Downloading data using curl

DATA PROCESSING IN SHELL

Susan Sun
Data Person

What is curl?

`curl` :

- is short for **C**lient for **URL**s
- is a Unix command line tool
- transfers data to and from a server
- is used to download data from HTTP(S) sites and FTP servers

Checking curl installation

Check `curl` installation:

```
man curl
```

If `curl` has **not** been installed, you will see:

```
curl command not found.
```

For full instructions, see https://curl.haxx.se/download.html.

Browsing the curl Manual

If `curl` is installed, your console will look like this:

```
curl(1)                                  Curl Manual                                  curl(1)

NAME
       curl - transfer a URL

SYNOPSIS
       curl [options] [URL...]

DESCRIPTION
       curl  is  a  tool  to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP,
       FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP,  RTSP,  SCP,  SFTP,  SMB,  SMBS,  SMTP,
       SMTPS, TELNET and TFTP). The command is designed to work without user interaction.

       curl offers a busload of useful tricks like proxy support, user authentication, FTP upload, HTTP post, SSL con-
       nections, cookies, file transfer resume, Metalink, and more. As you will see below, the number of features will
:
```

Browsing the curl Manual

Press `Enter` to scroll.

```
       curl [options] [URL...]

DESCRIPTION
       curl  is  a  tool  to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP,
       FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP,  RTSP,  SCP,  SFTP,  SMB,  SMBS,  SMTP,
       SMTPS, TELNET and TFTP). The command is designed to work without user interaction.

       curl offers a busload of useful tricks like proxy support, user authentication, FTP upload, HTTP post, SSL con-
       nections, cookies, file transfer resume, Metalink, and more. As you will see below, the number of features will
       make your head spin!

:
```

Press `q` to exit.

Learning curl Syntax

Basic `curl` syntax:

```
curl [option flags] [URL]
```

URL is required.

`curl` also supports `HTTP`, `HTTPS`, `FTP`, and `SFTP`.

For a full list of the options available:

```
curl --help
```

Downloading a Single File

Example:

A single file is stored at:

```
https://websitename.com/datafilename.txt
```

Use the optional flag `-O` to save the file with its original name:

```
curl -O https://websitename.com/datafilename.txt
```

To rename the file, use the lower case `-o` + new file name:

```
curl -o renameddatafilename.txt https://websitename.com/datafilename.txt
```

Downloading Multiple Files using Wildcards

Oftentimes, a server will host multiple data files, with similar filenames:

```
https://websitename.com/datafilename001.txt
https://websitename.com/datafilename002.txt
...
https://websitename.com/datafilename100.txt
```

Using Wildcards (*)

Download every file hosted on `https://websitename.com/` that starts with `datafilename` and ends in `.txt` :

```
curl -O https://websitename.com/datafilename*.txt
```

Downloading Multiple Files using Globbing Parser

Continuing with the previous example:

```
https://websitename.com/datafilename001.txt
https://websitename.com/datafilename002.txt
...
https://websitename.com/datafilename100.txt
```

Using Globbing Parser

The following will download every file sequentially starting with `datafilename001.txt` and ending with `datafilename100.txt` .

```
curl -O https://websitename.com/datafilename[001-100].txt
```

Downloading Multiple Files using Globbing Parser

Continuing with the previous example:

```
https://websitename.com/datafilename001.txt
https://websitename.com/datafilename002.txt
...
https://websitename.com/datafilename100.txt
```

Using Globbing Parser

Increment through the files and download every Nth file (e.g. `datafilename010.txt` , `datafilename020.txt` ,… `datafilename100.txt`)

```
curl -O https://websitename.com/datafilename[001-100:10].txt
```

Preemptive Troubleshooting

`curl` has two particularly useful option flags in case of timeouts during download:

- `-L` Redirects the HTTP URL if a 300 error code occurs.
- `-C` Resumes a previous file transfer if it times out before completion.

Putting everything together:

```
curl -L -O -C https://websitename.com/datafilename[001-100].txt
```

- All option flags come before the URL
- Order of the flags does not matter (e.g. `-L -C -O` is fine)

Happy curl-ing!

DATA PROCESSING IN SHELL

Downloading data using Wget

DATA PROCESSING IN SHELL

Susan Sun
Data Person

What is Wget?

`Wget` :

- derives its name from **World Wide Web** and **get**
- native to Linux but compatible for all operating systems
- used to download data from HTTP(S) and FTP
- better than **curl** at downloading multiple files recursively

Checking Wget Installation

Check if `Wget` is installed correctly:

```
which wget
```

If `Wget` **has** been installed, this will print the location of where `Wget` has been installed:

```
/usr/local/bin/wget
```

If `Wget` **has not** been installed, there will be no output.

Wget Installation by Operating System

Wget source code: https://www.gnu.org/software/wget/

Linux: run `sudo apt-get install wget`

MacOS: use homebrew and run `brew install wget`

Windows: download via **gnuwin32**

Browsing the Wget Manual

Once installation is complete, use the `man` command to print the `Wget` manual:

```
WGET(1)                                                    GNU Wget                                                    WGET(1)

NAME
       Wget - The non-interactive network downloader.

SYNOPSIS
       wget [option]... [URL]...

DESCRIPTION
       GNU Wget is a free utility for non-interactive download of files from the Web.  It supports HTTP, HTTPS, and FTP protocols,
       as well as retrieval through HTTP proxies.

       Wget is non-interactive, meaning that it can work in the background, while the user is not logged on.  This allows you to
       start a retrieval and disconnect from the system, letting Wget finish the work.  By contrast, most of the Web browsers
       require constant user's presence, which can be a great hindrance when transferring a lot of data.
```

Learning Wget Syntax

Basic `Wget` syntax:

`wget [option flags] [URL]`

URL is required.

`Wget` also supports `HTTP`, `HTTPS`, `FTP`, and `SFTP`.

For a full list of the option flags available, see:

```
wget --help
```

Downloading a Single File

Option flags **unique** to `Wget` :

`-b` : Go to background immediately after startup

`-q` : Turn off the `Wget` output

`-c` : Resume broken download (i.e. continue getting a partially-downloaded file)

```
wget -bqc https://websitename.com/datafilename.txt
```

```
Continuing in background, pid 12345.
```

Have fun Wget-ing!

DATA PROCESSING IN SHELL

Advanced downloading using Wget

DATA PROCESSING IN SHELL

Susan Sun
Data Person

Multiple file downloading with Wget

Save a list of file locations in a text file.

```
cat url_list.txt
```

```
https://websitename.com/datafilename001.txt
https://websitename.com/datafilename002.txt
...
```

Download from the URL locations stored within the file `url_list.txt` using `-i` .

```
wget -i url_list.txt
```

Setting download constraints for large files

Set upper download bandwidth limit (by default in **bytes per second**) with `--limit-rate` .

Syntax:

```
wget --limit-rate={rate}k {file_location}
```

Example:

```
wget --limit-rate=200k -i url_list.txt
```

Setting download constraints for small files

Set a mandatory pause time (in seconds) between file downloads with `--wait` .

Syntax:

```
wget --wait={seconds} {file_location}
```

Example:

```
wget --wait=2.5 -i url_list.txt
```

curl versus Wget

`curl` **advantages:**

- Can be used for downloading *and* uploading files from 20+ protocols.
- Easier to install across all operating systems.

`Wget` **advantages:**

- Has many built-in functionalities for handling multiple file downloads.
- Can handle various file formats for download (e.g. file directory, HTML page).

Let's practice!

DATA PROCESSING IN SHELL

Getting started with csvkit

DATA PROCESSING IN SHELL

Susan Sun
Data Person

What is csvkit?

`csvkit` :

- is a suite of command-line tools
- is developed in Python by Wireservice
- offers data processing and cleaning capabilities on CSV files
- has data capabilities that rival Python, R, and SQL
- documentation: https://csvkit.readthedocs.io/en/latest/

csvkit installation

Install `csvkit` using Python package manager `pip` :

```
pip install csvkit
```

Upgrade `csvkit` to the latest version:

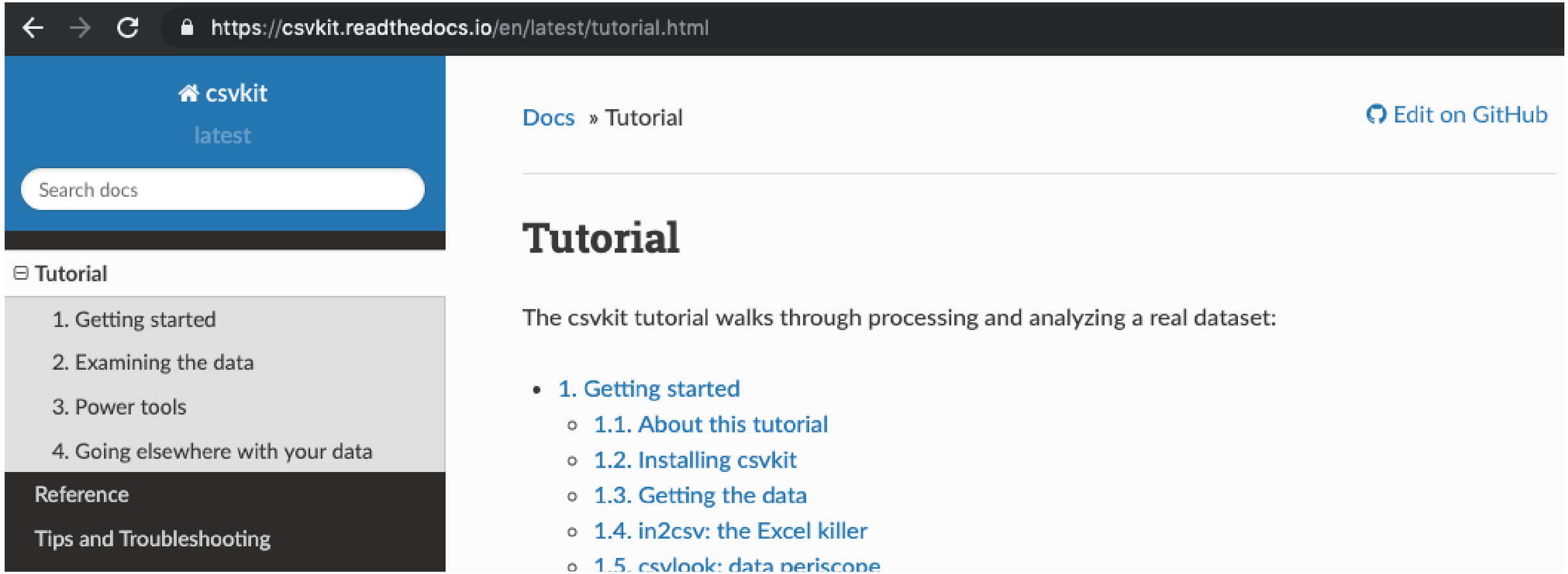
```
pip install --upgrade csvkit
```

Full instructions:

https://csvkit.readthedocs.io/en/latest/tutorial.html.

Browsing the csvkit manual

Web-based documentation: https://csvkit.readthedocs.io/en/latest/tutorial.html



in2csv: converting files to CSV

Web-based documentation:

https://csvkit.readthedocs.io/en/latest/scripts/in2csv.html

Command line-based documentation:

```
in2csv --help
in2csv -h
```

```
usage: in2csv [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
              [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-L LOCALE]
              [-S] [--blanks] [--date-format DATE_FORMAT]
              [--datetime-format DATETIME_FORMAT] [-H] [-K SKIP_LINES] [-v]
```

in2csv: converting files to CSV

Syntax:

```
in2csv SpotifyData.xlsx > SpotifyData.csv
```

Prints the first sheet in Excel to console and does not save

```
in2csv SpotifyData.xlsx
```

`>` redirects the output and saves it as a new file `SpotifyData.csv`

```
> SpotifyData.csv
```

in2csv: converting files to CSV

Use `--names` or `-n` option to print all sheet names in `SpotifyData.xlsx` .

```
in2csv -n SpotifyData.xlsx
```

```
Worksheet1_Popularity
Worksheet2_MusicAttributes
```

Use `--sheet` option followed by the sheet `"Worksheet1_Popularity"` to be converted.

```
in2csv SpotifyData.xlsx --sheet "Worksheet1_Popularity" > Spotify_Popularity.csv
```

in2csv: converting files to CSV

`in2csv` does not print logs to console.

```
in2csv SpotifyData.xlsx --sheet "Worksheet1_Popularity" > Spotify_Popularity.csv
```

Sanity check:

```
ls
```

```
SpotifyData.xlsx  Spotify_Popularity.csv  backup  bin
```

csvlook: data preview on the command line

`csvlook` : renders a CSV to the command line in a Markdown-compatible, fixed-width format

Documentation:

```
csvlook -h
```

```
usage: csvlook [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
               [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-L LOCALE]
               [-S] [--blanks] [--date-format DATE_FORMAT]
```

csvlook: data preview on the command line

Syntax:

```
csvlook Spotify_Popularity.csv
```

```
| track_id                | popularity |
| ----------------------- | ---------- |
| 118GQ70Sp6pMqn6w1oKuki |          7 |
| 6S7cr72a7a8RVAXzDCRj6m |          7 |
| 7h2qWpMJzIVtiP30E8VDW4 |          7 |
| 3KVQFxJ5CWOcbxdpPYdi4o |          7 |
| 0JjNrI1xmsTfhaiU1R6OVc |          7 |
| 3HjTcZt29JUHg5m60QhlMw |          7 |
```

csvstat: descriptive stats on CSV data files

`csvstat` : prints descriptive summary statistics on all columns in CSV (e.g. mean, median, unique values counts)

Documentation:

```
csvstat -h
```

```
usage: csvstat [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
               [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-S] [-H]
               [-K SKIP_LINES] [-v] [-l] [--zero] [-V] [--csv] [-n]
```

csvstat: descriptive stats on CSV data files

Syntax:

```
csvstat Spotify_Popularity.csv
```

```
  1. "track_id"

        Type of data:          Text
        Contains null values:  False
        Unique values:         24
        Longest value:         22 characters
        Most common values:    118GQ70Sp6pMqn6w1oKuki (1x)
                               6S7cr72a7a8RVAXzDCRj6m (1x)
```

Let's try some csvkit!

DATA PROCESSING IN SHELL

Filtering data using csvkit

DATA PROCESSING IN SHELL

Susan Sun
Data Person

What does it mean to filter data?

We can create a subset of the original data file by:

1. Filtering the data by column
2. Filtering the data by row

`csvcut` : filters data using **column** name or position

`csvgrep` : filters data by **row** value through exact match, pattern matching, or even regex

csvcut: filtering data by column

`csvcut` : filters and truncates CSV files by **column name** or **column position**

Documentation:

```
csvcut -h
```

```
usage: csvcut [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
              [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-S] [-H]
              [-K SKIP_LINES] [-v] [-l] [--zero] [-V] [-n] [-c COLUMNS]
```

csvcut: filtering data by column

Use `--names` or `-n` option to print all column names in `Spotify_MusicAttributes.csv` .

```
csvcut -n Spotify_MusicAttributes.csv
```

```
  1: track_id
  2: danceability
  3: duration_ms
```

csvcut: filtering data by column

```
1: track_id
2: danceability
3: duration_ms
```

Returns the first column in the data, by **position**:

```
csvcut -c 1 Spotify_MusicAttributes.csv
```

```
track_id
118GQ70Sp6pMqn6w1oKuki
6S7cr72a7a8RVAXzDCRj6m
```

csvcut: filtering data by column

```
1: track_id
2: danceability
3: duration_ms
```

Returns only the first column in the data, by **name**:

```
csvcut -c "track_id" Spotify_MusicAttributes.csv
```

```
track_id
118GQ70Sp6pMqn6w1oKuki
6S7cr72a7a8RVAXzDCRj6m
```

csvcut: filtering data by column

```
1: track_id
2: danceability
3: duration_ms
```

Returns the second and third column in the data, by **position**:

```
csvcut -c 2,3 Spotify_MusicAttributes.csv
```

```
danceability,duration_ms
0.787,124016.0
0.777,128016.0
0.795999999999999,132742.0
```

csvcut: filtering data by column

```
1: track_id
2: danceability
3: duration_ms
```

Returns the second and third column in the data, by **name**:

```
csvcut -c "danceability","duration_ms" Spotify_MusicAttributes.csv
```

```
danceability,duration_ms
0.787,124016.0
0.777,128016.0
0.795999999999999,132742.0
```

csvgrep: filtering data by row value

`csvgrep` :

- filters by **row** using exact match or regex fuzzy matching
- must be paired with one of these options:

`-m` : followed by the exact row value to filter

`-r` : followed with a regex pattern

`-f` : followed by the path to a file

Documentation:

```
csvgrep -h
```

csvgrep: filtering data by row value

Find in `Spotify_Popularity.csv` where `track_id` = `5RCPsfzmEpTXMCTNk7wEfQ`

```
csvgrep -c "track_id" -m 5RCPsfzmEpTXMCTNk7wEfQ Spotify_Popularity.csv
```

```
track_id,popularity
5RCPsfzmEpTXMCTNk7wEfQ,7.0
```

```
csvgrep -c 1 -m 5RCPsfzmEpTXMCTNk7wEfQ Spotify_Popularity.csv
```

```
track_id,popularity
5RCPsfzmEpTXMCTNk7wEfQ,7.0
```

Let's do data filtering with csvkit!

DATA PROCESSING IN SHELL

Stacking data and chaining commands with csvkit

DATA PROCESSING IN SHELL

Susan Sun
Data Person

csvstack: stacking multiple CSV files

`csvstack` : stacks up the rows from two or more CSV files

Documentation:

```
csvstack -h
```

```
usage: csvstack [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
                [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-S] [-H]
                [-n GROUP_NAME] [--filenames]
```

csvstack: stacking multiple CSV files

Stack two similar files `Spotify_Rank6.csv` and `Spotify_Rank7.csv` into one file.

Preview the data to check schema:

```
csvlook Spotify_Rank6.csv
```

```
| track_id                | popularity |
| ----------------------- | ---------- |
| 7JYCpIzpoidDOnnmxmHwtj |          6 |
| 0mmFibEg5NuULMwTVN2tRU |          6 |
```

csvstack: stacking multiple CSV files

```
csvlook Spotify_Rank7.csv
```

```
| track_id               | popularity |
| ---------------------- | ---------- |
| 118GQ70Sp6pMqn6w1oKuki |          7 |
| 6S7cr72a7a8RVAXzDCRj6m |          7 |
```

csvstack: stacking multiple CSV files

Syntax:

```
csvstack Spotify_Rank6.csv Spotify_Rank7.csv > Spotify_AllRanks.csv
```

```
csvlook Spotify_AllRanks.csv
```

```
| track_id               | popularity |
| ---------------------- | ---------- |
| 7JYCpIzpoidDOnnmxmHwtj |          6 |
| 0mmFibEg5NuULMwTVN2tRU |          6 |
| 118GQ70Sp6pMqn6w1oKuki |          7 |
| 6S7cr72a7a8RVAXzDCRj6m |          7 |
```

csvstack: stacking multiple CSV files

```
csvstack -g "Rank6","Rank7" \
Spotify_Rank6.csv Spotify_Rank7.csv > Spotify_AllRanks.csv
```

```
csvlook Spotify_AllRanks.csv
```

```
| group | track_id                | popularity |
| ----- | ----------------------- | ---------- |
| Rank6 | 7JYCpIzpoidDOnnmxmHwtj |          6 |
| Rank6 | 0mmFibEg5NuULMwTVN2tRU |          6 |
| Rank7 | 118GQ70Sp6pMqn6w1oKuki |          7 |
| Rank7 | 6S7cr72a7a8RVAXzDCRj6m |          7 |
```

csvstack: stacking multiple CSV files

```
csvstack -g "Rank6","Rank7" -n "source" \
Spotify_Rank6.csv Spotify_Rank7.csv > Spotify_AllRanks.csv
```

```
csvlook Spotify_AllRanks.csv
```

```
| source| track_id                | popularity |
| ----- | ----------------------- | ---------- |
| Rank6 | 7JYCpIzpoidDOnnmxmHwtj |          6 |
| Rank6 | 0mmFibEg5NuULMwTVN2tRU |          6 |
| Rank7 | 118GQ70Sp6pMqn6w1oKuki |          7 |
| Rank7 | 6S7cr72a7a8RVAXzDCRj6m |          7 |
```

chaining command-line commands

`;` links commands together and runs sequentially

```
csvlook SpotifyData_All.csv; csvstat SpotifyData_All.csv
```

`&&` links commands together, but only runs the 2nd command if the 1st succeeds

```
csvlook SpotifyData_All.csv && csvstat SpotifyData_All data.csv
```

chaining command-line commands

`>` re-directs the output from the 1st command to the location indicated as the 2nd

```
in2csv SpotifyData.xlsx > SpotifyData.csv
```

chaining command-line commands

`|` uses the output of the 1st command as input to the 2nd

Example:

Output of `csvcut` is not well formatted:

```
csvcut -c "track_id","danceability" Spotify_MusicAttributes.csv
```

```
track_id,danceability
118GQ70Sp6pMqn6w1oKuki,0.787
6S7cr72a7a8RVAXzDCRj6m,0.777
7h2qWpMJzIVtiP30E8VDW4,0.795
3KVQFxJ5CWOcbxdpPYdi4o,0.815
```

chaining command-line commands

Example (continued):

Re-format `csvcut` 's output by piping the output as input to `csvlook` :

```
csvcut -c "track_id","danceability"  Spotify_Popularity.csv | csvlook
```

```
| track_id                | danceability |
| ----------------------- | ------------ |
| 118GQ70Sp6pMqn6w1oKuki |        0.787 |
| 6S7cr72a7a8RVAXzDCRj6m |        0.777 |
| 7h2qWpMJzIVtiP30E8VDW4 |        0.796 |
| 3KVQFxJ5CWOcbxdpPYdi4o |        0.815 |
```

Let's put everything together!

DATA PROCESSING IN SHELL

Pulling data from databases

DATA PROCESSING IN SHELL

Susan Sun
Data Person

sql2csv: documentation

`sql2csv` :

- executes an SQL query on a large variety of SQL databases (e.g. MS SQL, MySQL, Oracle, PostgreSQL, Sqlite)
- outputs the result to a CSV file

Documentation:

```
sql2csv -h
```

```
usage: sql2csv [-h] [-v] [-l] [-V] [--db CONNECTION_STRING] [--query QUERY]
               [-e ENCODING] [-H]
               [FILE]
```

sql2csv: querying against the database

Sample syntax:

```
sql2csv --db "sqlite:///SpotifyDatabase.db" \
        --query "SELECT * FROM Spotify_Popularity" \
        > Spotify_Popularity.csv
```

1. Establishing database connection:

- `--db` is followed by the database connection string
- **SQLite**: starts with `sqlite:///` and ends with `.db`
- **Postgres** & **MySQL**: starts with `postgres:///` or `mysql:///` and with **no** `.db`

sql2csv: querying against the database

Sample syntax:

```
sql2csv --db "sqlite:///SpotifyDatabase.db" \
        --query "SELECT * FROM Spotify_Popularity" \
        > Spotify_Popularity.csv
```

2. Querying against the database:

- `--query` is followed by the SQL query string
- Use SQL syntax compatible with the database
- Write query in one line with no line breaks

sql2csv: querying against the database

Sample syntax:

```
sql2csv --db "sqlite:///SpotifyDatabase.db" \
        --query "SELECT * FROM Spotify_Popularity" \
        > Spotify_Popularity.csv
```

3. Saving the output:

- `>` : re-directs output to new local CSV file
- Otherwise, will only print query results to console

Let's practice!

DATA PROCESSING IN SHELL

Manipulating data using SQL syntax

DATA PROCESSING IN SHELL

Susan Sun
Data Person

csvsql: documentation

`csvsql` :

- applies SQL statements to one or more CSV files
- creates an in-memory SQL database that temporarily hosts the file being processed
- suitable for small to medium files only

Documentation:

```
csvsql -h
```

```
usage: csvsql [-h] [-d DELIMITER] [-t] [-q QUOTECHAR] [-u {0,1,2,3}] [-b]
              [-p ESCAPECHAR] [-z FIELD_SIZE_LIMIT] [-e ENCODING] [-L LOCALE]
```

csvsql: applying SQL to a local CSV file

Sample syntax:

```
ls
```

```
Spotify_MusicAttributes.csv
```

csvsql: applying SQL to a local CSV file

Sample syntax:

```
csvsql --query "SELECT * FROM Spotify_MusicAttributes LIMIT 1" \
    Spotify_MusicAttributes.csv
```

```
track_id,danceability,duration_ms,instrumentalness,loudness,tempo,time_signature
118GQ70Sp6pMqn6w1oKuki,0.787,124016.0,0.784,-10.457,119.988,4.0
```

csvsql: applying SQL to a local CSV file

Sample syntax:

```
csvsql --query "SELECT * FROM Spotify_MusicAttributes LIMIT 1" \
    data/Spotify_MusicAttributes.csv | csvlook
```

```
| track_id               | danceability | duration_ms | instrumentalness ...
| ---------------------- | ------------ | ----------- | ----------------- ...
| 118GQ70Sp6pMqn6w1oKuki |        0.787 |     124,016 |             0.784 ...
```

csvsql: applying SQL to a local CSV file

Sample syntax:

```
csvsql --query "SELECT * FROM Spotify_MusicAttributes LIMIT 1" \
    data/Spotify_MusicAttributes.csv > OneSongFile.csv
```

```
ls
```

```
OneSongFile.csv
```

csvsql: joining CSVs using SQL syntax

Sample syntax:

```
csvsql --query "SELECT * FROM file_a INNER JOIN file_b..." file_a.csv file_b.csv
```

Note:

- SQL Query must be written in one line, no breaks
- Indicate CSV files in order of appearance in SQL

Let's practice!

DATA PROCESSING IN SHELL

Pushing data back to database

DATA PROCESSING IN SHELL

Susan Sun
Data Person

csvsql: documentation

`csvsql` :

- execute SQL statements directly on a database
- supports both creating tables and inserting data.

More option arguments:

- `--insert`
- `--db`
- `--no-inference` & `--no-constraints`

csvsql: pushing data back to database

Sample syntax:

```
csvsql --db "sqlite:///SpotifyDatabase.db" \
       --insert Spotify_MusicAttributes.csv
```

Note:

1. Line break is used to keep code clean and readable
2. Use three forward slashes to initiate database name
3. End with file extension `.db` for SQLITE database

csvsql: pushing data back to database

Sample syntax:

```
csvsql --db "sqlite:///SpotifyDatabase.db" \
       --insert Spotify_MusicAttributes.csv
```

Documentation:

```
csvsql -h
```

```
--insert  In addition to creating the table, also insert the
          data into the table. Only valid when --db is specified.
```

csvsql: pushing data back to database

Sample syntax:

```
csvsql --no-inference --no-constraints \
       --db "sqlite:///SpotifyDatabase.db" \
       --insert Spotify_MusicAttributes.csv
```

Documentation:

```
csvsql -h
```

```
--no-inference   Disable type inference when parsing the input.
--no-constraints Generate a schema without length limits or null checks.
```

Let's practice!

DATA PROCESSING IN SHELL

Python on the command line

DATA PROCESSING IN SHELL

Susan Sun
Data Person

Python basics

Python

- comes pre-installed on MacOS, Linux
- needs to be user-install for Windows **instructions here**
- can be used with GUI interfaces (e.g Jupyter Notebook, Spyder, PyCharm, etc.)
- can also be accessed directly via the command line interface

Using Python documentation

Documentation:

```
man python
```

```
...
-V ,  --version
    Prints the Python version number of the executable and exits.
```

```
python --version
```

```
Python 3.5.2
```

Using Python documentation

Example 1: using native Python

```
which python
```

```
/usr/bin/python
```

Example 2: using Anaconda Python

```
which python
```

```
/anaconda3/bin/python
```

The Python interactive session

To activate a Python interactive session in the terminal:

```
python
```

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linuxType "help", "copyright", "credits" or
"license" for more information.
>>>
```

The Python interactive session

Inside the interactive session, only use Python syntax:

```
>>> print('hello world')
hello world
```

To exit the Python session and return to terminal:

```
>>> exit()
$
```

Python interactive session alternative

Python interactive session:

- easy to activate, intuitive
- not good for code reproducibility

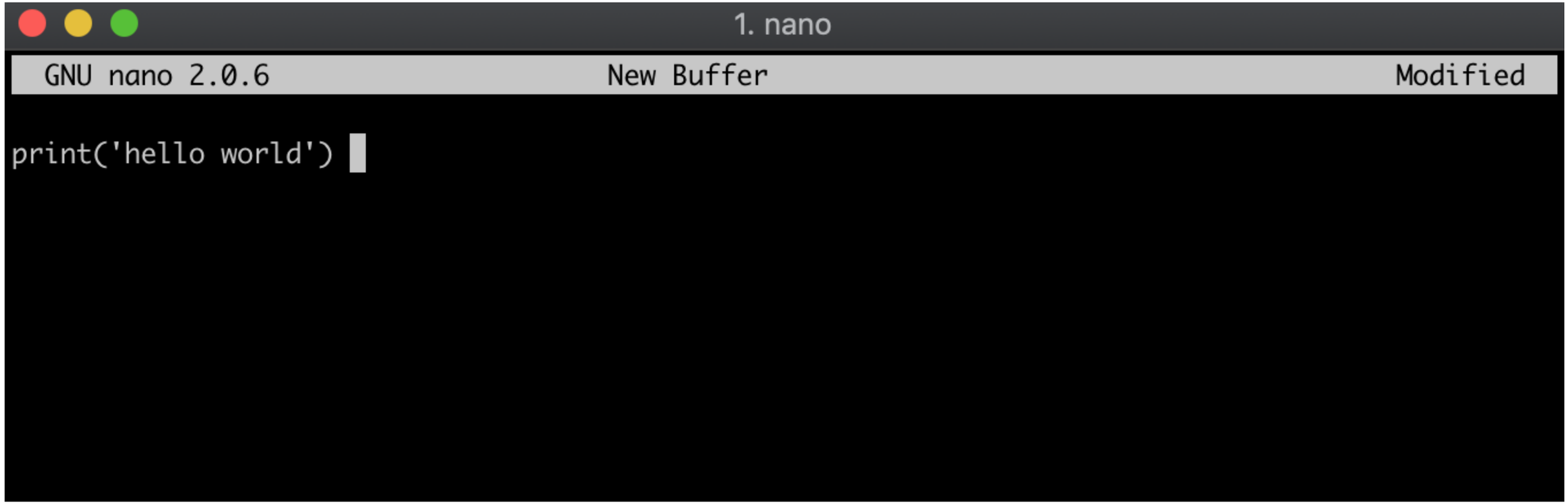
Alternative:

- save Python commands in a Python `.py` script
- execute script by calling `python` + script

Python script execution on the command line

Method 1

- Create a `.py` file using a text editor on the command line (e.g. nano, Vim, Emacs)



Python script execution on the command line

Method 2

- Create a `.py` file by `echo` -ing the Python syntax into the `hello_world.py` file, instantiating the Python file in the same step.

```
echo “print(‘hello world’)” > hello_world.py
```

Sanity check file content:

```
cat hello_world.py
```

```
print('hello world')
```

Python script execution on the command line

Make sure in the same directory as the `.py` file:

```
ls
```

```
hello_world.py
```

Execute `.py` file by preceding filename with `python` :

```
python hello_world.py
```

```
hello world
```

Let's practice!

DATA PROCESSING IN SHELL

Python package installation with pip

DATA PROCESSING IN SHELL

Susan Sun
Data Person

Python standard library

Python standard library has a collection of:

- built-in functions (e.g. print())
- built-in packages (e.g. math, os)

Data science packages like **scikit-learn** and **statsmodel**:

- are **NOT** part of the Python standard library
- can be installed through `pip`, the standard package manager for Python, **via the command line**

Using pip documentation

Documentation:

```
pip -h
```

```
Usage:
  pip <command> [options]

Commands:
  install       Install packages.
  uninstall     Uninstall packages.
  freeze        Output installed packages in requirements format.
  list          List installed packages.
```

Using pip documentation

Documentation:

```
pip --version
```

```
pip 19.1.1 from /usr/local/lib/python3.5/dist-packages/pip (python 3.5)
```

```
python --version
```

```
Python 3.5.2
```

Upgrading pip

If `pip` is giving an upgrade warning:

```
WARNING: You are using pip version 19.1.1, however version 19.2.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Upgrade `pip` using itself:

```
pip install --upgrade pip
```

```
Collecting pip
     |################################| 1.4MB 10.7MB/s
Successfully installed pip-19.2.1
```

pip list

`pip list` : displays the Python packages in your current Python environment

```
pip list
```

```
Package         Version
- - - - - - - - - - - - -
agate           1.6.1
agate-dbf       0.2.1
agate-excel     0.2.3
agate-sql       0.5.4
Babel           2.7.0
```

pip install one package

`pip install` installs the package specified **and** any other dependencies

```
pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading https://files.pythonhosted.org/packages/1f/af/e3c3cd6f6109383005913862 4dbc
     |#################################| 6.6MB 32.5MB/s
Collecting scipy>=0.17.0 (from scikit-learn)
  Downloading https://files.pythonhosted.org/packages/14/49/8f13fa215e10a7ab0731cc95b0e9
     |#################################| 25.1MB 35.5MB/s
...
```

pip install a specific version

By default, `pip install` will always install the latest version of the library.

```
pip install scikit-learn
```

```
Successfully built sklearn
Installing collected packages: joblib, scipy, scikit-learn, sklearn
Successfully installed joblib-0.13.2 scikit-learn-0.21.3 scipy-1.3.0 sklearn-0.0
```

pip install a specific version

To install a specific (or older) version of the library:

```
pip install scikit-learn==0.19.2
```

```
Collecting scikit-learn==0.19.2
  Downloading https://files.pythonhosted.org/packages/b6/e2/a1e254a4a4598588d4fe88b45ab8
     |################################| 4.9MB 15.6MB/s
Installing collected packages: scikit-learn
Successfully installed scikit-learn-0.19.2
```

Upgrading packages using pip

Upgrade the Scikit-Learn package using pip:

```
pip install --upgrade scikit-learn
```

```
Collecting scikit-learn
  Downloading https://files.pythonhosted.org/packages/1f/af/e3c3cd6f6109383005913862 4dbc
     |################################| 6.6MB 41.5MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.11.0 in /usr/local/lib/python3
Collecting scipy>=0.17.0 (from scikit-learn)
Installing collected packages: scipy, joblib, scikit-learn
Successfully installed joblib-0.13.2 scikit-learn-0.21.3 scipy-1.3.0
```

pip install multiple packages

To `pip install` multiple packages, separate the packages with spaces:

```
pip install scikit-learn statsmodels
```

Upgrade multiple packages:

```
pip install --upgrade scikit-learn statsmodels
```

pip install with requirements.txt

`requirements.txt` file contains a list of packages to be installed:

```
cat requirements.txt
```

```
scikit-learn
statsmodel
```

Most Python developers include `requirements.txt` files in their Python Github repos.

pip install with requirements.txt

`-r` allows `pip install` to install packages from a pre-written file:

```
-r, --requirement <file>
Install from the given requirements file. This option can be used multiple times.
```

In our example:

```
pip install -r requirements.txt
```

is the same as

```
pip install scikit-learn statsmodel
```

Let's practice!

DATA PROCESSING IN SHELL

Data job automation with cron

DATA PROCESSING IN SHELL

Susan Sun
Data Person

What is a scheduler?

- Scheduler runs jobs on a pre-determined schedule
- Commercial schedulers: Airflow, Luigi, Rundeck, etc.
- cron scheduler is
  - simple
  - free
  - customizable
  - purely command-line
  - native to MacOS and Linux

What is cron?

Cron:

- is a time-based job-scheduler
- comes pre-installed in MacOS, Unix
- can be installed in Windows via Cygwin or replaced with Windows Task Scheduler
- is used to automate jobs like system maintenance, bash scripts, Python jobs, etc.

What is crontab?

Crontab is a central file to keep track of cron jobs.

```
crontab -l
```

```
no crontab for <username>
```

Documentation:

```
man crontab
```

Add a job to crontab

Method 1: modify crontab using a text editor (e.g. nano, Vim, Emacs)

Method 2: echo the scheduler command into crontab

```
echo "* * * * * python create_model.py" | crontab
```

Check if the job is properly scheduled:

```
crontab -l
```

```
* * * * * python create_model.py
```

Learning to time a cron job

The most frequent schedule for cron jobs is **one minute**.

Breaking down the time component for a cron job:

```
.---------------- minute (0 - 59)
|  .------------- hour (0 - 23)
|  |  .---------- day of month (1 - 31)
|  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
|  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed ...
|  |  |  |  |
*  *  *  *  * command-to-be-executed
```

Learning to time a cron job

```
*  *  *  *  * python create_model.py
```

Interpretation:

- Run every minute of every hour of every day of every month and of every day of the week.
- In short, run every minute

Further resources:

- Use **https://crontab.guru/** to see more ways to schedule a cron job

Let's practice!

DATA PROCESSING IN SHELL

Course recap

DATA PROCESSING IN SHELL

Susan Sun
Data Person

Data downloading on the command line

- How to download data files via `curl` and `wget`
- Documentations, manuals (e.g. `man curl` , `wget --help`)
- Multiple file downloads (e.g. `wget --limit-rate=200k -i url_list.txt`)

Data processing on the command line

- Introduction to command line data toolkit: `csvkit`
- Convert files to csv using `in2csv`
- Print preview using `csvlook` , `csvstat`
- Filter data using `csvcut` , `csvgrep`
- Append multiple data files using `csvstack`

Database manipulation on the command line

- Database manipulation using `sql2csv`, `csvsql`
- Advanced SQL-like ETL commands using `csvkit`

Building data pipelines on the command line

- Execute Python on the command line
- Python package management using `pip`
- Automate Python model and build pipelines with `cron`

Thank you! So long!

DATA PROCESSING IN SHELL