Lecture 1: Data Management in a Big Data Environment

Contents

1.1. Learning objectives

- Reflect on various data sources and your approach to them.
- Gain an understanding of how data is presented on the web and different methods for analyzing it.
- Understand the application of the 3Vs of big data.
- Recognize how your questions impact the selection of data analysis tools.
- Understand how `storage`, `memory`, and `processing` apply when it's big data.
- Learn some best practices that you can incorporate when dealing with `big data locally`.

Review the table and consider how you obtained data for previous courses.

WHERE to get data?	**HOW to get data?**
Collect from people	Just ask them
From databases	SQL queries
From computer/ servers	By just logging in or downloading
From websites	Using Web scraping or API

Mostly, we used data available as a file on the internet or provided by the course instructor. In DSCI 513, you worked with data available within a database. However, you may not be familiar with ***how to obtain data from the web***. Data may not always be available as a file, and you may need to extract it from a website you wish to analyze.

In the following session, you will learn how data is presented on the web and various methods of retrieving it.

1.2. How data appears on a website

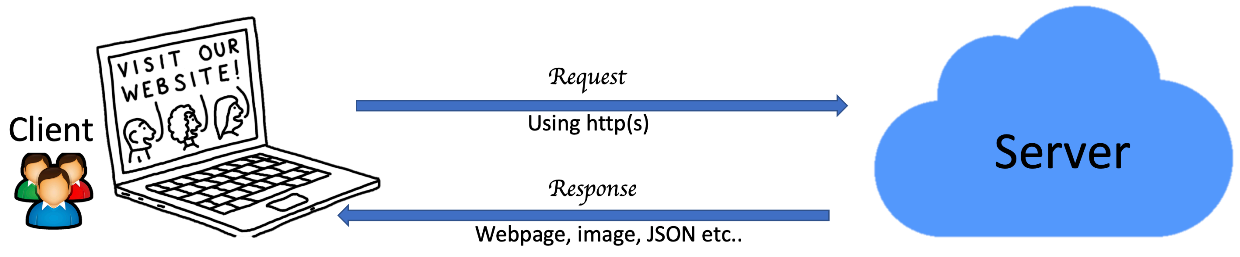
Data on a website is accessed through an `API` (Application Program Interface). The ***API acts as a messenger***, providing an interface for communication with software, websites, and servers while hiding underlying functions. When you visit a website or click on various options within a webpage, a series of API calls are made to the server behind the scenes, and the server responds with the requested information (as we will see in the demo). Communication and data transfer occur through the web using a protocol called `HTTP(s)` (HTTP requests). The convention for building these HTTP services is known as `REST(REpresentational State Transfer)ful API`. The table below illustrates how CRUD operations (similar to database operations) are translated to HTTP verbs.

CRUD	HTTP Verb
Create	Post
Read	Get
Update	Put
Delete	Delete

You have already learned that websites use the `client-server` model *(we will revisit this from the ground up when I introduce cloud computing)*, where the website is hosted on a server and the client communicates with the server via API calls to retrieve information. You don't have access to the provider's server, but there are certain `endpoints` that providers have permitted you to access as a consumer.

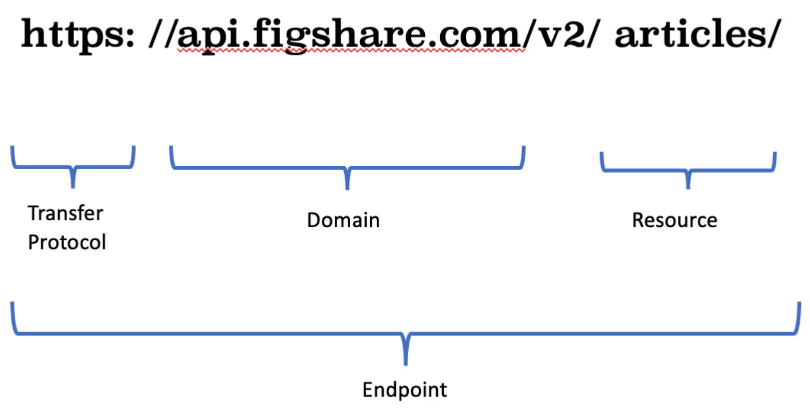
Attention

There are certain WINDOWS (`endpoints`) that you can access in the server, and you are not given DOOR (`ssh`) access for entering the server.



In short, we put a request to a certain `endpoint` within the provider's server using HTTP methods. And in response, it gives us back something (HTML, JSON, image, video, audio clip, etc…)

An `endpoint` can include a `URL` *(Uniform Resource Locator)* of a server, and here is the general structure of a URL.



Let's inspect a webpage to solidify what we discussed.

Question ?

We saw how data appear on the website, but how can we get this data to our computer if we want to analyze it?

- Using rest API

Go lower level to get direct access to the data. Many website providers give this option; here are a few examples;

- Twitter API
- Facebook API
- Pinterest API
- Figshare API

Here are some packages that you can use to interact with these API's

- requests (Python)
- httr (R)

- Web Scraping

Here are some packages that you can use to interact with these API's

- BeautifulSoup (Python)
- scrapy (python)
- rvest (R)

See also

Unfortunately, we don't have time to go into the details of scraping. However, the following articles can be helpful if you require yourself to gather data for analysis in the future. I am also trying to find some TA time for setting up a tutorial session on web scraping.

This article useful and straightforward in showing how to do web scraping in python.

These 2 articles(1,2) takes you through the difference between using API and scraping.

Another 2 articles that help you gain more insight into web scraping are:

- Webscraping part 1
- Webscraping part 2

For Milestone 1, you'll be using the Figshare API to pull data and analyze it in upcoming milestones. The section below on using the REST API can be useful in completing milestone1 task.

1.2.1. Using rest API (lab lecture)

```
import re
import os
import glob
import zipfile
import requests
from urllib.request import urlretrieve
import json
import pandas as pd
```

```
%cd /Users/gittugeorge/Desktop/525_2025/figshareexp
## Change it to the location that you want to download your files to.
```

```
/Users/gittugeorge/Desktop/525_2025/figshareexp
```

```
# Necessary metadata
article_id = 14226968  # this is the unique identifier of the article on figshare
url = f"https://api.figshare.com/v2/articles/{article_id}"
headers = {"Content-Type": "application/json"}
output_directory = "figshareairline/"
```

Here we are sending a GET request to list the available files

```
response = requests.request("GET", url, headers=headers)
data = json.loads(response.text)  # this contains all the articles data, feel free to check it out
files = data["files"]             # this is just the data about the files, which is what we want
files
```

```
[{'id': 26844650,
  'name': 'allyears.csv.zip',
  'size': 2405908113,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26844650',
  'supplied_md5': '9e046ac05ecd2c32a256a47dd1098b81',
  'computed_md5': '9e046ac05ecd2c32a256a47dd1098b81',
  'mimetype': 'application/zip'},
 {'id': 26863682,
  'name': 'individual_years.zip',
  'size': 1896206676,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/26863682',
  'supplied_md5': '921da748974b07b2a70bbfcc04535a77',
  'computed_md5': '921da748974b07b2a70bbfcc04535a77',
  'mimetype': 'application/zip'},
 {'id': 27515426,
  'name': 'combined_model_data.csv.zip',
  'size': 821308997,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/27515426',
  'supplied_md5': '7638434c44a7d29cbb29fe200b4fd65d',
  'computed_md5': '7638434c44a7d29cbb29fe200b4fd65d',
  'mimetype': 'application/zip'},
 {'id': 27520682,
  'name': 'combined_model_data_parti.parquet.zip',
  'size': 519743915,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/27520682',
  'supplied_md5': '02f4e3df8d16580a02291de225072689',
  'computed_md5': '02f4e3df8d16580a02291de225072689',
  'mimetype': 'application/zip'},
 {'id': 27520808,
  'name': 'combined_model_data.parquet',
  'size': 565872005,
  'is_link_only': False,
  'download_url': 'https://ndownloader.figshare.com/files/27520808',
  'supplied_md5': 'ae63699ab21ffa8006559c6afbcd2271',
  'computed_md5': 'ae63699ab21ffa8006559c6afbcd2271',
  'mimetype': 'application/octet-stream'}]
```

See also

- Here 'response' variable in the above case is the requests 'response Object'. There are various methods you can do on this, but we are using `.text` to return the response's content. Check here for entire list of methods.
- There are many more details to using the request library and rest API. Here is a good 20 min video showing you working with the requests library.

We are going to get the file named "individual_years.zip"

```
%%time
files_to_dl = ["individual_years.zip"]  # feel free to add other files here
for file in files:
    if file["name"] in files_to_dl:
        os.makedirs(output_directory, exist_ok=True)
        urlretrieve(file["download_url"], output_directory + file["name"])
```

```
CPU times: user 4.14 s, sys: 15.3 s, total: 19.4 s
Wall time: 2min 6s
```

```
%%time
with zipfile.ZipFile(os.path.join(output_directory, "individual_years.zip"), 'r') as f:
    f.extractall(output_directory)
```

```
CPU times: user 13.4 s, sys: 1.73 s, total: 15.1 s
Wall time: 15.1 s
```

```
%ls -ltr figshareairline/individual_years/
```

```
total 17616704
-rw-r--r--@ 1 gittugeorge  staff  571716440 Mar 20 12:19 1996.csv
-rw-r--r--@ 1 gittugeorge  staff  578067302 Mar 20 12:19 1997.csv
-rw-r--r--@ 1 gittugeorge  staff  568173026 Mar 20 12:19 1995.csv
-rw-r--r--@ 1 gittugeorge  staff  737443161 Mar 20 12:19 2008.csv
-rw-r--r--@ 1 gittugeorge  staff  637867078 Mar 20 12:19 2001.csv
-rw-r--r--@ 1 gittugeorge  staff  609846703 Mar 20 12:19 2000.csv
-rw-r--r--@ 1 gittugeorge  staff  565592293 Mar 20 12:19 2002.csv
-rw-r--r--@ 1 gittugeorge  staff  670191374 Mar 20 12:19 2003.csv
-rw-r--r--@ 1 gittugeorge  staff  754534453 Mar 20 12:19 2007.csv
-rw-r--r--@ 1 gittugeorge  staff  721202729 Mar 20 12:19 2006.csv
-rw-r--r--@ 1 gittugeorge  staff  718355182 Mar 20 12:19 2004.csv
-rw-r--r--@ 1 gittugeorge  staff  719753438 Mar 20 12:19 2005.csv
-rw-r--r--@ 1 gittugeorge  staff  591206878 Mar 20 12:19 1999.csv
-rw-r--r--@ 1 gittugeorge  staff  575771167 Mar 20 12:19 1998.csv
```

Let's combine all these files using regular python.

```
%%time
### just listing to get an idea how individual file looks like
use_cols = ["ArrDelay", "DepDelay", "Distance", "TailNum","UniqueCarrier","Origin","Dest"]
df = pd.read_csv("figshareairline/individual_years/1995.csv", usecols=use_cols,dtype={'TailNum': 'str'})
df
```

```
CPU times: user 1.91 s, sys: 180 ms, total: 2.09 s
Wall time: 2.09 s
```

	UniqueCarrier	TailNum	ArrDelay	DepDelay	Origin	Dest	Distance
0	UA	N7298U	15.0	12.0	ORD	PHL	678.0
1	UA	N7449U	1.0	3.0	ORD	PHL	678.0
2	UA	N7453U	-5.0	4.0	ORD	PHL	678.0
3	UA	N7288U	-9.0	0.0	ORD	PHL	678.0
4	UA	N7275U	-6.0	0.0	ORD	PHL	678.0
...	...	...	...	...	...	...	...
5327430	TW	N9409F	0.0	2.0	STL	DEN	770.0
5327431	TW	N912TW	23.0	1.0	STL	DEN	770.0
5327432	TW	N902TW	9.0	0.0	STL	DEN	770.0
5327433	TW	N954U	16.0	21.0	STL	DEN	770.0
5327434	TW	N920TW	-4.0	0.0	STL	DEN	770.0

5327435 rows × 7 columns

```
%%time
## here we are using a normal python way for merging the data
import pandas as pd
use_cols = ["ArrDelay", "DepDelay", "Distance", "TailNum","UniqueCarrier","Origin","Dest"]
files = glob.glob('figshareairline/individual_years/*.csv')
df = pd.concat((pd.read_csv(file, index_col=0, usecols=use_cols)
                .assign(year=re.findall("[0-9]+", file)[0])
                for file in files)
              )
df.to_csv("figshareairline/combined_data.csv")
```

```
CPU times: user 1min 58s, sys: 10.3 s, total: 2min 8s
Wall time: 2min 8s
```

```
df
```

	TailNum	ArrDelay	DepDelay	Origin	Dest	Distance	year
UniqueCarrier							
DL	N673DL	66.0	69.0	ATL	PHX	1587.0	1996
DL	N686DA	3.0	1.0	ATL	PHX	1587.0	1996
DL	N685DA	52.0	26.0	ATL	PHX	1587.0	1996
DL	N522DA	84.0	100.0	ATL	PHX	1587.0	1996
DL	N2824W	56.0	84.0	ATL	PHX	1587.0	1996
...	...	...	...	...	...	...	...
UA	N344UA	-4.0	-4.0	SMF	DEN	910.0	1998
UA	N347UA	23.0	8.0	SMF	DEN	910.0	1998
UA	N311UA	10.0	9.0	SMF	DEN	910.0	1998
UA	N315UA	13.0	24.0	SMF	DEN	910.0	1998
UA	N330UA	16.0	-1.0	SMF	DEN	910.0	1998

86289323 rows × 7 columns

```
%%sh
du -sh figshareairline/combined_data.csv
```

```
3.1G    figshareairline/combined_data.csv
```

```
print(df.shape)
```

```
(86289323, 7)
```

```
df.head()
```

	TailNum	ArrDelay	DepDelay	Origin	Dest	Distance	year
UniqueCarrier							
DL	N673DL	66.0	69.0	ATL	PHX	1587.0	1996
DL	N686DA	3.0	1.0	ATL	PHX	1587.0	1996
DL	N685DA	52.0	26.0	ATL	PHX	1587.0	1996
DL	N522DA	84.0	100.0	ATL	PHX	1587.0	1996
DL	N2824W	56.0	84.0	ATL	PHX	1587.0	1996

```
%reset -f
## This is to clear the memory, and you get a fresh start. When you run this, you will lose all the varia
```

1.3. Let's hear a story on `it Depends`.

From the table that we saw at the beginning of the lecture, you can probably think about and reflect on the ways and techniques you are familiar with for obtaining and processing data in various scenarios. The data can be of any kind, such as images, videos, doctor's notes, text files, etc., and any type, such as CSV, JSON, XML, etc. In your previous courses, you mainly dealt with CSV files and processed them mostly using `pandas` or `dplyr`.

Thoughts/Discussion ?

Will these packages work well when the data is big?

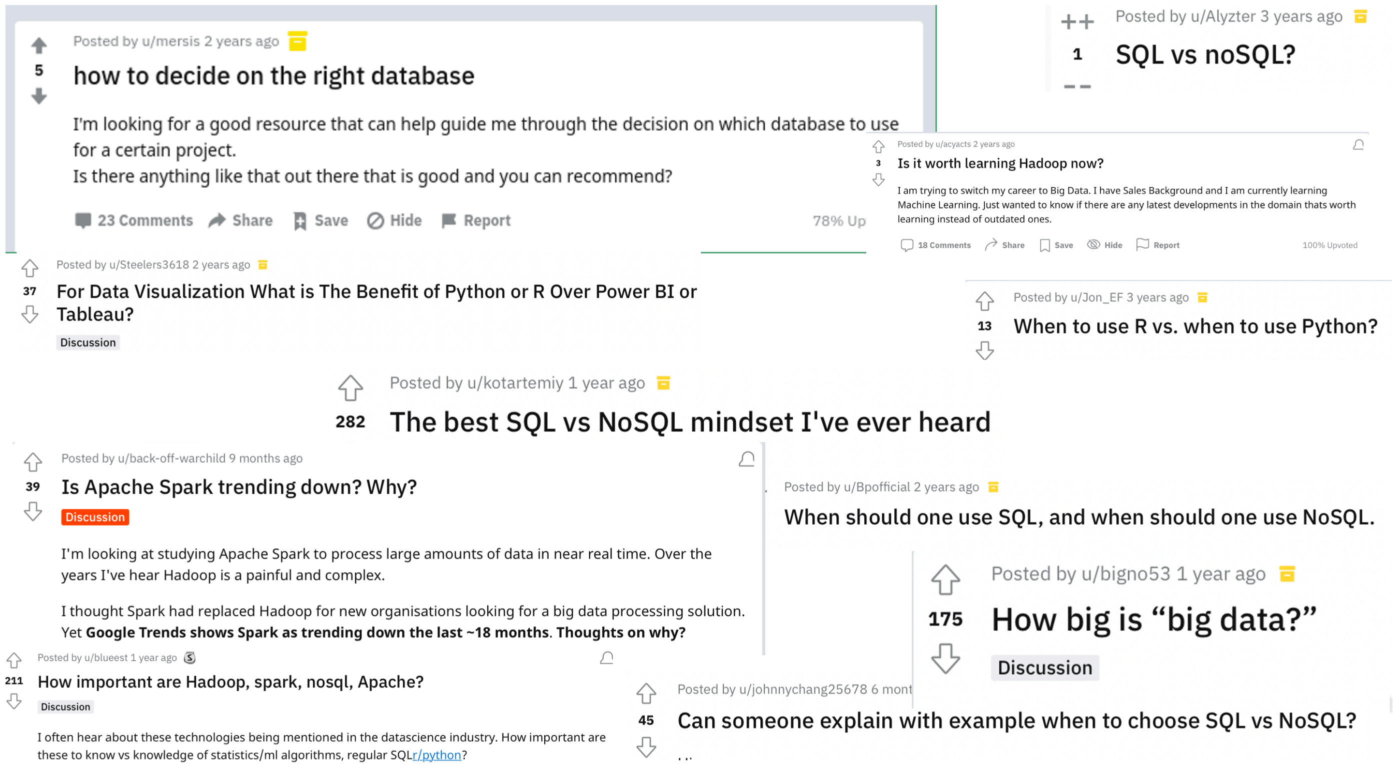
No, they won't. Because these get loaded into memory. More details on it in this lecture.

Thoughts/Discussion ?

Will database help you to work with big data?

You were also introduced to `databases` as a way to manage large data sets. However, can you always depend on a database? For example, if you want to process a 50 GB file that you downloaded from the internet, do you want to set up a database, load the data, and process it from there? Is that the best approach? Or are there other techniques (which we will learn about in this course, ranging from infrastructure-level to package-level solutions) that you would prefer to use in these scenarios? I'm not saying you should stop using databases; they can be very effective in many situations. However, whether you want to use a database and what kind of database you want to use ***depends on the situation***.

You have probably come across many questions like this on Reddit or StackOverflow.



5 Posted by u/mersis 2 years ago
how to decide on the right database
I'm looking for a good resource that can help guide me through the decision on which database to use for a certain project.
Is there anything like that out there that is good and you can recommend?
23 Comments Share Save Hide Report 78% Up

1 Posted by u/Alyzter 3 years ago
SQL vs noSQL?

3 Posted by u/acyacts 2 years ago
Is it worth learning Hadoop now?
I am trying to switch my career to Big Data. I have Sales Background and I am currently learning Machine Learning. Just wanted to know if there are any latest developments in the domain thats worth learning instead of outdated ones.
18 Comments Share Save Hide Report 100% Upvoted

37 Posted by u/Steelers3618 2 years ago
For Data Visualization What is The Benefit of Python or R Over Power BI or Tableau?
Discussion

13 Posted by u/Jon_EF 3 years ago
When to use R vs. when to use Python?

282 Posted by u/kotartemiy 1 year ago
The best SQL vs NoSQL mindset I've ever heard

39 Posted by u/back-off-warchild 9 months ago
Is Apache Spark trending down? Why?
Discussion
I'm looking at studying Apache Spark to process large amounts of data in near real time. Over the years I've hear Hadoop is a painful and complex.
I thought Spark had replaced Hadoop for new organisations looking for a big data processing solution. Yet **Google Trends shows Spark as trending down the last ~18 months. Thoughts on why?**

Posted by u/Bpofficial 2 years ago
When should one use SQL, and when should one use NoSQL.

175 Posted by u/bigno53 1 year ago
How big is "big data?"
Discussion

211 Posted by u/blueest 1 year ago
How important are Hadoop, spark, nosql, Apache?
Discussion
I often hear about these technologies being mentioned in the datascience industry. How important are these to know vs knowledge of statistics/ml algorithms, regular SQLr/python?

45 Posted by u/johnnychang25678 6 mont
Can someone explain with example when to choose SQL vs NoSQL?
. ..

You probably want to check out these questions and answers,1,2,3,4,5,6,7 to see what they are talking about. Reading these questions and answers is interesting, but before considering all these technologies, you need to first focus on the problem at hand. If you have a clear understanding of the business problem and list down all the features, you can then list the pros and cons of using one technology over another.

So which technology should I use? It all boils down to `it depends` on your business problem. Let me take you through some of the `Depends`.

- `Depends` on the output you need
  - Interactive or static graphs?
  - Integrated reports, tables, websites?
  - For yourself, for your company, for the public?
- `Depends` on the data you use
  - One large file, many small files?
  - Use the whole file at once or in small chunks?
  - Is data static (cold) or dynamic (hot)
- `Depends` on the question(s) you are trying to ask
  - Is it high value?
  - Are they complex (OLAP) or simple (OLTP)
  - Do you need results quickly?
  - For yourself, for your company, for the public?

`It Depends` matters a lot, especially in "Big Data," as

- The costs and benefits are much higher
- The impact on a business's bottom line can be significant

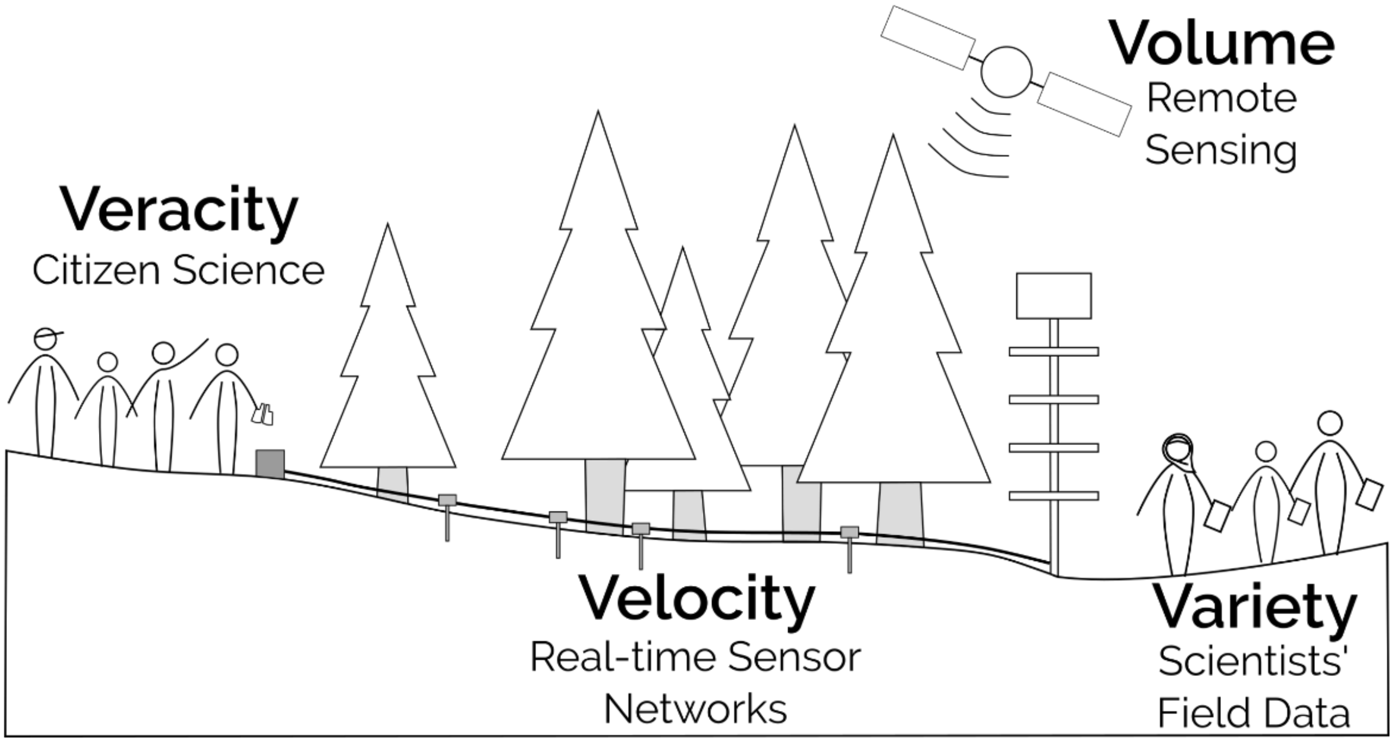
Thoughts/Discussion ?

How big the data should be for it to be big data ?

It depends. Maybe it is big when your computer can't process it.

1.4. 3 Vs of Big Data

Whenever you come across any big-data article, you will hear about the V's. The definition of 3V's was introduced in 2001 by Gartner Inc. analyst Doug Laney, and it has evolved significantly with many other V's. Here, I will discuss 3 of the most common V's in today's industry. The 3V's, broadly speaking, are ***Velocity, Variety, and Volume. These concepts were introduced to help understand the challenges of Big Data in computing and analytics. The concepts are broadly applicable across all disciplines. For example, check what my colleague wrote about big data in Ecology. Different datasets or problems are affected differently by each of the different axes, so understanding the different dimensions of big data is critical to finding the analytic solutions we intend to apply.



Source: Farley, Dawson, Goring & Williams, Ecography, 2018 Open Access

Other people have spoken about five Vs (Value), seven Vs (adding Variability, Visualization and Value), and even more. If you want to know more V's (42 of them) and add your vocabulary, you can check out this article. These additional V's can be informative, but, by and large, the 3V's provide the most insight into data challenges.

Let's checkout 3 V's in detail.

1.4.1. Volume

When we think about "Big Data, " this is often the most familiar dimension. We think of Big Data in terms of gigabytes, terabytes, or petabytes. The volume presents a significant ***challenge for data storage***, although modern technology has ***reduced*** this challenge to some degree. It also produces a ***challenge*** for recall (simply finding information) and ***for data processing*** (transforming and analyzing data). Here are the pointers to look out for:

- How big is the data?
- Total File Size
- Number of Data Points

1.4.1.1. Volume Solutions

- ***Cloud storage solutions***

But in most cases, it's not just about storing your data; it's more about how you will process the data. Then there will be limitations by read-write access or/and memory capacity.

- ***Partitioning/Sharding***

We must look at this from different angles, partitioning/sharding from a database perspective, which we will touch upon later in this course. Another option is to go for distributed file system where it stores data in small blocks so that it's easy to process when needed.

- ***Parallel processing***

Small chunks of files can be processed simultaneously by using different servers/cores and aggregate results at the end. This is mainly made possible in the industry by high-performance/cluster computing or other map-reduce-based frameworks like Hadoop/spark.

1.4.2. Velocity

Velocity is a second factor. We may be interested in only the most recent data, minutes or seconds, requiring low volumes of data, but at a high velocity. For example, quant traders might be using high-frequency stock data to maximize trading profits. This involves analysis with extremely fast turnover. Here are the pointers to look out for:

- How fast is the data arriving?
  - Annual data from Stats Can?
  - Real-time data from stock tickers or Twitter?
- How fast do we need our answers?
  - Monthly or annual strategy planning?
  - Real-time commodities purchasing?
- How fast is the data changing?
  - Changing interpretations?

1.4.2.1. Velocity Solutions

- Agile development.
- BASE databases (Basically available).
- Modular analytics & fault tolerance.
- Identify "Key Performance Indicators".
- Develop real-time reporting.
- Split the data into hot (Redis , RAM) and cold data (RDBMS, Disk Storage).

1.4.3. Variety

When we bring in ***multiple data sources*** or build data lakes, how well do data fields align? Are temporal scales aligned? Are addresses, zip codes, census tracts, or electoral districts aligned with other spatial data? Are financial values in standard units? If not, how do we transform the values to account for fluctuating exchange rates? Here are the pointers to look out for

- How different is the data source?
- Are data coming from multiple sources?
    - Do fields in sources align well?
    - Do files have similar formatting?
- How different are the data structures?
    - Can the data work in a relational model?
    - Do we need multiple data models?

1.4.3.1. Variety Solutions

- Normalize using multiple data sources.
- Clear interface layers (structured based on velocity and volume).
- Different management systems (RDBMS & Graph DB).

1.5. The 3 V’s and Analytic Workflows

These challenges come to the forefront when we’re working with data. The goto standard is to open up an Excel or read in a comma-separated file to look at the data, get a sense of what is happening, or summarise the key elements.

- When that file is <10MB in size, that’s often not a big problem, but as files get bigger and bigger, even calculating a simple mean is an issue (**Volume**).
- When the data contains text, images, values in different currencies, summary becomes problematic (**Variety**)
- When the 10MB you just summarized are out-of-date, as soon as you’re done outlining them, how do you present these results to your colleagues? (**Velocity**)

1.5.1. Problem Based Approaches

Many of these challenges have straightforward(ish) solutions, but how we apply those solutions and our choices are often specific to the problem we are trying to answer. For example, a common ***solution*** people present ***to Volume*** is to ***use a NoSQL database*** to store and index information. This is an appropriate solution in many cases; however, ***most data is well suited to relational databases***, and these often provide advantages over non-relational databases.

One of the most important steps in choosing how to set up your analytic workflow is the proper framing of your question. This will help you understand:

1. What data sources do you need?
2. How will you transform that data?
3. How will you represent your data?

1.6. Working with Big Data locally

Here are various elements that you need to watch out for when dealing with big data.

- `Storage` –> Hard disk space
- `Memory` –> RAM
- `Processing` –> CPU cores

1.6.1. Storage

Thoughts/Discussion ?

Is storage a serious issue when dealing with big data ?

Storage is very cheap these days, and we worry less about storage. For example, most laptops come with 1 TB HDD and/or 256 GB SDD. So it's not a big deal to store big data on a computer, and overall we worry less about storage these days. The main question is how we can efficiently process this stored data. You learned *(and also experienced)* how storing files in ***parquet*** format can significantly ***reduce the size of the files***, and in return, it ***reduces the I/O and network traffic***. Reduction in the I/O and network traffic is good when processing big data.

1.6.2. Memory

How are you making use of memory in your laptop? Let's check it out.

We will be using the airline data file. This is how our data and scehma looks like; Click on the toggle to see the data.

UniqueCarrier	TailNum	ArrDelay	DepDelay	Origin	Dest	Distance	year
AA	N433AA	192	26	ORD	PHL	678	2002
AA	N482AA	5	12	ORD	PHL	678	2002
AA	N251AA	-16	2	ORD	PHL	678	2002
AA	N569AA	-3	6	ORD	PHL	678	2002

Field	Description
UniqueCarrier	Airline Carrier Code
TailNum	Used to identify a specific airplane
ArrDelay	Arrival delay in minutes
DepDelay	Departure delay in minutes
Origin	Origin airport code
Dest	Destination airport code
Distance	Distance in miles
year	year

Let's define a question: Interested in knowing "UniqueCarrier" delays in the year 2004 where arrival delay ("ArrDelay") > 10 minutes.

Note

We will work on datasets with various sizes to understand how to handle big data. We will follow the below progression:

- Start with a 3 GB CSV file containing around 9 Million rows.
- Move on to a 10 GB CSV file containing around 27 Million rows.
- Then work on a 20 GB CSV file containing around 55 Million rows.

Warning

For the 10 GB and 20 GB - If you want to work with larger datasets and have fun, you can download them. However, proceed with caution and ensure that you have enough disk space before attempting to download. Here are the download links:

- 10 GB CSV file - Download here.
- 20 GB CSV file - Download here.

Note: The file is zipped, so you need to unzip it before analyzing it.

```
import os
os.environ['R_HOME'] = '/Users/gittugeorge/miniforge3/envs/525_dev_2025/lib/R'
# R_HOME = '/Users/gittugeorge/miniforge3/envs/525_dev_2025/bin/R'
```

Warning

You must change the above path to your R path for your Jupyter environment to work. You can run `conda run -n 525_dev_2024 python -m site` to get an idea of the path.

```
%load_ext rpy2.ipython
```

```
%cd /Users/gittugeorge/Desktop/525_2025/figshareexp
```

```
/Users/gittugeorge/Desktop/525_2025/figshareexp
```

Let's first check the size of the CSV files that we will be using;

```
%%sh
du -sh figshareairline/combined_data.csv
# The files below are not given to students
du -sh figshareairline/bigdata/combined_10gb.csv
du -sh figshareairline/bigdata/combined_20gb.csv
```

```
3.1G    figshareairline/combined_data.csv
9.3G    figshareairline/bigdata/combined_10gb.csv
 19G    figshareairline/bigdata/combined_20gb.csv
```

1.6.2.1. What we know! 🤓

Let's apply what we know for data crunching using Pandas and tidyverse.

```
import pandas as pd
```

```
%%R
suppressMessages(library(dplyr))
suppressMessages(library(readr))
```

1.6.2.1.1. 3 GB CSV file

Note

The 3GB data is what you combined previously in section Using rest API (lab lecture)

Let's see how it is in pandas

```
%%time
df = pd.read_csv("figshareairline/combined_data.csv")
print(df[(df.year ==2004) & (df.ArrDelay >10)]["UniqueCarrier"].value_counts())
```

```
UniqueCarrier
WN    225167
DL    189001
AA    176385
MQ    130511
UA    126692
NW    125030
US     95326
XE     94467
OO     87328
OH     83799
CO     74989
EV     68299
DH     66330
HP     55272
FL     42911
AS     42187
B6     19421
TZ     17751
HA      3642
Name: count, dtype: int64
CPU times: user 17.7 s, sys: 2.62 s, total: 20.3 s
Wall time: 20.3 s
```

Let’s see how it is in tidyverse

```
%%time
%%R
df <- read_csv("figshareairline/combined_data.csv")
result <- df %>%
    filter(year == 2004, ArrDelay >10) %>%
    count(UniqueCarrier)
print(result)
```

```
Rows: 86289323 Columns: 8
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (4): UniqueCarrier, TailNum, Origin, Dest
dbl (4): ArrDelay, DepDelay, Distance, year

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 19 × 2
   UniqueCarrier      n
   <chr>          <int>
 1 AA            176385
 2 AS             42187
 3 B6             19421
 4 CO             74989
 5 DH             66330
 6 DL            189001
 7 EV             68299
 8 FL             42911
 9 HA              3642
10 HP             55272
11 MQ            130511
12 NW            125030
13 OH             83799
14 OO             87328
15 TZ             17751
16 UA            126692
17 US             95326
18 WN            225167
19 XE             94467
CPU times: user 3min 1s, sys: 2.7 s, total: 3min 4s
Wall time: 25.3 s
```

1.6.2.1.2. 10 GB CSV file

```
%%time
df = pd.read_csv("figshareairline/bigdata/combined_10gb.csv")
print(df[(df.year ==2004) & (df.ArrDelay >10)]["UniqueCarrier"].value_counts())
```

```
UniqueCarrier
WN    675501
DL    567003
AA    529155
MQ    391533
UA    380076
NW    375090
US    285978
XE    283401
OO    261984
OH    251397
CO    224967
EV    204897
DH    198990
HP    165816
FL    128733
AS    126561
B6     58263
TZ     53253
HA     10926
Name: count, dtype: int64
CPU times: user 54.1 s, sys: 20 s, total: 1min 14s
Wall time: 1min 20s
```

```
%%time
%%R
df <- read_csv("figshareairline/bigdata/combined_10gb.csv")
result <- df %>%
    filter(year == 2004, ArrDelay >10) %>%
    count(UniqueCarrier)
print(result)
```

```
Rows: 258867969 Columns: 8
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (4): UniqueCarrier, TailNum, Origin, Dest
dbl (4): ArrDelay, DepDelay, Distance, year

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 19 × 2
   UniqueCarrier      n
   <chr>          <int>
 1 AA            529155
 2 AS            126561
 3 B6             58263
 4 CO            224967
 5 DH            198990
 6 DL            567003
 7 EV            204897
 8 FL            128733
 9 HA             10926
10 HP            165816
11 MQ            391533
12 NW            375090
13 OH            251397
14 OO            261984
15 TZ             53253
16 UA            380076
17 US            285978
18 WN            675501
19 XE            283401
CPU times: user 8min 17s, sys: 1min 4s, total: 9min 21s
Wall time: 2min 17s
```

How you all doing with this ? 😅

1.6.2.1.3. 20 GB CSV file

Most users will fail to run a 20GB file. I also used to fail in previous years, but I have upgraded my laptop, which now has 48GB of RAM. Hence, I am able to run it.

```python
df = pd.read_csv("figshareairline/bigdata/combined_20gb.csv")
print(df[(df.year ==2004) & (df.ArrDelay >10)]["UniqueCarrier"].value_counts())
```

```
UniqueCarrier
WN    1351002
DL    1134006
AA    1058310
MQ     783066
UA     760152
NW     750180
US     571956
XE     566802
OO     523968
OH     502794
CO     449934
EV     409794
DH     397980
HP     331632
FL     257466
AS     253122
B6     116526
TZ     106506
HA      21852
Name: count, dtype: int64
```

```
%%R
df <- read_csv("figshareairline/bigdata/combined_20gb.csv")
result <- df %>%
filter(year == 2004, ArrDelay >10) %>%
count(UniqueCarrier)
print(result)
```

```
Rows: 517735938 Columns: 8
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (4): UniqueCarrier, TailNum, Origin, Dest
dbl (4): ArrDelay, DepDelay, Distance, year

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 19 × 2
   UniqueCarrier       n
   <chr>           <int>
 1 AA            1058310
 2 AS             253122
 3 B6             116526
 4 CO             449934
 5 DH             397980
 6 DL            1134006
 7 EV             409794
 8 FL             257466
 9 HA              21852
10 HP             331632
11 MQ             783066
12 NW             750180
13 OH             502794
14 OO             523968
15 TZ             106506
16 UA             760152
17 US             571956
18 WN            1351002
19 XE             566802
```

```
df = pd.read_csv("figshareairline/bigdata/combined_20gb.csv")
print(df[(df.year ==2004) & (df.ArrDelay >10)]["UniqueCarrier"].value_counts())
```

```
df <- read_csv("figshareairline/bigdata/combined_20gb.csv")
result <- df %>%
filter(year == 2004, ArrDelay >10) %>%
count(UniqueCarrier)
print(result)
```

Here you are loading the entire data to the memory. This data is now around 20 GB, and my laptop got `RAM` of about 16 GB and my previous computer doesn't have enough memory to hold the entire file, and hence I get memory errors. 🤪

Thoughts/Discussion ?

How can we get over this problem ?

Get a computer with more memory. (This approach is called a `scale-UP` solution).

Alternatively, you can also consider a scale-OUT solution where you connect all the group members laptops to process the data.

We will learn more on about the following in our upcoming classes;

- Principles and applications of cloud computing
- Scale UP solutions
- Scale OUT solutions

Unfortunately, this week, this option is not available. You need to figure out ways to handle the data on your laptop. 😔

1.6.3. Some tactics to deal with memory issue 🤩

1.6.3.1. Load just what is needed

Here only load the columns that we plan to work with.

```
%%time
use_cols = ['ArrDelay','DepDelay','Distance','UniqueCarrier']
df = pd.read_csv("figshareairline/combined_data.csv",usecols=use_cols)
print(df["UniqueCarrier"].value_counts())
```

```
UniqueCarrier
WN    13194660
DL    10435886
AA     9672922
UA     8821384
US     8286980
NW     6946627
CO     4976761
MQ     3954895
OO     3090853
XE     2350309
HP     2224941
AS     2162672
TW     1890420
EV     1697172
OH     1464176
FL     1265138
YV      854056
B6      811341
DH      693047
9E      521059
F9      336958
HA      274265
TZ      208420
AQ      154381
Name: count, dtype: int64
CPU times: user 14.8 s, sys: 6.03 s, total: 20.8 s
Wall time: 38.7 s
```

1.6.3.2. Make use of the datatypes wisely

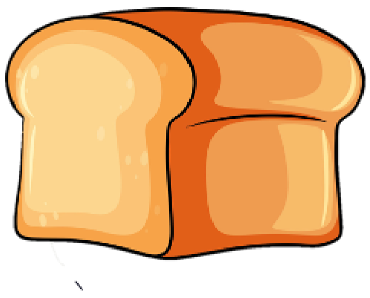
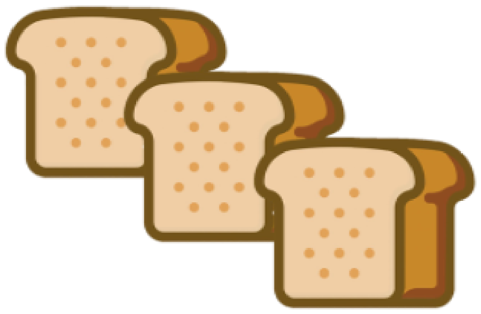
Some datatypes consume less space. In the following example, by just changing the type float64 -> float32 saved about 50% of space.

```
print(f"Memory usage with float64: {df[['ArrDelay','DepDelay','Distance']].memory_usage().sum() / 1e6:.2f
print(f"Memory usage with float32: {df[['ArrDelay','DepDelay','Distance']].astype('float32', errors='igno
Memory usage with float64: 2070.94 MB
Memory usage with float32: 1035.47 MB
```

Probably you already know this; you can use argument “dtype” to set the data type while reading to pandas.

1.6.3.3. Process data in chunks

Breaking the ~~bread~~ data to small chunks, and process on those small chunks

We can load the big file in small chunks and process on those chunks in an iterative fashion. We can pass `chunksize` as a parameter while using `read_csv`. Read more on it here.

```
%%time
counts = pd.Series(dtype=int)
for chunk in pd.read_csv("figshareairline/combined_data.csv", chunksize=10_000_000):
    counts = counts.add(chunk[(chunk.year ==2004) & (chunk.ArrDelay >10)]["UniqueCarrier"].value_counts()
print(counts.astype(int))
```

```
UniqueCarrier
AA    176385
AS     42187
B6     19421
CO     74989
DH     66330
DL    189001
EV     68299
FL     42911
HA      3642
HP     55272
MQ    130511
NW    125030
OH     83799
OO     87328
TZ     17751
UA    126692
US     95326
WN    225167
XE     94467
dtype: int64
CPU times: user 18.5 s, sys: 2.97 s, total: 21.5 s
Wall time: 21.5 s
```

1.6.3.4. Don’t Combine

In our example, the combined file you are dealing with was made from many files. So if you get many files, try to see if you can work on those small files and then combine the results.

1.6.4. Lets Apply

You probably already knew about the above 4 points, but you might have overlooked them. But these small tricks can make a big difference when dealing with big data.

```
%%time
use_cols = ['ArrDelay','DepDelay','Distance','UniqueCarrier']
df = pd.read_csv("figshareairline/bigdata/combined_20gb.csv",usecols=use_cols)
print(df["UniqueCarrier"].value_counts())
```

```
UniqueCarrier
WN    79167960
DL    62615316
AA    58037532
UA    52928304
US    49721880
NW    41679762
CO    29860566
MQ    23729370
OO    18545118
XE    14101854
HP    13349646
AS    12976032
TW    11342520
EV    10183032
OH     8785056
FL     7590828
YV     5124336
B6     4868046
DH     4158282
9E     3126354
F9     2021748
HA     1645590
TZ     1250520
AQ      926286
Name: count, dtype: int64
CPU times: user 1min 13s, sys: 12.1 s, total: 1min 25s
Wall time: 1min 29s
```

Woohoo!!! Now I am able to load only what I need from the 20 GB file. Previously, when I attempted to load everything, it failed. 💪🏼💪🏼

Thoughts/Discussion ?

Have you noticed that even when loading a portion of data into memory, it takes about 5 minutes to load? What could be the reason ?

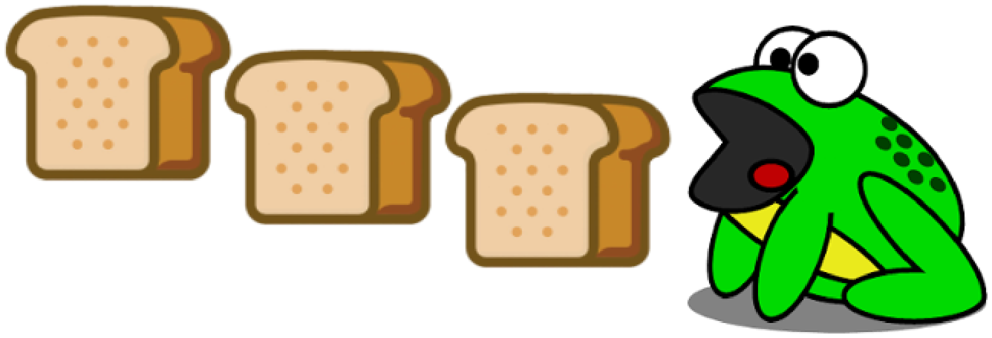
Row based file format, where we store data also matters. We will learn more about this in the upcoming classes.

1.6.5. Processing

Before we start, let's recollect the key skills that we have until now; We know...

- How to bring in just what we want into memory.
- How to process in chunks when the data don't fit in memory. *(remember the bread)*

By default, pandas or dplyr execute things `single-threaded` *(single ~~frog~~ CPU-core finds answers to these chunks)*, which means we are not able to process things parallelly. So even in the example where we broke our data into small chunks (using chunksize=), we are `iteratively` processing each chunk. Like this...



Thoughts/Discussion ?

Can you think of any options in improving speed ?

1.6.6. Use R and python interchangeably

We are detouring a bit here.

Attention

You only want to use R for your first milestone, and all other milestones will be done in Python. I am showing you this mainly for you to understand some very important concepts on serialization and deserialization (next class).

```
# I am only loading 100000 rows because if I try to load all the rows and transfer them to R, I get a mem
df = pd.read_csv("figshareairline/combined_data.csv", nrows=100000)
```

```
%%time
%%R -i df
start_time <- Sys.time()
suppressMessages(library(dplyr))
result <- df %>% count(UniqueCarrier)
print(result)
end_time <- Sys.time()
print(end_time - start_time)
```

```
   UniqueCarrier     n
1             AA  6961
2             AS  5587
3             CO  2856
4             DL 13512
5             HP  7422
6             NW  4139
7             TW 12254
8             UA 13728
9             US 14831
10            WN 18710
Time difference of 0.01170301 secs
CPU times: user 621 ms, sys: 26.1 ms, total: 647 ms
Wall time: 656 ms
```

Thoughts/Discussion ?

- Why do you think my laptop crashes when I try to transfer the entire data from python to R?
- Do you notice the time that the cell took to execute vs time that R took to run the code ? Why you think there is this difference ?

That is why we are learnign serialization and deserialization. We will learn more about this in the upcoming classes.. Look at the same experiment we will be doing in arrow

See also

Checkout this article to know more about sharing data among python and R.

1.7. What did we learn today?

- We revisited what you already know about data and the various ways you have used to approach it.
- Discussed how data appears on a website and the various ways of obtaining it.
- Explored big data and the various dimensions (5V) associated with it.
- Discussed various techniques for bringing a smaller footprint of data into memory.
- How chunking up can help with memory issues.

More than anything, you experienced difficulty dealing with big data. You now understand how storage, memory, and processing are critical when working with big data. We will introduce you to some technologies and learn some concepts that can help you work with big data. You will learn not only how to use them but also why certain techniques are faster and the underlying implementation that makes them a better choice when dealing with big data.

1.8. Activity

- Summarize your thoughts:
    - Did you experience difficulty with the big data ?
    - What do you think you need to learn to work with big data ? And this course is about that.

- Scale UP solutions
- Scale OUT solutions
- Cloud computing
- Better file format for storing data
- Parallel processing
- Distributed computing