

Joining data - Logistic regression

Frank Edwards

2/22/2019

Joining data with dplyr

- Add new rows to a data frame: `rbind()`
- Add new columns to a data frame: `cbind()`
- Join full data frames: `left_join()`

When to rbind()

Bind rows to a dataframe when you have new observations to append. The number, type, and names of columns *must* be identical.

I find rbind to be most useful when I need to add a single or small set of observations manually to a data frame.

How to rbind()

```
tail(gapminder)
```

```
## # A tibble: 6 x 6
```

##	country	continent	year	lifeExp	pop	gdpPercap
##	<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
## 1	Zimbabwe	Africa	1982	60.4	7636524	789.
## 2	Zimbabwe	Africa	1987	62.4	9216418	706.
## 3	Zimbabwe	Africa	1992	60.4	10704340	693.
## 4	Zimbabwe	Africa	1997	46.8	11404948	792.
## 5	Zimbabwe	Africa	2002	40.0	11926563	672.
## 6	Zimbabwe	Africa	2007	43.5	12311143	470.

```

new_data <- data.frame(country = "Zimbabwe", continent = "Africa", year = 2012,
  lifeExp = 50.2, pop = 12345678, gdpPercap = 500)
new_gap <- rbind(gapminder, new_data)
tail(new_gap)

```

```

## # A tibble: 6 x 6
##   country continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <dbl>   <dbl>   <dbl>   <dbl>
## 1 Zimbabwe Africa    1987    62.4  9216418    706.
## 2 Zimbabwe Africa    1992    60.4 10704340    693.
## 3 Zimbabwe Africa    1997    46.8 11404948    792.
## 4 Zimbabwe Africa    2002    40.0 11926563    672.
## 5 Zimbabwe Africa    2007    43.5 12311143    470.
## 6 Zimbabwe Africa    2012    50.2 12345678    500

```

When to cbind()

Bind columns to a dataframe when you have a new variable to append. The number of rows *must* be identical in both objects.

I use cbind most often when combining vectors to create data frames.

How to cbind()

```
captain <- c("Kirk", "Picard", "Janeway", "Sisko")
ship <- c("Enterprise", "Enterprise", "Voyager", "Defiant")
star_trek <- data.frame(cbind(captain, ship))
star_trek
```

```
##   captain      ship
## 1   Kirk Enterprise
## 2  Picard Enterprise
## 3 Janeway  Voyager
## 4   Sisko  Defiant
```


Joins are more flexible than `cbind()` and `rbind()`. Joins take two data frames and merge them according to your specification.

They are powerful tools for flexibly merging datasets with common identifier variables.

Joins merge data frames based on matching key values.

Kinds of join: from `?dplyr::join`

- `inner_join(x,y)`: Return all rows from x with matching values in y, and all columns from x and y.
- `left_join(x,y)`: My most frequently used join function. Returns all rows from x and all columns from x and y.
- `right_join(x,y)`: Returns all rows from y and all columns from x and y
- `full_join(x,y)`: Return all rows and columns from both x and y
- `semi_join(x,y)`: Return all rows from x with matching values in y, returning only columns in x
- `anti_join(x,y)`: Return all rows from x where there are not matching values in y, keeping only columns from x

What do joins do?

```
band_members
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
band_members %>% inner_join(band_instruments)
```

```
## # A tibble: 2 x 3  
##   name   band   plays  
##   <chr> <chr>   <chr>  
## 1 John   Beatles guitar  
## 2 Paul   Beatles  bass
```

```
band_members %>% left_join(band_instruments)
```

```
## # A tibble: 3 x 3  
##   name  band    plays  
##   <chr> <chr>   <chr>  
## 1 Mick  Stones <NA>  
## 2 John  Beatles guitar  
## 3 Paul  Beatles bass
```

```
band_members %>% right_join(band_instruments)
```

```
## # A tibble: 3 x 3  
##   name  band    plays  
##   <chr> <chr>   <chr>  
## 1 John  Beatles guitar  
## 2 Paul  Beatles bass  
## 3 Keith <NA>    guitar
```

```
band_members %>% full_join(band_instruments)
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

Filtering joins

```
band_members %>% semi_join(band_instruments)
```

```
## # A tibble: 2 x 2  
##   name    band  
##   <chr> <chr>  
## 1 John   Beatles  
## 2 Paul   Beatles
```


Filtering joins

```
band_members %>% anti_join(band_instruments)
```

```
## # A tibble: 1 x 2
```

```
##   name    band
```

```
##   <chr> <chr>
```

```
## 1 Mick  Stones
```

A practical example

```
g_c <- gapminder %>% group_by(continent, year) %>% summarise(gdp_mean_continent = mean(gdpPercap))

gapminder %>% left_join(g_c)
```

```
## # A tibble: 1,704 x 7
##   country      continent  year lifeExp      pop gdpPercap gdp_mean_contine-
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>      <dbl>
## 1 Afghanistan Asia      1952   28.8  8425333    779.      5195.
## 2 Afghanistan Asia      1957   30.3  9240934    821.      5788.
## 3 Afghanistan Asia      1962   32.0 10267083    853.      5729.
## 4 Afghanistan Asia      1967   34.0 11537966    836.      5971.
## 5 Afghanistan Asia      1972   36.1 13079460    740.      8187.
## 6 Afghanistan Asia      1977   38.4 14880372    786.      7791.
## 7 Afghanistan Asia      1982   39.9 12881816    978.      7434.
## 8 Afghanistan Asia      1987   40.8 13867957    852.      7608.
## 9 Afghanistan Asia      1992   41.7 16317921    649.      8640.
## 10 Afghanistan Asia      1997   41.8 22227415    635.      9834.
## # ... with 1,694 more rows
```

Another practical example

```
head(pop)
```

```
## # A tibble: 6 x 5
##   year race sex    age    pop
##   <dbl> <chr> <chr> <chr> <dbl>
## 1  1991 amind Female 0      22465
## 2  1991 amind Female 1-4    84894
## 3  1991 amind Female 10-14  97299
## 4  1991 amind Female 15-19  91647
## 5  1991 amind Female 20-24  86044
## 6  1991 amind Female 25-29  92740
```

```
head(mort)
```

```
## # A tibble: 6 x 5
##   year age race                sex    deaths
##   <dbl> <chr> <chr>                <chr> <dbl>
## 1  2008 0    African American    Female  3807
## 2  2008 0    African American    Male    4761
## 3  2008 0    American Indian/AK Native Female    169
## 4  2008 0    American Indian/AK Native Male     234
## 5  2008 0    Asian/Pacific Islander Female   448
## 6  2008 0    Asian/Pacific Islander Male     570
```

Check values on common variables

```
unique(pop$race)
```

```
## [1] "amind" "asian" "black" "latino" "white"
```

```
unique(mort$race)
```

```
## [1] "African American" "American Indian/AK Native"  
## [3] "Asian/Pacific Islander" "Latinx"  
## [5] "White"
```

Recode variable for join

```
pop<-pop%>%  
  mutate(race = case_when(  
  
    race == "amind" ~ "American Indian/AK Native",  
  
    race == "black" ~ "African American",  
  
    race == "asian" ~ "Asian/Pacific Islander",  
  
    race == "latino" ~ "Latinx",  
  
    race == "white" ~ "White"  
  ))
```

Check values on common variables

```
unique(pop$sex)
```

```
## [1] "Female" "Male"
```

```
unique(mort$sex)
```

```
## [1] "Female" "Male"
```

Check values on common variables

```
unique(pop$year)
```

```
## [1] 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002  
## [15] 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
```

```
unique(mort$year)
```

```
## [1] 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
```

Check values on common variables

```
unique(pop$age)
```

```
##  [1] "0"      "1-4"    "10-14"  "15-19"  "20-24"  "25-29"  "30-34"  
##  [9] "40-44"  "45-49"  "5-9"    "50-54"  "55-59"  "60-64"  "65-69"  
## [17] "75-79"  "80-84"  "85+"
```

```
unique(mort$age)
```

```
##  [1] "0"      "1-4"    "10-14"  "15-19"  "20-24"  "25-29"  
##  [8] "35-39"  "40-44"  "45-49"  "5-9"    "50-54"  "55-59"  
## [15] "65-69"  "70-74"  "75-79"  "80-84"  "85+"    "Missi"
```


Join population onto mortality

```
mort_join <- mort %>% left_join(pop)
```

Check results

```
head(mort_join)
```

```
## # A tibble: 6 x 6
##   year age  race                sex  deaths  pop
##   <dbl> <chr> <chr>                <chr>  <dbl>  <dbl>
## 1  2008  0     African American    Female   3807 341912
## 2  2008  0     African American    Male     4761 353833
## 3  2008  0     American Indian/AK Native Female    169  36155
## 4  2008  0     American Indian/AK Native Male     234  37284
## 5  2008  0     Asian/Pacific Islander Female    448 111074
## 6  2008  0     Asian/Pacific Islander Male     570 117174
```

```
nrow(mort)
```

```
## [1] 1996
```

```
nrow(mort_join)
```

```
## [1] 1996
```

```
nrow(pop)
```

```
## [1] 5320
```

What about a full_join()?

```
mort_join <- mort %>%  
full_join(pop) %>%  
arrange(year, age, race, sex)
```

Check results

```
head(mort_join)
```

```
## # A tibble: 6 x 6
##   year age  race                sex  deaths  pop
##   <dbl> <chr> <chr>                <chr>  <dbl>  <dbl>
## 1  1991  0     African American    Female    NA 313730
## 2  1991  0     African American    Male      NA 320671
## 3  1991  0     American Indian/AK Native Female    NA 22465
## 4  1991  0     American Indian/AK Native Male      NA 22924
## 5  1991  0     Asian/Pacific Islander Female    NA 68661
## 6  1991  0     Asian/Pacific Islander Male      NA 71908
```

```
nrow(mort)
```

```
## [1] 1996
```

```
nrow(mort_join)
```

```
## [1] 5416
```

```
nrow(pop)
```

```
## [1] 5320
```

Common join key variables

- Year
- Subject ID
- Geo ID (country, FIPS, state, etc)
- Agency ID
- Anything you want, but probably always a categorical variable

Break

Logistic regression

Read in the data for today

```
admissions <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")  
head(admissions)
```

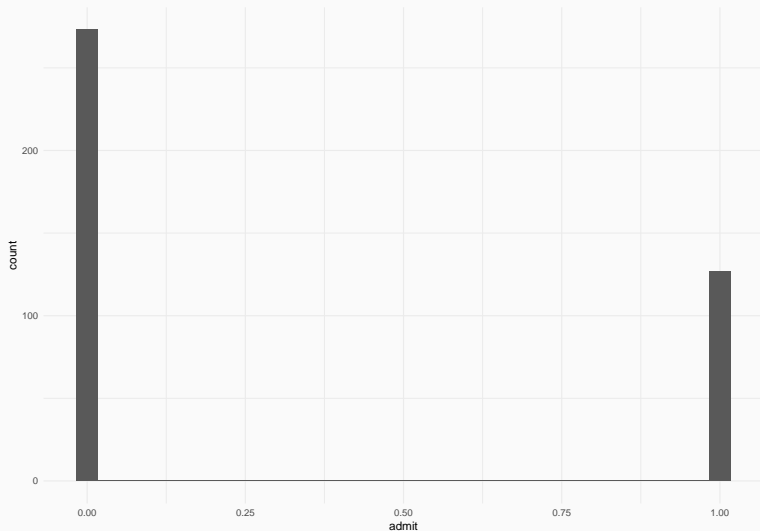
```
##   admit gre  gpa rank  
## 1     0 380 3.61    3  
## 2     1 660 3.67    3  
## 3     1 800 4.00    1  
## 4     1 640 3.19    4  
## 5     0 520 2.93    4  
## 6     1 760 3.00    2
```

```
nrow(admissions)
```

```
## [1] 400
```


Evaluate distribution of binary admission variable

```
ggplot(admissions, aes(x = admit)) + geom_histogram()
```



If y is an i.i.d. Bernoulli variable with probability p :

$$y \sim \text{Bernoulli}(p)$$

$$\Pr(y = 1) = p = 1 - \Pr(y = 0)$$

$$E(y) = \bar{y} = p$$

$$\text{Var}(y) = p(1 - p)$$

Summary of admit: What can we say about the probability of admission?

```
mean(admissions$admit)
```

```
## [1] 0.3175
```

```
sum(admissions$admit == 1)/nrow(admissions)
```

```
## [1] 0.3175
```

```
var(admissions$admit)
```

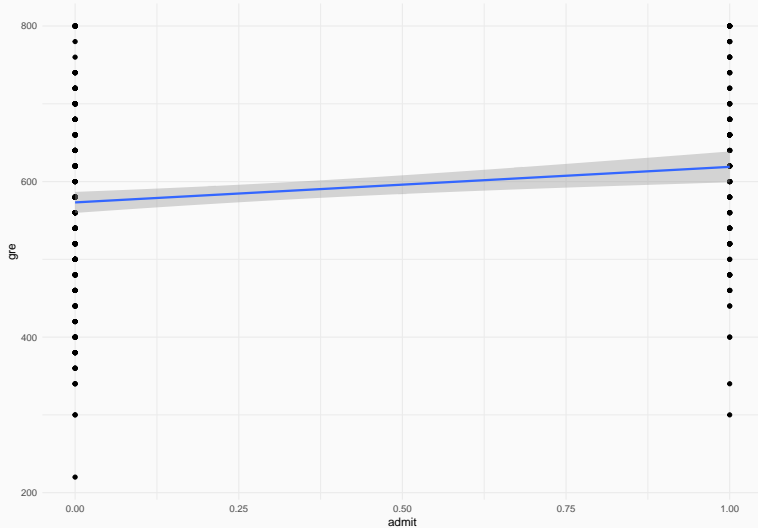
```
## [1] 0.2172368
```

```
mean(admissions$admit) * (1 - mean(admissions$admit))
```

```
## [1] 0.2166937
```

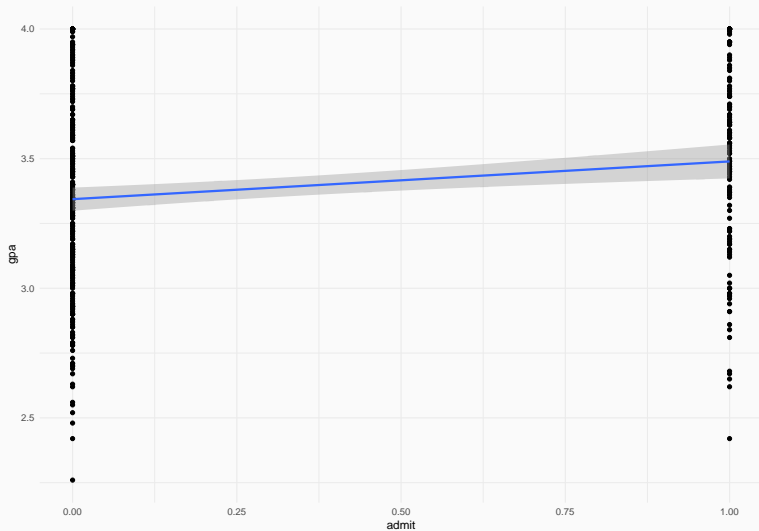
How does GRE relate to admission?

```
ggplot(admissions, aes(x = admit, y = gre)) + geom_point() + geom
```



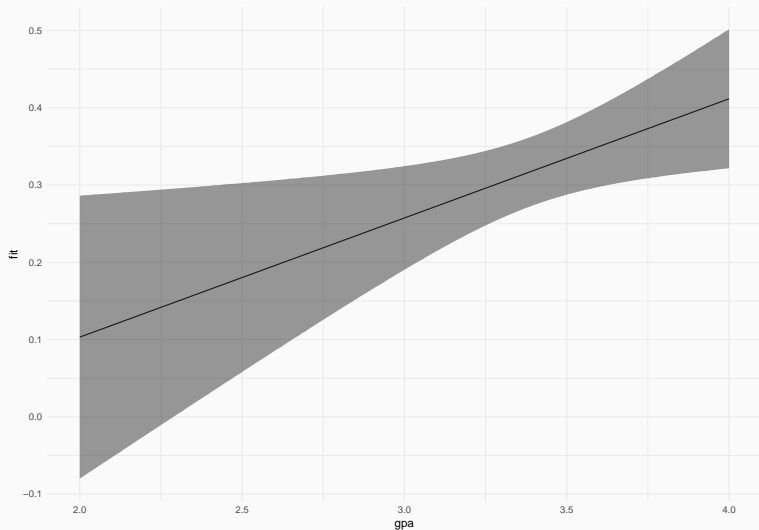
GPA?

```
ggplot(admissions, aes(x = admit, y = gpa)) + geom_point() + geom
```



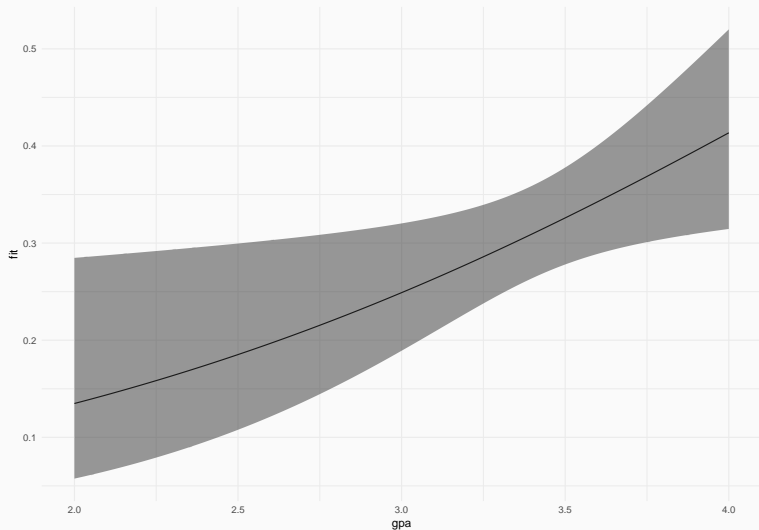
Can we fit a model to predict admission?

```
m1 <- lm(admit ~ gre + gpa, data = admissions)
```



Let's try a different approach

```
m2 <- glm(admit ~ gre + gpa, data = admissions, family = "binomial")
```



A generalized linear model

Our linear probability model was:

$$Pr(admit = 1) = \beta_0 + \beta_1 GRE + \beta_2 GPA + \beta_3 Rank + \varepsilon$$

Our logistic regression model takes the form:

$$logit(Pr(admit = 1)) = \beta_0 + \beta_1 GRE + \beta_2 GPA + \beta_3 Rank$$

The logit function is our link between the linear predictor term $X\beta$ and the outcome *admit*.

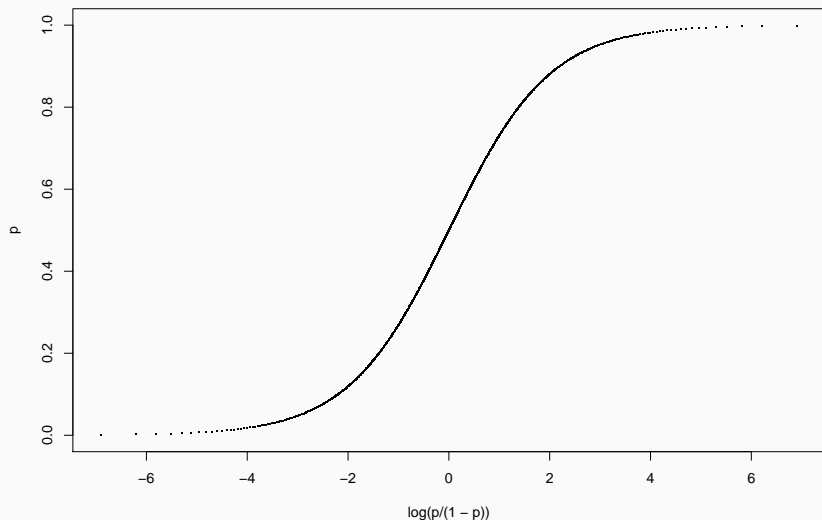
The logit function

The logit function transforms a probability value on $[0, 1]$ to a continuous distribution

$$\text{logit}(p) = \log \frac{p}{1 - p}$$

The logit function

```
p <- seq(0, 1, 0.001)
plot(log(p/(1 - p)), pch = ".", p)
```



Logistic regression is a GLM with a logit link

A generalized linear model with link function g takes the form:

$$g(y) = x\beta$$

For OLS, the link function is the identity function $g(y) = y$

For logistic regression, the link function is the logit function

$$\text{logit}(y) = x\beta$$

$$y = \text{logit}^{-1}(x\beta)$$

$$\text{logit}(p) = \log \frac{p}{1-p}$$

$$\text{logit}^{-1}(x) = \frac{\exp(x)}{\exp(x) + 1}$$

We can use these functions to transform values back and forth from our logit-linear scale and the probability scale.

Uses the logit function to model the probability of a binary outcome being equal to 1. The logit function transforms the bounded interval $[0, 1]$ to a continuous distribution, allowing us to proceed with building a regression model as we ordinarily would.

Logistic regression may have more accurate uncertainty estimates than a linear probability model for binary outcomes. Logistic regression also constrains model predictions to $[0, 1]$.

Running logistic models in R: the glm() function

```
m1 <- glm(admit ~ gpa, data = admissions, family = "binomial")
tidy(m1)
```

```
## # A tibble: 2 x 5
##   term          estimate std.error statistic    p.value
##   <chr>          <dbl>     <dbl>    <dbl>    <dbl>
## 1 (Intercept)   -4.36      1.04    -4.21 0.0000257
## 2 gpa           1.05      0.299     3.52 0.000437
```

How do we interpret the coefficients?

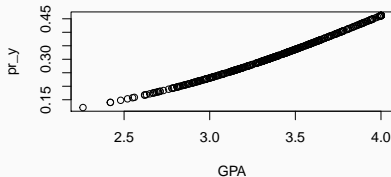
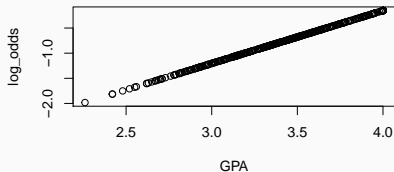
- Log odds: β_1
- Odds ratio: e^{β_1}
- Probability: $\text{logit}^{-1}(x) = \frac{\exp(x\beta)}{\exp(x\beta)+1}$

I tend to prefer transforming to a probability scale, as log odds and odds ratios are a bit confusing to define and are not especially intuitive.

To get predicted probabilities from m1

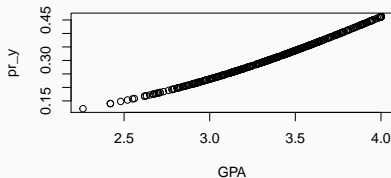
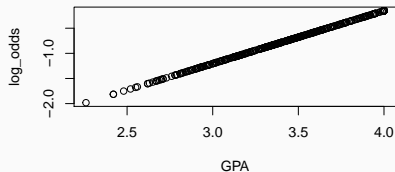
We need $X\beta$, then apply the logit inverse function

```
x <- cbind(rep(1, nrow(admissions)), admissions$gpa)
log_odds <- coef(m1) %*% t(x)
pr_y <- exp(log_odds)/(exp(log_odds) + 1)
par(mfrow = c(1, 2))
plot(x[, 2], log_odds, xlab = "GPA")
plot(x[, 2], pr_y, xlab = "GPA")
```



Alternatively

```
log_odds <- predict(m1)
pr_y <- predict(m1, type = "response")
```



- Complete paper proposal due next week: Provide me with a 200 - 500 word abstract that describes your proposed research. This proposal should be accompanied by a description or visualization of key variables if available. If not available, explain your plans for obtaining and working with the data in a timely fashion.
- Hw 4 is also due next week. Sorry to double up!