# Univariate and multivariable regression

::: callout-tip ## Extended Materials

You can find the original, extended version of this chapter here. ::

This page demonstrates the use of **base** R regression functions such as `glm()` and the **gtsummary** package to look at associations between variables (e.g. odds ratios, risk ratios and hazard ratios). It also uses functions like `tidy()` from the **broom** package to clean-up regression outputs.

> **Data Preparation**
>
> **Load packages**
>
> ```
> pacman::p_load(
>   rio,          # File import
>   here,         # File locator
>   tidyverse,    # data management + ggplot2 graphics,
>   stringr,      # manipulate text strings
>   purrr,        # loop over objects in a tidy way
>   gtsummary,    # summary statistics and tests
>   broom,        # tidy up results from regressions
>   lmtest,       # likelihood-ratio tests
>   parameters,   # alternative to tidy up results from regressions
>   see           # alternative to visualise forest plots
>   )
> ```
>
> **Import data**
>
> We import the dataset of cases from a simulated Ebola epidemic. If you want to follow along, click to download the "clean" linelist (as .rds file).
>
> ```
> # import the linelist
> linelist <- import("linelist_cleaned.rds")
> ```
>
> **Clean data**
>
> *   Store explanatory variables
> We store the names of the explanatory columns as a character vector. This will be referenced later.

```
## define variables of interest
explanatory_vars <- c("gender", "fever", "chills", "cough", "aches", "vomit")
```

*   Convert to 1's and 0's

Below we convert the explanatory columns from "yes"/"no", "m"/"f", and "dead"/"alive" to **1 / 0**, to cooperate with the expectations of logistic regression models. To do this efficiently, used **across()** from **dplyr** to transform multiple columns at one time. The function we apply to each column is **case_when()** (also **dplyr**) which applies logic to convert specified values to 1's and 0's.

Note: the "." below represents the column that is being processed by **across()** at that moment.

```
## convert dichotomous variables to 0/1
linelist <- linelist %>%
  mutate(across(
    .cols = all_of(c(explanatory_vars, "outcome")),  ## for each column listed and "outc
    .fns = ~case_when(
      . %in% c("m", "yes", "Death")   ~ 1,            ## recode male, yes and death to 1
      . %in% c("f", "no",  "Recover") ~ 0,            ## female, no and recover to 0
        TRUE                          ~ NA_real_)     ## otherwise set to missing
    )
  )
```

*   Drop rows with missing values

To drop rows with missing values, can use the **tidyr** function **drop_na()**. However, we only want to do this for rows that are missing values in the columns of interest.

The first thing we must to is make sure our **explanatory_vars** vector includes the column **age** (**age** would have produced an error in the previous **case_when()** operation, which was only for dichotomous variables). Then we pipe the **linelist** to **drop_na()** to remove any rows with missing values in the **outcome** column or any of the **explanatory_vars** columns.

Before running the code, the number of rows in the **linelist** is **nrow(linelist)**.

```
## add in age_category to the explanatory vars
explanatory_vars <- c(explanatory_vars, "age_cat")

## drop rows with missing information for variables of interest
linelist <- linelist %>%
  drop_na(any_of(c("outcome", explanatory_vars)))
```

2

> The number of rows remaining in `linelist` is `nrow(linelist)`.

## Univariate

Your use case will determine which R package you use. We present two options for doing univariate analysis:

- Use functions available in **base** R to quickly print results to the console. Use the **broom** package to tidy up the outputs.

- Use the **gtsummary** package to model and get publication-ready outputs

### base R

\*   Linear regression

The **base** R function `lm()` perform linear regression, assessing the relationship between numeric response and explanatory variables that are assumed to have a linear relationship.

Provide the equation as a formula, with the response and explanatory column names separated by a tilde `~`. Also, specify the dataset to `data =`. Define the model results as an R object, to use later.

```
lm_results <- lm(ht_cm ~ age, data = linelist)
```

You can then run `summary()` on the model results to see the coefficients (Estimates), P-value, residuals, and other measures.

```
summary(lm_results)
```

```
Call:
lm(formula = ht_cm ~ age, data = linelist)

Residuals:
     Min       1Q   Median       3Q      Max
-128.579  -15.854    1.177   15.887  175.483

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  69.9051     0.5979   116.9   <2e-16 ***
```

```
age              3.4354    0.0293   117.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.75 on 4165 degrees of freedom
Multiple R-squared:  0.7675,    Adjusted R-squared:  0.7674
F-statistic: 1.375e+04 on 1 and 4165 DF,  p-value: < 2.2e-16
```

Alternatively you can use the `tidy()` function from the **broom** package to pull the results in to a table. What the results tell us is that for each year increase in age the height increases by 3.5 cm and this is statistically significant.

```
tidy(lm_results)
```

```
# A tibble: 2 x 5
  term         estimate std.error statistic p.value
  <chr>           <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)     69.9     0.598      117.       0
2 age              3.44    0.0293     117.       0
```

You can then also use this regression to add it to a **ggplot**, to do this we first pull the points for the observed data and the fitted line in to one data frame using the `augment()` function from **broom**.

```
## pull the regression points and observed data in to one dataset
points <- augment(lm_results)

## plot the data using age as the x-axis
ggplot(points, aes(x = age)) +
  ## add points for height
  geom_point(aes(y = ht_cm)) +
  ## add your regression line
  geom_line(aes(y = .fitted), colour = "red")
```
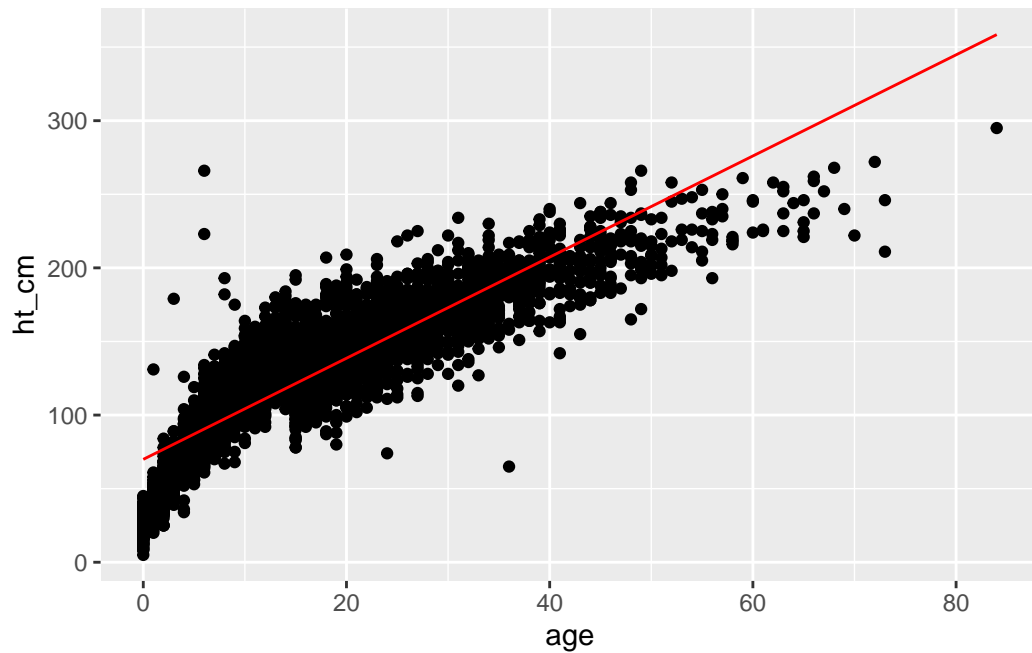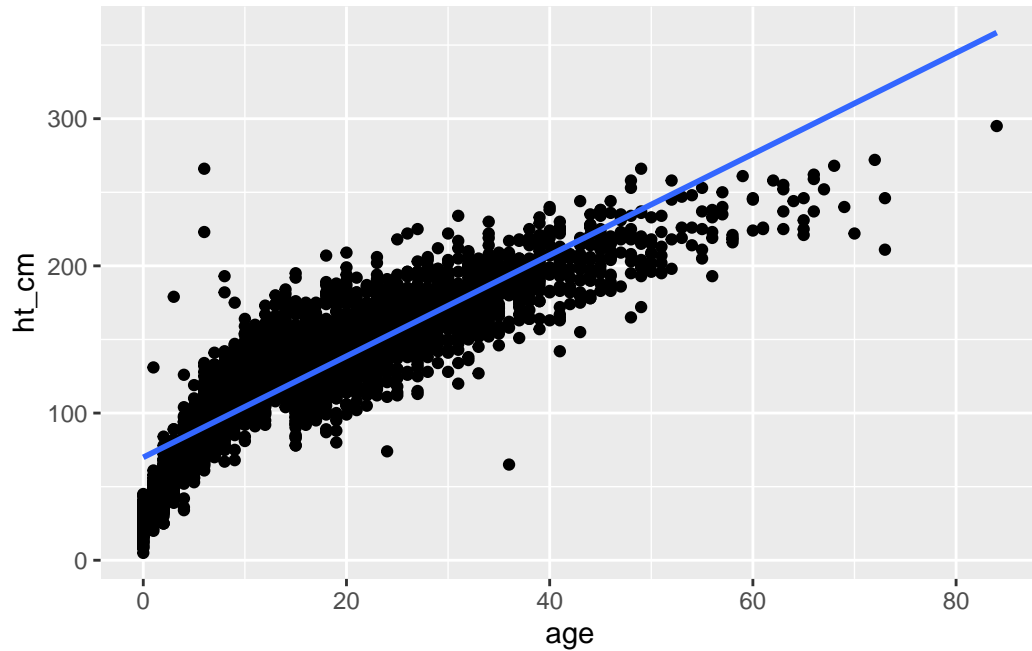
It is also possible to add a simple linear regression straight straight in **ggplot** using the `geom_smooth()` function.

```r
## add your data to a plot
 ggplot(linelist, aes(x = age, y = ht_cm)) +
  ## show points
  geom_point() +
  ## add a linear regression
  geom_smooth(method = "lm", se = FALSE)
```

`geom_smooth()` using formula = 'y ~ x'

See the Resource section at the end of this chapter for more detailed tutorials.

**\*** Logistic regression

The function `glm()` from the **stats** package (part of **base** R) is used to fit Generalized Linear Models (GLM).

`glm()` can be used for univariate and multivariable logistic regression (e.g. to get Odds Ratios). Here are the core parts:

```
# arguments for glm()
glm(formula, family, data, weights, subset, ...)
```

- `formula` = The model is provided to `glm()` as an equation, with the outcome on the left and explanatory variables on the right of a tilde `~`.

- `family` = This determines the type of model to run. For logistic regression, use `family = "binomial"`, for poisson use `family = "poisson"`. Other examples are in the table below.

- `data` = Specify your data frame

If necessary, you can also specify the link function via the syntax `family = familytype(link = "linkfunction")`. You can read more in the documentation about other families and optional arguments such as `weights =` and `subset = (?glm)`.

| Family | Default link function |
|---|---|
| "binomial" | (link = "logit") |
| "gaussian" | (link = "identity") |
| "Gamma" | (link = "inverse") |
| "inverse.gaussian" | (link = "1/mu^2") |
| "poisson" | (link = "log") |
| "quasi" | (link = "identity", variance = "constant") |
| "quasibinomial" | (link = "logit") |
| "quasipoisson" | (link = "log") |

When running `glm()` it is most common to save the results as a named R object. Then you can print the results to your console using `summary()` as shown below, or perform other operations on the results (e.g. exponentiate).

**\*** Univariate `glm()`

In this example we are assessing the association between different age categories and the outcome of death (coded as 1 in the Preparation section). Below is a univariate model of `outcome` by `age_cat`. We save the model output as `model` and then print it with `summary()` to the console. Note the estimates provided are the *log odds* and that the baseline level is the first factor level of `age_cat` ("0-4").

```
model <- glm(outcome ~ age_cat, family = "binomial", data = linelist)
summary(model)
```

```
Call:
glm(formula = outcome ~ age_cat, family = "binomial", data = linelist)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.233738   0.072805   3.210  0.00133 **
age_cat5-9  -0.062898   0.101733  -0.618  0.53640
age_cat10-14 0.138204   0.107186   1.289  0.19726
age_cat15-19 -0.005565  0.113343  -0.049  0.96084
age_cat20-29 0.027511   0.102133   0.269  0.78765
```

```
age_cat30-49   0.063764    0.113771    0.560  0.57517
age_cat50-69  -0.387889    0.259240   -1.496  0.13459
age_cat70+    -0.639203    0.915770   -0.698  0.48518
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5712.4  on 4166  degrees of freedom
Residual deviance: 5705.1  on 4159  degrees of freedom
AIC: 5721.1

Number of Fisher Scoring iterations: 4
```

To alter the baseline level of a given variable, ensure the column is class Factor and move the desired level to the first position with `fct_relevel()`. For example, below we take column `age_cat` and set "20-29" as the baseline before piping the modified data frame into `glm()`.

```
linelist %>%
  mutate(age_cat = fct_relevel(age_cat, "20-29", after = 0)) %>%
  glm(formula = outcome ~ age_cat, family = "binomial") %>%
  summary()
```

```
Call:
glm(formula = outcome ~ age_cat, family = "binomial", data = .)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.26125    0.07163   3.647 0.000265 ***
age_cat0-4   -0.02751    0.10213  -0.269 0.787652
age_cat5-9   -0.09041    0.10090  -0.896 0.370220
age_cat10-14  0.11069    0.10639   1.040 0.298133
age_cat15-19 -0.03308    0.11259  -0.294 0.768934
age_cat30-49  0.03625    0.11302   0.321 0.748390
age_cat50-69 -0.41540    0.25891  -1.604 0.108625
age_cat70+   -0.66671    0.91568  -0.728 0.466546
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 5712.4  on 4166  degrees of freedom
Residual deviance: 5705.1  on 4159  degrees of freedom
AIC: 5721.1

Number of Fisher Scoring iterations: 4
```

**\*** Printing results

For most uses, several modifications must be made to the above outputs. The function `tidy()` from the package **broom** is convenient for making the model results presentable.

Here we demonstrate how to combine model outputs with a table of counts.

1) Get the *exponentiated* log odds ratio estimates and confidence intervals by passing the model to `tidy()` and setting `exponentiate = TRUE` and `conf.int = TRUE`.

```r
model <- glm(outcome ~ age_cat, family = "binomial", data = linelist) %>%
  tidy(exponentiate = TRUE, conf.int = TRUE) %>%        # exponentiate and produce CIs
  mutate(across(where(is.numeric), round, digits = 2))  # round all numeric columns
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `across(where(is.numeric), round, digits = 2)`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))
```

Below is the outputted tibble `model`:

Search:

| term | | | | | | | std.⏿ |
|------|------|------|------|------|------|------|------|
| (Intercept) | 1.26 | 0.07 | 3.21 | 0 | 1.1 | 1.4 | |
| age_cat5-9 | 0.94 | 0.1 | -0.62 | 0.54 | 0.77 | 1.1 | |
| age_cat10-14 | 1.15 | 0.11 | 1.29 | 0.2 | 0.93 | 1.4 | |
| age_cat15-19 | 0.99 | 0.11 | -0.05 | 0.96 | 0.8 | 1.2 | |
| age_cat20-29 | 1.03 | 0.1 | 0.27 | 0.79 | 0.84 | 1.2 | |
| age_cat30-49 | 1.07 | 0.11 | 0.56 | 0.58 | 0.85 | 1.3 | |
| age_cat50-69 | 0.68 | 0.26 | -1.5 | 0.13 | 0.41 | 1.1 | |
| age_cat70+ | 0.53 | 0.92 | -0.7 | 0.49 | 0.07 | 3 | |

2) Combine these model results with a table of counts. Below, we create the a counts cross-table with the `tabyl()` function from **janitor**.

```
counts_table <- linelist %>%
  janitor::tabyl(age_cat, outcome)
```

Here is what this **counts_table** data frame looks like:

Show 8 ⌄ entries                                          Search: [          ]

| age_cat | | |
|---|---|---|
| 0-4 | 338 | 427 |
| 5-9 | 365 | 433 |
| 10-14 | 273 | 396 |
| 15-19 | 238 | 299 |
| 20-29 | 345 | 448 |
| 30-49 | 228 | 307 |
| 50-69 | 35 | 30 |
| 70+ | 3 | 2 |

Showing 1 to 8 of 8 entries                    Previous  1  Next

Now we can bind the `counts_table` and the `model` results together horizontally with `bind_cols()` (**dplyr**). Remember that with `bind_cols()` the rows in the two data frames must be aligned perfectly. In this code, because we are binding within a pipe chain, we use `.` to represent the piped object `counts_table` as we bind it to `model`. To finish the process, we use `select()` to pick the desired columns and their order, and finally apply the **base** R `round()` function across all numeric columns to specify 2 decimal places.

```
combined <- counts_table %>%          # begin with table of counts
  bind_cols(., model) %>%             # combine with the outputs of the regression
  select(term, 2:3, estimate,         # select and re-order cols
         conf.low, conf.high, p.value) %>%
  mutate(across(where(is.numeric), round, digits = 2)) ## round to 2 decimal places
```

Here is what the combined data frame looks like, printed nicely as an image with a function from **flextable**. The [Tables for presentation] explains how to customize such tables with **flextable**, or or you can use numerous other packages such as **knitr** or **GT**.

```
combined <- combined %>%
  flextable::qflextable()
```

**\*** Looping multiple univariate models

Below we present a method using `glm()` and `tidy()` for a more simple approach, see the section on **gtsummary**.

To run the models on several exposure variables to produce univariate odds ratios (i.e. not controlling for each other), you can use the approach below. It uses `str_c()` from **stringr** to create univariate formulas (see [Characters and strings]), runs the `glm()` regression on each formula, passes each `glm()` output to `tidy()` and finally collapses all the model outputs together with `bind_rows()` from **tidyr**. This approach uses `map()` from the package **purrr** to iterate.

1) Create a vector of column names of the explanatory variables. We already have this as `explanatory_vars` from the Preparation section of this page.

2) Use `str_c()` to create multiple string formulas, with `outcome` on the left, and a column name from `explanatory_vars` on the right. The period `.` substitutes for the column name in `explanatory_vars`.

```
explanatory_vars %>% str_c("outcome ~ ", .)
```

```
[1] "outcome ~ gender"  "outcome ~ fever"   "outcome ~ chills"
```

```
[4] "outcome ~ cough"    "outcome ~ aches"    "outcome ~ vomit"
[7] "outcome ~ age_cat"
```

3) Pass these string formulas to `map()` and set `~glm()` as the function to apply to each input. Within `glm()`, set the regression formula as `as.formula(.x)` where `.x` will be replaced by the string formula defined in the step above. `map()` will loop over each of the string formulas, running regressions for each one.

4) The outputs of this first `map()` are passed to a second `map()` command, which applies `tidy()` to the regression outputs.

5) Finally the output of the second `map()` (a list of tidied data frames) is condensed with `bind_rows()`, resulting in one data frame with all the univariate results.

```
models <- explanatory_vars %>%        # begin with variables of interest
  str_c("outcome ~ ", .) %>%          # combine each variable into formula ("outcome ~ vari

  # iterate through each univariate formula
  map(
    .f = ~glm(                        # pass the formulas one-by-one to glm()
      formula = as.formula(.x),       # within glm(), the string formula is .x
      family = "binomial",            # specify type of glm (logistic)
      data = linelist)) %>%           # dataset

  # tidy up each of the glm regression outputs from above
  map(
    .f = ~tidy(
      .x,
      exponentiate = TRUE,            # exponentiate
      conf.int = TRUE)) %>%           # return confidence intervals

  # collapse the list of regression outputs in to one data frame
  bind_rows() %>%

  # round all numeric columns
  mutate(across(where(is.numeric), round, digits = 2))
```

This time, the end object `models` is longer because it now represents the combined results of several univariate regressions. Click through to see all the rows of `model`.

| term | | | estimate | | | std.erro |
|---|---|---|---|---|---|---|
| (Intercept) | 1.28 | 0.04 | 5.67 | 0 | 1.18 | 1.4 |
| gender | 1 | 0.06 | -0.04 | 0.97 | 0.88 | 1.13 |
| (Intercept) | 1.28 | 0.07 | 3.44 | 0 | 1.11 | 1.48 |
| fever | 1 | 0.08 | 0.01 | 0.99 | 0.85 | 1.17 |
| (Intercept) | 1.28 | 0.03 | 6.98 | 0 | 1.19 | 1.37 |

As before, we can create a counts table from the `linelist` for each explanatory variable, bind it to `models`, and make a nice table. We begin with the variables, and iterate through them with `map()`. We iterate through a user-defined function which involves creating a counts table with **dplyr** functions. Then the results are combined and bound with the `models` model results.

```
## for each explanatory variable
univ_tab_base <- explanatory_vars %>%
  map(.f =
    ~{linelist %>%                      ## begin with linelist
        group_by(outcome) %>%           ## group data set by outcome
        count(.data[[.x]]) %>%          ## produce counts for variable of interest
        pivot_wider(                    ## spread to wide format (as in cross-tabulation)
          names_from = outcome,
          values_from = n) %>%
        drop_na(.data[[.x]]) %>%        ## drop rows with missings
        rename("variable" = .x) %>%     ## change variable of interest column to "variabl
        mutate(variable = as.character(variable))} ## convert to character, else non-dicho
      ) %>%

  ## collapse the list of count outputs in to one data frame
  bind_rows() %>%

  ## merge with the outputs of the regression
  bind_cols(., models) %>%

  ## only keep columns interested in
  select(term, 2:3, estimate, conf.low, conf.high, p.value) %>%

  ## round decimal places
  mutate(across(where(is.numeric), round, digits = 2))
```

Below is what the data frame looks like.

Show [5 ▾] entries                                                    Search: [        ]

| term | | | | 0 | | 1 | |
|------|---|---|---|---|---|---|---|
| (Intercept) | 909 | 1168 | 1.28 | 1.18 | 1.4 | 0 |
| gender | 916 | 1174 | 1 | 0.88 | 1.13 | 0.97 |
| (Intercept) | 340 | 436 | 1.28 | 1.11 | 1.48 | 0 |
| fever | 1485 | 1906 | 1 | 0.85 | 1.17 | 0.99 |
| (Intercept) | 1472 | 1877 | 1.28 | 1.19 | 1.37 | 0 |

Showing 1 to 5 of 20 entries                    Previous  [1]  2  3  4  Next

**gtsummary package**

Below we present the use of `tbl_uvregression()` from the **gtsummary** package. **gtsummary** functions do a good job of running statistics *and* producing professional-looking outputs. This function produces a table of univariate regression results.

We select only the necessary columns from the `linelist` (explanatory variables and the outcome variable) and pipe them into `tbl_uvregression()`. We are going to run univariate regression on each of the columns we defined as `explanatory_vars` in the data Preparation section (gender, fever, chills, cough, aches, vomit, and age_cat).

Within the function itself, we provide the `method =` as `glm` (no quotes), the `y =` outcome column (`outcome`), specify to `method.args =` that we want to run logistic regression via `family = binomial`, and we tell it to exponentiate the results.

The output is HTML and contains the counts

```
univ_tab <- linelist %>%
  dplyr::select(explanatory_vars, outcome) %>% ## select variables of interest

  tbl_uvregression(                            ## produce univariate table
    method = glm,                              ## define regression want to run (generalised
    y = outcome,                               ## define outcome variable
    method.args = list(family = binomial),     ## define what type of glm want to run (logist
    exponentiate = TRUE                        ## exponentiate to produce odds ratios (rather
  )

## view univariate results table
univ_tab
```

| Characteristic | N | OR | 95% CI | p-value |
|---|---|---|---|---|
| gender | 4,167 | 1.00 | 0.88, 1.13 | >0.9 |
| fever | 4,167 | 1.00 | 0.85, 1.17 | >0.9 |
| chills | 4,167 | 1.03 | 0.89, 1.21 | 0.7 |
| cough | 4,167 | 1.15 | 0.97, 1.37 | 0.11 |
| aches | 4,167 | 0.93 | 0.76, 1.14 | 0.5 |
| vomit | 4,167 | 1.09 | 0.96, 1.23 | 0.2 |
| age_cat | 4,167 | | | |
| 0-4 | | — | — | |
| 5-9 | | 0.94 | 0.77, 1.15 | 0.5 |
| 10-14 | | 1.15 | 0.93, 1.42 | 0.2 |
| 15-19 | | 0.99 | 0.80, 1.24 | >0.9 |
| 20-29 | | 1.03 | 0.84, 1.26 | 0.8 |

| Characteristic | N | OR | 95% CI | p-value |
|---|---|---|---|---|
| 30-49 | | 1.07 | 0.85, 1.33 | 0.6 |
| 50-69 | | 0.68 | 0.41, 1.13 | 0.13 |
| 70+ | | 0.53 | 0.07, 3.20 | 0.5 |

There are many modifications you can make to this table output, such as adjusting the text labels, bolding rows by their p-value, etc. See tutorials here and elsewhere online.

## Multivariable

For multivariable analysis, we again present two approaches:

- `glm()` and `tidy()`

- **gtsummary** package

The workflow is similar for each and only the last step of pulling together a final table is different.

### Conduct multivariable

Here we use `glm()` but add more variables to the right side of the equation, separated by plus symbols (`+`).

To run the model with all of our explanatory variables we would run:

```
mv_reg <- glm(outcome ~ gender + fever + chills + cough + aches + vomit + age_cat, family

summary(mv_reg)
```

```
Call:
glm(formula = outcome ~ gender + fever + chills + cough + aches +
    vomit + age_cat, family = "binomial", data = linelist)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.069054   0.131726   0.524    0.600
gender      0.002448   0.065133   0.038    0.970
fever       0.004309   0.080522   0.054    0.957
```

19

```
chills         0.034112   0.078924    0.432     0.666
cough          0.138584   0.089909    1.541     0.123
aches         -0.070705   0.104078   -0.679     0.497
vomit          0.086098   0.062618    1.375     0.169
age_cat5-9    -0.063562   0.101851   -0.624     0.533
age_cat10-14   0.136372   0.107275    1.271     0.204
age_cat15-19  -0.011074   0.113640   -0.097     0.922
age_cat20-29   0.026552   0.102780    0.258     0.796
age_cat30-49   0.059569   0.116402    0.512     0.609
age_cat50-69  -0.388964   0.262384   -1.482     0.138
age_cat70+    -0.647443   0.917375   -0.706     0.480


(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5712.4   on 4166   degrees of freedom
Residual deviance: 5700.2   on 4153   degrees of freedom
AIC: 5728.2


Number of Fisher Scoring iterations: 4
```

If you want to include two variables and an interaction between them you can separate them with an asterisk * instead of a +. Separate them with a colon : if you are only specifying the interaction. For example:

```
glm(outcome ~ gender + age_cat * fever, family = "binomial", data = linelist)
```

*Optionally*, you can use this code to leverage the pre-defined vector of column names and re-create the above command using `str_c()`. This might be useful if your explanatory variable names are changing, or you don't want to type them all out again.

```
## run a regression with all variables of interest
mv_reg <- explanatory_vars %>%   ## begin with vector of explanatory column names
  str_c(collapse = "+") %>%      ## combine all names of the variables of interest separate
  str_c("outcome ~ ", .) %>%     ## combine the names of variables of interest with outcome
  glm(family = "binomial",       ## define type of glm as logistic,
      data = linelist)           ## define your dataset
```

* Building the model

You can build your model step-by-step, saving various models that include certain explanatory variables. You can compare these models with likelihood-ratio tests using `lrtest()` from the package **lmtest**, as below:

*NOTE:* Using **base** `anova(model1, model2, test = "Chisq)` produces the same results

```r
model1 <- glm(outcome ~ age_cat, family = "binomial", data = linelist)
model2 <- glm(outcome ~ age_cat + gender, family = "binomial", data = linelist)

lmtest::lrtest(model1, model2)
```

```
Likelihood ratio test

Model 1: outcome ~ age_cat
Model 2: outcome ~ age_cat + gender
  #Df  LogLik Df Chisq Pr(>Chisq)
1   8 -2852.6
2   9 -2852.6  1 2e-04     0.9883
```

Another option is to take the model object and apply the `step()` function from the **stats** package. Specify which variable selection direction you want use when building the model.

```r
## choose a model using forward selection based on AIC
## you can also do "backward" or "both" by adjusting the direction
final_mv_reg <- mv_reg %>%
  step(direction = "forward", trace = FALSE)
```

You can also turn off scientific notation in your R session, for clarity:

```r
options(scipen=999)
```

As described in the section on univariate analysis, pass the model output to `tidy()` to exponentiate the log odds and CIs. Finally we round all numeric columns to two decimal places. Scroll through to see all the rows.

```r
mv_tab_base <- final_mv_reg %>%
  broom::tidy(exponentiate = TRUE, conf.int = TRUE) %>%  ## get a tidy dataframe of estima
  mutate(across(where(is.numeric), round, digits = 2))        ## round
```

Here is what the resulting data frame looks like:

Show 10 ⌄ entries                                                    Search: [          ]

| term | | | estimate | | | std.e |
|---|---|---|---|---|---|---|
| (Intercept) | 1.07 | 0.13 | 0.52 | 0.6 | 0.83 | 1.3 |
| gender | 1 | 0.07 | 0.04 | 0.97 | 0.88 | 1.1 |
| fever | 1 | 0.08 | 0.05 | 0.96 | 0.86 | 1.1 |
| chills | 1.03 | 0.08 | 0.43 | 0.67 | 0.89 | 1.2 |
| cough | 1.15 | 0.09 | 1.54 | 0.12 | 0.96 | 1.3 |
| aches | 0.93 | 0.1 | -0.68 | 0.5 | 0.76 | 1.1 |
| vomit | 1.09 | 0.06 | 1.37 | 0.17 | 0.96 | 1.2 |
| age_cat5-9 | 0.94 | 0.1 | -0.62 | 0.53 | 0.77 | 1.1 |
| age_cat10-14 | 1.15 | 0.11 | 1.27 | 0.2 | 0.93 | 1.4 |
| age_cat15-19 | 0.99 | 0.11 | -0.1 | 0.92 | 0.79 | 1.2 |

Showing 1 to 10 of 14 entries                   Previous | 1 | 2 | Next

**Combine univariate and multivariable**

**\*** Combine with **gtsummary**

The **gtsummary** package provides the `tbl_regression()` function, which will take the outputs from a regression (`glm()` in this case) and produce an nice summary table.

```
## show results table of final regression
mv_tab <- tbl_regression(final_mv_reg, exponentiate = TRUE)
```

Let's see the table:

```
mv_tab
```

Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.

| Characteristic | OR | 95% CI | p-value |
|---|---|---|---|
| gender | 1.00 | 0.88, 1.14 | >0.9 |
| fever | 1.00 | 0.86, 1.18 | >0.9 |
| chills | 1.03 | 0.89, 1.21 | 0.7 |
| cough | 1.15 | 0.96, 1.37 | 0.12 |
| aches | 0.93 | 0.76, 1.14 | 0.5 |
| vomit | 1.09 | 0.96, 1.23 | 0.2 |
| age_cat | | | |
| 0-4 | — | — | |
| 5-9 | 0.94 | 0.77, 1.15 | 0.5 |
| 10-14 | 1.15 | 0.93, 1.41 | 0.2 |
| 15-19 | 0.99 | 0.79, 1.24 | >0.9 |
| 20-29 | 1.03 | 0.84, 1.26 | 0.8 |
| 30-49 | 1.06 | 0.85, 1.33 | 0.6 |
| 50-69 | 0.68 | 0.40, 1.13 | 0.14 |
| 70+ | 0.52 | 0.07, 3.19 | 0.5 |

You can also combine several different output tables produced by **gtsummary** with the `tbl_merge()` function. We now combine the multivariable results with the **gtsummary** *univariate* results that we created above:

```
## combine with univariate results
tbl_merge(
  tbls = list(univ_tab, mv_tab),                    # combine
  tab_spanner = c("**Univariate**", "**Multivariable**")) # set header names
```

Table printed with `knitr::kable()`, not {gt}. Learn why at
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html
To suppress this message, include `message = FALSE` in code chunk header.

| Characteristic | N | OR | 95% CI | p-value | OR | 95% CI | p-value |
|---|---|---|---|---|---|---|---|
| gender | 4,167 | 1.00 | 0.88, 1.13 | >0.9 | 1.00 | 0.88, 1.14 | >0.9 |
| fever | 4,167 | 1.00 | 0.85, 1.17 | >0.9 | 1.00 | 0.86, 1.18 | >0.9 |
| chills | 4,167 | 1.03 | 0.89, 1.21 | 0.7 | 1.03 | 0.89, 1.21 | 0.7 |
| cough | 4,167 | 1.15 | 0.97, 1.37 | 0.11 | 1.15 | 0.96, 1.37 | 0.12 |
| aches | 4,167 | 0.93 | 0.76, 1.14 | 0.5 | 0.93 | 0.76, 1.14 | 0.5 |
| vomit | 4,167 | 1.09 | 0.96, 1.23 | 0.2 | 1.09 | 0.96, 1.23 | 0.2 |
| age_cat | 4,167 | | | | | | |
| 0-4 | | — | — | | — | — | |
| 5-9 | | 0.94 | 0.77, 1.15 | 0.5 | 0.94 | 0.77, 1.15 | 0.5 |
| 10-14 | | 1.15 | 0.93, 1.42 | 0.2 | 1.15 | 0.93, 1.41 | 0.2 |
| 15-19 | | 0.99 | 0.80, 1.24 | >0.9 | 0.99 | 0.79, 1.24 | >0.9 |
| 20-29 | | 1.03 | 0.84, 1.26 | 0.8 | 1.03 | 0.84, 1.26 | 0.8 |
| 30-49 | | 1.07 | 0.85, 1.33 | 0.6 | 1.06 | 0.85, 1.33 | 0.6 |
| 50-69 | | 0.68 | 0.41, 1.13 | 0.13 | 0.68 | 0.40, 1.13 | 0.14 |
| 70+ | | 0.53 | 0.07, 3.20 | 0.5 | 0.52 | 0.07, 3.19 | 0.5 |

**\*** Combine with **dplyr**

An alternative way of combining the `glm()`/`tidy()` univariate and multivariable outputs is with the **dplyr** join functions.

- Join the univariate results from earlier (`univ_tab_base`, which contains counts) with the tidied multivariable results `mv_tab_base`

- Use `select()` to keep only the columns we want, specify their order, and re-name them

- Use `round()` with two decimal places on all the column that are class Double

```
## combine univariate and multivariable tables
left_join(univ_tab_base, mv_tab_base, by = "term") %>%
  ## choose columns and rename them
  select( # new name =  old name
    "characteristic" = term,
    "recovered"      = "0",
    "dead"           = "1",
    "univ_or"        = estimate.x,
    "univ_ci_low"    = conf.low.x,
    "univ_ci_high"   = conf.high.x,
    "univ_pval"      = p.value.x,
    "mv_or"          = estimate.y,
    "mvv_ci_low"     = conf.low.y,
    "mv_ci_high"     = conf.high.y,
    "mv_pval"        = p.value.y
  ) %>%
  mutate(across(where(is.double), round, 2))
```

```
# A tibble: 20 x 11
   characteristic recovered  dead univ_or univ_ci_low univ_ci_high univ_pval
   <chr>              <dbl> <dbl>   <dbl>       <dbl>        <dbl>     <dbl>
 1 (Intercept)          909  1168    1.28        1.18          1.4         0
 2 gender               916  1174    1           0.88          1.13        0.97
 3 (Intercept)          340   436    1.28        1.11          1.48        0
 4 fever               1485  1906    1           0.85          1.17        0.99
 5 (Intercept)         1472  1877    1.28        1.19          1.37        0
 6 chills               353   465    1.03        0.89          1.21        0.68
 7 (Intercept)          272   309    1.14        0.97          1.34        0.13
 8 cough               1553  2033    1.15        0.97          1.37        0.11
 9 (Intercept)         1636  2114    1.29        1.21          1.38        0
10 aches                189   228    0.93        0.76          1.14        0.51
11 (Intercept)          931  1144    1.23        1.13          1.34        0
12 vomit                894  1198    1.09        0.96          1.23        0.17
13 (Intercept)          338   427    1.26        1.1           1.46        0
14 age_cat5-9           365   433    0.94        0.77          1.15        0.54
15 age_cat10-14         273   396    1.15        0.93          1.42        0.2
16 age_cat15-19         238   299    0.99        0.8           1.24        0.96
17 age_cat20-29         345   448    1.03        0.84          1.26        0.79
18 age_cat30-49         228   307    1.07        0.85          1.33        0.58
19 age_cat50-69          35    30    0.68        0.41          1.13        0.13
20 age_cat70+             3     2    0.53        0.07          3.2         0.49
# i 4 more variables: mv_or <dbl>, mvv_ci_low <dbl>, mv_ci_high <dbl>,
```

```
#   mv_pval <dbl>
```