Core ERM Problem Set 1

1088708

2025-05-09

```
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)

# import libraries
library('tidyverse')
library('fredr')
library('patchwork')

# set parameters

current_key = Sys.getenv("FRED_API_KEY")
fredr_set_key(current_key)

gdp_seriesname = 'GDP'
unemp_seriesname = 'UNRATE'
alpha = 0.05
n_parta = 3
n_partc = 5

a = 3
b = 10
```

Collatz

1.

```
# store (part (b))
seq_3 = c(n_parta)

# check if even or odd
if (n_parta %% 2 == 0) {
  # get the next thing in the sequence
  next_value = n_parta / 2
} else {
  # get the next thing in the sequence
  next_value = n_parta * 3 + 1
}
```

a)

```
# print next value when n = 3
print(paste0('second value of collatz sequence is ', next_value))
```

```
## [1] "second value of collatz sequence is 10"
```

b)

```
# print first two values when n = 3
seq_3 = c(seq_3, next_value)
print(seq_3)
```

```
## [1]  3 10
```

c)

```
# now use n - 5
# start with the first element (i.e. the original number)
seq_5 = c(n_partc)

# while loop
while (n_partc != 1) {

  # use the if statement from part (a)
  if (n_partc %% 2 == 0) {
    # get the next thing in the sequence
    n_partc = n_partc / 2
  } else {
    # get the next thing in the sequence
    n_partc = n_partc * 3 + 1
  }

  # add the next value
  seq_5 = c(seq_5, n_partc)

}

# print the collatz sequence for n = 5
print(seq_5)
```

```
## [1]  5 16  8  4  2  1
```

d)

```
# define my function
collatz = function(n) {

  # start with the initial number again
  seq_n = c(n)

  # while loop from part (c)
  while (n != 1) {

    # use the if statement from part (a)
    if (n %% 2 == 0) {
      # get the next thing in the sequence
      n = n / 2
    } else {
```

```
      # get the next thing in the sequence
      n = n * 3 + 1
    }

    # add the next value
    seq_n = c(seq_n, n)

  }

  # return the whole sequence
  return (seq_n)

}

# test using n = 5
output_1d = collatz(n=10)
print(output_1d)
```

```
## [1] 10  5 16  8  4  2  1
```

2.

a)

```
# iterate through range
for (i in a:b) {
  print(paste0('collatz sequence for n = ', i, ':'))
  # call collatz
  # print entire sequence
  print(collatz(i))
}
```

```
## [1] "collatz sequence for n = 3:"
## [1]  3 10  5 16  8  4  2  1
## [1] "collatz sequence for n = 4:"
## [1] 4 2 1
## [1] "collatz sequence for n = 5:"
## [1]  5 16  8  4  2  1
## [1] "collatz sequence for n = 6:"
## [1]  6  3 10  5 16  8  4  2  1
## [1] "collatz sequence for n = 7:"
##  [1]  7 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
## [1] "collatz sequence for n = 8:"
## [1] 8 4 2 1
## [1] "collatz sequence for n = 9:"
##  [1]  9 28 14  7 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
## [1] "collatz sequence for n = 10:"
## [1] 10  5 16  8  4  2  1
```

b), c), and d)

```
collatz_holds = function(a, b) {
  # store output
  values = c(a:b)
```

```
  checks = c()

  for (i in values) {
    # print entire sequence: part (b)
    print(paste0('collatz sequence for n = ', i, ':'))
    print(collatz(i))

    # print last value: part (c)
    last = tail(collatz(i), n=1)
    print(paste0('last value of collatz sequence for n = ', i, ':'))
    print(last)

    # add last value to the vector of last values
    checks = c(checks, last == 1)

  }

  # return a readable display: part (d)
  return (tibble(value=values, ends_in_1=checks))
}

# test using a = 3, b = 10
output_2d = collatz_holds(a=3, b=10)
```

```
## [1] "collatz sequence for n = 3:"
## [1]  3 10  5 16  8  4  2  1
## [1] "last value of collatz sequence for n = 3:"
## [1] 1
## [1] "collatz sequence for n = 4:"
## [1] 4 2 1
## [1] "last value of collatz sequence for n = 4:"
## [1] 1
## [1] "collatz sequence for n = 5:"
## [1]  5 16  8  4  2  1
## [1] "last value of collatz sequence for n = 5:"
## [1] 1
## [1] "collatz sequence for n = 6:"
## [1]  6  3 10  5 16  8  4  2  1
## [1] "last value of collatz sequence for n = 6:"
## [1] 1
## [1] "collatz sequence for n = 7:"
##  [1]  7 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
## [1] "last value of collatz sequence for n = 7:"
## [1] 1
## [1] "collatz sequence for n = 8:"
## [1] 8 4 2 1
## [1] "last value of collatz sequence for n = 8:"
## [1] 1
## [1] "collatz sequence for n = 9:"
##  [1]  9 28 14  7 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
## [1] "last value of collatz sequence for n = 9:"
## [1] 1
## [1] "collatz sequence for n = 10:"
## [1] 10  5 16  8  4  2  1
```

```
## [1] "last value of collatz sequence for n = 10:"
## [1] 1
```

```
print(output_2d)
```

```
## # A tibble: 8 x 2
##   value ends_in_1
##   <int> <lgl>
## 1     3 TRUE
## 2     4 TRUE
## 3     5 TRUE
## 4     6 TRUE
## 5     7 TRUE
## 6     8 TRUE
## 7     9 TRUE
## 8    10 TRUE
```

3.

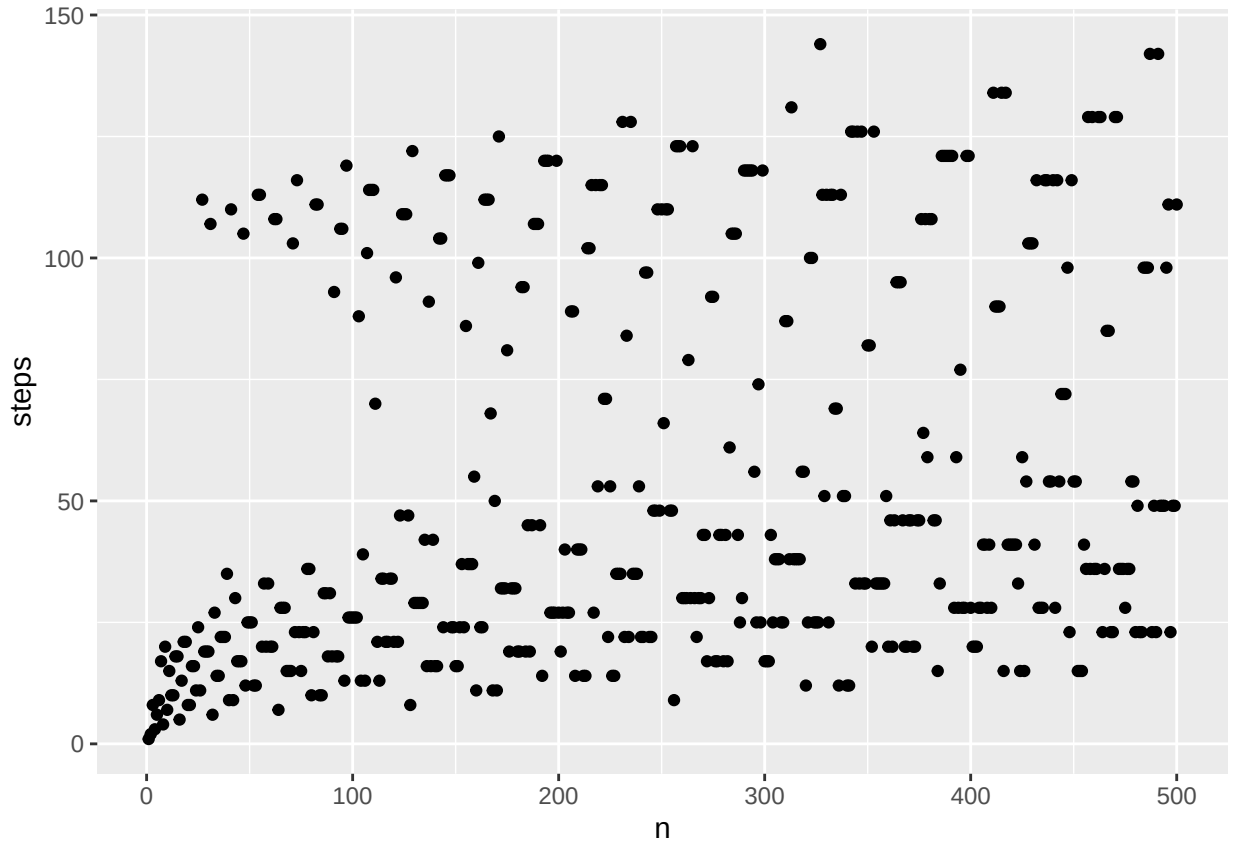
```
# define another function
longest_collatz = function(a, b) {
  # store output
  values = c(a:b)
  lengths = c()

  for (i in values) {
    # get sequence
    collatz_i = collatz(i)
    # save length of sequence
    lengths = c(lengths, length(collatz_i))
  }

  # find index of longest length
  max_length_idx = which.max(lengths)

  # return a readable display using index of longest length to find value
  return (tibble(n=values[max_length_idx], steps=lengths[max_length_idx]))
}

# run using a = 6, b = 100
output_3 = longest_collatz(a=6, b=100)
print(output_3)
```

```
## # A tibble: 1 x 2
##       n steps
##   <int> <int>
## 1    97   119
```

4.

```
# define another function
get_collatz_steps = function(n_max) {
  # store output
  values = c(1:n_max)
```

```
  lengths = c()

  # find collatz sequence length of each value from 1 to n_max
  for (i in values) {
    # get sequence
    collatz_i = collatz(i)
    # save length of sequence
    lengths = c(lengths, length(collatz_i))
  }

  # return a readable display
  return (tibble(n=values, steps=lengths))
}

output_4 = get_collatz_steps(n_max=500)

# scatter plot
ggplot(output_4, aes(x=n, y=steps)) + geom_point()
```



Lakisha

```
# import data
bm = read_csv('https://ditraglia.com/data/lakisha_aer.csv')
```

```
## Rows: 4870 Columns: 65
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr (10): id, ad, firstname, sex, race, city, kind, expminreq, schoolreq, ow...
## dbl (55): education, ofjobs, yearsexp, honors, volunteer, military, empholes...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1.

Bertrand and Mullainathan aim to determine if perception of race through the name provided on a resume influences resume responsiveness in the job market. They gather data by randomisation of resume quality, names, skills, and types of positions, submitting these randomised resumes to job openings and seeing which ones get callbacks. They found that applicants with "white"-sounding names had a statistically significant higher callback rate per resume sent compared to applicants with African-American-sounding names.

2.

a)

There are a lot of ways to display stuff to show the rows and columns but this shows 4870 rows and 65 columns. It describes rows and then columns since you usually write rows first when describing a matrix.

```
dim(bm)
```

```
## [1] 4870   65
```

b)

Shows that sex and race are dummies but use string encoding.

```
# subset columns
sex_names_race = bm[c('sex', 'race', 'firstname')]

# peek at values
str(sex_names_race)
```

```
## tibble [4,870 x 3] (S3: tbl_df/tbl/data.frame)
##  $ sex      : chr [1:4870] "f" "f" "f" "f" ...
##  $ race     : chr [1:4870] "w" "w" "b" "b" ...
##  $ firstname: chr [1:4870] "Allison" "Kristen" "Lakisha" "Latonya" ...
```

```
# just give data type
sapply(sex_names_race, class)
```

```
##         sex        race   firstname
## "character" "character" "character"
```

```
# see all the unique values for sex and race
print(unique(sex_names_race[['sex']]))
```

```
## [1] "f" "m"
```

```
print(unique(sex_names_race[['race']]))
```

```
## [1] "w" "b"
```

c)

```
# define new dummies using mutate()
bm = bm |>
  # if first argument true, new variable takes the value of second argument
  mutate(black = ifelse(race == 'b', TRUE, FALSE),
         female = ifelse(sex == 'f', TRUE, FALSE))
```

3.

a)

They created resumes by altering real resumes submitted online by job seekers in the cities of Boston and Chicago. These were collected from two employment websites. They also only collected resumes for four different job categories: sales, administrative support, clerical services, and customer services. To anonymise the resumes, they stripped all names and contact information from them and switched all Boston-related characteristics for Chicago ones and vice versa.

b)

They split the resumes into a group of good-quality and a group of bad-quality submissions. After anonymising the resumes, they judged resume quality using factors like experience and skills, but without consideration of race. They subtly inflated this quality gap by randomly adding additional desirable features to the good-quality resumes. Email addresses were almost exclusively used on the high-quality resumes.

c)

Using administrative data about births in the state of Massachusetts between 1974 and 1979, they found name frequencies given by black parents and white parents and categorised the names as White or African-American accordingly. They checked these classifications by asking survey respondents to categorise the names, upon which names with survey classifications not matching the classification assigned by the birth certificate data were discarded. The surnames were assigned with what the authors call "White-sounding" and "African-American-sounding" surnames. Each resume was then randomly assigned a name by picking out pairs of good- and bad-quality ones, giving a White and African-American name to each pair. Addresses were assigned randomly within each city.

4.

I ended up just writing a function to do this question. It is kind of sketchy but works for non-dummy categorical variables in general, and all kinds of encoding too. I just don't know how to make it run in not-O(n^2), but maybe that's not possible if I want to keep the function as general as possible. Also I'm just about to submit this and I realised that this function is probably not what you wanted, but I don't really know how to do this question with sum() but without being repetitive. I would've wanted a function that has a pipe in it where a groupby and summarise happen, but then I wouldn't know how to add arguments to the function since you can't refer to a variable as a string but you can't input an undefined variable into a function—a variable itself doesn't know what it's called.

```
proportions = function(data, group_var, target_var) {
  # how many categories of the variable you're grouping by
  group_vals = unique(data[[group_var]])
  # how many categories of the variable you're checking the balance of
  target_vals = unique(data[[target_var]])

  store_proportions = list()
```

```
  for (g in group_vals) {
    # go into each group
    subset_group = data[data[[group_var]] == g, ]
    # count obs
    total_group = nrow(subset_group)

    # now go into each of the other variable
    for (t in target_vals) {
      # find how many of each type there are
      count = sum(subset_group[[target_var]] == t)
      # divide by each group number of obs
      prop = count / total_group

      # 1-indexing is weird
      # but i'm adding all the information and then merging it together
      store_proportions[[length(store_proportions) + 1]] = tibble(
        group = g,
        category = t,
        proportion = prop
      )
    }
  }

  # put it all together
  group_results = bind_rows(store_proportions)
  return(group_results)
}
```

a)

Count sum and race: proportion of females is pretty close for black and white respondents, but there are more females because they only have female resumes for the administration or clerical jobs.

```
proportions(bm, 'black', 'female')
```

```
## # A tibble: 4 x 3
##   group category proportion
##   <lgl> <lgl>         <dbl>
## 1 FALSE TRUE          0.764
## 2 FALSE FALSE         0.236
## 3 TRUE  TRUE          0.775
## 4 TRUE  FALSE         0.225
```

```
bm %>%
  group_by(black) %>%
  summarise(
    prop_female = sum(female) / n()
  )
```

```
## # A tibble: 2 x 2
##   black prop_female
##   <lgl>       <dbl>
## 1 FALSE       0.764
## 2 TRUE        0.775
```

b)

computer skillzzz

```
proportions(bm, 'black', 'computerskills')
```

```
## # A tibble: 4 x 3
##   group category proportion
##   <lgl>    <dbl>      <dbl>
## 1 FALSE        1      0.809
## 2 FALSE        0      0.191
## 3 TRUE         1      0.832
## 4 TRUE         0      0.168
```

c)

jobs

```
proportions(bm, 'black', 'ofjobs')
```

```
## # A tibble: 14 x 3
##    group category proportion
##    <lgl>    <dbl>      <dbl>
##  1 FALSE        2    0.143
##  2 FALSE        3    0.298
##  3 FALSE        1    0.0222
##  4 FALSE        4    0.329
##  5 FALSE        5    0.106
##  6 FALSE        6    0.0998
##  7 FALSE        7    0.00287
##  8 TRUE         2    0.147
##  9 TRUE         3    0.289
## 10 TRUE         1    0.0230
## 11 TRUE         4    0.333
## 12 TRUE         5    0.113
## 13 TRUE         6    0.0908
## 14 TRUE         7    0.00493
```

education

```
proportions(bm, 'black', 'education')
```

```
## # A tibble: 10 x 3
##    group category proportion
##    <lgl>    <dbl>      <dbl>
##  1 FALSE        4    0.716
##  2 FALSE        3    0.211
##  3 FALSE        1    0.00739
##  4 FALSE        2    0.0583
##  5 FALSE        0    0.00739
##  6 TRUE         4    0.723
##  7 TRUE         3    0.202
##  8 TRUE         1    0.00903
##  9 TRUE         2    0.0542
## 10 TRUE         0    0.0115
```

d)

mean and standard deviation of yearsexp by race

```
bm |>
  # get mean and sd for both groups individually
  group_by(black) |>
  summarise(mean=mean(yearsexp), sd=sd(yearsexp))
```

```
## # A tibble: 2 x 3
##   black  mean    sd
##   <lgl> <dbl> <dbl>
## 1 FALSE  7.86  5.08
## 2 TRUE   7.83  5.01
```

e)

They want the two racial groups being compared to be similar in non-race characteristics so that they can isolate the impact of race.

f)

computer skills

```
proportions(bm, 'female', 'computerskills')
```

```
## # A tibble: 4 x 3
##   group category proportion
##   <lgl>    <dbl>      <dbl>
## 1 TRUE         1      0.868
## 2 TRUE         0      0.132
## 3 FALSE        1      0.662
## 4 FALSE        0      0.338
```

education

```
proportions(bm, 'female', 'education')
```

```
## # A tibble: 10 x 3
##    group category proportion
##    <lgl>    <dbl>      <dbl>
##  1 TRUE         4    0.671
##  2 TRUE         3    0.261
##  3 TRUE         1    0.00454
##  4 TRUE         2    0.0553
##  5 TRUE         0    0.00774
##  6 FALSE        4    0.880
##  7 FALSE        3    0.0249
##  8 FALSE        1    0.0205
##  9 FALSE        2    0.0596
## 10 FALSE        0    0.0151
```

These are not balanced across sex because these are the real characteristics of the resumes collected from real people which aren't necessarily balanced. They weren't trying to compare job search results across sex.

5.

a)

```
# ignore NaNs (not that there are any but in general...)
mean(bm$call, na.rm = TRUE)
```

```
## [1] 0.08049281
```

b)

Like in Table 1, the black == FALSE group (the resumes with White names) have a callback rate of about 9.65% and the black == TRUE group (the resumes with African-American names) have a callback rate of about 6.45%. The White resumes have a callback rate of about 1.5 times the African-American resumes.

```
bm |>
  # get average by both race and sex by grouping
  group_by(black) |>
  summarise(average_callback = mean(call), .groups = 'drop')
```

```
## # A tibble: 2 x 2
##   black average_callback
##   <lgl>            <dbl>
## 1 FALSE           0.0965
## 2 TRUE            0.0645
```

c)

In the data, males have a slightly lower average callback rate across race. For both males and females, White resumes have a higher average callback rate than African-American resumes.

```
bm |>
  # get average by both race and sex by grouping
  group_by(female, black) |>
  summarise(average_callback = mean(call), .groups = 'drop') |>
  # i tried to make it look like the picture
  pivot_wider(names_from = black,
              values_from = average_callback,
              names_prefix = 'black == ')
```

```
## # A tibble: 2 x 3
##   female `black == FALSE` `black == TRUE`
##   <lgl>             <dbl>           <dbl>
## 1 FALSE            0.0887          0.0583
## 2 TRUE             0.0989          0.0663
```

6.

a)

```
call_black = bm |>
  # take only black resumes
  filter(black == TRUE, na.rm=TRUE) |>
  pull(call)

call_white = bm |>
  # take only white resumes
```

```
  filter(black == FALSE, na.rm=TRUE) |>
  pull(call)
```

b)

Test the null hypothesis that there is no difference in callback rates between black- and white-sounding names against its two-sided alternative to find out

```
# sample sizes
n_black = length(call_black)
n_white = length(call_white)

# sample means
meancall_black = mean(call_black)
meancall_white = mean(call_white)

# sample variances
varcall_black = var(call_black)
varcall_white = var(call_white)

# test of unpaired sample means
teststatistic = (meancall_black - meancall_white) /
  sqrt(varcall_black/n_black + varcall_white /n_white)

print(teststatistic)
```

```
## [1] -4.114705
```

c) and d)

The p value is small enough to show up as zero on the table up to four decimal places so it's hard to compare aside from that, but the standard normal P-value is also zero up to four decimal places.

```
# standard normal critical value
criticalvalue = qnorm(alpha/2, lower.tail = FALSE)

# pnorm gives the distribution, so calculate standard normal p value from that
# use absolute value to make it always the same calculation
# x2 for two tailed
pvalue = 2 * pnorm(abs(teststatistic), lower.tail = FALSE)
```

e)

```
# decision rule
if (abs(teststatistic) > criticalvalue) {
  print ('reject')
} else {
  print ('fail to reject')
}
```

```
## [1] "reject"
```

Using the asymptotic distribution of the test statistic, we reject the null hypothesis of no difference in callback rates between black- and white-sounding names at the 5% level and conclude that there is a difference between callback rates.

7.

a)

I think it really depends on the culture or language the names are coming from so it's hard to compare how names are used across languages, but within different languages we can look at cultural factors. For example, Chinese given names tend to be really similar between siblings and people usually say their family names first (as a collectivist culture, I guess it puts more importance on the family you are in rather than yourself as an individual), so there might not be a lot of information that can be gathered by siblings with similar names. Meanwhile, large Quiverfull families like the Duggars may evoke different perceptions by naming all their kids J names. There was a trend a few years ago on Tiktok for the "old money" aesthetic and people were making Tiktoks with names that were giving old money (like Bartholomew) and everyone was making fun of them because they were super old-fashioned, so names can give perceptions of class and show generational differences.

b)

The outcome variable is one of their weaknesses, since with a bunch of fake resumes we can't see whether the fake applicant gets the job or not. Also, using names to suggest race meant that the names they actually included had to be distinguishable enough to perceive a race difference, so the results may not be representative enough of the average Black American who doesn't necessarily have that distinctive of a name. The last thing they listed was the fact that they only used newspaper ads, which aren't the only way to find a job. Actually I didn't know you could find a job in the newspaper.

c)

One factor mentioned for the lower callback rates for Black Americans is that hiring managers may be targeting their employee pool to match population demographics. Since Black Americans are a minority, they then receive lower callbacks. The authors find that some of their results are consistent with this kind of hiring rule. Another one they mentioned is that hiring managers care more about social background and are actually inferring social background through names rather than caring about race, but based on their results about the addresses chosen for the candidates, they consider this implausible. Finally, they propose the idea of "reverse discrimination" where screening candidates. Because hiring managers think that Black candidates are in high demand, they're less likely to try and recruit them, but the authors do not find evidence of this either.

d)

Taste-based discrimination is when employers deliberately refuse to hire qualified minorities that they don't like interacting with for whatever reason.Statistical discrimination is when you have imperfect information about a group of people and use statistical information to infer characteristics about them, like productivity. Because the resumes include verifiable information, statistical discrimination wouldn't be a good explanation in this experiment, unless the employers think some kind of affirmative action thing is going on that means the Black candidates don't need to put as much effort in to get these qualifications. Meanwhile, taste-based discrimination doesn't appear to have an impact because of how uniform the callback disparity is in the different fields of employment that they tested.

e)

They find that there was a rise in differences between the names that Black and White parents name their kids in the 1970s after the Black Power movement. Prior to this, Black and White parents did not name their children very differently. Fryer and Levitt do not find compelling evidence of culturally Black names having a negative impact on adulthood economic outcomes (controlling for circumstances like family characteristics and birth characteristics).

f)

While the Bertrand and Mullainathan paper doesn't observe outcomes (the fake applicant getting a job or not), the Fryer and Levitt paper doesn't necessarily observe the discrimination of employers directly (they only have some measures of outcomes). Fryer and Levitt propose three possible reconciliations of these results. First, they propose that Black applicants don't receive as many callbacks, but get offers at least at the same rate as White applicants because the name on the resume itself is irrelevant once a job interview is arranged. Second, they show that Black names can signal labour market productivity to employers. Finally, they propose that the outcome measurements they use aren't able to reflect discrimination in callbacks and unemployment.

FREDR

1.

b)

```
head(fredr(series_id = 'GDP'), 10)
```

```
## # A tibble: 10 x 5
##    date       series_id value realtime_start realtime_end
##    <date>     <chr>     <dbl> <date>         <date>
##  1 1946-01-01 GDP         NA  2025-04-30     2025-04-30
##  2 1946-04-01 GDP         NA  2025-04-30     2025-04-30
##  3 1946-07-01 GDP         NA  2025-04-30     2025-04-30
##  4 1946-10-01 GDP         NA  2025-04-30     2025-04-30
##  5 1947-01-01 GDP        243. 2025-04-30     2025-04-30
##  6 1947-04-01 GDP        246. 2025-04-30     2025-04-30
##  7 1947-07-01 GDP        250. 2025-04-30     2025-04-30
##  8 1947-10-01 GDP        260. 2025-04-30     2025-04-30
##  9 1948-01-01 GDP        266. 2025-04-30     2025-04-30
## 10 1948-04-01 GDP        273. 2025-04-30     2025-04-30
```

c) and d)

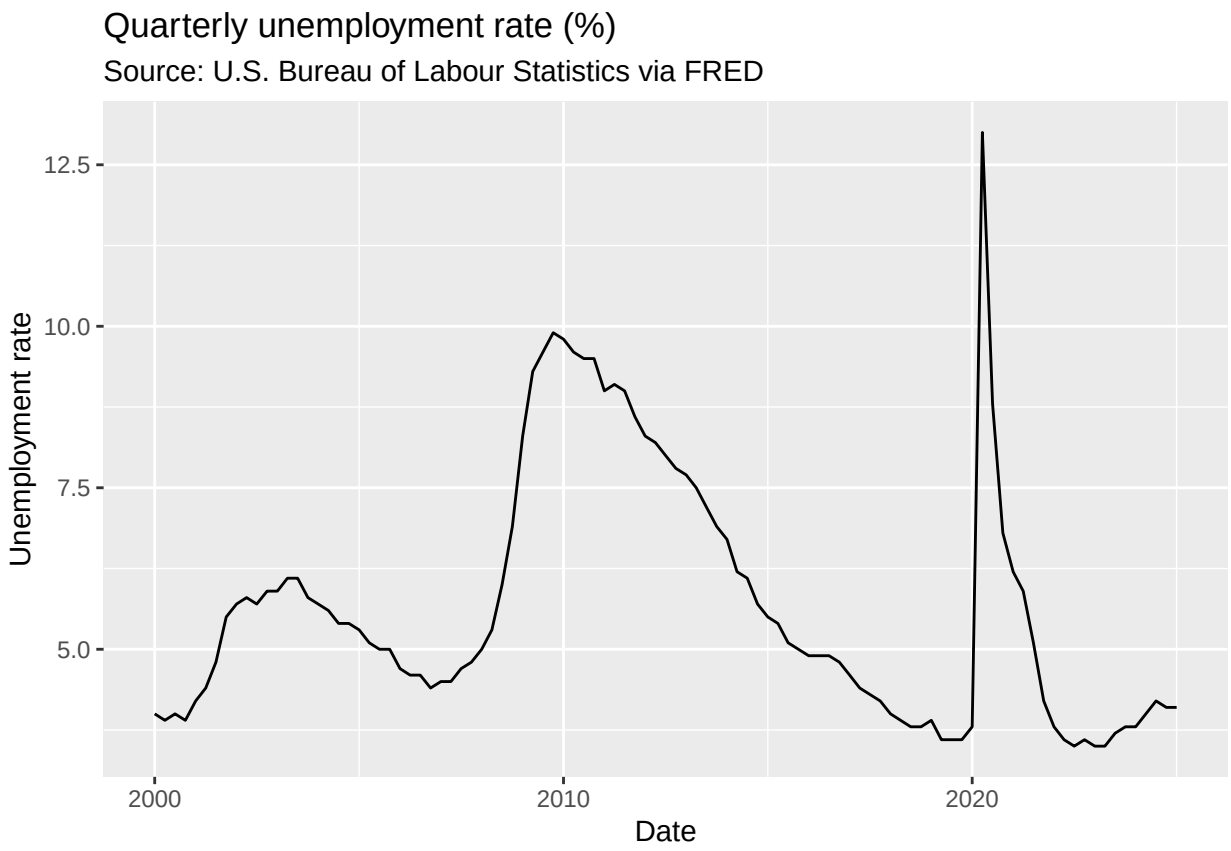
```
# gdp series
gdp = fredr(series_id = gdp_seriesname,
            observation_start=as.Date('2000-01-01'),
            frequency='q')[c('date', 'value')]

# unemployment series
u = fredr(series_id = unemp_seriesname,
          observation_start=as.Date('2000-01-01'),
          frequency='q')[c('date', 'value')]
```

2.

a)

```
u |>
  # don't plot NaNs
  drop_na() |>
  # say the data for the axes
  ggplot(aes(x=date, y=value)) +
```

```
  # line graph
  geom_line() +
  # labels
  ggtitle('Quarterly unemployment rate (%)',
          subtitle='Source: U.S. Bureau of Labour Statistics via FRED') +
  xlab('Date') +
  ylab('Unemployment rate')
```



b)

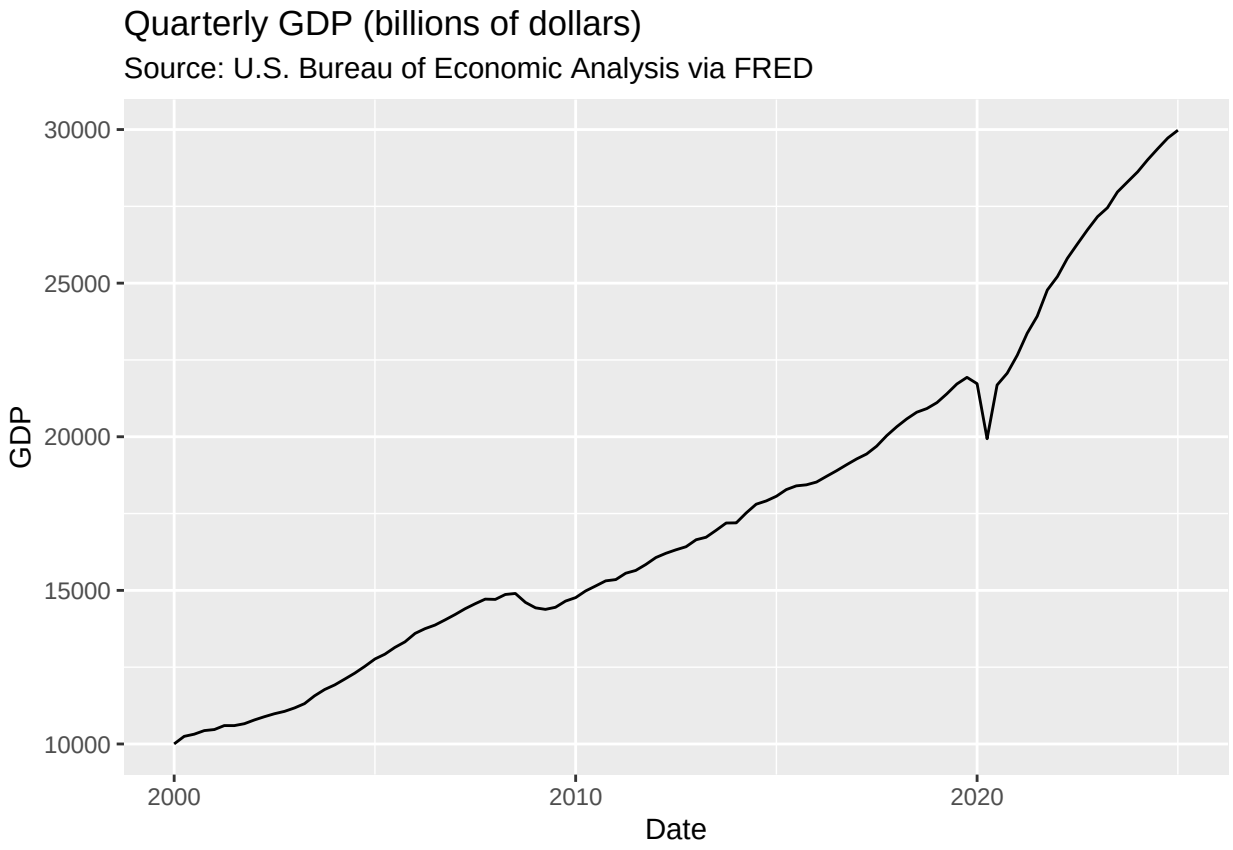
I can see increases right around COVID-19, the 2008 recession and what I'm assuming is SARS (early 2000s). after which unemployment always drops again. However, it took longer for unemployment to go back to its initial levels after the 2008 recession compared to the other two which were more like random shocks. COVID unemployment was the highest level of unemployment seen in this period, but also fell much quicker.

c)

This GDP series is the nominal value of total output in billions of USD. It's pretty good at showing the overall trend but since it's nominal the scale doesn't show cycles very well.

```
gdp |>
  # don't plot NaNs
  drop_na() |>
  # say the data for the axes
  ggplot(aes(x=date, y=value)) +
  # line graph
  geom_line() +
  # labels
```

```
  ggtitle('Quarterly GDP (billions of dollars)',
          subtitle='Source: U.S. Bureau of Economic Analysis via FRED') +
  xlab('Date') +
  ylab('GDP')
```



d)

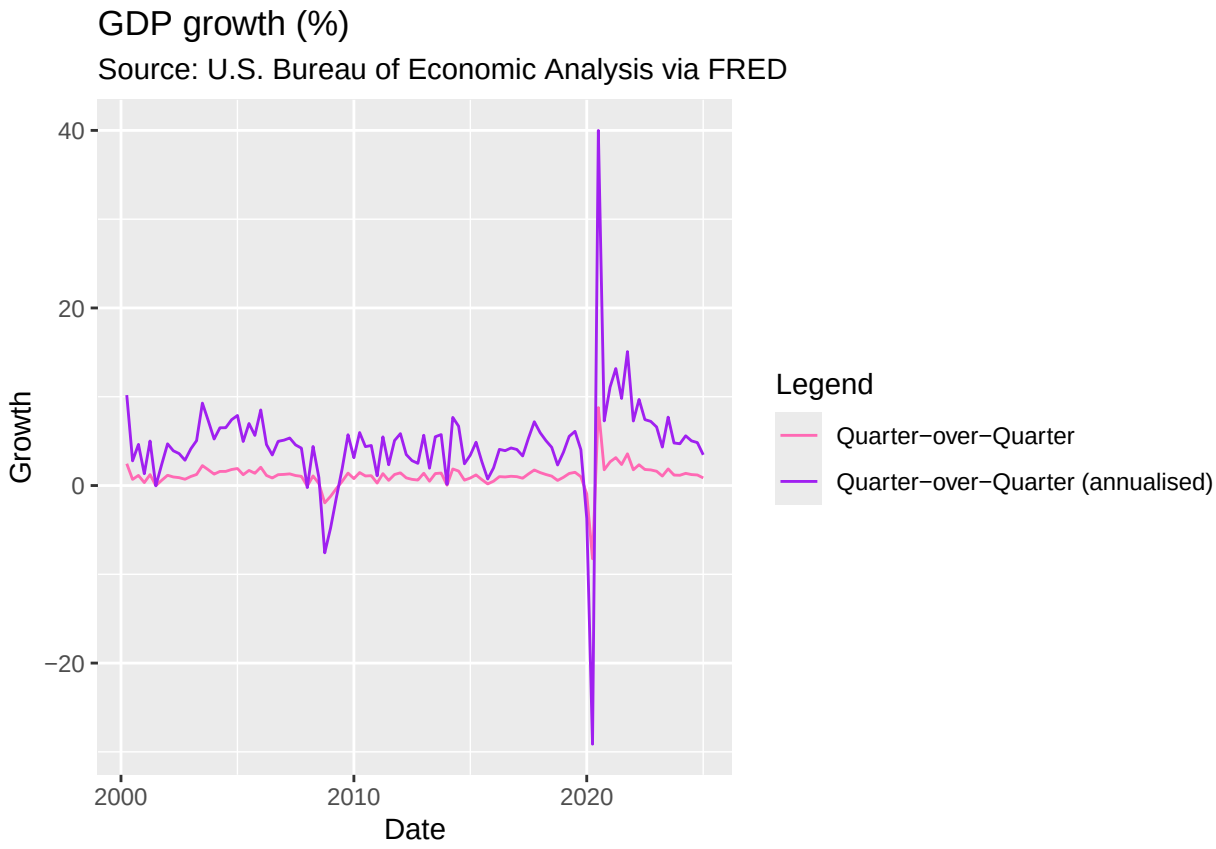
I don't actually get the "Removed 1 row containing missing values" warning and I think this is because the entire first row of the dataframe isn't made up of all missing values (the regular GDP series still has a non-empty value at the first period) but the first period will always be missing in the growth rate variables since it's needed as the initial value to calculate the growth rate.

```
gdp = gdp |>
  # add quarter over quarter growth and get it in percentage points
  mutate(qoq = (value / lag(value) - 1) * 100) |>
  # annualised quarter over quarter growth and get it in percentage points
  mutate(qoq_annualised = ((1 + qoq / 100)^4 - 1) * 100)
```

e)

```
gdp |>
  # don't plot NaNs
  drop_na() |>
  # create the x axis
  ggplot(aes(date)) +
  # draw the two lines
  geom_line(aes(y=qoq,
```

```
                colour='Quarter-over-Quarter')) +
  geom_line(aes(y=qoq_annualised,
                colour='Quarter-over-Quarter (annualised)')) +
  # labels
  ggtitle('GDP growth (%)',
          subtitle='Source: U.S. Bureau of Economic Analysis via FRED') +
  labs(y = 'Growth', x= 'Date', colour = 'Legend') +
  # colours
  scale_color_manual(
    values = c(
      'Quarter-over-Quarter' = 'hotpink',
      'Quarter-over-Quarter (annualised)' = 'purple'
    )
  )
```



e)

The two columns won't be identical since the FRED data is rounded, but they can be "close enough".

```
# get growth rate
gdp_growth = fredr(series_id = gdp_seriesname,
            observation_start=as.Date('2000-04-01'),
            frequency='q',
            units='pch')[c('date', 'value')] |>
  rename(qoq_fromfred = value)

# add it to the rest of the data
```

```r
gdp_merged = merge(x=gdp, y=gdp_growth, by.x='date', by.y='date', all= TRUE)

# check if the values are identical-ish across all obs, ignoring NaNs
# this is the largest difference and it is clearly very small
biggestdif = max(gdp_merged$qoq - gdp_merged$qoq_fromfred, na.rm=TRUE)
print(paste0('biggest difference between calculated and FRED: ', biggestdif))
```

```
## [1] "biggest difference between calculated and FRED: 4.42137984246749e-06"
```

```r
# or just check if all are equal enough
print('values are equal up to 5 decimal places: ')
```

```
## [1] "values are equal up to 5 decimal places: "
```

```r
all(abs(gdp_merged$qoq - gdp_merged$qoq_fromfred) < 0.00001, na.rm = TRUE)
```

```
## [1] TRUE
```

3.

```r
recessions <- tibble(
  start = as.Date(c("2001-03-01", "2007-12-01", "2020-02-01")),
  end   = as.Date(c("2001-11-30", "2009-06-30", "2020-04-30"))
)
```

a) and b)

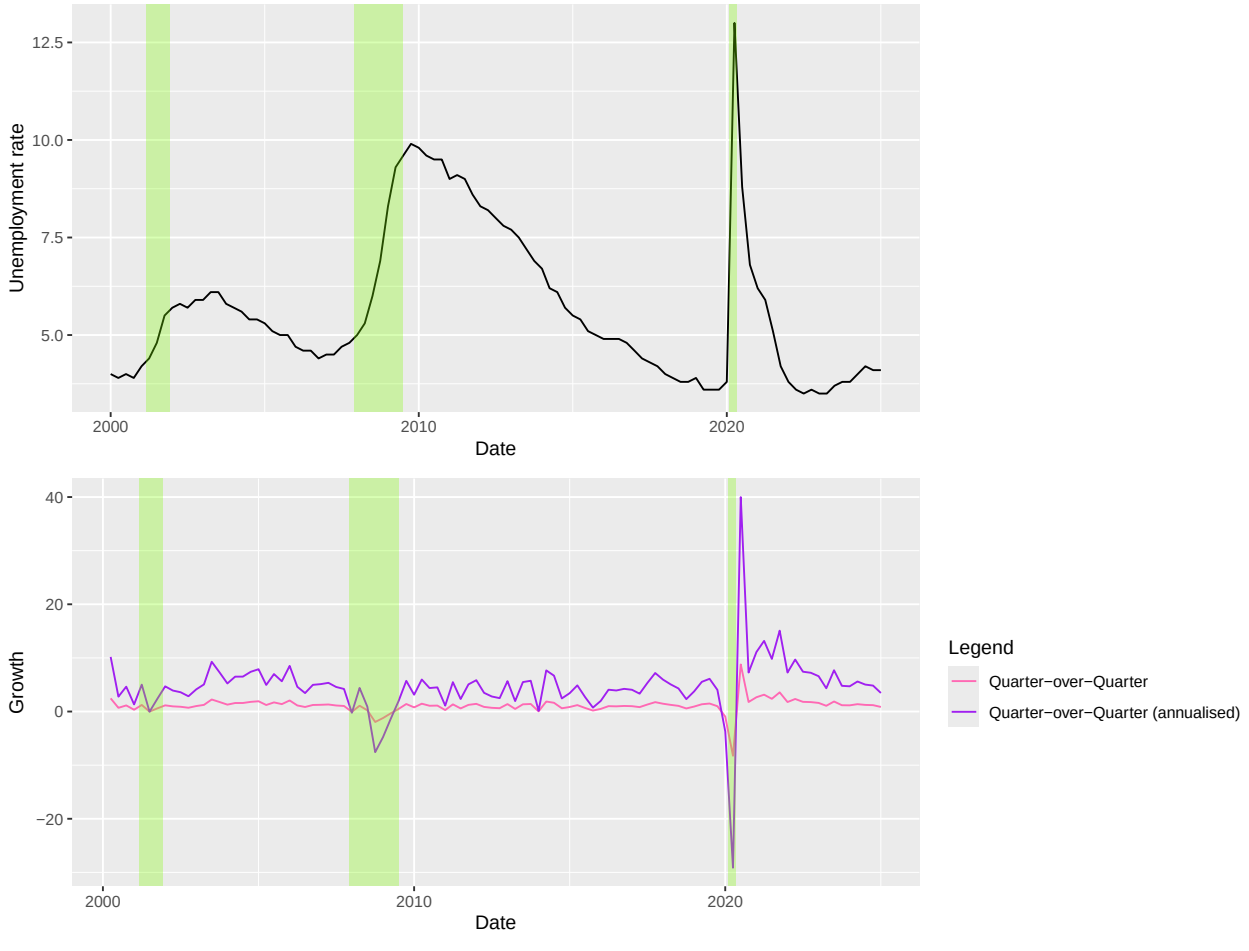
```r
# unemployment
plot1 = u |>
  # don't plot NaNs
  drop_na() |>
  # say the data for the axes
  ggplot(aes(x=date, y=value)) +
  # line graph
  geom_line() +
  # shading
  geom_rect(data = recessions,aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
            fill='chartreuse', alpha=0.3, inherit.aes=FALSE) +
  # labels
  xlab('Date') +
  ylab('Unemployment rate')

# GDP
plot2 = gdp |>
  # don't plot NaNs
  drop_na() |>
  # set the x axis
  ggplot(aes(date)) +
  # plot the lines
  geom_line(aes(y=qoq,
                colour='Quarter-over-Quarter')) +
  geom_line(aes(y=qoq_annualised,
                colour='Quarter-over-Quarter (annualised)')) +
  # shading
  geom_rect(data = recessions,aes(xmin=start, xmax=end, ymin=-Inf, ymax=Inf),
```

```
            fill='chartreuse', alpha=0.3, inherit.aes=FALSE) +
  # labels
  labs(y = 'Growth', x= 'Date', colour = 'Legend') +
  scale_color_manual(
    values = c(
      'Quarter-over-Quarter' = 'hotpink',
      'Quarter-over-Quarter (annualised)' = 'purple'
    )
  )
```

c)

```
# both
# patchwork plot_annotation
(plot1 / plot2) +
  plot_annotation(
    title = 'the theme of my graph is glinda and elphaba from wicked',
    caption = 'Source: U.S. BLS, BEA via FRED; NBER'
  )
```

the theme of my graph is glinda and elphaba from wicked



Source: U.S. BLS, BEA via FRED; NBER