Module_2_key

Elyse, Jeanne & Sheila

Table of contents

Learning Objectives (Module 2)

By the end of this module, you will be able to:

- **Understand** the mindset shift from "loading" to "viewing" data
- **Read and explore** large datasets without loading them into memory
- **Convert** between file formats (CSV Parquet) efficiently
- **Apply** basic dplyr operations with Arrow's lazy evaluation
- **Optimize** query performance through proper `collect()` placement
- **Build** multi-step pipelines that process gigabytes of data

The Mindset Shift: "View" Don't "Load"

Traditional R Approach vs. Arrow Approach

Traditional R thinking:

```
# Load EVERYTHING into memory first

data <- read_csv("huge_file.csv")  #   Crash!

data |>
```

```
  filter(year == 2023)
```

Arrow thinking:

```
# Create a "view" of the data, then filter

data <- open_dataset("huge_file.csv")  #   Instant! d

ata |> filter(year == 2023)
  |> collect()  # Only brings filtered results to memory
```

Rule of thumb:

- **< 100MB**: Regular R is fine
- **100MB - 1GB**: Arrow is nice to have
- **1GB+**: Arrow becomes really valuable
- **RAM size+**: Arrow is essential

Key Concept: Lazy Evaluation

Arrow uses **lazy evaluation** - it builds up a query plan without actually executing it until you call `collect()`.

Think of it like:

- **Traditional R**: "Cook the entire meal, then throw away what you don't want"
- **Arrow**: "Plan the meal, shop for only what you need, then cook just that"

Arrow Basics (10 minutes)

Setting Up Our Big Data Playground - done in pre-workshop materials

We're working with the **real Seattle Library dataset** - over 40 million rows of checkout data!

Step 1: Create a Directory

First, let's create a special folder to store our data:

```
# Create a "data" directory if it doesn't exist already
# Using showWarnings = FALSE to suppress warning if directory already exists

dir.create("data", showWarnings = FALSE)
```

Step 2: Download the Dataset

Now for the fun part! We'll download the Seattle Library dataset (9GB).

Important: This is a 9GB file, so:

- Make sure you have enough disk space

```
# Download Seattle library checkout dataset:

# 1. Fetch data from AWS S3 bucket URL
# 2. Save to local data directory
# 3. Use resume = TRUE to allow continuing interrupted downloads

curl::multi_download("https://r4ds.s3.us-west-2.amazonaws.com/seattle-library-checkouts.csv"
```

Why USE: `curl::multi_download()`

- Shows a progress bar (great for tracking large downloads)
- Can resume if interrupted (super helpful for big files!)
- More reliable than base R download methods

Step 3: Verify the Download

After the download completes, let's make sure everything worked:

```
# Check if the Seattle library dataset file exists and print its size:
# 1. Verify file exists at specified path
# 2. Calculate file size in gigabytes by dividing bytes by 1024^3

file.exists("data/seattle-library-checkouts.csv")
```

```
[1] TRUE
```

```
file.size("data/seattle-library-checkouts.csv") / 1024^3  # Size in GB
```

```
[1] 8.579315
```

"Who thinks their computer could handle loading 9GB into memory?"

The `open_dataset()` Magic

Now let's see the fundamental difference between `read_csv()` and `open_dataset()`:

DON'T RUN!

```
# Traditional approach - would crash most computers!

#seattle_library_checkouts <- read_csv("useR_memory/data/seattle-library-checkouts.csv") # D
```

RUN

```
#load in the packages and install if needed with code below


# Function to check and install required packages
required_packages <- c("tidyverse", "arrow")

# Install missing packages
for (pkg in required_packages) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg)
    #library(pkg, character.only = TRUE)
  }
}

#Load libraries
lapply(required_packages, library, character.only = TRUE)
```

```
Warning: package 'tidyverse' was built under R version 4.3.3
```

```
Warning: package 'tibble' was built under R version 4.3.3
```

```
Warning: package 'tidyr' was built under R version 4.3.3

Warning: package 'readr' was built under R version 4.3.3

Warning: package 'purrr' was built under R version 4.3.3

Warning: package 'dplyr' was built under R version 4.3.3

Warning: package 'stringr' was built under R version 4.3.3

Warning: package 'forcats' was built under R version 4.3.3

Warning: package 'lubridate' was built under R version 4.3.3

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom

Warning: package 'arrow' was built under R version 4.3.3


Attaching package: 'arrow'

The following object is masked from 'package:lubridate':

    duration

The following object is masked from 'package:utils':

    timestamp
```

```
[[1]]
 [1] "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"     "readr"
 [7] "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"
[13] "grDevices" "utils"     "datasets"  "methods"   "base"

[[2]]
 [1] "arrow"     "lubridate" "forcats"   "stringr"   "dplyr"     "purrr"
 [7] "readr"     "tidyr"     "tibble"    "ggplot2"   "tidyverse" "stats"
[13] "graphics"  "grDevices" "utils"     "datasets"  "methods"   "base"
```

What `open_dataset()` does:

- Creates an Arrow dataset object that "points to" your CSV file
- Doesn't actually load the data into memory yet
- Acts like a "view" or "window" into your data
- When you run this code with `open_dataset()`, Arrow does something clever:

  1. It peeks at the first few thousand rows
  2. Figures out what kind of data is in each column
  3. Creates a roadmap of the data
  4. Then… it stops!

  That's right - it doesn't load the whole 9GB file. Imagine Arrow as a really efficient librarian who:

  - Creates an index of where everything is
  - Only gets books (data) when you specifically ask for them
  - Keeps track of what's where without moving everything

```
# Arrow approach - creates a "view" without loading
seattle_csv <- open_dataset("useR_memory/data/seattle-library-checkouts.csv", format = "csv"

# View Object
seattle_csv
```

```
FileSystemDataset with 1 csv file
12 columns
UsageClass: string
CheckoutType: string
MaterialType: string
CheckoutYear: int64
CheckoutMonth: int64
Checkouts: int64
Title: string
ISBN: string
Creator: string
Subjects: string
Publisher: string
PublicationYear: string
```

```r
library(glue) # string interpolation - cleaner alternative to paste()
```

```
Warning: package 'glue' was built under R version 4.3.3
```

```r
# Check out how much memory this is using.
glue("Memory used by Arrow object: {format(object.size(seattle_csv), units = 'KB')}")
```

```
Memory used by Arrow object: 0.5 Kb
```

```r
# Let's see what the file size we are actually working with
file_size_bytes <- file.size("useR_memory/data/seattle-library-checkouts.csv")
file_size_gb <- file_size_bytes / (1024^3)  # Convert to GB
glue("Estimated file size: {round(file_size_gb, 2)} GB")
```

```
Estimated file size: 10.13 GB
```

```r
# Peek at the structure without loading
seattle_csv |>
  glimpse()
```

```
FileSystemDataset with 1 csv file
47,854,892 rows x 12 columns
$ UsageClass      <string> "Physical", "Digital", "Digital", "Digital", "Physica~
$ CheckoutType    <string> "Horizon", "OverDrive", "OverDrive", "OverDrive", "Ho~
$ MaterialType    <string> "VIDEODISC", "EBOOK", "AUDIOBOOK", "EBOOK", "BOOK", "~
$ CheckoutYear     <int64> 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024,~
$ CheckoutMonth    <int64> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
$ Checkouts        <int64> 9, 1, 3, 2, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1,~
$ Title           <string> "Unlocked / Lionsgate Premiere ; Grindstone Entertain~
$ ISBN            <string> "", "9781626254008", "9781781107966", "9780306826610"~
$ Creator         <string> "", "Raychelle Cassada Lohmann", "J. K. Rowling", "Zo~
$ Subjects        <string> "United States Central Intelligence Agency Drama, Bio~
$ Publisher       <string> "Lions Gate Entertainment,", "New Harbinger Publicati~
$ PublicationYear <string> "[2017]", "2023", "2022", "2022", "[2023]", "[2019]",~
```

Key insight: Arrow creates a "catalog" or "view" of your data without actually reading it into memory!

Understanding `collect()`: The Bridge Between Arrow and R

What is `collect()`?

- `collect()` is the function that **executes** your Arrow query and brings the results into regular R memory
- Think of it as the "Go!" button that turns your query plan into actual data

Exploring Structure Without Loading

```r
# 1. BASIC DATASET INFO (no data loading)
glue("=== DATASET OVERVIEW ===")
```

```
=== DATASET OVERVIEW ===
```

```r
glue("File size: {format(file.size('useR_memory/data/seattle-library-checkouts.csv'), units
```

```
File size: 10882220734
```

```r
glue("Memory used by Arrow object: {format(object.size(seattle_csv), units = 'KB')}")
```

```
Memory used by Arrow object: 0.5 Kb
```

```r
# 2. SCHEMA EXPLORATION (metadata only)
seattle_csv$schema  # Column types
```

```
Schema
UsageClass: string
CheckoutType: string
MaterialType: string
CheckoutYear: int64
CheckoutMonth: int64
Checkouts: int64
Title: string
ISBN: string
Creator: string
Subjects: string
Publisher: string
PublicationYear: string
```

```r
glue("Number of columns: {ncol(seattle_csv)}")
```

```
Number of columns: 12
```

```r
# 3. SMART SAMPLING (minimal data loading)
seattle_csv |>
  slice_sample(n = 1000) |>  # Random sample instead of just head()
  glimpse()
```

```
FileSystemDataset with 1 csv file (query)
?? rows x 12 columns
$ UsageClass      <string> "Physical", "Physical", "Physical", "Physical", "Phys~
$ CheckoutType    <string> "Horizon", "Horizon", "Horizon", "Horizon", "Horizon"~
$ MaterialType    <string> "SOUNDDISC", "BOOK", "BOOK", "BOOK", "BOOK", "BOOK", ~
$ CheckoutYear     <int64> 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024,~
$ CheckoutMonth    <int64> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
$ Checkouts        <int64> 1, 7, 1, 1, 4, 1, 1, 1, 1, 1, 3, 1, 1, 3, 2, 1, 1, 14~
$ Title           <string> "Fear fun / Father John Misty.", "Friends, lovers, an~
$ ISBN            <string> "", "9798885783866", "0732298903, 9780732298906", "17~
$ Creator         <string> "Misty, Father John", "Perry, Matthew, 1969-2023,", "~
$ Subjects        <string> "Rock music 2011 2020", "Perry Matthew 1969 2023, Fri~
$ Publisher       <string> "Sup Pop,", "Thorndike Press, a part of Gale, a Cenga~
$ PublicationYear <string> "[2012]", "2022.", "[2017]", "2018", "2022.", "[2017]~
Call `print()` for query details
```

```r
# 4. COLUMN-WISE EXPLORATION (targeted queries)
seattle_csv |>
  summarise(
    total_rows = n(),
    across(where(is.character), ~n_distinct(.x, na.rm = TRUE)),
    across(where(is.numeric), list(min = ~min(.x, na.rm = TRUE),
                                   max = ~max(.x, na.rm = TRUE)))
  ) |>
  collect()
```

```
# A tibble: 1 x 16
  total_rows UsageClass CheckoutType MaterialType  Title   ISBN Creator Subjects
       <int>      <int>        <int>        <int>  <int>  <int>   <int>    <int>
1   47854892          2            5           71 1.98e6 613827  437607   949390
# i 8 more variables: Publisher <int>, PublicationYear <int>,
#   CheckoutYear_min <int>, CheckoutYear_max <int>, CheckoutMonth_min <int>,
#   CheckoutMonth_max <int>, Checkouts_min <int>, Checkouts_max <int>
```

```r
##instead save so you don't have to re-processing the massive dataset to explore later!:

# Save the summary (this might take a few minutes for 10GB)
data_summary <- seattle_csv |>
  summarise(
    total_rows = n(),
    across(where(is.character), ~n_distinct(.x, na.rm = TRUE)),
    across(where(is.numeric), list(min = ~min(.x, na.rm = TRUE),
                                   max = ~max(.x, na.rm = TRUE)))
  ) |>
  collect()

# Then print when it's done
data_summary
```

```
# A tibble: 1 x 16
  total_rows UsageClass CheckoutType MaterialType  Title   ISBN Creator Subjects
       <int>      <int>        <int>        <int>  <int>  <int>   <int>    <int>
1   47854892          2            5           71 1.98e6 613827  437607   949390
# i 8 more variables: Publisher <int>, PublicationYear <int>,
#   CheckoutYear_min <int>, CheckoutYear_max <int>, CheckoutMonth_min <int>,
#   CheckoutMonth_max <int>, Checkouts_min <int>, Checkouts_max <int>
# 5. CHECK FOR MISSING DATA
missing_nas <- seattle_csv |>
  summarise(across(everything(), ~sum(is.na(.x)))) |>
  collect()

#inspect
missing_nas
```

```
# A tibble: 1 x 12
  UsageClass CheckoutType MaterialType CheckoutYear CheckoutMonth Checkouts
       <int>        <int>        <int>        <int>         <int>     <int>
1          0            0            0            0             0         0
# i 6 more variables: Title <int>, ISBN <int>, Creator <int>, Subjects <int>,
#   Publisher <int>, PublicationYear <int>
```

```
# Check for missing values in character columns
missing_char <- seattle_csv |>
  summarise(across(where(is.character), ~sum(is.na(.x) | .x == ""))) |>
  collect()

#inspect
missing_char
```

```
# A tibble: 1 x 9
  UsageClass CheckoutType MaterialType Title     ISBN Creator Subjects Publisher
       <int>        <int>        <int> <int>    <int>   <int>    <int>     <int>
1          0            0            0     0 40688095  1.32e7  1801677   9523549
# i 1 more variable: PublicationYear <int>
```

```
# Get a cleaner view with percentages
total_rows <- seattle_csv |>
  summarise(n = n()) |>
  collect() |>
  pull(n)

missing_char |>
  pivot_longer(everything(),
               names_to = "column",
               values_to = "missing_count") |>
  mutate(missing_percentage = round(missing_count / total_rows * 100, 2)) |>
  arrange(desc(missing_count))
```

```
# A tibble: 9 x 3
  column          missing_count missing_percentage
  <chr>                   <int>              <dbl>
1 ISBN                 40688095              85.0
2 Creator              13180278              27.5
3 PublicationYear       9844863              20.6
4 Publisher             9523549              19.9
5 Subjects              1801677               3.76
6 UsageClass                  0               0
7 CheckoutType                0               0
8 MaterialType                0               0
9 Title                       0               0
```

"Look Ma, No Loading!"

Showing Query Plans with `show_query()`

Before we execute anything, let's see what Arrow plans to do:

```r
query_plan <- seattle_csv |>
  filter(CheckoutYear == 2020) |>
  group_by(MaterialType) |>
  summarise(total_checkouts = sum(Checkouts))  # See what Arrow will do (without doing it!)

query_plan |>
  show_query()  # This shows the execution plan - Arrow is incredibly smart about optimizati
```

Reading the Plan (bottom to top):

Node 0: SourceNode{}

- Start with your CSV file

Node 1: FilterNode{filter=(CheckoutYear == 2020)}

- Filter to only 2020 data (reduces data early!)

Node 2: ProjectNode{projection=["total_checkouts": Checkouts, MaterialType]}

- Select only the columns needed: `Checkouts` (renamed to `total_checkouts`) and `MaterialType`

Node 3: GroupByNode{keys=[“MaterialType”], aggregates=[hash_sum(…)]}

- Group by MaterialType and sum up the checkouts

Node 4: SinkNode{}

- Final output destination

What this shows you:

Arrow is being smart!

- It filters first (reduces data volume)
- Only selects needed columns (reduces memory)
- Does the grouping efficiently with hash operations
- Plans the whole operation before executing

When do you need `collect()`?

The Golden Rule:

- **Arrow operations**: filter, select, group_by, summarise → **no collect() needed**
- **R operations**: ggplot, view(), head(), mathematical operations → **collect() first**

Why This Matters

Without `collect()`, you’re working with an Arrow query object:

```r
# This creates a PLAN, not data
query <- seattle_csv |>
  filter(CheckoutYear == 2023)

class(query)  # "ArrowTabular" - not a data frame!
```

```
[1] "arrow_dplyr_query"
```

```r
# This executes the plan and gives you data
actual_data <- query |>
  collect()
class(actual_data)  # "data.frame" - now you can use it in R!
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

File Format Magic (15 minutes)

CSV vs Parquet: The Speed Revolution

Let's convert our massive CSV to Parquet and see the dramatic improvements:

```r
# Convert the entire dataset efficiently

seattle_csv |>
  write_dataset("useR_memory/data/seattle-library-checkouts-parquet/", format = "parquet")
```

```r
# Compare file sizes
csv_size <- file.size("useR_memory/data/seattle-library-checkouts.csv")

parquet_dir_size <- sum(file.size(list.files("useR_memory/data/seattle-library-checkouts-parq


# Show the comparison
glue("Original CSV size: {csv_size}")
glue("Parquet size: {parquet_size}")
glue("Size reduction: {round(as.numeric(csv_size) / as.numeric(parquet_size), 1)}x smaller!")
```

Best practice for reproducible analysis:

1. Share your **code** on GitHub (always < 1MB)
2. Include **sample data** for testing (< 25MB)
3. **Document** where to get full dataset
4. **Provide** conversion scripts for creating Parquet files

Speed Test: CSV vs Parquet

```r
# Time reading and filtering
csv_time <- system.time({csv_result <- seattle_csv |>
  filter(CheckoutYear == 2020) |>
  collect() })

seattle_parquet <- open_dataset("useR_memory/data/seattle-library-checkouts-parquet/")

parquet_time <- system.time({parquet_result <- seattle_parquet |>
  filter(CheckoutYear == 2020) |>
  collect() })

#check the speed of files

glue("CSV time: {round(csv_time[3], 2)} seconds")
```

```
CSV time: 45.15 seconds
```

```r
glue("Parquet time: {round(parquet_time[3], 2)} seconds")
```

```
Parquet time: 4.57 seconds
```

```r
if (parquet_time[3] > 0) {
  glue("Speed improvement: {round(csv_time[3] / parquet_time[3], 1)}x faster")
}
```

```
Speed improvement: 9.9x faster
```

```r
# Verify we got the same results

glue("CSV result rows: {nrow(csv_result)}")
```

```
CSV result rows: 1721376
```

```r
glue("Parquet result rows: {nrow(parquet_result)}")
```

```
Parquet result rows: 1721376
```

Working with Your Optimized Dataset

```r
#check out glimpse
seattle_parquet |>
  glimpse()
```

```
FileSystemDataset with 1 Parquet file
47,854,892 rows x 12 columns
$ UsageClass      <string> "Physical", "Digital", "Digital", "Digital", "Physica~
$ CheckoutType    <string> "Horizon", "OverDrive", "OverDrive", "OverDrive", "Ho~
$ MaterialType    <string> "VIDEODISC", "EBOOK", "AUDIOBOOK", "EBOOK", "BOOK", "~
$ CheckoutYear     <int64> 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024, 2024,~
$ CheckoutMonth    <int64> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
$ Checkouts        <int64> 9, 1, 3, 2, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1,~
$ Title           <string> "Unlocked / Lionsgate Premiere ; Grindstone Entertain~
$ ISBN            <string> "", "9781626254008", "9781781107966", "9780306826610"~
$ Creator         <string> "", "Raychelle Cassada Lohmann", "J. K. Rowling", "Zo~
$ Subjects        <string> "United States Central Intelligence Agency Drama, Bio~
$ Publisher       <string> "Lions Gate Entertainment,", "New Harbinger Publicati~
$ PublicationYear <string> "[2017]", "2023", "2022", "2022", "[2023]", "[2019]",~
```

```r
# Let's see what columns we have to work with
seattle_parquet |>
  head() |>
  collect()
```

```
# A tibble: 6 x 12
  UsageClass CheckoutType MaterialType CheckoutYear CheckoutMonth Checkouts
  <chr>      <chr>        <chr>               <int>         <int>     <int>
1 Physical   Horizon      VIDEODISC            2024             3         9
2 Digital    OverDrive    EBOOK                2024             3         1
3 Digital    OverDrive    AUDIOBOOK            2024             3         3
4 Digital    OverDrive    EBOOK                2024             3         2
5 Physical   Horizon      BOOK                 2024             3         3
6 Physical   Horizon      VIDEODISC            2024             3         2
# i 6 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,
#   Publisher <chr>, PublicationYear <chr>
```

```r
# Quick exploration - much faster now!
```

Key takeaways:

- Parquet files are typically 3-5x smaller than CSV
- Parquet files are often 5-10x faster to read
- Parquet preserves data types (no more parsing!)

Your First Big Data Pipeline (20 minutes)

Now let's build progressively more complex pipelines, always remembering: **filter early, collect late!**

Exercise 1: Basic Filtering (5 minutes)

Question: "What happened to library usage during the pandemic year of 2020?"

```r
pandemic_data <- seattle_parquet  |>
  filter(CheckoutYear == 2020) |>
  collect()
# Check what we got - from 40+ million rows to...
```

```r
pandemic_data |>
  count(MaterialType, sort = TRUE)
```

```
# A tibble: 43 x 2
   MaterialType                n
   <chr>                   <int>
 1 EBOOK                  772386
 2 BOOK                   467596
 3 AUDIOBOOK              317202
 4 VIDEODISC               88170
 5 SOUNDDISC               67530
 6 REGPRINT                 2109
 7 MUSIC                    2083
 8 VIDEO                    1819
 9 SOUNDDISC, VIDEODISC      652
10 LARGEPRINT                312
# i 33 more rows
```

Turn and talk (2 minutes): "What do you notice about the material types during 2020? Any surprises?" Why might we be interested in this year?

Key insight: We filtered from 40+ million rows to a manageable subset BEFORE bringing data into memory!

Exercise 2: Group and Summarize (7 minutes)

Question: "Which types of materials were most popular in 2020 - did people shift to digital?"

Build the pipeline: filter → group → summarise → arrange → collect

```r
 material_summary <- seattle_parquet |>
   filter(CheckoutYear == 2020) |>
   group_by(MaterialType) |>
   summarise(total_checkouts = sum(Checkouts),
             avg_checkouts = mean(Checkouts),
             checkout_count = n(),
             .groups = "drop"    ) |>
   arrange(desc(total_checkouts)) |>
   collect()

# Look at the summary
material_summary
```

```
# A tibble: 43 x 4
   MaterialType          total_checkouts avg_checkouts checkout_count
   <chr>                           <int>         <dbl>          <int>
 1 EBOOK                         2793961          3.62         772386
 2 AUDIOBOOK                     1513625          4.77         317202
 3 BOOK                          1241999          2.66         467596
 4 VIDEODISC                      361587          4.10          88170
 5 SOUNDDISC                      116221          1.72          67530
 6 MIXED                            9118        217.               42
 7 REGPRINT                         7573          3.59           2109
 8 VIDEO                            2430          1.34           1819
 9 MUSIC                            2404          1.15           2083
10 SOUNDDISC, VIDEODISC             1049          1.61            652
# i 33 more rows
```

Turn and talk (2 minutes): "What do you notice, did people shift to digital 2020? Any surprises?"

Extension to this exercise below

Exercise 3: Multi-step Pipeline with Visualization (8 minutes)

Question: "How did checkout patterns change month by month during 2020?"

Build the pipeline: filter → group → summarise → collect → plot

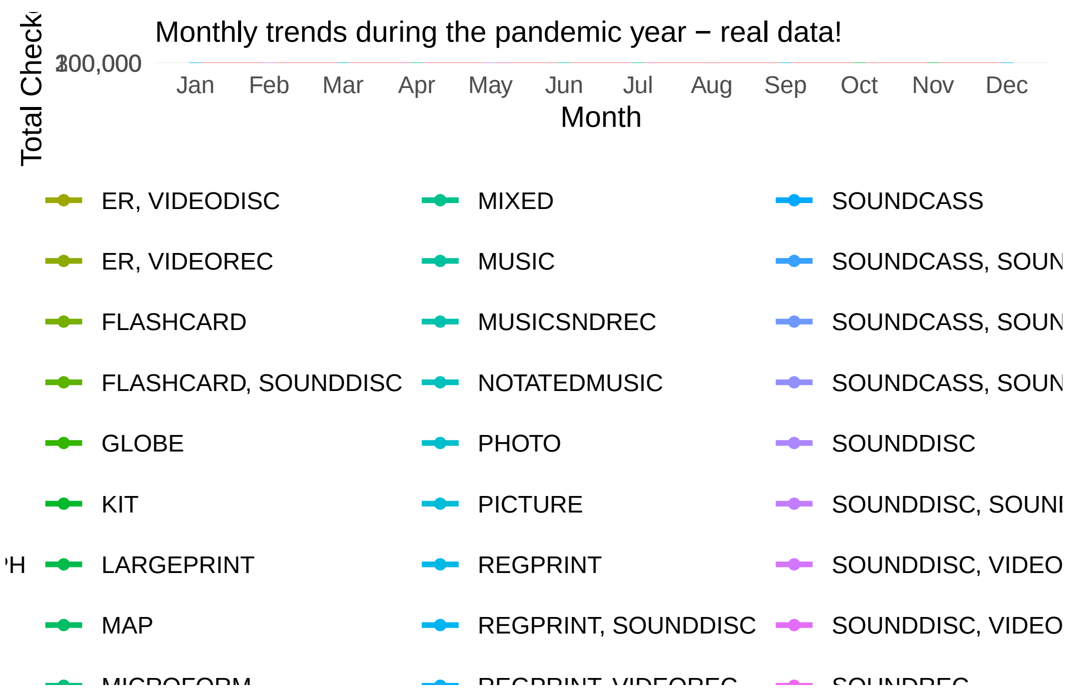
```
monthly_trends <- seattle_parquet |>
  filter(CheckoutYear == 2020) |>
  group_by(CheckoutMonth, MaterialType) |>
  summarise(total_checkouts = sum(Checkouts),.groups = "drop") |>
  collect()

#inspect
monthly_trends
```

```
# A tibble: 264 x 3
   CheckoutMonth MaterialType total_checkouts
           <int> <chr>                  <int>
 1             8 BOOK                    8276
 2             8 EBOOK                 253472
 3             8 AUDIOBOOK             133910
 4             9 EBOOK                 242337
 5            10 EBOOK                 236818
 6             1 BOOK                  343328
 7             1 EBOOK                 198356
 8             5 AUDIOBOOK             128189
 9             5 EBOOK                 255745
10             2 AUDIOBOOK             117082
# i 254 more rows
```

Create Visualization

```
monthly_trends |>
  ggplot(aes(x = CheckoutMonth, y = total_checkouts, color = MaterialType)) +
  geom_line(linewidth = 1) +
  geom_point() +
  scale_x_continuous(breaks = 1:12, labels = month.abb) +
  scale_y_continuous(labels = scales::comma) +
  labs(title = "Seattle Library Checkouts by Material Type (2020)",
       subtitle = "Monthly trends during the pandemic year - real data!",
       x = "Month",
       y = "Total Checkouts",
       color = "Material Type") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Gallery walk (2 minutes): Look at others' results. What patterns do you see?

When was the library closure most visible?

Troubleshooting Checkpoint (5 minutes)

Common Mistakes and How to Fix Them

1. Premature `collect()` - The #1 Error!

```
#   WRONG: Collecting too early
seattle_parquet |>
  collect() |>  # Brings ALL data into memory first!
  filter(CheckoutYear == 2020)   # Then filters - too late!
```

```
# A tibble: 1,721,376 x 12
   UsageClass CheckoutType MaterialType CheckoutYear CheckoutMonth Checkouts
   <chr>      <chr>        <chr>               <int>         <int>     <int>
 1 Physical   Horizon      BOOK                 2020             8         1
 2 Digital    OverDrive    EBOOK                2020             8         1
 3 Digital    OverDrive    EBOOK                2020             8         5
 4 Digital    OverDrive    EBOOK                2020             8         1
 5 Digital    OverDrive    EBOOK                2020             8         1
 6 Digital    OverDrive    EBOOK                2020             8         5
 7 Digital    OverDrive    EBOOK                2020             8         1
 8 Digital    OverDrive    EBOOK                2020             8         2
 9 Digital    OverDrive    AUDIOBOOK            2020             8         1
10 Digital    OverDrive    EBOOK                2020             8        16
# i 1,721,366 more rows
# i 6 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,
#   Publisher <chr>, PublicationYear <chr>
```

```
#  RIGHT: Filter first, collect last se
seattle_parquet |>
  filter(CheckoutYear == 2020) |>  # Filter on disk
  collect() # Only bring filtered data to memory
```

```
# A tibble: 1,721,376 x 12
   UsageClass CheckoutType MaterialType CheckoutYear CheckoutMonth Checkouts
   <chr>      <chr>        <chr>               <int>         <int>     <int>
 1 Physical   Horizon      BOOK                 2020             8         1
 2 Digital    OverDrive    EBOOK                2020             8         1
 3 Digital    OverDrive    EBOOK                2020             8         5
 4 Digital    OverDrive    EBOOK                2020             8         1
 5 Digital    OverDrive    EBOOK                2020             8         1
 6 Digital    OverDrive    EBOOK                2020             8         5
 7 Digital    OverDrive    EBOOK                2020             8         1
 8 Digital    OverDrive    EBOOK                2020             8         2
 9 Digital    OverDrive    AUDIOBOOK            2020             8         1
10 Digital    OverDrive    EBOOK                2020             8        16
# i 1,721,366 more rows
# i 6 more variables: Title <chr>, ISBN <chr>, Creator <chr>, Subjects <chr>,
#   Publisher <chr>, PublicationYear <chr>
```

2. Forgetting `collect()` - Nothing Happens!

```
#  This creates a query plan but doesn't execute it
query_only <- seattle_parquet |>
  filter(CheckoutYear==2020) |>
  summarize(total = sum(Checkouts))

print("Query plan created, but no results yet:")
```

```
[1] "Query plan created, but no results yet:"
```

```
print(class(query_only))
```

```
[1] "arrow_dplyr_query"
```

```
#   Execute the query with collect() a
actual_result <-
  query_only |>
  collect()
print("Now we have actual results:")
```

```
[1] "Now we have actual results:"
```

```
print(actual_result)
```

```
# A tibble: 1 x 1
    total
    <int>
1 6053717
```

3. File Path Issues

```
#  Common file path errors

#open_dataset("wrong_path.csv")  # File not found

#open_dataset("folder_name.csv") # Trying to read folder as file

#   Check if files exist first i

if (file.exists("useR_memory/data/seattle-library-checkouts-parquet")) {
  data <- open_dataset("useR_memory/data/seattle-library-checkouts-parquet/")
  print("  Parquet dataset opened successfully!")
} else if (file.exists("useR_memory/data/seattle-library-checkouts.csv")) {
  data <- open_dataset("useR_memory/data/seattle-library-checkouts.csv", format = "csv")
  print("  CSV dataset opened successfully!")
} else {
  print("  Please check your file path - dataset not found")
}
```

```
[1] "  Parquet dataset opened successfully!"
```

Arrow Choose-Your-Own Challenge (15 minutes)

Work in pairs. Choose the challenge that matches your comfort level, then try the stretch goals!

Beginner Challenge 1A: Material Type Analysis

Goal: Build a dplyr pipeline to summarize checkouts by MaterialType across all years.

Your mission:

1. Start with the Arrow dataset
2. Group by MaterialType
3. Calculate total checkouts, average checkouts, and count of records
4. Sort by total checkouts (highest first)
5. Don't forget to `collect()`!

Helpful hints for beginners:

- Use `group_by()` to group your data
- Use `summarise()` to calculate statistics
- Use `sum()` for totals, `mean()` for averages, `n()` for counts
- Use `arrange(desc())` to sort from highest to lowest
- End with `collect()` to bring the results into R

```r
# Your code here! Use the real Seattle data

material_analysis <- seattle_parquet |>
  # Step 1: Group by MaterialType
  group_by(MaterialType) |>
  # Step 2: Calculate summaries
  summarise(
    total_checkouts = sum(Checkouts),
    avg_checkouts = mean(Checkouts),
    record_count = n(),
    .groups = "drop"
  ) |>
  # Step 3: Sort by total checkouts
  arrange(desc(total_checkouts)) |>
  # Step 4: Collect the results
  collect()

# Print your results
material_analysis
```

```
# A tibble: 71 x 4
   MaterialType total_checkouts avg_checkouts record_count
   <chr>                  <int>         <dbl>        <int>
 1 BOOK                70786029          2.90     24392796
 2 VIDEODISC           32597357          8.57      3802314
 3 EBOOK               26229167          3.07      8548251
 4 AUDIOBOOK           16126638          4.04      3988282
 5 SOUNDDISC           14958972          3.28      4561763
 6 VIDEOCASS            1501190          3.00       499844
 7 SONG                 1298137          1.34       969118
 8 MIXED                 424055          3.19       132780
 9 MUSIC                 393116          1.38       285313
10 MAGAZINE              389867         42.5          9175
# i 61 more rows
```

Goal: Analyze checkout trends by year to see library usage patterns.

Your mission:

1. Start with the Arrow dataset
2. Group by CheckoutYear
3. Calculate total checkouts and number of unique titles per year
4. Sort by year (oldest first)
5. Don't forget to `collect()`!

Helpful hints for beginners:

- Use `group_by(CheckoutYear)` to group by year
- Use `n_distinct(Title)` to count unique titles
- Use `arrange(CheckoutYear)` to sort chronologically (no `desc()` needed)

```r
# Your code here! Analyze trends over time

yearly_analysis <- seattle_parquet |>
  # Step 1: Group by year
  group_by(CheckoutYear) |>
  # Step 2: Calculate summaries
  summarise(
    total_checkouts = sum(Checkouts),
    unique_titles = n_distinct(Title),
    .groups = "drop"
  ) |>
  # Step 3: Sort by year
  arrange(CheckoutYear) |>
  # Step 4: Collect the results
  collect()

# Print your results
yearly_analysis
```

```
# A tibble: 21 x 3
   CheckoutYear total_checkouts unique_titles
          <int>           <int>         <int>
 1         2005         3798685        318481
 2         2006         6599318        360618
 3         2007         7126627        374243
 4         2008         8438486        394316
 5         2009         9135167        403787
 6         2010         8608966        397475
 7         2011         8321732        444151
 8         2012         8163046        467486
 9         2013         9057096        498268
10         2014         9136081        516796
# i 11 more rows
```

Stretch goals:

- Add a filter for years 2015-2023 only
- Create a simple bar chart of your results
- Calculate what percentage each material type represents

Intermediate: Multi-step Pipeline + Visualization

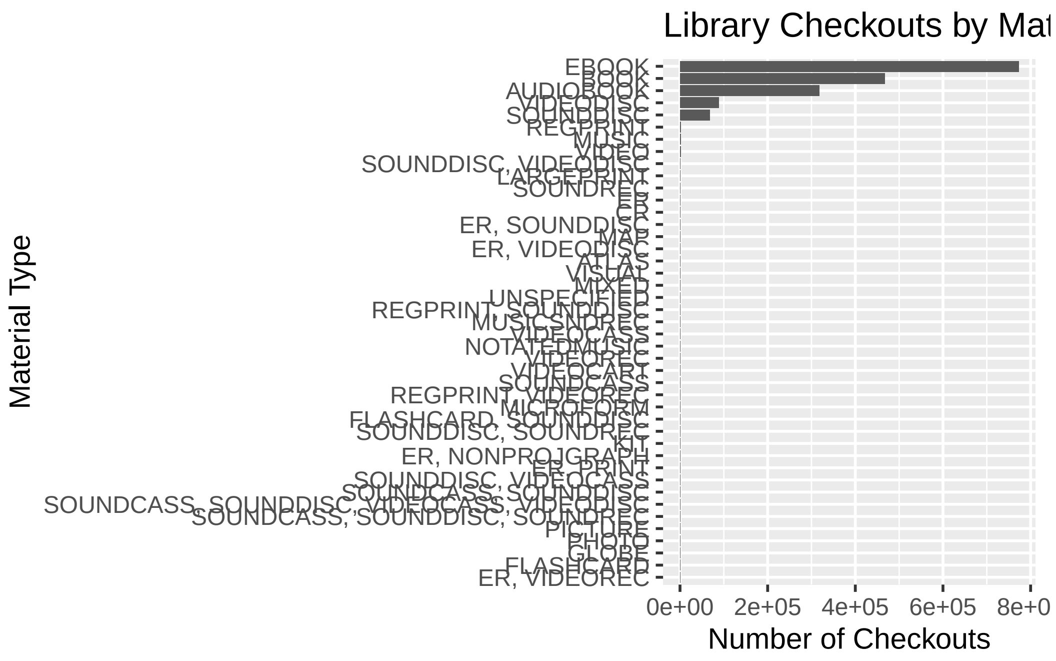
Intermediate Challenge 1A Extension intermediate

Goal: Create a quick visualization to see what is happening

Your mission:

1. Use the 2020 pandemic data we already created
2. Count how many times each MaterialType appears
3. Sort from most to least common
4. Create a horizontal bar chart
5. Add clear labels and title

```r
# Quick bar chart of material types
pandemic_data |>
  count(MaterialType, sort = TRUE) |>
  ggplot(aes(x = reorder(MaterialType, n), y = n)) +
  geom_col() +
  coord_flip() +
  labs(title = "Library Checkouts by Material Type (2020)",
       x = "Material Type", y = "Number of Checkouts")
```



Challenge 1B: Digital vs Physical Comparison

Goal: Categorize materials as digital or physical and compare their usage

Your mission:

1. First explore what material types exist in our data
2. Create logical categories (Digital, Physical, Other)
3. Group by these new categories
4. Calculate totals and counts for each category
5. Sort by total checkouts

```r
# See all the material types we have
material_summary$MaterialType
```

```
 [1] "EBOOK"
 [2] "AUDIOBOOK"
 [3] "BOOK"
 [4] "VIDEODISC"
 [5] "SOUNDDISC"
 [6] "MIXED"
 [7] "REGPRINT"
 [8] "VIDEO"
 [9] "MUSIC"
[10] "SOUNDDISC, VIDEODISC"
[11] "CR"
[12] "LARGEPRINT"
[13] "ER"
[14] "SOUNDREC"
[15] "ER, VIDEODISC"
[16] "MAP"
[17] "ER, SOUNDDISC"
[18] "ATLAS"
[19] "VISUAL"
[20] "UNSPECIFIED"
[21] "REGPRINT, SOUNDDISC"
[22] "VIDEOREC"
[23] "MUSICSNDREC"
[24] "VIDEOCASS"
[25] "ER, NONPROJGRAPH"
[26] "VIDEOCART"
```

```
[27] "REGPRINT, VIDEOREC"
[28] "KIT"
[29] "SOUNDCASS"
[30] "NOTATEDMUSIC"
[31] "SOUNDDISC, SOUNDREC"
[32] "FLASHCARD, SOUNDDISC"
[33] "MICROFORM"
[34] "ER, PRINT"
[35] "SOUNDCASS, SOUNDDISC"
[36] "SOUNDDISC, VIDEOCASS"
[37] "ER, VIDEOREC"
[38] "PHOTO"
[39] "GLOBE"
[40] "FLASHCARD"
[41] "SOUNDCASS, SOUNDDISC, SOUNDREC"
[42] "PICTURE"
[43] "SOUNDCASS, SOUNDDISC, VIDEOCASS, VIDEODISC"
```

Then categorize them:

```r
# Compare digital vs physical
material_summary |>
  mutate(category = case_when(
    MaterialType == "EBOOK" ~ "Digital",
    MaterialType == "AUDIOBOOK" ~ "Digital",
    MaterialType == "BOOK" ~ "Physical",
    str_detect(MaterialType, "VIDEODISC|DVD") ~ "Physical",
    str_detect(MaterialType, "SOUNDDISC|CD") ~ "Physical",
    str_detect(MaterialType, "VIDEO|VIDEOCASS") ~ "Physical",
    str_detect(MaterialType, "MUSIC|SOUND") ~ "Physical",
    MaterialType %in% c("REGPRINT", "LARGEPRINT") ~ "Physical",
    MaterialType == "MAP" ~ "Physical",
    MaterialType == "ATLAS" ~ "Physical",
    TRUE ~ "Other"
  )) |>
  group_by(category) |>
  summarise(
    total_checkouts = sum(total_checkouts),
    material_count = n()
  ) |>
  arrange(desc(total_checkouts))
```

```
# A tibble: 3 x 3
```

```
  category total_checkouts material_count
  <chr>              <int>          <int>
1 Digital          4307586              2
2 Physical         1735392             28
3 Other              10739             13
```

Exercise 2 Extension intermediate Challenge 1

Goal: Analyze checkout trends by month during 2020 (pandemic year)

Your mission:

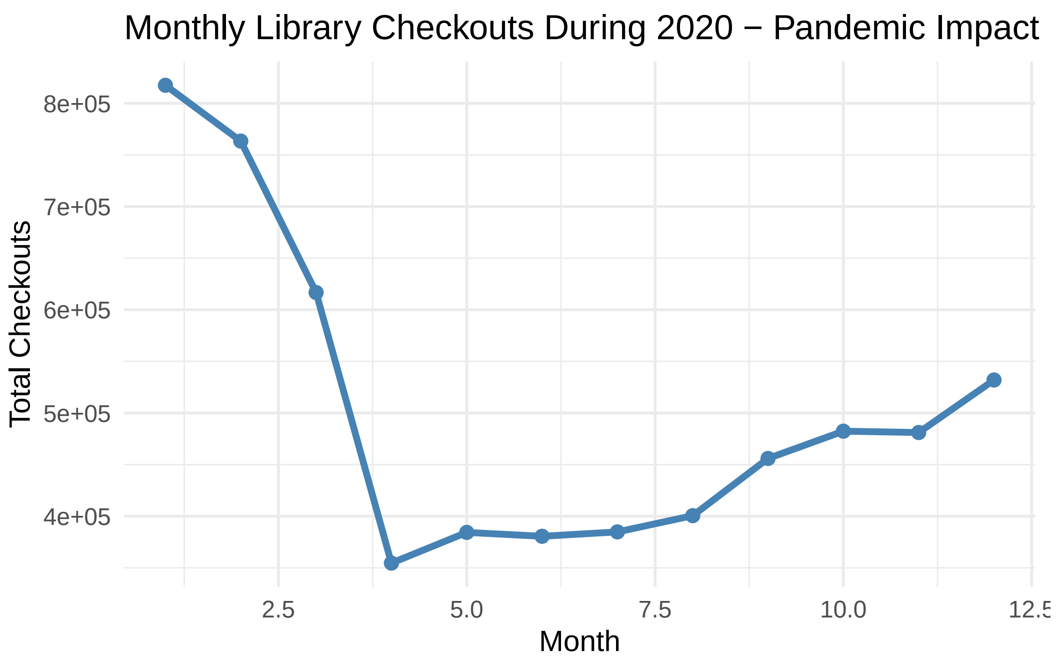
1. Filter to 2020 data only
2. Group by CheckoutMonth
3. Calculate total checkouts per month
4. Create a line chart showing the monthly trend
5. Add a title that mentions “Pandemic Impact”

```
 # Add your pipeline here for monthly 2020 analysis
  monthly_trends <- seattle_parquet |>
  filter(CheckoutYear == 2020) |>
  group_by(CheckoutMonth) |>
  summarise(total_checkouts = sum(Checkouts), .groups = "drop") |>
  collect()



# Create line chart Here
monthly_trends |>
  ggplot(aes(x = CheckoutMonth, y = total_checkouts)) +
  geom_line(size = 1.2, color = "steelblue") +
  geom_point(size = 2, color = "steelblue") +
  labs(
    title = "Monthly Library Checkouts During 2020 - Pandemic Impact",
    x = "Month",
    y = "Total Checkouts"
  ) +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



Stretch goals:

- Add monthly granularity to see seasonal patterns
- Include confidence intervals or trend lines
- Compare pre-pandemic (2015-2019) vs. pandemic (2020-2023) trends

Optional Brain Break

Feeling overwhelmed? It's totally normal! Take a brain break - you can always do these laterr:

- Step away from your computer and stretch
- Grab some water or coffee
- Check out the GitHub examples for inspiration
- Ask a neighbor how they're approaching the challenge

Key Takeaways from Module 2

What You've Accomplished Today

Congratulations! You've just:

- Processed datasets without loading them entirely into memory
- Converted massive files for 10x+ speed improvements
- Used the same dplyr syntax on much larger datasets
- Built multi-step analytical pipelines with lazy evaluation
- Learned to optimize performance with proper `collect()` placement

Essential Arrow Patterns to Remember

1. **The Lazy Evaluation Pattern**:

   ```
   open_dataset("file") |>     # Create view
     filter() |>               # Filter on disk
     group_by() |>             # Group on disk
     summarise() |>            # Aggregate on disk
     collect()                 # Bring results to memory
   ```

2. **The File Conversion Pattern**:

   ```
   open_dataset("data.csv") |>
     write_dataset("data_parquet/", format = "parquet")
   ```

3. **What's under the hood**

   ```
   query |>
     show_query()        # See what will happen

   query |>
     collect()           # Actually execute
   ```

Coming Up: Module 3 - DuckDB Superpowers

In our next module, we'll add even more power to your toolkit:

- **Complex joins** across multiple datasets
- **Window functions** for advanced analytics
- **SQL integration** that feels like dplyr
- **When to choose** Arrow vs. DuckDB for different tasks

The promise continues: Same familiar syntax, even more analytical power!

Quick Self-Assessment

Before we move on, take 30 seconds to reflect:

I feel confident that I can:

- Explain the difference between "loading" and "viewing" data
- Use `open_dataset()` instead of `read_csv()` for large files
- Place `collect()` in the right spot in my pipelines
- Convert CSV files to Parquet format
- Build multi-step Arrow pipelines

Please write on a sticky note if you'd like more practice with:

- Understanding when to use `collect()`
- File format conversions
- Complex pipeline building
- Performance optimization
- Troubleshooting errors

Ready to add database superpowers to your toolkit? Let's continue to Module 3: DuckDB!

NSF Acknowledgement: This material is based upon work supported by the National Science Foundation under Grant #DGE-2222148. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.