Data 301
Introduction to Data Science

Dennis Sun

March 30, 2016

Why are you here?

- #1 Job in America (Glassdoor)
- #1 Job for Work-Life Balance (Forbes)
- Median Salary: $117,000

Data Scientist FAQ

- **What do data scientists do?**
  They extract meaning from data.
- **How is a data scientist different from a statistician?**
  Data scientists are typically much more adept with computers than traditional statisticians, since they have to deal with messy and large data sets.

Is It All Hype?

Some people think data science will be automated within a decade.

Dashboards will automatically uncover interesting patterns in data and/or make it trivial to summarize data at the click of a button.

This is probably true. But complex and subtle analyses will still require a human for the foreseeable future.

Example: A supermarket tries a different arrangement of soft drinks on its shelves each week to determine which arrangement is best.

- **Which arrangement results in the most weekly revenue?** Probably automated within a few years.
- **But the week in which we had the most revenue was Thanksgiving! How do we account for this effect?** Will require careful human thinking for many years to come.

Goal of this Course

Therefore, the goal of this course is to expose you to different ways that data can be represented and organized.

The focus is **not** specific algorithms and tools because they will likely become obsolete within a few years.

1. Python
    - Lists
    - Dictionaries
2. Lab Introduction

Why Python for Data Science?

- Python has great built-in data structures (lists, dicts).
- Python has great string handling capabilities. (Necessary for working with messy data, which often involves processing text.)
- Python is extensible. You can use it for the entire data science pipeline—from scraping the data from the web, to analyzing the data, to deploying the web server that hosts your analysis.

1. Python
    - Lists
    - Dictionaries
2. Lab Introduction

Lists in Python

- **Lists** are Python's version of arrays.

```
fib = [1, 1, 2, 3, 5, 8]
```

- The items in a list do not all have to be of the same type.

```
weird = [1, 1, 2, 3, 5, 8, "x"]
```

- We can get a single element of a list in the normal way. (Note that Python uses 0-based indexing.)

```
fib[0]   # returns 1
fib[5]
```

- We can get a subset of consecutive elements by specifying a range:

```
fib[0:2]   # returns [1, 1]
fib[:2]
fib[2:]
fib[-2:]
fib[1:4]
```

List Methods

- We can add an element to a list using the `.append()` method:

  ```
  fib.append(13)
  ```

  Note that this appends the element to the list *in place*. Contrast this with R, where the vector has to first be copied:

  ```
  fib <- c(fib, 13)
  ```

- We can sort a list using the `.sort()` method:

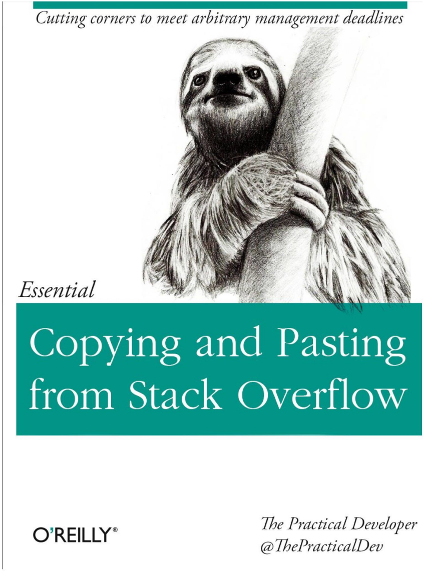
  ```
  fib.sort(reverse=True)
  ```

  Again, sorting is done *in place*.

- Check the Python documentation for all the methods you can use with lists:

  `https://docs.python.org/3/tutorial/datastructures.html`

An Essential Skill



Cutting corners to meet arbitrary management deadlines

Essential

Copying and Pasting from Stack Overflow

O'REILLY®

The Practical Developer
@ThePracticalDev

Iterating Over Lists

In Python, `for` loops iterate over lists directly:

```
for x in fib:
        print(x)
```

No nonsense like

```
for(i=0; i<n; i++) {
        printf("%d", x[i])
}
```

But what if you wanted to know the index `i` as well?

Try using the `enumerate()` function:

```
for i, x in enumerate(fib):
        print(i, x)
```

List Comprehensions

In-Class Exercise

How would you create a list `squares` *containing the first 100 perfect squares (i.e.,* `[0, 1, 4, 9, ...]`*)?*

Hint: *You may want to use* `range(100)`.

Naively, you might do something like this:

```
squares = []
for x in range(100):
        squares.append(x**2)
```

This is ugly! We have to initialize an empty list and then append elements one by one.

A nicer solution is to use **list comprehensions**:

```
squares = [x**2 for x in range(100)]
```

1. Python
    - Lists
    - Dictionaries
2. Lab Introduction

Dict(ionarie)s

Suppose we wanted to count up how many times each word appeared in a text. Would it be a good idea to store these counts in a list?

No! We need to be able to associate each count with a word and be able to look up the counts efficiently!

This is what **dictionaries** (or **dicts**) are for!

```
word_counts = {
        "a": 1052,
        "an": 216,
        "any": 76,
        ...
}
```

You can “look up” the **value** for any **key**: `word_counts["any"]`.

You assign a **value** for a **key** as follows: `word_counts["on"] = 91`.

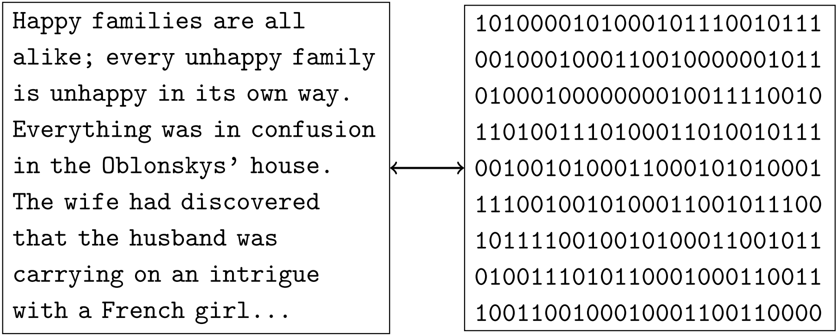
Word Counts

In-Class Exercise

Open up the Jupyter notebook called "In Class Exercise - Word Counts" and fill in the cells to count up the words in War and Peace, *a long novel by Leo Tolstoy.*

1. Python
    - Lists
    - Dictionaries
2. Lab Introduction

Compression



```
Happy families are all
alike; every unhappy family
is unhappy in its own way.
Everything was in confusion
in the Oblonskys’ house.
The wife had discovered
that the husband was
carrying on an intrigue
with a French girl...
```

```
1010000101000101110010111
0010001000110010000001011
0100010000000010011110010
1101001110100011010010111
0010010100011000101010001
1110010010100011001011100
1011110010010100011001011
0100111010110001000110011
1001100100010001100110000
```

We’ll have a **lookup table** that maps each character to a code:

```
{ 'a': '1010001',
  'b': '0011110',
  'c': '1011101',
  'd': '1011010',
  'e': '0100011',
  'f': '0000101',
  ...                }
```

In this coding, every code has the same length.

Wouldn’t it be better to use shorter codes for characters that appear more often?

Compression

Huffman Coding

```
{ 'a': .060,            { 'a': '1000',
  'b': .010,              'b': '1100101',
  'c': .017,              'c': '110000',
  'd': .034,      ⟹       'd': '10101',
  'e': .094,              'e': '000',
  'f': .015,              'f': '100100',
  ...       }             ...              }
```

We can show that, on average, this coding will result in shorter encoded texts.