Data Acquisition

Get started with a workflow, read data from various sources

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com

Table of Contents

Methods

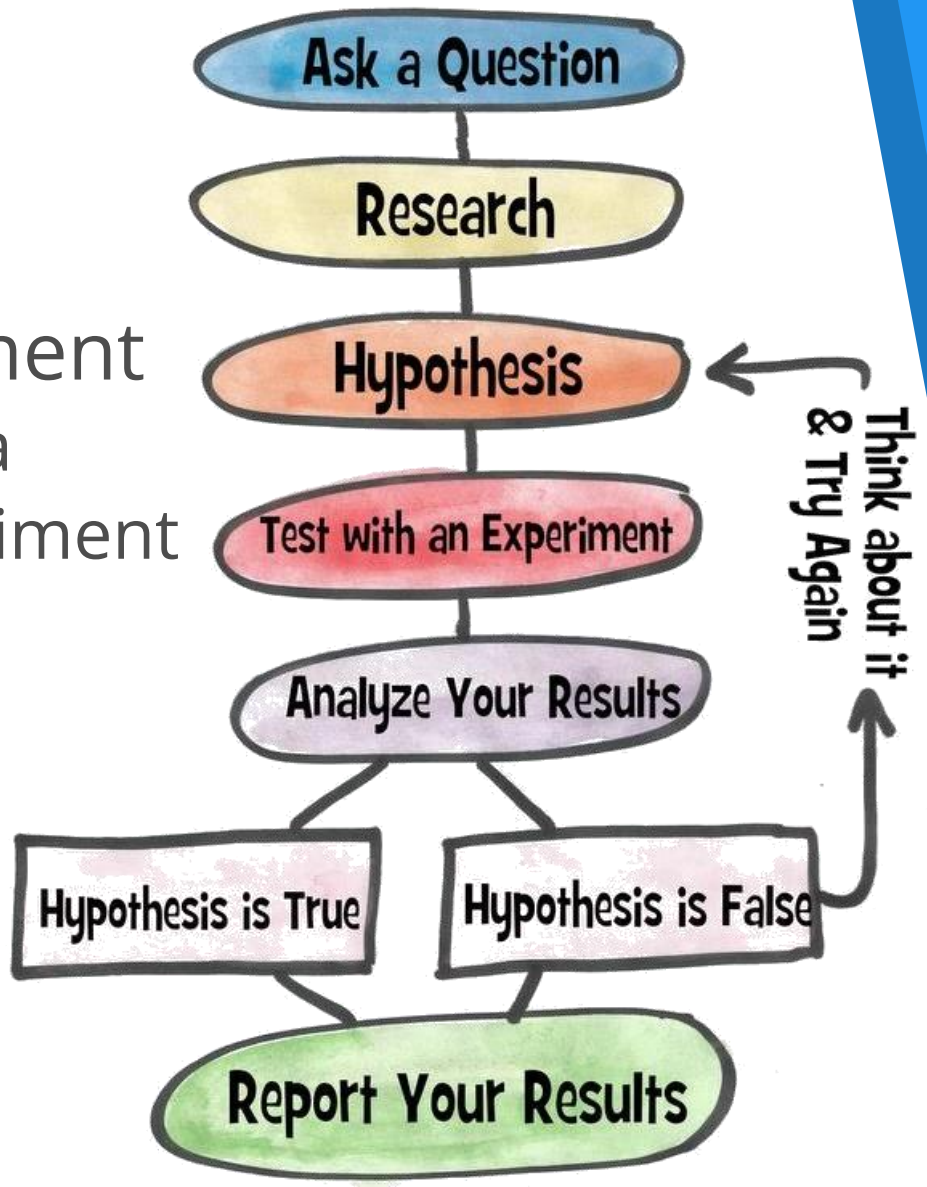
How not to get lost

Divide and conquer

- Useful for any kind of problem
  - Especially in algorithms and debugging
  - … also when invading countries
- Assumption: Complicated things are a combination of very simple things
  - Algorithms: Merge sort, Fourier transform
  - Software architecture:
    - "I want to build an ecommerce system"
      => I want shop owners to add new products
      => I want to store products in the DB => …
      => `def save_product(name, price)`
  - Debugging
    - The bug is somewhere in my code => …
      => the bug is `">="` instead of `">"` on line 45 in `user.py`

The Scientific Method Steps

- Ask a question
- Do some research
- Form a hypothesis
- Test the hypothesis with an experiment
  - Experiment works => Analyze the data
  - Experiment doesn't work => Fix experiment
- Results align with hypothesis => OK
- Results don't align with hypothesis => new question, new hypothesis
- Communicate the results



Why use the Scientific Method?

- Useful when we're exploring something new
  - A new algorithm
  - A new codebase we've just been hired to work on
- Based on common logic
- Experiments
- **Example:** performance testing
  - **Research:** My logs show that this Web page on my server takes too much time to load
  - **Hypothesis:** This piece of code is too slow. I need to improve it
  - **Control:** Measure the runtime (in seconds)
  - **Experiment:** Try to fix the problem and repeat the runtime test
    - Did the fix bring a considerable performance gain?
  - **Communication:** Show the results and implement the fix

Setting Up Our Environment

Getting ready is easy,
getting up in the morning is not

Anaconda

- You can install the Python interpreter and all libraries manually
  - Hard, boring and repetitive work
  - Error-prone
- Easy solution: platforms like **Anaconda**
  - Everything you need to get started with Python for science: Python interpreter, packages (720+), package manager, IDE
- Download from https://www.anaconda.com/download/
- Current version (Nov 2017): Anaconda 5.0.1
  - Choose your platform (Windows, Linux, or MacOS)
  - Download the **Python 3.6** version
  - Follow the installer

Python Tools for Visual Studio (Optional)

- You can use the built-in IDE called **Spyder**
  - You can even use Notepad if that's your thing
- If you want to use another IDE, you have to configure it to work with Python
  - Syntax highlighting, autocomplete, etc.
- If you're using Visual Studio
  - Python Tools
  - https://www.visualstudio.com/vs/python/
- Visual Studio Code
  - If you prefer something lightweight, Visual Studio Code is a good alternative
  - https://code.visualstudio.com/docs/languages/python

Python Online

- There are places where you can execute your code online
  - If you don't have access to Anaconda
  - Or you want to test something very quickly
- https://www.python.org/shell/
  - Provides a Python shell
- https://www.pythonanywhere.com/try-ipython/
  - Provides an implementation of IPython (Interactive Python)
  - REPL (Read-Execute-Print Loop)
  - No major difference to the Python shell
- To share your code, you can use http://ideone.com, http://pythonfiddle.com/ or http://pastebin.com/

Jupyter Notebook

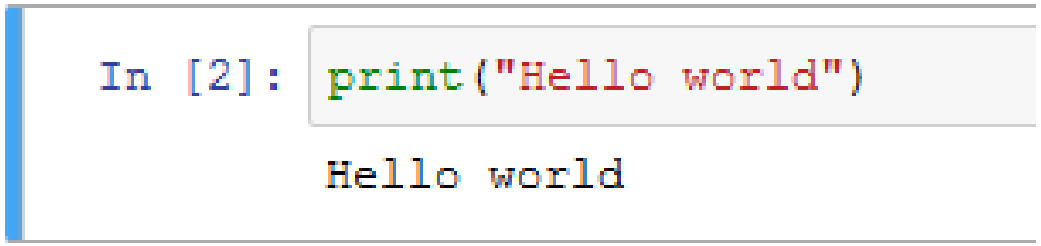
- A very nice and clean way to document your research
- Included in Anaconda
- Can create documents that contain live code, equations, visualizations and explanatory text
  - HTML / CSS / JavaScript
  - Markdown
  - LaTeX
  - Python
- Start – use the Anaconda shortcut
  - …or type into the Command Prompt

```
jupyter notebook
```

How to Use Jupyter

- Create a new notebook
  - New > Python 3
- Every piece of text or code is in a cell
  - Text cells just contain text or Markdown

  - Code cells contain code (obviously)
  - Code can be executed
  - Jupyter "remembers" the code



- Execute cell: **`Ctrl + Enter`**
  - Or use the menus

Getting Data

Reading data from various sources

The pandas Library

- Provides a way to read and work with data
  - Table (`DataFrame`)
    - May have many dimensions
    - We usually call this a “dataset”
  - List (`Series`)
    - One-dimensional
- Usage

```
import pandas as pd
```

- General requirements
  - Rows and columns are indexed, columns may have names
  - Each column has a fixed data type
    - Python will try to infer the best type according to the data

Data Sources

- In order to work with the data, we need to represent it in a tabular form
  - Sometimes our data is tabular – we just need to read it
  - In other cases, we need to create our tables
    - Unstructured data: data that doesn't have a **model**
      - There is some structure, it's just not very clear
      - Examples: Images, plain text, audio, web pages
- Most common sources
  - Tables in a text format such as .csv
  - Spreadsheets (such as Excel or Google Sheets)
  - Web services
  - Databases

Reading a Local File

- Let's read the file `accidents.csv`
  - Copy the file to a data folder
    - Not required, just makes working with many data files easier
  - Inspect the file (use a text editor or Excel) just to see what it contains
  - ```
    accidents_data = pd.read_csv("data/accidents.csv")
    ```
    - `read_csv()` docs
- You'll see that all `read_*()` functions have a lot of optional arguments
  - They make working with different formats easy, e.g.
    - Instead of "True" and "False", the table contains "Yes" and "No"
    - The actual table starts at line 30 of the file
    - There are blank / comment lines which should be skipped
    - There are no column names in the file
    ```

Exploring the Dataset

- In Python, we can print the variable

```
print(accidents_data)
```

- Even better, in Jupyter, a cell outputs its last returned value
  - This will create a nicer output

```
accidents_data
```

- We can see that
  - Rows have numerical indices starting at 0 by default
  - Columns have names taken from the first line in the .csv file
- Column names: `accidents_data.columns`
- Index values: `accidents_data.index`
- Dimensions: `accidents_data.shape`
  - Format: `(rows, columns)`

Reading Data from Other Files

- The process is very similar
- Other text-based formats
  - `pd.read_table()` is the most general function
    - All others (`read_csv()`, `read_fwf()`, etc.) just apply some settings
  - If we come across a file, we can apply our own settings
    - The point is to match the format in the best possible way
    - Example: AutoMPG dataset
- Excel
  - Read the "green_tripdata_2015-09.xls" file using `pd.read_excel()`
  - Explore the file dimensions

Reading Data from Web Services

- Web services work over the HTTP protocol and provide data in several formats
  - Most commonly used: JSON and XML
  - Some APIs to try
- Example: Open Library API
  - We want information about books with ISBNs
    - Example: these 4 books
    - We can put the URL directly, `pandas` will perform a GET request
  - Function: `pd.read_json()`
    - We can provide the parameter `orient = "index"` to arrange the dataset better
      - Books should be placed by rows, their properties – by columns
      - More details on this – next time
  - More complex queries require more pre-processing

Reading Data from SQL

- Relational databases store data in tables
  - Very similar to the datasets we use
- First, install a library to connect to databases
  - From the command line: `conda install pyodbc`
- Then, import the library and connect to the database
  - **Note:** The sample credentials will change after the lecture

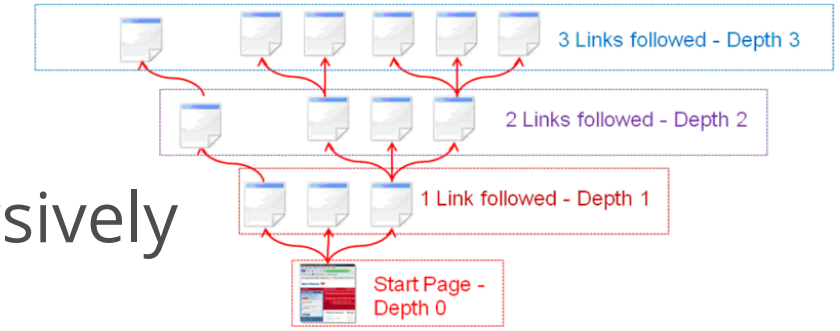
```
import pyodbc
conn = pyodbc.connect("DRIVER={SQL Server};conn_string")
```

- Perform a query

```
laptops = pd.read_sql("select * from Laptop", conn)
```

Web Scraping

- Another method for getting data
- Sometimes combined with crawling
  - Traversing a Web page structure recursively
- Basic procedure
  - Read a Web page as HTML
  - Use the HTML to obtain the data
    - A webpage is unstructured
    - We need to create and maintain the structure
    - We usually need more libraries to do that
- Examples
  - Get all job listings from a website
  - Get user contact details from a Web page



Using Multiple Sources

Constraints and Validity

Data Guidelines

- Some queries will not be simple
  - E.g. scraping, dealing with "freeform" text, audio data, networks
  - We need to create a tabular structure from the raw data
    - How? We'll discuss this later in the course
- After we read the data, we have to ensure it's been read without errors
  - A very simple first check: check the dimensions (`dataframe.shape`) and show the first few rows (`dataframe.head()`)
  - We may need to rename columns
  - We may need to perform different manipulations to ensure the data is in a proper state
    - We'll do this in the next lectures

Merging Many Data Sources

- **Automate the process** as much as possible
  - From reading the raw data to getting the processed dataset
  - If the dataset changes or updates, you'll just re-run your code
- **Document the process**
- Create as few datasets as possible
  - I.e. merge many sources into one table if you can
    - We'll talk more about combining relations next time
- Ensure the different sources are compatible and consistent
  - If they aren't, process the raw data
    - Most common example: Mismatched IDs
- Make sure all column types are correct
  - Check: `dataframe.dtypes`
    - Example: `"str"` type for a numeric column

Summary

- Methods
  - Divide and conquer
  - Scientific method
- Setting up the environment
- Reading data from different sources
  - Text files
  - Excel
  - Web services
  - SQL databases
- Data consolidation principles

Questions?