	returns items up to n	`lst[:2]` is `['hi', 7]`

Selection: ‘if’, ‘elif’, ‘else’

In Python, selection is implemented using the `if`, `elif`, `else` constructions.

For example: A holistic 0-5 homework grading scheme might translate into:

```
if grade == 5:
    print ``Everything was outstanding!"
elif grade == 4:
    print ``Everything was good with no major mistakes"
elif grade == 3 or grade == 2:
    print ``Good with some major mistakes"
elif grade == 1:
    print ``Hmm...there seem to be lots of missing solutions"
elif grade == 0:
    print ``Oops! You forgot to submit this one!"
else:
    print ``That's not a valid grade!"
```

Iterating over Lists

We can directly **iterate over the items in a list** (from 0-th index to end).

```
In [9]: my_list = [1, 2, 3, 4]

        for val in my_list:
            print val
```

```
1
2
3
4
```

Iterating over Ranges of Numbers

We can **iterate over a range of numbers**.

```
In [10]: my_list = [1, 2, 3, 4]

         for index in range(4):
             print my_list[index]

1
2
3
4
```

Iterating over Ranges of Numbers

Variations on `range()`:

- `range(10)` produces a list-like of numbers 0 thru 9:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- `range(5, 10)` will produce a list-like of numbers starting at 5 and thru 9

  5, 6, 7, 8, 9

- `range(0, 10, 2)` will produce a list-like of numbers between 0 and 9 counting by 2:

  0, 2, 4, 6, 8

Try It Yourself!

- *Write and test a function that takes a list of numbers and returns the largest number (without using pre-defined functions). Find a pre-defined function that performs the same task (use Google!).
- *Write a function that takes a list and does each of the following:
  - prints every other item in the list
  - prints each element of the list in reverse order
  - prints the last 5 elements in the list
- **Write a function that takes a string and returns `True` if the string is a palindrome, otherwise it returns `False`.
- ***Write a function that zips two lists together. That is, given

  `[1, 2, 3]` and `[a, b, c]`

  we produce

  `[[1, a], [2, b], [3, c]]`

  Do this using **list comprehension**.

Data Gathering & Cleaning

The Data Science Process - Again

Recall the Data Science Process we outlined earlier:

- Ask questions
- Data Collection
- Data Exploration
- Data Modeling
- Data Analysis
- Visualization and Presentation of Results

Today we'll be addressing data collection and exploration. Tomorrow we'll addressing building models for data and analyzing the results.

What Is Data?

‘A **datum** is a single measurement of something on a scale that is understandable to both the recorder and the reader. **Data** is multiple such measurements.’

Provocative claim: everything is (can be) data!

Where Does It Come From?

- **Internal sources:** already collected by or is part of the overall data collection of you organization.

  For example: business-centric data that is available in the organization data base to record day to day operations; scientific or experimental data

- **Existing External Sources:** available in ready to read format from an outside source for free or for a fee.

  For example: public government databases, stock market data, Yelp reviews

- **External Sources Requiring Collection Efforts:** available from external source but acquisition requires special processing.

  For example: data appearing only in print form, or data on websites

Where Does It Come From?

How to get data generated, published or hosted online:

- **API (Application Programming Interface)**: using a prebuilt set of functions developed by a company to access their services. Often pay to use.

  For example: Google Map, Facebook, Twitter API

- **RSS (Rich Site Summary)**: summarizes frequently updated online content in standard format. Free to read if the site has one.

  For example: news-related sites, blogs

- **Web scraping:** using software, scripts or by-hand extracting data from what is displayed on a page or what is contained in the HTML file.

Web Scraping

- **Why do it?** Older government or smaller news sites might not have APIs for accessing data, or publish RSS feeds or have databases for download. You don't want to pay to use the API or the database.
- **How do you it?** See Tutorial on `beautifulsoup`
- **Should you do it?**
  - **You just want to explore:** Are you violating their terms of service? Privacy concerns for website and their clients?
  - **You want to publish your analysis or product:** Do they have an API or fee that you're bypassing? Are they willing to share this data? Are you violating their terms of service? Are there privacy concerns?

What Does It Look Like?

How is your data represented and stored (data format)?

- **Tabular Data:** a dataset that is a two-dimensional table, where each row typically represents a single data record, and each column represents one type of measurement (csv, tsp, xlsx etc.).
- **Structured Data:** each data record is presented in a form of a, possibly complex and multi-tiered, dictionary (json, xml etc.)
- **Textual Data:** data is unstructured text
- **Temporal Data:** data is indexed by time (time-series)
- **Geolocation Data:** data is longitude, latitude etc.

More on Tabular Data

In tabular data, we expect each record or **observation** to represent a set of measurements of a single object or event.

	Hight	Radius	Do I Like It?
Cylinder # 1	10	5	Yes
Cylinder # 2	3	7.5	No

Each type of measurement is called a **variable**, an **attribute** or a **feature** of the data. The number of attributes is called the **dimension** of the data or of the **feature space**.

We expect each table to contain a set of records or observations of the same kind of object or event (e.g. our table above contains observations of cylinders).

More on Tabular Data

You'll see later that it's important to distinguish between classes of variables or attributes based on the type of values they can take on.

- **Quantitative variable**: is numerical and can be
  - **discrete** - a finite number of values are possible in any bounded interval

    For example: 'Number of siblings' is a discrete variable
  - **continuous** - an infinite number of values are possible in any bounded interval

    For example: 'Height' is a continuous variable

- **Categorical variable**: no inherent order among the values

  For example: 'What kind of pet you have' is a categorical variable

Is the Data Any Good?

Common issues with data:

- **Missing values:** how do we fill in?
- **Wrong values:** how can we detect and correct?
- **Messy format**
- **Not usable:** the data cannot answer the question posed

Handling Messy Data

The following is a table accounting for produce deliveries over a weekend.

What are the variables in this dataset?

What object or event are we measuring?

	Friday	Saturday	Sunday
Morning	15	158	10
Afternoon	2	90	20
Evening	55	12	45

Handling Messy Data

We're measuring individual deliveries; the variables are Time, Day, Number of Produce.

	Friday	Saturday	Sunday
Morning	15	158	10
Afternoon	2	90	20
Evening	55	12	45

Problem: each column header represents a single ***value*** rather than a ***variable***. Row headers are 'hiding' the Time variable. The values of the variable, 'Number of Produce', is not recorded in a single column.

Handling Messy Data

We need to reorganize the information to make explicit the event we're observing and the variables associated to this event.

Delivery	**Time**	**Day**	**No. of Produce**
1	Morning	Friday	15
2	Morning	Saturday	158
3	Morning	Sunday	10
4	Afternoon	Friday	2
5	Afternoon	Saturday	90
6	Afternoon	Sunday	20
7	Evening	Friday	55
8	Evening	Saturday	12
9	Evening	Sunday	45

Handling Messy Data

What object or event are we measuring?

What are the variables in this dataset?

Delivery	Amount
On Sunday	
10:30	43
12:30	12
12:35	30
On Monday	
11:30	29
11:57	87
11.59	63
On Tuesday	
11:33	19
11:15	27
12.59	54

Handling Messy Data

We’re measuring individual deliveries; the variables are Time, Day, Number of Produce:

Days	times	Amount
Sunday	10:30	43
Sunday	12:30	12
Sunday	12:35	30
Monday	11:30	29
Monday	11:57	87
Monday	11.59	63
Tuesday	11:33	19
Tuesday	11:15	27
Tuesday	12.59	54

Handling Messy Data

Common causes of messiness are:

- Column headers are values, not variable names
- Variables are stored in both rows and columns
- Multiple variables are stored in one column
- Multiple types of experimental units stored in same table

In general, **we want each file to correspond to a dataset, each column to represent a single variable and each row to represent a single observation**.

Tabular Data as CSV Files

A comma-separated values (CSV) file stores tabular data in plain text. Each line of the file is a single record. Each record consists of one or more values, numeric or text, separated, typically, by commas.

```
1,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,13
1,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,5
2,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,7
1,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,8
0,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,5
0,0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,6
2,0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,18
2,0.545,0.425,0.125,0.768,0.294,0.1495,0.26,14
1,0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,7
2,0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,17
2,0.525,0.38,0.14,0.6065,0.194,0.1475,0.21,12
1,0.43,0.35,0.11,0.406,0.1675,0.081,0.135,8
1,0.49,0.38,0.135,0.5415,0.2175,0.095,0.19,9
2,0.535,0.405,0.145,0.6845,0.2725,0.171,0.205,8
2,0.47,0.355,0.1,0.4755,0.1675,0.0805,0.185,8
1,0.5,0.4,0.13,0.6645,0.258,0.133,0.24,10
```

numpy Arrays

numpy is a python package for scientific computation. The classes and functions in numpy are made available by **importing**:

```
import numpy as np
```

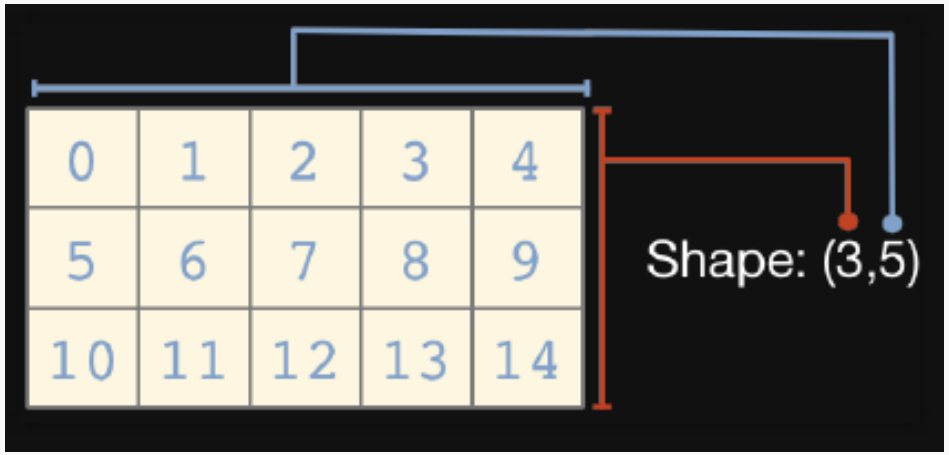
Notably, numpy provides an multi-dimensional array object which optimizes storing and manipulating data.

CSV data can be read into numpy arrays by:

```
my_data = np.loadtxt('mass_aq_data/boston_year_to_date/boston_co.csv',
                     delimiter=',')
```

numpy 1D Arrays

Each array has a shape, recorded as a tuple $(n, m, ...)$.



Creating 1D arrays:

```
In [19]: my_array = np.array([1, 2, 3, 4])
         print my_array.shape

         (4,)
```

Note: indexing with 1D arrays work just like with lists and strings!

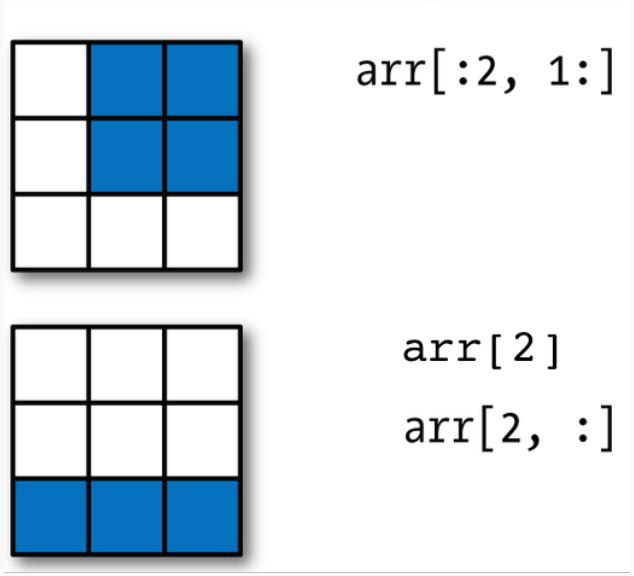
numpy 2D Arrays

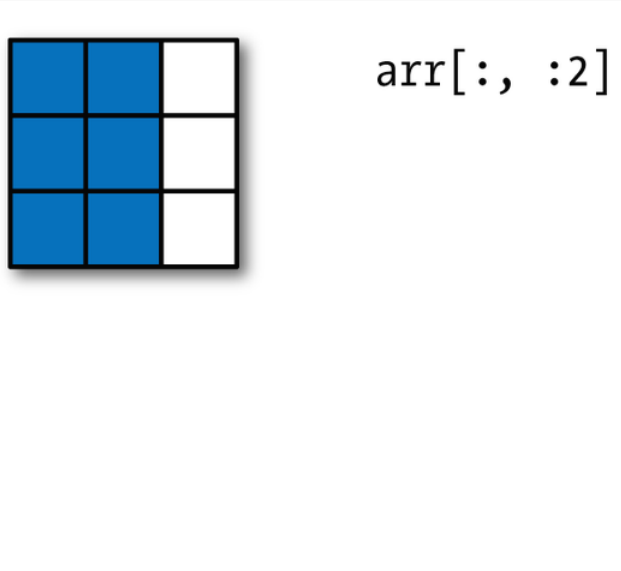
Creating 2D arrays: we can make 2-D arrays out of a list of rows, each row is a list of values.

```
In [24]: my_array = np.array([[1, 2], [3, 4]])
         print my_array

         [[1 2]
          [3 4]]

In [25]: print my_array.shape

         (2, 2)
```

numpy 2D Arrays

Indexing 2D arrays: The element at the n-th row and the m-th column is indexed as `[n, m]`. Just like lists, you can also get multiple array values at a time:





Finally, you can get a 'list' rows: `arr[[0, 1]]` or `arr[[0, 1], :]`

numpy 2D Arrays

Filtering 2D arrays: Even more sophisticated, you can get values from an array that satisfy a bunch of criteria!

```
In [24]: my_array = np.array([[1, 2], [3, 4]])
         print my_array

         [[1 2]
          [3 4]]

In [44]: my_array = np.array([[1, 2], [3, 4]])
         print my_array[:, 0] == 1

         [ True False]

In [34]: print my_array[my_array[:, 0] == 1]

         [[1 2]]

In [37]: print my_array[(my_array[:, 0] == 1) | (my_array[:, 0] == 3)]

         [[1 2]
          [3 4]]
```

Question: how do you get the values that are greater than one? What is the shape of this array of filtered values?

Try It Yourself!

- read the Carbon Monoxide data for Boston into an `numpy` array
- get the shape of this array. How many observations are there? How many features?
- filter the data for observations taking at the site 'Boston Roxbury'. How many observations are taken from this site?

Observations

Working with `numpy` has it's draw-backs!

- It's not easy to read tabular data from csv files where the values are mixed in type (some strings, some floats)
- It's not easy to read in and store the column headers (strings) in the numpy array representing the data
- We can only reference columns by position rather than column header. E.g. I want the 'height' column, but I need to remember that it's the 3rd column in the array.

Observations

Wish List: we want a data structure

- that can easily store variables of different types
- that stores column names
- where we can reference column by position as well as by column name
- that comes with built in functions for manipulating this structure

Answer: Python has a package/library for that, `pandas`.

Observations

Question: Why learn/use `numpy`?

Answer:

- `pandas` is a set of objects and tools built on top of `numpy`.
- computing with `pandas` data structures vs `numpy` arrays can mean performance difference!

Introduction to pandas

pandas objects can be thought of as 'enhanced' versions of numpy arrays in which the rows and columns are identified with labels rather than integers.

- **Series:** 1D array of data with an index object (labels).

```
In [4]: import pandas as pd

        column = pd.Series([0.25, 0.5, 0.75, 1.0],
                           index=['one', 'two', 'three', 'four'])
        column

Out[4]: one      0.25
        two      0.50
        three    0.75
        four     1.00
        dtype: float64

In [6]: column['four']

Out[6]: 1.0
```

Each series has a 'values' component and an 'index' component.

Introduction to pandas

pandas objects can be thought of as 'enhanced' versions of numpy arrays in which the rows and columns are identified with labels rather than integers.

- **Series:** 1D array of data with an index object (labels).

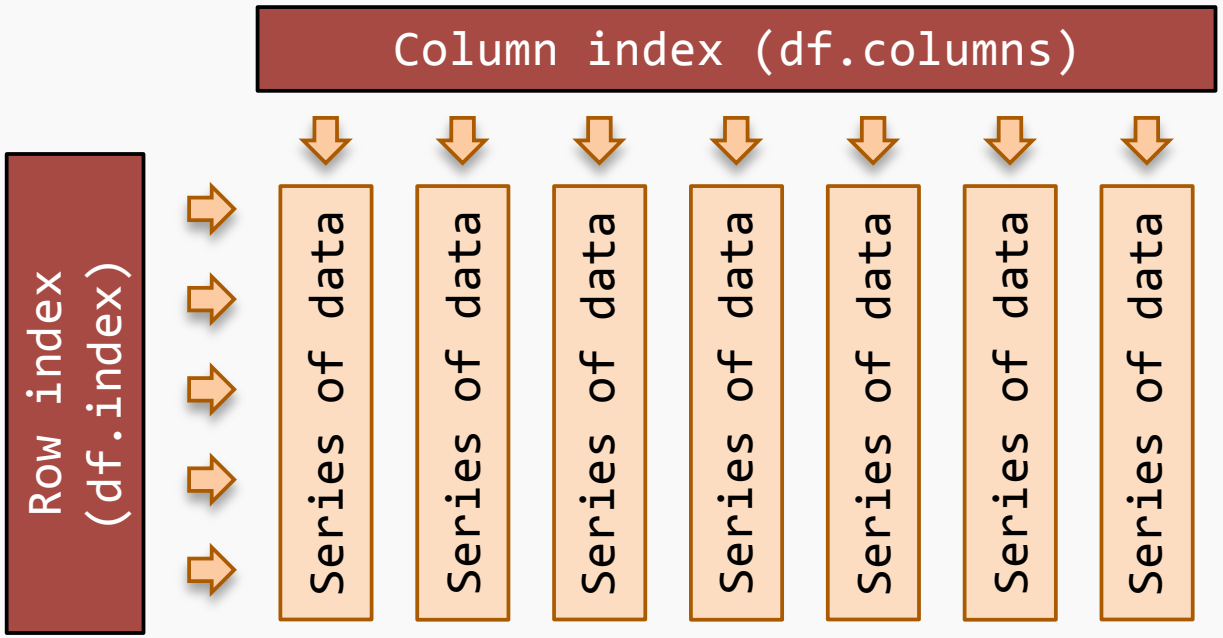
```
In [7]: column.index
Out[7]: Index([u'one', u'two', u'three', u'four'], dtype='object')

In [11]: column.values
Out[11]: array([ 0.25,  0.5 ,  0.75,  1.  ])
```

  Each series has a 'values' component and an 'index' component.

Introduction to pandas

pandas objects can be thought of as 'enhanced' versions of numpy arrays in which the rows and columns are identified with labels rather than integers.

- **DataFrame:** 2D table of data with column and row index objects (labels).



Each column in the data frame is a ***series***.

We can create a data frame from columns (series objects):

In [15]:
```
column_1 = pd.Series(range(4),
                     index=['one', 'two', 'three', 'four'])

column_2 = pd.Series(range(4, 8),
                     index=['one', 'two', 'three', 'four'])

table = pd.DataFrame({'col_1': column_1,
                      'col_2': column_2})

table
```

Out[15]:

	col_1	col_2
one	0	4
two	1	5
three	2	6
four	3	7

Getting Started with pandas

We can import tabular data in a csv file into a data frame:

```
In [16]: df = pd.read_csv('dataset_HW0.txt')
         df
```

Out[16]:

	birth_weight	femur_length	mother_age
0	2.969489	1.979156	16
1	4.038963	3.555681	16
2	5.302643	3.385633	15
3	6.086107	4.495427	17

Exploring Your Dataframe

We start with a rough sense of what's in the data

- **The indices of your data frame:**

```
In [32]: df.columns
Out[32]: Index([u'birth_weight', u'femur_length', u'mother_age'], dtype='object')

In [5]: df.columns.values
Out[5]: array(['birth_weight', 'femur_length', 'mother_age'], dtype=object)

In [6]: df.index
Out[6]: Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
                    ...
                    390, 391, 392, 393, 394, 395, 396, 397, 398, 399],
                   dtype='int64', length=400)

In [7]: df.index.values
Out[7]: array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
                13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
                26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
                39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
```

The `.index` and `.columns` attributes give access to the index objects of rows and columns (resp).

Exploring Your Dataframe

We start with a rough sense of what's in the data

- **The shape of your data frame:**

```
In [23]: df.shape
Out[23]: (400, 3)

In [10]: len(df.index)
Out[10]: 400
```

Exploring Your Dataframe

We start with a rough sense of what's in the data

- **The first entries in your data frame:**

```
In [25]: df.head(n=5)
```

Out[25]:

	birth_weight	femur_length	mother_age
0	2.969489	1.979156	16
1	4.038963	3.555681	16
2	5.302643	3.385633	15
3	6.086107	4.495427	17
4	5.749260	4.017437	16

The `.head()` function returns a (row-wise) truncated version of your data frame!

Accessing Columns & Rows

- **Accessing a column by label:**

```
In [34]: type(df['birth_weight'])

Out[34]: pandas.core.series.Series

In [35]: df['birth_weight']

Out[35]: 0      2.969489
         1      4.038963
         2      5.302643
         3      6.086107
         4      5.749260
         5      6.049903
```

You can access a column by it's column name or position (you can also access a ***list*** of columns)!

Accessing Columns & Rows

- **Accessing the values of column:**

```
In [36]: df['birth_weight'].values

Out[36]: array([ 2.9694893 ,  4.03896294,  5.30264328,  6.08610661,  5.74926036,
                 6.04990317,  5.42681579,  6.23910323,  5.34504952,  4.16297458,
                 5.27487188,  5.57627684,  5.49364519,  6.66031745,  4.79466787,
                 5.98546786,  4.62521954,  5.60683336,  4.52477222,  6.3162985 ,
                 5.5922901 ,  6.23730155,  5.19645533,  4.61051962,  4.38347209,
                 5.00708476,  4.10801732,  5.18226899,  3.91916625,  5.8955964 ,
```

You can access a column by it's column name or position (you can also access a ***list*** of columns)!

Accessing Columns & Rows

- **Accessing columns by position:**

```
In [63]: df[[0, 1]]
```

Out[63]:

	birth_weight	femur_length
0	2.969489	1.979156
1	4.038963	3.555681
2	5.302643	3.385633
3	6.086107	4.495427
4	5.749260	4.017437
5	6.049903	4.378892
6	5.426816	2.851801

You can access a column by it's column name or position (you can also access a ***list*** of columns)!

Accessing Columns & Rows

- **Accessing a row by position:**

```
In [46]: type(df.iloc[0])
Out[46]: pandas.core.series.Series

In [47]: df.iloc[0]
Out[47]: birth_weight     2.969489
         femur_length     1.979156
         mother_age      16.000000
         Name: 0, dtype: float64
```

You can access a column by it's row name or position!

Accessing Columns & Rows

- **Accessing a row by label:**

```
In [52]: table
```

Out[52]:

	col_1	col_2
one	0	4
two	1	5
three	2	6
four	3	7

```
In [54]: type(table.loc['one'])
Out[54]: pandas.core.series.Series

In [53]: table.loc['one']
Out[53]: col_1    0
         col_2    4
         Name: one, dtype: int64
```

You can access a column by it's row name or position!

Filtering

Filtering works very much like with `numpy` arrays!

```
In [57]: df[(df['mother_age'] > 18) & (df['mother_age'] < 35)]
```

Out[57]:

	birth_weight	femur_length	mother_age
100	6.904530	4.164637	34
101	8.096642	4.536759	22
102	8.165373	5.507030	20
104	6.255286	3.769024	19
105	6.515220	5.568954	23
106	6.464462	3.310628	25
107	6.579616	3.670224	20
108	7.171024	5.159946	24

Try It Yourself!

- read the Carbon Monoxide data for Boston into an `numpy` array
- get the shape of this array. How many observations are there? How many features?
- filter the data for observations taking at the site 'Boston Roxbury'. How many observations are taken from this site?
- do the same thing using `pandas` dataframes.

Data Exploration: Descriptive Statistics

Basic Terms

Population versus sample:

- **Population** is the entire set of objects or events under study. Population can be hypothetical ‘all students’ or all students in a class.
- **Sample** is a ‘representative’ subset of the objects or events under study. Needed because it’s impossible or intractable to obtain or compute with population data.

Basic Terms

Biases in sampling:

- **Selection:** some subjects or records are more likely to be selected
- **Volunteer/nonresponse:** subjects or records who are not easily available are not represented

  For example: I usually only hear from students for whom something has gone terribly wrong in the course.

Describing Data

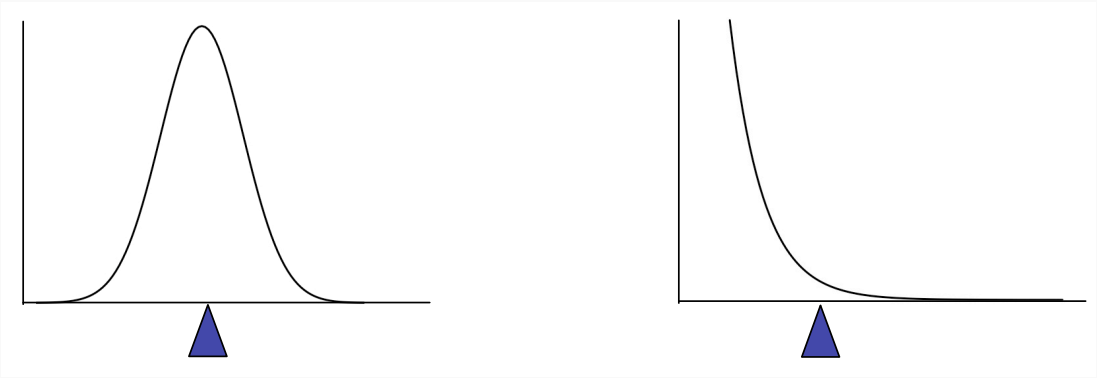
Given some large dataset, we'd like to compute a few quantities that intuitively summarizes the data. To begin with we'd like to know

- what are 'typical' values for our variables or attributes?
- how 'representative' are these typical values?

Centrality

The **mean** of a set of n number of samples of a variable is denoted $\overline{x}$ and is defined by

$$\overline{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i$$



The mean describes what a ‘typical’ sample value looks like, or where is the ‘center’ of the distribution of the data.

Centrality

The **median** of a set of n number of samples, ordered by value, of a variable is is defined by

$$\text{Median} = \begin{cases} x_{\lfloor n/2 \rfloor + 1}, & \text{if } n \text{ is odd} \\ \dfrac{x_{n/2} + x_{n/2+1}}{2}, & \text{if } n \text{ is even} \end{cases}$$

Example:

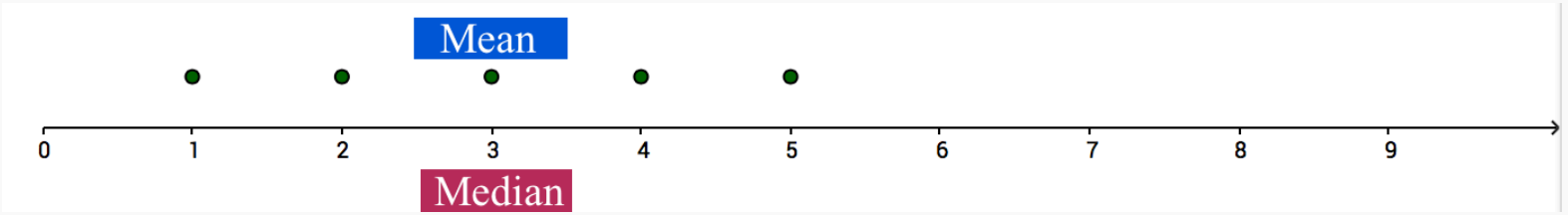
Ages: 17, 19, 21, 22, 23, 23, 23, 38

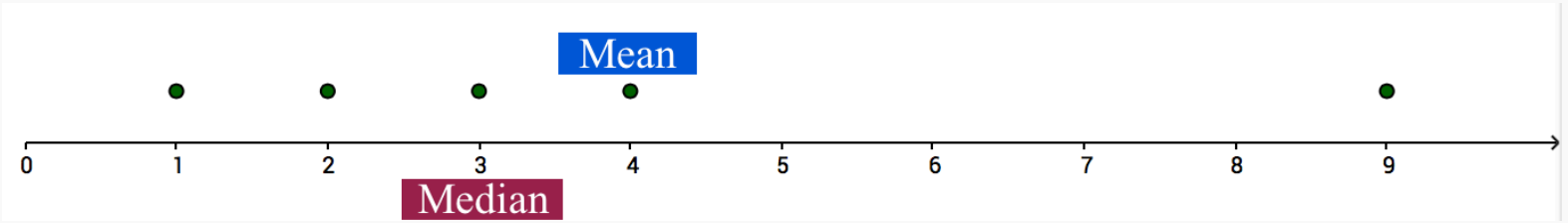
Median $= \frac{22+23}{2} = 22.5$

The median describes what a ‘typical’ sample looks like, or where is the ‘center’ of the distribution of the samples.

Centrality

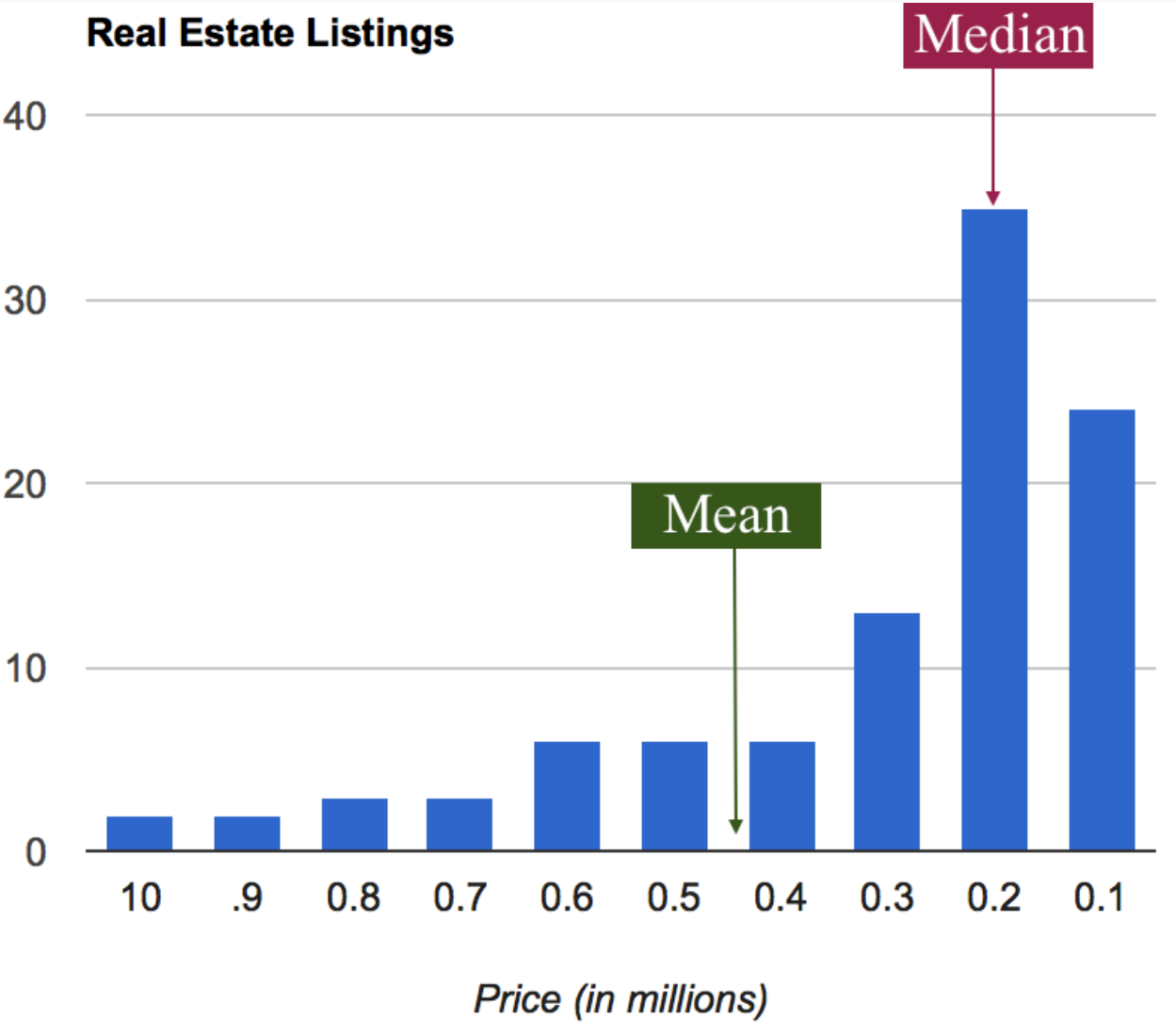
The mean is ***sensitive to outliers***.





Centrality

The mean is ***sensitive to skewness (asymmetry) of distributions***.



Centrality

How hard (in terms of algorithmic complexity) is it to calculate

- **the mean**
- **the median**

Centrality

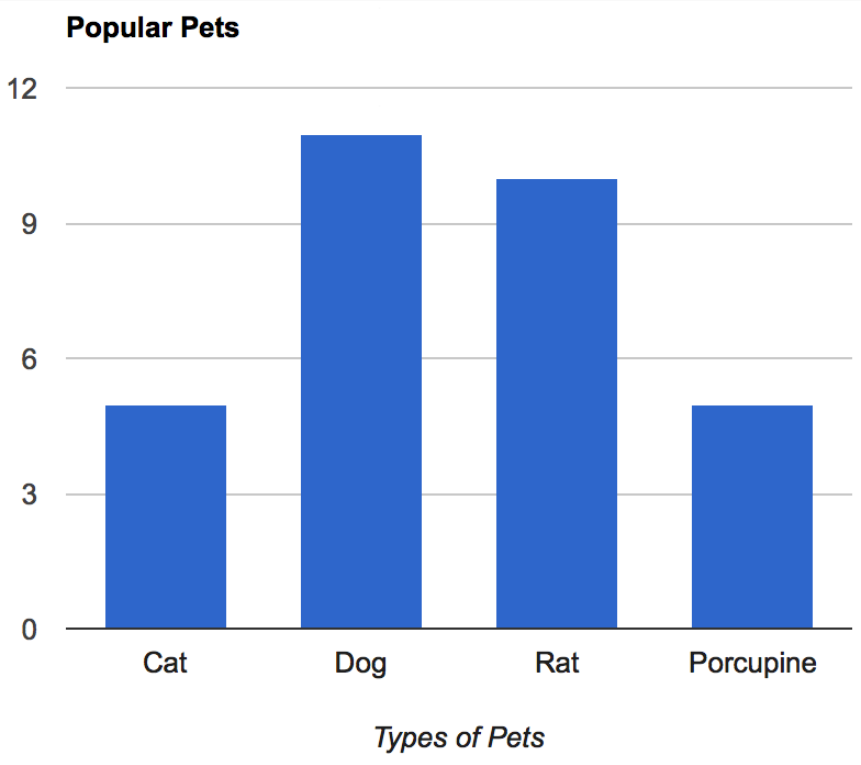
How hard (in terms of algorithmic complexity) is it to calculate

- **the mean:** at most $O(n)$
- **the median:** at leat $O(n \log n)$

Note: Practicality of implementation has to be considered!

Centrality

For samples of categorical variables, neither mean or median make sense.



The **mode** might be a better way to find the most ‘representative’ value.

Spread

The spread of samples measures how well the mean or median describes the sample set.

One way to measuring spread of a set of samples is via the **range**.

$$\text{Range} = \text{Maximum Value} - \text{Minimum Value}$$

Spread

The (sample) **variance**, denoted s^2, measures how much on average the sample values 'deviates' from the mean

$$s^2 = \frac{\sum_{i=1}^{n} |x_i - \overline{x}|^2}{n-1}$$

Note: the term $|x_i - \overline{x}|$ measure the amount by which x_i deviates from the mean $\overline{x}$. Squaring these deviation means that s^2 is sensitive to extreme values (outliers).

Note: s^2 doesn't have the same units as x_i! What does a variance of $1,008$ mean? Or 0.0001?

Spread

The (sample) **standard deviation**, denoted s, is the square root of the variance

$$s = \sqrt{\frac{\sum_{i=1}^{n} |x_i - \overline{x}|^2}{n-1}}$$

Note: s has the same units as x_i!

Descriptive Stats with numpy and pandas

You can compute descriptive stats by either calling the appropriate function belonging to the numpy array object or by applying a numpy's stats function to the array.

```
In [21]: print my_array.mean()
         print np.mean(my_array)

         2.5
         2.5
```

pandas objects also have method for computing stats.

Data Exploration: Data Visualization

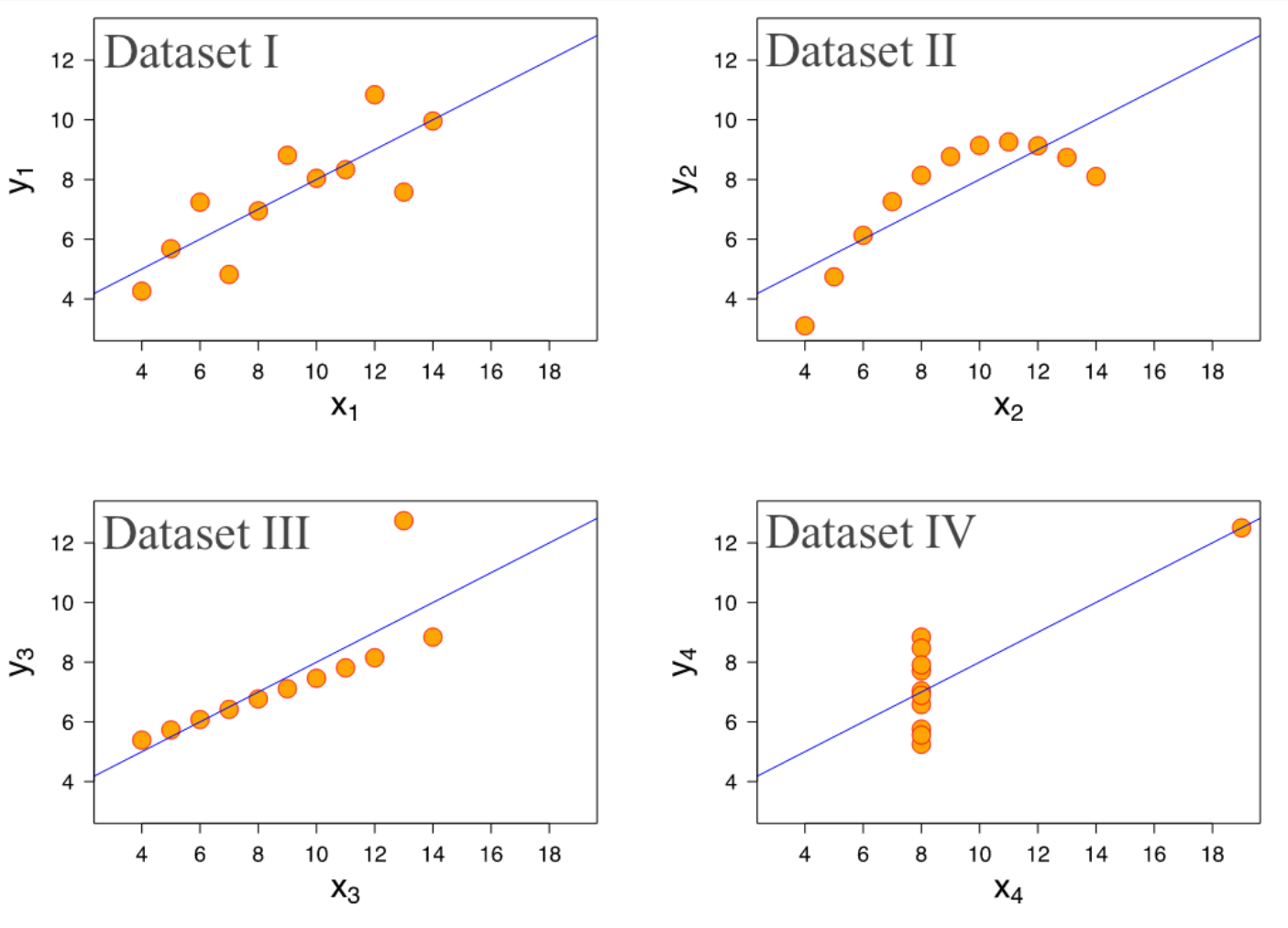
Why Data Visualization?

The following data sets comprise the Anscombe's Quartet; all four sets of data have identical simple summary statistics.

	Dataset I		Dataset II		Dataset III		Dataset IV	
	x	y	x	y	x	y	x	y
	10	8.04	10	9.14	10	7.46	8	6.58
	8	6.95	8	8.14	8	6.77	8	5.76
	13	7.58	13	8.74	13	12.74	8	7.71
	9	8.81	9	8.77	9	7.11	8	8.84
	11	8.33	11	9.26	11	7.81	8	8.47
	14	9.96	14	8.1	14	8.84	8	7.04
	6	7.24	6	6.13	6	6.08	8	5.25
	4	4.26	4	3.1	4	5.39	19	12.5
	12	10.84	12	9.13	12	8.15	8	5.56
	7	4.82	7	7.26	7	6.42	8	7.91
	5	5.68	5	4.74	5	5.73	8	6.89
Sum:	99.00	82.51	99.00	82.51	99.00	82.51	99.00	82.51
Avg:	9.00	7.50	9.00	7.50	9.00	7.50	9.00	7.50
Std:	3.32	2.03	3.32	2.03	3.32	2.03	3.32	2.03

Why Data Visualization?

The following data sets comprise the Anscombe's Quartet; all four sets of data have identical simple summary statistics.



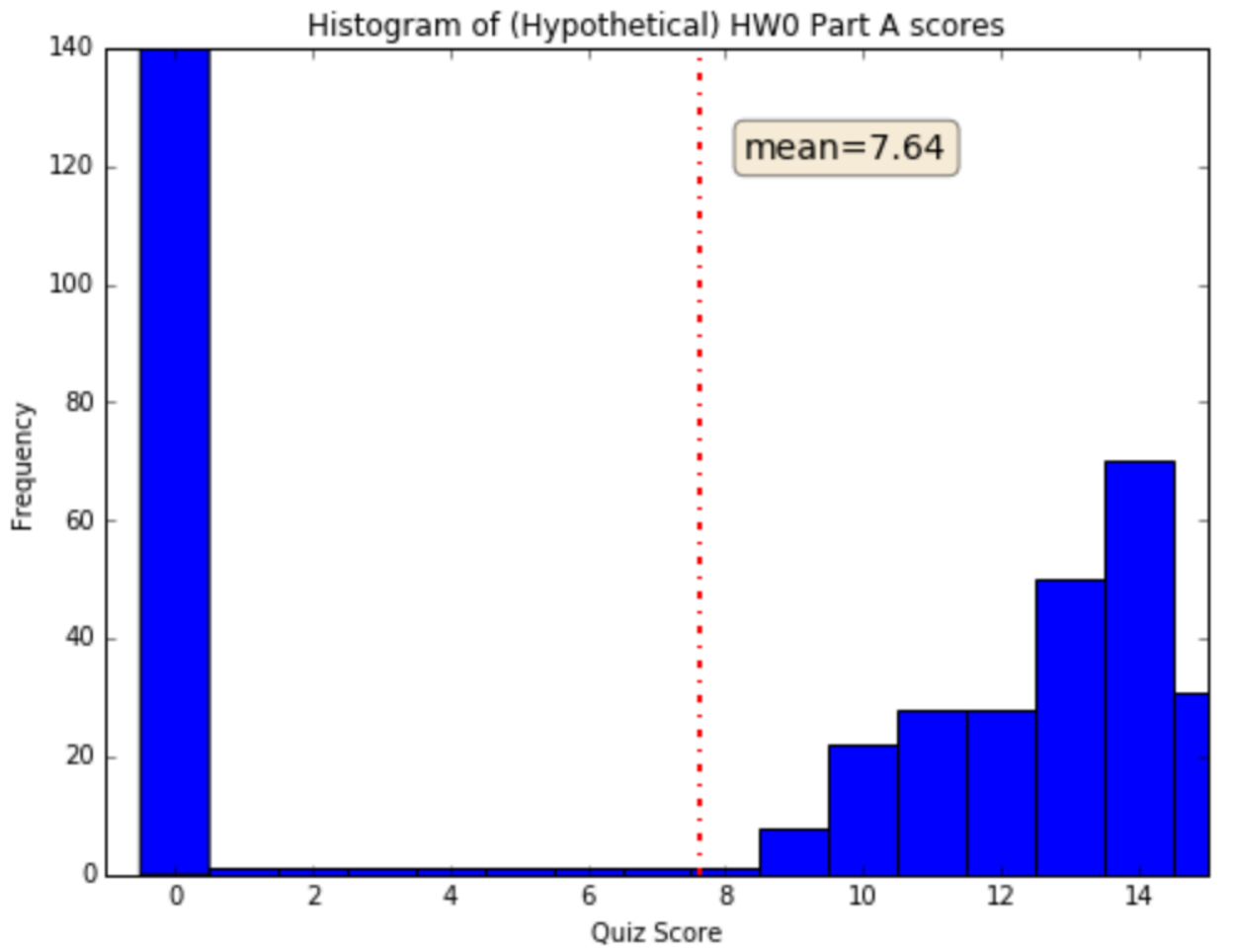
Why Data Visualization?

If I tell you that the average score for an assignment is: 7.64/15.

What does that suggest?

Why Data Visualization?

If I then show you the following graph, what does it suggest?



What is Data Visualization Good For?

Analyze:

- Identify hidden patterns and trends
- Help formulate/test hypothese
- Help determine the next step in analysis/modeling

What is Data Visualization Good For?

Communicate:

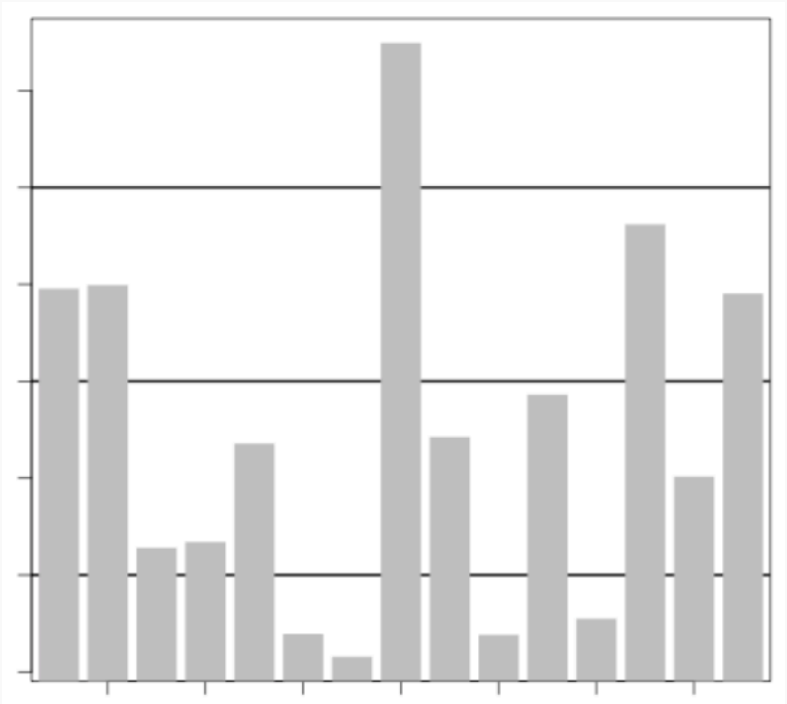
- Present information and ideas succinctly
- Provide evidence and support
- Influence and persuade

Visualization Design Principles

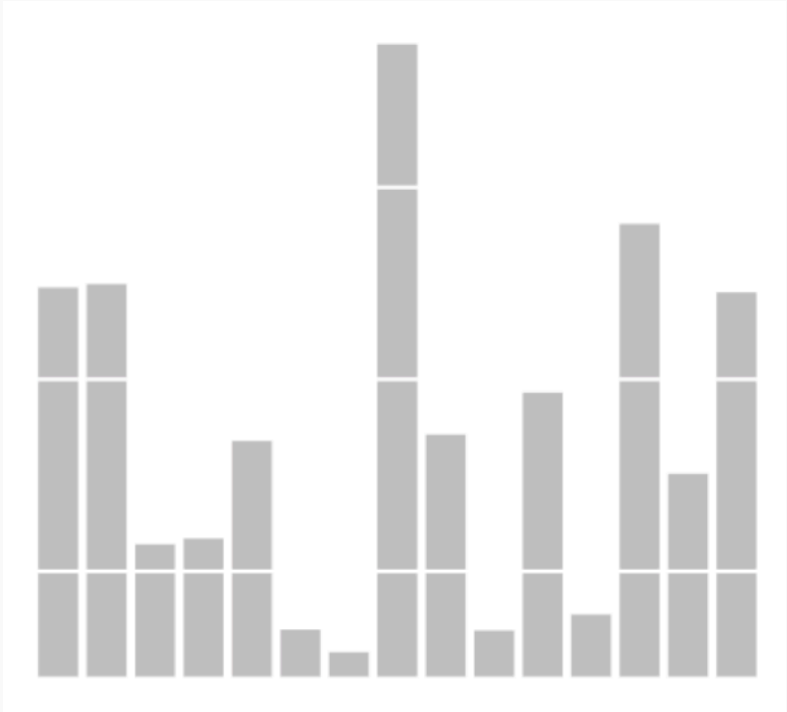
Basic data visualization guidelines from Edward Tufte:

- Maximize data to ink ratio: show the data

Bad



Better

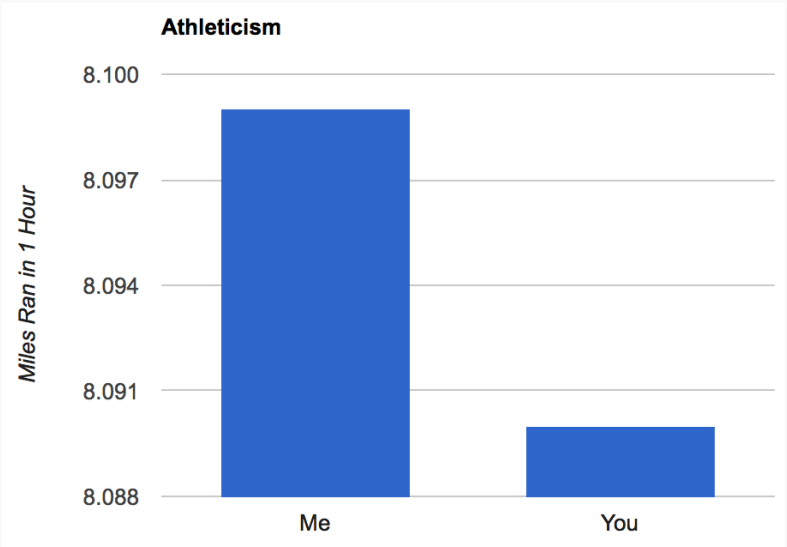


Visualization Design Principles

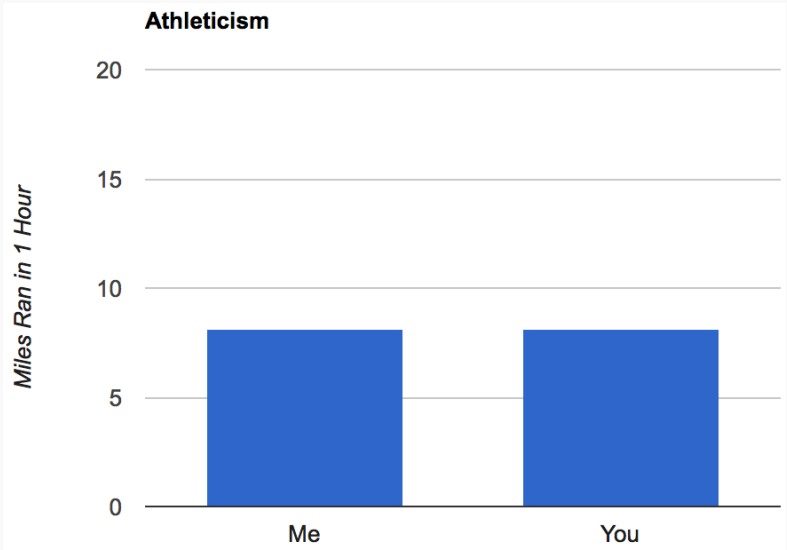
Basic data visualization guidelines from Edward Tufte:

- Maximize data to ink ratio: show the data
- Don't lie with scale: minimize $\frac{\text{size of effect in graph}}{\text{size of effect in data}}$ (Lie Factor)

Bad



Better

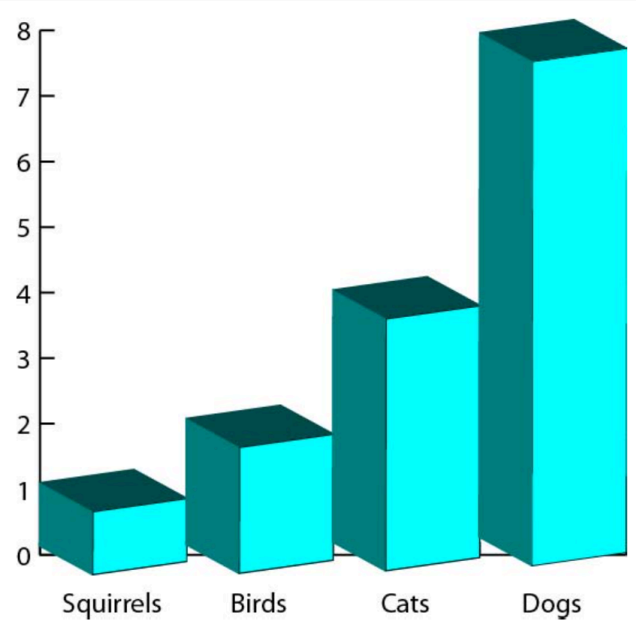


Visualization Design Principles

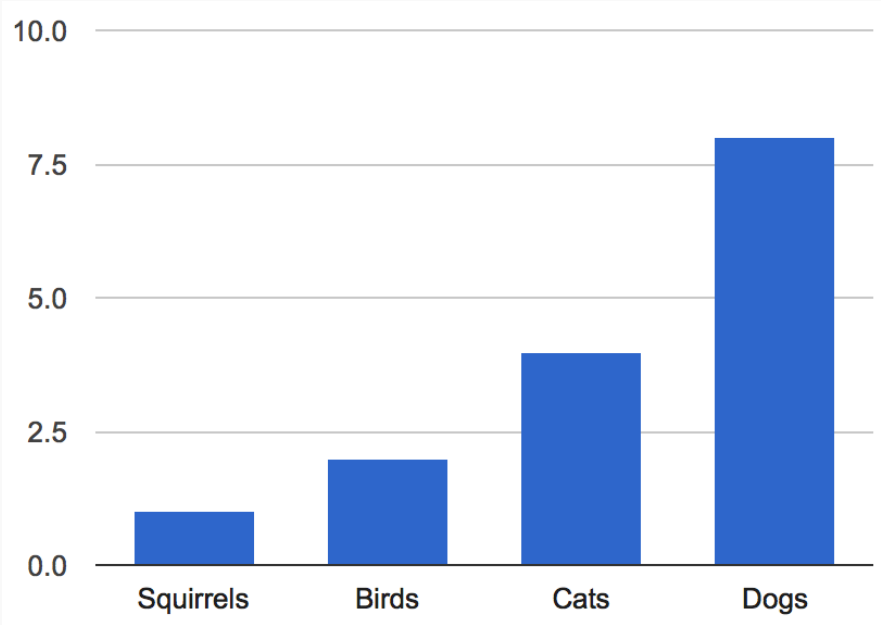
Basic data visualization guidelines from Edward Tufte:

- Maximize data to ink ratio: show the data
- Don't lie with scale: minimize $\frac{\text{size of effect in graph}}{\text{size of effect in data}}$ (Lie Factor)
- Minimize chart-junk: show data variation, not design variation

Bad



Better



Visualization Design Principles

Basic data visualization guidelines from Edward Tufte:

- Maximize data to ink ratio: show the data
- Don't lie with scale: minimize $\frac{\text{size of effect in graph}}{\text{size of effect in data}}$ (Lie Factor)
- Minimize chart-junk: show data variation, not design variation
- Clear, detailed and thorough labeling (including important events)

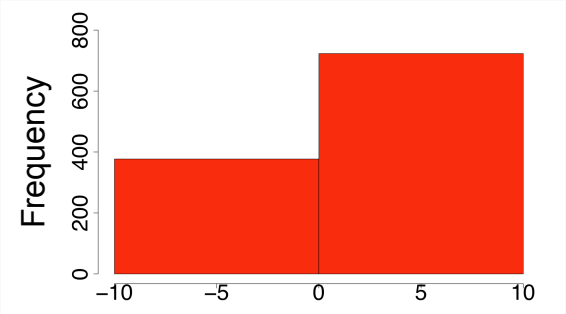
Types of Data Visualizations

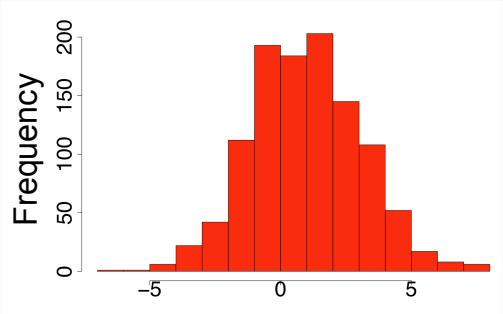
What do you want your visualization to show about your data?

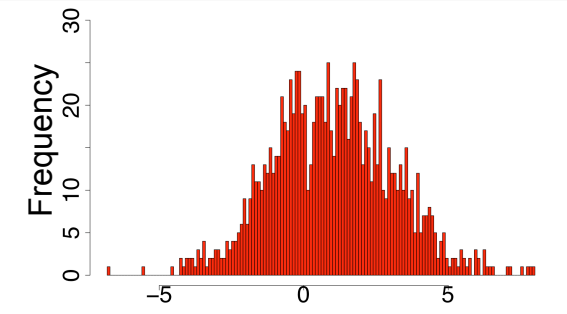
- **Distribution:** how a variable or variables in the dataset distribute over a range of possible values.
- **Relationship:** how the values of multiple variables in the dataset relate
- **Composition:** how the dataset breaks down into subgroups
- **Comparison:** how trends in multiple variable or datasets compare

Distribution

A **histogram** is a way to visualize how 1-dimensional data is distributed across certain values.







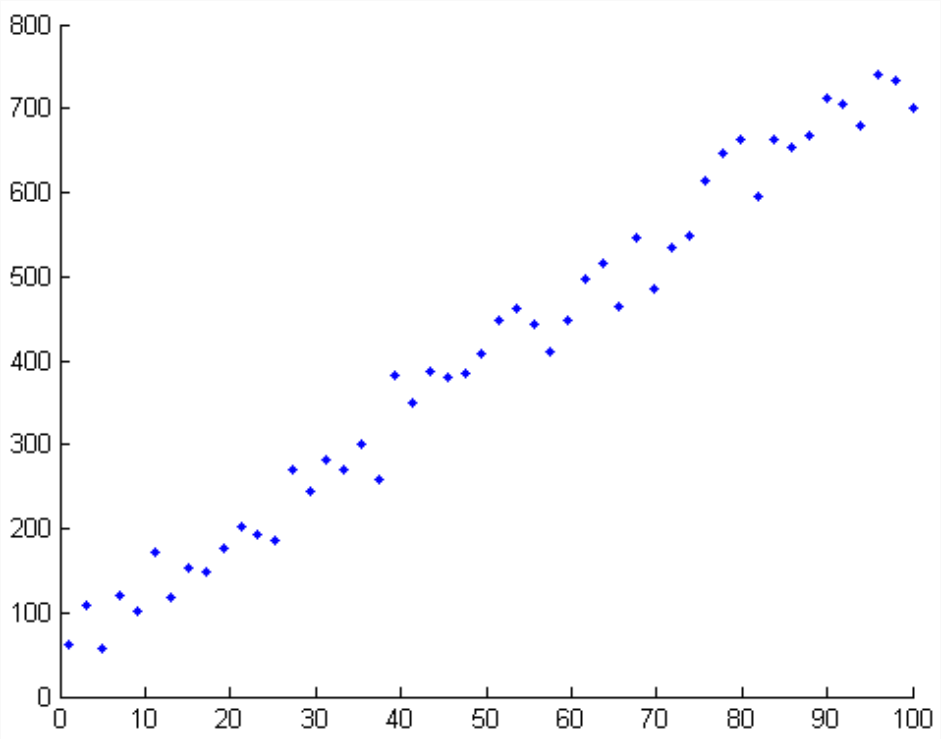
Note: Trends in histograms are sensitive to number of bins.

Distribution

A **scatter plot** is a way to visualize how multi-dimensional data is distributed across certain values.

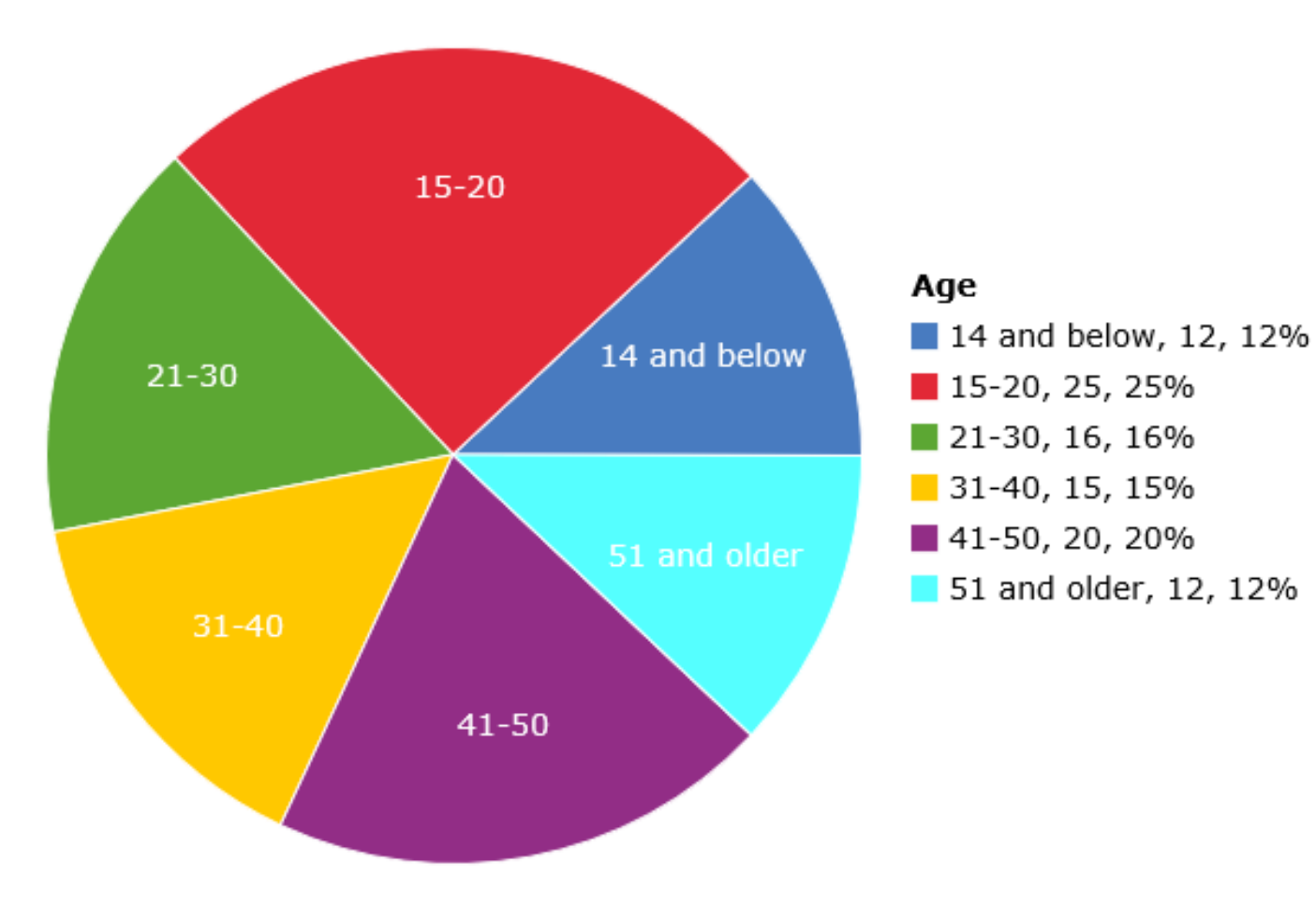
Relationships

A **scatter plot** is also a way to visualize the relationship between the different attributes of multi-dimensional data.



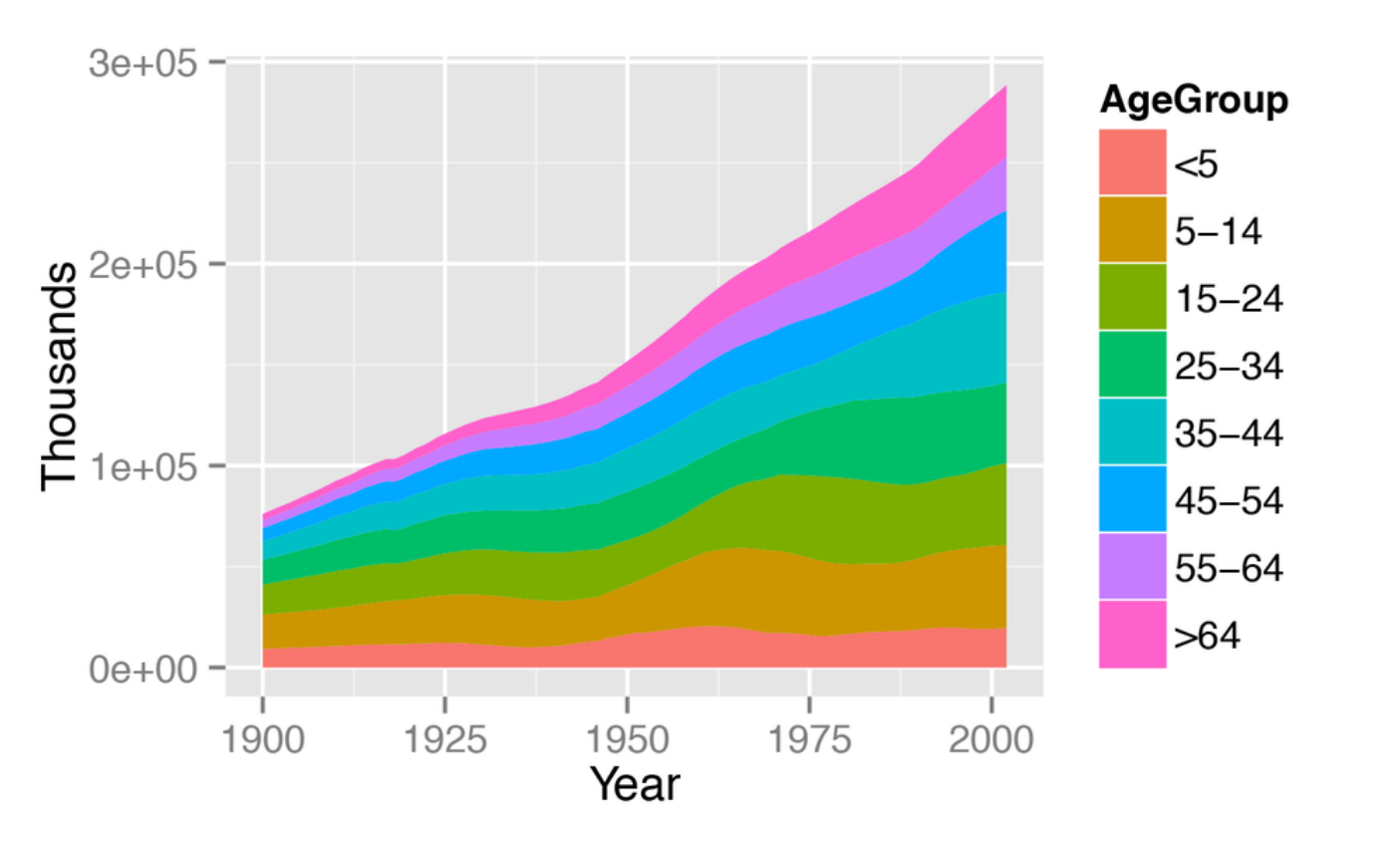
Composition

A **pie chart** is a way to visualize the static composition of a group.



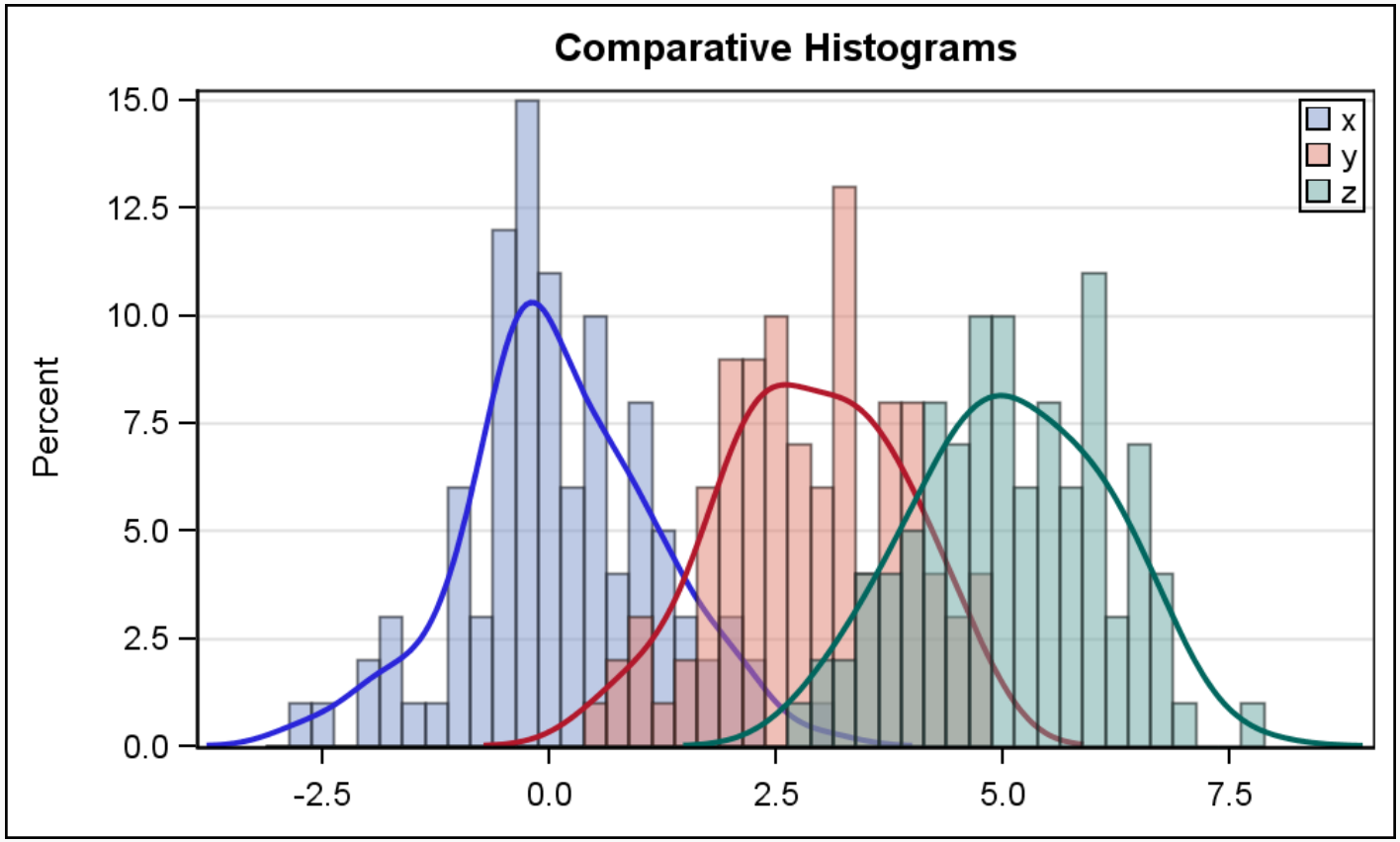
Composition

A **stacked area graph** is a way to visualize the composition of a group as it changes over time.



Comparisons

Plotting multiple histograms or curves on the same axes is a way to visualize how different variables compare.



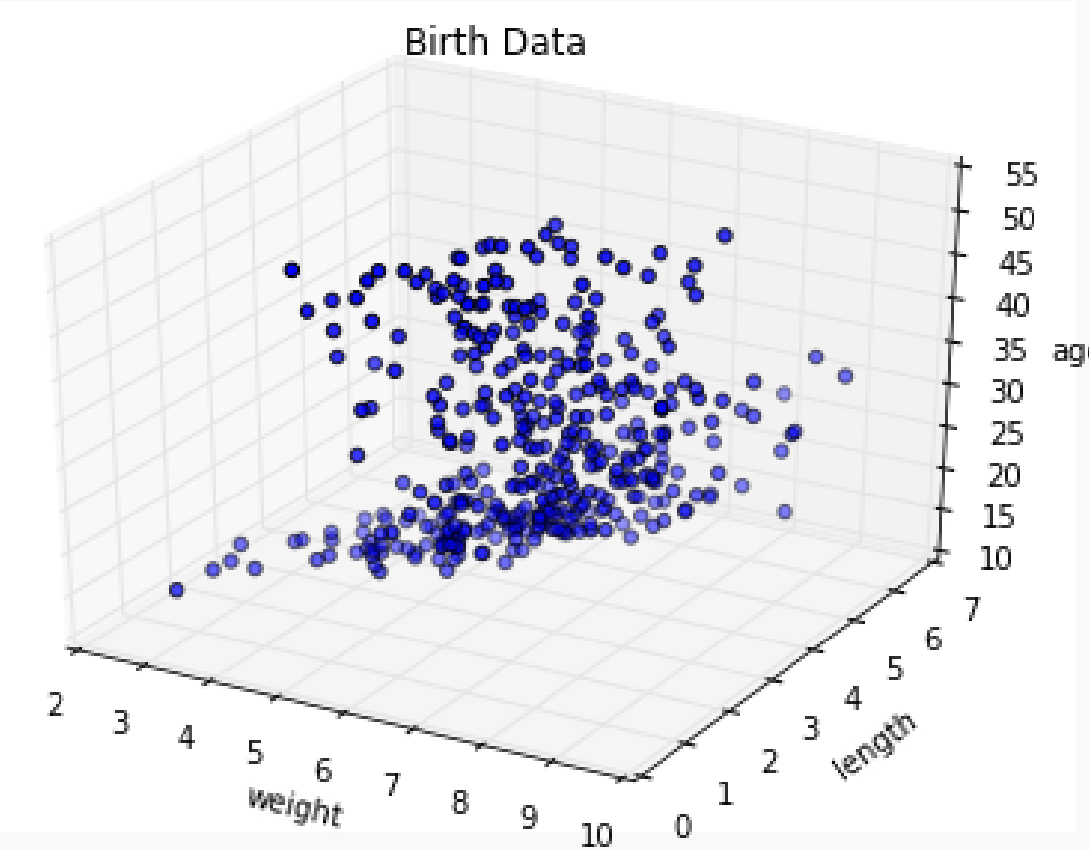
Visualizing the Impossible

Often your dataset seem too complex to visualize:

- Data is too high dimensional (how do you plot 100 variables on the same set of axes?)
- Some variables are categorical (how do you plot values like ‘Cat’ or ‘No’?)

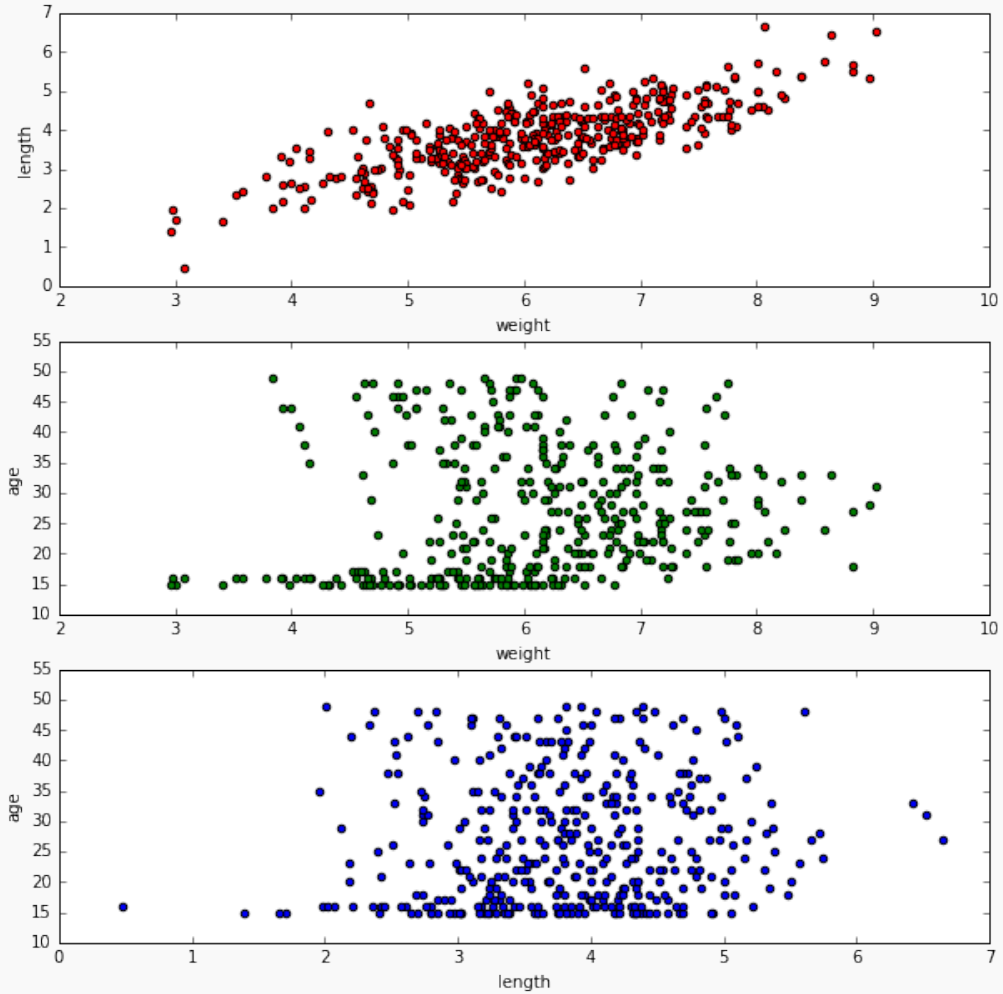
Reducing the Dimension

When the data is high dimensional, a scatter plot of all data attributes can be impossible or unhelpful.



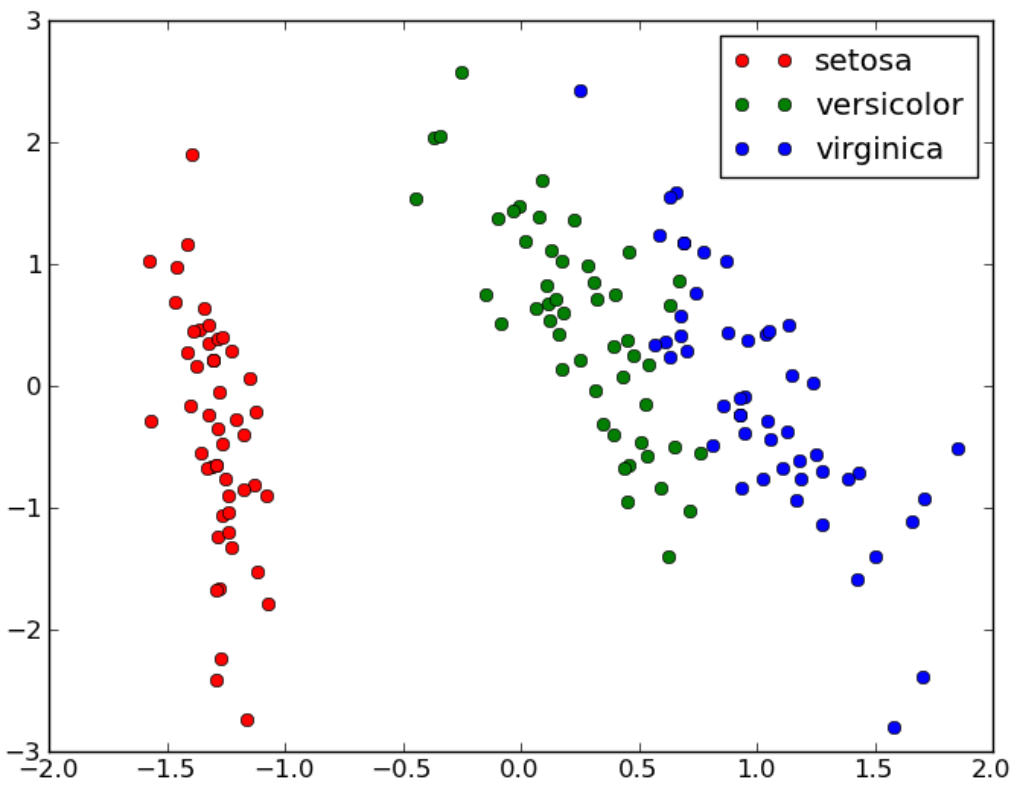
Reducing the Dimension

Relationships may be easier to spot by producing **multiple plots** of **lower dimensionality**.



Adding Extra Dimensions

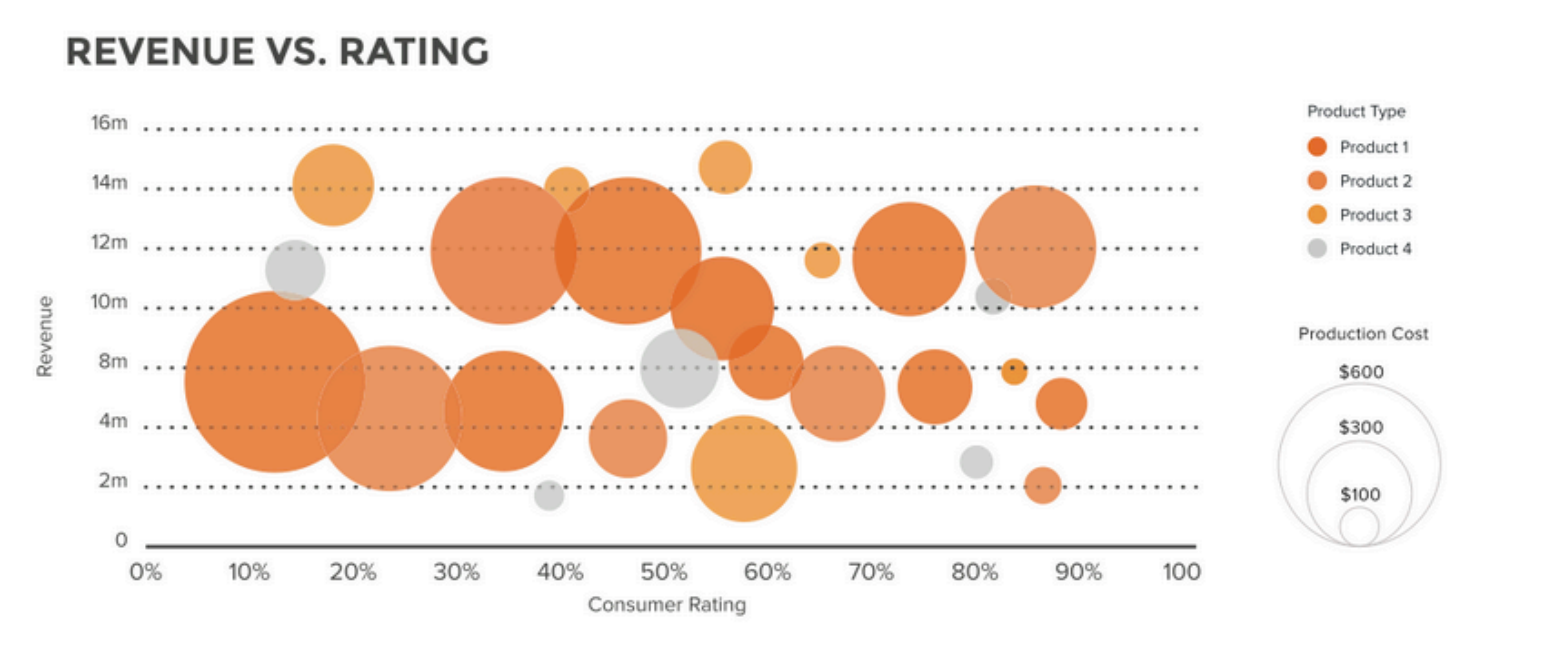
For 3D data, ***color coding*** a categorical attribute can be effective.



The above visualizes a set of Iris measurements. The variables are: **petal length, sepal length, Iris type** (setosa, versicolor, virginica).

Adding Extra Dimensions

For 3D data, a quantitative attribute can be encoded by ***size*** in a **bubble chart**.



The above visualizes a set of consumer products. The variables are: **revenue, consumer rating, product type and product cost**.

matplotlib

`matplotlib` is a plotting library, the `pyplot` module contains a set of functions especially useful for generating a wide range of simple plots.

Plotting function	Input	Result
`plt.plot(x, y)`	x-coords and y-coords	curve defined by the set of x, y coords
`plt.scatter(x, y)`	x-coords and y-coords	scatter plot defined by the set of x, y coords
`plt.hist(vals)`	an array or list of values	histogram of the list of values
`plt.title(plot_title)`	a string	adds title
`plt.show()`	none	displays all figures

Fancier Plotting: 3D, Subplots

To generate a group of 3 plots in a 3×1 grid, say. We want to explicitly create a figure and add subplots to particular positions of the grid.

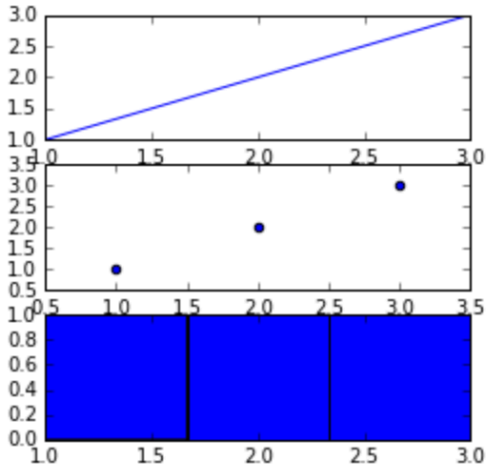
Function	Input	Result
`plt.figure()`	(optional) figure size	returns a new figure
`figure.add_subplot(n, m, k)`	row, column, subplot number	returns an axes for the k-th subplot in the nxm-grid column
`figure.add_subplot(n, m, k, projection='3d')`	row, column, subplot number, projection type	returns an axes for the k-th subplot in the nxm-grid column

You can do all your favorite plotting on the axes of each subplot.

Fancier Plotting: 3D, Subplots

To generate a group of 3 plots in a 3×1 grid, say. We want to explicitly create a figure and add subplots to particular positions of the grid.

```
In [72]: fig = plt.figure(figsize=(4, 4))

         ax1 = fig.add_subplot(311)
         ax1.plot([1, 2, 3], [1, 2, 3])

         ax2 = fig.add_subplot(312)
         ax2.scatter([1, 2, 3], [1, 2, 3])

         ax3 = fig.add_subplot(313)
         ax3.hist([1, 2, 3], bins=3)

         fig.tight_layout()
         plt.show()
```



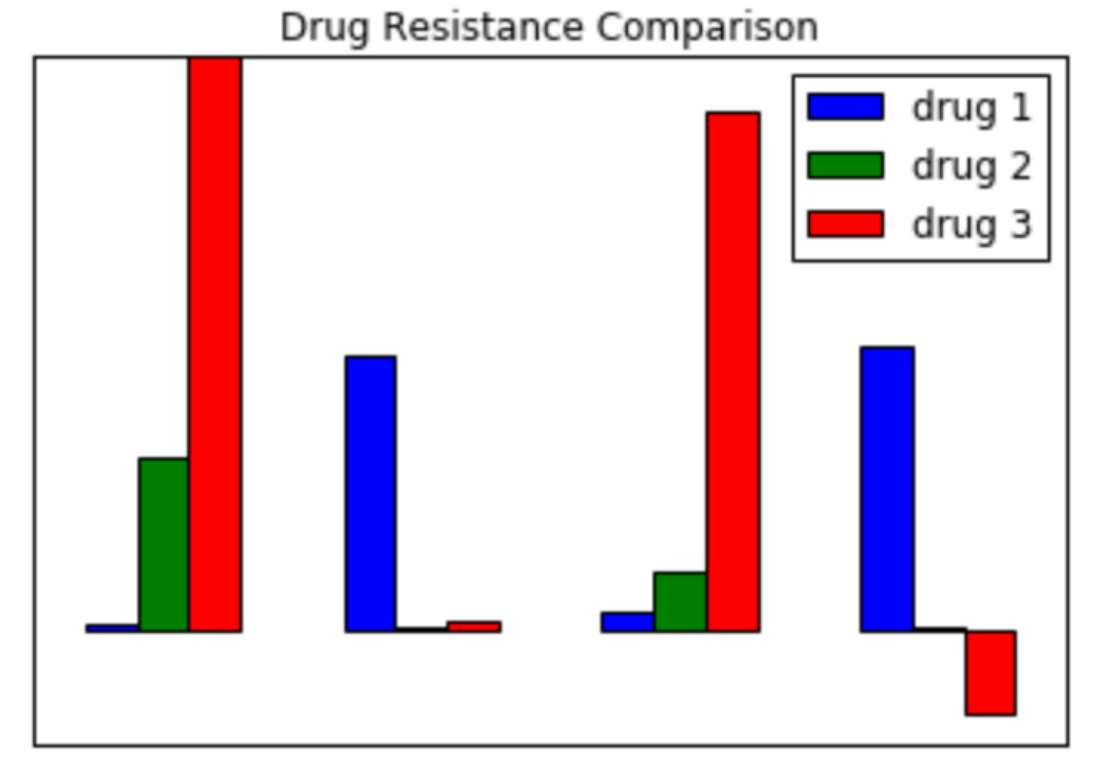
Try It Yourself

Use some simple graphs to explore the following dataset.

Bacteria Name	Group No.	Drug1 Res.	Drug2 Res.	Drug3 Res.
Brucella abortus	1	0.1	3	49
Diplococcus pneumoniae	2	4.75	0.007	0.125
Aerobacter aerogenes	1	0.3	1	47.2
Streptococcus viridans	2	4.9	0.03	-1.45

Try It Yourself

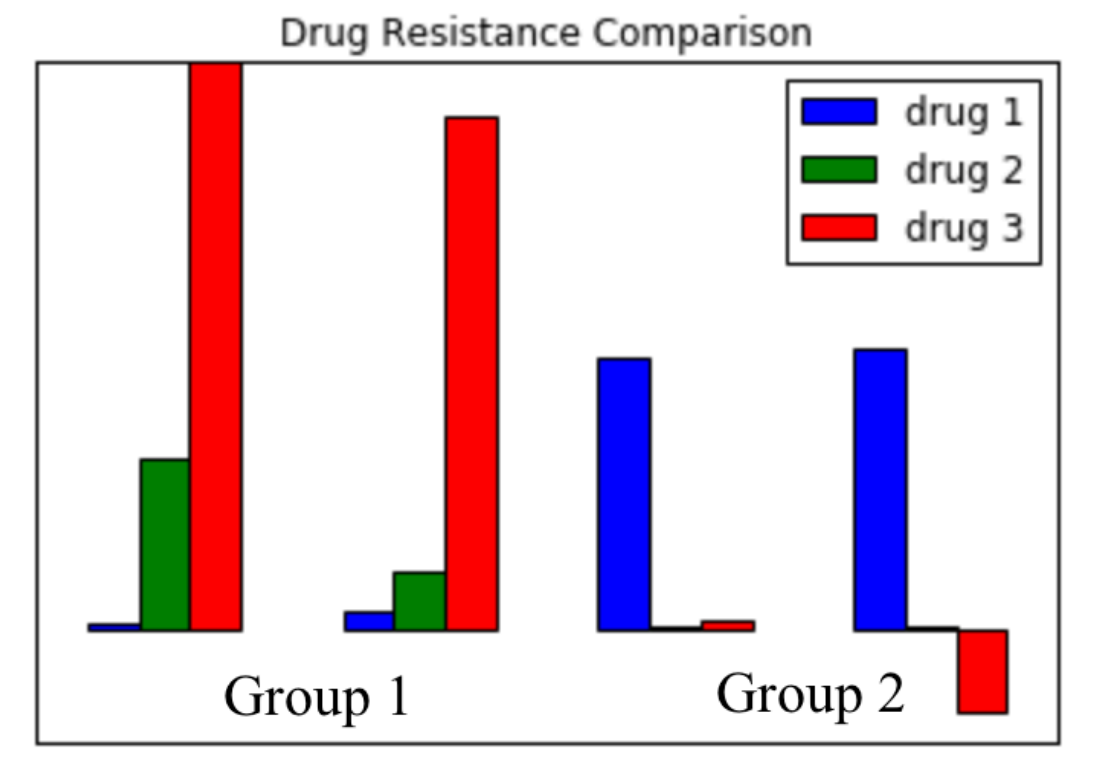
Bar graph showing resistance of each bacteria to each drug:



Any patterns?

Try It Yourself

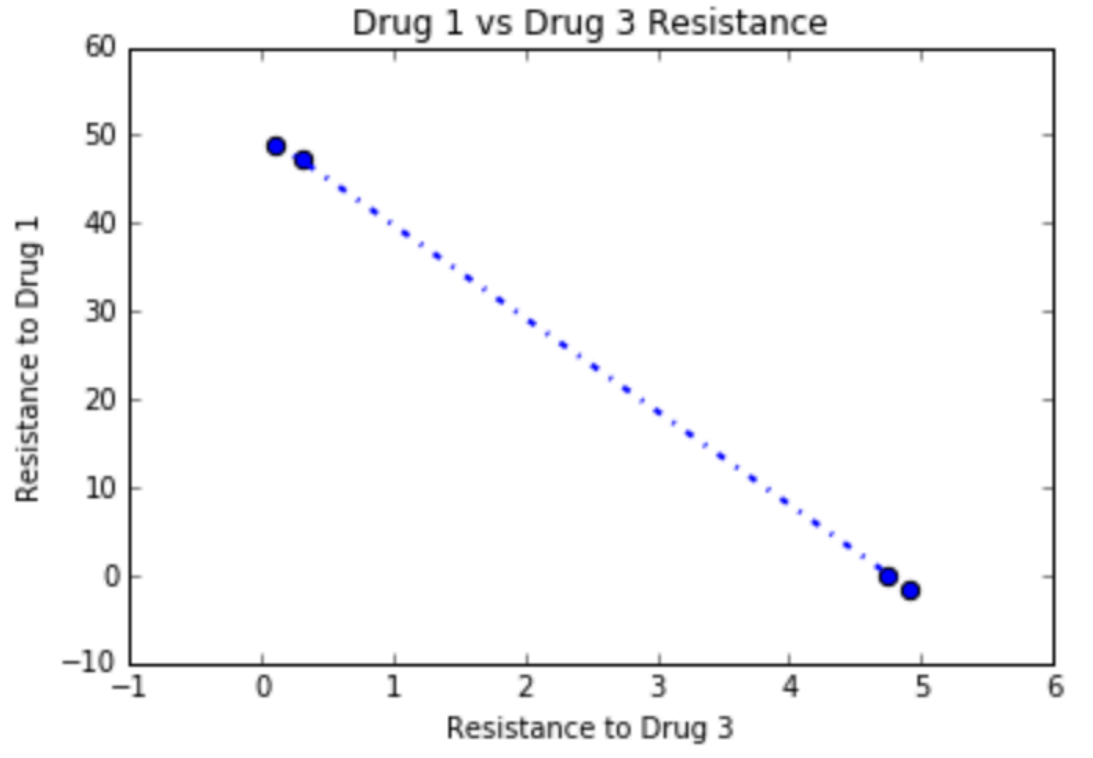
Bar graph showing resistance of each bacteria to each drug (grouped by Group Number):



Any patterns?

Try It Yourself

Scatter plot of Drug #1 vs Drug #3 resistance:



Note: The process of data exploration is iterative (visualize for trends, re-visualize to confirm)!