Coding self-assessment

Harris Coding Camp

As part of the statistics curriculum, you will be asked to analyze data using the programming language R. R is an open source language that is widely used by data analysts and data scientists. In the coding camp and coding lab, we provide an introduction to R coding focused on data analysis.

This is a self-assessment. If you feel comfortable completing this assignment by yourself (with the help of Google), then you are free to skip the coding camp and coding lab. Otherwise, you can use this to pick the right track for you.

Access the R file and data files you need to complete the tasks below, using this link.

Task 1:[1]

1. Install R and RStudio.
2. Install the package `readxl` and `tidyverse`.
3. Adjust the following code block to read in the provided data set:
   `incarceration_counts_and_rates_by_type_over_time.xlsx`

```
library(tidyverse)
library(readxl)
setwd(<Put path to file here>)
incarceration_data <- read_xlsx("incarceration_counts_and_rates_by_type_over_time.xlsx",
               range = "A7:CO10") %>%
    rename("type" = ...1) %>%
    pivot_longer(`1925`:`2016`, names_to = "year", values_to = "counts")
```

4. What does the code `library(tidyverse)` do and why is it necessary?
5. What does the code `library(readxl)` do and why is it necessary?
6. Why do you need to set a working directory (`setwd()`)?
7. How many vectors are there in this dataset? How many observations?
8. Briefly explain the difference between vectors, lists and data frame.

If you had trouble with `readxl`, we provide a csv file as well. You can load the data with the following code:

```
incarceration_data <- read_csv("incarceration_counts_and_rates_by_type_over_time.csv")
```

[1] Copying and pasting from the pdf will create issues in syntax–particularly it messes up the type of quotes used. We provide a file with this code in a text file. Alternatively, you can re-type the code or copy and paste and then fix syntax issues.

Task 2:

We want to analyze state prison counts by decade. We'll prepare the data in the following ways. Store the following changes in a new tibble (data frame) called `state_data`.

1. Add a column called `decade` that reflects which decade the observation comes from.
2. Filter the data so that you only have data from State prisons.
3. Use `select` to reorder the columns so that your data is organized as below:

```
## # A tibble: 10 x 4
##    type          counts decade  year
##    <chr>          <dbl>  <dbl> <dbl>
##  1 State prisons  85239   1920  1925
##  2 State prisons  91188   1920  1926
##  3 State prisons 101624   1920  1927
##  4 State prisons 108157   1920  1928
##  5 State prisons 107532   1920  1929
##  6 State prisons 117268   1930  1930
##  7 State prisons 124118   1930  1931
##  8 State prisons 125721   1930  1932
##  9 State prisons 125962   1930  1933
## 10 State prisons 126258   1930  1934
```

4. Finally, find out the mean, standard deviation, max and min value of `counts` for all observations from State prisons.

Task 3:

In this section, you'll use `group_by()` and `summarize()` to answer questions about state prison counts by decade.

1. Which decade saw the largest percentage growth in State prisons? Measure percent growth as $\frac{C_{d_e} - C_{d_s}}{C_{d_s}}$ where C_{d_e} is the count at the end of decade and C_{d_s} is the start of the decade). You may consider using the `first()` and `last()` functions.

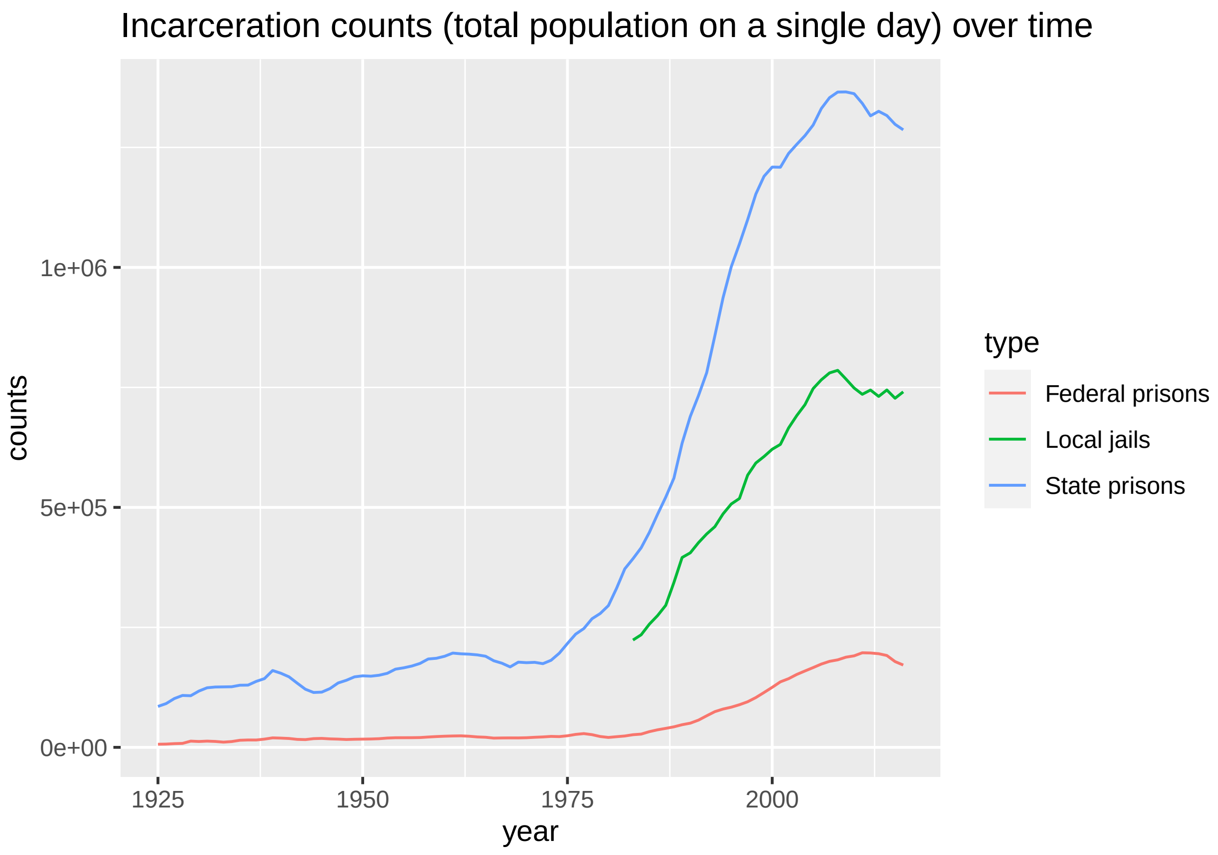
```
## # A tibble: 10 x 2
##    decade percentage_growth
##     <dbl>             <dbl>
##  1   1920            0.262
##  2   1930            0.365
##  3   1940           -0.0490
##  4   1950            0.245
##  5   1960           -0.0644
##  6   1970            0.581
##  7   1980            1.15
##  8   1990            0.725
##  9   2000            0.129
## 10   2010           -0.0553
```

Task 4:

You want to make a graph visualizing the change in incarceration counts in the United States over time.

```
incarceration_data %>%
  ggplot(???) +
  geom_???() +
  labs(???)
```

Adjust the code above in order to reproduce the following graph, including the choice of both axes, labels on both axes, choice of line type and title.



Task 5:

First, let's create two small datasets – Copy and run the code chunk below to assign these to `addr` and `phone`.

```r
addr <- data.frame(name = c("Alice","Bob",
                            "Carol","Dave",
                            "Eve"),
                   email = c("alice@company.com",
                             "bob@company.com",
                             "carol@company.com",
                             "dave@company.com",
                             "eve@company.com"),
                   stringsAsFactors = FALSE)

phone <- data.frame(fullname = c("Bob","Carol",
                                 "Dave","Eve",
                                 "Frank"),
                    phone = c("919 555-1111",
                              "919 555-2222",
                              "919 555-3333",
                              "310 555-4444",
                              "919 555-5555"),
                    stringsAsFactors = FALSE)
```

1. How would you correctly **left join** these two datasets? What is the resulting data frame? Is there any missing value?
2. Repeat the above step using **inner join**. What is the resulting data frame? Is there any missing value?
3. Repeat the above step using **full join**. What is the resulting data frame? Is there any missing value?

Task 6:

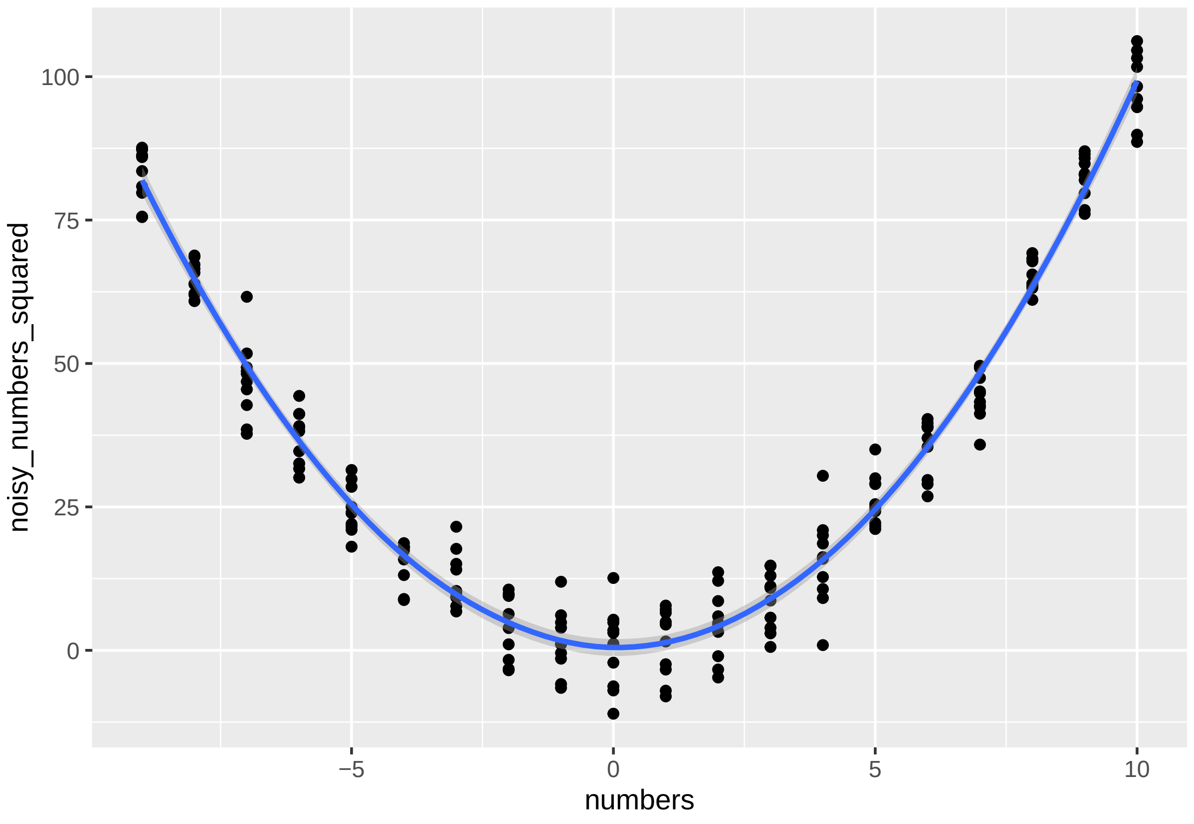
1. Take `numbers <- rep(seq(-9, 10, 1), 10)`. Using a for-loop, save the square of each number in a new vector called `numbers_squared`.
2. Take `numbers`. Using a for-loop, save the square of each number and add random noise using a call to `rnorm(1, sd = 5)` in a new vector called `noisy_numbers_squared`.

   You should be able to reproduce the graph below:

```
numbers_data <- tibble(numbers = numbers,
                       noisy_numbers_squared = noisy_numbers_squared)

numbers_data %>%
  ggplot(aes(x = numbers, y = noisy_numbers_squared)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Task 7:

1. Write a function called `notice_gpa` that takes `gpa` as an input and does the following:

- if `gpa` less than 2, prints: "Your GPA is `gpa`. You are on academic probation."
- else if `gpa` is greater than or equal to 3.5, prints: "Your GPA is `gpa`. You made the Dean's list. Congrats!"
- otherwise, prints: "Your GPA is `gpa`".

```r
notice_gpa <- function(gpa) {
  if (...) {
    ...
  } else if (...) {
    ...
  } else {
    ...
  }
}

# When running each of the following, you should get different results!
notice_gpa(1.9)
notice_gpa(3.5)
notice_gpa(3)
```

2. Before presenting task, we'll create a sample dataset `df` that contains some negative values. To begin with, simply run the following chunk of code to create `df`; don't worry about understanding this code.

```r
set.seed(54321)  # so that we all get the same random numbers
df <- tibble('id' = 1:100,
             'age' = sample(c(seq(11, 20, 1), -97, -98, -99),
                            size = 100,
                            replace = TRUE,
                            prob = c(rep(.09, 10), .1, .1, .1)),
             'sibage' = sample(c(seq(5, 12, 1), -97, -98, -99),
                               size = 100,
                               replace = TRUE,
                               prob = c(rep(.115, 8), .1, .1, .1)),
             'parage' = sample(c(seq(45, 55, 1), -4, -7, -8),
                               size = 100,
                               replace = TRUE,
                               prob = c(rep(.085, 11), .1, .1, .1))
)
```

(a) Write a function that counts the number of observations with **negative** values for `age` from `df`. (Hint: use `sum(x < 0)`)

(b) Write another function that counts the number of observations with **missing** values for `age` from `df`. (Hint: use `sum(is.na(x))`)