CDS 101 Portfolio

Chris Dano

December 14, 2017

A portfolio of the things I learned in CDS 101

R language

In this particular class, we used the R programming language. We conducted all of our coding in the Rstudio environment, which was a workspace where we can write out code.

R packages

Alone, base R can do many types of calculations and produce plots. It can also run a few statistical tests. There are R *packages* that can be installed using the code **install.packages()** that further enhance R by adding more functions. These can be loaded after installation with the function, **library()**.

R markdown package

Most, if not all of the coding was done on an rmarkdown file. Rmarkdown is a useful package that can create or *knit* a report out of the coding done in the file. We can knit our coding into a .docx or a .pdf file for a neat-looking report.

Tidyverse package

The package that we loaded in almost every assignment was the *tidyverse* package.

This package is actually a collection of packages that provides many functions for exploratory data analysis, which will be discussed soon in more depth. Briefly, this package provided functions to transform datasets that we import, and visualize the data in different ways. It also came with its own sets of data to practice these functions on.

Exploratory Data Analysis

The tidyverse package was useful for exploratory data analysis as I mentioned before. Rather than give a few sentences to define it, I want to describe the process as it was taught in class.

Data import

The first thing we did for every analysis was import some sort of dataset. We usually used the function **read.csv()** for excel data, or we loaded some of the pre-existing datasets using **library()**.

Data cleaning - "tidying"

Many of the datasets that we worked with were not formatted properly for an analysis through tidyverse. A proper dataset, or a *tidy* dataset was a datset that met the three conditions below:

- Each observation had its own row
- Each variable had its own columns

- And each value had its own cell

If those three conditions are met, then the dataset is considered tidy and we can move on to actually analysing the data.

If not, there are several useful functions from the *tidyverse* package for *tidying* the dataset.

The functions we primariliy used were

- **gather()**; which we used for columns which were not variables, but rather categories of a variable.
- **separate()**; which we used to split a column into multiple columns, if the column represented multiple variables.

NA data values

There may be cases where a dataset has NA in a few of its cells. As a data scientist we have to figure out what the NAs mean in the cleaning process so that they do not become problematic during the analysis. We have to determine if they are missing values or values of zero, among other things.

Transformation, Visualization, and Modeling

After tidying the dataset, we can start transforming, visualizing and modeling the data. I have these three together even though they are different parts of the process, because:

1. they are done one after the other as the analysis itself, beginning with tranformation and finishing with modelling.
2. these steps can be viewed as a cycle. In an analysis, you may have to transform data, visualize it, then create some models. Afterwards, you may have to repeat the process if the current results do not reveal anything about the data or the current results are not enough.

Transformation

Sometimes we had to transform the dataset after cleaning it, so that we can visualize the data better or view the data table in a certain way.

We transformed datasets mainly with these functions:

- **filter()**; which allowed us to filter out certain observations/rows in the data based on criteria we specifed.
- **select()**; which allowed us to select certain variables/columns from the data.
- **arrange()**; which allowed us to sort the data in ascending or descending order of some variable.
- **mutate()**; which can be used to edit a column or calculate a new column.
- **group_by()**; this function groups the data by a specified categorical variable.
- **summarize()**; usually used after **group_by()**, this allowed us to make calculations like averaging the data per group, or counting the observations in each group.

Visualization

After we transformed the datasets, we visualized them using one package from *tidyverse*, *ggplot2*. This package of tidyverse provided plethora of plotting functions:

- We used **geom_bar()** or **geom_col()** to make a bar graph.
- We used **geom_point()** to make scatterplots.
- We also used **geom_histogram()** for histograms.

Modeling

We usually looked at the outputs of the plots to determine some sort of relationship between the plotted variables. Graphs alone; however, do not explain the whole story. There were many instances where there was a clear relationship between two variables where if one increased, so did the other. But we used several modeling techniques to determine *correlation*, if a variable changed because of the change in another variable.

Such tests include:

- Single Predictor Linear Regression
- Multiple Predictor Linear Regression
- Logistic Regression

There were also cases where we had data representing a sample of a population, rather than data of the entire population. We ran several statistical tests or hypothesis tests to determine whether some statistic such as the sample mean is representative of the population or close to population mean. We often used confidence intervals to determine if a sample mean can possibly be representative of the population.

Such tests include:

- t-tests; which tests the sample mean against the true mean.
- paired t-tests; which tests the difference in two means for two categories of a variable.
- Spearman test; tests dependence of two numerical variables.
- Chi-squared test; tests dependence of two categorical variables.

Communication

The final step in exploratory data analysis is the communication of our findings to an audience. The instructor for the lab portion, CDS 102, always told us to tell a story. In essence, we want to be able to present our analysis in a way that the audience will understand. Although we will understand our own analysis, the people we present to; policy makers, the general public, business managers, etc, may not understand it.

We usually want to explain the plots we make and the statistical test we run. It would also be nice to have a neat and presentable report. In the case of this portfolio, I used an rmarkdown file.

Example Analysis

Here is a simple example of the process of exploratory analysis. Because communication is key part of of exploratory analysis, I documented the process as well as the outputs.

Loading the necessary packages

First, I want to load all of the necessary packages, such as tidyverse.

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

Data Import

I also want to load a dataset to analyze.

For the mid-term project, My group analyzed a dataset about pneumonia and influenza deaths across the US. I'll use the same dataset for this analysis.

The code below loads the data set into the environment. The chunk will not appear in the report

```
deaths <- read_csv("Deaths_in_122_U.S._cities_-_1962-2016._122_Cities_Mortality_Reporting_System.csv")
```

Data Tidy

Now we move on to cleaning the dataset.

Renaming Variables

The first thing I noticed with this dataset was the naming convention of variables, which I can pull up using the **names()** function.

```
names(deaths)
```

```
##  [1] "Year"                           "WEEK"
##  [3] "Week Ending Date"               "REGION"
##  [5] "State"                          "City"
##  [7] "Pneumonia and Influenza Deaths" "All Deaths"
##  [9] "<1 year (all cause deaths)"     "1-24 years (all cause deaths)"
## [11] "25-44 years"                    "45-64 years (all cause deaths)"
## [13] "65+ years (all cause deaths)"
```

- Some of them are really long and have characters such as parentheses. Some use capitalization and spacing, which can make typing the variables in code difficult and confusing.

With **rename()**, I can rename the variables to names that will be easier to call in R.

```
deaths2 <- rename(deaths, year = Year,
                  week = WEEK,
                  week_end_date = "Week Ending Date",
                  region = REGION,
                  state = State,
                  city = City,
                  pideaths = "Pneumonia and Influenza Deaths",
                  all_deaths = "All Deaths",
                  under_1 = "<1 year (all cause deaths)",
                  "1-24" = "1-24 years (all cause deaths)",
                  "25-44" = "25-44 years",
                  "45-64" = "45-64 years (all cause deaths)",
                  "65+" = "65+ years (all cause deaths)")
```

Finding and determining NAs

The **filter()** function can be used to find NAs in the dataset. Since I want to check every variable, I used a variant of **filter()** called **filter_all()**.

- **any_vars(is.na(.))** looks through every observation and returns observations that have *na* in any of the variables.

- The "." in is.na is a placeholder for the variables. Normally one would specify a variable in **is.na()**.

```
filter_all(deaths2, any_vars(is.na(.)))
```

- I elected to omit the results in the report because the table is too large.

Since there are NAs and they were determined to be missing values, we can remove them using the filter function.

```
deaths3 <- filter_all(deaths2, all_vars(!is.na(.)))
```

- this code snippet filters for rows that have some value other than NA for all of the variables. If even one variable has NA, the row is excluded.

Create Age Group Variable

The biggest concern with the dataset was that age groups were separated into different columns when age group could be in its own column as a variable.

I used **gather()** to bring all of the age groups to one column.

```
deaths4 <- gather(deaths3, under_1:"65+", key = age_group, value = deaths)
```

Analysis

Pneumonia and Influenza deaths versus Year/Decade

One thing I wanted to see was how pneumonia and influenza deaths varied through time.

Transformation

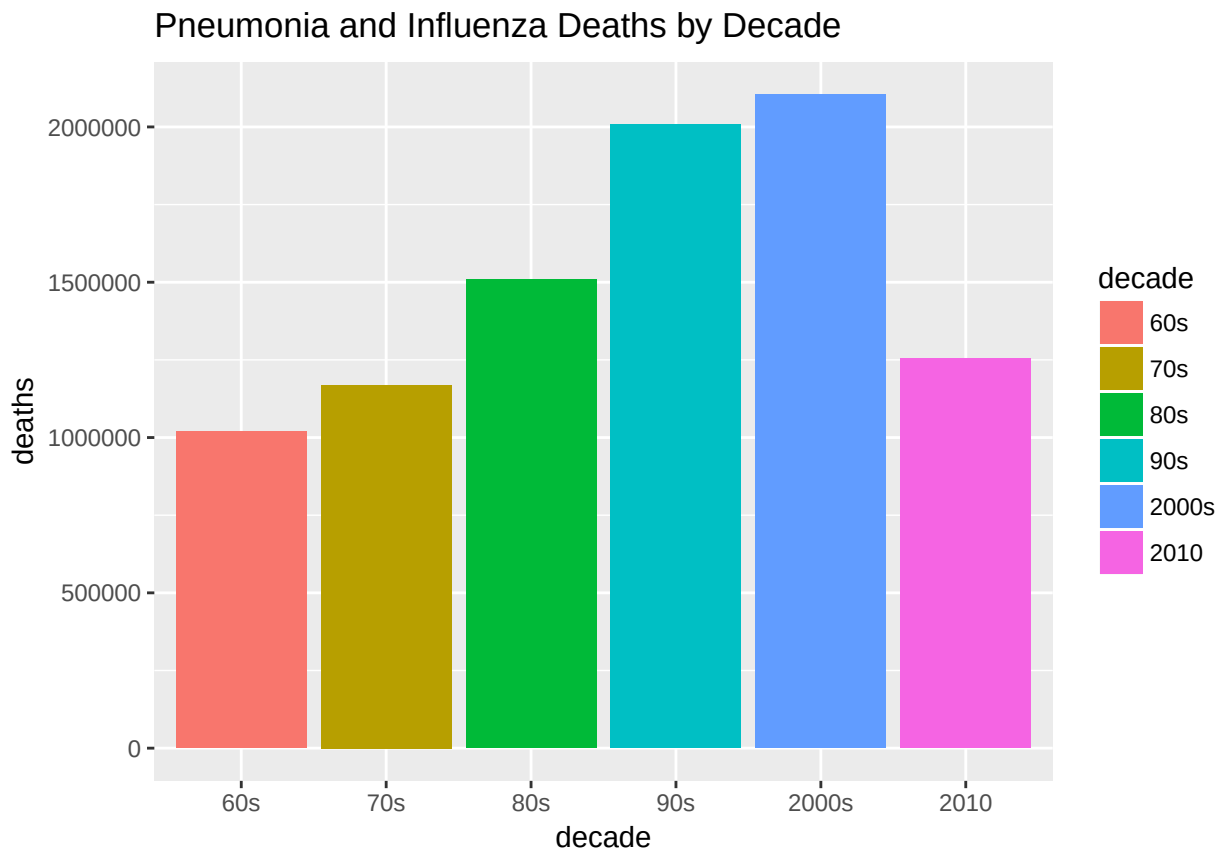
One way to to that is to check the deaths by decades.

The original dataset does not have a "decades" variable; however, we can make a new column with such a variable using the **mutate()** function with the **cut()** function below.

```
deathsdecades <- deaths4 %>%
  mutate(decade = cut(year, breaks = c(-Inf, 1969, 1979, 1989, 1999, 2009, 2019), labels = c("60s", "70s
```

Visualization

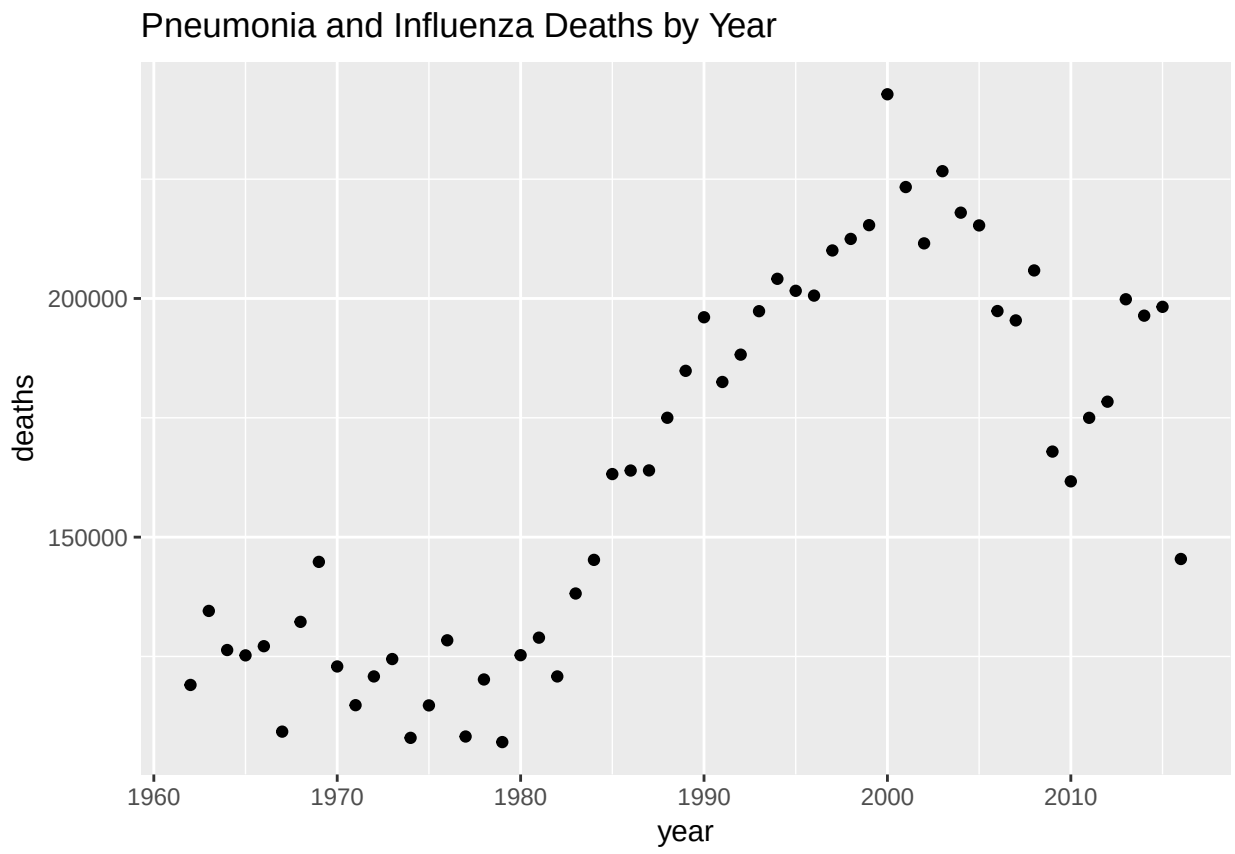
```
deathsdecades %>%
  group_by(decade) %>%
  summarise(deaths = sum(pideaths)) %>%
  ggplot() +
  geom_col(aes(x = decade, y = deaths, fill = decade)) +
  labs(title = "Pneumonia and Influenza Deaths by Decade")
```



As we can see from the output, the number of deaths increases from the 60s to the 2000s, but drops in the next decade.

We can also visualize a relationship between years and pneumonia/influenza deaths rather than decades using a scatterplot. This way, we can see the relationship in a way that a bar graph cannot show us.

```
deathsdecades %>%
  group_by(year) %>%
  summarise(deaths = sum(pideaths)) %>%
  ggplot() +
  geom_point(aes(x = year, y = deaths)) +
  labs(title = "Pneumonia and Influenza Deaths by Year")
```



- With the scatterplot, we can also see some sort of relationship between year and deaths by pneumonia and influenza.

Modeling

Unfortunately, the relationship does not appear to be linear so a linear regression model would not be reasonable.

Spearman test

One of the tests we learned in class, the spearman test, can be used to test the dependence of two numerical variables.

For the **spearman_test()** function, we need the *coin* package.

```
library(coin)
```

```
## Warning: package 'coin' was built under R version 3.4.2
```

```
## Loading required package: survival
```

I first created a new dataset with just the years and the number of deaths due to influenza and pneumonia.

I then used **cor()** to find the correlation coeffiecient for the two variables.

- This coefficient tells me the strength and direction of the relationship. It is a value between -1 and 1. Values closer to 1 or negative 1 as opposed to 0 indicate a strong relationship. A positive value indicates an upward trend while a negative number indicates downward.

```
deathsyear <- deathsdecades %>%
  group_by(year) %>%
```

```
  summarise(deaths = sum(pideaths))
cor(x = deathsyear$year, y = deathsyear$deaths)
```

```
## [1] 0.7732725
```

```
spearman_test(deaths ~ year, data = deathsyear)
```

```
##
##  Asymptotic Spearman Correlation Test
##
## data:  deaths by year
## Z = 5.5093, p-value = 3.602e-08
## alternative hypothesis: true rho is not equal to 0
```

- We see that the p-value is less than 0.05, so we can say that the NUll hypothesis is rejected. There is association between years and pneumonia/influenza deaths.

The correlation coefficient tells us that, at around 0.77, the relationship is pretty strong and is an upward trending relationship.