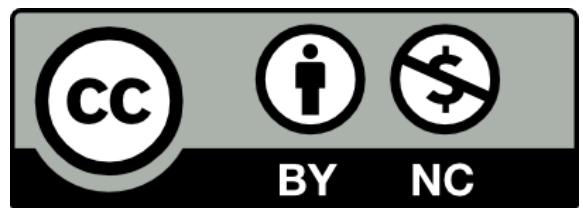
Business 4720 - Class 5

Data Management in R using Tidyverse

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
jevermann@mun.ca



MEMORIAL UNIVERSITY

This Class

What You Will Learn:

- Introduction to R
- Introduction to the Tidyverse set of packages
  - ”Tidy” data
  - Manipulating & cleaning data
  - Joining data
  - Summarizing and reporting data
- Data cleaning

MEMORIAL UNIVERSITY

Intro to R

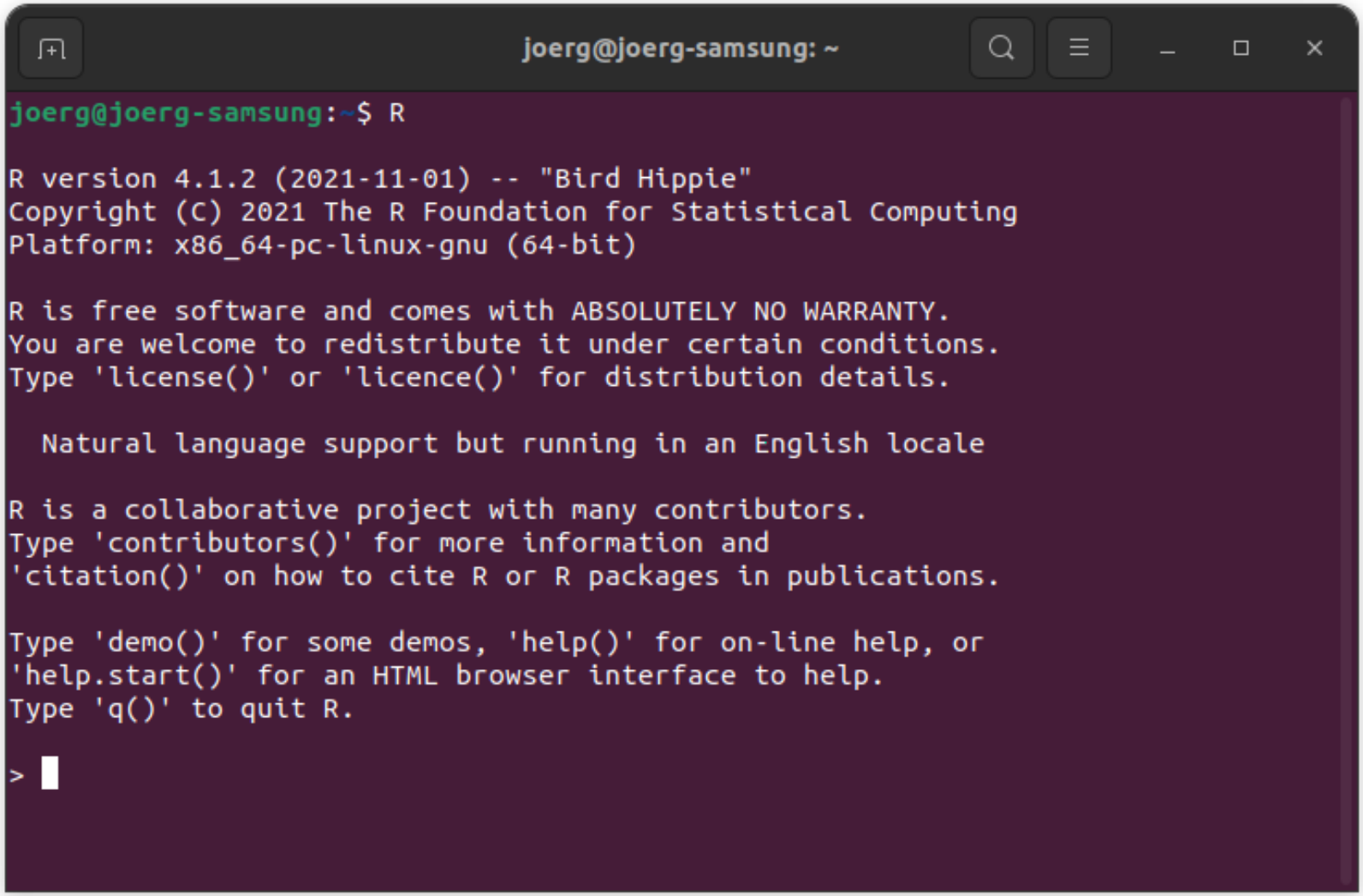
What is R?

- System for statistical analyses
- Created in 1993
- R programming language
- Open-source, cross-platform
- Widely used, popular
- Extensible, thousands of packages
- Scripting and programming

Intro Tutorial: `https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf`

MEMORIAL UNIVERSITY

R



MEMORIAL UNIVERSITY

Tips for Working with R

To make using R more efficient, consider doing the following:

- Use the up-arrow key to retrieve earlier commands.
- The `history()` function shows your command history.
- Use a notepad app to assemble and edit your commands easily, then copy/paste to R for execution.
- Use a notepad app for your results, copy/paste from R.
- The Ubuntu terminal uses SHIFT-CTRL-X, SHIFT-CTRL-C, SHIFT-CTRL-V for cut/copy/paste.
- Use multiple terminal and R windows (e.g. one for executing commands, one for reading help documentation or for listing files).
- Don't update packages in the middle of a project.
- Ensure you have a *repeatable, automatable script* for your entire data analysis at the end of a project.

MEMORIAL UNIVERSITY

My First R Commands

R can do math:

```
> 1+1
[1] 2
```

R knows variables:

```
> a <- 3
> b <- 2
> print(a * b)
[1] 6
```

Note: You can also assign using "="

MEMORIAL UNIVERSITY

R knows if its not a number:

```
> 2 / 0
[1] Inf
> 0 / 0
[1] NaN
```

R Basics [cont'd]

R knows boolean logic:

```
> TRUE & FALSE
FALSE
> TRUE | FALSE
TRUE
```

Tip: TRUE and FALSE can be abbreviated T and F

R know character strings:

```
label1 = 'I Love R'
label2 = 'and BUSI 4760'
paste(label1, label2, sep=' ')
strsplit('Hello World! My first string', ' ')
```

Note: Strings may be enclosed in double or single quotes.

MEMORIAL UNIVERSITY

R Basics [cont'd]

R types and type coercion:

```
# Check whether it is numeric
is.numeric(2)
# Check whether it is an integer number
is.integer(3.14)
# Make it an integer
as.integer(3.14)
# Make it a character string
as.character(3.14)
# Make it a character string
as.character(TRUE)
# Make it a number
as.numeric('3.1415')
```

R Environment

R can manipulate its objects (”workspace”)

```
# Show objects in workspace
> ls()
[1] "a"           "b"           "v"
# Remove one object
> rm(v)
> ls()
[1] "a"           "b"
```

R Environment [cont'd]

R can help:

```
help()
help(lm)
?lm
??lm
help.start()
```

R Environment [cont'd]

R has a working directory in which it stores and reads its data:

```
# Get the working directory
getwd()
# Set the working directory
setwd('DataSets')
getwd()
# List files in working directory
list.files()
```

Tip: It is often more convenient to change the working directory in the terminal, prior to invoking.

MEMORIAL UNIVERSITY

R Environment [cont'd]

R uses packages:

```
# Install the tidyverse library
install.packages('tidyverse')
# Load/attach the tidyverse library
library(tidyverse)
# List all installed packages
installed.packages()
```

R can read command files:

```
source('MyFirstScript.R')
```

Note: Sourcing a script turns off auto-printing, you must use explicit `print()` commands

MEMORIAL UNIVERSITY

R Environment [cont'd]

Say goodbye to R:

```
quit()
```

R stores its **workspace** in each directory in a file called ".RData" and will read it when restarted. R stores its **command history** in each directory in a file called ".Rhsitory" and will read it when restarted.

R Vectors

All elements have the same type:

```
> v <- c(1, 'a', TRUE)
> v
[1] "1"    "a"    "TRUE"
```

R does vector math:

```
> v <- c(1, 2, 3, 4)
> v
[1] 1 2 3 4
> v*3
[1]  3  6  9 12
```

Note: No print statement needed in interactive mode

```
# Generate a sequence
s <- seq(0, 6, by=.5)
print(s)
# Repeat a value
r <- rep(3.5, 5)
print(r)
```

R Vectors [cont'd]

More basic functions on vectors:

```
length(v)
max(v)
min(v)
sqrt(v)
var(v)
sd(v)
# Vectors get flattened
vv <- c(v, c(7, 8, 9), v)
print(vv)
```

R Vectors [cont'd]

Vector indexing:

```
vv < 5
vv[vv < 5]
vv[vv < 5] <- vv[vv < 5] + 5
# Indexing is inclusive
vv[3:7]
# Exclude some elements
vv[-(3:7)]
```

Important: R indexes start at 1!

MEMORIAL UNIVERSITY

R Missing Values

R knows NA:

```
# Introduce a missing value
v[3] <- NA
# Missing value arithmetic
v*3
# Testing for missing values
is.na(v)
# Missing values in aggregate functions
sum(v)
sum(v, na.rm=TRUE)
```

MEMORIAL UNIVERSITY

R Vectors [cont'd]

Regular Expressions:

```
# Match/find North American phone numbers
> grep('^([0-9]{3})[ -]?[0-9]{3}[ -]?[0-9]{4}$',
    c('709 864 5000', 'abc def 9999', '709-865-5000'))
[1] 1 3
# Match/find Canadian postal codes
> grep('[A-V][0-9][A-V] [0-9][A-V][0-9]',
    c('A0P 1L0', '0AB L2K', 'A0X 1Z0'))
[1] 1
```

Levenshtein distances:

```
# Match/find strings up to Levenshtein distance of 3
> agrep('apple',
    c('apricot', 'banana', 'grape', 'pineapple'),
    max.distance=3)
[1] 1 3 4
```

MEMORIAL UNIVERSITY

R Arrays and Matrices

Arrays and matrices have dimensions:

```
# Default is to fill by column
a <- array(1:20, dim=c(4,5))
a
# Result:
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    1    5    9   13   17
# [2,]    2    6   10   14   18
# [3,]    3    7   11   15   19
# [4,]    4    8   12   16   20
# Indexing is inclusive and starts at 1
a[,2]
a[,2:4]
a[3,2:4]
a[3:1,2:4]
```

MEMORIAL UNIVERSITY

R Arrays and Matrices [cont'd]

- The first dimension is the row, the second is the column
- Initially, the array is created from a range of numbers between 1 and 20, and the `dim` argument specifies the dimensionality.
- R fills arrays by column, unless otherwise specified
- A dimension need not be slided or indexed, as in `a[,2]` or `a[,2:4]` which do not subset the first dimension (rows). The result is that all rows are returned in these examples.
- Reversing the index reverses the result that is returned, as in `a[3:1,2:4]` which reverses the indexing of the first dimension (rows).

MEMORIAL UNIVERSITY

R Arrays and Matrices [cont'd]

```r
b <- matrix(20:1, nrow=5, byrow=T)
b
# Result:
#      [,1] [,2] [,3] [,4]
# [1,]   20   19   18   17
# [2,]   16   15   14   13
# [3,]   12   11   10    9
# [4,]    8    7    6    5
# [5,]    4    3    2    1
# Test if it is a matrix
is.matrix(b)
is.matrix(a)
# Transpose
t(b)
# Bind (combine) by columns
cbind(a, t(b))
# Bind (combine) by rows
rbind(t(a), b)
```

MEMORIAL UNIVERSITY

R Data Frames

```r
# Create a vector of 50 normally distributed random variables
x <- rnorm(50)
# Create another vector with random variables
y <- 2*x + rnorm(50)
# Create a data frame from the two vectors
data <- data.frame(x, y)
#  Get the column names
colnames(data)
# Update the column names
colnames(data) <- c('Pred', 'Crit')
# Get the number of rows and columns
nrow(data)
ncol(data)
# Get the "Pred" column of the data frame
data$Pred
# Print a summary
summary(data)
# Print first and last rows
head(data)
tail(data)
# Calculate the covariance matrix
cov(data)
```

Data Frames [cont'd]

Writing data to CSV:

```
# Write CSV file into current working directory
# Omit the row names
write.csv(data, 'data.csv', row.names=FALSE)
```

Reading data from CSV:

```
# Read the data from the current working directory
new.data <- read.csv('data.csv')
```

MEMORIAL UNIVERSITY

Hands-On Exercise

1. Create an array with 3 columns and 50 rows of random numbers with mean of 2 and standard deviation of 4 (use the `rnorm()` function)
2. Create a dataframe from the array and name the columns as ”A”, ”B”, ”C”
3. ”Clip” the values so that all values lie between -3 and $+7$
4. Summarize the data
5. Print the pairwise covariance matrix of the three columns in the data frame
6. Find the square root of each of the diagonal entries of the covariance matrix, compare this to the standard deviation of 4. Tip: Use the `diag()` function.
7. Save the data frame in a CSV file using your first name as file name (file ending ’.csv’)

MEMORIAL UNIVERSITY

Tidyverse

```
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.3     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.0
✔ ggplot2   3.4.4     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.0
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package to force all conflicts to become errors
>
```

Intro books: `https://r4ds.hadley.nz/`

”Cheat sheets:
`https://posit.co/resources/cheatsheets/`

MEMORIAL UNIVERSITY

Tidyverse Packages

dplyr	Manipulate data
forcats	Work with categorical variables (factors)
ggplot2	Grammar of Graphics
lubridate	Date and time parsing and arithmetic
purrr	Functional programming
readr	Read files in various formats
stringr	Work with character strings
tibble	A tibble is better than a table
tidyr	Make data tidy

MEMORIAL UNIVERSITY

Example Dataset

- Government of Canada, Open Government Portal
- Fuel Consumption Ratings – Battery-electric vehicles – 2012–2023
- https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64

Column	Data Type
Make	Categorical (string)
Model	Categorical (string)
Year	Numeric
Category	Categorical (string)
City	Numeric
Hwy	Numeric
Comb	Numeric
Range	Numeric

MEMORIAL UNIVERSITY

Read Data

Load Tidyverse library and read data into a ”tibble”

```
# Load library
library(tidyverse)
# Read CSV into a Tibble
data <- read_csv('https://evermann.ca/busi4720/fuel.csv')
# Examine the data
# Dimensions (rows, columns)
dim(data)
# Column names
colnames(data)
# Summary
summary(data)
```

Tibbles are an extension of data frames and provide more capabilities. Tibbles and data frames are automatically converted into each other when required.

MEMORIAL UNIVERSITY

Summary of DPlyr "Verbs"

Basic	
`filter`	filters by row
`select`	selects columns to retain
`mutate`	creates new columns
`rename`	renames columns
`distinct`	finds unique values
`arrange`	sorts data rows
`relocate`	moves data columns
`group_by`	groups data
`summarize`	compute aggregate information
`print`	prints a tibble
Advanced	
`nest`	nests data, tibbles in tibbles
`full_join`	Joins tibbles (also `outer join`, `left_join`, `inner_join`, `right_join`)

MEMORIAL UNIVERSITY

Filter()

Filters data based on conditions:

```
# Pipe a data frame or tibble into a filter and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  print()

# Equivalent without pipe and boolean operator in filter
filter(data, Make=='Ford' & Year==2023)
```

Equivalent SQL:

```
SELECT *
   FROM data
   WHERE Make=='Ford' AND Year==2023;
```

MEMORIAL UNIVERSITY

Select()

Selects specified columns:

```
# Pipe a data frame or tibble into a filter,
# select specific columns and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  select(Model, Category, Range) |>
  print()
```

Equivalent SQL:

```
SELECT Model, Category, Range
   FROM data
   WHERE Make=='Ford' AND Year==2023;
```

MEMORIAL UNIVERSITY

Mutate()

Creates new calculated columns:

```
# Pipe a data frame or tibble into a filter,
# create a new calculated column,
# select specific columns and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  mutate(HwyRange = Range * Comb / Hwy) |>
  select(Model, Category, Range, HwyRange) |>
  print()
```

Equivalent SQL:

```
SELECT Model, Category, Range, (Range*Comb)/Hwy AS HwyRange
   FROM data
   WHERE Make=='Ford' AND Year==2023;
```

MEMORIAL UNIVERSITY

Rename()

Renames columns:

```
# Pipe a data frame or tibble into a filter,
# create new calculated columns,
# rename an existing column,
# select specific columns and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  mutate(HwyRange = Range * Comb / Hwy) |>
  mutate(CityRange = Range * Comb / City) |>
  rename(CombRange = Range) |>
  select(Model, Category, CombRange, CityRange, HwyRange) |>
  print()
```

Equivalent SQL:

```
SELECT Model, Category,
       Range AS CombRange,
       (Range*Comb)/Hwy AS HwyRange,
       (Range*Comb)/City As CityRange
   FROM data
   WHERE Make=='Ford' AND Year==2023;
```

Distinct()

Returns unique values:

```
# Pipe a data frame or tibble into a filter,
# select distinct value combinations and print results
data |>
  distinct(Make, Model) |>
  print()
```

Equivalent SQL:

```
SELECT DISTINCT Make, Model
  FROM data;
```

MEMORIAL UNIVERSITY

Arrange()

Sorts/orders data by value:

```
# Pipe a data frame or tibble into a filter,
# select specific columns,
# order the data and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  select(Model, Category, Range) |>
  arrange(Category, desc(Range)) |>
  print()
```

Equivalent SQL:

```
SELECT Model, Category, Range
   FROM data
   WHERE Make=='Ford' AND Year==2023
   ORDER BY Category ASC, Range DESC;
```

MEMORIAL UNIVERSITY

Relocate()

Reorders columns:

```
# Pipe a data frame or tibble into a filter,
# select specific columns,
# order the data,
# move columns and print results
data |>
  filter(Make=='Ford', Year==2023) |>
  select(Model, Category, Range) |>
  arrange(Category, desc(Range)) |>
  relocate(Category, Range) |>
  print()
```

Equivalent SQL:

```
SELECT Category, Range, Model
   FROM data
   WHERE Make=='Ford' AND Year==2023
   ORDER BY Category ASC, Range DESC;
```

MEMORIAL UNIVERSITY

Group_by() and summarize()

Groups data and calculates summary values for each group:

```
# Pipe a data frame or tibble into a filter,
# group the data
# summarize the data,
# filter the summary information,
# order the data,
# relocate columns and print results
data |>
  filter(Year==2023) |>
  group_by(Make, Category) |>
  summarize(meanCity = mean(City),
            meanHwy = mean(Hwy),
            meanComb = mean(Comb),
            maxRange = max(Range),
            nVehicle = n()) |>
  filter(nVehicle > 1) |>
  arrange(Category, meanComb) |>
  relocate(Category, meanComb) |>
  print()
```

MEMORIAL UNIVERSITY

Group_by() and summarize()

Equivalent SQL:

```sql
SELECT Category,
       AVG(Comb) AS meanComb,
       Make,
       AVG(City) AS meanCity,
       AVG(Hwy) AS meanHwy,
       MAX(Range) AS maxRange,
       COUNT(*) AS nVehicle
   FROM data
   WHERE Year==2023
   GROUP BY Make, Category
   HAVING COUNT(*) > 1
   ORDER BY Category ASC, meanComb ASC;
```

MEMORIAL UNIVERSITY

Pagila Database in R

Read data into tibbles:

```
rentals <- read_csv('http://evermann.ca/busi4720/rentals.csv')
actors <-
  read_csv('https://evermann.ca/busi4720/actors.categories.csv')
addresses <-
  read_csv('https://evermann.ca/busi4720/addresses.csv')
```

Fix the column datatypes:

```
attach(rentals)
rating <- as.factor(rating)
language <- as.factor(language)
customer_address <- as.integer(customer_address)
customer_store <- as.integer(customer_store)
rental_staff <- as.integer(rental_staff)
payment_staff <- as.integer(payment_staff)
rental_duration <- as.integer(rental_duration)
detach(rentals)

addresses$phone <- as.character(addresses$phone)
```

Pagila Database in R [cont'd]

Examine the NA's:

```
rentals |>
  filter(if_any(everything(), is.na)) |>
  select(last_name, rental_date, return_date,
         title, amount) |>
  print(n=Inf, width=Inf)
```

Interpretation:

- Some films have not been rented
- Some rentals have not been returned

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Find all films and the actors that appeared in them, ordered by film category and year, for those films that are rated PG:

```
rentals |>
    full_join(actors,
        by='title',
        suffix=c('_customer', '_actor'),
        relationship='many-to-many') |>
    filter(rating == 'PG') |>
    mutate(actor =
        paste(last_name_actor, ', ',
        first_name_actor, sep='')) |>
    rename(year=release_year) |>
    select(actor, title, category, year) |>
    distinct(actor, title, category, year) |>
    group_by(category, year, title) |>
    nest() |>
    arrange(category, year, title) |>
    relocate(category, year, title) |>
    print(n=Inf, width=Inf)
```

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Find the most popular actors in the rentals in each city:

```
rentals |>
    inner_join(addresses,
        by=c('customer_address'='address_id')) |>
    inner_join(actors,
        by='title',
        suffix=c('_customer', '_actor'),
        relationship='many-to-many') |>
    mutate(actor =
        paste(last_name_actor, ', ',
        first_name_actor, sep='')) |>
    group_by(city, actor) |>
    summarize(count=n()) |>
    mutate(ranking = min_rank(desc(count))) |>
    filter(ranking < 4) |>
    arrange(city, ranking, actor) |>
    print(n=25)
```

Note: Use `rank()` to break ties, `dense_rank()` for no gaps

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Find the customers who spend the most on rentals, with their phone numbers and cities, and the number of rentals with the higest total rental payments for each category grouped by rental duration.

```
full_data <-
   rentals |>
     inner_join(addresses,
       by=c('customer_address'='address_id')) |>
     inner_join(actors,
       by='title',
       suffix=c('_customer', '_actor'),
       relationship='many-to-many')
```

Pagila Database in R [cont'd]

```
full_data |>
   mutate(customer=
     paste(first_name_customer, last_name_customer)) |>
   select(customer, amount, rental_duration,
          category, phone, city) |>
   group_by(category, rental_duration, customer ) |>
   mutate(payments=sum(amount), num_rentals=n()) |>
   select(-amount) |>
   group_by(category, rental_duration) |>
   mutate(ranking = min_rank(desc(payments))) |>
   slice(which.min(ranking)) |>
   print(n=Inf, width=Inf)
```

- No `summarize()`
- "Negative" `select()`
- Multiple `group_by()`
- Uses `slice()`

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Get the total rental revenue, number of rentals, and the mean and standard deviation of the rental amounts for each country.

```
full_data |>
  group_by(country) |>
  summarize(revenue=sum(amount),
            numrentals=n(),
            mean_amount=mean(amount),
            sd_amount=sd(amount)) |>
  arrange(desc(mean_amount),
          desc(revenue)) |>
  print(n=Inf, width=Inf)
```

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Get the top 5 and the bottom 5 grossing customers for each quarter.

```
full_data |>
  mutate(customer=
   paste(first_name_customer,last_name_customer)) |>
  mutate(q=
   as.character(quarter(rental_date, with_year=T))) |>
  select(customer, q, amount, rental_date) |>
  group_by(q, customer) |>
  mutate(payments=sum(amount)) |>
  select(-amount) |>
  distinct(customer, q, payments) |>
  group_by(q) |>
  mutate(rank_top = min_rank(desc(payments))) |>
  mutate(rank_bot = min_rank(payments)) |>
  filter(rank_top < 6 | rank_bot < 6) |>
```

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Continued from previous slide . . .

```
arrange(q, desc(payments)) |>
relocate(q, customer, payments,
             rank_top, rank_bot) |>
print(n=Inf, width=Inf)
```

- No `summarize()`
- Uses `quarter()` function from package `lubridate`
- Uses `filter()` instead of slice `slice()`

MEMORIAL UNIVERSITY

Pagila Database in R [cont'd]

Find the set of film titles by rental customer and the total number rentals for each customer

```
full_data |>
  mutate(customer=
    paste(first_name_customer,last_name_customer)) |>
  select(customer, title) |>
  nest(titles=title) |>
  rowwise() |>
  mutate(rentals=nrow(titles)) |>
  mutate(unique_titles=list(distinct(titles))) |>
  select(-titles) |>
  arrange(customer)
```

- Work with nested data using `nest` and `rowwise`

MEMORIAL UNIVERSITY

Hands-On Exercises

1. Find all films with a rating of 'PG'
2. List all customers who live in Canada (with their address)
3. Find the average *actual* rental duration for all films
   - This requires date arithmetic, use the `lubridate` package
4. Find the average overdue time for each customer
   - This requires date arithmetic, use the `lubridate` package
5. List all films that have never been rented
6. List the names of actors who have played in more than 15 films

MEMORIAL UNIVERSITY

R knows SQL

The `sqldf` Package

- Set up an in-memory SQLite database (or use existing database connection)
- Move dataframes to database tables
- Run SQL query against database
- Move result set to R dataframe
- Tear down the in-memory database (optional)

Example

```
library(sqldf)
result_df <-
   sqldf('select distinct(title) from full_data')
```

MEMORIAL UNIVERSITY

SQL Databases versus R/Tidyverse

Consider:

- **Size of data**: R is memory limited, RDBMS scale massively larger
- **Access speed**: RDBMS have sophisticated indexes and query planners
- **Currency**: Operational system RDBMS has live data
- **Transactions**: RDBMS ensure consistent views of data across multi-user, concurrent updates
- **Impact**: Queries impact transaction processing (updates of data) performance in RDBMS
- **Tools**: R has tools for statistical analysis and visualization, beyond mere reporting

MEMORIAL UNIVERSITY

SQL Databases versus R/Tidyverse [cont'd]

Recommendations:

- Do not "hit" operational RDBMS for heavy-weight or frequent analytics
- Regularly export consistent data from RDBMS
- Use separate in-memory or on-disk RDBMS for analytics (e.g. with `sqldf`) if desired/required
- If size of data is large, consider distributed tools such as Hadoop/Spark

MEMORIAL UNIVERSITY