Demo 2: Biomedical & Clinical Informatics

Merrimack College DSE6630: Healthcare & Life Sciences Analytics

Katherine S. Geist, PhD

24 April 2024

Contents

Introduction

You have just been brought on as a data science consultant for a patient advocacy watchdog. They have asked you to source information about hospital readmissions, because they would like to flag specific geographic areas or types of hospitals where hospital readmission is unexpectedly high. They have heard that datasets exist to investigate these types of questions, but they are not really sure where to begin.

You remember this one data science class you took - and the dataset from the **Centers for Medicare & Medicaid Services** managed by Medicare.gov. You start poking around the Hospitals dataset and decide it is exactly what you need.

The problem is, this dataset is massive, so you are not only tasked with identifying a reasonable question you can ask from the dataset but managing the cleaning, pre-processing, and exploratory data analysis on the merged datasets you choose to explore.

Question

For **Project 2**, you are going to choose your own question! But for this demo analysis, we are focusing on **pneumonia-related hospital readmissions**.

Why pneumonia-related readmissions?

Hospital readmission rates for patients with pneumonia is almost unbelievably high, with about $\frac{1}{5}$ or 20% of patients hospitalized with pneumonia re-admitted within 30 days (de Alba and Amin, 2014). Further, pneumonia is both a leading cause of death among the elderly as well as a commonly communicated disease in hospitals (as in, people may come into a hospital for a different condition but be readmitted for pneumonia).

Thus, given that pneumonia is the only communicable disease on this list, it is serving as our best proxy for diseases spread person-to-person within hospitals because of lack of availability.

Our research question: Can we predict pneumonia-related hospital readmissions and, more generally, disease transfer within hospitals based on characteristics of hospitals, including their patient-ratings, how much Medicare money they receive, and other metrics that might indicate how well the hospital is doing at diagnostics and disease prevention?

Our hypothesis & prediction: Hospitals are, by their very nature, hot zones for increased spread of disease, including pneumonia. We will be able to predict pneumonia-related readmissions because of intrinsic or extrinsic factors that affect the quality of care, and thus the likelihood of disease treatment or transmission, because these factors impact pneumonia treatment and spread. We specifically predict that factors indicating lower quality-of-care will predict higher than average rates of pneumonia-related hospital readmissions.

Objective

Our objective for the patient advocacy watchdog is clean, merge, explore, and predict which hospitals tend to have higher-than-average rates of pneumonia-related readmissions. This is to enable the watchdog group to put together a dashboard or otherwise deploy this information to share with patients so they know how their local hospitals are performing, allowing them to make healthier, more informed choices.

Data & Data Cleaning

Download the Hospitals dataset as a zip directory, and unzip it. Make sure to adjust the name and/or path of the directory in your directory if it is different from mine. Note that mine is living on my Desktop.

WARNING! Do not try to put these data onto GitHub. They are too large to push. If you need help adjusting your path or have questions, please let me know.

Read in the hospital-level data iteratively, while dynamically naming & storing each file as a dataframe.

You may have to dig through the accompanying data dictionary **HOSPITAL__Data__Dictionary.pdf** to get a better handle on what is in the hospital-level files.

Question 1: [**1 point**] Take a look at the chunk below, which performs **dynamic variable assignment** as it reads in each of the files. Your task is to add a comment to each line of code to describe what that line does. > Your answer in the code chunk as comments.

```r
filepath <- "~/Desktop/hospitals_current_data/"
files <- list.files(path = filepath, pattern = "Hospital.csv")

for(f in 1:length(files)) {
    dat <- clean_names(read_csv(paste0(filepath, files[f]),
                                show_col_types = FALSE),
                       case = "upper_camel")
    filename <- gsub(".Hospital\\.csv", "", files[f])
    assign(filename, dat)
}

files <- gsub(".Hospital\\.csv", "", files) %>% data.frame()
names(files) <- "File Name"
```

Now, let's look at a list of the files that we've read in:

Table 1. List of hospital-level data files.

File Name
Complications_and_Deaths
FY_2024_HAC_Reduction_Program
FY_2024_Hospital_Readmissions_Reduction_Program
HCAHPS
Healthcare_Associated_Infections
Maternal_Health
Medicare_Hospital_Spending_Per_Patient
Outpatient_Imaging_Efficiency
Payment_and_Value_of_Care
Timely_and_Effective_Care
Unplanned_Hospital_Visits

Question 2: [1 point] Take a look at some of the dataframes, e.g., `HCAHPS` and `FY_2024_Hospital_Readmissions_Reducti` What do you notice about the format / structure of the data? Does the data have an issue with the dimensionality? **Hint:** Check to see if each row represents a single line of data or not!

> Your answer here.

Tidy the data response variables' dataframe: Fiscal Year 2024 Hospital Readmissions Reduction Program.

We are going to focus on the dataset `FY_2024_Hospital_Readmissions_Reduction_Program` to work on best practices to tidy our data. Although I do not personally choose to follow everything that Hadley Wickham and the **Tidy Data** Movement has espoused, I do think a lot of the best practices are useful principles especially when we are just getting use to working with data. Further, we can leverage a lot of the functions to make our lives MUCH easier! And who doesn't like easier?!

Question 3: [1 point] Take a deeper look at `FY_2024_Hospital_Readmissions_Reduction_Program` at some of the numeric variables in the dataset, e.g., `NumberOfReadmissions`? > Your answer here.

Hint: Columns 5 & 7:10 have problems with the `NA` class.

Question 4: [1 point] Investigate the `replace_with_na_all()` function that is part of the `naniar` package. Use this function to fix the `NA` class in the aberrant columns in `FY_2024_Hospital_Readmissions_Reduction_Program`.

```r
# ?replace_with_na_all
# Your code here.
```

You will notice that this aberrant class of missing value **caused** the numeric columns to be coerced into character columns when they were imported by `read_csv()`. Although we could try using `as.numeric()` by itself, which would assign `NA` to any non-numeric entry in `FY_2024_Hospital_Readmissions_Reduction_Program`, the problem is that we could lose important information that way. Thus, as annoying as this is, we should fix it a little more specifically…

Question 5: [**1 point**] Look more closely at the `NumberOfReadmissions` column in the `FY_2024_Hospital_Readmissions_`
What other issue do you see with this specific column that is coercing it to the character type, and why does it exist?

Hint: Make sure to use the accompanying data dictionary **HOSPITAL__Data__Dictionary.pdf** to also examine **WHY** this problem exists!

> Your answer here.

Question 6: [**1 point**] Investigate the `gsub()` function that is part of base `R` or any other function of your choosing. Use the function you choose to **replace** the problematic text in the `NumberOfReadmissions` column in `FY_2024_Hospital_Readmissions_Reduction_Program` with a **5**.

```
# ?gsub
# Your code here.
```

Question 7: [**1 point**] Speculate as to why I asked you to replace the text with a **5** rather than a **0** or an `NA`. Can you think of any other logical choices for the substitution? Do you agree with my choice? (You're free to disagree, but please defend your decision either way!)

> Your answer here.

Question 8: [**1 point**] Now, imagine an alternative scenario where instead of replacing it with a **5**, I instead asked you to replace it with a randomly sampled integer from 1 to 10. What would this do? Would `gsub()` work here or would I need to execute it a different way, e.g., with an `lapply()` function or a `for()` loop?

> Your answer here.

Question 9: [**3 points**] You probably anticipated this! We just decided that, rather than a **5** we want a randomly sampled integer from 1 to 10.

Hint 1: You might find it easier to read the data in from fresh and, after commenting out your `gsub` from Question 5, pattern match on the problematic text.

Hint 2: This is NOT the only way to do this, but I solved this using `gregexpr()`, `regmatches()`, and `lapply()`.

```
# Your code here.
```

Question 10: [**1 point**] It is finally time to fix those aberrant character columns by making them numeric. However, just using `as.numeric()` would require us to do it over and over on each of the columns in multiple lines of code. Instead, investigate the function `mutate_at()` which will allow you to pass it a list of columns you want to convert with the function you want to use.

```
# ?mutate_at
# Your code here.
```

Let's look more closely at Measure Names. Notice, that what I have done here first is used the `gsub()` function to `mutate()` the `MeasureName` column to remove the text that flanked the medical conditions.

```
FY_2024_Hospital_Readmissions_Reduction_Program <-  FY_2024_Hospital_Readmissions_Reduction_Program %>%
  mutate(MeasureName = gsub("READM-30-", "", MeasureName)) %>%
  mutate(MeasureName = gsub("-HRRP", "", MeasureName))
```

Let's look at what medical conditions the acronyms stand for:

Table 2. Acronyms of medical conditions for which hospital readmissions are tracked.

Acronym	Definition
HIP-KNEE	Total Hip/Knee Arthroplasty
HF	Heart Failure
COPD	Chronic Obstructive Pulmonary Disease
AMI	Acute Myocardial Infarction
CABG	Coronary Artery Bypass Graft
PN	Pneumonia

It's important to understand that these medical conditions are the **ones that Medicaid/Medicare tracks for hospital readmissions**. These are not the only conditions in which a patient might be readmitted, but these are the ones that the agency uses to keep track of hospital performance.

Question 11: [1 point] Investigate the function `pivot_wider()` or `spread()` from the `tidyr` package, which comes bundled with `tidyverse`. Try pivoting the `FY_2024_Hospital_Readmissions_Reduction_Program` dataset wider. Make sure you correctly identify the `names_from` and `values_from` column(s). **Make sure to save it as a separate dataframe called `wideDF`. Use the `dim()` function to prove to me that you successfully pivoted wider.

Note: If at this point you've had any issues with any of the previous problems 2-10, you can load the cleaned readmissions data, `readmissionsClean.Rdata`.

```r
load(file = "readmissionsClean.Rdata")
```

Filtering for specific conditions. Ideally, though, before we'd pivot wider we would decide if we were going to filter for any specific hospital readmission conditions. For your **Project 2**, you will be choosing which medical condition(s) you want to focus on for predicting hospital readmission. However, for the remainder of this demo, we're going to focus on just **pneumonia**.

Ideas you might consider for your project are:

- surgical interventions (`HIP-KNEE`, `CABG`)
- heart-related conditions (`HF`, `AMI`, & `CABG`)
- all conditions (warning: this might be too unwieldy!)
- any other condition(s) that you motivate with a statement or two in support

Therefore, let's filter for just pneumonia-related readmissions which will actually eliminate the need to pivot wider in just this case (because we only chose a single condition).

```r
readmissionsClean <-
  readmissionsClean %>%
  filter(MeasureName == "PN")
```

Dynamic creation of merged dataframe objects: supporting dataframes.

We will often be asked to deal with large, unwieldy datasets and sometimes we need to be able to handle them dynamically. **Dynamic variable assignment**, as I demonstrated when we read in the files, is going to come in handy - but it can sometimes be tricky to accomplish. The whole point, though, is to create code that can handle ANYTHING you throw at it.

In a similar vein, we can also perform **dynamic dataframe creation** when we need to modify and stitch together more than one pre-existing dataframe - ESPECIALLY when we might decide down the line to alter our choices by adding or removing other dataframes.

Writing your own function to create objects dynamically. We are going to take all the cleaning steps we did above that was not specific to the `FY_2024_Hospital_Readmissions_Reduction_Program` dataframe and tidy, pivot, & filter for specific readmission conditions, and iteratively join as many dataframes as needed. The goal is to make something flexible enough that, if you change a set of inputs, it will create a joined table from ANY set of dataframe inputs. There is more than one way to do this; but leveraging some of the existing `tidyverse` and `tidyr` functions will likely help you make this very flexible. Please note, however, that you do not have to use those packages.

I start you out with something I find helpful when joining together dataframes that sometimes have *redundant* columns BUT not every column is in every dataframe. I start by making a dataframe called `hospitalInfo` that contains the pertinent information on each hospital in the dataset, so you can drop those columns from the other dataframes to prevent inflation or issues when you later join the dataframes together. You may want to join each of the other dataframes to `hospitalInfo`.

Also, as I was writing my own function, I found that the `Payment and Value of Care` table must be separated into two tables to join well, at least with how I wrote my function. So, I also do that for you below. I also clean up `HCAHPS` to make it easier to work for the function I wrote, so it may also be useful for you. Note that what I'm doing in these cases is making sure that `MeasureName` is the title of the column **I will want to join on**.

Question 12: [**5 points**] Write a function that will (1) fix any issues identified previously, e.g., issues with the `NA` class or pivoting, if needed; (2) join any number of a list of dataframes you give it; and (3) filters the data for given condition(s) prior to pivoting.

For full credit, make sure that your function can clean up and join together at least three of the dataframes. **Note**: Make sure to join with `readmissionsClean` at the end!

- **Hint 1:** Feel free to drop the `startDate` and `endDate` columns, as well as the `footnote` column. I would even suggest dropping `MeasureId`.
- **Hint 2:** `select(-any_of(c(...)))` can be used to drop any of a list of columns, regardless of whether it exists in every dataframe or not.
- **Hint 3:** If you choose to drop all those extraneous columns, then the column you will need to merge/join on will always be in the second position, `MeasureName`.
- **Hint 4:** `full_join()` in `dplyr` is likely what you will need; you will want to execute the join on `FacilityId`.
- **Hint 5:** Make sure to check for and/or remove duplicate rows.
- **Hint 6:** If you are really stuck, move ahead and I give you a cleaned, merged data set to work with.

Okay, take a stab at it!

```
# Your function here.
```

The full, merged, tidied dataset for pneumonia.

```r
load("pneumoniaFull.Rdata")
```

I ultimately chose to merge **eight** of the datasets together, although I selectively filtered to focus on my question: **Can we predict pneumonia-related hospital readmissions based on factors that might indicate how well the hospital is doing at diagnostics and disease prevention?**

Table 3. List of hospital-level data sets chosen for the pneumonia-related readmissions analysis.

Dataset	Information Used	Rationale
Healthcare_Associated_Infections	MRSA Bacteremia	Hospital-transmitted disease rate
paymentOnly	Mean Medicare payment per patient received by hospital	
Outpatient_Imaging_Efficiency	Abdomen CT Use of Contrast Material	Proxy for imaging ability to diagnose pneumonia appropriately
Complications_and_Deaths	Death rate for pneumonia patients	
Complications_and_Deaths	Perioperative pulmonary embolism or deep vein thrombosis rate	Diagnostic ability of hospital
Complications_and_Deaths	CMS Medicare PSI 90: Patient safety and adverse events composite	General hospital safety
Complications_and_Deaths	Postoperative respiratory failure rate	General ability for respiratory diagnostics
Medicare_Hospital_Spending_Per_Patient	Score	Score rather than $ amount
Timely_and_Effective_Care	Healthcare workers given influenza vaccination	Likelihood of influenza transmission
Timely_and_Effective_Care	% healthcare personnel completed COVID-19 primary vaccination series	

Likelihood of COVID-19 transmission

Timely_and_Effective_Care

Average (median) minutes patients spent in the ED before leaving (lower is better)

Proxy for overall effective diagnosis & treatment

Timely_and_Effective_Care

Left before being seen in ED

Proxy for overall effective diagnosis & treatment

Timely_and_Effective_Care

Venous Thromboembolism Prophylaxis

Diagnostic ability for another critical and silent disease

Timely_and_Effective_Care

ICU Thromboembolism Prophylaxis

Diagnostic ability for another critical and silent disease

Timely_and_Effective_Care

Emergency department volume

Proxy for how overwhelmed the hospital is

Unplanned_Hospital_Visits

Hospital return days for pneumonia patients

HCAHPS

Nurse communication

Overall rating as linear score

HCAHPS

Doctor communication

Overall rating as linear score

HCAHPS

Staff responsiveness

Overall rating as linear score

HCAHPS

Communication about medicines

Overall rating as linear score

HCAHPS

Discharge information

Overall rating as linear score

HCAHPS

Care transition

Overall rating as linear score

HCAHPS

Cleanliness

Overall rating as linear score

HCAHPS

Quietness

Overall rating as linear score

HCAHPS

Overall hospital rating

Overall rating as linear score

HCAHPS

Recommend hospital

Overall rating as linear score

Whew! Notice that this dataset currently has **100 features** and **4,816 rows**. That's definitely too many features, so let's perform some additional feature selection.

First Pass: Pre-processing & Feature Selection

Feature selection can take many forms, from what we've already done (choosing features relevant to our question) to culling non-informative columns to using more machine-guided approaches. We're going to leverage all types here.

Additionally, we are going to perform **encoding** to our categorical variables. It will be important to perform encoding **BEFORE** using automated methods for feature selection.

Culling Features (and more tidying!)

Although we ideally could just include these steps (and perhaps you did!) in our tidying & joining function from Question 12, I wanted to make sure we talk about this explicitly.

Question 13: [**1 point**] Looking through the feature names, you probably notice names like `LowerEstimate_Death rate for pneumonia patients`. We do not need the higher and lower estimates in this case; just the point estimates. Use the `contains()` function to **drop** any columns that contain `LowerEstimate` or `HigherEstimate`. Let's also go ahead and drop any columns containing `Denominator` and `HcahpsAnswerPercent` as well. How many features did you drop? > Your answer here.

```
# Your code here.
```

Note: Please drop `FacilityName`, `TelephoneNumber`, `Address`, & `CityTown` at this stage as well, if they're still in your full dataset.

```
# Your code here.
```

Encoding

Mystery encoding! (Well, it won't be a mystery for long.)

Question 14: [**1 point**] We need to perform encoding on all of the categorical columns, ultimately. However, we will focus first on the columns that begin with '`ComparedToNational_`, e.g., `ComparedToNational_Death rate for pneumonia patients`. What kind of encoding should we perform on these columns? Why? **What other column do we also need to perform this kind of encoding on?**

> Your answer here.

Question 15: [3 points] You probably expected this - attempt to encode those columns the method you selected. It's okay if you can only think of a way to brute-force this for now; I will show you code to help you do it faster after the assignment. :)

- **Hint 1:** You could use the `contains()` or `startsWith()` functions to help you quickly grab those columns so you can make the changes you want to make.
- **Hint 2:** You may want to explore the `grepl()` function or regular expressions, as they can allow us to match in a "fuzzy" way.
- **Hint 3:** You may want to create a temporary dataframe that you execute the changes on and then replace the original columns from there. That way, you don't have to keep reloading the original data if you make mistakes!
- **Hint 3:** For full credit, don't forget to double check that you managed to preserve all the data! The `table()` function is sufficient here.

```
# Your code here.
```

Note: If you struggled with Question 15, load the data below and keep going:

```
load("pneumoniaFullEncoded.Rdata")
```

Frequency encoding state and county.

We also decide that we want a way to try to preserve and analyze the geographic information without causing terrible overfitting. So, we will try to apply **frequency encoding** to those two categorical columns.

Question 16: [1 points] Go through the **frequency encoding** code chunk below and comment each line. What does it do? Make sure to comment why you think the choice was made, if appropriate.

```
cols2encode <- c("State",
               "CountyParish")

temp <- pneumoniaFullEncoded[, names(pneumoniaFullEncoded) %in% cols2encode]

add_freq <- function(data, column_name) {
  frequency_map <- table(data[[column_name]], useNA = "always")
  data[[column_name]] <- frequency_map[match(data[[column_name]],
                                             names(frequency_map))]
  return(data)
}

for (col in names(temp)) {
  temp <- add_freq(temp, col)
}

for (c in 1:length(cols2encode)) {
  pneumoniaFullEncoded[, cols2encode[c]] <- temp[, cols2encode[c]]
}
```

Question 17: [1 point] Use your finally & freshly encoded version of `pneumoniaFullEncoded` to answer this vital question for `State` and `CountyParish` features: are we justified using these features? Why or why not? You will need to write code to answer this question.

Hint 1: Use your original `pneumoniaFull` dataset to compare the features in `pneumoniaFullEncoded`.

Hint 2: **Heavily** duplicated frequency values can make it ill-advised to proceed with a frequency encoded feature. Low to moderate duplication can be kept if there is a strong justification for doing so.

Hint 3: If you decide it is ill-advised for either or both columns, make sure to remove those features from the dataset!

```
# Your code here.
```

Your answer here.

Finishing touches

```
load("pneumoniaAnalyze.Rdata")

pneumoniaAnalyze <- pneumoniaAnalyze %>%
  mutate(across(where(is.character), as.numeric))
```

1. Ensure that every feature is numeric.

```
pneumoniaAnalyze <- pneumoniaAnalyze %>%
  ## arbitrarily chose as the rep as they are identical
  mutate(NumberSurveysCompleted = NumberOfCompletedSurveys_Cleanliness,
         SurveyResponseRate = SurveyResponseRatePercent_Cleanliness/100) %>%       ## Turned into an ac
  select(-contains(c("NumberOfCompletedSurveys_", "SurveyResponseRatePercent_")))   ## drop the others
```

2. Collapse the `NumberOfCompletedSurveys_...` and `SurveyResponseRatePercent_...` features into a single one, as they are identical / redundant: Excellent! Now we are down to just 51 features...

3. Identify the target variable. Our very, very last steps! Your task is to identify the appropriate target variable from the dataset. I have purposefully not gone into great detail about what it would be because choosing the best target is sometimes tricky.

Question 18: [2 points] Use `pneumoniaAnalyze` to calculate an `observed_readmission_rate` as the number of readmissions divided by the number of discharges, multiplied by 100, then drop the two columns used to make this new one. Now, using the data and the data dictionary, try to understand the relationship between `observed_readmission_rate` that we just created, the `PredictedReadmissionRate`, `ExpectedReadmissionRate`, and the `ExcessReadmissionRatio`. What is the relationship? What is the appropriate target to choose and why?

Hint: Think about (or perhaps try to investigate)why the **predicted** rate exists. What would be the (dis)advantage to using the predicted vs. the newly created `observed_readmission_rate`?

Your answer here.

Exploratory Data Analysis

For your final question, you will kick start our EDA by focusing on the target variable that you identified in Question 18. **You will not be penalized for choosing the wrong target as long as you made an attempt to choose and defend a rational choice.**

Question 19: [3 points] Explore the target variable you identified in Question 18 with at least one other variable. Try to push yourself to try a new style of plot; for example, have you ever made a `density` or `violin` plot? These can be excellent choices when exploring the distribution of a target variable against another variable. Note that, if you choose to compare the target against one of the encoded categorical variables, you will need to properly re-label the categories for your plot.

Full points are awarded for professional plots with axis labels, labeled legends (if appropriate), and creative use of multivariable information. **One bonus point will be awarded if you make a faceted plot or otherwise include information from at least 3 variables.**

Need inspiration that comes with code?!? Check out the R Graph Gallery!

```
# Your code here.
```

Lastly, make sure to **interpret** your graphic.

> Your answer here.

References

De Alba, Israel, and Alpesh Amin. 2014. "Pneumonia Readmissions: Risk Factors and Implications." *Ochsner Journal* 14 (4): 649–54.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.