Intro to R

Statistics

Summary

- `ggplot()` specifies what data use and what variables will be mapped to where
- inside `ggplot()`, `mapping = 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 `mapping = 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

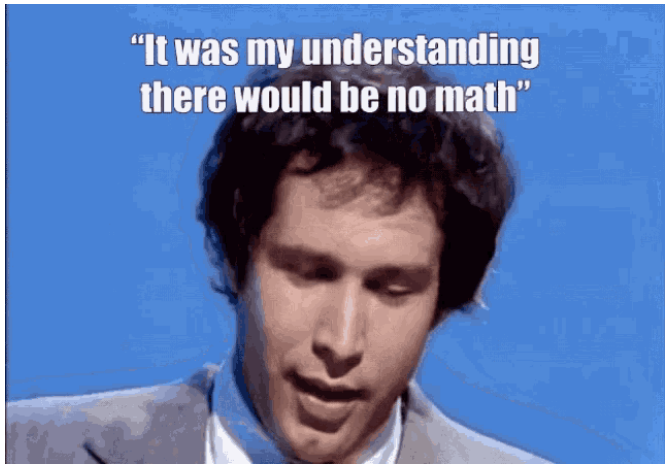
Summary

- 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` , the `as_factor()` (of `forcats` package) or `factor()` function and specifying the levels with the `levels` argument
- the `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



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, as well as dedicated Biostatistics series (at different advancement levels) at the JHU School of Public Health.

Check out www.opencasestudies.org for deep dives on some of the concepts covered here.

Correlation

Correlation

Function `cor()` computes correlation in R

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

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

Correlation

https://jhudatascience.org/intro_to_r/data/Charm_City_Circulator_Ridership.csv

```
library(jhur)
circ <- read_circulator()
head(circ)
```

```
# A tibble: 6 × 15
  day       date  orangeBoardings orangeAlightings orangeAverage purpleBoardin
  <chr>     <chr>           <dbl>            <dbl>         <dbl>           <db
1 Monday    01/1…             877             1027           952
2 Tuesday   01/1…             777              815           796
3 Wednesday 01/1…            1203             1220          1212.
4 Thursday  01/1…            1194             1233          1214.
5 Friday    01/1…            1645             1643          1644
6 Saturday  01/1…            1457             1524          1490.
# … with 9 more variables: purpleAlightings <dbl>, purpleAverage <dbl>,
#   greenBoardings <dbl>, greenAlightings <dbl>, greenAverage <dbl>,
#   bannerBoardings <dbl>, bannerAlightings <dbl>, bannerAverage <dbl>,
#   daily <dbl>
```

Correlation for two vectors

First, we compute correlation by providing two vectors.

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

```
x <- circ %>% pull(orangeAverage)
y <- circ %>% pull(purpleAverage)
```

```
cor(x, y)
```

```
[1] NA
```

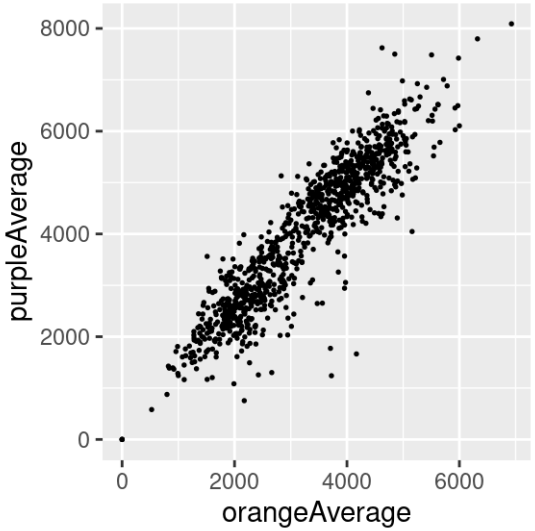
```
cor(x, y, use = "complete.obs")
```

```
[1] 0.9195356
```

Correlation for two vectors with plot

In plot form...

```
circ %>%
  ggplot(aes(x = orangeAverage, y = purpleAverage)) +
  geom_point(size = 0.3)
```



Correlation for data frame columns

We can compute correlation for all pairs of columns of a data frame / matrix. We typically just say, *"compute correlation matrix"*.

Columns must be all numeric!

```
circ_subset_Average <- circ %>% select(ends_with("Average"))
dim(circ_subset_Average)
```

```
[1] 1146    4
```

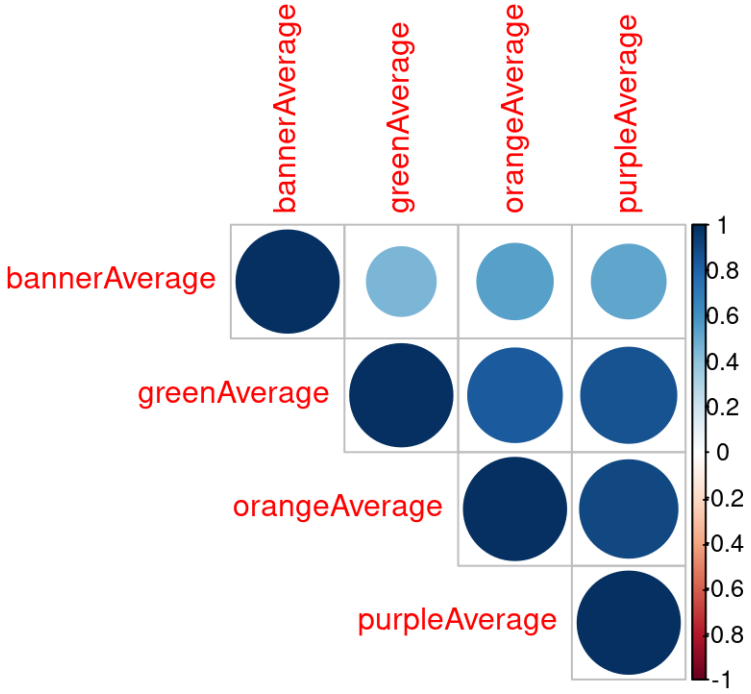
```
cor_mat <- cor(circ_subset_Average, use = "complete.obs")
cor_mat
```

```
              orangeAverage purpleAverage greenAverage bannerAverage
orangeAverage     1.0000000     0.9078826    0.8395806     0.5447031
purpleAverage     0.9078826     1.0000000    0.8665630     0.5213462
greenAverage      0.8395806     0.8665630    1.0000000     0.4533421
bannerAverage     0.5447031     0.5213462    0.4533421     1.0000000
```

Correlation for data frame columns with plot

- Google, *"r correlation matrix plot"*

```
library(corrplot)
corrplot(cor_mat, type = "upper", order = "hclust")
```



T-test

T-test

The commonly used 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 in R is one to address the above.

```
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 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`)

```
x <- circ %>% pull(orangeAverage)
t.test(x)
```

```
    One Sample t-test

data:  x
t = 83.279, df = 1135, p-value < 0.00000000000000022
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 2961.700 3104.622
sample estimates:
mean of x
 3033.161
```

Running two-sample t-test

It tests test 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`)

```
x <- circ %>% pull(orangeAverage)
y <- circ %>% pull(purpleAverage)
t.test(x, y)
```

```
    Welch Two Sample t-test

data:  x and y
t = -17.076, df = 1984, p-value < 0.00000000000000022
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1096.7602  -870.7867
sample estimates:
mean of x mean of y
 3033.161  4016.935
```

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)

```
library(broom)

result <- t.test(x, y)
result_tidy <- tidy(result)
result_tidy
```

```
# A tibble: 1 × 10
  estimate estimate1 estimate2 statistic  p.value parameter conf.low conf.high
     <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>    <dbl>     <dbl>
1    -984.     3033.     4017.     -17.1 4.20e-61     1984.   -1097.     -871.
# … with 2 more variables: method <chr>, alternative <chr>
```

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")
```

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

Some other statistical tests

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

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.

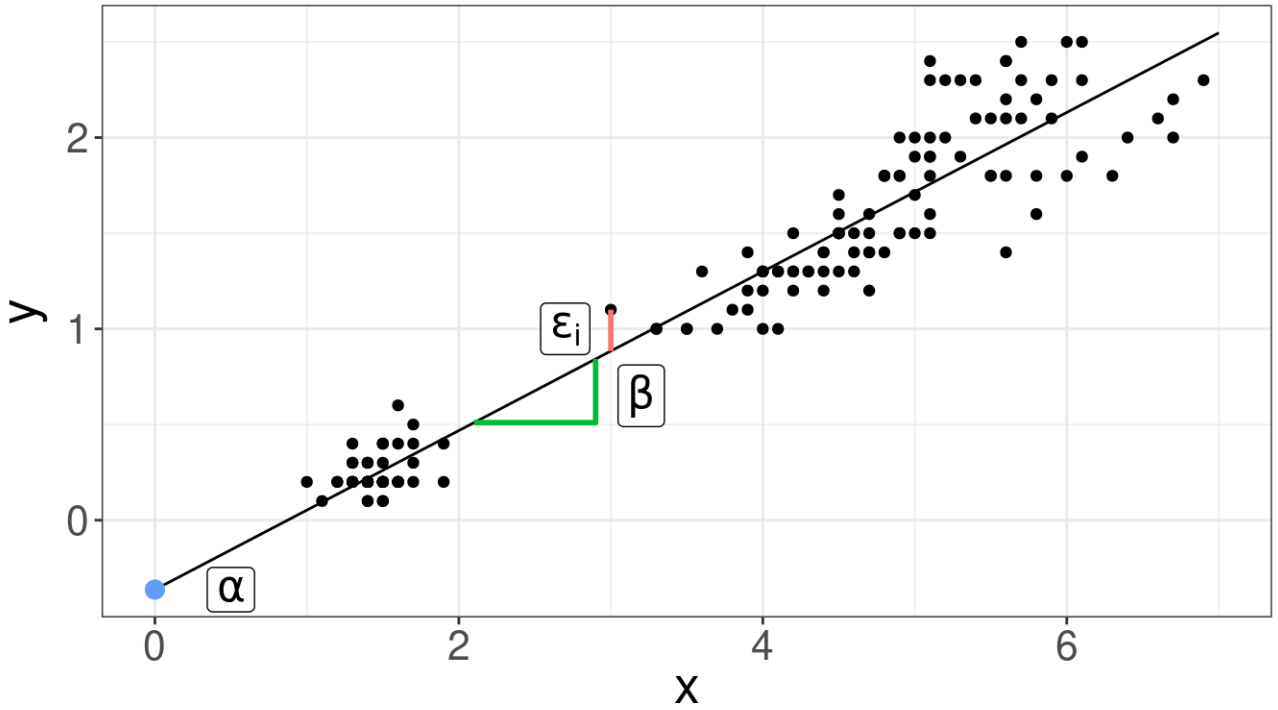
We provide a little notation here so some of the commands are easier to put in the proper context.

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

where:

- y_i is the outcome for person i
- α is the intercept
- β is the slope
- 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 for variables x_{i1}, x_{i2}, x_{i3}
- x_{i1}, x_{i2}, x_{i3} are the predictors for person i
- ε_i is the residual variation for person i

Linear regression fit in R

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

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`, `location` then our formula would be:

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

Linear regression

We will use data about emergency room doctor complaints.

"Data was recorded on 44 doctors working in an emergency service at a hospital to study the factors affecting the number of complaints received."

```
# install.packages("faraway")
library(faraway)

data(esdcomp)
esdcomp
```

```
   visits complaints residency gender revenue   hours
1    2014          2         Y      F  263.03 1287.25
2    3091          3         N      M  334.94 1588.00
3     879          1         Y      M  206.42  705.25
4    1780          1         N      M  226.32 1005.50
5    3646         11         N      M  288.91 1667.25
6    2690          1         N      M  275.94 1517.75
7    1864          2         Y      M  295.71  967.00
8    2782          6         N      M  224.91 1609.25
9    3071          9         N      F  249.32 1747.75
10   1502          3         Y      M  269.00  906.25
11   2438          2         N      F  225.61 1787.75
12   2278          2         N      M  212.43 1480.50
13   2458          5         N      M  211.05 1733.50
14   2269          2         N      F  213.23 1847.25
15   2431          7         N      M  257.30 1433.00
16   3010          2         Y      M  326.49 1520.00
17   2234          5         Y      M  290.53 1404.75
```

Linear regression: model fitting

We fit linear regression model with the number of patient visits (`visits`) as an outcome and total number of hours worked (`hours`) as a predictor.

```
fit <- glm(visits ~ hours, data = esdcomp)
fit
```

```
Call:  glm(formula = visits ~ hours, data = esdcomp)

Coefficients:
(Intercept)        hours
    140.288        1.584

Degrees of Freedom: 43 Total (i.e. Null);  42 Residual
Null Deviance:      16920000
Residual Deviance: 5383000  AIC: 646.3
```

Linear regression: model summary

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

```
summary(fit)
```

```
Call:
glm(formula = visits ~ hours, data = esdcomp)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-797.48  -196.35   -43.15   169.58   864.65

Coefficients:
            Estimate Std. Error t value         Pr(>|t|)
(Intercept)  140.288    242.723   0.578            0.566
hours          1.584      0.167   9.488 0.000000000000526 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 128155.3)

    Null deviance: 16919101  on 43  degrees of freedom
Residual deviance:  5382524  on 42  degrees of freedom
AIC: 646.3

Number of Fisher Scoring iterations: 2
```

Linear regression: multiple predictors

Let's try adding another explanatory variable to our model, dollars per hour earned by the doctor (`revenue``).

```
fit2 <- glm(visits ~ hours + revenue, data = esdcomp)
summary(fit2)
```

```
Call:
glm(formula = visits ~ hours + revenue, data = esdcomp)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-420.57  -166.87   -17.45   140.79   616.20

Coefficients:
             Estimate Std. Error t value             Pr(>|t|)
(Intercept) -2078.1369   327.9157  -6.337        0.00000014326 ***
hours           1.6179     0.1081  14.968 < 0.0000000000000002 ***
revenue         8.3437     1.0828   7.706        0.00000000169 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 53620.97)

    Null deviance: 16919101  on 43  degrees of freedom
Residual deviance:  2198460  on 41  degrees of freedom
AIC: 608.91

Number of Fisher Scoring iterations: 2
```

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.

`residency` takes values Y or N to indicate whether the doctor is a resident.

```
esdcomp %>% count(residency)
```

```
  residency  n
1         N 24
2         Y 20
```

Linear regression: factors

```
fit_3 <- glm(visits ~ residency, data = esdcomp)
summary(fit_3)
```

```
Call:
glm(formula = visits ~ residency, data = esdcomp)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1356.30   -371.55    -62.79    400.46   1527.70

Coefficients:
            Estimate Std. Error t value            Pr(>|t|)
(Intercept)   2510.8      126.3   19.87 <0.0000000000000002 ***
residencyY    -275.5      187.4   -1.47               0.149
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 383122.6)

    Null deviance: 16919101  on 43  degrees of freedom
Residual deviance: 16091148  on 42  degrees of freedom
AIC: 694.49

Number of Fisher Scoring iterations: 2
```

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")`
- `poisson(link = "log")`, and other.

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

Logistic regression

We will use data about breast cancer tumors.

“The purpose of this study was to determine whether a new procedure called fine needle aspiration which draws only a small sample of tissue could be effective in determining tumor status.”

```
data(wbca)
wbca
```

```
    Class Adhes BNucl Chrom Epith Mitos NNucl Thick UShap USize
1       1     1     1     3     2     1     1     5     1     1
2       1     5    10     3     7     1     2     5     4     4
3       1     1     2     3     2     1     1     3     1     1
4       1     1     4     3     3     1     7     6     8     8
5       1     3     1     3     2     1     1     4     1     1
6       0     8    10     9     7     1     7     8    10    10
7       1     1    10     3     2     1     1     1     1     1
8       1     1     1     3     2     1     1     2     2     1
9       1     1     1     1     2     5     1     2     1     1
10      1     1     1     2     2     1     1     4     1     2
11      1     1     1     3     1     1     1     1     1     1
12      1     1     1     2     2     1     1     2     1     1
13      0     3     3     4     2     1     4     5     3     3
14      1     1     3     3     2     1     1     1     1     1
15      0    10     9     5     7     4     5     8     5     7
16      0     4     1     4     6     1     3     7     6     4
17      1     1     1     2     2     1     1     4     1     1
18      1     1     1     3     2     1     1     4     1     1
19      0     6    10     4     4     2     1    10     7     7
20      1     1     1     3     2     1     1     6     1     1
21      0    10    10     5     5     4     4     7     2     3
22      0     3     7     7     6     1    10    10     5     5
23      1     1     1     2     2     1     1     3     1     1
24      1     1     1     3     2     1     1     1     1     1
25      0     4     7     3     2     1     6     5     3     2
26      1     1     1     2     1     1     1     3     1     2
27      1     1     1     2     2     1     1     5     1     1
```

Logistic regression

`Class` is a 0/1-valued variable indicating if the tumor was malignant (0 if malignant, 1 if benign).

```
binom_fit <- glm(Class ~ UShap + USize, data = wbca, family = binomial())
summary(binom_fit)
```

```
Call:
glm(formula = Class ~ UShap + USize, family = binomial(), data = wbca)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.8262  -0.0171   0.1929   0.1929   4.0082

Coefficients:
            Estimate Std. Error z value             Pr(>|z|)
(Intercept)   5.6868     0.4359  13.047 < 0.0000000000000002 ***
UShap        -0.8431     0.1593  -5.292          0.000000121 ***
USize        -0.8686     0.1690  -5.139          0.000000277 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 881.39  on 680  degrees of freedom
Residual deviance: 218.28  on 678  degrees of freedom
AIC: 224.28

Number of Fisher Scoring iterations: 7
```

Odds ratios

This data shows whether people became ill after eating ice cream in the 1940s.

```
head(ice_cream)
```

```
  ill vanilla.ice.cream
1   1                 1
2   1                 1
3   1                 1
4   1                 1
5   1                 1
6   1                 1
```

```
ice_cream %>% count(ill, vanilla.ice.cream)
```

```
  ill vanilla.ice.cream  n
1   0                 0 18
2   0                 1 11
3   1                 0  3
4   1                 1 43
```

Odds ratios

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

```
library(epitools)
response <- ice_cream %>% pull(ill)
predictor <- ice_cream %>% pull(vanilla.ice.cream)
oddsratio(predictor, response)
```

```
$data
         Outcome
Predictor  0  1 Total
    0     18  3    21
    1     11 43    54
    Total 29 46    75

$measure
         odds ratio with 95% C.I.
Predictor estimate    lower    upper
        0  1.00000       NA       NA
        1 21.40719 5.927963 109.4384

$p.value
         two-sided
Predictor      midp.exact    fisher.exact      chi.square
        0              NA              NA              NA
        1 0.0000002698215 0.0000002597451 0.0000001813314

$correction
[1] FALSE

attr(,"method")
```

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

- Use `cor()` to calculate correlation between two vectors. `corrplot()` is nice for a quick visualization!
- `t.test()` tests the difference in means between two vectors
- `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 = binomial()` to do a logistic regression
  - `summary()` gives useful statistics
- `oddsratio()` from the `epitools` package can calculate odds ratios
- this is just the tip of the iceberg!

Lab Part 2

Class Website

Lab

Image by Gerd Altmann from Pixabay

Extra Slides

Adding correlation value to a plot

Note that you can add the correlation value to a plot, via the `annotate()`.

```
cor_val <- cor(x, y, use = "complete.obs")
cor_val_label <- paste0("r = ", round(cor_val, 3))

circ %>%
  ggplot(aes(x = orangeAverage, y = purpleAverage)) +
  geom_point(size = 0.3) +
  annotate("text", x = 2000, y = 7500, label = cor_val_label, size = 5)
```