

Essential R Skills

Jon Anderson

2020-06-01

Contents

| | | |
|----------|--|-----------|
| 1 | Motivation | 5 |
| 2 | Getting Started | 7 |
| 2.1 | Packages | 7 |
| 2.2 | The tidyverse Package | 7 |
| 2.3 | Gapminder Data: | 8 |
| 2.4 | Set Working Directory | 8 |
| 2.5 | Reading Data From a CSV File | 8 |
| 3 | Overview of a Dataframe | 11 |
| 3.1 | glimpse | 11 |
| 3.2 | head | 12 |
| 3.3 | summary | 12 |
| 3.4 | Dataframe Details: funModeling package | 13 |
| 3.5 | Dataframe Details: skimr package | 15 |
| 3.6 | describe: Hmisc package | 16 |
| 4 | Introduction to Data Wrangling | 19 |
| 4.1 | Subset using filter | 19 |
| 4.2 | Subset using multiple conditions | 20 |
| 4.3 | Saving as a new dataframe | 21 |
| 4.4 | Subset using select | 22 |
| 4.5 | Order using arrange | 24 |
| 4.6 | Grouped Filter | 26 |
| 4.7 | New Variables Using Mutate | 26 |
| 4.8 | Missing Values | 28 |
| 5 | Univariate Graphical Displays | 31 |
| 5.1 | A Quantitative Variable | 31 |
| 5.2 | Displays of a Categorical Variable | 44 |
| 6 | Summary Statistics For One Variable | 51 |
| 6.1 | One Quantitative Variable | 51 |
| 6.2 | One Categorical Variable | 55 |

| | |
|---|-----------|
| 7 Exploratory Data Analysis For One Quantitative Variable: by Groups | 59 |
| 7.1 Summary Statistics: dplyr | 59 |
| 7.2 Summary Statistics: skimr | 61 |
| 7.3 Graphical Displays of a quantitative variable, separated by groups | 61 |
| 8 Analysis of One Categorical Variable by another categorical variable | 71 |
| 8.1 Tables | 71 |
| 8.2 Graphical Displays | 73 |

Chapter 1

Motivation

We have found that students enter our courses with wide variation in experience and comfort using statistical software for computation and making graphical displays. This document represents our expectations for the basic R skills that students should know upon completing an introductory course in statistics. Analysis methods may appear at times in this document, but the emphasis here is upon basic R usage for data wrangling, and exploratory data analysis using numeric and graphical methods.

Chapter 2

Getting Started

2.1 Packages

When you start R studio, basic functionality is initially available. However, in most projects we will want to use some special code and functions contained in packages that are not initially available whe R starts. Before packages can be used in our analyses, they must be installed in our R workspace. We presume that the R Studio development environment is being used by our students. Any package can be installed by clicking the “Packages” tab in the lower right panel of the R Studio workspace. Then click “Install” to produce an entry bar where you type the name of the desired package.

Once a package is installed into your R Studio environment, you make it available by loading with the `library()` command. For this document, some additional packages are needed, and are loaded in the next code block. The *knitr* and *tidyverse* packages have been previously installed. If you attempt to load a package (in this case *zelig*) that has not been installed, you will get an error message similar to this:

```
Error in library(zelig) : there is no package called ‘zelig’  
# this block loads R packages that may be needed for the analysis.  
library(knitr)  
library(tidyverse)
```

2.2 The tidyverse Package

The tidyverse package is very special - it is a package of other packages. The tidyverse website tidyverse describes the tidyverse as: *The tidyverse is an opin-*

ionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

The most important packages inside the tidyverse package for this document are: dplyr, magrittr, and ggplot.

2.3 Gapminder Data:

This dataset (named gapminder) is contained in an R package called *gapminder*, and needs to be loaded before the dataset can be used.

```
library(gapminder)
```

2.4 Set Working Directory

In the “Files” tab in the lower right portion of the R Studio work area, you can choose where you want to store files and conduct your work by navigating to a suitable folder by making folders and sub-folders and then clicking to navigate to a suitable work area.

You should notify R Studio and the R software to this location called the working directory. Once you have navigated to where you want files, data, and results to reside, you notify R Studio by clicking the blue “More” gear and choose “set working directory.” This will help R understand where to expect files and data to be located.

One of the most common problems students experience is that they work on files in a location not specified as the “working directory.”

2.5 Reading Data From a CSV File

The most common way to read data into R is from an excel spreadsheet that has been saved into a comma-separated-values (csv) file. This means that data elements are separated from each other by commas “,”.

We consider a data file named (file.csv) that contains variable names in the first row of the file. Place this file in your working directory and read,

```
dataframe <- read.csv("file.csv",header=TRUE)
```

The *readr* package inside the *tidyverse* family of packages has a slightly nicer read csv function you should know about. We use the `readr::` prefix to inform readers that the `read_csv` function resides in the *readr* package.


```
dataframe <- readr::read_csv("file.csv", col_names = TRUE)
```

Reading data directly from excel spreadsheets is more complex and you should read documentation for the *readxl* package.

Chapter 3

Overview of a Dataframe

Datasets in R are usually called dataframes or tibbles. The distinction between these names is not important for our purposes - we will usually refer to a dataset as a dataframe.

3.1 glimpse

Let's look at what is inside the `gapminder` dataset using the `glimpse` command from the `dplyr` package. The `dplyr` package is contained in the package “tidyverse” that was loaded previously. The `glimpse(gapminder)` command would have executed without any errors. We use the `dplyr::` prefix to inform readers that the `glimpse` function resides in the `dplyr` package.

```
# the next command would also execute if
# dplyr or tidyverse was loaded..
#glimpse(gapminder)
dplyr::glimpse(gapminder)

## Observations: 1,704
## Variables: 6
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, Afghani...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia,...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997,...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40....
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 1...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134,...
```

This shows it contains economic and demographic information about different countries across years. There are 1704 rows (observations) and 6 columns (variables).

Each variable name is listed along with a variable type designation.

- fct: means a factor variable, also known as a categorical variable.
- int: means a quantitative variable that takes only integer or whole number values.
- dbl: means double precision, a quantitative variable that is essentially continuous - taking decimal values.

3.2 head

By default, the `head` command will show the first 6 rows of the dataset `gapminder`. Datasets in R are called “dataframes.” The `gapminder` dataframe is denoted as a “tibble” which is a type of dataframe.

Options to the `head` command can change the rows displayed.

```
# default is to show 6 rows
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      1952   28.8  8425333    779.
## 2 Afghanistan Asia      1957   30.3  9240934    821.
## 3 Afghanistan Asia      1962   32.0 10267083    853.
## 4 Afghanistan Asia      1967   34.0 11537966    836.
## 5 Afghanistan Asia      1972   36.1 13079460    740.
## 6 Afghanistan Asia      1977   38.4 14880372    786.
```

```
# show only 4 rows...
head(gapminder,n=4)
```

```
## # A tibble: 4 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      1952   28.8  8425333    779.
## 2 Afghanistan Asia      1957   30.3  9240934    821.
## 3 Afghanistan Asia      1962   32.0 10267083    853.
## 4 Afghanistan Asia      1967   34.0 11537966    836.
```

3.3 summary

This command shows a basic summary of the values in each variable.

```
# A basic, base R command
summary(gapminder)
```

```
##          country      continent      year      lifeExp
## Afghanistan: 12 Africa :624 Min. :1952 Min. :23.60
## Albania : 12 Americas:300 1st Qu.:1966 1st Qu.:48.20
## Algeria : 12 Asia :396 Median :1980 Median :60.71
## Angola : 12 Europe :360 Mean :1980 Mean :59.47
## Argentina : 12 Oceania : 24 3rd Qu.:1993 3rd Qu.:70.85
## Australia : 12 Max. :2007 Max. :82.60
## (Other) :1632
##          pop      gdpPercap
## Min. :6.001e+04 Min. : 241.2
## 1st Qu.:2.794e+06 1st Qu.: 1202.1
## Median :7.024e+06 Median : 3531.8
## Mean :2.960e+07 Mean : 7215.3
## 3rd Qu.:1.959e+07 3rd Qu.: 9325.5
## Max. :1.319e+09 Max. :113523.1
##
```

The next command illustrates a “pipe” - here the dataframe `gapminder` is “piped” into the `summary` function to be processed. Note the same output is produced as using `summary(gapminder)`. Note, the pipe operation `%>%` is contained in tidyverse package: *magrittr* which is loaded when *tidyverse* is loaded.

```
# Same idea, but using tidyverse pipe
gapminder %>% summary()
```

```
##          country      continent      year      lifeExp
## Afghanistan: 12 Africa :624 Min. :1952 Min. :23.60
## Albania : 12 Americas:300 1st Qu.:1966 1st Qu.:48.20
## Algeria : 12 Asia :396 Median :1980 Median :60.71
## Angola : 12 Europe :360 Mean :1980 Mean :59.47
## Argentina : 12 Oceania : 24 3rd Qu.:1993 3rd Qu.:70.85
## Australia : 12 Max. :2007 Max. :82.60
## (Other) :1632
##          pop      gdpPercap
## Min. :6.001e+04 Min. : 241.2
## 1st Qu.:2.794e+06 1st Qu.: 1202.1
## Median :7.024e+06 Median : 3531.8
## Mean :2.960e+07 Mean : 7215.3
## 3rd Qu.:1.959e+07 3rd Qu.: 9325.5
## Max. :1.319e+09 Max. :113523.1
##
```

3.4 Dataframe Details: funModeling package

The *funModeling* package contains the `df_status` command which also summarizes a dataframe - showing different aspects like missing values, percentage of

zero values, and also the number of unique values.

```
funModeling::df_status(gapminder)
```

```
##   variable q_zeros p_zeros q_na p_na q_inf p_inf   type unique
## 1  country      0      0  0  0  0  0 factor    142
## 2 continent      0      0  0  0  0  0 factor      5
## 3   year      0      0  0  0  0  0 integer    12
## 4  lifeExp      0      0  0  0  0  0 numeric  1626
## 5    pop      0      0  0  0  0  0 integer  1704
## 6 gdpPercap      0      0  0  0  0  0 numeric  1704
```

```
di=funModeling::data_integrity(gapminder)
```

```
# returns a detailed summary of all variables
print(di)
```

```
## $vars_num_with_NA
## [1] variable q_na      p_na
## <0 rows> (or 0-length row.names)
##
## $vars_cat_with_NA
## [1] variable q_na      p_na
## <0 rows> (or 0-length row.names)
##
## $vars_cat_high_card
##   variable unique
## 1  country    142
##
## $MAX_UNIQUE
## [1] 35
##
## $vars_one_value
## character(0)
##
## $vars_cat
## [1] "country" "continent"
##
## $vars_num
## [1] "year"      "lifeExp"   "pop"       "gdpPercap"
##
## $vars_char
## character(0)
##
## $vars_factor
## [1] "country" "continent"
##
## $vars_other
```

```
## character(0)
```

3.5 Dataframe Details: skimr package

The *skimr* package contains many useful functions for summarizing a dataframe. When we supply a dataframe to the `skim_without_charts` function, dataframe details are separated by variable types.

```
gapminder %>%
  skimr::skim_without_charts()
```

```
## Data Summary
##
## Name Piped data
## Number of rows 1704
## Number of columns 6
## -----
## Column type frequency:
## factor 2
## numeric 4
## -----
## Group variables None
##
## Variable type: factor
## skim_variable n_missing complete_rate ordered n_unique
## 1 country 0 1 FALSE 142
## 2 continent 0 1 FALSE 5
## top_counts
## 1 Afg: 12, Alb: 12, Alg: 12, Ang: 12
## 2 Afr: 624, Asi: 396, Eur: 360, Ame: 300
##
## Variable type: numeric
## skim_variable n_missing complete_rate mean sd p0 p25
## 1 year 0 1 1980. 17.3 1952 1966.
## 2 lifeExp 0 1 59.5 12.9 23.6 48.2
## 3 pop 0 1 29601212. 106157897. 60011 2793664
## 4 gdpPercap 0 1 7215. 9857. 241. 1202.
## p50 p75 p100
## 1 1980. 1993. 2007
## 2 60.7 70.8 82.6
## 3 7023596. 19585222. 1318683096
## 4 3532. 9325. 113523.
```

3.6 describe: Hmisc package

The *Hmisc* package contains the `describe` function that gives a helpful overview of numeric and categorical variables.

```
gapminder %>%
```

```
  Hmisc::describe()
```

```
## .
##
## 6 Variables      1704 Observations
## -----
## country
##      n missing distinct
##    1704      0      142
##
## lowest : Afghanistan      Albania      Algeria      Angola      Argentina
## highest: Vietnam      West Bank and Gaza Yemen, Rep.      Zambia      Zimbabwe
## -----
## continent
##      n missing distinct
##    1704      0      5
##
## lowest : Africa  Americas Asia      Europe  Oceania
## highest: Africa  Americas Asia      Europe  Oceania
##
## Value      Africa Americas      Asia  Europe  Oceania
## Frequency      624      300      396      360      24
## Proportion      0.366      0.176      0.232      0.211      0.014
## -----
## year
##      n missing distinct      Info      Mean      Gmd      .05      .10
##    1704      0      12      0.993      1980      19.87      1952      1957
##      .25      .50      .75      .90      .95
##    1966      1980      1993      2002      2007
##
## lowest : 1952 1957 1962 1967 1972, highest: 1987 1992 1997 2002 2007
##
## Value      1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002
## Frequency      142 142 142 142 142 142 142 142 142 142 142
## Proportion 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083 0.083
##
## Value      2007
## Frequency      142
## Proportion 0.083
## -----
```



```
## lifeExp
##      n missing distinct      Info      Mean      Gmd      .05      .10
##    1704         0      1626         1    59.47    14.82    38.49    41.51
##      .25        .50        .75        .90        .95
##    48.20     60.71     70.85     75.10     77.44
##
## lowest : 23.599 28.801 30.000 30.015 30.331, highest: 81.701 81.757 82.000 82.208 82.603
## -----
## pop
##      n missing distinct      Info      Mean      Gmd      .05      .10
##    1704         0      1704         1 29601212 46384459 475459 946367
##      .25        .50        .75        .90        .95
##   2793664 7023596 19585222 54801370 89822054
##
## lowest :      60011      61325      63149      65345      70787
## highest: 1110396331 1164970000 1230075000 1280400000 1318683096
## -----
## gdpPercap
##      n missing distinct      Info      Mean      Gmd      .05      .10
##    1704         0      1704         1    7215    8573    548.0    687.7
##      .25        .50        .75        .90        .95
##   1202.1   3531.8   9325.5  19449.1  26608.3
##
## lowest :    241.1659    277.5519    298.8462    299.8503    312.1884
## highest: 80894.8833 95458.1118 108382.3529 109347.8670 113523.1329
## -----
```


Chapter 4

Introduction to Data Wrangling

In this chapter we present some very basic data handling and processing functions (data wrangling) that will be necessary for doing basic analyses, comparisons, and graphics. Most of the commands presented in this section stress the functions and R packages in the *tidyverse* - a set or family of packages that have similar syntax and behaviors.

4.1 Subset using filter

Suppose we wish to examine a subset of data for only one country, Jon’s favorite country, Australia!! The following code starts by taking the `gapminder` dataset and then “pipes” it into the filtering (selecting rows) action so that only dataset rows from Australia are selected. The pipe function is `%>%` and is similar to a plumbing pipe that goes one direction: from left to right. After the “Australia” rows are selected, the result is “piped” into the `head` function for display. The `head` function says show the top 12 rows. When no rows are specified in the `head` function, the default is 6 rows. Note that the `filter` function resides in the *dplyr* package within the *tidyverse* family.

If the *tidyverse* or *dplyr* packages have been loaded, you don’t need to supply the `dplyr::` prefix to the `filter` command.

```
#gapminder %>% filter(country=="Australia") %>% head(n=12)
gapminder %>% dplyr::filter(country=="Australia") %>% head(n=12)
```

```
## # A tibble: 12 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>      <int>  <dbl>    <int>    <dbl>
```

```
## 1 Australia Oceania 1952 69.1 8691212 10040.
## 2 Australia Oceania 1957 70.3 9712569 10950.
## 3 Australia Oceania 1962 70.9 10794968 12217.
## 4 Australia Oceania 1967 71.1 11872264 14526.
## 5 Australia Oceania 1972 71.9 13177000 16789.
## 6 Australia Oceania 1977 73.5 14074100 18334.
## 7 Australia Oceania 1982 74.7 15184200 19477.
## 8 Australia Oceania 1987 76.3 16257249 21889.
## 9 Australia Oceania 1992 77.6 17481977 23425.
## 10 Australia Oceania 1997 78.8 18565243 26998.
## 11 Australia Oceania 2002 80.4 19546792 30688.
## 12 Australia Oceania 2007 81.2 20434176 34435.
```

4.2 Subset using multiple conditions

Let's select by continent and year. The `head` function will then show some of the rows selected. Here the `gapminder` dataframe is piped to the `filter` function to select rows to be further piped to the `head()` function for display. The logical condition inside `filter` restricts continent to "Oceania" AND (AND condition is "&") year to be 1997. Both of these conditions must be TRUE for the row to enter the dataframe to be displayed by the `head()` function.

```
gapminder %>%
  dplyr::filter(continent=="Oceania" & year==1997) %>%
  head()
```

```
## # A tibble: 2 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int> <dbl>    <int>    <dbl>
## 1 Australia Oceania    1997  78.8 18565243  26998.
## 2 New Zealand Oceania    1997  77.6  3676187  21050.
```

The next example uses an "or" condition to specify the desired rows in the first `filter` expression - the next `filter` permits only observations from 1997.

```
gapminder %>%
  dplyr::filter(continent=="Oceania" | continent=="Americas") %>%
  dplyr::filter(year==1997) %>%
  head()
```

```
## # A tibble: 6 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int> <dbl>    <int>    <dbl>
## 1 Argentina Americas    1997  73.3 36203463  10967.
## 2 Australia Oceania    1997  78.8 18565243  26998.
## 3 Bolivia   Americas    1997  62.0  7693188   3326.
## 4 Brazil    Americas    1997  69.4 168546719  7958.
```

```
## 5 Canada      Americas  1997    78.6 30305843    28955.
## 6 Chile       Americas  1997    75.8 14599929    10118.
```

The next example selects observations/rows from a list of countries and also restricts year to 1997.

```
gapminder %>%
  filter(country %in% c("Australia", "New Zealand", "Argentina") & year==1997) %>%
  head()
```

```
## # A tibble: 3 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>   <int>   <dbl>
## 1 Argentina  Americas  1997   73.3 36203463   10967.
## 2 Australia  Oceania   1997   78.8 18565243   26998.
## 3 New Zealand Oceania   1997   77.6 3676187    21050.
```

The next example selects observations by omitting one continent (Oceania is excluded) and then specifies a year. The code that causes “omit” is the “!=” syntax. In the code `year==1997`, the double equal sign `==` means make a logical check if year is 1997. Only rows where both aspects of the filter conditions pass through to be displayed by `head`. Again, the logical operator “AND” is expressed by the `&` expression.

```
gapminder %>%
  filter(continent!="Oceania" & year==1997) %>%
  head()
```

```
## # A tibble: 6 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>  <dbl>   <int>   <dbl>
## 1 Afghanistan Asia      1997   41.8 22227415    635.
## 2 Albania    Europe   1997   73.0 3428038    3193.
## 3 Algeria    Africa   1997   69.2 29072015    4797.
## 4 Angola     Africa   1997   41.0 9875024    2277.
## 5 Argentina  Americas 1997   73.3 36203463   10967.
## 6 Austria    Europe   1997   77.5 8069876    29096.
```

Please note that in all the above examples, the `filter` function accepts/rejects rows or observations in a dataframe according to the logical conditions specified inside the filter function.

4.3 Saving as a new dataframe

Here we save the the modified dataset as a new dataframe called `gap97`.

```
gap97 <- gapminder %>%
  filter(continent!="Oceania" & year==1997)
```

```
#
dplyr::glimpse(gap97)

## Observations: 140
## Variables: 6
## $ country <fct> Afghanistan, Albania, Algeria, Angola, Argentina, Austria, ...
## $ continent <fct> Asia, Europe, Africa, Africa, Americas, Europe, Asia, Asia,...
## $ year <int> 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997,...
## $ lifeExp <dbl> 41.763, 72.950, 69.152, 40.963, 73.275, 77.510, 73.925, 59...
## $ pop <int> 22227415, 3428038, 29072015, 9875024, 36203463, 8069876, 59...
## $ gdpPercap <dbl> 635.3414, 3193.0546, 4797.2951, 2277.1409, 10967.2820, 2909...
```

4.4 Subset using select

The `filter` function controls the rows of the dataframe. Sometimes we might want to include only a few of the variables (columns) in a dataset. We frequently want to create a data subset with only a few variables when the original dataset has hundreds of variables. The `select` function is used to select and rename variables.

```
# the next command selects three variables and renames two of them:
gapminder %>% dplyr::select(country, Year=year, LifeExp=lifeExp) %>% head()
```

```
## # A tibble: 6 x 3
##   country      Year LifeExp
##   <fct>      <int>   <dbl>
## 1 Afghanistan 1952    28.8
## 2 Afghanistan 1957    30.3
## 3 Afghanistan 1962    32.0
## 4 Afghanistan 1967    34.0
## 5 Afghanistan 1972    36.1
## 6 Afghanistan 1977    38.4
```

```
# to change the order of display, puts year first in the list of variables
gapminder %>% select(year, everything()) %>% head()
```

```
## # A tibble: 6 x 6
##   year country      continent lifeExp      pop gdpPercap
##   <int> <fct>      <fct>      <dbl>    <int>    <dbl>
## 1 1952 Afghanistan Asia          28.8  8425333    779.
## 2 1957 Afghanistan Asia          30.3  9240934    821.
## 3 1962 Afghanistan Asia          32.0 10267083    853.
## 4 1967 Afghanistan Asia          34.0 11537966    836.
## 5 1972 Afghanistan Asia          36.1 13079460    740.
## 6 1977 Afghanistan Asia          38.4 14880372    786.
```

The `profiling_num` command from the *funModeling* package produces a lot of output, some we might not want. We will show how to modify the output of this command here. The command produces a dataframe which has many columns we might not wish to display or consider further.

We begin by removing some columns of summary statistics that we wish to ignore. Selecting a list of column names with a “minus” - sign in front of the list will remove these items from the dataframe and keep the rest in place. The command below pipes the modified dataframe to the `kable` command in the *knitr* package for a more pleasing tabular display.

```
# Let's observe the contents of profiling_num:
funModeling::profiling_num(gapminder) %>% dplyr::glimpse()

## Observations: 4
## Variables: 16
## $ variable      <chr> "year", "lifeExp", "pop", "gdpPercap"
## $ mean          <dbl> 1.979500e+03, 5.947444e+01, 2.960121e+07, 7.215327e+03
## $ std_dev       <dbl> 1.726533e+01, 1.291711e+01, 1.061579e+08, 9.857455e+03
## $ variation_coef <dbl> 0.008722066, 0.217187544, 3.586268548, 1.366182632
## $ p_01          <dbl> 1952.0000, 33.4926, 154117.9200, 369.2201
## $ p_05          <dbl> 1952.0000, 38.4924, 475458.9000, 547.9964
## $ p_25          <dbl> 1965.750, 48.198, 2793664.000, 1202.060
## $ p_50          <dbl> 1979.5000, 60.7125, 7023595.5000, 3531.8470
## $ p_75          <dbl> 1.993250e+03, 7.084550e+01, 1.958522e+07, 9.325462e+03
## $ p_95          <dbl> 2007.000, 77.437, 89822054.500, 26608.333
## $ p_99          <dbl> 2.007000e+03, 8.023892e+01, 6.319900e+08, 3.678357e+04
## $ skewness      <dbl> 0.0000000, -0.2524798, 8.3328742, 3.8468819
## $ kurtosis      <dbl> 1.783217, 1.873099, 80.716151, 30.431702
## $ iqr           <dbl> 2.750000e+01, 2.264750e+01, 1.679156e+07, 8.123402e+03
## $ range_98      <chr> "[1952, 2007]", "[33.4926, 80.23892]", "[154117.92, 63..."
## $ range_80      <chr> "[1957, 2002]", "[41.5108, 75.097]", "[946367.1, 54801..."

# now remove unwanted columns from summary display
funModeling::profiling_num(gapminder) %>%
  select(-c("variation_coef", "skewness", "kurtosis", "range_98", "range_80", "p_01", "p_99")) %>%
  knitr::kable()
```

| variable | mean | std_dev | p_05 | p_25 | p_50 | p_75 | |
|-----------|--------------|--------------|-------------|-------------|--------------|--------------|--------------|
| year | 1.979500e+03 | 1.726533e+01 | 1952.0000 | 1965.750 | 1979.5000 | 1.993250e+03 | 2007.000 |
| lifeExp | 5.947444e+01 | 1.291711e+01 | 38.4924 | 48.198 | 60.7125 | 7.084550e+01 | 77.437 |
| pop | 2.960121e+07 | 1.061579e+08 | 475458.9000 | 2793664.000 | 7023595.5000 | 1.958522e+07 | 89822054.500 |
| gdpPercap | 7.215327e+03 | 9.857455e+03 | 547.9964 | 1202.060 | 3531.8470 | 9.325462e+03 | 26608.333 |

In the next command we take a different approach - we explicitly select the statistics (columns) we want to keep and display. The most commonly used summaries are chosen.

```
funModeling::profiling_num(gapminder) %>%
  select(c("variable", "mean", "std_dev", "p_25", "p_50", "p_75")) %>%
  knitr::kable()
```

| variable | mean | std_dev | p_25 | p_50 | p_75 |
|-----------|--------------|--------------|-------------|--------------|--------------|
| year | 1.979500e+03 | 1.726533e+01 | 1965.750 | 1979.5000 | 1.993250e+03 |
| lifeExp | 5.947444e+01 | 1.291711e+01 | 48.198 | 60.7125 | 7.084550e+01 |
| pop | 2.960121e+07 | 1.061579e+08 | 2793664.000 | 7023595.5000 | 1.958522e+07 |
| gdpPercap | 7.215327e+03 | 9.857455e+03 | 1202.060 | 3531.8470 | 9.325462e+03 |

4.5 Order using arrange

Sometimes we might want to know the countries with the largest or smallest values of some variables. In the following examples we sort/order by the values of life expectancy. In the code below, when we use the command `filter(year==1997)`, the double equal sign means make a logical check if year is 1997, and only allow dataframe rows where this is true to pass through to the next stage of the analysis pipeline.

```
# This command will show the countries with highest life expectancy because
# the data are arranged in descending order of life expectancy (larger to smaller)
gapminder %>%
  dplyr::filter(year==1997) %>%
  dplyr::select(country, continent, lifeExp) %>%
  dplyr::arrange(desc(lifeExp)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   country      continent lifeExp
##   <fct>        <fct>      <dbl>
## 1 Japan        Asia         80.7
## 2 Hong Kong, China Asia         80
## 3 Sweden        Europe        79.4
## 4 Switzerland  Europe        79.4
## 5 Iceland       Europe        79.0
## 6 Australia     Oceania       78.8
```

```
# This command uses the default ascending (increasing) order with
# respect to life expectancy (order smaller to larger)
gapminder %>%
  filter(year==1997) %>%
  select(country, continent, lifeExp) %>%
  arrange(lifeExp) %>%
  head()
```

```
## # A tibble: 6 x 3
```



```
##   country      continent lifeExp
##   <fct>        <fct>      <dbl>
## 1 Rwanda      Africa      36.1
## 2 Sierra Leone Africa      39.9
## 3 Zambia       Africa      40.2
## 4 Angola       Africa      41.0
## 5 Afghanistan Asia        41.8
## 6 Liberia      Africa      42.2
```

The `top_n` function from the *dplyr* package will select the `n` rows with the largest values of a variable. This is similar to the code above that orders the rows - then use `head` function to select the number of desired rows.

This first example uses the default alphabetical ordering of country name.

```
gapminder %>%
  filter(year==1997) %>%
  select(country, continent, lifeExp) %>%
  dplyr::top_n(n=6, wt=lifeExp) %>%
  knitr::kable()
```

| country | continent | lifeExp |
|------------------|-----------|---------|
| Australia | Oceania | 78.83 |
| Hong Kong, China | Asia | 80.00 |
| Iceland | Europe | 78.95 |
| Japan | Asia | 80.69 |
| Sweden | Europe | 79.39 |
| Switzerland | Europe | 79.37 |

The results can then be ordered by the life expectancy:

```
gapminder %>%
  filter(year==1997) %>%
  select(country, continent, lifeExp) %>%
  dplyr::top_n(n=6, wt=lifeExp) %>%
  dplyr::arrange(desc(lifeExp)) %>%
  knitr::kable()
```

| country | continent | lifeExp |
|------------------|-----------|---------|
| Japan | Asia | 80.69 |
| Hong Kong, China | Asia | 80.00 |
| Sweden | Europe | 79.39 |
| Switzerland | Europe | 79.37 |
| Iceland | Europe | 78.95 |
| Australia | Oceania | 78.83 |

The countries with the largest life expectancy can then be ordered by another variable like population. Here we find the 6 countries in 1997 with the highest life expectancy - then display them in order of population size.

```
gapminder %>%
  filter(year==1997) %>%
  select(country, continent, lifeExp, pop) %>%
  dplyr::top_n(n=6, wt=lifeExp) %>%
  dplyr::arrange(desc(pop)) %>%
  knitr::kable()
```

| country | continent | lifeExp | pop |
|------------------|-----------|---------|-----------|
| Japan | Asia | 80.69 | 125956499 |
| Australia | Oceania | 78.83 | 18565243 |
| Sweden | Europe | 79.39 | 8897619 |
| Switzerland | Europe | 79.37 | 7193761 |
| Hong Kong, China | Asia | 80.00 | 6495918 |
| Iceland | Europe | 78.95 | 271192 |

4.6 Grouped Filter

Another useful verb in the *tidyverse* is `group_by`. Suppose we wanted to view the two countries with the highest life expectancy in 1997, in each continent.

```
gapminder %>%
  filter(year==1997) %>%
  select(country, continent, lifeExp, pop) %>%
  dplyr::group_by(continent) %>%
  dplyr::top_n(n=2, wt=lifeExp) %>%
  dplyr::arrange(continent) %>%
  knitr::kable()
```

| country | continent | lifeExp | pop |
|------------------|-----------|---------|-----------|
| Reunion | Africa | 74.772 | 684810 |
| Tunisia | Africa | 71.973 | 9231669 |
| Canada | Americas | 78.610 | 30305843 |
| Costa Rica | Americas | 77.260 | 3518107 |
| Hong Kong, China | Asia | 80.000 | 6495918 |
| Japan | Asia | 80.690 | 125956499 |
| Sweden | Europe | 79.390 | 8897619 |
| Switzerland | Europe | 79.370 | 7193761 |
| Australia | Oceania | 78.830 | 18565243 |
| New Zealand | Oceania | 77.550 | 3676187 |

4.7 New Variables Using Mutate

In many problems we may wish to create a new variable based on an existing variable. Here we illustrate by making a new variable - the natural logarithm

of population - based on the original variable `pop`.

```
gapminder %>%
  dplyr::mutate(logpopulation = log(pop)) %>%
  dplyr::glimpse()

## Observations: 1,704
## Variables: 7
## $ country    <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, Afg...
## $ continent  <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, A...
## $ year       <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1...
## $ lifeExp    <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854,...
## $ pop        <int> 8425333, 9240934, 10267083, 11537966, 13079460, 1488037...
## $ gdpPercap  <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1...
## $ logpopulation <dbl> 15.94675, 16.03915, 16.14445, 16.26115, 16.38655, 16.51...
```

If I want to change the name of the new variable from `logpopulation` to something shorter like `logPop`, we could re-run the `mutate` command, or use a `rename` function.

In addition we create a new version of the `gapminder` dataset that contains the new variable - called `gapVers1`. This dataframe is now available to be used in the ongoing analysis.

```
gapVers1 <- gapminder %>%
  dplyr::mutate(logpopulation = log(pop)) %>%
  dplyr::rename(logPop=logpopulation)
#
dplyr::glimpse(gapVers1)

## Observations: 1,704
## Variables: 7
## $ country    <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, Afghani...
## $ continent  <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia,...
## $ year       <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997,...
## $ lifeExp    <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40...
## $ pop        <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 1...
## $ gdpPercap  <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134,...
## $ logPop     <dbl> 15.94675, 16.03915, 16.14445, 16.26115, 16.38655, 16.51555,...
```

The next code uses a `mutate` command with logical conditions to make a new, two-level categorical variable `region` as a character variable. Then we use `mutate` again to convert `region` (character) to a factor variable named `regionf`. In statistical models, factor variables are preferred, but in data handling stages, character versions are probably easier to manipulate.

The `if_else` function from `dplyr` has the form ‘`if_else(logical condition,value if TRUE, value if FALSE)`’.

The next example uses the “T-pipe” function `%T>%` to break the piping so that the result of the second `mutate` flows to both `glimpse` and to `head` - in this construction, it is understood the output of `glimpse` does not pipe to `head`, but rather the original data flow from the second `mutate`.

```
gapminder %>%
  dplyr::mutate(region = if_else(country=="Oceania","Oceania","NotOceania")) %>%
  dplyr::mutate(regionf = as_factor(region)) %T>%
  dplyr::glimpse() %>%
  head()
```

```
## Observations: 1,704
## Variables: 8
## $ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, Afghani...
## $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia,...
## $ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997,...
## $ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 40...
## $ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372, 1...
## $ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.1134,...
## $ region    <chr> "NotOceania", "NotOceania", "NotOceania", "NotOceania", "No...
## $ regionf   <fct> NotOceania, NotOceania, NotOceania, NotOceania, NotOceania,...
```

```
## # A tibble: 6 x 8
##   country    continent year lifeExp   pop gdpPercap region  regionf
##   <fct>      <fct>    <int> <dbl>   <int>   <dbl> <chr>    <fct>
## 1 Afghanistan Asia      1952  28.8  8425333   779. NotOceania NotOceania
## 2 Afghanistan Asia      1957  30.3  9240934   821. NotOceania NotOceania
## 3 Afghanistan Asia      1962  32.0 10267083   853. NotOceania NotOceania
## 4 Afghanistan Asia      1967  34.0 11537966   836. NotOceania NotOceania
## 5 Afghanistan Asia      1972  36.1 13079460   740. NotOceania NotOceania
## 6 Afghanistan Asia      1977  38.4 14880372   786. NotOceania NotOceania
```

4.8 Missing Values

If a variable is not complete and contains empty places, these are denoted in R as `NA`. We will often wish to create a dataframe without any missing values, or discover how many rows contain variables with missing values.

First let's create a small dataset with missing values:

```
x <- c(1,2,NA,4)
y <- c(11,12,13,NA)
z <- c(7,8,9,10)
tempdf <- data.frame(x,y,z)
tempdf
```

```
##      x  y  z
```

```
## 1  1 11  7
## 2  2 12  8
## 3 NA 13  9
## 4  4 NA 10
```

```
# count missing values for variable x
tempdf %>%
  dplyr::summarise(count = sum(is.na(x)))
```

```
##   count
## 1     1
```

```
# count rows with missing y
tempdf %>%
  dplyr::tally(is.na(y))
```

```
##   n
## 1 1
```

```
# subset of rows with complete data for specified columns
tempdf %>%
  dplyr::select(y,z) %>%
  tidyr::drop_na() %>%
  head()
```

```
##   y z
## 1 11 7
## 2 12 8
## 3 13 9
```

```
#
tempdf %>%
  tidyr::drop_na() %>%
  head()
```

```
##   x y z
## 1 1 11 7
## 2 2 12 8
```


Chapter 5

Univariate Graphical Displays

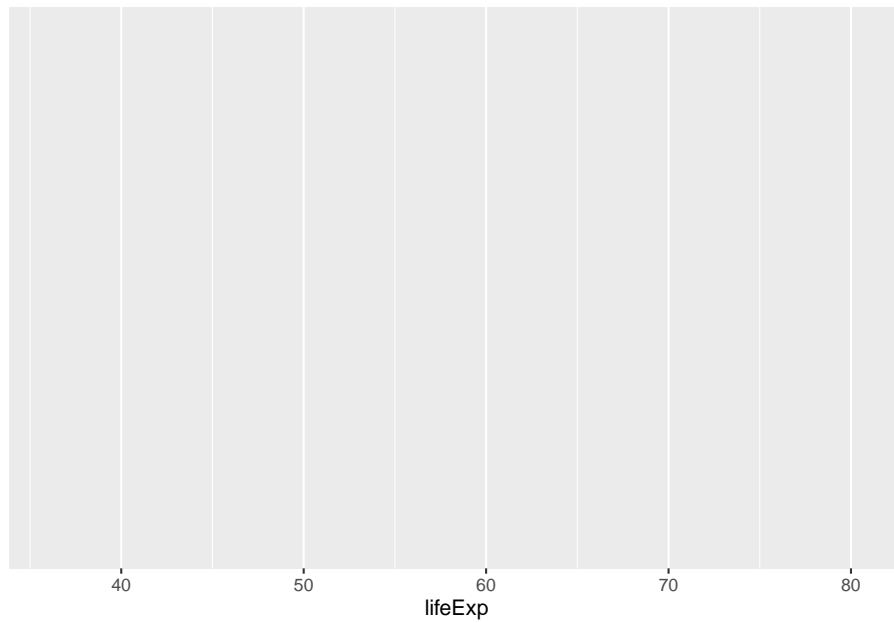
In this section we will show examples of how to create graphical displays of a single variable - with examples for both quantitative and categorical variables. In each example, the first line creates the dataset to be graphed - followed by a command making the display. We will focus on graphical displays made by functions in the *ggplot* family - that is, the *ggplot* package which is also part of the *tidyverse* family of functions. If *tidyverse* is loaded, *ggplot* functions will work without explicitly loading the *ggplot* package.

5.1 A Quantitative Variable

5.1.1 Dotplot

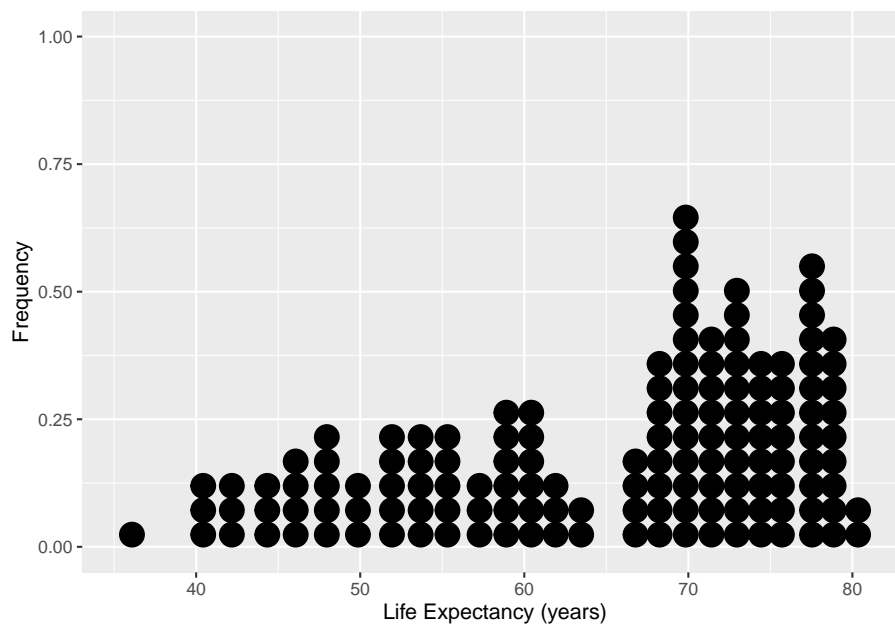
The next block of code takes the *gapminder* dataframe and “pipes” (`%>%`, a pipeline like plumbing) the data through a filter so that only data from year 1997 flows through to define the new dataset named `ds`. The `ggplot` command uses dataset `ds`, and variable `x` life expectancy. The next example shows what using only the `ggplot` command produces an empty graphical region that is awaiting further instructions:

```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp))
```



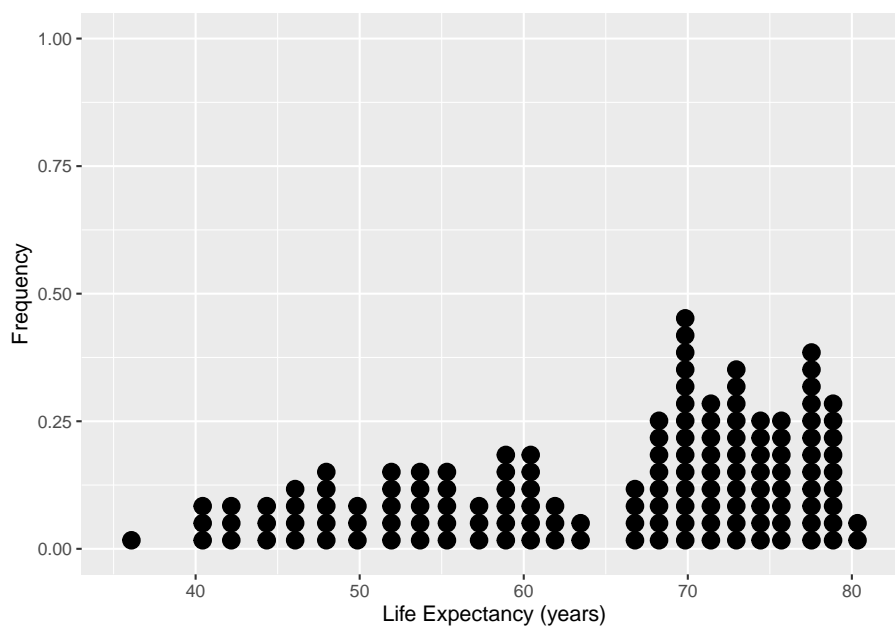
Now we use additional code to place the dotplot in the existing graphical region. In ggplot graphics we make graphical objects with a `geom` function - here a dotplot so we use `geom_dotplot()` to produce the dotplot specified using the variable mappings in the aesthetics command `aes` in the ggplot command.

```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp)) +
  geom_dotplot() +
  xlab("Life Expectancy (years)") + ylab("Frequency")
```

Here we change the default size for the dots.

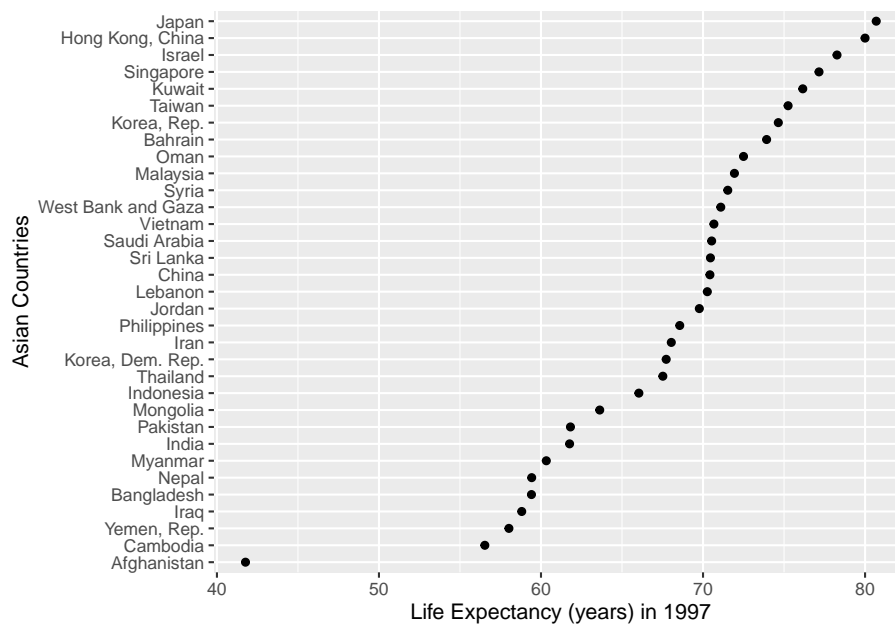
```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp)) +
  geom_dotplot(dotsize=0.70) +
  xlab("Life Expectancy (years)") + ylab("Frequency")
```



5.1.1.1 Dotplot with observations identified and ordered

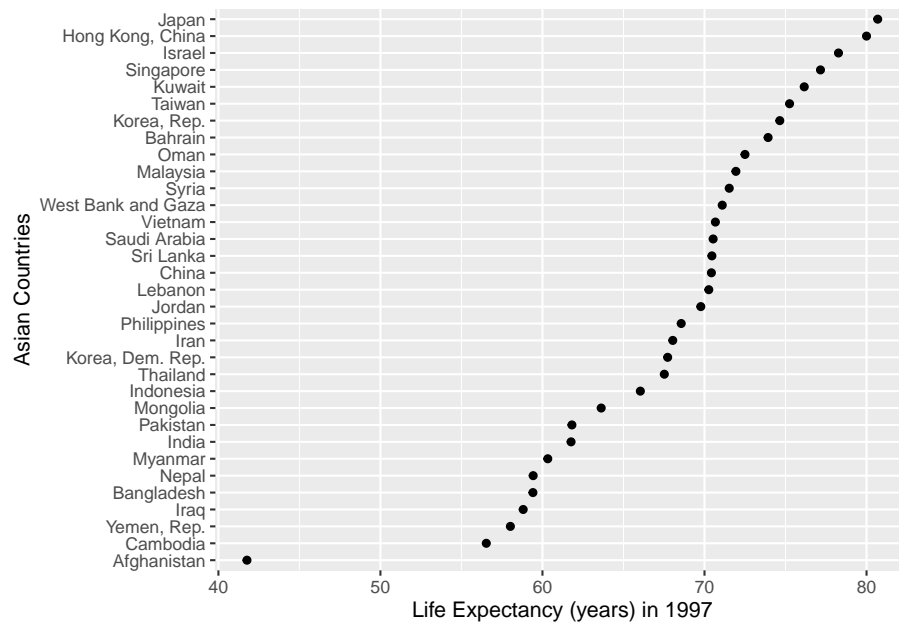
Here we produce a display so that life expectancy is displayed for each country in Asia, and the values are ordered.

```
ds <- gapminder %>% filter(continent=="Asia",year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp, y= reorder(country,lifeExp))) +
  geom_point() +
  xlab("Life Expectancy (years) in 1997") +
  ylab("Asian Countries")
```



Notice that in the next example we simply pipe the modified dataset into the first argument of the `ggplot` command so that there is no need to save the modified dataset to make the display.

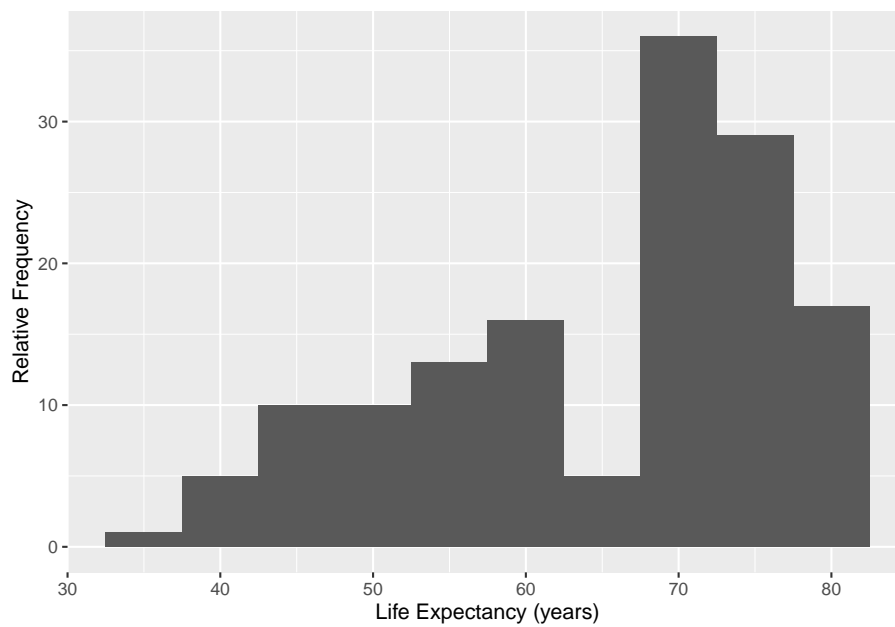
```
gapminder %>% filter(continent=="Asia",year==1997) %>%
ggplot( mapping=aes(x=lifeExp, y= reorder(country,lifeExp))) +
  geom_point() +
  xlab("Life Expectancy (years) in 1997") +
  ylab("Asian Countries")
```



5.1.2 Histogram

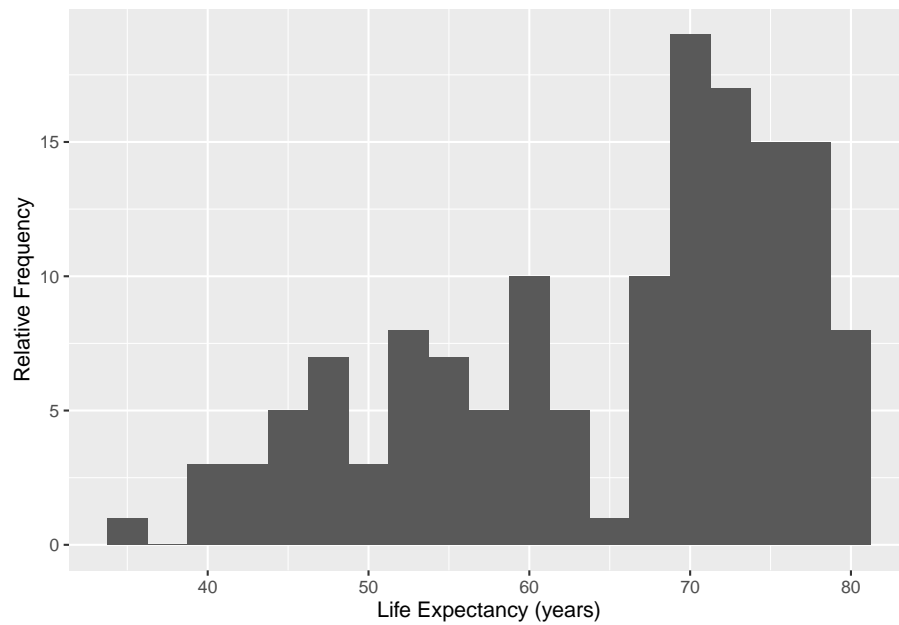
This code block is similar to the dotplot commands, but the `geom_histogram` function controls the bin width in units of the x variable - in this case 5 years.

```
gapminder %>% filter(year==1997) %>%
ggplot(mapping=aes(x=lifeExp)) +
  geom_histogram(binwidth=5) +
  xlab("Life Expectancy (years)") +
  ylab("Relative Frequency")
```



Here we change the binwidth:

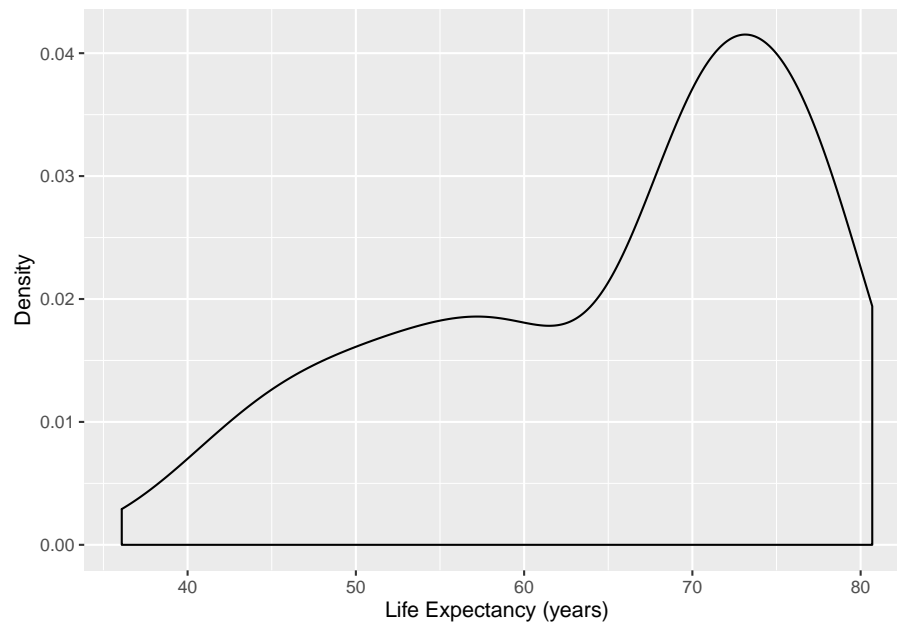
```
gapminder %>% filter(year==1997) %>%  
ggplot(mapping=aes(x=lifeExp)) +  
  geom_histogram(binwidth=2.5) +  
  xlab("Life Expectancy (years)") +  
  ylab("Relative Frequency")
```



5.1.3 Density Plot

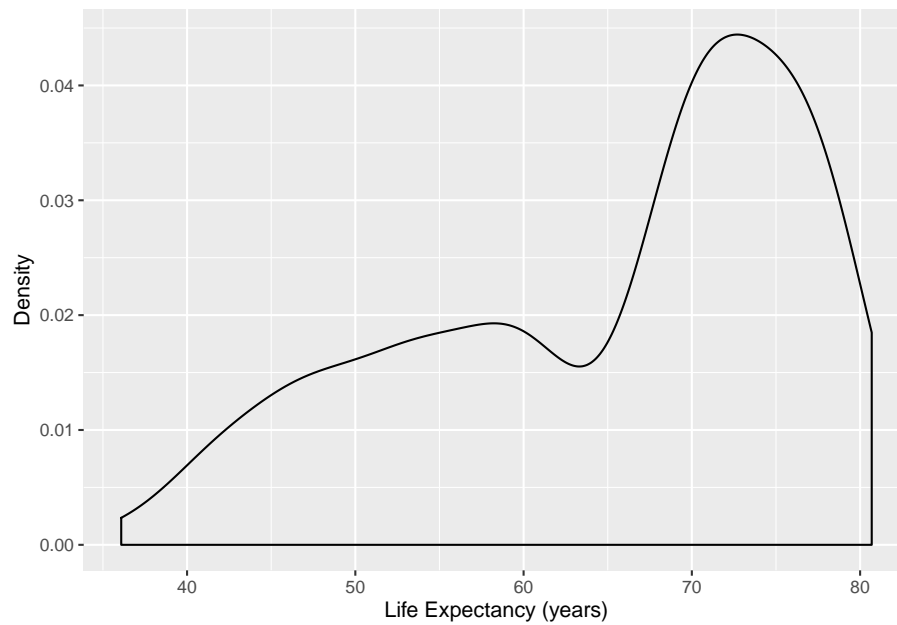
Density plots produces a smoothing of a histogram to display the distribution.

```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp)) +
  geom_density() +
  xlab("Life Expectancy (years)") +
  ylab("Density")
```



The `adjust` option controls the amount of smoothing relative to a default value of 1. A smaller value gives less smoothing (more responsive line to small changes in the data distribution), and larger values will make a smoother curve that is less sensitive to the data pattern.

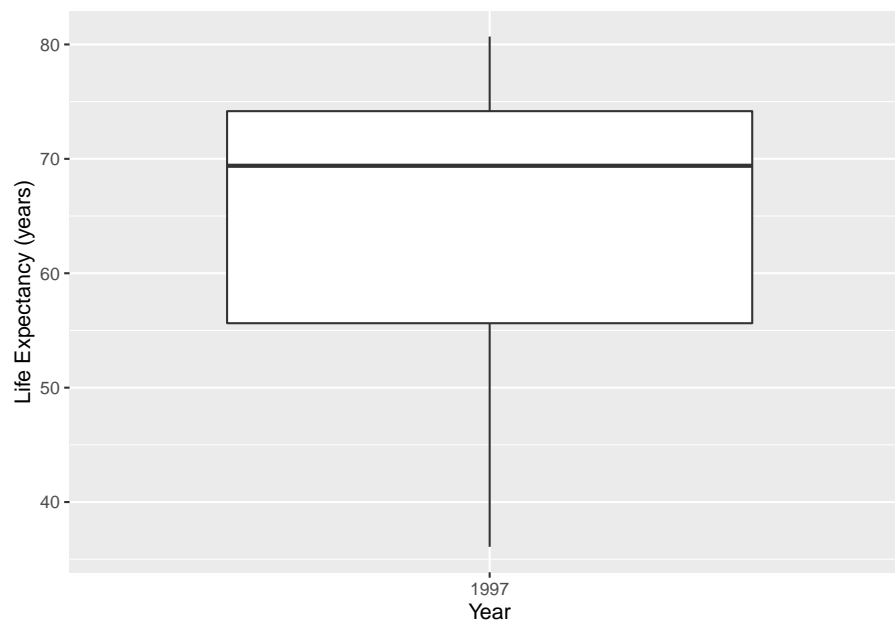
```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=lifeExp)) +
  geom_density(adjust=0.75) +
  xlab("Life Expectancy (years)") +
  ylab("Density")
```



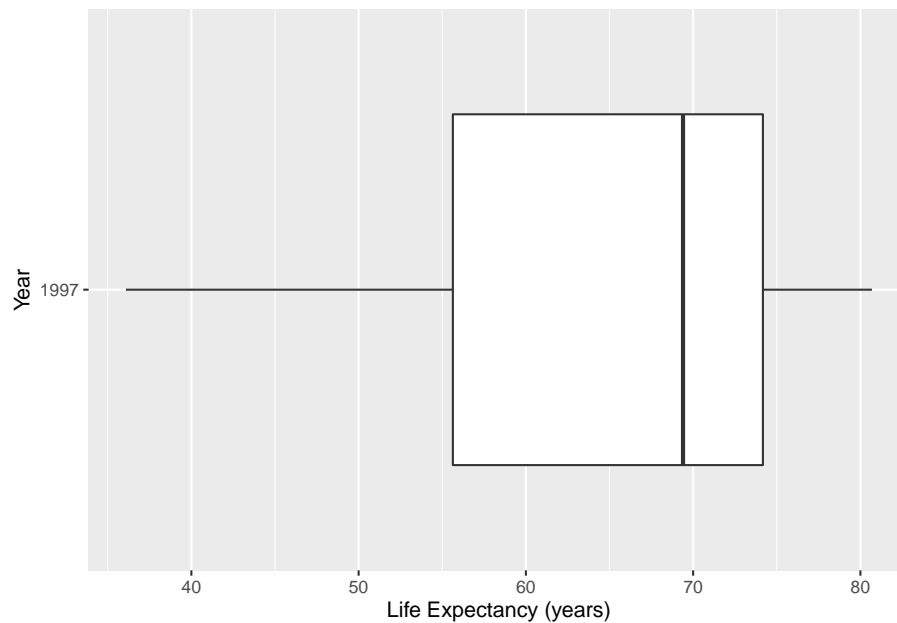
5.1.4 Boxplot

The boxplot display really needs only a single quantitative variable (here life expectancy) for the numeric axis. However, the other axis looks better with some sort of factor variable - so here we supply the year for the display, where the quantitative variable `year` has temporarily being used as a category/factor variable by being processed by the `factor` function before used in the graphic:

```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=factor(year),y=lifeExp)) +
  geom_boxplot() +
  labs(x="Year",y="Life Expectancy (years)")
```

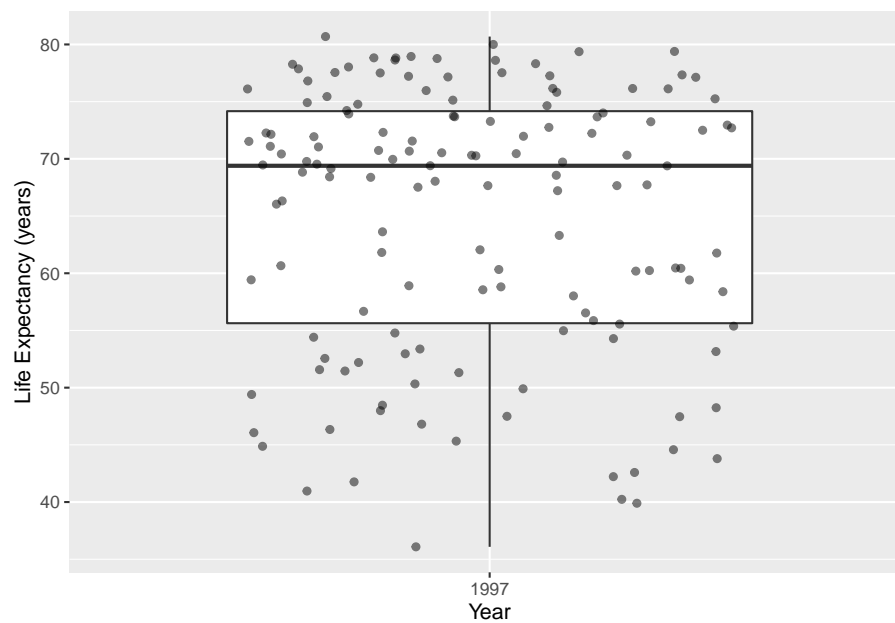



```
# Change orientation
ggplot(data=ds, mapping=aes(x=factor(year),y=lifeExp)) +
  geom_boxplot() +
  coord_flip() +
  labs(x="Year",y="Life Expectancy (years)")
```



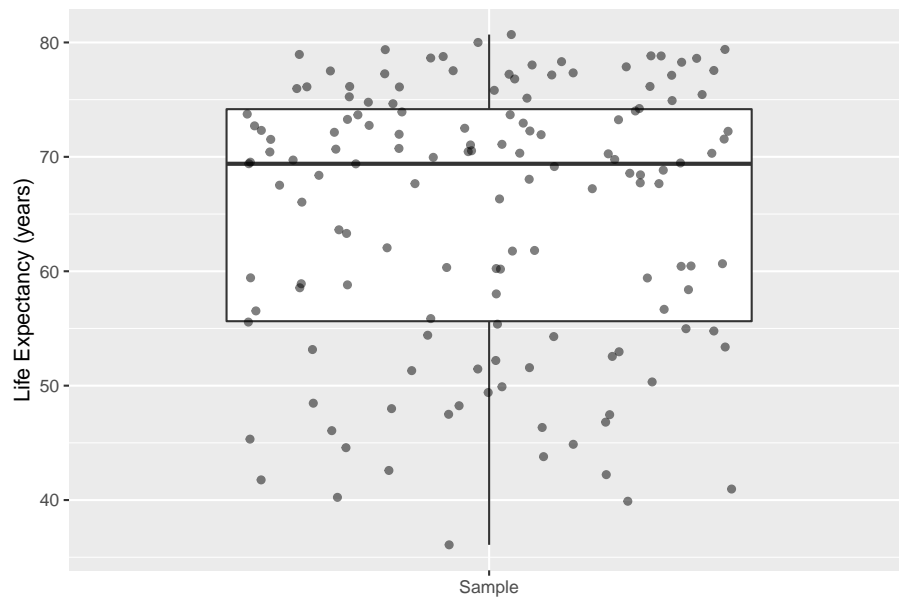
Now we overlay points on top of the boxplot display. Note the `geom_jitter` that overlays the points has an argument `alpha=0.5` signifying a slightly transparent plot symbol. An alpha value of 1 means the plot symbol is opaque, and a value of 0 is completely transparent. Careful use of alpha in large datasets will enable the analyst to correctly perceive point density. Without using a smaller value of `alpha` the plot may be one large blob of ink - making it difficult to judge the density of points in the display.

```
ds <- gapminder %>% filter(year==1997)
#
ggplot(data=ds, mapping=aes(x=factor(year),y=lifeExp)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(alpha=0.5, width=0.35) +
  labs(x="Year",y="Life Expectancy (years)")
```



If the dataframe has only one quantitative variable, we can make a character variable called “sample”, then this code will produce an acceptable display.

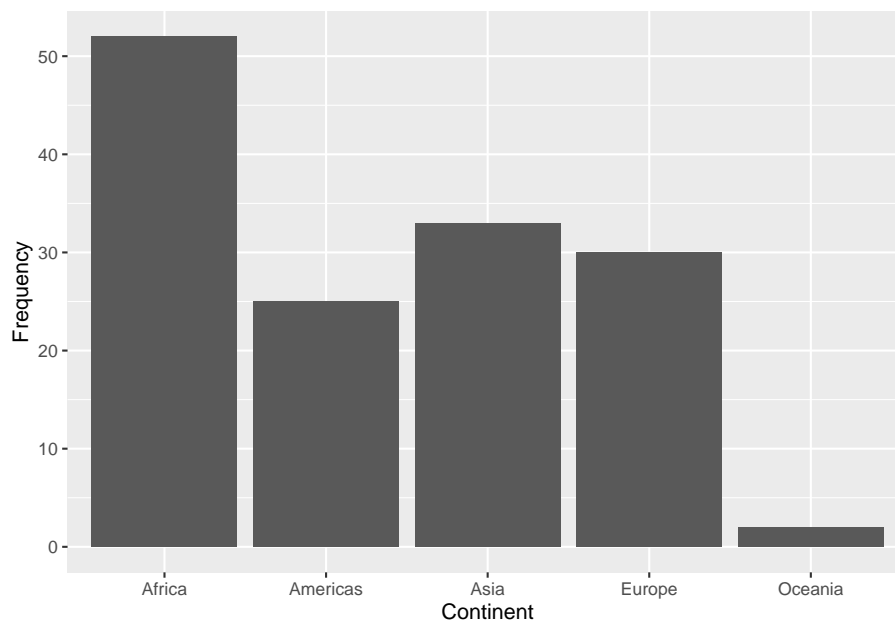
```
ds <- gapminder %>% filter(year==1997) %>%  
  mutate(sample="Sample")  
#  
ggplot(data=ds, mapping=aes(x=sample,y=lifeExp)) +  
  geom_boxplot(outlier.shape = NA) +  
  geom_jitter(alpha=0.5, width=0.35) +  
  labs(x="",y="Life Expectancy (years)")
```



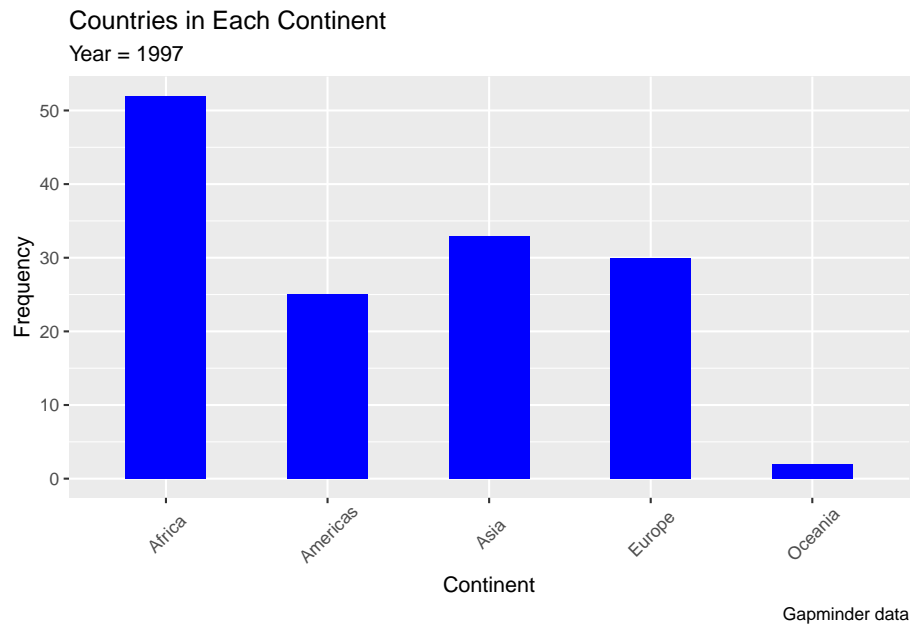
5.2 Displays of a Categorical Variable

5.2.1 Bar Graph

```
ds <- gapminder %>%
  filter(year==1997) %>%
  group_by(continent)
# Frequency of countries in each continent in 1997.
ggplot(data=ds, mapping=aes(x=continent)) +
  geom_bar() +
  labs(x="Continent", y="Frequency")
```

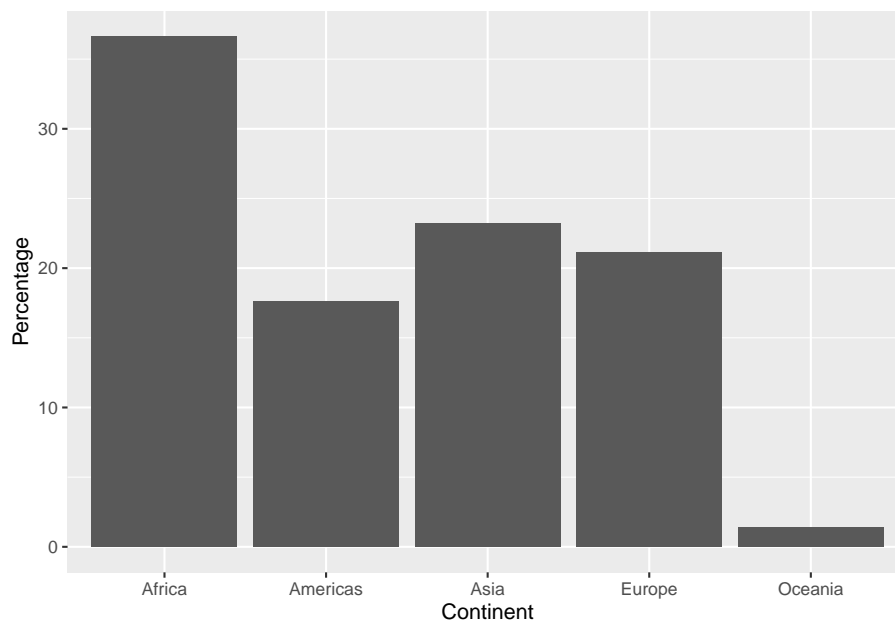


```
#  
ggplot(data=ds, mapping=aes(x=continent)) +  
  geom_bar(width=0.5, fill="blue") +  
  labs(title="Countries in Each Continent",  
        subtitle = "Year = 1997",  
        caption = "Gapminder data",  
        x="Continent",  
        y="Frequency") +  
  theme(axis.text.x = element_text(angle=45, vjust = 0.6))
```

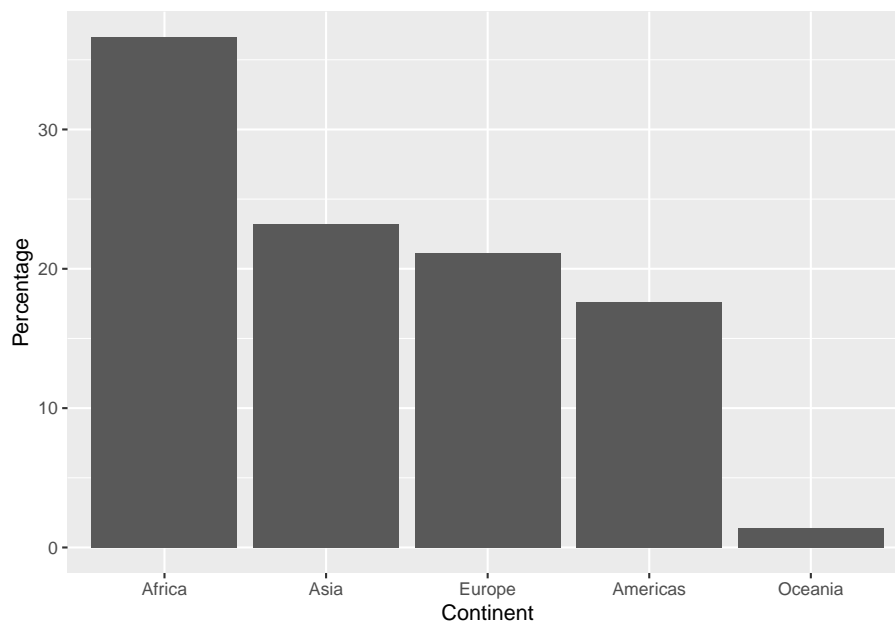


Bar graph with percentages on vertical axis.

```
ds <- gapminder %>%
  filter(year==1997) %>%
  group_by(continent) %>%
  summarise (n = n()) %>%
  mutate(pct = 100*n / sum(n))
#
ggplot(data=ds, mapping=aes(x = continent, y = pct)) +
  geom_bar(stat = "identity") +
  xlab("Continent") + ylab("Percentage")
```

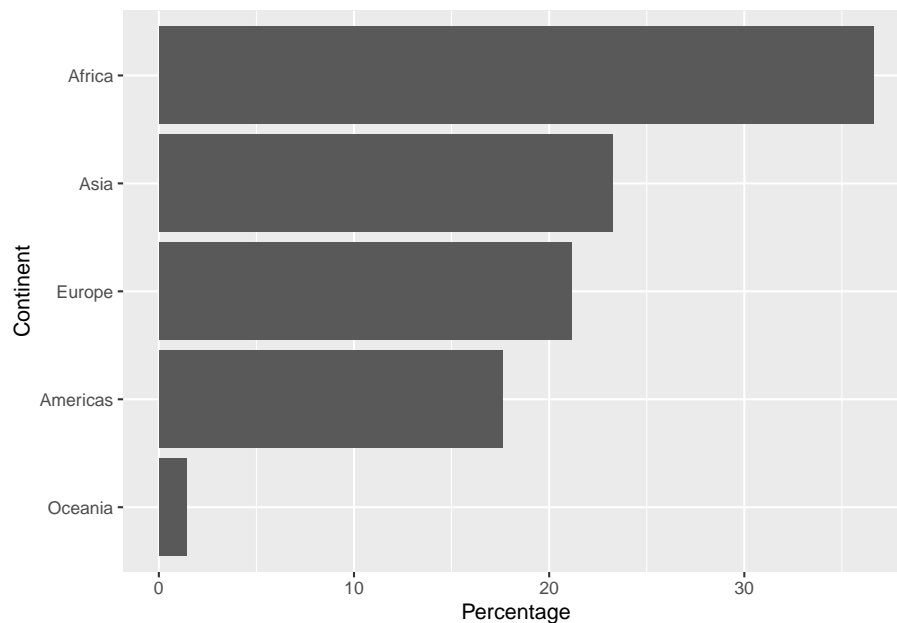


```
# change order of continents in decreasing frequency order  
ggplot(data=ds, mapping=aes(x = reorder(continent, -pct), y = pct)) +  
  geom_bar(stat = "identity") +  
  xlab("Continent") + ylab("Percentage")
```



Sometimes it is more convenient to have the bars oriented horizontally. Notice we set up the aesthetic mappings as usual and then flip the axes with the `coord_flip` command.

```
ds <- gapminder %>%
  filter(year==1997) %>%
  group_by(continent) %>%
  summarise (n = n()) %>%
  mutate(pct = 100*n / sum(n))
#
ggplot(data=ds, mapping=aes(x = reorder(continent, pct), y = pct)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  xlab("Continent") + ylab("Percentage")
```



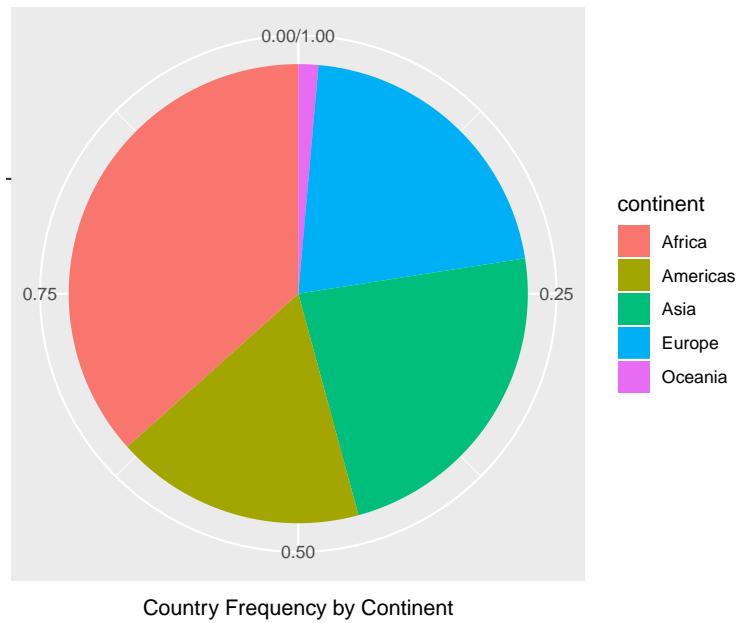
5.2.2 Pie Graph

Pie graphs are not recommended, but the code needed to make one is given here.

```
contin.prop<- gapminder %>%
  group_by(continent) %>%
  summarise (n = n()) %>%
  mutate(freq = n / sum(n))
#
```



```
ggplot(data=contin.prop, mapping=aes(x="",y=freq,fill=continent)) +  
  geom_bar(width=1,stat="identity") +  
  coord_polar("y",start=0) +  
  xlab("") + ylab("Country Frequency by Continent")
```



Chapter 6

Summary Statistics For One Variable

6.1 One Quantitative Variable

6.1.1 Using base R summary function

```
gapminder %>% filter(year==1997) %>% select(lifeExp) %>% summary()

##      lifeExp
##  Min.   :36.09
## 1st Qu.:55.63
##  Median :69.39
##   Mean  :65.01
## 3rd Qu.:74.17
##   Max.  :80.69
```

6.1.2 Using dplyr summarise function

It is often helpful to create data summaries during preliminary phases of examination. Here is how to use the summarise command in the analysis pipeline system.

```
gapminder %>% filter(year==1997) %>%
  dplyr::summarise(meanLE=mean(lifeExp,na.rm=TRUE),
                   medLE=median(lifeExp,na.rm=TRUE),
                   sd=sd(lifeExp,na.rm=TRUE),
                   iqr=IQR(lifeExp,na.rm=TRUE),
                   Q1=quantile(lifeExp,probs=0.25,na.rm=TRUE),
```

```
Q3=quantile(lifeExp,probs=0.75),
n=n())
```

```
## # A tibble: 1 x 7
##   meanLE medLE    sd   iqr    Q1    Q3    n
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
## 1   65.0   69.4  11.6  18.5  55.6  74.2  142
```

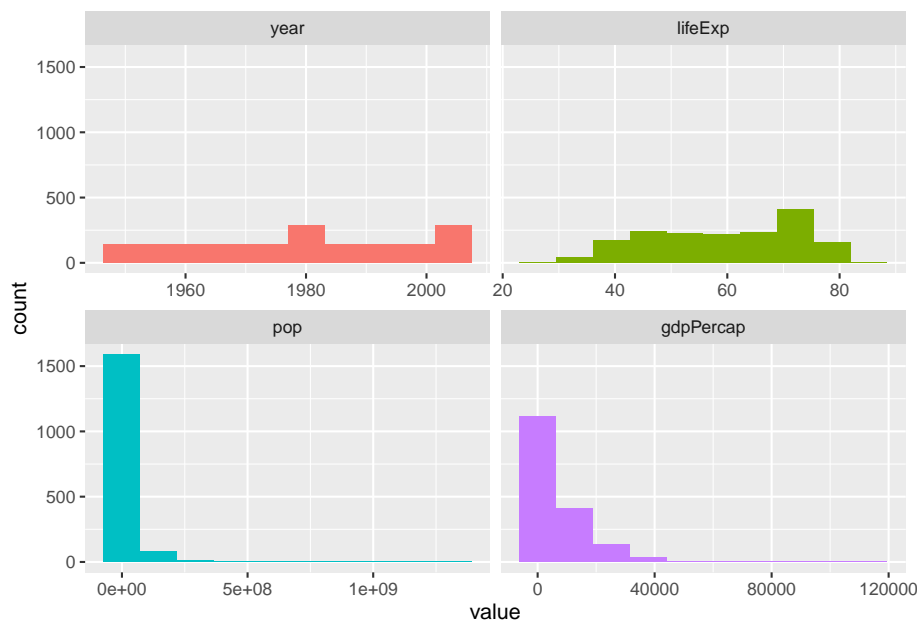
6.1.3 Summary Statistics Using funModeling package

The `profiling_num` and `plot_num` functions from the *funModeling* package help give a concise numeric and visual overview of the numeric variables in the dataframe.

```
funModeling::profiling_num(gapminder)
```

```
##   variable      mean    std_dev variation_coef      p_01      p_05
## 1   year 1.979500e+03 1.726533e+01 0.008722066 1952.0000 1952.0000
## 2  lifeExp 5.947444e+01 1.291711e+01 0.217187544 33.4926 38.4924
## 3    pop 2.960121e+07 1.061579e+08 3.586268548 154117.9200 475458.9000
## 4 gdpPercap 7.215327e+03 9.857455e+03 1.366182632 369.2201 547.9964
##      p_25      p_50      p_75      p_95      p_99 skewness
## 1  1965.750  1979.5000 1.993250e+03 2007.000 2.007000e+03 0.0000000
## 2   48.198   60.7125 7.084550e+01 77.437 8.023892e+01 -0.2524798
## 3 2793664.000 7023595.5000 1.958522e+07 89822054.500 6.319900e+08 8.3328742
## 4  1202.060  3531.8470 9.325462e+03 26608.333 3.678357e+04 3.8468819
##      kurtosis      iqr      range_98
## 1  1.783217 2.750000e+01 [1952, 2007]
## 2  1.873099 2.264750e+01 [33.4926, 80.23892]
## 3 80.716151 1.679156e+07 [154117.92, 631990000.000002]
## 4 30.431702 8.123402e+03 [369.220127794, 36783.5723707]
##      range_80
## 1 [1957, 2002]
## 2 [41.5108, 75.097]
## 3 [946367.1, 54801369.5]
## 4 [687.71836128, 19449.138209]
```

```
funModeling::plot_num(gapminder)
```

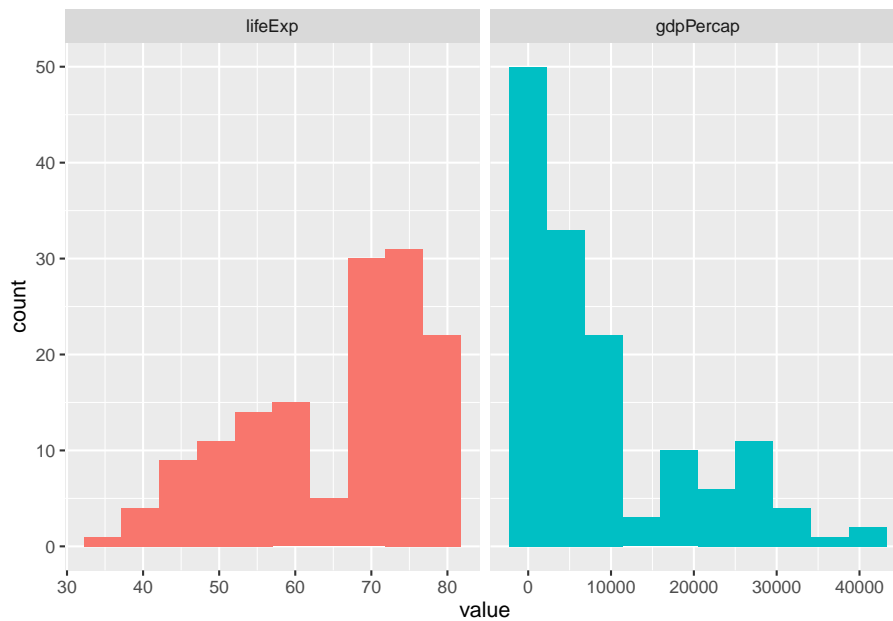


This example shows summary statistics for two quantitative variables. For only one variable, simply use `select` for only one variable.

```
gapminder %>%
  filter(year==1997) %>%
  select(lifeExp,gdpPercap) %>%
  funModeling::profiling_num()
```

```
##   variable      mean    std_dev variation_coef    p_01    p_05
## 1  lifeExp  65.01468  11.55944    0.1777974  40.03681  43.83415
## 2  gdpPercap 9090.17536 10171.49326    1.1189546  434.72721  590.90598
##      p_25    p_50    p_75    p_95    p_99 skewness kurtosis
## 1  55.63375  69.394  74.16975  78.7635  79.7499 -0.6427906  2.218599
## 2 1366.83796 4781.825 12022.86719 29088.8709 38442.0133  1.2979366  3.604446
##      iqr      range_98      range_80
## 1   18.536    [40.03681, 79.7499]    [47.4671, 77.548]
## 2 10656.029 [434.727210598, 38442.0133187] [789.29339925, 26905.596049]
```

```
#
gapminder %>%
  filter(year==1997) %>%
  select(lifeExp,gdpPercap) %>%
  funModeling::plot_num()
```



6.1.4 Summary Statistics: `skimr` package

The `skimr` package produces summary statistics about variables and overviews for dataframes. It is easy to manipulate and use pipes, `select`, and `filter` from the tidyverse family of packages.

The next code supplies a dataframe that contains both categorical variables (`continent`), and numeric variables (`lifeExp`, `gdpPercap`). Numeric variables are chosen with the `yank` function, then some attributes are omitted from the display (`n_missing`, `complete_rate`) using the `select` function from `dplyr`.

```
varlist <- c("n_missing", "complete_rate")
gapminder %>% filter(year==1997) %>%
  select(-year, -country, -pop) %>%
  skimr::skim_without_charts() %>%
  skimr::yank("numeric") %>%
  dplyr::select(-one_of(varlist))
```

Variable type: numeric

| skim_variable | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---------------|---------|----------|--------|---------|---------|----------|----------|
| lifeExp | 65.01 | 11.56 | 36.09 | 55.63 | 69.39 | 74.17 | 80.69 |
| gdpPercap | 9090.18 | 10171.49 | 312.19 | 1366.84 | 4781.83 | 12022.87 | 41283.16 |

6.2 One Categorical Variable

6.2.1 Counting Values

The next command counts the number of rows in the dataset for each continent - then we show a variant which pipes the output into the kable function for a more attractive table.

```
gapminder %>% count(continent)
```

```
## # A tibble: 5 x 2
##   continent     n
##   <fct>       <int>
## 1 Africa      624
## 2 Americas    300
## 3 Asia        396
## 4 Europe      360
## 5 Oceania     24
```

```
#
gapminder %>% count(continent) %>% knitr::kable()
```

| continent | n |
|-----------|-----|
| Africa | 624 |
| Americas | 300 |
| Asia | 396 |
| Europe | 360 |
| Oceania | 24 |

```
#
gapminder %>% count(continent, sort=TRUE) %>% knitr::kable()
```

| continent | n |
|-----------|-----|
| Africa | 624 |
| Asia | 396 |
| Europe | 360 |
| Americas | 300 |
| Oceania | 24 |

The previous code tells us how many lines (rows) for each continent, but many rows are repeated for each country - just different years.

```
gapminder %>% filter(year==1997 | year==1967) %>%
  dplyr::group_by(continent) %>%
  dplyr::summarise(n = n(), n_countries = n_distinct(country)) %>% knitr::kable()
```

| continent | n | n_countries |
|-----------|-----|-------------|
| Africa | 104 | 52 |
| Americas | 50 | 25 |
| Asia | 66 | 33 |
| Europe | 60 | 30 |
| Oceania | 4 | 2 |

6.2.2 Categorical variable: skimr package

Here we summarize a categorical variable (continent), and observe it has 5 unique values (levels) and the most frequent values are displayed.

```
gapminder %>% filter(year==1997) %>%
  select(lifeExp,continent) %>%
  skimr::skim_without_charts() %>%
  skimr::yank("factor") %>%
  dplyr::select(-n_missing,-ordered,-complete_rate)
```

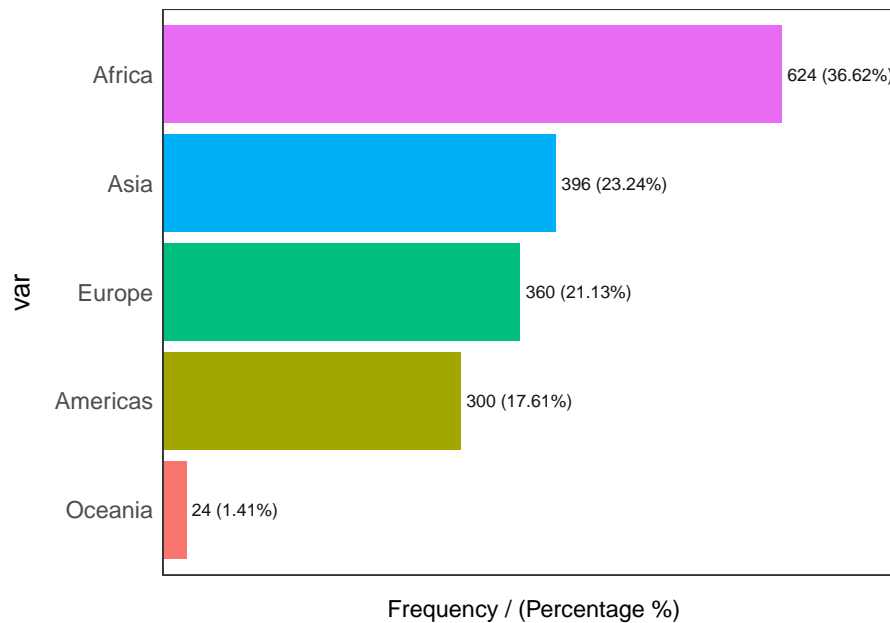
Variable type: factor

| skim_variable | n_unique | top_counts |
|---------------|----------|------------------------------------|
| continent | 5 | Afr: 52, Asi: 33, Eur: 30, Ame: 25 |

6.2.3 Categorical variable: funModeling package

The *funModeling* package gives an easy way to learn about categorical variables of types: character and factor. There are two categorical variables in the gapminder dataframe: country and continent. There are a lot of countries, so we demonstrate this command for only the continent variable.

```
# Frequency distribution of entire dataframe
# will produce lots of output and warnings
#funModeling::freq(gapminder)
# next command for one category variable: continent
funModeling::freq(gapminder$continent)
```

```
##      var frequency percentage cumulative_perc
## 1  Africa      624      36.62           36.62
## 2   Asia      396      23.24           59.86
## 3  Europe      360      21.13           80.99
## 4 Americas      300      17.61           98.60
## 5 Oceania       24       1.41          100.00
```

There are a lot of observations (rows) for Africa and very few for Oceania (Australia, New Zealand, etc).

6.2.4 Categorical variable: janitor package

Let's begin with the base R function `table`:

```
gapminder %>%
  filter(year==1997) %>%
  select(continent) %>%
  table()
```

```
## .
##   Africa Americas      Asia  Europe  Oceania
##      52      25      33     30      2
```

Now contrast with the `tabyl` function from the *janitor* package:

```
gapminder %>%
  filter(year==1997) %>%
```

```
janitor::tabyl(continent,sort=TRUE) %>%
knitr::kable()
```

| continent | n | percent |
|-----------|----|-----------|
| Africa | 52 | 0.3661972 |
| Americas | 25 | 0.1760563 |
| Asia | 33 | 0.2323944 |
| Europe | 30 | 0.2112676 |
| Oceania | 2 | 0.0140845 |

```
#
gapminder %>%
  filter(year==1997) %>%
  janitor::tabyl(continent,sort=TRUE) %>%
  janitor::adorn_pct_formatting(digits=2,affix_sign = TRUE) %>%
  knitr::kable()
```

| continent | n | percent |
|-----------|----|---------|
| Africa | 52 | 36.62% |
| Americas | 25 | 17.61% |
| Asia | 33 | 23.24% |
| Europe | 30 | 21.13% |
| Oceania | 2 | 1.41% |

Chapter 7

Exploratory Data Analysis For One Quantitative Variable: by Groups

It is often helpful to create data summaries of a quantitative variable for each level of a grouping variable.

7.1 Summary Statistics: dplyr

Using *dplyr* and *tidyverse* for summary statistics across the levels of a group variable (of type factor/categorical) requires the use of the verb `group_by`. Here we produce summary statistics of life expectancy across the levels of continent.

```
# Output presented in initial continent order (alphabetic)
gapminder %>% filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent) %>%
  summarise(meanLE=mean(lifeExp,na.rm=TRUE),
            medLE=median(lifeExp,na.rm=TRUE),
            sd=sd(lifeExp,na.rm=TRUE),
            iqr=IQR(lifeExp,na.rm=TRUE),
            Q1=quantile(lifeExp, probs=0.25,na.rm=TRUE),
            Q3=quantile(lifeExp,probs=0.75),
            n=n())
```

```
## # A tibble: 4 x 8
##   continent meanLE medLE    sd   iqr    Q1    Q3    n
##   <fct>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
```

```
## 1 Africa      53.6  52.8  9.10 11.9   47.3  59.2   52
## 2 Americas   71.2  72.1  4.89 4.83   69.4  74.2   25
## 3 Asia       68.0  70.3  8.09 10.7   61.8  72.5   33
## 4 Europe     75.5  76.1  3.10 4.97   73.0  78.0   30
```

```
#
# Output rows ordered by decreasing values of a statistic (mean Life Expectancy):
gapminder %>% filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent) %>%
  summarise(meanLE=mean(lifeExp,na.rm=TRUE),
            medLE=median(lifeExp,na.rm=TRUE),
            sd=sd(lifeExp,na.rm=TRUE),
            iqr=IQR(lifeExp,na.rm=TRUE),
            min=min(lifeExp),
            max=max(lifeExp),
            n=n()) %>%
  arrange(desc(meanLE))
```

```
## # A tibble: 4 x 8
##   continent meanLE medLE   sd   iqr  min  max    n
##   <fct>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
## 1 Europe     75.5  76.1  3.10  4.97  68.8  79.4   30
## 2 Americas   71.2  72.1  4.89  4.83  56.7  78.6   25
## 3 Asia       68.0  70.3  8.09 10.7   41.8  80.7   33
## 4 Africa     53.6  52.8  9.10 11.9   36.1  74.8   52
```

Next, we save the statistics table to an object called `statstable`, then we use the `kable` function for display.

```
statstable <- gapminder %>% filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent) %>%
  summarise(meanLE=mean(lifeExp,na.rm=TRUE),
            medLE=median(lifeExp,na.rm=TRUE),
            sd=sd(lifeExp,na.rm=TRUE),
            iqr=IQR(lifeExp,na.rm=TRUE),
            min=min(lifeExp),
            max=max(lifeExp),
            n=n()) %>%
  arrange(desc(meanLE))
#
knitr::kable(statstable)
```

| continent | meanLE | medLE | sd | iqr | min | max | n |
|-----------|----------|--------|----------|----------|--------|--------|----|
| Europe | 75.50517 | 76.116 | 3.104677 | 4.96625 | 68.835 | 79.390 | 30 |
| Americas | 71.15048 | 72.146 | 4.887584 | 4.83500 | 56.671 | 78.610 | 25 |
| Asia | 68.02052 | 70.265 | 8.091171 | 10.68100 | 41.763 | 80.690 | 33 |
| Africa | 53.59827 | 52.759 | 9.103387 | 11.92825 | 36.087 | 74.772 | 52 |

7.2 Summary Statistics: skimr

Here we implement the `group_by` function to display descriptive statistics for numeric variables by continent, for two quantitative variables using functions from the *skimr* package.

```
gapminder %>% filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent) %>%
  skimr::skim_without_charts() %>%
  skimr::yank("numeric") %>%
  dplyr::filter(skim_variable %in% c("lifeExp", "gdpPercap")) %>%
  knitr::kable()
```

| skim_variable | continent | n_missing | complete_rate | mean | sd | p0 | p25 |
|---------------|-----------|-----------|---------------|-------------|--------------|-----------|------------|
| lifeExp | Africa | 0 | 1 | 53.59827 | 9.103387 | 36.0870 | 47.30025 |
| lifeExp | Americas | 0 | 1 | 71.15048 | 4.887584 | 56.6710 | 69.38800 |
| lifeExp | Asia | 0 | 1 | 68.02052 | 8.091171 | 41.7630 | 61.81800 |
| lifeExp | Europe | 0 | 1 | 75.50517 | 3.104677 | 68.8350 | 73.02350 |
| gdpPercap | Africa | 0 | 1 | 2378.75956 | 2820.728117 | 312.1884 | 791.90197 |
| gdpPercap | Americas | 0 | 1 | 8889.30086 | 7874.225145 | 1341.7269 | 4684.31381 |
| gdpPercap | Asia | 0 | 1 | 9834.09330 | 11094.180481 | 415.0000 | 1902.25210 |
| gdpPercap | Europe | 0 | 1 | 19076.78180 | 10065.457716 | 3193.0546 | 9946.59931 |

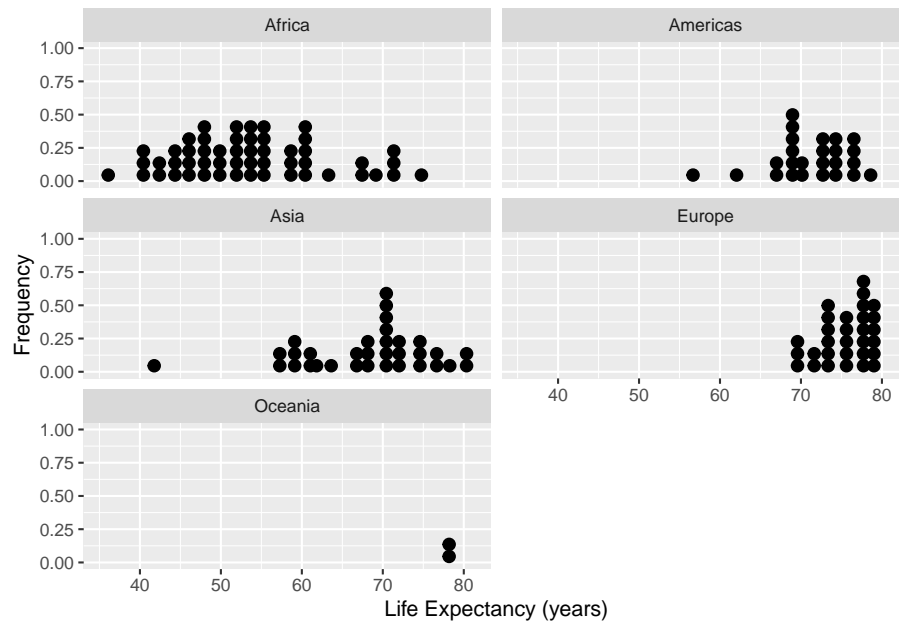
7.3 Graphical Displays of a quantitative variable, separated by groups

In each example, the first lines create the dataset to be graphed - followed by a `ggplot` command making the display. Several of the examples make use of the principle of “small-multiples” so that each level of the factor variable has a separate panel for the quantitative variable display.

7.3.1 Dotplots

```
ds <- gapminder %>% filter(year==1997)
#
```

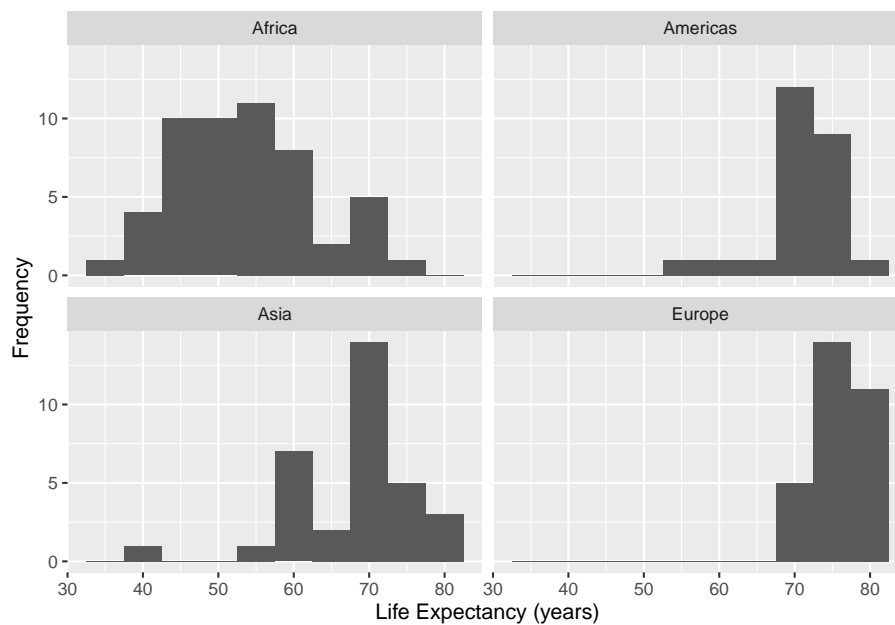
```
ggplot(data=ds,mapping=aes(x=lifeExp)) +
  geom_dotplot() +
  facet_wrap( ~ continent,ncol=2) +
  xlab("Life Expectancy (years)") +
  ylab("Frequency")
```



7.3.2 Histograms

```
ds <- gapminder %>%
  filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent)
#
ggplot(data=ds, mapping=aes(x=lifeExp)) +
  geom_histogram(binwidth=5) +
  facet_wrap( ~ continent,ncol=2) +
  xlab("Life Expectancy (years)") +
  ylab("Frequency")
```

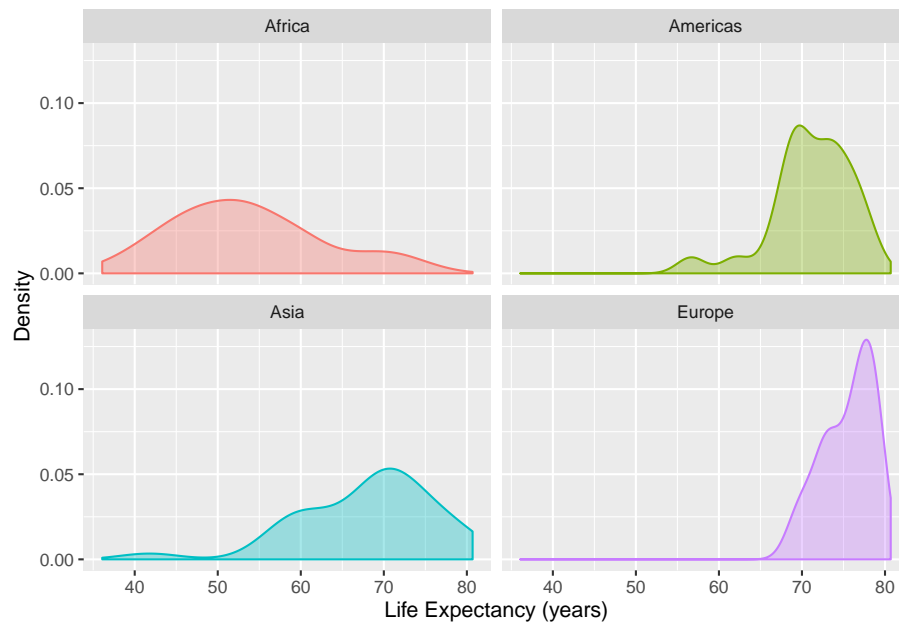
7.3. GRAPHICAL DISPLAYS OF A QUANTITATIVE VARIABLE, SEPARATED BY GROUPS⁶³



7.3.3 Density Plots in Facets

The code given here shows how to produce a density plot in separate panels for each continent.

```
ds <- gapminder %>%  
  filter(year==1997) %>%  
  filter(continent != "Oceania") %>%  
  group_by(continent)  
#  
ggplot(data=ds, mapping=aes(x=lifeExp, colour=continent, fill=continent)) +  
  geom_density(alpha = 0.35) +  
  xlab("Life Expectancy (years)") +  
  ylab("Density") +  
  facet_wrap( ~ continent, ncol = 2) +  
  theme(legend.position = "none")
```

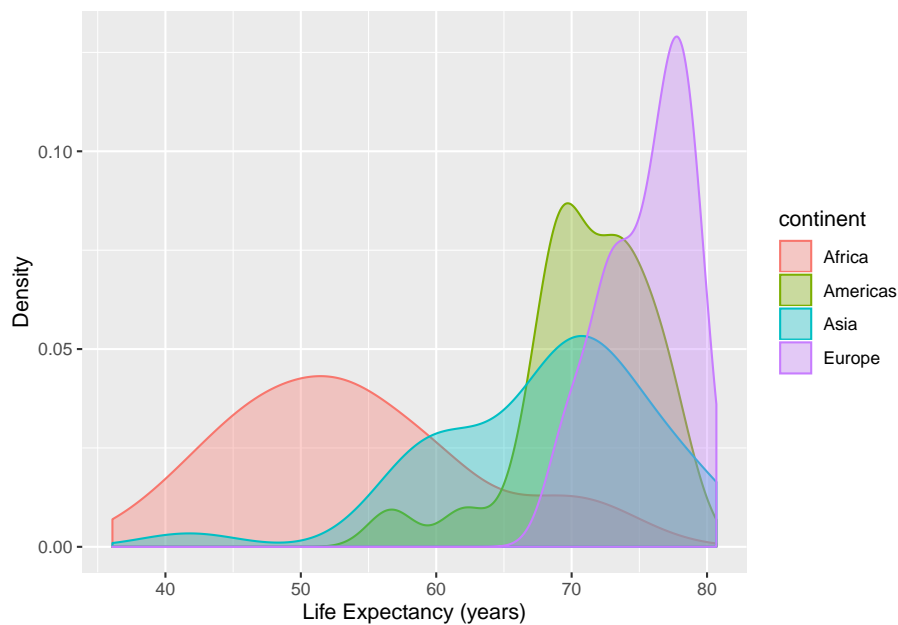


7.3.4 Overlaid Density Plots

The initial command below takes the `gapminder` data and consider only observations (rows) from 1997, but exclude all observations from Oceania. The `alpha` setting controls the amount of transparency in the densities for each continent - smaller values of `alpha` (between 0 and 1) are more transparent.

```
gapminder %>%
  filter(year==1997) %>%
  filter(continent != "Oceania") %>%
  group_by(continent) %>%
  ggplot(mapping=aes(x=lifeExp, colour=continent, fill=continent)) +
  geom_density(alpha = 0.35) +
  xlab("Life Expectancy (years)") +
  ylab("Density")
```

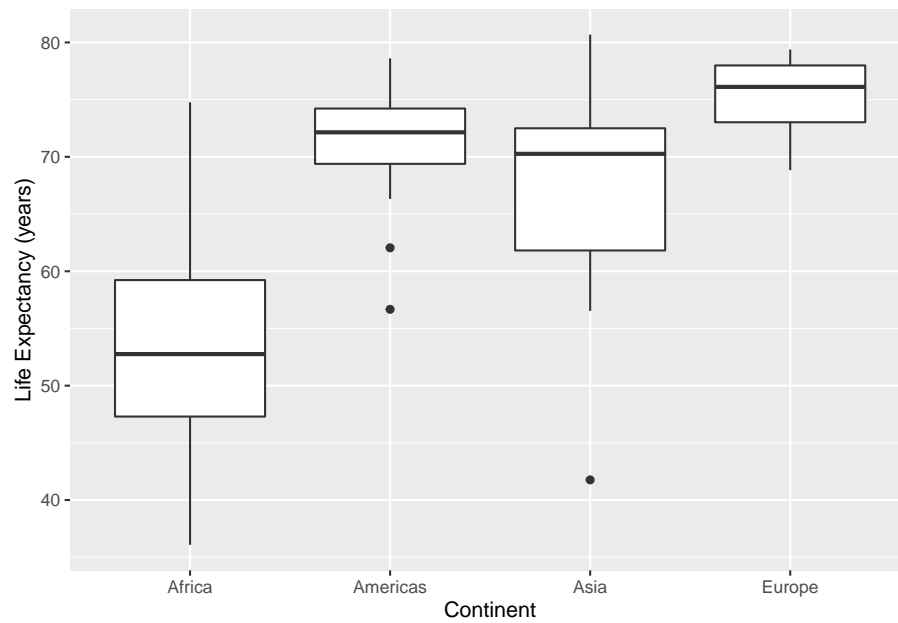

7.3. GRAPHICAL DISPLAYS OF A QUANTITATIVE VARIABLE, SEPARATED BY GROUPS65



7.3.5 Boxplots, Grouped Data

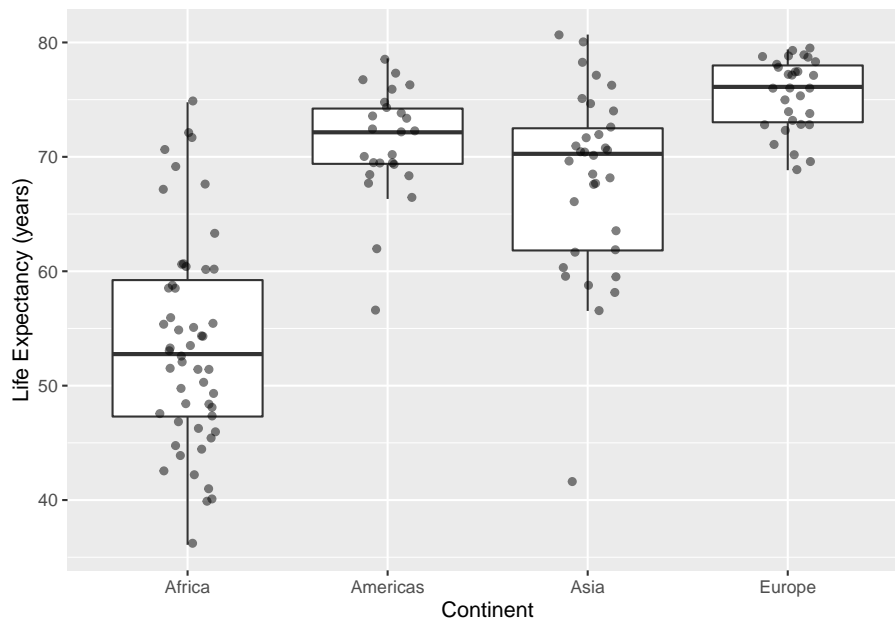
In the code below, the `alpha` value again controls the transparency of the points `alpha=1` means opaque, `alpha=0` means completely see-through. When there is a lot of data, use a smaller value of `alpha`.

```
ds <- gapminder %>%  
  filter(year==1997) %>%  
  filter(continent != "Oceania") %>%  
  group_by(continent)  
#  
ggplot(data=ds, mapping=aes(x=continent,y=lifeExp)) +  
  geom_boxplot() +  
  labs(x="Continent",y="Life Expectancy (years)")
```



```
#
ggplot(data=ds, mapping=aes(x=continent,y=lifeExp)) +
  geom_boxplot(outlier.colour = NA) +
  geom_point(position = position_jitter(width = 0.15, height = 0.15),alpha=.50) +
  labs(x="Continent",y="Life Expectancy (years)")
```

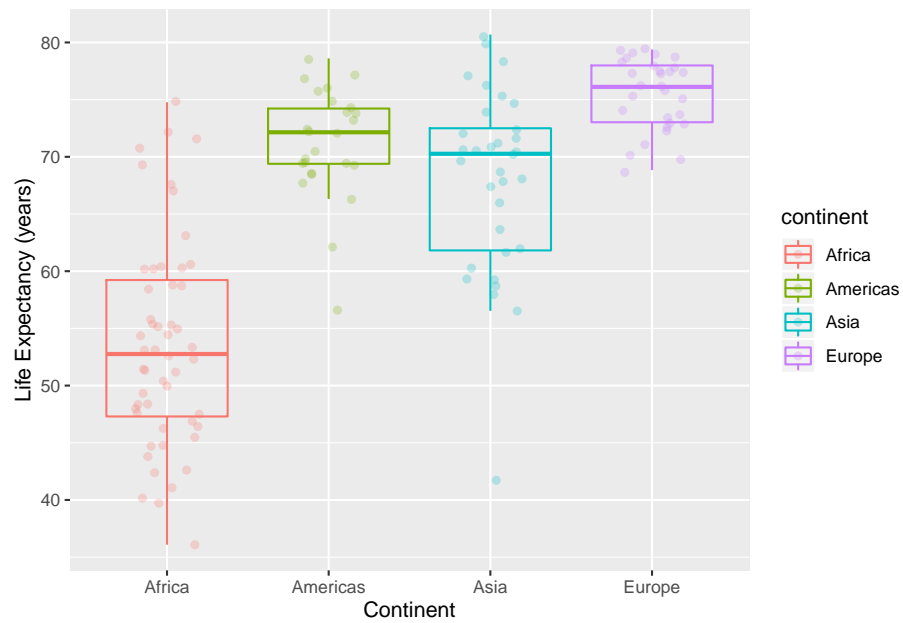
7.3. GRAPHICAL DISPLAYS OF A QUANTITATIVE VARIABLE, SEPARATED BY GROUPS67



7.3.6 Boxplots, overlay points on the boxplots with color control

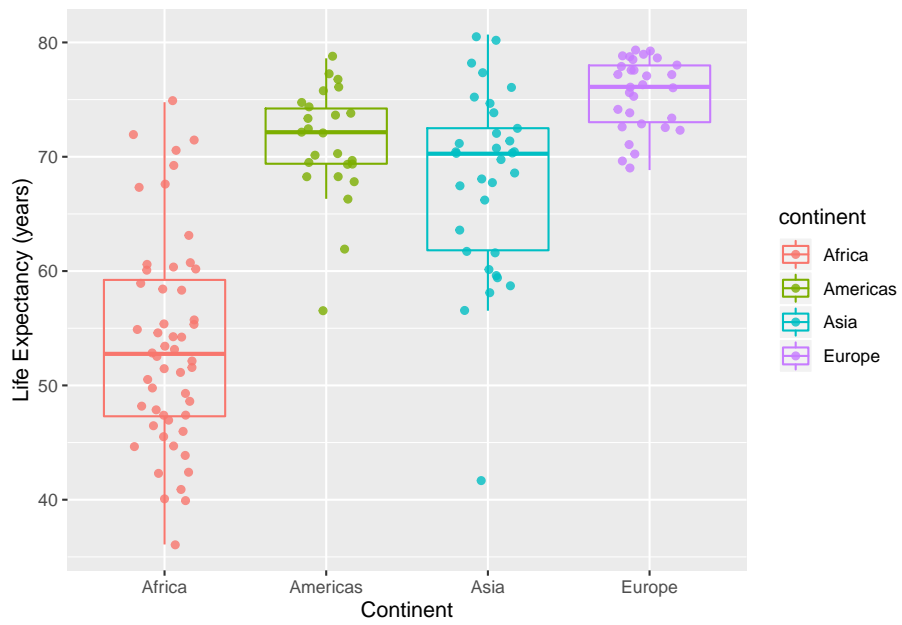
In the code below, the alpha value controls the transparency of the points alpha=1 means opaque, alpha=0 means completely see-through.

```
ds <- gapminder %>%  
  filter(year==1997) %>%  
  filter(continent != "Oceania") %>%  
  group_by(continent)  
#  
ggplot(data=ds, mapping=aes(x=continent,y=lifeExp, colour=continent)) +  
  geom_point(position = position_jitter(width = 0.2, height = 0.2),alpha=.25) +  
  geom_boxplot(outlier.colour = NA, fill = NA) +  
  labs(x="Continent",y="Life Expectancy (years)")
```



```
#
ggplot(data=ds, mapping=aes(x=continent,y=lifeExp, colour=continent)) +
  geom_point(position = position_jitter(width = 0.2, height = 0.2),alpha=.80) +
  geom_boxplot(outlier.colour = NA, fill = NA) +
  labs(x="Continent",y="Life Expectancy (years)")
```

7.3. GRAPHICAL DISPLAYS OF A QUANTITATIVE VARIABLE, SEPARATED BY GROUPS69



Chapter 8

Analysis of One Categorical Variable by another categorical variable

To demonstrate graphical displays of two categorical variables, we need a new dataset with two categorical variables. We use the `congress_age` dataframe from the *fivethirtyeight* package. In these displays we will use categorical variables:

- party affiliation (party) with values: D, I, R.
- congressional chamber (chamber) with values: house, senate

We will restrict ourselves to the 113th congress, a meeting of the legislative branch of the United States federal government, from January 3, 2013, to January 3, 2015, during the fifth and sixth years of Barack Obama's presidency.

8.1 Tables

```
congage <- fivethirtyeight::congress_age
ds1 <- congage %>% filter(congress > 112) %>% select(congress, chamber, state, party, incumbent, age)
# We declare party and chamber as factor/categorical variables, and control their levels.
ds1 <- ds1 %>% mutate(party = factor(party, levels=c("D", "I", "R")),
                     chamber = factor(chamber))
ds1 <- ds1 %>% na.omit()
ds <- ds1
#
table(ds$chamber, ds$party)
```

```
##
##           D    I    R
##   house  202    0  237
##   senate   57    2   46

#
mytable <- table(ds$chamber, ds$party)
#
prop.table(mytable) # cell percentages

##
##           D           I           R
##   house  0.371323529 0.000000000 0.435661765
##   senate 0.104779412 0.003676471 0.084558824

prop.table(mytable, 1) # row percentages

##
##           D           I           R
##   house  0.46013667 0.00000000 0.53986333
##   senate 0.54285714 0.01904762 0.43809524

prop.table(mytable, 2) # column percentages

##
##           D           I           R
##   house  0.7799228 0.0000000 0.8374558
##   senate 0.2200772 1.0000000 0.1625442

ds %>% janitor::tabyl(chamber, party)

## chamber    D I    R
##   house  202 0  237
##   senate   57 2   46

#
t2 <- ds %>% janitor::tabyl(chamber, party)
t2 %>%
  janitor::adorn_percentages("row") %>%
  janitor::adorn_pct_formatting(digits = 2) %>%
  janitor::adorn_ns()

## chamber           D           I           R
##   house 46.01% (202) 0.00% (0) 53.99% (237)
##   senate 54.29% (57) 1.90% (2) 43.81% (46)

# column percentages
t2 %>%
  janitor::adorn_percentages("col") %>%
  janitor::adorn_pct_formatting(digits = 2) %>%
```



```
janitor::adorn_ns()
```

```
## chamber          D          I          R
##   house 77.99% (202)  0.00% (0) 83.75% (237)
##   senate 22.01%  (57) 100.00% (2) 16.25%  (46)
```

```
# both row and column percentages
```

```
t2 %>%
  janitor::adorn_percentages("all") %>%
  janitor::adorn_pct_formatting(digits = 2) %>%
  janitor::adorn_ns()
```

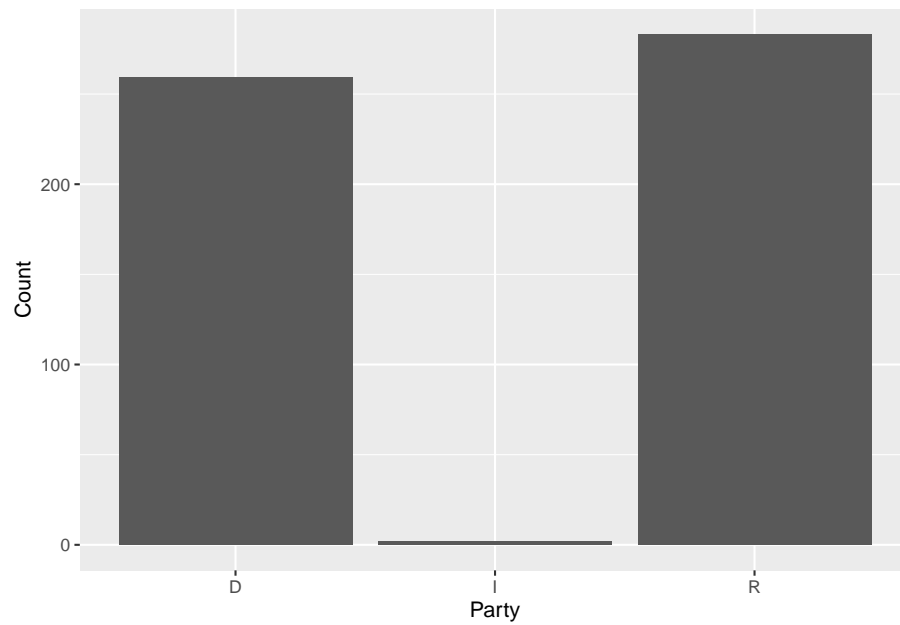
```
## chamber          D          I          R
##   house 37.13% (202) 0.00% (0) 43.57% (237)
##   senate 10.48%  (57) 0.37% (2)  8.46%  (46)
```

```
congage <- fivethirtyeight::congress_age
ds1 <- congage %>% filter(congress > 112) %>% select(congress, chamber, state, party, incumbent, age)
# We declare party and chamber as factor/categorical variables, and control their levels.
ds1 <- ds1 %>% mutate(party = factor(party, levels=c("D", "I", "R")),
                     chamber = factor(chamber))
ds1 <- ds1 %>% na.omit()
ds <- ds1
ds %>% group_by(chamber, party) %>%
  dplyr::count() %>%
  tidyr::pivot_wider(names_from = party, values_from = n)
```

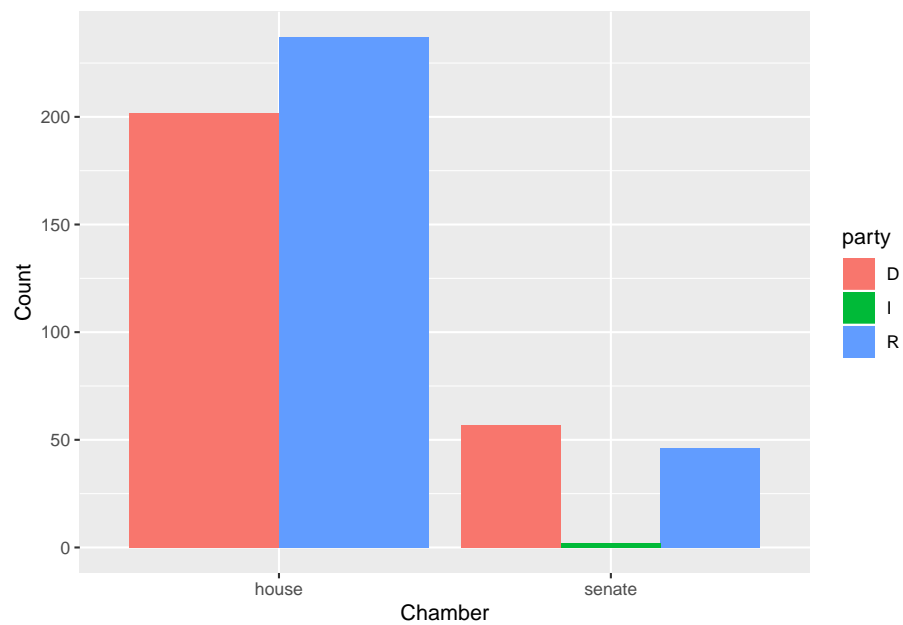
```
## # A tibble: 2 x 4
## # Groups:   chamber [2]
##   chamber    D    R    I
##   <fct>   <int> <int> <int>
## 1 house     202   237   NA
## 2 senate     57    46    2
```

8.2 Graphical Displays

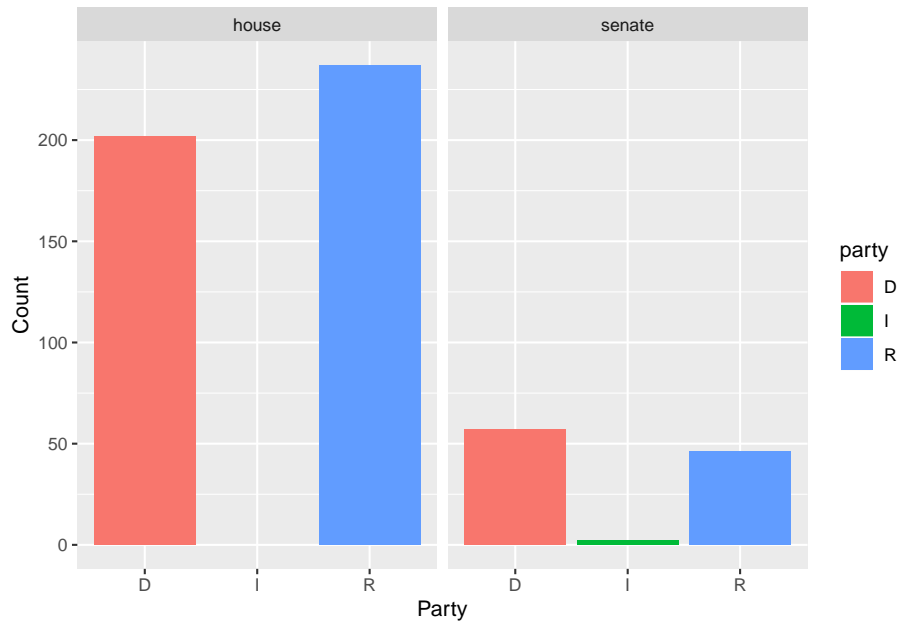
```
# basic bar plot of party affiliation
ggplot(data=ds, aes(x=party)) + geom_bar() +
  labs(x="Party", y="Count")
```



```
ds <- ds1 %>% group_by(chamber)
#
ggplot(data=ds, aes(x=chamber)) + geom_bar(aes(fill=party), position="dodge") +
  labs(x="Chamber", y="Count")
```



```
#
ggplot(data=ds, aes(x=party)) +
  geom_bar(aes(fill=party)) +
  facet_wrap(~ chamber) +
  labs(x="Party", y="Count")
```

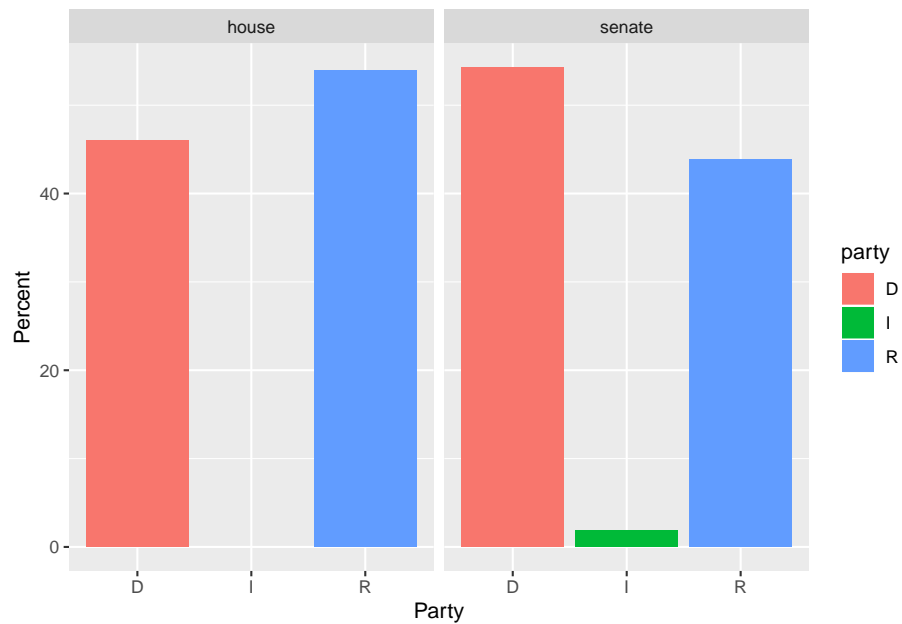


```
# The next display attempts to use percentages on the vertical axis defined within chamber.
# This means the next command must list chamber as the FIRST group_by variable.
ds <- ds1 %>% group_by(chamber, party) %>%
  summarise(n = n()) %>%
  mutate(pct = 100*n / sum(n))
#
ds
```

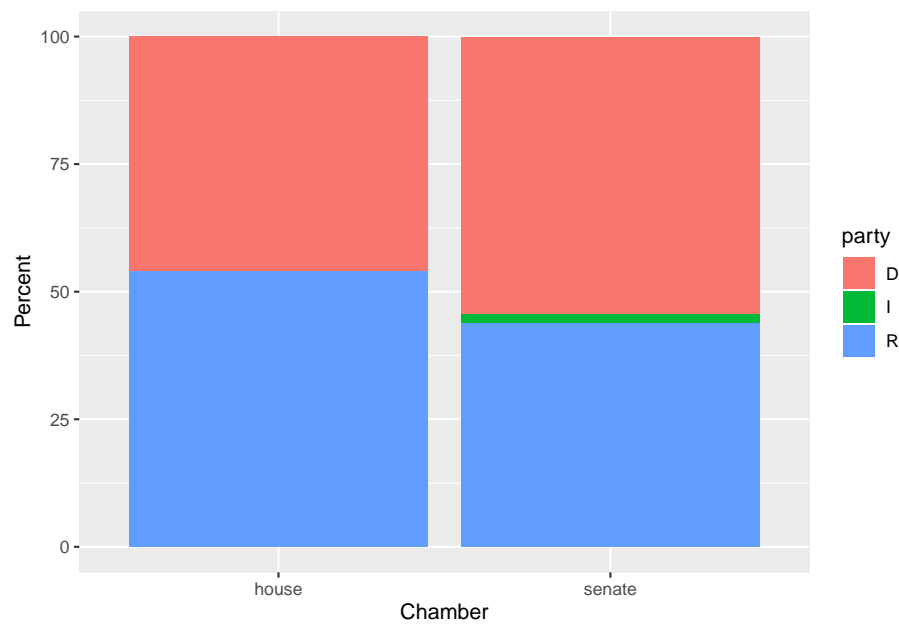
```
## # A tibble: 5 x 4
## # Groups:   chamber [2]
##   chamber party    n  pct
##   <fct>   <fct> <int> <dbl>
## 1 house    D       202  46.0
## 2 house    R       237  54.0
## 3 senate   D        57  54.3
## 4 senate   I         2   1.90
## 5 senate   R        46  43.8
```

```
#
ggplot(data=ds, aes(x=party, y=pct)) + geom_bar(aes(fill=party), stat="identity") +
```

```
facet_wrap(~ chamber) +
labs(x="Party", y="Percent")
```



```
ds1 <- congage %>% filter(congress > 112) %>% select(congress, chamber, state, party, incun
mutate(party = factor(party, levels=c("D", "I", "R")),
      chamber = factor(chamber)) %>%
  na.omit()
#
ds <- ds1 %>% group_by(chamber, party) %>%
  summarise(n = n()) %>%
  mutate(pct = 100*n / sum(n))
#
ggplot(data=ds, mapping=aes(x=chamber, y=pct, fill=party)) +
  geom_col() +
  labs(x="Chamber", y="Percent")
```



```
ds1 <- congage %>% filter(congress > 112) %>% select(congress, chamber, state, party, incumbent, age)
mutate(party = factor(party, levels=c("D", "I", "R")),
       chamber = factor(chamber)) %>%
  na.omit()
#
ds1 %>% group_by(chamber, party) %>%
  summarise (n = n()) %>%
  mutate(pct = 100*n / sum(n)) %>%
ggplot( mapping=aes(x=chamber, y=pct, fill=party)) +
  geom_col() +
  coord_flip() +
  labs(x="Chamber", y="Percent")
```

