Statistics

Summary

- `ggplot()` specifies what data to use and what variables will be mapped to where
- inside `ggplot()`, `aes(x = , y = , color = )` specify what variables correspond to what aspects of the plot in general
- layers of plots can be combined using the `+` at the **end** of lines
- use `geom_line()` and `geom_point()` to add lines and points
- sometimes you need to add a `group` element to `aes()` if your plot looks strange
- make sure you are plotting what you think you are by checking the numbers!
- `facet_grid(~variable)` and `facet_wrap(~variable)` can be helpful to quickly split up your plot

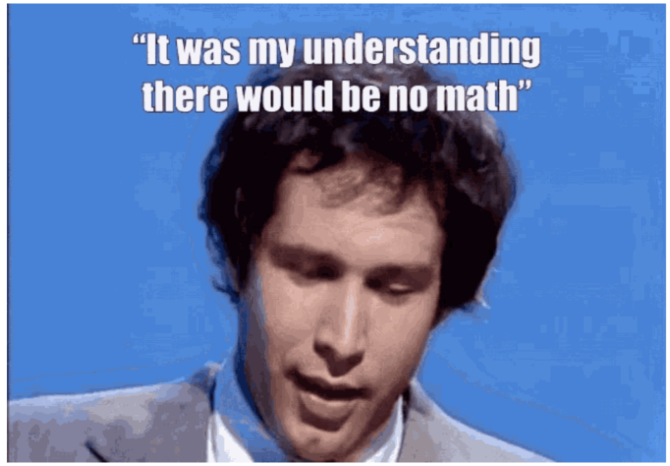
Summary

- the factor class allows us to have a different order from alphanumeric for categorical data
- we can change data to be a factor variable using `mutate()`, `as_factor()` (in the `forcats` package), or `factor()` functions and specifying the levels with the `levels` argument
- `fct_reorder({variable_to_reorder}, {variable_to_order_by})` helps us reorder a variable by the values of another variable
- arranging, tabulating, and plotting the data will reflect the new order

Overview

We will cover how to use R to compute some of basic statistics and fit some basic statistical models.

- Correlation
- T-test
- Linear Regression / Logistic Regression


"It was my understanding
there would be no math"

Overview

We will focus on how to use R software to do these. We will be glossing over the statistical **theory** and “formulas” for these tests. Moreover, we do not claim the data we use for demonstration meet **assumptions** of the methods.

There are plenty of resources online for learning more about these methods.

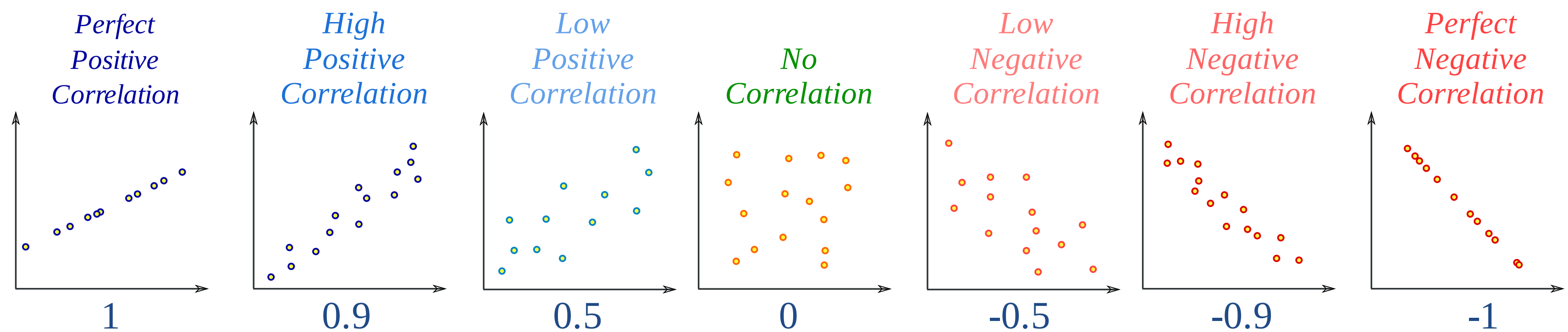
Check out www.opencasestudies.org for deeper dives on some of the concepts covered here and the resource page for more resources.

Correlation

Correlation

The correlation coefficient is a summary statistic that measures the strength of a linear relationship between two numeric variables.

- The strength of the relationship - based on how well the points form a line
- The direction of the relationship - based on if the points progress upward or downward



source

See this case study for more information.

Correlation

Function `cor()` computes correlation in R.

```
cor(x, y = NULL, use = c("everything", "complete.obs"),
    method = c("pearson", "kendall", "spearman"))
```

- provide two numeric vectors of the same length (arguments `x`, `y`), or
- provide a data.frame / tibble with numeric columns only
- by default, Pearson correlation coefficient is computed

Correlation test

Function `cor.test()` also computes correlation and tests for association.

```
cor.test(x, y = NULL, alternative(c("two.sided", "less", "greater")),
    method = c("pearson", "kendall", "spearman"))
```

- provide two numeric vectors of the same length (arguments `x`, `y`), or
- provide a data.frame / tibble with numeric columns only
- by default, Pearson correlation coefficient is computed
- alternative values:
    - two.sided means true correlation coefficient is not equal to zero (default)
    - greater means true correlation coefficient is > 0 (positive relationship)
    - less means true correlation coefficient is < 0 (negative relationship)

GUT CHECK!

What class of data do you need to calculate a correlation?

A. Character data

B. Factor data

C. Numeric data

Correlation

Let's look at the dataset of yearly CO2 emissions by country.

```
yearly_co2 <-
  read_csv(file = "https://daseh.org/data/Yearly_CO2_Emissions_1000_tonnes.csv")
```

Correlation for two vectors

First, we create two vectors.

```
# x and y must be numeric vectors
y1980 <- yearly_co2 |> pull(`1980`)
y1985 <- yearly_co2 |> pull(`1985`)
```

Like other functions, if there are `NA`s, you get `NA` as the result. But if you specify `use = "complete.obs"`, then it will give you correlation using the non-missing data.

```
cor(y1980, y1985, use = "complete.obs")
```

```
[1] 0.9936257
```

Correlation coefficient calculation and test

```
cor.test(y1980, y1985)

	Pearson's product-moment correlation

data:  y1980 and y1985
t = 114.59, df = 169, p-value < 0.00000000000000022
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9913844 0.9952853
sample estimates:
      cor 
0.9936257 
```

Broom package

The `broom` package helps make stats results look tidy

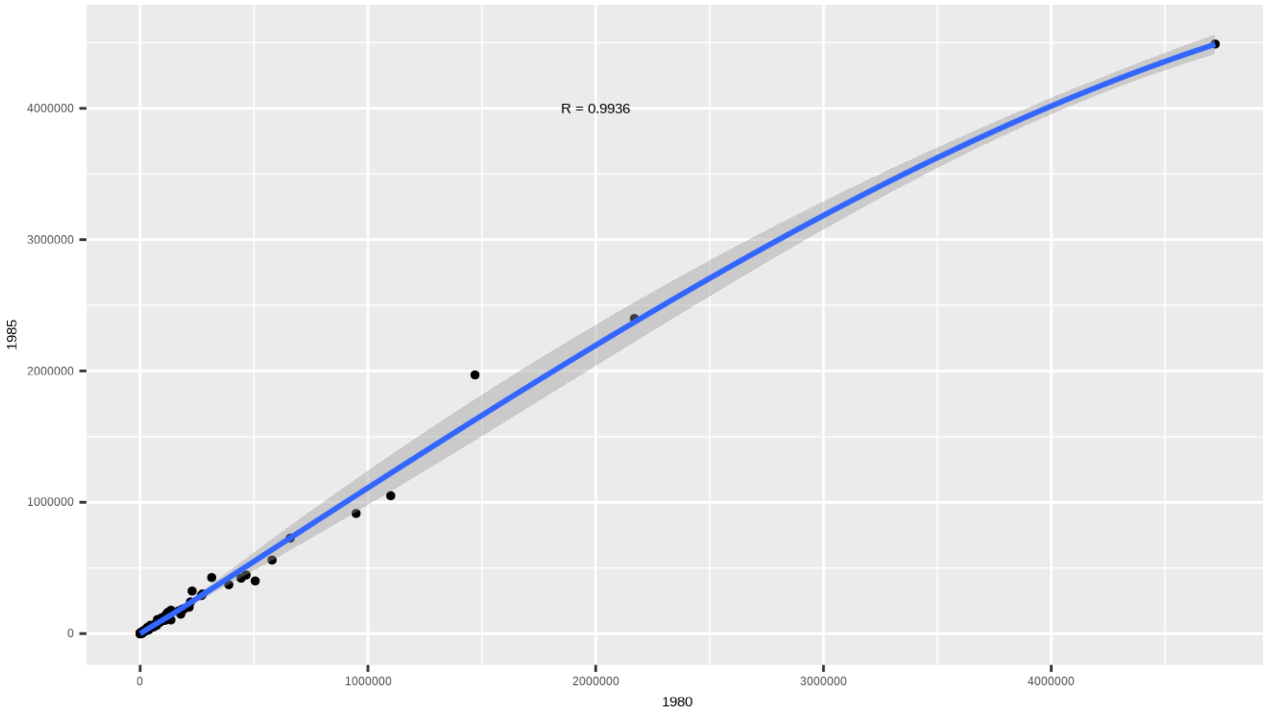
```
library(broom)
cor_result <- tidy(cor.test(y1980, y1985))
glimpse(cor_result)
```

```
Rows: 1
Columns: 8
$ estimate    <dbl> 0.9936257
$ statistic   <dbl> 114.5851
$ p.value     <dbl> 0.000000000000000000000000000000000000000000000000000000000000…
$ parameter   <int> 169
$ conf.low    <dbl> 0.9913844
$ conf.high   <dbl> 0.9952853
$ method      <chr> "Pearson's product-moment correlation"
$ alternative <chr> "two.sided"
```

Correlation for two vectors with plot

In plot form… `geom_smooth()` and `annotate()` can look very nice!

```
corr_value <- pull(cor_result, estimate) |> round(digits = 4)
cor_label <- paste0("R = ", corr_value)
yearly_co2 |>
  ggplot(aes(x = `1980`, y = `1985`)) + geom_point(size = 1) + geom_smooth() +
  annotate("text", x = 2000000, y = 4000000, label = cor_label)
```



Correlation for data frame columns

We can compute correlation for all pairs of columns of a data frame / matrix. This is often called, *“computing a correlation matrix”*.

Columns must be all numeric!

```
co2_subset <- yearly_co2 |>
  select(c(`1950`, `1980`, `1985`, `2010`))

head(co2_subset)
```

```
# A tibble: 6 × 4
  `1950` `1980` `1985` `2010`
   <dbl>  <dbl>  <dbl>  <dbl>
1   84.3   1760   3510   8460
2  297     5170   7880   4600
3 3790    66500  72800 119000
4   NA       NA     NA    517
5  187     5350   4700  29100
6   NA      143    249    524
```

Correlation for data frame columns

We can compute correlation for all pairs of columns of a data frame / matrix. This is often called, *“computing a correlation matrix”*.

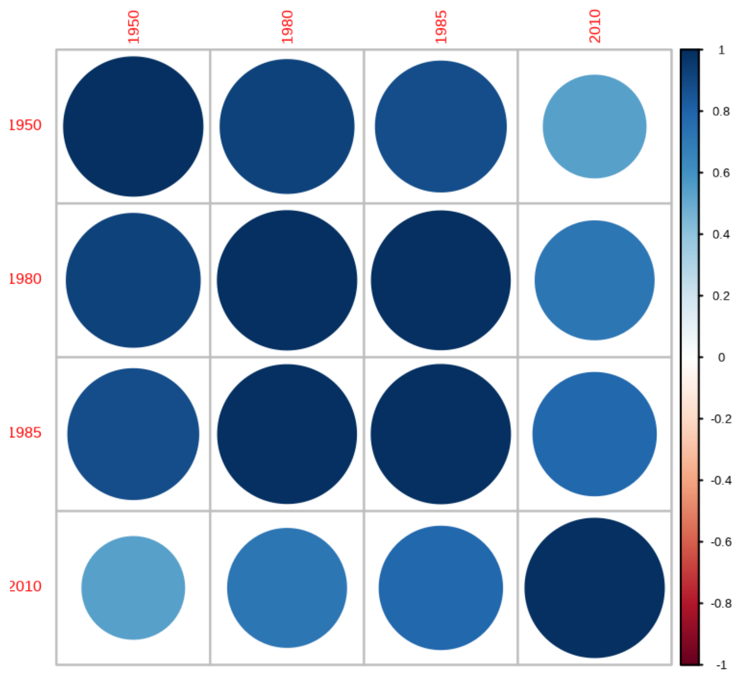
```
cor_mat <- cor(co2_subset, use = "complete.obs")
cor_mat
```

```
          1950      1980      1985      2010
1950 1.0000000 0.9228253 0.8818288 0.5415047
1980 0.9228253 1.0000000 0.9935477 0.7270839
1985 0.8818288 0.9935477 1.0000000 0.7827256
2010 0.5415047 0.7270839 0.7827256 1.0000000
```

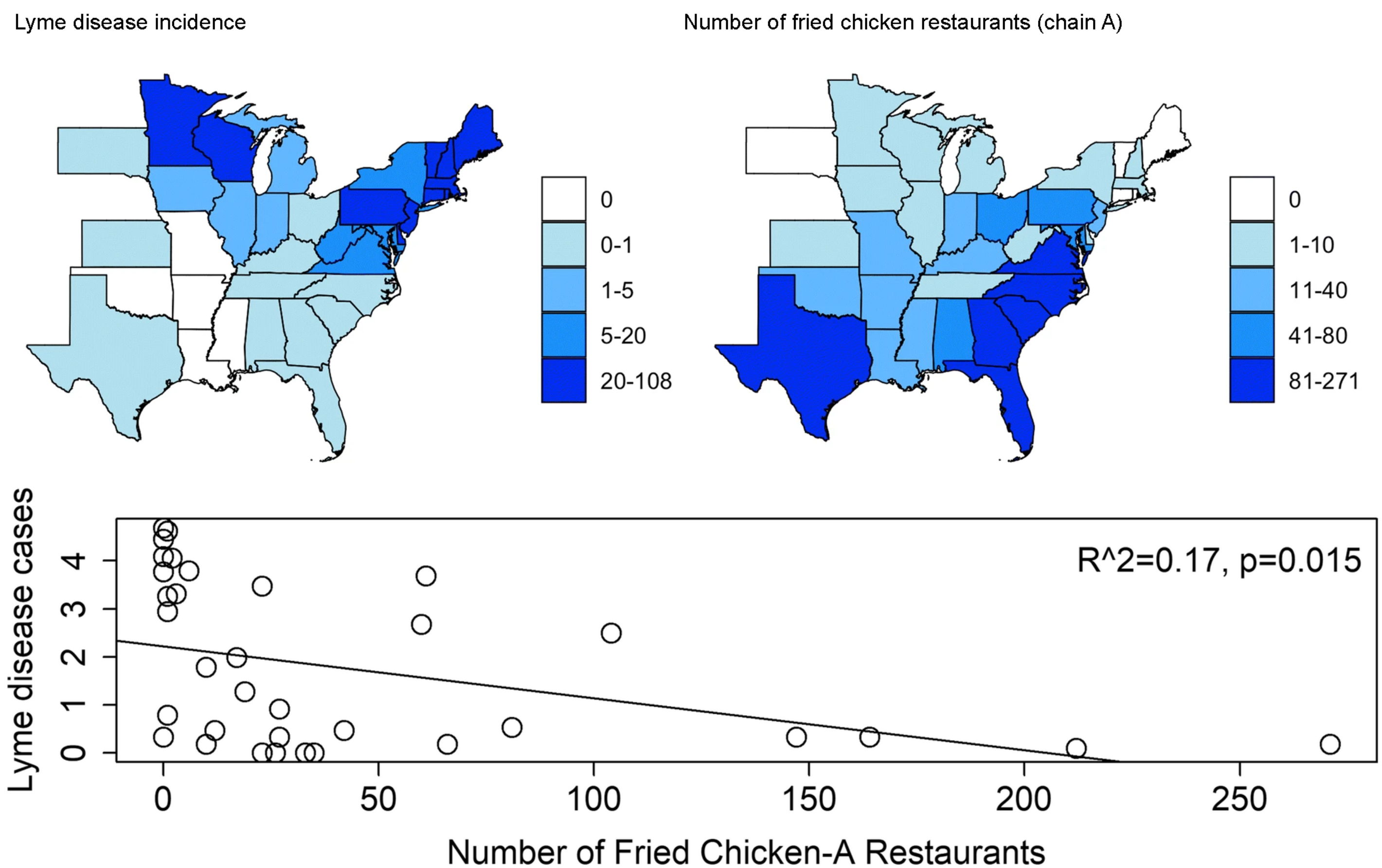
Correlation for data frame columns with plot

`corrplot` package can make correlation matrix plots

```
library(corrplot)
corrplot(cor_mat)
```



Correlation does not imply causation



source

T-test

T-test

The commonly used t-tests are:

- **one-sample t-test** – used to test mean of a variable in one group
- **two-sample t-test** – used to test difference in means of a variable between two groups
    - if the “two groups” are data of the *same* individuals collected at 2 time points, we say it is two-sample paired t-test)

The `t.test()` function does both.

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

Running one-sample t-test

It tests the mean of a variable in one group. By default (i.e., without us explicitly specifying values of other arguments):

- tests whether a mean of a variable is equal to 0 (`mu = 0`)
- uses “two sided” alternative (`alternative = "two.sided"`)
- returns result assuming confidence level 0.95 (`conf.level = 0.95`)
- omits `NA` values in data

Let’s look at the CO2 emissions data again.

```
t.test(y1980)
```

```

    One Sample t-test

data:  y1980
t = 3.3324, df = 170, p-value = 0.001056
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  44745.81 174792.25
sample estimates:
mean of x
   109769
```

Running two-sample t-test

It tests the difference in means of a variable between two groups. By default:

- tests whether difference in means of a variable is equal to 0 (`mu = 0`)
- uses "two sided" alternative (`alternative = "two.sided"`)
- returns result assuming confidence level 0.95 (`conf.level = 0.95`)
- assumes data are not paired (`paired = FALSE`)
- assumes true variance in the two groups is not equal (`var.equal = FALSE`)
- omits `NA` values in data

Check out this this case study and this case study for more information.

Running two-sample t-test in R

```
t.test(y1980, y1985)

    Welch Two Sample t-test

data:  y1980 and y1985
t = -0.090533, df = 341, p-value = 0.9279
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -95902.79  87462.97
sample estimates:
mean of x mean of y 
 109769.0  113988.9 
```

T-test: retrieving information from the result with `broom` package

The `broom` package has a `tidy()` function that can organize results into a data frame so that they are easily manipulated (or nicely printed)

```
result <- t.test(y1980, y1985)
result_tidy <- tidy(result)
glimpse(result_tidy)
```

```
Rows: 1
Columns: 10
$ estimate    <dbl> -4219.909
$ estimate1   <dbl> 109769
$ estimate2   <dbl> 113988.9
$ statistic   <dbl> -0.09053303
$ p.value     <dbl> 0.9279168
$ parameter   <dbl> 340.999
$ conf.low    <dbl> -95902.79
$ conf.high   <dbl> 87462.97
$ method      <chr> "Welch Two Sample t-test"
$ alternative <chr> "two.sided"
```

P-value adjustment

You run an increased risk of Type I errors (a “false positive”) when multiple hypotheses are tested simultaneously.

Use the `p.adjust()` function on a vector of p values. Use `method =` to specify the adjustment method:

```
my_pvalues <- c(0.049, 0.001, 0.31, 0.00001)
p.adjust(my_pvalues, method = "BH") # Benjamini Hochberg
```

```
[1] 0.06533333 0.00200000 0.31000000 0.00004000
```

```
p.adjust(my_pvalues, method = "bonferroni") # multiply by number of tests
```

```
[1] 0.19600 0.00400 1.00000 0.00004
```

```
my_pvalues * 4
```

```
[1] 0.19600 0.00400 1.24000 0.00004
```

See here for more about multiple testing correction. Bonferroni also often done as p value threshold divided by number of tests (0.05/test number).

Some other statistical tests

- `wilcox.test()` – Wilcoxon signed rank test, Wilcoxon rank sum test
- `shapiro.test()` – Test normality assumptions
- `ks.test()` – Kolmogorov-Smirnov test
- `var.test()`– Fisher's F-Test
- `chisq.test()` – Chi-squared test
- `aov()` – Analysis of Variance (ANOVA)

Summary

- Use `cor()` to calculate correlation between two vectors, `cor.test()` can give more information.
- `corrplot()` is nice for a quick visualization!
- `t.test()` one sample test to test the difference in mean of a single vector from zero (one input)
- `t.test()` two sample test to test the difference in means between two vectors (two inputs)
- `tidy()` in the `broom` package is useful for organizing and saving statistical test output
- Remember to adjust p-values with `p.adjust()` when doing multiple tests on data

Lab Part 1

- Class Website
- Lab

Regression

Linear regression

Linear regression is a method to model the relationship between a response and one or more explanatory variables.

Most commonly used statistical tests are actually specialized regressions, including the two sample t-test, see here for more.

Linear regression notation

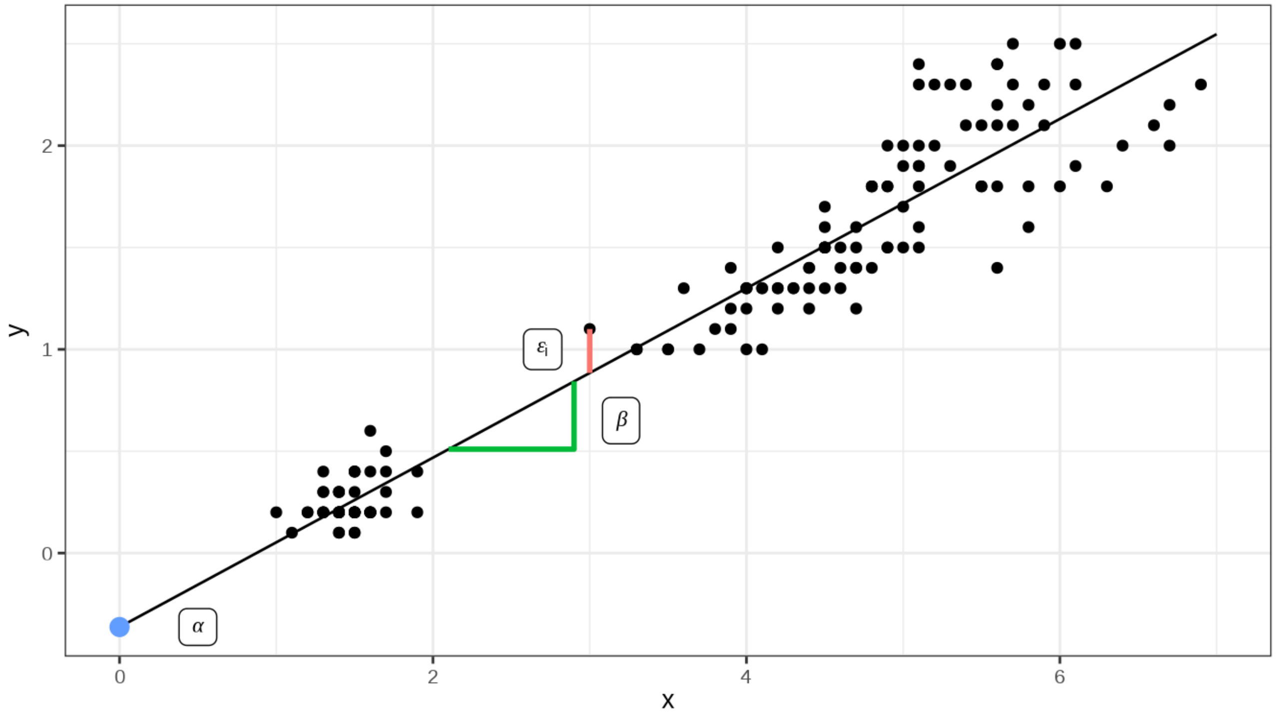
Here is some of the notation, so it is easier to understand the commands/results.

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

where:

- y_i is the outcome for person i
- α is the intercept
- β is the slope (also called a coefficient) - the mean change in y that we would expect for one unit change in x (“rise over run”)
- x_i is the predictor for person i
- ε_i is the residual variation for person i

Linear regression



Linear regression

Linear regression is a method to model the relationship between a response and one or more explanatory variables.

We provide a little notation here so some of the commands are easier to put in the proper context.

$$y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \varepsilon_i$$

where:

- y_i is the outcome for person i
- α is the intercept
- β_1, β_2, β_2 are the slopes/coefficients for variables x_{i1}, x_{i2}, x_{i3} - average difference in y for a unit change (or each value) in x while accounting for other variables
- x_{i1}, x_{i2}, x_{i3} are the predictors for person i
- ε_i is the residual variation for person i

See this case study for more details.

Linear regression fit in R

To fit regression models in R, we use the function `glm()` (Generalized Linear Model).

You may also see `lm()` which is a more limited function that only allows for normally/Gaussian distributed error terms (aka typical linear regressions).

We typically provide two arguments:

- `formula` – model formula written using names of columns in our data
- `data` – our data frame

Linear regression fit in R: model formula

Model formula

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

In R translates to

```
y ~ x
```

Linear regression fit in R: model formula

Model formula

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

In R translates to

```
y ~ x
```

In practice, `y` and `x` are replaced with the **names of columns from our data set**.

For example, if we want to fit a regression model where outcome is `income` and predictor is `years_of_education`, our formula would be:

```
income ~ years_of_education
```

Linear regression fit in R: model formula

Model formula

$$y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \varepsilon_i$$

In R translates to

```
y ~ x1 + x2 + x3
```

In practice, `y` and `x1`, `x2`, `x3` are replaced with the **names of columns from our data set**.

For example, if we want to fit a regression model where outcome is `income` and predictors are `years_of_education`, `age`, and `location` then our formula would be:

```
income ~ years_of_education + age + location
```

Linear regression example

Let’s look variables that might be able to predict the number of heat-related ER visits in Colorado.

We’ll use the dataset that has ER visits separated out by age category.

We’ll combine this with a new dataset that has some weather information about summer temperatures in Denver, downloaded from https://www.weather.gov/bou/DenverSummerHeat.

We will use this as a proxy for temperatures for the state of CO as a whole for this example.

Linear regression example

```
er <- read_csv(file = "https://daseh.org/data/CO_ER_heat_visits_by_age.csv")

temps <- read_csv(file = "https://daseh.org/data/Denver_heat_data.csv")

er_temps <- full_join(er, temps)
er_temps
```

```
# A tibble: 60 × 8
    year age      rate lower95cl upper95cl visits highest_temp no_days_above_100
   <dbl> <chr>   <dbl>     <dbl>     <dbl>  <dbl>        <dbl>             <dbl>
 1  2011 0-4 ye…  3.52      1.82      6.16     12           98                 0
 2  2011 15-34 …  7.34      5.95      8.74    106           98                 0
 3  2011 35-64 …  5.84      4.80      6.88    121           98                 0
 4  2011 5-14 y…  5.20      3.50      6.90     36           98                 0
 5  2011 65+ ye…  8.34      5.98     10.7      48           98                 0
 6  2012 0-4 ye…  3.58      1.85      6.25     12          105                13
 7  2012 15-34 …  8.88      7.36     10.4     130          105                13
 8  2012 35-64 …  5.81      4.77      6.84    121          105                13
 9  2012 5-14 y…  4.14      2.63      5.64     29          105                13
10  2012 65+ ye…  7.69      5.49      9.88     47          105                13
# ℹ 50 more rows
```

Linear regression: model fitting

For this model, we will use two variables:

- **visits** - number of visits to the ER for heat-related illness
- **highest_temp** - the highest recorded temperature of the summer

```
fit <- glm(visits ~ highest_temp, data = er_temps)
fit
```

```

Call:  glm(formula = visits ~ highest_temp, data = er_temps)

Coefficients:
 (Intercept)  highest_temp
     -32.497         1.134

Degrees of Freedom: 52 Total (i.e. Null);  51 Residual
  (7 observations deleted due to missingness)
Null Deviance:      155200
Residual Deviance: 154800   AIC: 579.3
```

Linear regression: model summary

The `summary()` function returns a list that shows us some more detail

```
summary(fit)
```

```
Call:
glm(formula = visits ~ highest_temp, data = er_temps)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   -32.497    336.802  -0.096    0.924
highest_temp    1.134      3.328   0.341    0.735

(Dispersion parameter for gaussian family taken to be 3035.262)

    Null deviance: 155151  on 52  degrees of freedom
Residual deviance: 154798  on 51  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 579.33

Number of Fisher Scoring iterations: 2
```

tidy results

The broom package can help us here too!

The estimate is the coefficient or slope.

for every 1 degree increase in the highest temperature, we see 1.134 more heat-related ER visits. The error for this estimate is pretty big at 3.328. This relationship appears to be insignificant with a p value = 0.735.

```
tidy(fit) |> glimpse()
```

```
Rows: 2
Columns: 5
$ term      <chr> "(Intercept)", "highest_temp"
$ estimate  <dbl> -32.496558, 1.134499
$ std.error <dbl> 336.802026, 3.327615
$ statistic <dbl> -0.09648564, 0.34093451
$ p.value   <dbl> 0.9235130, 0.7345535
```

Linear regression: multiple predictors

Let's try adding another other explanatory variable to our model, year (`year`).

```
fit2 <- glm(visits ~ highest_temp + year, data = er_temps)
summary(fit2)
```

```
Call:
glm(formula = visits ~ highest_temp + year, data = er_temps)

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -9569.3739  4322.3126  -2.214   0.0314 *
highest_temp    -0.1764     3.2616  -0.054   0.9571
year             4.7959     2.1675   2.213   0.0315 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 2819.85)

    Null deviance: 155151  on 52  degrees of freedom
Residual deviance: 140993  on 50  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 576.37

Number of Fisher Scoring iterations: 2
```

Linear regression: multiple predictors

Can also use `tidy` and `glimpse` to see the output nicely.

```
fit2 |>
  tidy() |>
  glimpse()
```

```
Rows: 3
Columns: 5
$ term      <chr> "(Intercept)", "highest_temp", "year"
$ estimate  <dbl> -9569.373863, -0.176384, 4.795897
$ std.error <dbl> 4322.312598, 3.261619, 2.167462
$ statistic <dbl> -2.21394766, -0.05407867, 2.21267889
$ p.value   <dbl> 0.03142154, 0.95708799, 0.03151445
```

Linear regression: factors

Factors get special treatment in regression models - lowest level of the factor is the comparison group, and all other factors are **relative** to its values.

Let's add age category (`age`) as a factor into our model. We'll need to convert it to a factor first.

```
er_temps <- er_temps |> mutate(age = factor(age))
```

Linear regression: factors

The comparison group that is not listed is treated as intercept. All other estimates are relative to the intercept.

```
fit3 <- glm(visits ~ highest_temp + year + age, data = er_temps)
summary(fit3)
```

```
Call:
glm(formula = visits ~ highest_temp + year + age, data = er_temps)

Coefficients:
                 Estimate Std. Error t value            Pr(>|t|)    
(Intercept)    -6249.8166  1682.3061  -3.715            0.000549 ***
highest_temp       1.6330     1.2623   1.294            0.202244    
year               3.0263     0.8461   3.577            0.000833 ***
age15-34 years   114.3610    10.3790  11.019 0.00000000000001703 ***
age35-64 years   119.0277    10.3790  11.468 0.00000000000000438 ***
age5-14 years     14.0464    10.4803   1.340            0.186743    
age65+ years      42.1110    10.3790   4.057            0.000190 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 410.7963)

    Null deviance: 155151  on 52  degrees of freedom
Residual deviance:  18897  on 46  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 477.86

Number of Fisher Scoring iterations: 2
```

Linear regression: factors

Maybe we want to use the age group “65+ years” as our reference. We can relevel the factor.

The ages are relative to the level that is not listed.

```
er_temps <-
  er_temps |>
  mutate(age = factor(age,
    levels = c("65+ years", "35-64 years", "15-34 years", "5-14 years", "0-4 years")
  ))

fit4 <- glm(visits ~ highest_temp + year + age, data = er_temps)
summary(fit4)
```

```

Call:
glm(formula = visits ~ highest_temp + year + age, data = er_temps)

Coefficients:
                 Estimate Std. Error t value         Pr(>|t|)    
(Intercept)    -6207.7056  1684.1411  -3.686         0.000599 ***
highest_temp       1.6330     1.2623   1.294         0.202244    
year               3.0263     0.8461   3.577         0.000833 ***
age35-64 years    76.9167     8.2744   9.296 0.00000000000394 ***
age15-34 years    72.2500     8.2744   8.732 0.00000000002527 ***
age5-14 years    -28.0646     8.4647  -3.315         0.001791 ** 
age0-4 years     -42.1110    10.3790  -4.057         0.000190 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 410.7963)

    Null deviance: 155151  on 52  degrees of freedom
Residual deviance:  18897  on 46  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 477.86
```

Linear regression: factors

You can view estimates for the comparison group by removing the intercept in the GLM formula

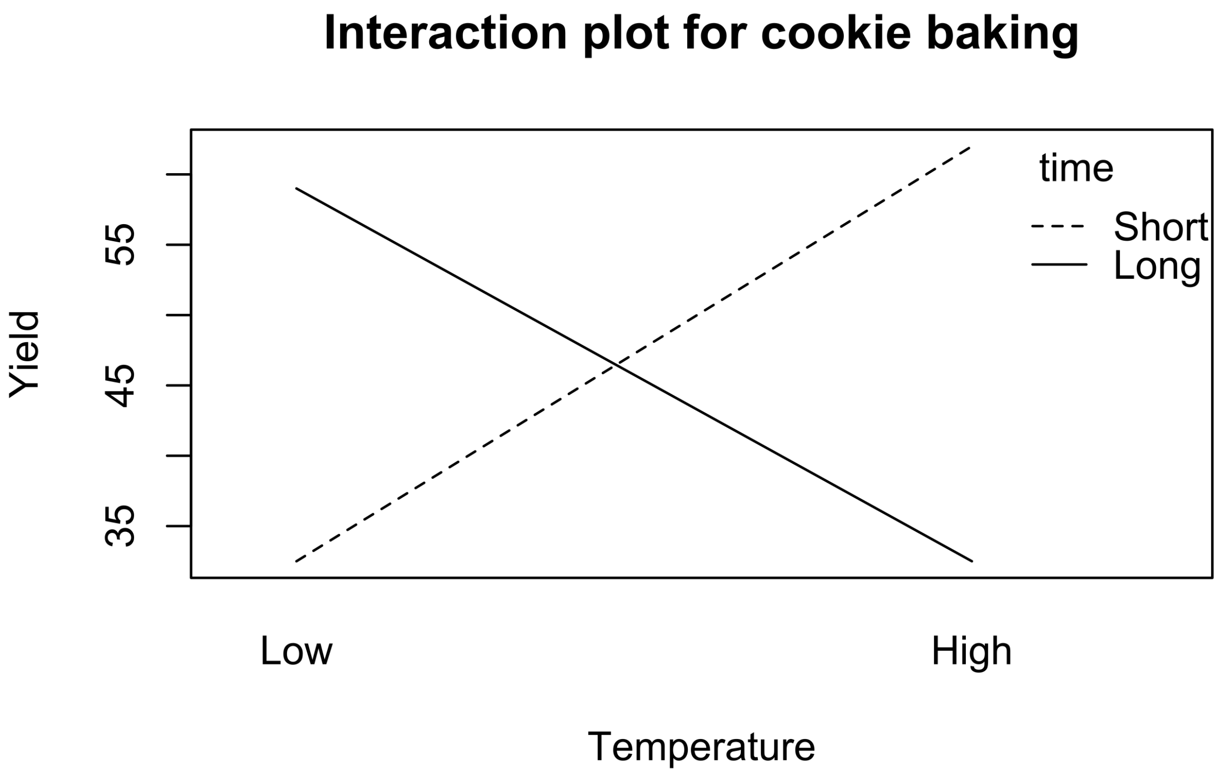
```
y ~ x - 1
```

Caveat is that the p-values change, and interpretation is often confusing.

```
fit5 <- glm(visits ~ highest_temp + year + age - 1, data = er_temps)
summary(fit5)
```

```

Call:
glm(formula = visits ~ highest_temp + year + age - 1, data = er_temps)

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
highest_temp        1.6330     1.2623   1.294 0.202244    
year                3.0263     0.8461   3.577 0.000833 ***
age65+ years    -6207.7056  1684.1411  -3.686 0.000599 ***
age35-64 years  -6130.7889  1684.1411  -3.640 0.000688 ***
age15-34 years  -6135.4556  1684.1411  -3.643 0.000682 ***
age5-14 years   -6235.7702  1683.8723  -3.703 0.000569 ***
age0-4 years    -6249.8166  1682.3061  -3.715 0.000549 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 410.7963)

    Null deviance: 514152  on 53  degrees of freedom
Residual deviance:  18897  on 46  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 477.86

Number of Fisher Scoring iterations: 2
```

Linear regression: interactions



source

Linear regression: interactions

You can also specify interactions between variables in a formula `y ~ x1 + x2 + x1 * x2`. This allows for not only the intercepts between factors to differ, but also the slopes with regard to the interacting variable.

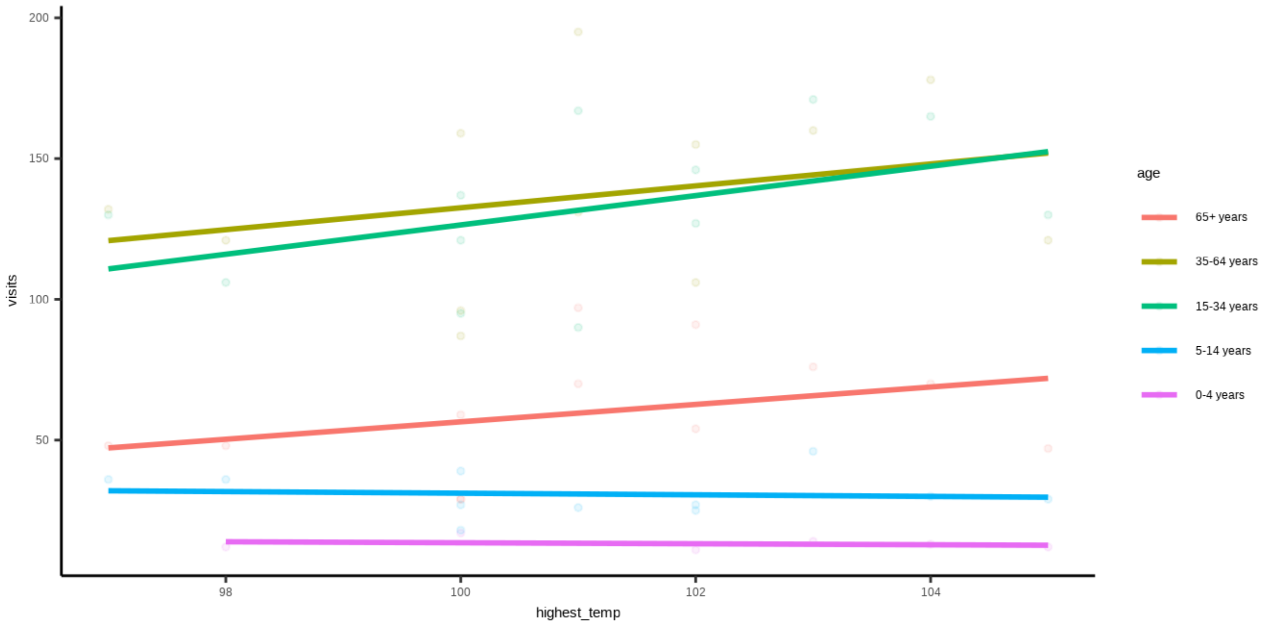
```
fit6 <- glm(visits ~ highest_temp + year + age + age*highest_temp, data = er_temps
)
tidy(fit6)
```

```
# A tibble: 11 × 5
   term                        estimate std.error statistic  p.value
   <chr>                          <dbl>     <dbl>     <dbl>    <dbl>
 1 (Intercept)                -6373.     1716.      -3.71   0.000597
 2 highest_temp                   2.23      2.67     0.836  0.408
 3 year                           3.08      0.853    3.61   0.000809
 4 age35-64 years                -3.86    380.      -0.0102 0.992
 5 age15-34 years              -142.      380.      -0.373  0.711
 6 age5-14 years                315.      380.       0.829  0.412
 7 age0-4 years                 369.      447.       0.825  0.414
 8 highest_temp:age35-64 years    0.799     3.76     0.213  0.833
 9 highest_temp:age15-34 years    2.12      3.76     0.564  0.576
10 highest_temp:age5-14 years    -3.39      3.76    -0.903  0.371
11 highest_temp:age0-4 years     -4.04      4.40    -0.918  0.364
```

Linear regression: interactions

By default, `ggplot` with a factor added as a color will look include the interaction term. Notice the different intercept and slope of the lines.

```
ggplot(er_temps, aes(x = highest_temp, y = visits, color = age)) +
  geom_point(size = 1, alpha = 0.1) +
  geom_smooth(method = "glm", se = FALSE) +
  theme_classic()
```



Generalized linear models (GLMs)

Generalized linear models (GLMs) allow for fitting regressions for non-continuous/normal outcomes. Examples include: logistic regression, Poisson regression.

Add the `family` argument – a description of the error distribution and link function to be used in the model. These include:

- `binomial(link = "logit")` - outcome is binary
- `poisson(link = "log")` - outcome is count or rate
- others

Very important to use the right test!

See this case study for more information.

See `?family` documentation for details of family functions.

Logistic regression

Let's look at a logistic regression example. We'll use the `er_temps` dataset again.

We will create a new binary variable `high_rate`. We will say a visit rate of more than 8 qualifies as a high visit rate.

```
er_temps <-
  er_temps |> mutate(high_rate = rate > 8)

glimpse(er_temps)
```

```
Rows: 60
Columns: 9
$ year             <dbl> 2011, 2011, 2011, 2011, 2011, 2012, 2012, 2012, 2012…
$ age              <fct> 0-4 years, 15-34 years, 35-64 years, 5-14 years, 65+…
$ rate             <dbl> 3.523598, 7.344520, 5.840845, 5.197288, 8.341661, 3.…
$ lower95cl        <dbl> 1.820694, 5.946380, 4.800143, 3.499551, 5.981890, 1.…
$ upper95cl        <dbl> 6.155016, 8.742659, 6.881547, 6.895024, 10.701431, 6…
$ visits           <dbl> 12, 106, 121, 36, 48, 12, 130, 121, 29, 47, 17, 121,…
$ highest_temp     <dbl> 98, 98, 98, 98, 98, 105, 105, 105, 105, 105, 100, 10…
$ no_days_above_100 <dbl> 0, 0, 0, 0, 0, 13, 13, 13, 13, 13, 2, 2, 2, 2, 2, 1,…
$ high_rate        <lgl> FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE, FALSE…
```

Logistic regression

Let's explore how `highest_temp`, `year`, and `age` might predict `high_rate`.

```
# General format
glm(y ~ x, data = DATASET_NAME, family = binomial(link = "logit"))
```

```
binom_fit <- glm(high_rate ~ highest_temp + year + age, data = er_temps, family = binomial(link = "logit"))
summary(binom_fit)
```

```
Call:
glm(formula = high_rate ~ highest_temp + year + age, family = binomial(link = "logit"),
    data = er_temps)

Coefficients:
                Estimate Std. Error z value Pr(>|z|)  
(Intercept)    -508.6008   259.3254  -1.961   0.0499 *
highest_temp      0.2187     0.1915   1.142   0.2536  
year              0.2412     0.1284   1.878   0.0604 .
age35-64 years   -1.8949     1.0624  -1.784   0.0745 .
age15-34 years    0.8707     0.9537   0.913   0.3613  
age5-14 years   -19.7694  3008.3832  -0.007   0.9948  
age0-4 years    -19.5166  4117.4125  -0.005   0.9962  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 64.920  on 52  degrees of freedom
Residual deviance: 36.529  on 46  degrees of freedom
  (7 observations deleted due to missingness)
AIC: 50.529

Number of Fisher Scoring iterations: 18
```

Logistic Regression

See this case study for more information.

Odds ratios

An odds ratio (OR) is a measure of association between an exposure and an outcome. The OR represents the odds that an outcome will occur given a particular exposure, compared to the odds of the outcome occurring in the absence of that exposure.

Check out this paper.

Use `oddsratio(x, y)` from the `epitools()` package to calculate odds ratios.

Odds ratios

During the years 2012, 2018, 2021, and 2022, there were multiple consecutive days with temperatures above 100 degrees. We will code this as `heatwave`.

```
library(epitools)

er_temps <-
  er_temps |>
  mutate(heatwave = year %in% c(2012, 2018, 2021, 2022))

glimpse(er_temps)
```

```
Rows: 60
Columns: 10
$ year             <dbl> 2011, 2011, 2011, 2011, 2011, 2012, 2012, 2012, 2012…
$ age              <fct> 0-4 years, 15-34 years, 35-64 years, 5-14 years, 65+…
$ rate             <dbl> 3.523598, 7.344520, 5.840845, 5.197288, 8.341661, 3.…
$ lower95cl        <dbl> 1.820694, 5.946380, 4.800143, 3.499551, 5.981890, 1.…
$ upper95cl        <dbl> 6.155016, 8.742659, 6.881547, 6.895024, 10.701431, 6…
$ visits           <dbl> 12, 106, 121, 36, 48, 12, 130, 121, 29, 47, 17, 121,…
$ highest_temp     <dbl> 98, 98, 98, 98, 98, 105, 105, 105, 105, 105, 100, 10…
$ no_days_above_100 <dbl> 0, 0, 0, 0, 0, 13, 13, 13, 13, 13, 2, 2, 2, 2, 2, 1,…
$ high_rate        <lgl> FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE, FALSE…
$ heatwave         <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE,…
```

Odds ratios

In this case, we're calculating the odds ratio for whether a heatwave is associated with having a visit rate greater than 8.

```
response <- er_temps |> pull(high_rate)
predictor <- er_temps |> pull(heatwave)

oddsratio(predictor, response)
```

```
$data
         Outcome
Predictor FALSE TRUE Total
    FALSE    28    7    35
    TRUE      9    9    18
    Total    37   16    53

$measure
         odds ratio with 95% C.I.
Predictor estimate    lower    upper
    FALSE 1.000000       NA       NA
    TRUE  3.855878 1.113406 14.22934

$p.value
         two-sided
Predictor midp.exact fisher.exact chi.square
    FALSE         NA           NA         NA
    TRUE   0.0331246   0.03178645 0.02425672

$correction
[1] FALSE

attr(,"method")
[1] "median-unbiased estimate & mid-p exact CI"
```

Odds ratios

The Odds Ratio is 3.86.

When the predictor is TRUE (aka it was a heatwave year), the odds of the response (high hospital visitation) are 3.86 times greater than when it is FALSE (not a heatwave year).

```
$data
         Outcome
Predictor FALSE TRUE Total
    FALSE    28    7    35
    TRUE      9    9    18
    Total    37   16    53

$measure
         odds ratio with 95% C.I.
Predictor estimate    lower    upper
    FALSE 1.000000       NA       NA
    TRUE  3.855878 1.113406 14.22934

$p.value
         two-sided
Predictor midp.exact fisher.exact chi.square
    FALSE         NA           NA         NA
    TRUE   0.0331246   0.03178645 0.02425672

$correction
[1] FALSE

attr(,"method")
[1] "median-unbiased estimate & mid-p exact CI"
```

Final note

Some final notes:

- Researcher's responsibility to **understand the statistical method** they use – underlying assumptions, correct interpretation of method results
- Researcher's responsibility to **understand the R software** they use – meaning of function's arguments and meaning of function's output elements

Summary

- `glm()` fits regression models:
  - Use the `formula =` argument to specify the model (e.g., `y ~ x` or `y ~ x1 + x2` using column names)
  - Use `data =` to indicate the dataset
  - Use `family =` to do a other regressions like logistic, Poisson and more
  - `summary()` gives useful statistics
- `oddsratio()` from the `epitools` package can calculate odds ratios (outside of logistic regression - which allows more than one explanatory variable)
- this is just the tip of the iceberg!

Resources (also on the website!)

For more check out:

- this chapter on modeling in this tidyverse book
- this chart on when to do what test
- opencasestudies.org

Content for similar topics as this course can also be found on Leanpub.

Lab Part 2

- Class Website
- Lab
- Day 8 Cheatsheet

Image by Gerd Altmann from Pixabay

Extra Slides

Wilcoxon Test

The Wilcoxon test is a good alternative to the t-test when the normal distribution of the differences between paired individuals cannot be assumed.

```
wilcox.test(x, y, ..)
```

- Like t-test, provide one or two vectors (x, y)
- Choose from `alternative = c("two.sided", "less", "greater")`
- Use `paired = TRUE` for paired values (e.g., before and after)

Shapiro Test

Can tell you if a vector is normally distributed.

```
shapiro.test(x)
```

The smaller the p-value, the more likely the data violates normality assumptions.

Kolmogorov-Smirnov test

Can tell you if two groups come from different distributions.

```
ks.test()
```

The smaller the p-value, the more likely the data are from different distributions.

Fisher's F-Test

Performs an F test to compare the variances of two samples from normal populations.

```
var.test()
```

Chi-squared test

For categorical data/ratios, can tell you if observations match expected values or if two categorical variables are independent.

```
chisq.test()
```

Analysis of Variance (ANOVA)

For balanced designs, determine if multiple variables influence a dependent variable. “Within versus among group variance”.

```
aov()
```