

DMAC Training Modules

Kyle Roell, Lauren Koval, Julia Rager

2021-06-23

Contents

1	Introduction	5
2	Setting Up Your R Environment	7
2.1	R and RStudio	7
2.2	Scripting Basics	13
3	The Basics for Data Organization	17
3.1	Basic Data Manipulation	17
4	Finding and Visualizing Data Trends	23
4.1	Basic Statistical Tests and Visualizations of Data	23
4.2	Heat maps	29
4.3	Clustering	29
4.4	Data reduction (PCA)	29
5	Multi-Omics Analyses for Environmental Health	31
5.1	Exposomics	31
5.2	Transcriptomics	31
5.3	Genome-wide MicroRNA	31
5.4	Genome-wide DNA Methylation	31
5.5	Proteomics	32
6	Mixtures Analyses for Environmental Health	33
6.1	Sufficient Similarity	33
6.2	Mixtures Modeling through qgcomp	33

7	Environmental Health Databases	35
7.1	Comparative Toxicogenomics Database (CTD)	35
7.2	Gene Expression Omnibus (GEO)	35
7.3	NHANES	35

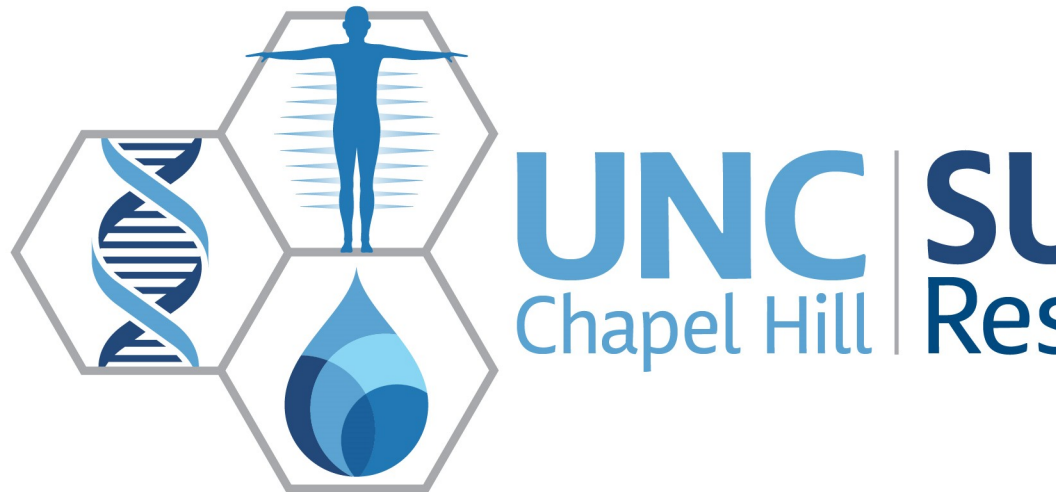
Chapter 1

Introduction

The UNC-Superfund Research Program (SRP) seeks to develop new solutions for reducing exposure to inorganic arsenic and prevent arsenic-induced diabetes through mechanistic and translational research.

The Data Analysis and Management Core (DMAC) provide the UNC-Superfund Research Program with critical expertise in bioinformatics, statistics, data management and data integration. Our goal is to support the data management, integration, and analysis needs of the researchers to reveal multi-factorial determinants of inorganic arsenic-induced metabolic dysfunction/diabetes.

All code for these modules can be found at the [UNC-SRP Github Page](#).



Chapter 2

Setting Up Your R Environment

Before learning about data manipulation and statistical methods for analyzing environmental health datasets, we will provide a brief introduction to R, RStudio, and setting up an R environment and simple scripts.

2.1 R and RStudio

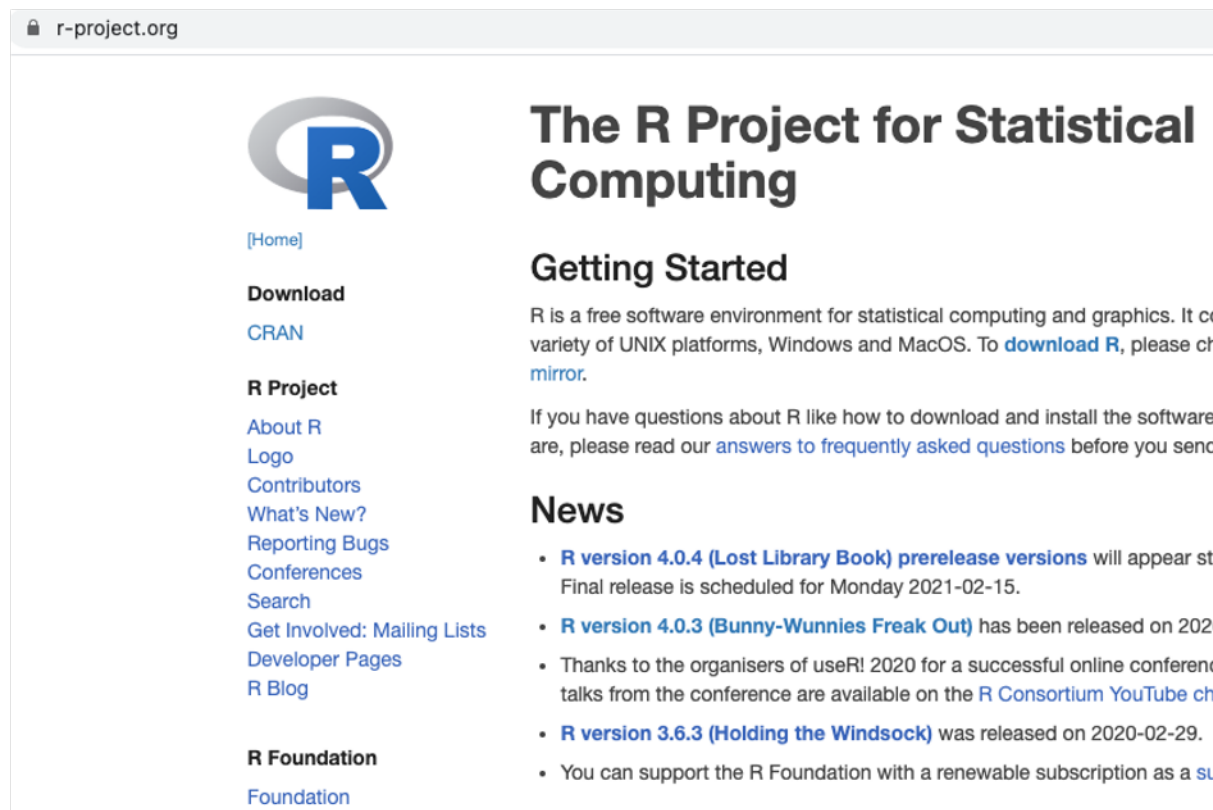
R is a free, open source programming language for statistical computing and graphics that anyone can download and use. It doesn't require a license and is good for reproducible analyses. There exists a large, diverse collection of packages and very comprehensive documentation.

It is easy to download, install, and setup R. Additionally, RStudio is an open source integrated development environment for R. RStudio makes programming in R and using R scripts and features more user friendly. R should be downloaded prior to downloading RStudio.

2.1.1 Downloading R and RStudio

The following is a walkthrough on how to download R and RStudio.

1. **Navigate to R Website**
2. **Select the appropriate CRAN mirror (Duke is fastest if at UNC)**
3. **Select the appropriate R distribution**
4. **Download R**
5. **Navigate to RStudio website and download RStudio (free edition)**

Figure 2.1: R Website, <https://www.r-project.org>

CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud	https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rstudio
Algeria	https://cran.usthb.dz/	University of Science and Technology Houari Boumediene
Argentina	http://mirror.fcaglp.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	https://cran.csiro.au/ https://mirror.aarnet.edu.au/pub/CRAN/ https://cran.ms.unimelb.edu.au/ https://cran.curtin.edu.au/	CSIRO AARNET School of Mathematics and Statistics, University of Melbourne Curtin University
Austria	https://cran.wu.ac.at/	Wirtschaftsuniversität Wien
Belgium	https://www.freeststatistics.org/cran/ https://lib.ugent.be/CRAN/	Patrick Wessa Ghent University Library
Brazil	https://nbcgib.uesc.br/mirrors/cran/ https://cran-r-c3sl.ufpr.br/ https://cran.fiocruz.br/ https://vps.fmvz.usp.br/CRAN/ https://briegeer.esalq.usp.br/CRAN/	Computational Biology Center at Universidade Estadual de Santa Cruz Universidade Federal do Parana Oswaldo Cruz Foundation, Rio de Janeiro University of Sao Paulo, Sao Paulo University of Sao Paulo, Piracicaba



USA	https://mirror.las.iastate.edu/CRAN/ http://ftp.usgs.iu.edu/CRAN/ https://rweb.crmdb.ku.edu/cran/ https://repo.miserver.it.umich.edu/cran/ http://cran.wustl.edu/ http://archive.linux.duke.edu/cran/ https://cran.case.edu/ https://ftp.osuosl.org/pub/cran/ http://lib.stat.cmu.edu/R/CRAN/ http://cran.mirrors.hoobly.com/ https://mirrors.nics.utk.edu/cran/ https://cran.microsoft.com/	Iowa State University, Ames, IA Indiana University University of Kansas, Lawrence, KS MBNI, University of Michigan, Ann Arbor, MI Washington University, St. Louis, MO Duke University, Durham, NC Case Western Reserve University, Cleveland, OH Oregon State University Statlib, Carnegie Mellon University, Pittsburgh, PA Hoobly Classifieds, Pittsburgh, PA National Institute for Computational Sciences, Oak Ridge, TN Revolution Analytics, Dallas, TX
-----	--	---

Figure 2.2: CRAN Mirror, <https://cran.r-project.org/mirrors.html>

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want to download these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code, which must be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2020-10-10, Bunny-Wunnies Freak Out) [R-4.0.3.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features](#) and [bug reports](#) filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read the [frequently asked questions](#) before you send an email.

Figure 2.3: R Download Link, <http://archive.linux.duke.edu/cran/>

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find releases for these systems (which is R 1.7.1) [here](#). Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the [old](#) directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with any downloaded file.

Package binaries for R versions older than 3.2.0 are only available from the [CRAN archive](#) so users of such versions should adjust the CRAN mirror setting (<https://cran-archive.r-project.org/>).

R 4.0.3 "Bunny-Wunnies Freak Out" released on 2020/10/10

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
openssl sha1 R-4.0.3.pkg
```

in the *Terminal* application to print the SHA1 checksum for the R-4.0.3.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-4.0.3.pkg
```

R-4.0.3.pkg (notarized and signed)

SHA1-hash: 8402f586ae1fdb12c6c34c73b286697318db1bc
(ca. 85MB)

[NEWS](#) (for Mac GUI)

[Mac-GUI-1.73.tar.gz](#)

SHA1-hash: 764b1d050757ce78545bdeb9d178a69d13046aa1

Note: Previous R versions for El Capitan can be found in the [el-capitan/base](#) directory.

Latest release:

R 4.0.3 binary for macOS 10.13 (High Sierra) and higher, signed and notarized package. Contains R 4.0.3 framework, R.app for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be omitted when installing, they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires [XQuartz](#) to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version.

Important: this release uses Xcode 10.1 and GNU Fortran 8.2. If you wish to compile R packages from sources, you will need Xcode 10.1 and GNU Fortran 8.2 - see the [tools](#) directory.

News features and changes in the R.app Mac GUI


Sources for the R.app GUI 1.73 for Mac OS X. This file is only needed if you want to join the development of the GUI, it is not for regular users. Read the INSTALL file for further instructions.

Figure 2.4: R Download, <http://archive.linux.duke.edu/cran/bin/macosx/>

RStudio Desktop 1.4.1103

[- Release Notes](#)


1. Install R. RStudio requires R 3.0.1+.
2. Download RStudio Desktop. Recommended for your system:



DOWNLOAD RSTUDIO FOR MAC

1.4.1103 | 152.77MB

Requires macOS 10.13+ (64-bit)



All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy. RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.4.1103.exe	156.96 MB	c3384189
macOS 10.13+	RStudio-1.4.1103.dmg	152.77 MB	20148bd6
Ubuntu 16	rstudio-1.4.1103-amd64.deb	119.26 MB	f0857e27
Ubuntu 18/Debian 10	rstudio-1.4.1103-amd64.deb	120.30 MB	76864349
Fedora 19/Red Hat 7	rstudio-1.4.1103-x86_64.rpm	138.02 MB	8fcb2d29
Fedora 28/Red Hat 8	rstudio-1.4.1103-x86_64.rpm	138.01 MB	e2bf11e9
Debian 9	rstudio-1.4.1103-amd64.deb	120.45 MB	4a4d159c
OpenSUSE 15	rstudio-1.4.1103-x86_64.rpm	122.02 MB	fdc33f7a

Figure 2.5: RStudio Download, <https://rstudio.com/products/rstudio/download/>

2.1.2 Installing R and RStudio

Once R and RStudio have been downloaded, install R first and then RStudio, following the instructions of the installer.

2.1.3 Installing and Loading Packages

Packages in R are units of shareable code that contain functions, data, and documentation on how to use all of these resources. Because R is an open source programming language, packages are constantly being developed and updated. There are many R packages that exist spanning many topics such as graphics and plotting, machine learning, and data manipulation. R packages are often written by R users and submitted to the Comprehensive R Archive Network (CRAN), or another host such as BioConductor or GitHub.

Packages can be installed from the host, but need to be loaded into the workspace. Most of the time, you do not need to download anything from a website. Instead, you can install packages through running code in R or RStudio.

```
install.packages("ggplot2", repos = "https://cran.rstudio.com")
```

Once a package is installed, it needs to be loaded using the *library* function or explicitly referenced to use functions or datasets from that package.

```
library(ggplot2)
```

2.2 Scripting Basics

Before demonstrating the basics of writing R code and scripts, it is worth noting that a function can be queried in RStudio by typing a question mark before the name of the function (e.g. `?install.packages`). This will bring up documentation in the viewer window. Additionally, RStudio will autofill function names, variable names, etc. by pressing tab while typing. If multiple matches are found, RStudio will provide you with a drop down list to select from, which may be useful when searching through newly installed packages or trying to quickly type variable names in an R script.

R also allows for scripts to contain non-code elements, called comments, that will not be run or interpreted. To make a comment, simply use a `#` followed by the comment. A `#` only comments out a single line of code, i.e. only that line will not be run. Comments are useful to help make code more interpretable for others or to add reminders of what and why parts of code may have been written.

```
# This is an R comment!  
  
# Loading ggplot2 package  
library(ggplot2)
```

2.2.1 Setting Working Directory

When working in R, it can be helpful to set the working directory to a local directory where data are located or output files will be saved. The current working directory can also be displayed.

```
# Show current working directory  
getwd()
```

```
## [1] "/Users/kroell/Documents/IEHS/UNC-SRP/test1"
```

```
# Set working directory  
setwd("~/Documents/UNCSR/UNCSR/Data/")
```

2.2.2 Importing and Exporting Files

After setting the working directory, importing and exporting files can be done using various functions based on the type of file being read or written. Often, it is easiest to import data into R that are in a comma separated values, comma delimited, (CSV) file or tab delimited file. Other datatypes such as SAS data files, large csv files, etc. may require different functions to be more efficiently read in and some of these file formats will be discussed in future modules.

```
# Read in CSV data  
csv.dataset = read.csv(file="~/Documents/UNCSR/UNCSR/Data/example_data.csv")  
  
# Read in tab delimited data  
tab.dataset = read.table(file="~/Documents/UNCSR/UNCSR/Data/example_data.txt")
```

There are many ways to export data in R. Data can be written out into a CSV file, tab delimited file, RData file, etc. There are also many functions within packages that write out specific datasets generated by that package.

```
# Write out to a CSV file  
write.csv(csv.output, file="~/Documents/UNCSR/UNCSR/Output/csv_output.csv")  
  
# Write out to a tab delimited file  
write.table(tab.output, file="~/Documents/UNCSR/UNCSR/Output/tsv_output.txt", sep="\t")
```

R also allows objects to be saved in RData files. These files can be read into R, as well, and will load the object into the current workspace. Entire workspaces are also able to be saved.

```
# Read in saved single R data object
r.obj = readRDS(file=~ /Documents/UNCSR/Output/data.rds")

# Write single R object to file
saveRDS(object, file=~ /Documents/UNCSR/Output/single_object.rds")

# Read in multiple saved R objects
load(file=~ /Documents/UNCSR/Output/multiple_data.RData")

# Save multiple R objects
save(object1, object2, file=~ /Documents/UNCSR/Output/multiple_objects.RData")

# Save entire workspace
save.image(file=~ /Documents/UNCSR/Output/entire_workspace.RData")

# Load entire workspace
load(file=~ /Documents/UNCSR/Output/entire_workspace.RData")
```

2.2.3 Viewing Data

After data has been loaded into R, or created within R, it is good to inspect it. Datasets can be viewed in their entirety or subset to quickly look at part of the data.

```
# View first 5 rows of the previously loaded dataset
csv.dataset[1:5,]
```

```
##      Sample Var1 Var2 Var3
## 1 sample1     1    2    1
## 2 sample2     2    4    4
## 3 sample3     3    6    9
## 4 sample4     4    8   16
## 5 sample5     5   10   25
```

```
# View the entire dataset in RStudio
View(csv.dataset)
```


Chapter 3

The Basics for Data Organization

One of the first things that is useful to learn about is data organization and manipulation. While this will not be a complete guide, as there are many things you can do in R to manipulate your data, hopefully this will get you started and will provide you with the fundamentals for working with data in R.

3.1 Basic Data Manipulation

Data manipulation generally refers to organizing and formatting data in a way that makes it easier to read and work with. This can be done through base R operations and functions or through a collection of packages (and philosophy) known as Tidyverse.

In this brief tutorial we will first go over the basic operations and functions you can use to manipulate data, and then show the exact same steps using the Tidyverse packages, functions, and syntax.

3.1.1 Merging

Here, we will be looking at merging as the joining together of two or more datasets, connected using a common identifier, generally some sort of ID. This is useful if you various datasets describing different variables across the same participants. In our example, we will be joining demographic data and environmental exposure data.

```
#First, let's read in our datasets
demo.data = read.csv("~/Downloads/demo_data.csv");
chem.data = read.csv("~/Downloads/chem_data.csv");

#We can merge these by their shared ID column
full.data = merge(demo.data, chem.data, by="ID")

#We could also merge by different ID columns using: by.x = "ID.Demo", by.y="ID.Chem"
```

3.1.2 Filtering & subsetting

Filtering and subsetting data is useful when you need to restrict your dataset to samples or participants with specific conditions. It is also useful for simply removing particular variables or samples.

```
#Let's define a vector of columns we want to keep
subset.columns = c("BMI", "MAge", "MEdu");

#Now we can simply subset our data using those columns
subset.data1 = full.data[,subset.columns];

#If we want to remove those columns, need to identify which columns need removing
remove.columns = colnames(full.data) %in% subset.columns;
cat(remove.columns);
```

```
## FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
#Now we can subset our dataset by only keeping those that we didn't specify
subset.data1a = full.data[,!remove.columns];
```

```
#It is easy to subset data based on rows
#Keeping only first 100 rows
subset.data2 = full.data[1:100,];
#Removing the first 100 rows
subset.data2a = full.data[-c(1:100),];
```

```
#We can also filter data using conditional statements
subset.data3 = full.data[which(full.data$BMI > 25 & full.data$MAge > 31),];
head(subset.data3);
```

```
##      ID      BMI      MAge      MEdu      BW      GA      DWAs      DWCd      DWCr
## 4    4 30.09710 34.81796 3.760030 3266.844 18.60876 5.906656 2.0752589 50.92745
## 5    5 37.41737 42.68440 4.484686 3664.088 32.57351 7.181873 2.7626433 55.16882
```

```
## 9 9 36.94794 33.58589 3.242318 3260.482 37.37241 9.074928 2.7277549 55.72826
## 13 13 33.65870 33.82961 2.924433 3481.293 33.68744 7.101634 0.8443918 47.11677
## 22 22 25.68414 37.08028 2.704409 3387.046 52.48203 7.207447 2.8088453 48.08648
## 31 31 28.39896 47.85761 6.453362 3173.033 20.10080 6.032807 2.1929549 45.71856
##          UAs          UCd          UCr
## 4 8.719123 0.9364825 42.47987
## 5 9.436559 1.4977829 47.78528
## 9 10.818153 1.6585757 42.58577
## 13 9.967185 -0.3466431 36.74220
## 22 9.446643 1.9891049 34.16921
## 31 9.917588 1.1194851 37.82297
```

#This can also be done using the subset function

```
subset.data4 = subset(full.data, BMI > 25 & MAge > 31);
```

#Additionally, we can subset and select only specific columns to keep

```
subset.data4 = subset(full.data, BMI < 22 | BMI > 27, select=c("BMI", "MAge", "MEdu"));
```

3.1.3 Melt and Cast

Melting and casting refers to converting data to “long” or “wide” form. Generally, our data is in wide format, but long format is necessary for some procedures, such as plotting with ggplot2.

#Melt and cast functions from reshape2 package

```
library(reshape2);
```

#Melting data converts it to long format

#Here, we are saying we want a row for each sample (ID) and each column

```
full.melted = melt(full.data, id="ID")
```

#Let's look at the new data

```
head(full.melted);
```

```
## ID variable value
## 1 1 BMI 27.69249
## 2 2 BMI 26.79125
## 3 3 BMI 33.23209
## 4 4 BMI 30.09710
## 5 5 BMI 37.41737
## 6 6 BMI 33.28761
```

#Casting puts data into wide format

#We are telling the cast function to give us a sample (ID)

#for every variable in the variable column

```
full.cast = dcast(full.melted, ID ~ variable);
head(full.cast);
```

```
##   ID      BMI      MAge      MEdu      BW      GA      DWAs      DWcd      DWCr
## 1  1 27.69249 22.99928 3.136759 3180.058 31.79569 6.426464 1.292941 51.67987
## 2  2 26.79125 30.05142 2.656019 3210.823 53.34991 7.832384 1.798535 50.10409
## 3  3 33.23209 28.04660 3.259116 3311.551 53.60599 7.516569 1.288461 48.74001
## 4  4 30.09710 34.81796 3.760030 3266.844 18.60876 5.906656 2.075259 50.92745
## 5  5 37.41737 42.68440 4.484686 3664.088 32.57351 7.181873 2.762643 55.16882
## 6  6 33.28761 24.94960 3.596075 3328.988 38.12531 9.723429 3.054057 51.14812
##           UAs      UCd      UCr
## 1 10.192695 0.7537104 42.60187
## 2 11.815088 0.9789506 41.30757
## 3 10.079057 0.1903262 36.47716
## 4  8.719123 0.9364825 42.47987
## 5  9.436559 1.4977829 47.78528
## 6 11.589403 1.6645837 38.26386
```

3.1.4 Tidyverse

Tidyverse is a collection of packages that are used together along with a specific type of syntax, dataset and formatting protocols, etc.

Here, we will perform all the of the previous exercises using Tidyverse!

```
library(tidyverse);

#Merging
full.tidy = inner_join(demo.data, chem.data, by="ID");

#To merge with different IDs use: by = c("ID.Demo"="ID.Chem")

#Subsetting columns
subset.tidy1 = full.tidy %>% select(all_of(subset.columns));
subset.columns1 = c(subset.columns, "NotAColName");
subset.tidy1a = full.tidy %>% select(any_of(subset.columns1));

#Removing columns
subset.tidy1b = full.tidy %>% select(-subset.columns);

#Subsetting Rows
subset.tidy2a = full.tidy %>% slice(1:100);
subset.tidy2b = full.tidy %>% slice(-c(1:100));
```

```
#Filtering data
```

```
subset.tidy3 = full.tidy %>% filter(BMI > 25 & MAge > 31);  
head(subset.tidy3);
```

```
##   ID      BMI      MAge      MEdu      BW      GA      DWAs      DWCd      DWCr  
## 1  4 30.09710 34.81796 3.760030 3266.844 18.60876 5.906656 2.0752589 50.92745  
## 2  5 37.41737 42.68440 4.484686 3664.088 32.57351 7.181873 2.7626433 55.16882  
## 3  9 36.94794 33.58589 3.242318 3260.482 37.37241 9.074928 2.7277549 55.72826  
## 4 13 33.65870 33.82961 2.924433 3481.293 33.68744 7.101634 0.8443918 47.11677  
## 5 22 25.68414 37.08028 2.704409 3387.046 52.48203 7.207447 2.8088453 48.08648  
## 6 31 28.39896 47.85761 6.453362 3173.033 20.10080 6.032807 2.1929549 45.71856  
##           UAs           UCd           UCr  
## 1  8.719123  0.9364825 42.47987  
## 2  9.436559  1.4977829 47.78528  
## 3 10.818153  1.6585757 42.58577  
## 4  9.967185 -0.3466431 36.74220  
## 5  9.446643  1.9891049 34.16921  
## 6  9.917588  1.1194851 37.82297
```

```
subset.tidy4 = full.tidy %>% filter(BMI > 25 & MAge > 31) %>% select(BMI, MAge, MEdu);
```

```
#Melting and Casting (pivoting in Tidyverse)
```

```
full.pivotlong = full.tidy %>% pivot_longer(-ID, names_to = "var", values_to = "value");  
head(full.pivotlong, 15);
```

```
## # A tibble: 15 x 3  
##       ID var      value  
##   <int> <chr>   <dbl>  
## 1     1 BMI      27.7  
## 2     1 MAge     23.0  
## 3     1 MEdu      3.14  
## 4     1 BW      3180.  
## 5     1 GA       31.8  
## 6     1 DWAs      6.43  
## 7     1 DWCd      1.29  
## 8     1 DWCr     51.7  
## 9     1 UAs      10.2  
## 10    1 UCd       0.754  
## 11    1 UCr      42.6  
## 12    2 BMI      26.8  
## 13    2 MAge     30.1  
## 14    2 MEdu      2.66  
## 15    2 BW      3211.
```

```
full.pivotwide = full.pivotlong %>% pivot_wider(names_from = "var", values_from="value")
head(full.pivotwide);
```

```
## # A tibble: 6 x 12
##       ID   BMI  MAge  MEdu   BW    GA  DWAs  DWCd  DWCr   UAs  UCd  UCr
##   <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1  27.7  23.0  3.14 3180.  31.8  6.43  1.29  51.7  10.2  0.754  42.6
## 2     2  26.8  30.1  2.66 3211.  53.3  7.83  1.80  50.1  11.8  0.979  41.3
## 3     3  33.2  28.0  3.26 3312.  53.6  7.52  1.29  48.7  10.1  0.190  36.5
## 4     4  30.1  34.8  3.76 3267.  18.6  5.91  2.08  50.9   8.72  0.936  42.5
## 5     5  37.4  42.7  4.48 3664.  32.6  7.18  2.76  55.2   9.44  1.50   47.8
## 6     6  33.3  24.9  3.60 3329.  38.1  9.72  3.05  51.1  11.6  1.66   38.3
```

Chapter 4

Finding and Visualizing Data Trends

4.1 Basic Statistical Tests and Visualizations of Data

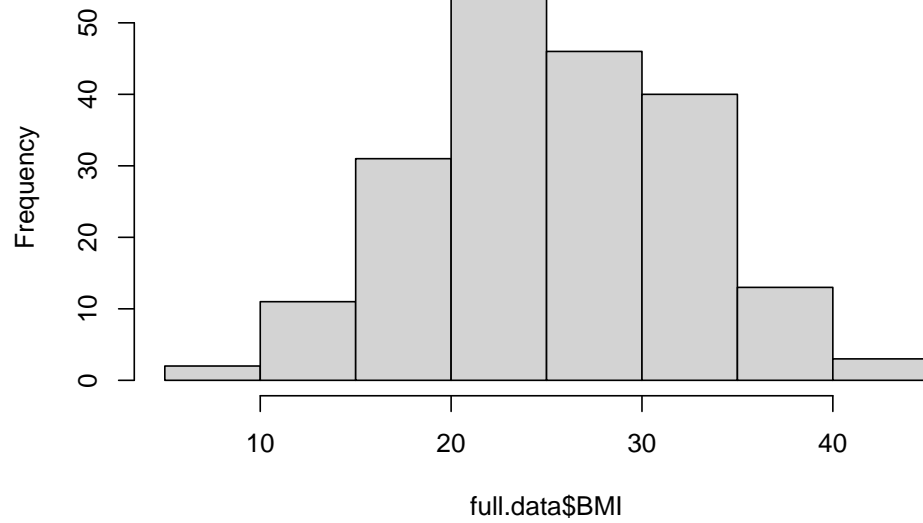
Need an example dataset – maybe ELGAN shuffled/deidentified, with made-up environmental exposure column?

4.1.1 Normality

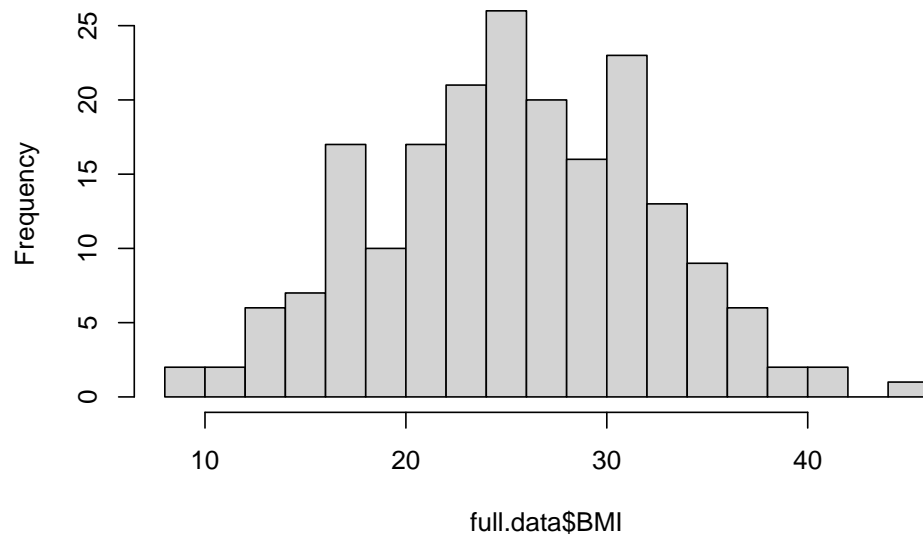
Many statistical tests and methods rely on assumptions of normality. There are a few ways to look at the normality of a dataset, both formally and informally. Plotting data using histograms, densities, or qqplots, can graphically help inform if a variable is normally distributed. There also exist statistical tests, such as the Kolmogorov-Smirnov (K-S) normality test and Shapiro-Wilk's test, that will formally test if data come from a normal distribution. When using these tests, it is important to remember that the null hypothesis is that the sample distribution is normal, and a significant p-value means the distribution is non-normal.

Even when using more formal statistical tests to test for normality, it is important to visualize the data, as well.

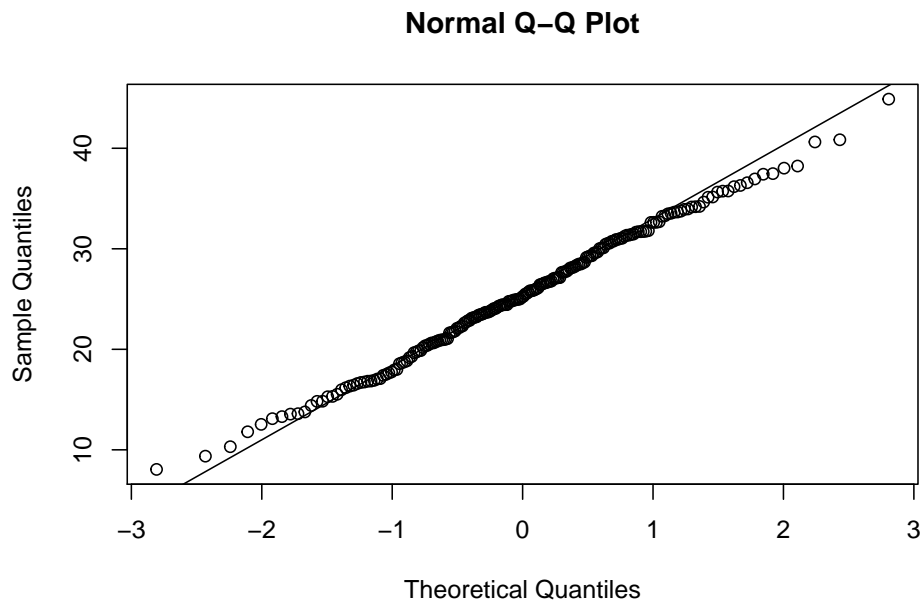
```
#Histograms to visualize data  
hist(full.data$BMI);
```

Histogram of full.data\$BMI

```
#Can decrease bin size (using breaks parameter) to better visualize  
hist(full.data$BMI, breaks=20);
```

Histogram of full.data\$BMI


```
#Also look at normal qqplot using qqnorm function
qqnorm(full.data$BMI);
#Add reference line
qqline(full.data$BMI);
```



From visual inspection, BMI seems to be pretty normally distributed. But, we can also check using a more formal statistical test, the Shapiro-Wilk's test.

```
#Perform Shapiro-Wilk normality test (from base stats package)
shapiro.test(full.data$BMI);
```

```
##
##  Shapiro-Wilk normality test
##
## data:  full.data$BMI
## W = 0.99617, p-value = 0.9014
```

We get a p-value of .9014, so cannot reject the null hypothesis. This means that we can assume normality of this data.

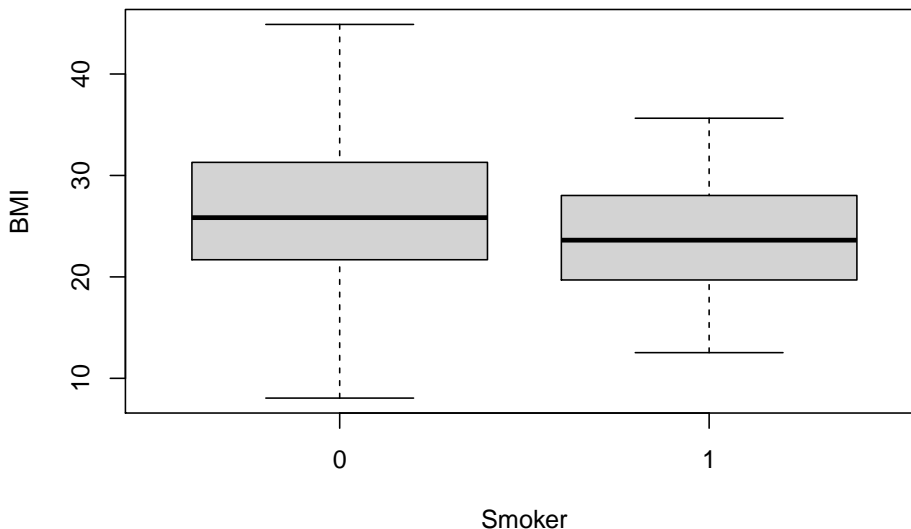
4.1.2 T-tests

T-tests are used to test for a significant difference between the means of two groups. There are a few types of t-tests, but here, we will be comparing BMI

between two groups, smokers and non-smokers, and will be using a two sample t-test (or independent samples t-test).

TALK ABOUT ASSUMPTIONS...

#It is nice to visualize the data across groups using things like boxplots
`boxplot(data=full.data, BMI ~ Smoker);`



#It is easy to perform a t-test on these data using t.test() from base stats package
`t.test(data=full.data, BMI ~ Smoker);`

```
##
## Welch Two Sample t-test
##
## data: BMI by Smoker
## t = 2.1275, df = 96.269, p-value = 0.03593
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.1431141 4.1270354
## sample estimates:
## mean in group 0 mean in group 1
##      25.92961      23.79453
```

#We can also save the results into a variable and access various output values
`ttest.res = t.test(data=full.data, BMI ~ Smoker);`

#For example, we can access the p-value
`ttest.res$p.value;`

```
## [1] 0.03593188
```

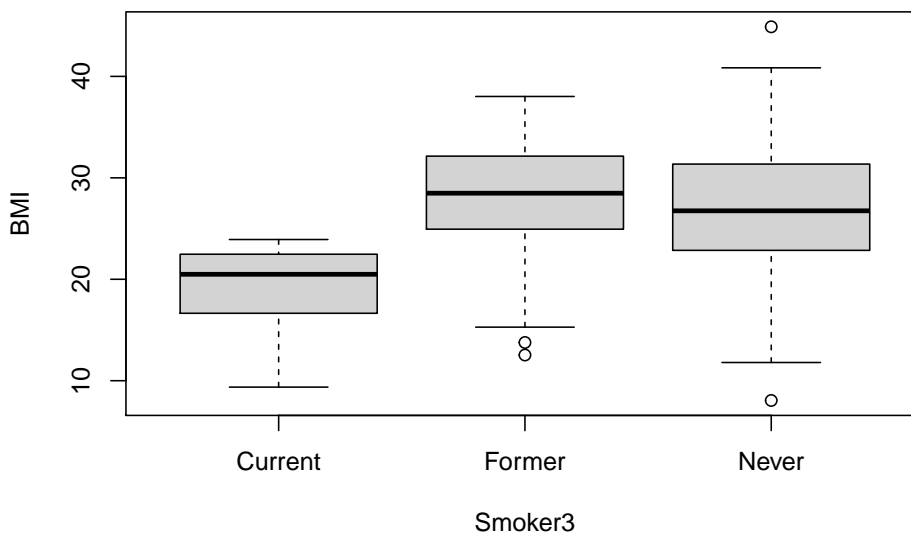
From the boxplots, it's clear that there are differences in the BMIs between smokers and non-smokers. And, in fact, when running the t-test, we see that means across groups are not equal.

4.1.3 ANOVA

Analysis of Variance (ANOVA) is a statistical method that is used to compare means of more than two groups.

TALK ABOUT ASSUMPTIONS and other stuff.

```
#Boxplots to look at data across groups
boxplot(data=full.data, BMI ~ Smoker3);
```



```
#Can also get group means
full.data %>% group_by(Smoker3) %>% summarise(mean(BMI));
```

```
## # A tibble: 3 x 2
##   Smoker3 `mean(BMI)`
##   <chr>      <dbl>
## 1 Current    19.2
## 2 Former     27.7
## 3 Never      26.8
```

```
#Use ANOVA to compare groups (use aov to fit ANOVA model)
aov(data=full.data, BMI ~ Smoker3);
```

```
## Call:
##   aov(formula = BMI ~ Smoker3, data = full.data)
##
## Terms:
##               Smoker3 Residuals
## Sum of Squares 1967.516 7250.059
## Deg. of Freedom      2      197
##
## Residual standard error: 6.066492
## Estimated effects may be unbalanced
```

```
#We can get the typical ANOVA table using either summary or anova on fitted object
anova(aov(data=full.data, BMI ~ Smoker3));
```

```
## Analysis of Variance Table
##
## Response: BMI
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Smoker3      2 1967.5   983.76   26.731 5.357e-11 ***
## Residuals 197 7250.1    36.80
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the ANOVA table, we can conclude that the group means are not all equal.

4.1.4 Regression: linear regression and logistic regression

4.1.5 Chi-squared test – box plots

4.1.6 Fisher’s exact test

4.2 Heat maps

4.2.1 pheatmap

4.2.2 heatmap2

4.2.3 superheat

4.3 Clustering

Examples with genomics: Rager et al. 2014

4.3.1 Hierarchical

4.3.2 K-means

4.4 Data reduction (PCA)

4.4.1 Visualize PCA Plot

4.4.2 Identify % of variance captured

Chapter 5

Multi-Omics Analyses for Environmental Health

5.1 Exposomics

5.1.1 Placenta Exposome

about to be submitted to EI Dust NTA data

5.2 Transcriptomics

5.2.1 DESeq2 / RNAseq

Wildfire dataset, available through GEO

5.3 Genome-wide MicroRNA

Rager et al. 2014 miRNAs

5.4 Genome-wide DNA Methylation

5.4.1 Illumina array data

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE58499>

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE28368>

5.5 Proteomics

Bailey et al arsenic dataset maybe?

Chapter 6

Mixtures Analyses for Environmental Health

6.1 Sufficient Similarity

Botanicals example with chemistry and tox profiling – Julia has dataset

6.2 Mixtures Modeling through qgcomp

Could use published wildfire analysis here (Rager et al. 2021, STOTEN), or online example provide through Alex Keil’s studies

Chapter 7

Environmental Health Databases

7.1 Comparative Toxicogenomics Database
(CTD)

7.2 Gene Expression Omnibus (GEO)

7.3 NHANES