Problem set 2 - CSCI E-88b Spring 2020

Due date: Friday, March 27 2020, 10pm EST.

Please write down the last 5 digits of your Harvard ID: 70196

Please write down the URL of your Renku project: https://renkulab.io/projects/stephen.t.knapp14/problem2 (https://renkulab.io/projects/stephen.t.knapp14/problem2)

Please indicate who you may have worked with on this problem and which section you worked with someone on. If you did not work with anyone then you can leave this blank: No one

This problem set corresponds to 1/6 of your final grade. It is graded out of 10.

Question 1.0: 1pt

Question 1.1: 3pts

Question 1.2: 3pts

Question 1.3: 3pts

How to save your changes?

You can keep track your progress with the following commands in a terminal:

```
git add *
git commit -m "My commit message"
git push
```

Question 2.0: Keep it private!

If your problem set is not private, your grade will be 0/10.

Question: Please confirm that your project is private. You can edit the markdown cell from `[ ]` to `[x]` to check a box.

Your answer (1.0):

- [X] Yes, my project is private.
- [ ] No, my project is not private.

Question 2.1: Big data and version control

It is good practice not to have large files under git version control. Git LFS is a git extension for versioning large files (https://git-lfs.github.com/ (https://git-lfs.github.com/)). Git LFS is availble with Renku.

> Git Large File Storage (LFS) replaces large files such as audio samples, videos, datasets, and graphics with text pointers inside Git, while storing the file contents on a remote server like GitHub.com or GitHub Enterprise.

In folder `data/medicare` , we have added two publicly available data files about Medicare Provider Utilization and Payment Data.

Question:

2.1.1. How do you interpret the output of the following code snippet?

In [1]:
```
library(readr)
d <- read_csv("../data/medicare/Medicare_Charge_Outpatient_APC28_CY2015_Provid
er.csv")
head(d)
```

```
Parsed with column specification:
cols(
  `version https://git-lfs.github.com/spec/v1` = col_character()
)
```

A tibble: 2 × 1

version https://git-lfs.github.com/spec/v1
<chr>
oid sha256:2f41f66a32c15ab71dd29cf1c89ccc027c9e7ca78434488e621d6052d142cb41
size 5279936

2.1.2. What is the purpose of the following commands (from a Jupyter terminal)?

git lfs fetch git lfs checkout

2.1.3. After having run the commands of 2.1.2. in a terminal, what happens when you rerun the code of 2.1.1.?

In [2]:
```
d <- read_csv("../data/medicare/Medicare_Charge_Outpatient_APC28_CY2015_Provider.csv")
head(d)
```

```
Parsed with column specification:
cols(
  APC = col_character(),
  Provider_ID = col_character(),
  Provider_Name = col_character(),
  Provider_Street_Address = col_character(),
  Provider_City = col_character(),
  Provider_State = col_character(),
  Provider_Zip_Code = col_character(),
  Hospital_Referral_Region = col_character(),
  Outpatient_Services = col_double(),
  Average_Submitted_Charges = col_double(),
  Average_Total_Payments = col_double()
)
```

A tibble: 6 × 11

APC	Provider_ID	Provider_Name	Provider_Street_Address	Provider_City	Provider_State
<chr>	<chr>	<chr>	<chr>	<chr>	<chr>
0012 - Level I Debridement & Destruction	010001	Southeast Alabama Medical Center	1108 Ross Clark Circle	Dothan	AL
0012 - Level I Debridement & Destruction	010005	Marshall Medical Center South	2505 U S Highway 431 North	Boaz	AL
0012 - Level I Debridement & Destruction	010007	Mizell Memorial Hospital	702 N Main St	Opp	AL
0012 - Level I Debridement & Destruction	010008	Crenshaw Community Hospital	101 Hospital Circle	Luverne	AL
0012 - Level I Debridement & Destruction	010016	Shelby Baptist Medical Center	1000 First Street North	Alabaster	AL
0012 - Level I Debridement & Destruction	010024	Jackson Hospital & Clinic Inc	1725 Pine Street	Montgomery	AL

Your answers (2.1):

2.1.1. The first line of code loads the package/library called "readr". According to the package documentation for "readr", "the goal of 'readr' is to proivde a fast and friendly way to read rectangualr data (like 'csv', 'tsv', and'fwf')." This library allows us to use the the "read_csv" function called in the next line.

The "read_csv" function is a special use case of the "read_delim" function from the "readr" library. This function points to the "Medicare_Charge_Outpatient_APC28_CY2015_Provider" file located in the folder "data". This is a comma separate value file. This csv file is turned into a dataframe assigned to the variable called "d".

Note that this excel file does not contain the raw data that you might expect. Instead it contains a hyperlink address and text pointer to the actual data.

The third line: "head(d)", will show, by default, the column headers and the first 6 lines of data. This command is useful for checking the dataframe is imported correctly and for the user to examine how the data frame is structured. Note that because the lfs was not run beforehand, "d" is really just assigned the hyperlink and test pointer located in the .csv file, it is not the actual data. Therefore we will only see the three lines of "data" that the file itself contains. The read_csv command also has a default assuming there are headers in the data, therefore it reads the hyperlink in the first row as a column header and the remaining two rows as the data values.

2.1.2. "git lfs fetch" retreives and downloads the large file storage object for the current ref and default remote. The current ref is not specified therefore the currently checked out ref is used. This means my current project is being referenced as the ref. The remote was unspecified therefore the default will be used. The default remote is the file structure of this Renku project. In summary, this command looks for all LFS objects in my current project and downloads their data.

"git lfs checkout" scans my current project for all LFS objects needed and then pulls the data downloaded in the previous line and places it in the LFS files. In this specific case it changed the .csv file from the previous questions to the actual raw data, whereas before this command was run the .csv file was just the text pointer and hyperlink.

2.1.3. When you rerun the code from the first sub-question, the read_csv command reads in the raw data from the .csv file as a dataframe and assigns that dataframe the name "d". This time the csv file actually contains the real data, not a text pointer and hyperlink.

"Head(d)" then shows the column headers and the first 6 lines of data in "d". Note that the previous "d" from the first time we ran the code was overwritten by the new "d".

Question 2.2: `linmod` methods and utilities

In Module 2, we have defined `print.linmod` to pretty-print objects of class `linmod`.

In [3]:
```r
linreg <- function(x, y) {
  b <- solve(t(x) %*% x) %*% t(x) %*% y
  ## degrees of freedom
  df <- nrow(x) - ncol(x)
  ## returns a list
  list(
    b = b,
    df = df
  )
}

linmod <- function(x, ...)
  UseMethod("linmod")

linmod.default <- function(x, y, ...) {
  x <- as.matrix(x)
  y <- as.numeric(y)
  result <- linreg(x, y)
  result$call <- match.call()
  class(result) <- "linmod"
  return(result)
}

linmod.formula <- function(formula, data = list(), ...) {
  mf <- model.frame(formula = formula, data = data)
  x <- model.matrix(attr(mf, "terms"), data = mf)
  y <- model.response(mf)
  result <- linmod(x, y, ...)
  result$call <- match.call()
  result$formula <- formula
  return(result)
}

print.linmod <- function(x, ...) {
  cat("Call:\n")
  print(x$call)
  cat("\nCoefficients:\n")
  print(x$b)
}
```

We have defined a function call and methods (such as `print`) similar to `lm`:

In [4]:
```
m <- linmod(mpg ~ wt * hp + qsec^2, data = mtcars)
print(m)
```

```
Call:
linmod.formula(formula = mpg ~ wt * hp + qsec^2, data = mtcars)

Coefficients:
                   [,1]
(Intercept) 40.31040985
wt          -8.68151580
hp          -0.10618066
qsec         0.50316325
wt:hp        0.02779133
```

Question

2.2. What is the difference between `model.frame` and `model.matrix` ?

In [5]:
```
library(magrittr) # to use pipes https://magrittr.tidyverse.org/

model.frame(mpg ~ wt * hp + qsec^2, data = mtcars) %>%
  head()
```

A data.frame: 6 × 4

	mpg	wt	hp	qsec
	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	2.620	110	16.46
Mazda RX4 Wag	21.0	2.875	110	17.02
Datsun 710	22.8	2.320	93	18.61
Hornet 4 Drive	21.4	3.215	110	19.44
Hornet Sportabout	18.7	3.440	175	17.02
Valiant	18.1	3.460	105	20.22

In [6]:
```
model.matrix(mpg ~ wt * hp + qsec^2, data = mtcars) %>%
  head()
```

A matrix: 6 × 5 of type dbl

	(Intercept)	wt	hp	qsec	wt:hp
Mazda RX4	1	2.620	110	16.46	288.20
Mazda RX4 Wag	1	2.875	110	17.02	316.25
Datsun 710	1	2.320	93	18.61	215.76
Hornet 4 Drive	1	3.215	110	19.44	353.65
Hornet Sportabout	1	3.440	175	17.02	602.00
Valiant	1	3.460	105	20.22	363.30

Your answer (2.2):

2.2. Model.matrix is the design matrix of the model. In order to obtain predictions, the model matrix must be mulitplied by a vector of coefficients as defined in object "m".

The model matrix holds the values of the explanitory variables for each data point and their interactions. "wt * hp" in the model's formula means we want to use weight and horsepower as separate explanatory variables and also include their interaction affect. Therefore "wt times hp" is also an explanitory value, hence the presence of the wt:hp column. The wt:hp column which consists of a list of "wt times hp" values. Although qsec is squared in the model only the original data is shown.

Model.frame takes the original dataframe "mtcars" and selects the explanitory variables and outcome specified in the model formula only. It then puts those into the model frame. As you can see from the outputs, the model frame includes the the raw values for mpg, wt, hp, and qsec. It does not include an intercept column or different combinations of variables as the model matrix did. The model frame is essentially a data frame that only contains the values needed to fit the model as required by the model's formula. The dataset "mtcars" includes far more variables but those were not included in the model frame because they were not necessary to construct the model.

In summary the model.matrix is a collection of variables and their interactions that can be multiplied by the model's coeffient matrix to obtain predictions. The model.frame is just a set of unaltered predictor variables and the outcome that the model needs to compute.

Question 2.3: Let's retrieve artifacts from CI/CD!

We can use the CI/CD features of GitLab to build and check our `linpkg` package. To do so, it is enough to modify the file `.gitlab-ci.yml` . Lines 1-22 correspond to Renku images and should not be modified.

If you have difficulties finding and modifying `.gitlab-ci.yml` from the Jupyter interface (it's a hidden file because it starts with a `.` symbol), you can use the GitLab file editor, edit your file and save your changes. From your notebook, `git pull` would retrieve the modified file from GitLab to your Renku session.)

The syntax of `.gitlab-ci.yml` is detailed in https://docs.gitlab.com/ee/ci/yaml/ (https://docs.gitlab.com/ee/ci/yaml/). **INDENTATION MATTERS!** so be careful with spaces and tabs!

We can add a `build_tar_gz` **job** for the runners of the CI/CD to build the package by appending the following snippet after line 22 of `.gitlab-ci.yml` :

build_tar_gz: stage: build image: rocker/verse script: - cd dev/linpkg/ - R CMD build .

In a similar way, it is possible to use the CI to check a package, run units tests, etc.

From GitLab, we can monitor the two jobs the CI/CD is running:

 3 jobs

The `build_tar_gz` job creates a tar.gz archive on a runner. It's a CI/CD **artifact** that we could download from GitLab: https://docs.gitlab.com/ee/user/project/pipelines/job_artifacts/ (https://docs.gitlab.com/ee/user/project/pipelines/job_artifacts/)

1. How do you modify the `build_tar_gz` job to generate the tar.gz artifact?
2. How do you download the tar.gz artifact from GitLab?

Your answers (2.3):

2.3.1. The new build_tar_gz job should look like this:

artifacts:

```
paths:

- dev/linpkg/
```

Note that I added three lines to the job at the bottom. "artifacts:" is indented to the same line as "script:" above it.

2.3.2.

You can download the artifact by going to the project's CI/CD on Gitlab. A list of pipelines is displayed for each change that was committed. Once the code above is added and committed, on the righthand side of the pipeline there is now a download button that appears. Click it, and the option, "Download tar_build_gz artifacts" appears. Click that option and the file will download as a zip file.

Self-assessment (not graded)

Questions

1. What do you think I was hoping for you to learn through this homework?
2. Did you find anything particularly challenging?

Your answer

1. I think you wanted us to learn how to use large file structures, a little about different data frame types relating to models and about artifacts for CI/CD.
2. I found the last question the most challenging because Renku was not working properly when I was working on this problem. I tried many different combinations of code modifications and they were all failing. I finally realized it was a technical issue not a coding issue. I tried again the next morning and solved the problem quickly.

Submitting your work

To submit the problem set, export the notebook to HTML and upload the file to Canvas.

`[ File -> Export Notebook As... -> Export Notebook to HTML ]`