Data Analysis 2

Ryan Womack

2025-09-15

Data Analysis 2

statistical tests, distributions, regression, bootstrap, and more + comparison with Python

1 Overview

This workshop reviews the implementation of basic statistical tests and methods in R, with discussion contextualizing the appropriate use of these tests. For comparison purposes, an alternative Python approach to performing similar statistical analysis is illustrated in some cases. This workshop takes inspiration from the **Introduction to Modern Statistics** available via github and online text, part of the OpenIntro project.

Note that "packages" in R are the equivalent of "modules" in Python.

2 Setup and preparing data

2.1 R packages required

We will use the *pak* package for installation as a more complete approach to package management. Replace the pkg commands with *install.packages()* versions if you prefer.

This session relies on the *tidyverse* suite of packages for data manipulation as a preference, although the same tasks could be accomplished in base R. The statistical functions used are those from base R. Install *pak* and *tidyverse* if you don't already have them on your system. We will also need *reticulate* to run Python code chunks.

```
install.packages("pak", dependencies=TRUE)
library(pak)
pkg_install("tidyverse")
pkg_install("reticulate")
pkg_install("infer")
pkg_install("TOSTER")
devtools::session_info()
```

Now let's load the tidyverse, infer, TOSTER, and reticulate (for Python support only)

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 -
-
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.2
v ggplot2   4.0.0     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() -
-
x purrr::%||%()   masks base::%||%()
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become err
```

```
library(infer)
library(TOSTER)
Sys.setenv(RETICULATE_PYTHON = "/usr/bin/python3")
library(reticulate)
```

2.2 Data import and preparation

Now let's grab some data. We will use a realistic data example, the World Bank's Gender Statistics database, whose raw data is directly downloadable. Other World Bank Open Data is available as well. See genderdata.worldbank.org for more background on the Gender Data portal. Note that we have to inspect the data and understand the variables first before manipulating in R – this is not an automatic process.

```
getOption("timeout")
```

```
[1] 60
```

```
options(timeout=6000)
download.file("https://databank.worldbank.org/data/download/Gender_Stats_CSV.zip",
 ↪  "gender.zip")
unzip("gender.zip")
gender_data <- read_csv("Gender_StatsCSV.csv")
```

```
Rows: 338405 Columns: 69
-- Column specification --------------------------------------------------------
---
Delimiter: ","
chr  (4): Country Name, Country Code, Indicator Name, Indicator Code
dbl (65): 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, ...
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now we'll perform a few steps to clean the data, focusing on generating a useable file for a few countries (Central Asia and Mongolia plus selected high population or high income countries), from the latest available data year with complete data, typically the year before the last in the data set. For this session, we'll just run these steps without explaining them. Data cleaning and wrangling is covered in more detail in the Data Analysis 1 workshop (forthcoming). The final filtered output is the *gender_data_final* file, which we *attach* so that a copy of the data is made the default dataset for this session.

```
# clean the data to remove superfluous columns
names(gender_data)
gender_data <- gender_data[,c(-2,-4)]
names(gender_data)

# select countries of interest
country_list <- c("China", "Germany", "India", "Japan", "Kazakhstan", "Kyrgyz
 ↪  Republic", "Mongolia", "Russian Federation", "Tajikistan",
 ↪  "Turkmenistan", "United States", "Uzbekistan")
gender_data2 <-
  gender_data %>%
  filter(`Country Name` %in% country_list)

# clean the data to focus on a recent more complete time period
gender_data3 <-
   gender_data2 %>%
   pivot_longer(3:67, names_to = "Year", values_to = "Value")

#filter by year
gender_data2023 <-
  gender_data3 %>%
  filter(Year=="2023")

gender_data2023 <- gender_data2023[,-3]

gender_data2023wide <-
  gender_data2023 %>%
  pivot_wider(names_from = "Indicator Name", values_from = "Value")

# now use a little sapply trick to select variables that don't have much
 ↪  missing data - here the proportion is 0.75 (the 0.25 in the function is
 ↪  1-proportion desired)

gender_data_filtered <- gender_data2023wide[,!sapply(gender_data2023wide,
 ↪  function(x) mean(is.na(x)))>0.25]

# and lastly simplify the dataset by removing some of the topics we won't use
```

```
phrases <- c("Worried", "Made", "Received", "Saved", "Used", "Coming",
 ↪  "Borrowed")

gender_data_final <-
  gender_data_filtered %>%
  select(!starts_with(phrases))

# we'll also generate a couple of variables for future use

gender_data_final$female_high_labor <- gender_data_final$`Labor force
 ↪  participation rate, female (% of female population ages 15-64) (modeled
 ↪  ILO estimate)`>70
gender_data_final$male_high_labor <- gender_data_final$`Labor force
 ↪  participation rate, male (% of male population ages 15-64) (modeled ILO
 ↪  estimate)`>70

attach(gender_data_final)
```

Let's test for some basic relationships between labor participation, gender, fertility, and income, using the World Bank's own definition of gender:

> "Gender refers to the social, behavioral, and cultural attributes, expectations, and norms associated with being male or female."

Note that this is just for the purposes of demonstration, and not a serious investigation into these important research issues.

3 Statistical Tests

3.1 t-test

In statistics there are many **hypothesis tests**. Per the Wikipedia entry on statistical hypothesis tests,

> "a statistical hypothesis test is a method of statistical inference used to decide whether the data sufficiently supports a particular hypothesis. A statistical hypothesis test typically involves a calculation of a test statistic. Then a decision is made, either by comparing the test statistic to a critical value or equivalently by evaluating a p-value computed from the test statistic. Roughly 100 specialized statistical tests have been defined."

The hypothesis is usually framed as a "Null hypothesis", describing the situation where there is no statistically significant difference, and the alternative hypothesis. Then a test is chosen appropriate to the situation, which often involves invoking a statistical distribution.

The *t-test* is one of the most common tests, used to determine if the difference between two groups, according to some numerical measure, is statistically significant. The Null hypothesis, usually denoted H_0, is that there is no difference (a difference of zero). The Alternative, usually denoted H_1 or H_A, is that the difference is not zero.

The t-test was originally, and sometimes still is called "Student's t-test". The story of the student who invented this test, and his affiliation with a certain well known Irish stout, is interesting from both a statistical and human perspective. The *t*-test has a few variants: a one sample test, a two sample test with unpaired results, and two sample test with paired results. The one sample test simply checks whether one measure is significantly different from the null. The two sample test with unpaired results compares whether two separate sets of observations are different, such as sampling the populations of two different cities. The two sample test with paired results implies that we have the same subjects in the dataset who are measured twice, perhaps at different intervals or via different measures. The *t*-distribution is the underlying statistical distribution determining the test statistic and critical value that we check for. These are conveniently summarized by the *p-value*, which expresses the level of significance. We typically look for a $p<.05$ to determine statistical significance, but this is a convention: other *p*-value cutoffs can be used.

3.2 One Sample t-test

We start with a one sample *t*-test to check the labor force participation rates of females and males across the countries in our dataset.

```
t.test(`Labor force participation rate, female (% of female population ages
 ↪  15-64) (modeled ILO estimate)`)
```

```

    One Sample t-test

data:  Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)
t = 12.855, df = 11, p-value = 5.714e-08
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 49.41461 69.83056
sample estimates:
mean of x
 59.62258
```

This first example returns a statistically significant, but not very interesting result. The default is to test whether the variable is different than zero. It is not surprising that female labor force participation is higher than zero.

```
t.test(`Labor force participation rate, male (% of male population ages
 ↪  15-64) (modeled ILO estimate)`, mu=70)
```

```

    One Sample t-test

data:  Labor force participation rate, male (% of male population ages 15-
64) (modeled ILO estimate)
t = 1.569, df = 11, p-value = 0.1449
```

```
alternative hypothesis: true mean is not equal to 70
95 percent confidence interval:
 67.77376 83.28024
sample estimates:
mean of x
   75.527
```

We refine the test by passing the option *mu*=70, to test whether the male labor participation rate is significantly different than 70. With a p>.05 and a 95% confidence interval of (67,83), it is *not* significantly different than 70 in a statistical sense.

Note the use of options is a common R syntax technique. Base execution of the command without options gives sensible results, but one can pass many options to tweak the function's behavior. Most of the things demanded by statisticians are available through these option tweaks. Typing *?t.test* (the question mark followed by the function name) will pull up the help that displays the possibilities.

```
?t.test
```

3.3 Paired t-test

Perhaps a more interesting question is to see if there is a significantly significant difference in male and female labor participation rates. We can compare them, country-by-country, using the paired=TRUE option, in the paired two-sample test below. For one explanation of this see STHDA.

```
t.test(`Labor force participation rate, female (% of female population ages
 ↪  15-64) (modeled ILO estimate)`,`Labor force participation rate, male (%
 ↪  of male population ages 15-64) (modeled ILO estimate)`, paired = TRUE,
 ↪  alternative = "two.sided")
```

```

    Paired t-test

data:  Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate) and Labor force participation rate, male (% of male population ages 15-
64) (modeled ILO estimate)
t = -3.9943, df = 11, p-value = 0.002106
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -24.66817  -7.14066
sample estimates:
mean difference
      -15.90442
```

There is a statistically significant difference, with the 95% confidence interval of female participation being between 7 and 25 percentage points lower than male rates, and a mean difference of -16 points.

3.4 Python t-test

We will be giving a flavor of the Python approach to these problems, without going in depth. Python is covered in other workshops in greater detail.

Running Python in RStudio requires the *reticulate* package. Your Python installation should also have *numpy* and *scipy* installed. Python will not have direct access to the R data structures, so we use a simplified example here from Builtin.com

How to pass data from R to Python and vice-versa is described in this blog post by Dima Diachkov

Note that our code chunk is labeled Python. For more *reticulate* package, see this post

Also note that it is also possible to invoke R from a Python environment using rpy2

```
import numpy as np
import pandas as pd
from scipy import stats

# import from R
gender_python = r.gender_data_final

# print(gender_python)

labor = gender_python.loc[:,"Labor force participation rate, female (% of
 ↪  female population ages 15-64) (modeled ILO estimate)"]

# Hypothesized population mean
mu = 70

# Perform one-sample t-test
# t_stat, p_value = stats.ttest_1samp(labor, mu)
t_stat, p_value = stats.ttest_1samp(labor, mu)
print("T statistic:", t_stat)


T statistic: -2.2375199095291447

print("P-value:", p_value)


P-value: 0.046904605502055996

# Setting significance level
alpha = 0.05

# Interpret the results
if p_value < alpha:
    print("Reject the null hypothesis; there is a significant difference
     ↪  between the sample mean and the hypothesized population mean.")
else:
```

```
    print("Fail to reject the null hypothesis; there is no significant
     ↪  difference between the sample mean and the hypothesized population
     ↪  mean.")
```

```
Reject the null hypothesis; there is a significant difference between the sample mean and the hypot
```

3.5 Chi-squared test (in R and Python)

Many operations in R are a question of finding the appropriate function, either using the built-in R help via the ? (for help on a specific command) or ?? (to search across functions by keywords), or by using the built-in help system. Using a search engine like DuckDuckGo for a topic plus "in R" or a help site like stackoverflow.com will also turn up useful leads. Or one can consult books such as the many R-related book series in SpringerLink for more comprehensive how-to's. I would not recommend using AI tools for anything important, since one must understand the results well enough to check them for errors.

So, for example, if we wanted to run a chi-squared test instead of a t-test, some noodling about will turn up the *chisq.test* function. We won't discuss the distribution or mathematical background of the chi-squared test here.

Take a look at this table of the number of countries with "high" labor participation rates of over 70%, for males and females.

```
table(female_high_labor,male_high_labor)
```

```
                male_high_labor
female_high_labor FALSE TRUE
            FALSE     2    6
            TRUE      0    4
```

We can test whether this is a statistically significant pattern via the chi-squared test, implemented with:

```
chisq.test(table(female_high_labor,male_high_labor))
```

```
Warning in chisq.test(table(female_high_labor, male_high_labor)): Chi-squared
approximation may be incorrect
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  table(female_high_labor, male_high_labor)
X-squared = 0.075, df = 1, p-value = 0.7842
```

Which is *not* statistically significant, probably due to the small size of the matrix.

3.6 How to choose the appropriate statistical test?

This is a question that depends on many factors, that we won't go into detail about in this introductory workshop, but you can consult the following resources:

- UCLA's quick guide
- a nice brief summary article
- You can also consult Sage Research Methods' Which Stats Test? for more information on choosing the appropriate statistical test. The other guides in Sage Research Methods are good too for more in-depth exploration of specific tests and topics! - *Rutgers-restricted*

3.7 Distributions

One great aspect of R is the availability of tools to work with almost any statistical distribution. The density, distribution function, quantile function and random generation of a number are easily available with R functions, via the prefixes *d, p, q, and r.* For example for the normal distribution, we have:

```
# the density of a standard normal distribution at the value 2 (2 above mean
 ↪  of zero)

dnorm(2)
```

```
[1] 0.05399097
```

```
# the cumulative percentage of a standard normal distribution below the value
 ↪  2

pnorm(2)
```

```
[1] 0.9772499
```

```
# mean and standard deviation of the distribution can also be specified
# in this example the cumulative percentage of a normal distribution with
 ↪  mean 100 and s.d. 20, below the value 90

pnorm(90, mean=100, sd=20)
```

```
[1] 0.3085375
```

```
# provides the numeric value of a particular quantile of the distribution
 ↪  (again standard normal in this example)

qnorm(.9)
```

```
[1] 1.281552
```

```
# a single random draw from the standard normal distribution

rnorm(1)
```

```
[1] -0.06961225
```

```
# five random draws from a normal distribution with mean 100 and s.d. 20

rnorm(5, mean=100, sd=20)
```

```
[1] 103.34722 109.18629 117.95550  90.83920  99.86816
```

```
# getting help

?rnorm
```

As usual the question mark will provide details about implementation, variables, and options.

Other distributions and their functions can be found on the TaskViews on Distributions page.

4 Correlation

We can also perform simple correlations to evaluate the strength of a relationship between two variables, but there are some cautions.

```
cor(`GDP per capita (constant 2015 US$)`,`Fertility rate, total (births per
 ↪  woman)`, na.rm=TRUE)
```

Why won't this work? The defaults and options of R commands are not very standardized. Instead try this option:

```
cor(`GDP per capita (constant 2015 US$)`,`Fertility rate, total (births per
 ↪  woman)`, use="complete.obs")
```

```
[1] -0.5703267
```

The *cor.test* function gives more complete output

```
cor.test(`GDP per capita (constant 2015 US$)`,`Fertility rate, total (births
 ↪  per woman)`)
```

```

    Pearson's product-moment correlation

data:  GDP per capita (constant 2015 US$) and Fertility rate, total (births per woman)
t = -2.1956, df = 10, p-value = 0.05283
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.862064717  0.005314307
sample estimates:
       cor
-0.5703267
```

5 Regression

Regression is a common and fundamental technique to model a relationship between a *response* variable (also called dependent or outcome variable) and one or more *explanatory* variables (also called independent or predictor variables). Proper implementation of regression requires careful attention to the data and examination of the model fit, variable selection, and more. The quick and dirty approach below is simply designed to show R syntax in action.

5.1 Linear Regression

We start at the beginning, with linear regresion and the *lm* command. Let's see if there is a relationship between female labor participation and GDP:

```
lm(`Labor force participation rate, female (% of female population ages
 ↪  15-64) (modeled ILO estimate)`~`GDP per capita (constant 2015 US$)`)
```

```
Call:
lm(formula = `Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)` ~
    `GDP per capita (constant 2015 US$)`)

Coefficients:
                         (Intercept)  `GDP per capita (constant 2015 US$)`
                           5.202e+01                                4.519e-04
```

Which predicts about a 0.5 increase in the percentage labor participation of females for every $1000 rise in GDP. However, the presentation of the result is a bit odd. R only presents the coefficients by default. The *summary* command must be used to tease out the additional information that we expect in a regression table to evaluate fit and significance.

```
summary(lm(`Labor force participation rate, female (% of female population
 ↪  ages 15-64) (modeled ILO estimate)`~`GDP per capita (constant 2015 US$)`))
```

```
Call:
lm(formula = `Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)` ~
    `GDP per capita (constant 2015 US$)`)

Residuals:
    Min      1Q  Median      3Q     Max
-19.565 -10.689   3.864   8.156  19.525

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                        5.202e+01  5.198e+00  10.007 1.58e-06 ***
`GDP per capita (constant 2015 US$)` 4.519e-04  2.001e-04   2.258   0.0475 *
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.71 on 10 degrees of freedom
Multiple R-squared:  0.3378,    Adjusted R-squared:  0.2715
F-statistic:   5.1 on 1 and 10 DF,  p-value: 0.0475
```

Which is significant at the 0.05 level. Note that we must decide on the appropriate p-value to test for in advance and resist the temptation to say that this is "almost significant". Note that this is heavily influenced by the fact that we are using a small sample of 12 countries, which could be an argument to use a 0.10 signficance level.

Let's check some additional relationships:

```
summary(lm(`Labor force participation rate, female (% of female population
 ↪  ages 15-64) (modeled ILO estimate)`~`Fertility rate, total (births per
 ↪  woman)`))
```

```

Call:
lm(formula = `Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)` ~
    `Fertility rate, total (births per woman)`)

Residuals:
    Min      1Q  Median      3Q     Max
-28.406  -2.038   2.802   4.065  25.988

Coefficients:
                                        Estimate Std. Error t value Pr(>|t|)
(Intercept)                               83.269     11.438   7.280 2.66e-05
`Fertility rate, total (births per woman)`  -10.804      4.898  -2.206   0.0519

(Intercept)                              ***
`Fertility rate, total (births per woman)` .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.82 on 10 degrees of freedom
Multiple R-squared:  0.3273,    Adjusted R-squared:   0.26
F-statistic: 4.865 on 1 and 10 DF,  p-value: 0.05193
```

```
summary(lm(`Fertility rate, total (births per woman)`~`GDP per capita
 ↪  (constant 2015 US$)`))
```

```

Call:
lm(formula = `Fertility rate, total (births per woman)` ~ `GDP per capita (constant 2015 US$)`)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-1.2918 -0.5261  0.1835  0.5335  1.0035

Coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                          2.584e+00  2.778e-01   9.301 3.08e-06 ***
`GDP per capita (constant 2015 US$)` -2.348e-05  1.070e-05  -2.196   0.0528 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.733 on 10 degrees of freedom
Multiple R-squared:  0.3253,    Adjusted R-squared:  0.2578
F-statistic: 4.821 on 1 and 10 DF,  p-value: 0.05283
```

Here our p-values are significant at the 10% level but not the 5% level. Also note that if want to run the regression without an intercept, we can just add a -1 to the equation.

```
summary(lm(`Labor force participation rate, female (% of female population
 ↪  ages 15-64) (modeled ILO estimate)`~`GDP per capita (constant 2015
 ↪  US$)`-1))
```

```
Call:
lm(formula = `Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)` ~
    `GDP per capita (constant 2015 US$)` - 1)

Residuals:
   Min     1Q Median     3Q    Max
-45.89  24.87  37.89  50.74  56.68

Coefficients:
                                     Estimate Std. Error t value Pr(>|t|)
`GDP per capita (constant 2015 US$)` 0.0017497  0.0004822   3.628  0.00397 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 43.39 on 11 degrees of freedom
Multiple R-squared:  0.5448,    Adjusted R-squared:  0.5034
F-statistic: 13.16 on 1 and 11 DF,  p-value: 0.003969
```

5.2 Multiple Regression

And if we need to explore *multiple regression*, with multiple explanatory variables, this is as easy as using a + sign in the equation:

```
summary(lm(`Labor force participation rate, female (% of female population
 ↪  ages 15-64) (modeled ILO estimate)`~`Fertility rate, total (births per
 ↪  woman)`+`GDP per capita (constant 2015 US$)`))
```

```
Call:
lm(formula = `Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)` ~
   `Fertility rate, total (births per woman)` + `GDP per capita (constant 2015 US$)`)

Residuals:
    Min      1Q  Median      3Q     Max
-23.259  -4.659   3.096   5.191  24.219

Coefficients:
                                             Estimate Std. Error t value
(Intercept)                                 69.4191018 15.8793413   4.372
`Fertility rate, total (births per woman)` -6.7351397  5.8180954  -1.158
`GDP per capita (constant 2015 US$)`        0.0002937  0.0002396   1.226
                                            Pr(>|t|)
(Intercept)                                  0.00179 **
`Fertility rate, total (births per woman)`   0.27682
`GDP per capita (constant 2015 US$)`         0.25127
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.49 on 9 degrees of freedom
Multiple R-squared:  0.4236,    Adjusted R-squared:  0.2955
F-statistic: 3.307 on 2 and 9 DF,  p-value: 0.08381
```

5.3 Stored Regression Objects

A very important feature of R is the flexibility that derives from the ability to store (and use) our regression output as an R object. The output becomes an R object, and we can access its variables/components using the familiar $ notation. This can allow us to easily store and compare multiple models and use regression data such as residuals or predicted values as inputs into other equations and analyses.

```
regoutput<-lm(`Labor force participation rate, female (% of female population
 ↪  ages 15-64) (modeled ILO estimate)`~`Fertility rate, total (births per
 ↪  woman)`+`GDP per capita (constant 2015 US$)`)
names(regoutput)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

```
regoutput$residuals
```

```
        1         2         3         4         5         6         7
 3.582201  2.887776 -23.259257  3.303619 24.218814  5.172425  5.244711
        8         9        10        11        12
 8.434951 -16.056077 -1.281381 -9.053288 -3.194494
```

Once we have store the regression output, numerous quick functions are available that build off of the regression results, such as *predict* for predicted values and *anova* for the Analysis of Variance.

```
# predicted values
predict(regoutput)
```

```
       1        2        3        4        5        6        7        8
66.35780 72.98422 56.78426 72.19138 52.49719 51.61558 52.54329 63.02505
       9       10       11       12
49.14208 53.61638 77.77329 46.94049
```

```
# analysis of variance (anova)
anova(regoutput)
```

```
Analysis of Variance Table

Response: Labor force participation rate, female (% of female population ages 15-
64) (modeled ILO estimate)
                                          Df  Sum Sq Mean Sq F value  Pr(>F)
`Fertility rate, total (births per woman)`  1  929.32  929.32  5.1104 0.05013 .
`GDP per capita (constant 2015 US$)`        1  273.39  273.39  1.5034 0.25127
Residuals                                   9 1636.65  181.85
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5.4 Regression Diagnostics

Regression diagnostics are used to evaluate the fit and appropriateness of the regression model as applied. Once again, we are providing some illustrative examples without going into depth here.

One thing that we may wish to test is the normality of the regression residuals (i.e., the difference between predicted and actual values of the observations is normally distributed). See this discussion of Normality at U Wisconsin and this article available in PubMedCentral for a brief explanation.

We can apply a typically used test for normality, the Shapiro-Wilk Normality Test, as follows. We get standardized residuals from the model with *rstandard* (or more simply the residuals accessed with *regoutput$residuals*), and then use the *shapiro.test* function to perform the test, or the (less powerful) Kolmogorov-Smirnov Test with *ks.test*, which also allows us to test against different distributions (here we just specific *'pnorm'* for the normal distribution).

A p<.05 in both the Shapiro-Wilk and Kolmogorov-Smirnov test implies *rejection* of normality. But note in the example below that we must be careful to standardize the residuals so that the test is applied correctly.

```
shapiro.test(rstandard(regoutput))
```

```
    Shapiro-Wilk normality test

data:  rstandard(regoutput)
W = 0.93721, p-value = 0.4628
```

```
ks.test(regoutput$residuals, 'pnorm')
```

```
    Exact one-sample Kolmogorov-Smirnov test

data:  regoutput$residuals
D = 0.58139, p-value = 0.0002217
alternative hypothesis: two-sided
```

```
ks.test(rstandard(regoutput), 'pnorm', mean=0, sd=1)
```

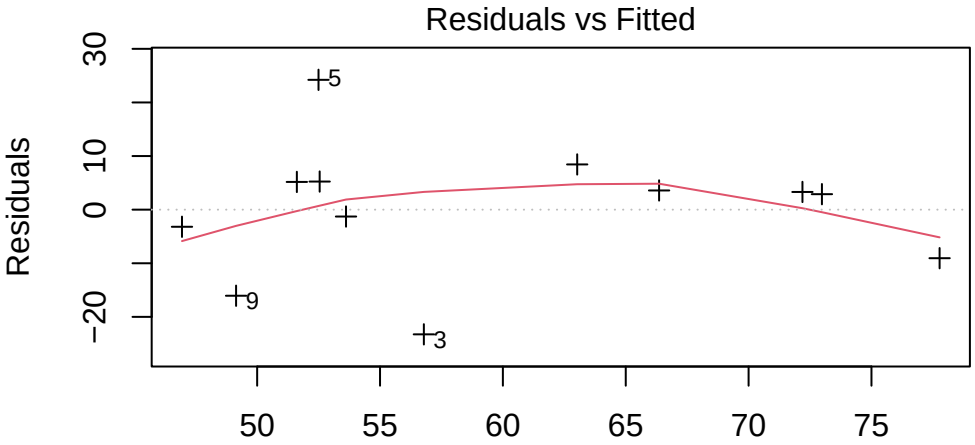
```
    Exact one-sample Kolmogorov-Smirnov test

data:  rstandard(regoutput)
D = 0.18069, p-value = 0.7658
alternative hypothesis: two-sided
```

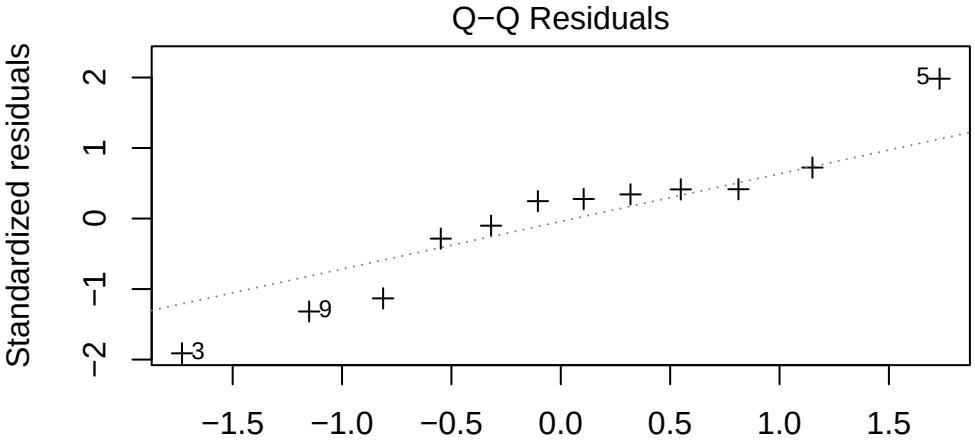
Note that standardizing the residuals makes a big difference for the Kolmogorov-Smirnov test!

We can also easily pull up regression diagnostic plots:

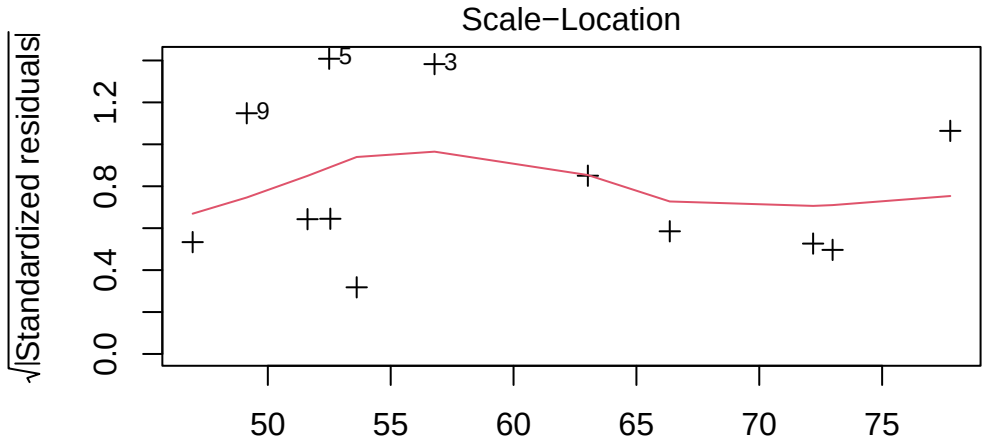
```
plot(regoutput, pch=3)
```



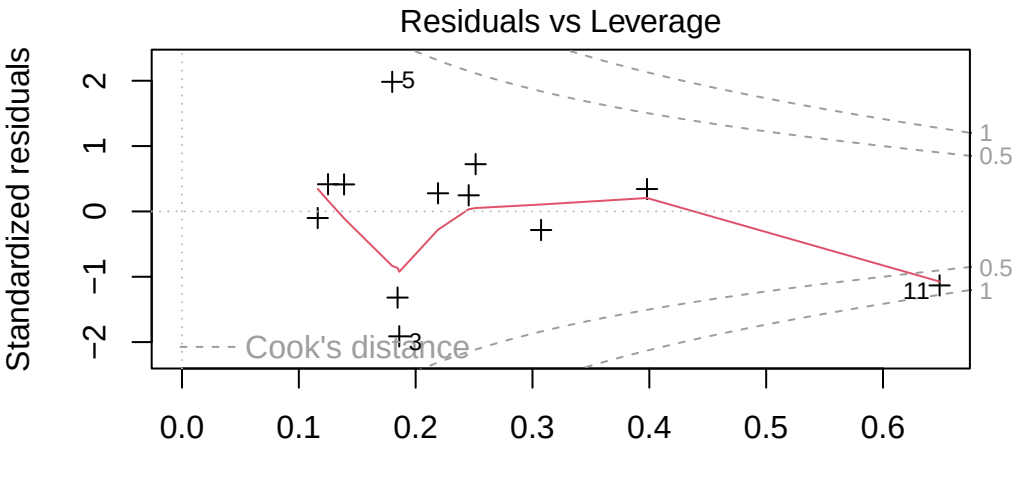
Fitted values
lm(‘Labor force participation rate, female (% of female population ages 15



Theoretical Quantiles
lm(‘Labor force participation rate, female (% of female population ages 15



Fitted values
lm(‘Labor force participation rate, female (% of female population ages 15



5.5 glm and Logistic Regression

R is developed by statisticians for statisticians, so you will find every variant of regression that you might eventually need. The *glm* command implements generalized linear models (or alternatively, the *glm2* package).

One commonly used alternative, necessary where the response variable is binary (categorical) is *logistic regression*. The *glm* command is used, with "family=binomial" as an argument. The binomial distribution represents the fact that the response variable is binary (either a two-level factor or directly coded as 0/1 - referred to as "one-hot" coding in certain circles).

We can try this out with our categorized variable that labels female labor participation as high if over 70% (otherwise low):

```
logistic_output <- glm(female_high_labor ~ `Fertility rate, total (births per
 ↪  woman)`+`GDP per capita (constant 2015 US$)`, family=binomial)

summary(logistic_output)
```

```
Call:
glm(formula = female_high_labor ~ `Fertility rate, total (births per woman)` +
    `GDP per capita (constant 2015 US$)`, family = binomial)

Coefficients:
                                        Estimate Std. Error z value
(Intercept)                            8.427e-01  2.532e+00   0.333
`Fertility rate, total (births per woman)` -8.776e-01  1.011e+00  -0.868
`GDP per capita (constant 2015 US$)`    1.489e-05  3.676e-05   0.405
                                       Pr(>|z|)
(Intercept)                               0.739
`Fertility rate, total (births per woman)`    0.385
```

```
`GDP per capita (constant 2015 US$)`          0.685

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 15.276  on 11  degrees of freedom
Residual deviance: 13.306  on  9  degrees of freedom
AIC: 19.306

Number of Fisher Scoring iterations: 4
```

We won't delve further into the interpretation of logistic regression or other regression alternatives here, but just know that nearly anything is possible with R!

5.6 Python regression

Linear Regression is part of the scikit-learn module which forms the core of machine learning in Python.

Some basic information on linear regression in Python is at RealPython

We include this code as a quick illustration of the approach. As is typical, the output in Python must be individually extracted with commands. The stargazer module is at least one approach to automating the process of generating output tables. The statsmodels module is less commonly used, but provides an alternative in Python that generates output tables more easily.

```
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.linear_model import LinearRegression

# import from R
gender_python = r.gender_data_final

# specify variables
labor = gender_python.loc[:,"Labor force participation rate, female (% of
 ↪  female population ages 15-64) (modeled ILO estimate)"]
gdp = gender_python.loc[:,"GDP per capita (constant 2015 US$)"]

# this step is important to put the data in Python-friendly form
labor2 = labor.array.reshape(-1, 1)
gdp2 = gdp.array.reshape(-1, 1)
model = LinearRegression().fit(labor2,gdp2)

print(model.intercept_, model.coef_)
```

```
[-27728.21536913] [[747.3999527]]
```

```
r_sq = model.score(labor2, gdp2)
print(f"R-squared: {r_sq}")
```

```
R-squared: 0.33775699752387656
```

6 Bootstrap

This section is modeled off the approach used by Introduction to Modern Statistics, Second Edition by Mine Çetinkaya-Rundel and Johanna Hardin, which goes into greater depth and provides many useful code exercises. We'll refer to this text as *IMS* below. As elsewhere, we are providing a quick and dirty demo, but please use these resources to go further!

In particular take a look at the section on bootstrapping and the supplementary tutorials.

Why the "bootstrap"? Many traditional statistical techniques rely on assumptions about the behavior or distribution of the data, assumptions that we cannot always be sure of satsifying. According to IMS, "some statistics do not have simple theory for how they vary, and bootstrapping provides a computational approach for providing interval estimates for almost any population parameter."

The bootstrap works by sampling and resampling the original collection of data. The variability of the samples can be used to estimate the variability of the underlying population, without requiring any further assumptions about its characteristics.

Essentially, we resample our data to construct an empirical distribution, which we can then use to estimate standard errors and corresponding confidence intervals, as well as other parameters. We call it the "bootstrap" because we are "pulling ourselves up by our bootstraps" and creating the standard error ourselves via this replicate sampling process.

R makes this easy via the *infer* package from the tidyverse.

The bootstrap in R is constructed by a three-part sequence of commands: *specify*, *generate*, and *calculate*: - we *specify* the variable we want to sample - then *generate* the number of sample replicates we want to generate - then *calculate* the statistic of interest on the replicates

Before any sampling we can simply generate the point estimate of our proportion, in this case one of our indicator variables for the gender data, whether a woman can work in dangerous jobs in the same way as a man.

Note that |> is the "pipe" in base R (relatively newly introduced, so you won't see it as often yet), versus the %>% "pipe" in tidyverse via *magrittr* (traditional).

```
point_estimate <- gender_data_final %>%
  specify(response = `A woman can work in a job deemed dangerous in the same
   ↪  way as a man (1=yes; 0=no)`) %>%
  calculate(stat = "mean")
```

```
Warning: Removed 1 rows containing missing values.
```

```
point_estimate
```

```
Response: A woman can work in a job deemed dangerous in the same way as a man...
# A tibble: 1 x 1
   stat
  <dbl>
1 0.545
```

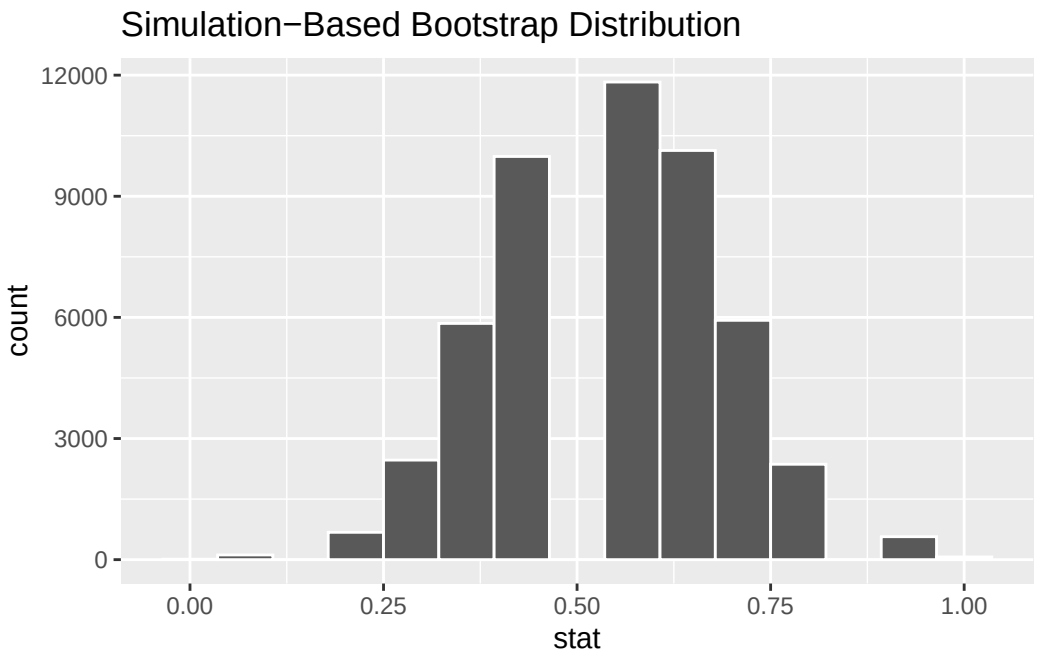
Then run replicate samples that we will use to estimate the confidence interval.

```
boot_dist_dangerous <- gender_data_final %>%
  specify(response = `A woman can work in a job deemed dangerous in the same
   ↪  way as a man (1=yes; 0=no)`) %>%
  generate(reps = 50000, type = "bootstrap") %>%
  calculate(stat = "mean")
```

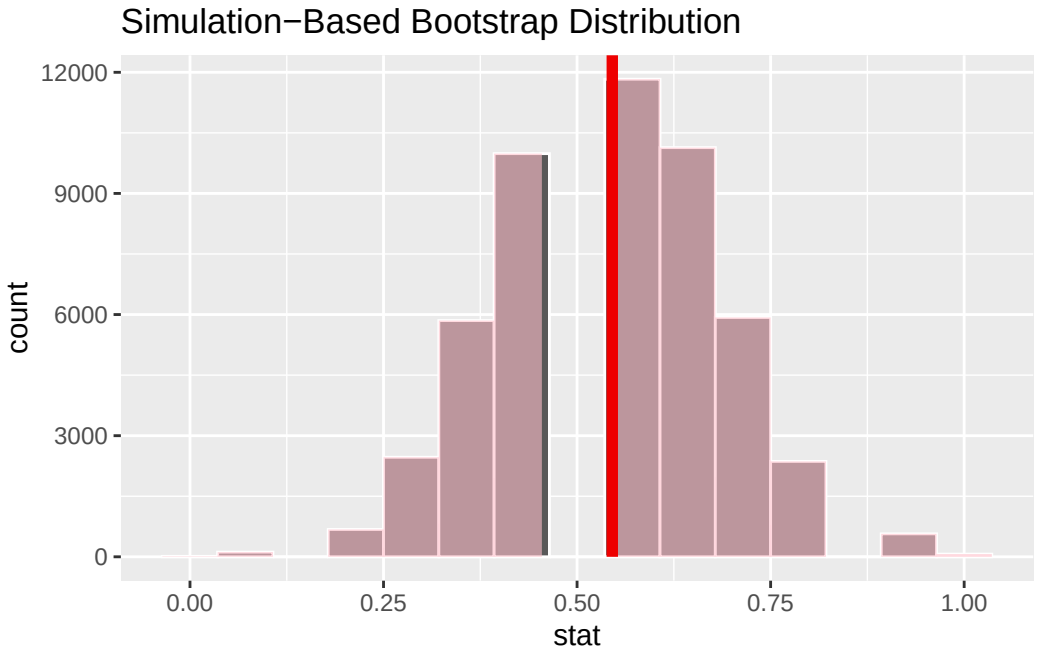
```
Warning: Removed 1 rows containing missing values.
```

We can easily visualize the distribution of the replicate samples.

```
boot_dist_dangerous %>%
  visualize()
```




```
boot_dist_dangerous %>%
  visualize() +
  shade_p_value(obs_stat = point_estimate, direction = "two-sided")
```




```
boot_dist_dangerous %>%
  get_confidence_interval(
    point_estimate = point_estimate,
    level = 0.95,
    type = "se"
  )
```

```
# A tibble: 1 x 2
  lower_ci upper_ci
     <dbl>    <dbl>
1    0.252    0.839
```

Note that we are resampling from a small population of 12, so even with large replications, our confidence interval is wide. Using this technique on the type of country data we have is atypical, and is just for demonstration purposes. A more typical usage would be on survey data with individual responses selected from a larger population. Again to quote *IMS*, "bootstrapping is best suited for modeling studies where the data have been generated through random sampling from a population."

6.1 t-test via simulation

IMS has some examples of using *infer* to simulate t-test type results, however this is not as well-suited to the case of a paired t-test, for which the TOSTER package has a built in function that quickly implements a bootstrapped version of the t-test. This is just one example of how we can switch among the many ways of accomplishing a task in R.

First, instead of preserving the pairs, we get at the same idea by computing the difference in labor participation for males and females for each country.

```
labor_diff <- `Labor force participation rate, male (% of male population
 ↪  ages 15-64) (modeled ILO estimate)` - `Labor force participation rate,
 ↪  female (% of female population ages 15-64) (modeled ILO estimate)`

labor_diff
```

```
 [1]  9.865  7.854 47.096 11.856  6.267 25.371 14.414  8.813 19.792 -3.810
[11]  9.482 33.853
```

Then apply the bootstrap t-test from the *TOSTER* package, specifying 1000 replications.

```
boot_t_test(labor_diff, R=1000)
```

```
    Bootstrapped One Sample t-test

data:  x
p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  8.819708 30.719636
sample estimates:
mean of x
 15.90442
```

6.2 bootstrap to estimate variability of the slope of the regression

We can also bootstrap our way to some of the results of a regression model. Here we also use the *hypothesize* option of the *infer* package. Note that for this small sample size, the bootstrap estimate fluctuates quite a bit if you repeat the experiment.

```
var_slope <- gender_data_final |>
  specify(`Labor force participation rate, female (% of female population
   ↪  ages 15-64) (modeled ILO estimate)`~`Fertility rate, total (births per
   ↪  woman)`) |>
  hypothesize(null = "independence") |>
  generate(reps = 500, type = "permute") |>
  calculate(stat = "slope")

var_slope |>
  # Ungroup the dataset
  ungroup() |>
  # Calculate summary statistics
  summarize(
    # Mean of stat
    mean_stat = mean(stat),
    # Std error of stat
```

```
    std_err_stat = sd(stat)
  )

# A tibble: 1 x 2
  mean_stat std_err_stat
      <dbl>        <dbl>
1     0.411         5.96
```

6.3 Python bookstrap

Finally we'll briefy look at the Python approach to bootstrapping.

```
from scipy.stats import bootstrap
import numpy as np

labor = r.labor_diff

#convert array to sequence
data = (labor,)

#calculate 95% bootstrapped confidence interval for median
bootstrap_ci = bootstrap(data, np.mean, confidence_level=0.95,
                         random_state=1, method='percentile')

#view 95% boostrapped confidence interval
print(bootstrap_ci.confidence_interval)
```

```
ConfidenceInterval(low=np.float64(8.982916666666675), high=np.float64(23.89338749999999))
```

There's more that can be done of course, but we'll stop here.

Enjoy R! (and Python!)