Biostatistics and R Coding Resource Sheet: An Attempt to Summarize

Table of Contents

What Is R

In the simplest terms, R is a calculator, often chosen by statisticians and data visualizers due to its vast package support and standards.

More complicatedly, R is a functional programming language with aspects of an object-oriented one written in C and Fortran, though less rigid. Computationally, R is slow and utilizes only a single computer core and RAM, which may not impact most tasks unless dealing with large datasets or complex algorithms. However, R compensates with its rich ecosystem of packages, offering extensive functionality for statistical analysis, data manipulation, visualization, and more, making it a popular choice for both beginners and experienced data scientists alike. Additionally, its open-source (Publicly available) nature fosters a vibrant community where users can collaborate, contribute, and continually improve the language and its capabilities. Python is a common comparison; it offers better computational optimization and better flexibility for tasks beyond statistics however it lacks package quality standards, many of the visualization tools, and statistics specific functions. The other comparison is SAS which falls on the opposite side of the spectrum; it is typically seen as less daunting and simpler but lacks many of the tool's R and python have easily available. SAS does have extensive package support, testing before release, and isn't open source making it the choice of government for stability, at a cost.

Steps of Analysis

1. What is your question?
2. Gather and Process Data
3. Exploratory Data Analysis (EDA)
4. Choose Analysis or Model Technique(s)
5. Conduct Analysis
6. Interpret Results
7. Communicate Findings of Analysis

Basics of R

Functions – Similar to math, these take input and produce an output.
All functions have inputs; these can Be populated or require input.
If you are ever confused about what a Function does or its inputs (*params*), Simply type ?*function* in the console And the documentation will appear.

Assign output to a variable Like this:
Out <-***Function***(*Input*) ; <- is assign
Additionally, get data like this:
Var <- Data*$Var*

Writing Reports

Introduction (1)
Data (2) – Who, What, Where, When, How many
Exploratory Data Analysis (3) – Chart Time!
Methods (4) – Try to explain your brain process
Results (5 & 6) – Get output, explain output
Conclusion (7) – explain output (Non-Statistically)

Important Maybe Not So Basics

Conditional (Booleans) – "If" statements;
For computer science TRUE/T = yes or 1; FALSE/F = no or 0.
Conditionals are: If T then X, else (if not X) then Y

Besides the typical less then or greater then you can use ! to indicate not. !***is.na***() would return False for NA values, when it normally would give True. Additionally you can use == to specify something to match. "Apple" == "Apple" ; TRUE

Similarly, when attempting to bunch conditionals use | to say or, and use & to say and.
Ifelse(*Data, X > Y & X < Z*) is saying X larger then Y but smaller then Z.

Important Maybe Not So Basics

Data Storage and Types in R:
At the base of computational statistics and programming is data and data structures. These not only have different uses but using the wrong types can cause issues in functions.

Vectors – single dimension that can hold a single type of data; highly efficient.

Matrices – Two-dimension storage of only numeric data; allows for matrix math and is more efficient.

Data Frames – The most common. two-dimension storage that allows columns to have different types of data; Inefficient and slow but allows for superior data manipulation of all types.

Lists – The holder, allows for storage of other data structures; slow and require careful syntax

Arrays – Matrices with additional dimension; efficient, supports math, but is memory intensive.

Tibbles – essentially a data frame with extra features; can cause issues in functions.

Biostatistics and R Coding Resource Sheet: An Attempt to Summarize

On This Page:

Types of Data – Review

Numeric

Continuous: Measure
79.2, Heart rate

Discrete: Count
1,2,3…, I have one cat

Factors (Categorical)

Ordinal: Groups with Order
1-5 scale, income group

Nominal: A group without order
Nationality or Ethnicity

Binary or Indicator:
0 or 1; Yes or No

In R these often are considered only **These** unless you specify

Packages

What is a package?

Packages, or libraries (for our case they are the same), are collections of functions or tools which can be used to perform a task. In R many of the packages revolve around data and statistics making these packages essential to most of the work we do in R.

Often they are significantly faster than we could write ourselves, not only because they already exist, but because they a written in a faster programming language and then use R as an interface. Simply, if a function exists in a package, use that one.

Packages Used

Tidyverse – Collection of common package essential for most people
Tidymodels – Collection of packages for model testing and training
Psych – Used for summary statistics functions
Ggstatsplot – Cool plots, not essential
Ggpubr – Used for themes, creating tables/charts for publication
Ggtexttable – Create tables from existing dataframes
Gtsummary – Used in creation of Table 1 (lifesaver)
Flextable – Same as above
RColorBrewer – Used to choose non-standard colors of plots
Ggfortify – Used for a couple of things, truthfully not sure how to explain
Jtools – Statistical test output
ResourceSelection – Statistical test output
Lmtest – Statistical test output
Car – Used for a couple things normally, for us statistical test output
Survival – Core package of survival analysis
Multcomp – Statistical test output
Survminer – Builds on survival, can be finicky
MASS – Statistical test output, model selection
VGAM – Ordinal regression
Ggcorrplot – Fun way to visualize correlation plots
AER – Testing Dispersion

Installing and Loading Packages

Install.packages(“Lib”) ; library(Lib)

To use a function from a specific package do: package::Function()

Data Loading

Readr : Common Package used for various types of data
Readxl: Package to read excel files
Haven: Package developed to read SAS file outputs

Code Example

Data <- read_csv(“File_Path”)
“csv” can be replaced with specific file of interest. Xlsx has some weird extra parameters so just don’t use them. Excel will make you sad.
Use “/” for paths; for current directory use “./” and “../” to go back one

Data Processing (Forcats, Dplyr)

The following can be combined with tidyverse pipes; you don’t need to specify data within the function permitted the variable exists in the input.

Data pipe examples: *NewData <- Data %>% Function()*

Processing Functions:

As.XXX such as ***as.numeric***(), ***as.factor***() – Used to change how R sees variables. For example we may see 1 and 0, but R sees continuous

Mutate: Used to create new variables or change existing ones
*Data %>% **mutate**(NewVar = Function(OldVar))*

Ifelse: Base conditional; if x, then A, else, B
Ifelse(Data, Conditional(Var), Then, else) – Useful for creating indicators

Case_when() or Fct_recode() – Both are used to change variables but is cleaner than doing multiple **ifelse**() functions

Case_when():
Data %>% case_when(conditional ~ "Category A", conditional ~ "Category B", TRUE ~ "Other") ; TRUE = Not captured by conditional

Fct_recode() :
Mutate(NewVar = fct_recode(Var, `Level1` = "1", `Level2` = "2"))

Subset(), Select(), Filter() - All Similar function to get more specific data
subset(*Data*, ***conditional1*** & ***conditional2***) - Get rows match conditional
Select(*Data$Column1, Data$Column2*) – select specific columns
Filter() – Nearly exact to subset but more efficient with larger data
Relevel() – Set the group you want as Null, or want compared against.
Relevel(*Factor_Var, Ref*) ; Ref is the group you want compared too

Biostatistics and R Coding Resource Sheet: An Attempt to Summarize

On This Page:

Global R Options

Just some basics that can be incredibly useful.

Options(scipen = 999)

This will stop R from reporting large digits in scientific notation

Options(digits = 3)

This will set default rounding in R to 3 or what you please. This does not always work

Set.seed(91)

This normally is not important. However if you randomize or bootstrap this will make it reproducible by others.

Summary Statistics

Key Functions

Str(Data) – provides high level overview of data, its variables, and types

round(Data, round = n) – when numeric is inputted, it will round to n

describe(Data$Var) – provides summary statistics (mean, med, sd, iqr,...)

describeBy(Data$Var, group = Data$Group_Var) – Similar but by group

summary(Model) ; *Summ(Model)* – both of these exist to give output from models. Summary is built into R although summ gives more information

tidy(Model) – creates a "tidy" data frame as output for models

Table

Table() and *prop.table()* can be used to give information about two categorical variables. Placing the output of table() into prop table will give the proportions of the groups. *Table(Row_Var, Col_Var)*

T-test

t.test(Var, Group, alternative = "two.sided")

besides running a t-test, providing only a continuous variable will return the confidence interval ; for models use confint(Model)

Correlations

This normally will require you to filter NA's using *na.omit(Data)*

Cor(Data) ; to specify two rows do cor(Var1, Var2)

Alternatively, if you don't want to remove NA's

Cor(Data, use = "pairwise.complete.obs") – This will impute NA's

Creation of Visuals - Tables

Table 1

*Table <- tbl_summary(data = Data, include = c(Var1, Var2,....), by = Group_Var, statistic = list(all_continuous() ~ "{mean} ({sd})", all_categorical() ~ "{n} ({p}%)"), label = list(Var1~ "Full Name"), missing = "no", digits = all_continuous() ~ 2) %>% add_n() %>% add_overall() %>% modify_header(label = "Table Title") %>% modify_spanning_header(c("stat_1", "stat_2") ~ "**Group_Var**") %>% add_stat_label() ; for each level in group add additional "stat_n"*

This is one of few functions which gives clear errors and help.

Ggtexttable – Allows to convert dataframe to table output easy

Table <- ggtexttable(Data, rows = c("Row1",...) , cols = c("col1",...) , theme = ttheme("theme")) %>% tab_add_title(text = "title")

Tip: use the tidy function to create a data frame you can input here

Creation of Visuals - Charts

What and What to Use (Briefly)

Single continuous – Histogram + density
Continuous and Categorical – Boxplot + violin
Two continuous – point (scatter) + Smooth (line)
Discrete and categorical – Column

Color

When using R you have the ability to set colors by using names such as "blue" or using hex code (#0000FF). In ggplot the default color palette is set up to be colorblind friendly and encourage thinking about that.

ggplot

Core function

Ggplot(Data, aes(x = Var1, y = Var2, fill = "Color", alpha = opacity)) + geom_XXXX() + ggtitle("Title") + xlab("X Axis") + ylab("Y Axis")

Geom are the code for types of plots; *geom_hist()* = histogram

The *aes()* block can be specified in the initial function or if you want to plot different variables, or even datasets you can put it inside of the geom.

Additional things

To add things to a ggplot simple just put a + and then function.

Facet_wrap(.~Group, scales = "free") – produce multiple plots by group. If you specific scale = free, the x and y can use differ mins and maxs

Coord_flip() - should you have too many categories this will flip x and y. This will produce a tall plot instead of a wide plot; useful for documents

Geom_point() – This is your scatterplot geom; other geoms are normal

Theme() – this is customization function. It allows you to alter every single aspect on a ggplot from subtle movements of elements, removing the legend, or changing the font. While not complicated, it is dense.

Other oddity: when trying to change the color, one would think it is the color option, that changes the outline. The fill option changes main color.

ggcorrplot

Often times ggplot will have packages built on top of it, meaning external functions will use ggplot to produce output. This can be seen with *autoplot()*, and the survminer package. Additionally, using the correlations, we can use the ggcorrplot package and the *ggcorrplot()* function to produce a heatmap. This makes it easy to visualize correlations

Ggcorrplot(Cor_Data)

Biostatistics and R Coding Resource Sheet: An Attempt to Summarize

On This Page:

General Format

The following pages are arranged to attempt to give a background of what these things are, when you'd want to use them, and why. Then the code will be given, followed by assumptions, and lastly some limitations.

General things in regression:

P Fishing: Adding more variables usually will make a model more significant; that doesn't make it more useful. Additionally, having 30,000 participants is amazing; it also makes most things significant.
Look beyond P.

Overfitting: due to us rarely having every possible data point, when designing models to be used outside of an analysis it is important to not include too many variables. This may result in the model being super predictive of your dataset, but not others.

Linear Regression

What, When Why

Used for understanding relationships between continuous variables.

How

lm_model <- lm(Y ~ X1 + X2, data = YourData) ; can also use glm()

Assumptions
- L: Linearity
- I: Independence
- N: Normality
- E: Equal Variance

Can be checked using plots the **autoplot**() function from **ggfortify**

autoplot(lm_model)

Produces: Residuals vs Fitted, Q-Q plot, Scale Vs Location, Cooks

Other ways to check assumptions

shapiro.test(residuals(lm _model)) – Normal residuals (Normality)

bptest(lm _model)- Homoscedasticity (Equal Variance)

car::vif(lm _model) – No Multicollinearity (independence)

Limitations
- Limited to Additive Relationships: Linear regression struggles with capturing interactions or non-linear relationships.
- Limited Extrapolation: Linear regression is not suitable for making predictions outside the range of observed data.
- Assumption of Linearity: Assumes linear relationships, may miss complex patterns which often exist in life.

Logistic Regression

What, When Why

Estimates the probability of binary outcomes, often used for classification. It's valuable in predicting the likelihood of events or binary responses.

How

logit_model<- glm(Y ~ Predictor, data = YourData, family = "binomial")

Assumptions
- Independence
- Linearity of Log Odds
- No Multicollinearity (see Linear Regression; vif())

fitted_values <- predict(logit_model, type = "link")

ggplot(data, aes(x = independent_var, y = fitted_values)) + geom_point() + geom_smooth(method = "lm", se = FALSE)

GOF <- hoslem.test(Outcome_Var, fitted(logit_model))

Limitations
- Need Binary Outcome
- Inflexible compared to advanced statistical methods.

Poisson Regression

What, When Why

Models count data, assuming events occur independently at a constant rate over time/Space. Suitable for analyzing rare events or incidence rates.

How

poisson_model <- glm(Y ~ Predictor, data = YourData, family = "poisson") ; If over dispersed: *family = quasipoisson(link = "log")*

Variable Denominator

poisson_model <- glm(Outcome ~ Predictor, data = YourData, offset = log(Offset_Variable), family = poisson(link = "log"))

Assumptions
- Count Data: Assumes the outcome variable is counts of events.
- Independence: Assumes observations are independent.
- Linear Log Rates: predictors and log rates have linear relationship.
- Homogeneity of Rates: Assumes variance equals the mean

dispersiontest(poisson_model)

Limitations
- Assumes Equal Mean and Variance: assumes variance equals mean, often unrealistic.

Biostatistics and R Coding Resource Sheet: An Attempt to Summarize

On This Page:

Survival & Cox Regression

What, When Why

Analyzes time-to-event data, considering censoring and identifying factors affecting survival probabilities. Cox Proportional Hazards Model is particularly useful for estimating hazard ratios.

Censoring

Censoring in survival analysis is when some subjects' outcomes aren't observed within the study period, often due to loss to follow-up.

How

Cox_model <- coxph(Surv(Time, Event) ~ Predictor, data = YourData)

Assumptions

- Proportional Hazards: Hazard ratios are constant over time.
- Independence of Censoring: Censoring is non-informative.

Cox_Sum <- Cox.zph(Cox_model)

Ggcoxzph(Cox_Sum)

Limitations

- Proportional Hazards: Violations can lead to biased estimates.
- Independence of Censoring: Assumes that censoring is unrelated to the outcome.

Ordinal Regression

What, When Why

Models the relationship between predictors and ordered categorical outcomes. It's beneficial for analyzing data with ordered response levels.

How

Ordinal_model <- polr(Ordinal ~ Predictor, data = YourData)

Assumptions

- Proportional Odds: Odds of higher outcome categories versus lower ones are constant across predictor levels.
- Independence of Observations: Observations are independent.

Test_poe(Ordinal_model) – Test for Proportional Odds

Limitations

- Proportional Odds: Violations indicate differing relationships between predictors and categories.
- Personally dislike this method because it seems odds for interpretation; multinomial regression can be accomplished by using family = multinomial() in glm, but doesn't factor order

Log-Binomial Regression

What, When Why

Estimates relative risks directly for binary outcomes, providing straightforward interpretations of predictor effects.

How

LB_model <- glm(Dependent ~ Predictor, data = YourData, family = binomial(link = "log"))

Assumptions

- Binomial Distribution: Outcome variable follows a binomial distribution.
- Log-linearity: Predictors have a log-linear relationship with the log-odds.

Limitations

- Convergence Issues: May not converge, especially with small sample sizes or high outcome prevalence.

Model Selection

What, When, Why

Use the computer to select model covariates to optimize the model. I believe this is most useful for predictive models although in data where little is known about covariates it can provide useful insight.

How:

full_model <- lm(y ~ ., data = data)

stepwise_model <- stepAIC(full_model, direction = "both")

Notes:

Direction can be "both", "forward", "backward"

Forward: Start at empty model and add important variables

Backward: Start at full model and remove variable with minimal impact

Both: Start at nothing and add, however allows computer to remove or go backwards to provide optimal output

Limitations:

- Computational Demand: Realistically you always want to use both in prediction as it will always provide the best output (forward is good for discovery though). However, in cases where your model has extensive dimensions you may opt for forward or backward due to computational demand of using the "both" setting with lots of data.