Big Data Workshop #1:

An Introduction to Big Data

Melissa Van Bussel, Benjamin Burr, Amin Nabavi, Dr. Shirley Mills

Queen Elizabeth Scholarship - Advanced Scholars Program
September 9th, 2021

Today's workshop

- Introduction to Big Data
- Installing Apache Spark (for R)

The slides for today's workshop are available at the following link:
https://github.com/melissavanbussel/QES-AS-R-Workshops

Goals for today

There are a lot of buzz words (or phrases) surrounding this topic:

- Big Data
- Databases
- The cloud
- Parallelization
- Hadoop
- Hive
- Spark

The goal for today is to gain a rough sense of these words, and realize that they're not as scary as they sound.

What is "Big Data"?

Defintion of Big Data

- Big Data refers to data that is, well, big
- "How big" depends on the specific scenario, but for now, think of "Big Data" as data that is too big to be stored or processed using traditional methods
  - We can't just keep it saved on our computers and load it into R, for example
- Consider the `palmerpenguins` dataset from the previous workshops
  - This file was 344 observations of 8 variables
  - It was about 0.02GB
- What about datasets that have millions of observations? Hundreds of columns? Thousands of GB?
- What about if we had multiple datasets, all of which are themselves "big"?

The 3 V's

In 2001, an analyst from Gartner Inc. (Doug Laney) coined the "3 V's" of Big Data. These 3 words can be considered characteristics of big data:

Volume:

- "Big" data is often in the TBs (1 TB = 1000GB)

Velocity:

- The speed at which the data is processed and generated
- Big data is usually available in real-time and can be analyzed efficiently (more about this later, but this is because we typically use more powerful computing set-ups and more efficient storage methods than we would to work with the `palmerpenguins` dataset, for example)

Variety:

- Big data comes in a "variety" of formats and types -- we may have numeric data, text data, images...

Some additional V's

While the 3 V's from the previous slide are generally agreed upon by everyone, there are some additional V's that you may hear about, depending on the source.

Veracity:

- The quality of the data
- Big data needs to be of high quality

Value:

- Big data needs to be valuable
- Just collecting data is not useful; the value comes from analyzing it to gain useful insights

A quick aside

The word "data" is a plural word. The singular version of the word is "datum".

Many datum = data

For example:

> Each **datum is** visualized on the plot

> The **data are** reliable

"Big Data" is a noun itself. For example, you may hear sentences like:

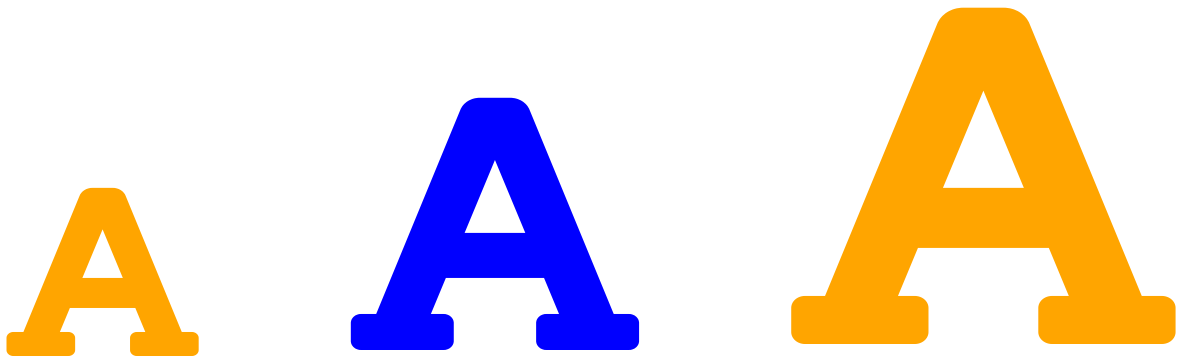
- Big Data is a popular field
- Big data is difficult to anlayze

How do we deal with medium-sized data?

How do we deal with medium-sized data?

Before we talk about working with big data, let's talk about working with medium-sized data. This will introduce us to some key concepts we'll need later.

- "Medium data" isn't *really* a term
- Let's use it loosely here to mean data that are:
  - Big enough that we should be mindful about working with them
  - But small enough that we can still analyze them on our own computers



Code optimization

- When working with datasets that are large, but still small enough to be analyzed with our own computers, we need to optimize our code
- Code optimization is our first strategy for dealing with large datasets
- We need to think about the most **efficient** way to perform a task
- In some cases, this may mean rewriting existing code that we have already written

An example of optimization

Often, there's more than one correct way that we can complete a task.

Usually, there is a "**fastest**" way of completing a task -- an "**optimized**" way.

Doing things the optimized way makes a huge difference, especially if we have to repeat the task many times.

Benchmarking

- If we have more than one way of completing a task and want to know which way is the fastest, we can time how long each method takes
- This is known as **benchmarking**
- Benchmarking is a commonly used strategy for optimizing code

An example of benchmarking in R

```
# Load package for "benchmarking" code
library(bench)

# Generate 1,000,000 random numbers in [0,1]
random_nums <- runif(1000000)

# Compute the mean of random_nums
# Using two different approaches
compute_times <- bench::mark(mean(random_nums),
sum(random_nums) / length(random_nums))

# How long did each option take?
compute_times
```

In R, we can compute the mean of a set of numbers by using the `mean` function. Alternatively, we can first compute the `sum` of the values, and divide it by the number of observations we have. Which method is faster?

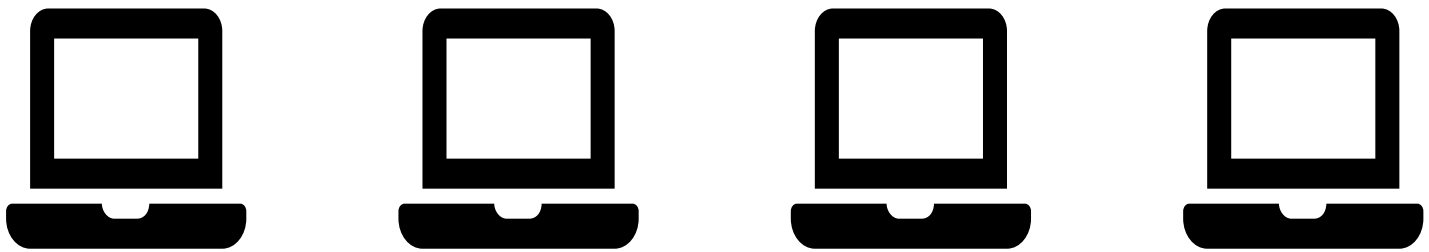
The fastest method is not always what you expect! Benchmarking can help you determine where the bottleneck in your program is.

```
## # A tibble: 2 x 6
##   expression                                   min median `itr/sec` mem_alloc
##   <bch:expr>                                <bch:> <bch:>     <dbl> <bch:byt>
## 1 mean(random_nums)                         2.79ms 3.68ms      264.    22.7KB
## 2 sum(random_nums)/length(random_nums) 1.33ms 1.52ms      560.        0B
## # ... with 1 more variable: `gc/sec` <dbl>
```

Parallel processing

Parallel processing is our second strategy for dealing with medium-sized data.

But...what is it?



Parallel processing

Let's say we work at a restaurant, and we have 100 dishes that need to be washed, and 4 employees available to wash dishes. What would be the best way to divide the work?

a) 1 person washes all 100 dishes while the other 3 employees stand and watch

b) Each person washes 25 dishes (all employees wash the dishes at the same time)

- Of course, we would pick option B!
- When we have code that can easily be broken up into independent tasks (like washing dishes), we should use parallel processing
- These days, most computers and laptops have 4 (or more) cores, and we can spread out (many) tasks across the cores
- This isn't the behaviour by default, though! We have to specifically ask our computers to use all of the cores.

Parallel processing in R

(You may have to have to have admin privileges on your computer to do this -- may need to change firewall settings)

```
library(parallel)
detectCores() # This computer has 8 cores
```

```
## [1] 8
```

```
# You can use up to the number of cores you have
# but you can also use less
my_cluster <- makeCluster(4)

# For each number in 1:20, add 3 to it
parSapply(my_cluster, 1:20, get("+"), 3)
```

```
##  [1]  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

```
# Close it at the end
stopCluster(my_cluster)
```

Databases

A third strategy for working with medium-sized (and big) data is to use a database.

A **dataset** is one table (think of the `palmerpenguins` dataset):

A **database** is a collection of datasets, typically stored in a way that is easier / faster to access and manipulate the data:

Relational databases

A **relational database** is a type of database where the tables are related to one another, often by an ID variable.

Students

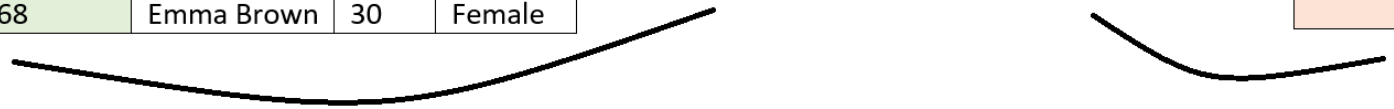
Student ID	Name	Age	Gender
1234	Bob Smith	21	Male
5678	Alice Jones	23	Female
2468	Emma Brown	30	Female

Course_Marks

Student ID	Final grade	Course code
5678	A+	BIOL1000
1234	B-	BIOL1000

Course_Descriptions

Course code	Course instructor	Course name
BIOL1000	Dr. Someone	Introduction to Biology
PHYS1100	Ms. Anybody	Introductory Physics for the Life Sciences



What is SQL?

SQL stands for **"Structured Query Language"**. It's a language for accessing and manipulating databases. With SQL, you can (for example):

- Create new databases or tables within an existing database
- Insert, update, or delete records from an existing database
- Perform queries on a table (or tables) in an existing database

Working with databases in R

Remember when we learned about the `dplyr` functions in R? We saw:

- `select`
- `filter`
- `arrange`
- `group_by`
- `summarise`
- `mutate`

SQL can do all of these operations! In fact, there's a package called `dbplyr` that will allow you to connect to a database using R, and analyze the database using the `dplyr` syntax you already learned (no need to learn SQL!). If you're interested: https://db.rstudio.com/r-packages/dplyr/

How do we deal with "big" data?

How do we deal with "big" data?

We need to use specialized tools and methods:

- For storing data
- For analyzing data

Storing big data

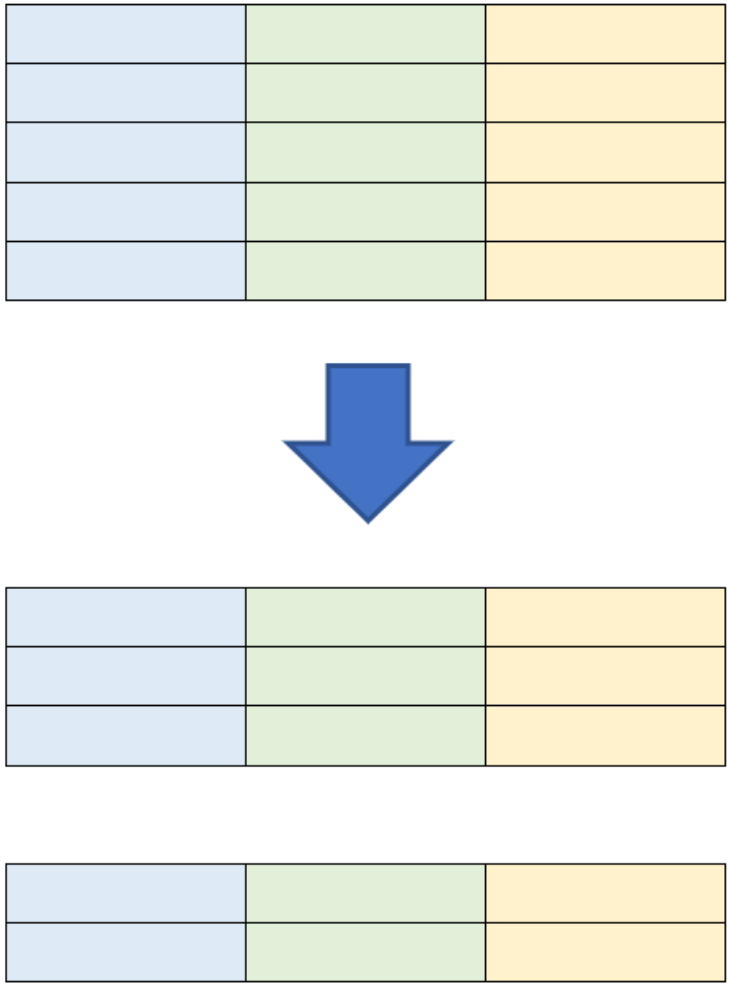
Data partitioning

If a dataset is too large to be managed as a single file, it must be **partitioned** (split up into smaller pieces).

There are two main ways to partition a dataset:

1. Horizontal partitioning ("by rows")
2. Vertical partitioning ("by columns")

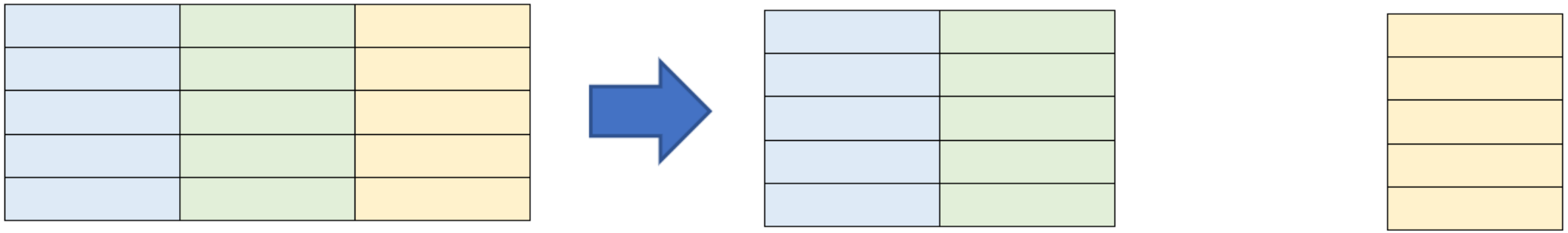
Horizontal partitioning



With horizontal partitioning, different rows go into different tables/files.

Vertical partitioning

With vertical partitioning, different columns go into different tables/files.



Why is (big) data partitioned?

There are many reasons, including:

- **Size**: Some datasets are just too big to store in a single file (there may be physical hardware limits).
- **Speed**: Since analyses can be performed on a smaller subset of the data, analyses will run faster.
- **Convenience**: Less frequently used rows (in the case of horizontal partitioning) or columns (in the case of vertical partitioning) can be stored separately from frequently used rows/columns.
- **Paralellization**: If the dataset should be analyzed in parallel, different partitions can be sent to different cores.
- **Security**: Sometimes, some parts of a dataset are more confidential than others (e.g., certain variables are confiential). Different security measures can be applied to the different partitions.

Storing big data

There are also special types of files and file management systems that are commonly used for storing big data. We will talk about these very briefly in a few slides.

Analyzing big data

An analogy

Let's say we need a vehicle, but don't currently have one. We have two main options:

1) **Buy** a new car, or

2) **Rent** a car whenever we need one

An analogy

The first option is appealing, but the second option is a better choice in any of the following scenarios:

- We don't have enough money to afford a car right now
- We have enough money, but we won't use the car frequently enough for it to be a "worthwhile" purchase
- We have enough money, but only enough to afford a car that won't meet our needs

(The other benefit of renting is that we don't need to worry about car maintenance 😄)

Cloud computing

Many organizations are moving towards cloud computing. This means they rent services and infrastructure, on a pay-as-you-go basis. They might rent:

- Data storage
- Applications
- Processing / computing power

You've likely used cloud services, too! Think of Google Drive: you (or your organization) pays a subscription fee in order for you to have file storage and access tools such as Google Docs, rather than having to purchase a one-time license for an installation of software like Microsoft Word.

Specialized tools - a brief history

- Google introduced the "Google File System" in 2003
  - Search engines were unable to store the information required for web searches on a single computer, needed to be partitioned into several files across different machines
- Google introduced "MapReduce" in 2004
  - A way for performing operations/analysis across the Google File System
- Yahoo introduced "Hadoop" and the "Hadoop Distributed File System" (HDFS) in 2006
  - Combined the Google File System and MapReduce into one open source project
- Facebook introduced "Hive" in 2008
  - Made it easier to work with Hadoop (added SQL support)

Apache Spark

- "Spark" was introduced in 2010 as an open source project, donated to the Apache Spark Foundation in 2012 ("Apache Spark")
- An improvement on Hadoop
  - Provided more verbs than Hadoop
  - Was more optimized (faster)
  - Was easier to use

	Hadoop record (2013)	**Apache Spark record (2014)**
Size of data	102.5TB	100TB
Elapsed time	72 minutes	23 minutes
Number of computers	2,100	206

Apache Spark

According to Apache:

> Apache Spark is a unified analytics engine for large-scale data processing.

In other words, it's a tool for analyzing big data. It supports:

- Many different file storage systems
- Many different libraries (i.e., tools for analyzing, visualizing, and modelling big data)
- Many different cluster management systems (for parallelization across many computers)

It also has integration with other programming languages, such as R!

sparklyr

- `sparklyr` is an R package
- It's an R interface to Apache Spark
- Instead of writing code in Scala (the programming language that Apache Spark is written in), you can write code in R, and R will translate it for you

Installing sparklyr

First, you'll need to have Java installed on your computer. It will likely already be installed for many people. To check:

```
system("java -version")
```

If Java isn't already installed, go to https://java.com/en/download/.

Installing sparklyr

Install the `sparklyr` package in R:

```
install.packages("sparklyr")
```

Load the `sparklyr` package:

```
library(sparklyr)
```

Install Apache Spark:

```
spark_install()
```