Lab 2: R Basics and Descriptives

Contents

Purpose

The purpose of today's lab is to introduce you to some simple tools that will allow you to calculate and visualize **descriptive statistics** in R. This will not be a deep dive into the theoretical meaning behind descriptives, but rather a guide to some practical steps toward getting the most basic summary information out of a given dataset. Along the way, we will implement some of the skills we learned in last week's lab, such as creating variables, working with data frames, installing packages, and learning how to use new functions.

Today's lab will cover:

1. Visualizing Distributions
2. Basic Descriptives

3. Summarizing a dataset
4. Bivariate Descriptives

Getting Started

We'll use the same 2015 World Happiness Report dataset from lecture throughout today's lab.

1. Download the data here.

2. Create a new `.Rmd` document to follow along with the examples. Go ahead and load `{rio}` and `{psych}`, as we are going to need functions from these packages throughout. You will need to install these packages using `install.packages()` if you have not already done so.

```
# load libraries
library(rio)
library(psych)
```

3. Import the World Happiness data using the `rio` packageand save it to an object called `world_happiness`. (See here for a refresher on importing data.)

If the file is saved in your `Downloads` folder, your code might look something like this:

```
world_happiness <- import("~/Downloads/world_happiness_2015.csv")
```

We can take a peek at the data using the `str()` function, which shows us the structure of the dataset. This tells us we have a data frame with 136 observations of 8 different variables: `Country`, `Happiness`, `GDP`, `Support`, `Life`, `Freedom`, `Generosity` and `Corruption`.

```
str(world_happiness)
```

```
## 'data.frame':	136 obs. of  8 variables:
##  $ Country   : chr  "Albania" "Argentina" "Armenia" "Australia" ...
##  $ Happiness : num  4.61 6.7 4.35 7.31 7.08 ...
##  $ GDP       : num  9.25 NA 8.97 10.68 10.69 ...
##  $ Support   : num  0.639 0.926 0.723 0.952 0.928 ...
##  $ Life      : num  68.4 67.3 65.3 72.6 70.8 ...
##  $ Freedom   : num  0.704 0.881 0.551 0.922 0.9 ...
##  $ Generosity: num  -0.0823 NA -0.1867 0.3157 0.0891 ...
##  $ Corruption: num  0.885 0.851 0.901 0.357 0.557 ...
```

We can also look at the first few rows of the data frame using `head()`. Notice that there are some `NA`'s, indicating that there is missing data. This will become important later.

```
head(world_happiness)
```

```
##      Country Happiness       GDP   Support     Life   Freedom  Generosity
## 1    Albania  4.606651  9.251464 0.6393561 68.43517 0.7038507 -0.08233768
## 2  Argentina  6.697131        NA 0.9264923 67.28722 0.8812237          NA
## 3    Armenia  4.348320  8.968936 0.7225510 65.30076 0.5510266 -0.18669653
## 4  Australia  7.309061 10.680326 0.9518616 72.56024 0.9218710  0.31570196
## 5    Austria  7.076447 10.691354 0.9281103 70.82256 0.9003052  0.08908856
## 6 Azerbaijan  5.146775  9.730904 0.7857028 61.97585 0.7642895 -0.22263514
##   Corruption
## 1  0.8847930
## 2  0.8509062
## 3  0.9014622
## 4  0.3565544
## 5  0.5574796
## 6  0.6155525
```

Visualizing Distributions

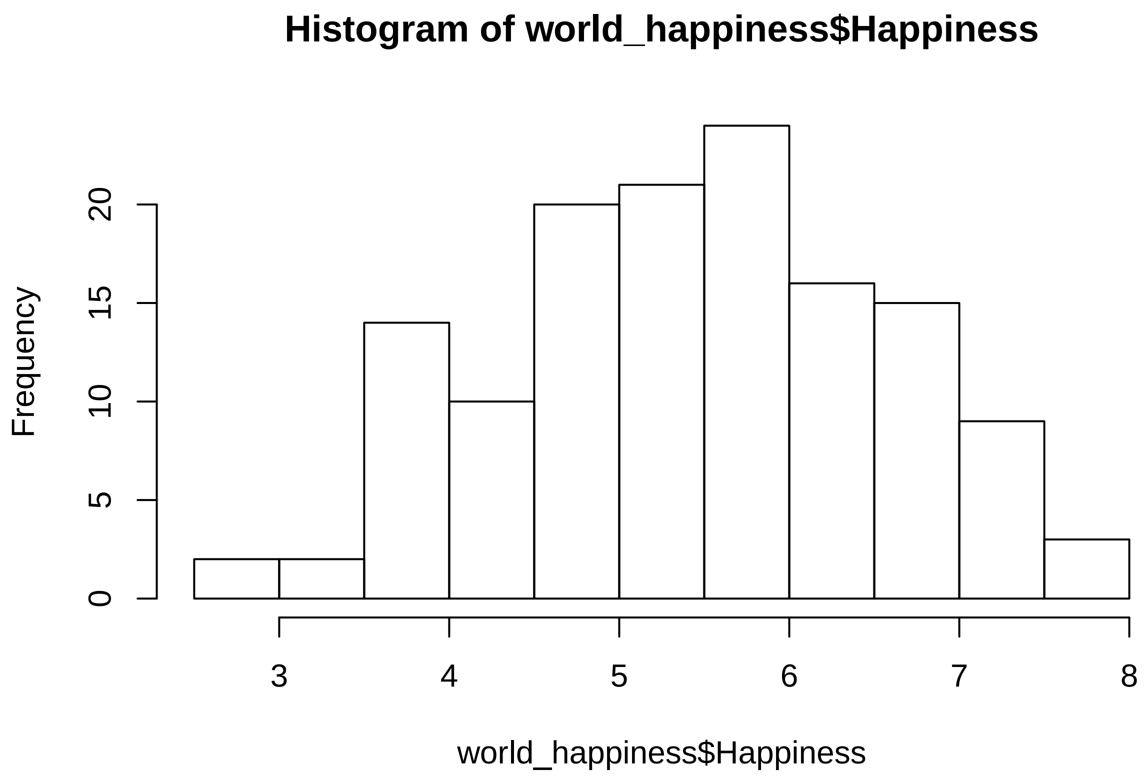
Recall from lecture that a **distribution** often refers to a description of the (relative) number of times a given variable will take each of its unique values.

Histogram

One common way of visualizing distributions is using a **histogram**, which plots the frequencies of different values for a given variable.

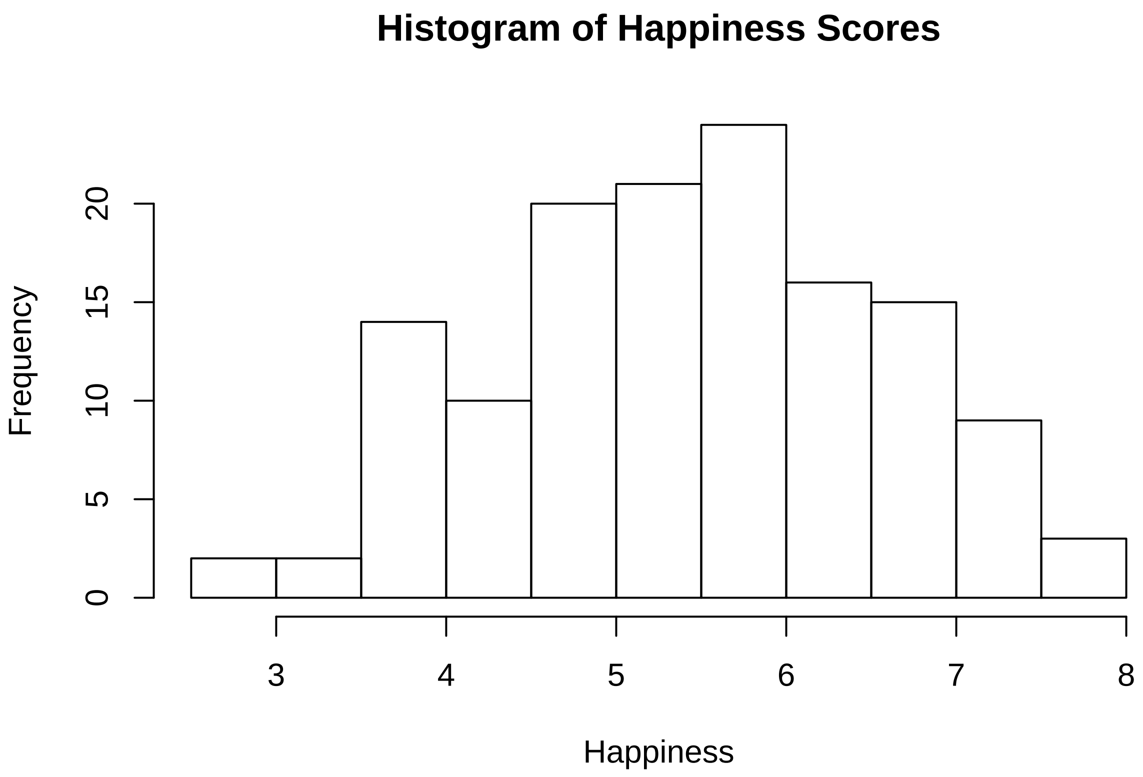
For example, let's take a look at a distribution of the `Happiness` variable. We do this using the `hist()` function. (Remember, you can check out the help documentation using `?hist`).

```r
hist(x = world_happiness$Happiness)
```



This looks ok, but it's kind of ugly. Let's make it better by adding our own title and a better label for the x-axis.

```r
hist(x = world_happiness$Happiness, main = "Histogram of Happiness Scores", xlab = "Happiness")
```

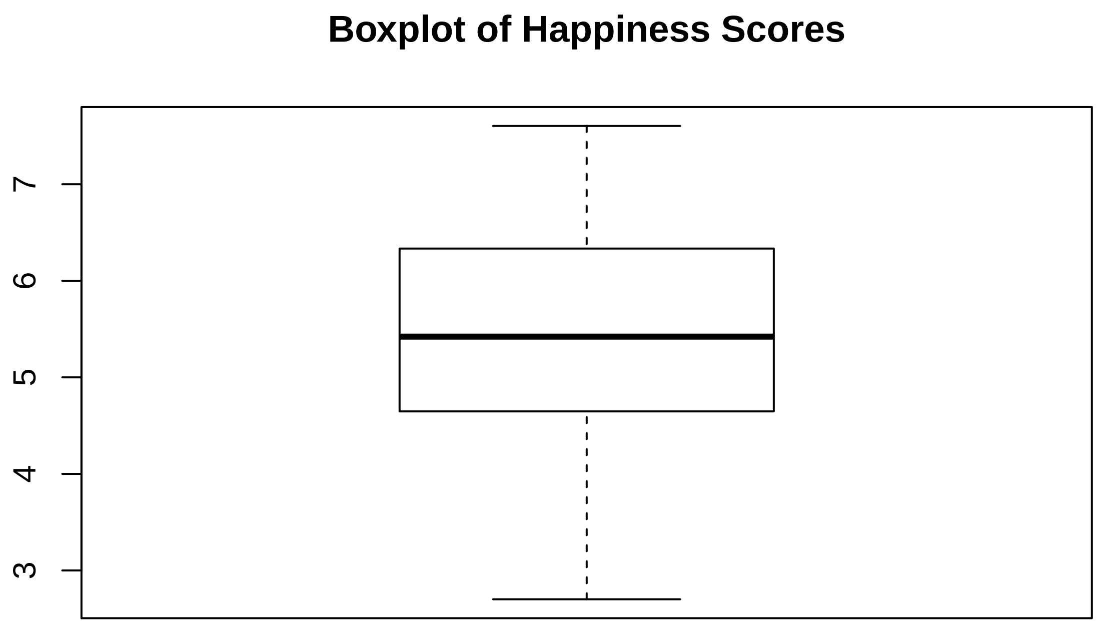


You can also change the number of bins (i.e. bars) in your histogram using the `breaks` argument. Try different values and see how they affect how the histogram looks.

Box Plot

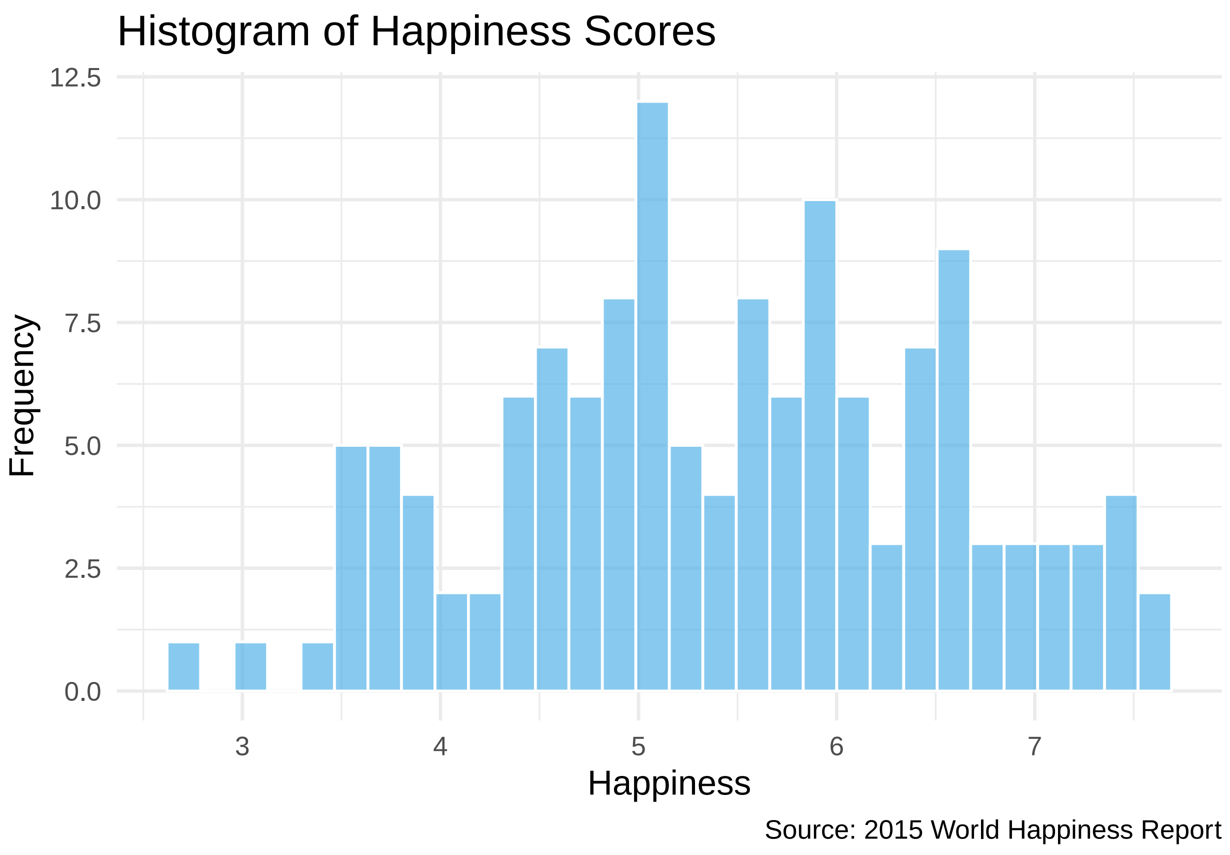
We can also visualize distributions using a **boxplot**, which gives us different information. For a short guide on how to read boxplots, see here or refer to this section of your textbook.

```
boxplot(x = world_happiness$Happiness, main = "Boxplot of Happiness Scores")
```



Looking ahead…

So far we have been plotting in base R. However, the ggplot2 package is generally a much better tool for plotting. For now we'll stick with base plotting to keep things simple, but in a future class you will learn how to use `ggplot` to make better-looking plots, such as this:



Ok, so now that we know how to vizualize a basic distribution, let's think about how we commonly *characterize* distributions with descriptive statistics…

Basic Descriptives

Measures of Central Tendency

For a given set of observations, measures of central tendency allow us to get the "gist" of the data. They tell us about where the "average" or the "mid-point" of the data lies. Let's take a really simple example first. We'll use a vector of integers as our dataset.

```
simple_vector <- c(1, 1, 1, 2, 2, 2, 2, 2, 3, 5, 5, 6, 7)
```

Mean

The mean is the arithmetic average of a set of values. It represents the sum of those values divided by the total number of values. Let's do this using our simple example dataset.

```r
sum(simple_vector)/length(simple_vector)
```

```
## [1] 3
```

However, a faster (and better) way to find the mean is to use the aptly named `mean()` function from base R.

```r
mean(simple_vector)
```

```
## [1] 3
```

What happens if we try to calculate the mean when there are missing values in our data? Let's see what happens when we add a couple `NA`'s to our simple vector and calculate the mean.

```r
# add NA's
simple_vector_missing <- c(1, 1, NA, 1, 2, 2, 2, 2, 2, 3, 5, NA, 5, 6, 7)

mean(simple_vector_missing)
```

```
## [1] NA
```

...It gives us an `NA`! The reason for this is that the mean is calulated by using every value for a given variable, so if you don't remove (or impute) the missing values before getting the mean, it won't work. If there are missing values in your data (which there often are in real life), then you will have to add an additional argument: `na.rm = TRUE`.

```r
mean(simple_vector_missing, na.rm = TRUE)
```

```
## [1] 3
```

Median

The median is the middle value of a set of observations: 50% of the data points fall below the median, and 50% fall above. To find the median, we can use the `median()` function.

```r
median(simple_vector)
```

```
## [1] 2
```

Mode

Oddly enough, the core packages in R do not contain a function to find the mode. In the case of our very simple dataset, it is easy to determine that the mode (i.e. the most common value) is `2`. Your first minihack exercise will be to find the mode using the R package that accompanies your textbook.

Measures of Variability

Range

The range gives us the distance between the smallest and largest value in a dataset. You can find the range using the `range()` function, which will output the minimum and maximum values.

```r
range(simple_vector)
```

```
## [1] 1 7
```

Variance and standard deviation

To find the variance and standard deviation, we use `var()` and `sd()`, respectively.

```
# variance
var(simple_vector)
```

```
## [1] 4.166667
```

```
# standard deviation
sd(simple_vector)
```

```
## [1] 2.041241
```

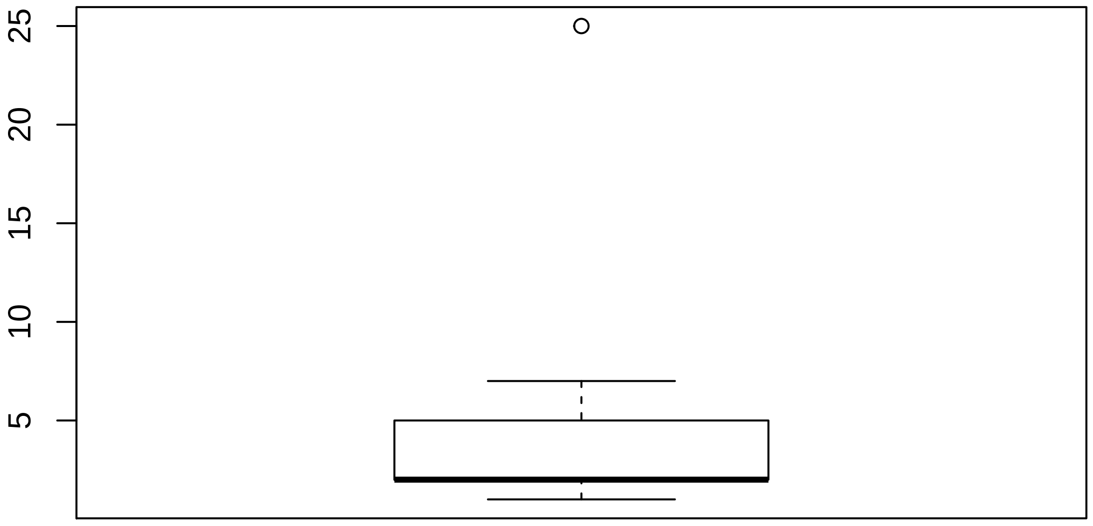
***Note: As it pertains to missing data, the same rule described above for the `mean()` function also applies to these functions (namely, that you need to add the argument `na.rm = TRUE` when there is missing data).

Outliers

Outliers are extreme values in the data. They don't follow the same general pattern as all the other data. One easy way to detect outliers is to visually inspect a distribution for values that stand out. For example, if we add an extreme value to our simple vector, it is easy to see that it is an outlier when we look at the boxplot.

```
# add extreme value (25)
simple_vector_outlier <- c(1, 1, 1, 2, 2, 2, 2, 2, 3, 5, 5, 6, 7, 25)

# visualize outlier with boxplot
boxplot(simple_vector_outlier)
```



z-scores

Another way to identify outliers is to **standardize** the scores in our dataset, i.e. to convert them to z-scores. To do this, we take each score's distance from the mean and divide it by the standard deviation:

$$z = \frac{x_i - M}{s}$$

This tells us, for each score, how many standard deviations it is away from the mean. Some people use the rule of thumb that if a score is above or below 3 standard deviations from the mean, it should be considered an outlier.

To standardize scores in our dataset, we can use the `scale()` function.

```
simple_vector_outlier <- c(1, 1, 1, 2, 2, 2, 2, 2, 3, 5, 5, 6, 7, 25)

as.numeric(scale(simple_vector_outlier)) # the as.numeric() here just returns a vector instead of a mat
```

```
##  [1] -0.57620491 -0.57620491 -0.57620491 -0.41486753 -0.41486753
##  [6] -0.41486753 -0.41486753 -0.41486753 -0.25353016  0.06914459
## [11]  0.06914459  0.23048196  0.39181934  3.29589207
```

Based on this output, how many standard deviations above the mean is our extreme value, 25? Should it be considered an outlier?

Summarizing data

So far we have been calculating various descriptive statistics (somewhat painstakingly) using an assortment of different functions. So what if we have a dataset with a bunch of variables we want descriptive statistics for? Surely we don't want to calculate descriptives for each variable by hand...

Fortunately for us, there is a function called `describe()` from the `{psych}` package, which we can use to quickly summarize a whole set of variables in a dataset.

Let's look back at our `world_happiness` dataset...

`psych::describe()`

This function automatically calculates all of the descriptives we reviewed above (and more!)

```
psych::describe(world_happiness)
```

```
## Warning in psych::describe(world_happiness): NAs introduced by coercion

## Warning in FUN(newX[, i], ...): no non-missing arguments to min; returning
## Inf

## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf

##            vars   n  mean   sd median trimmed  mad   min   max range  skew
## Country*      1 136   NaN   NA     NA     NaN   NA   Inf  -Inf  -Inf    NA
## Happiness     2 136  5.43 1.11   5.42    5.44 1.20  2.70  7.60  4.90 -0.07
## GDP           3 121  9.22 1.16   9.45    9.28 1.20  6.61 11.43  4.82 -0.43
## Support       4 135  0.80 0.12   0.83    0.81 0.12  0.43  0.99  0.55 -0.88
## Life          5 135 63.12 7.46  64.64   63.73 7.35 43.74 76.04 32.30 -0.67
## Freedom       6 132  0.75 0.13   0.78    0.76 0.16  0.40  0.98  0.58 -0.45
## Generosity    7 120  0.00 0.16  -0.03   -0.01 0.15 -0.28  0.46  0.74  0.59
## Corruption    8 125  0.73 0.20   0.81    0.77 0.12  0.09  0.96  0.87 -1.48
##            kurtosis   se
## Country*         NA   NA
## Happiness     -0.69 0.10
## GDP           -0.78 0.11
## Support        0.11 0.01
## Life          -0.34 0.64
## Freedom       -0.60 0.01
## Generosity    -0.27 0.01
## Corruption     1.53 0.02
```

NOTE: `Country` is not `numeric`, it is a **categorical** variable of type `character`. By default, the `describe()` function forces non-numeric variables to be numeric and attempts to calculate descriptives for them. These variables are marked with an asterisk (`*`). In this case, it doesn't make sense to calculate descriptive statistics for `Country`, so we get a warning message and a bunch of `NaN`'s and `NA`'s for this variable.

A better approach would be to remove non-numeric variables before you attempt to run numerical calculations on your dataset. You will do just that in Minihack #4.

If we want to focus on a particular variable, we can calculate descriptives for that variable only.

```
psych::describe(world_happiness$Happiness)
```

```
##    vars   n mean   sd median trimmed mad min max range  skew kurtosis  se
## X1    1 136 5.43 1.11   5.42    5.44 1.2 2.7 7.6   4.9 -0.07    -0.69 0.1
```

***Note that `psych::describe()` also gives us skew and kurtosis.

Bivariate Descriptives

Bivariate descriptives pertain to the relationship between two variables.

Correlation

`cor()` function

To find the strength of the association between two variables, we can use the `cor()` function from the `{psych}` package.

For example, in our `world_happiness` dataset, we might want to know the strength of the relationship between a country's `Happiness` and average level of social support, i.e. `Support`.

```
cor(world_happiness$Happiness, world_happiness$Support, use = "pairwise.complete.obs")
```

```
## [1] 0.7258971
```

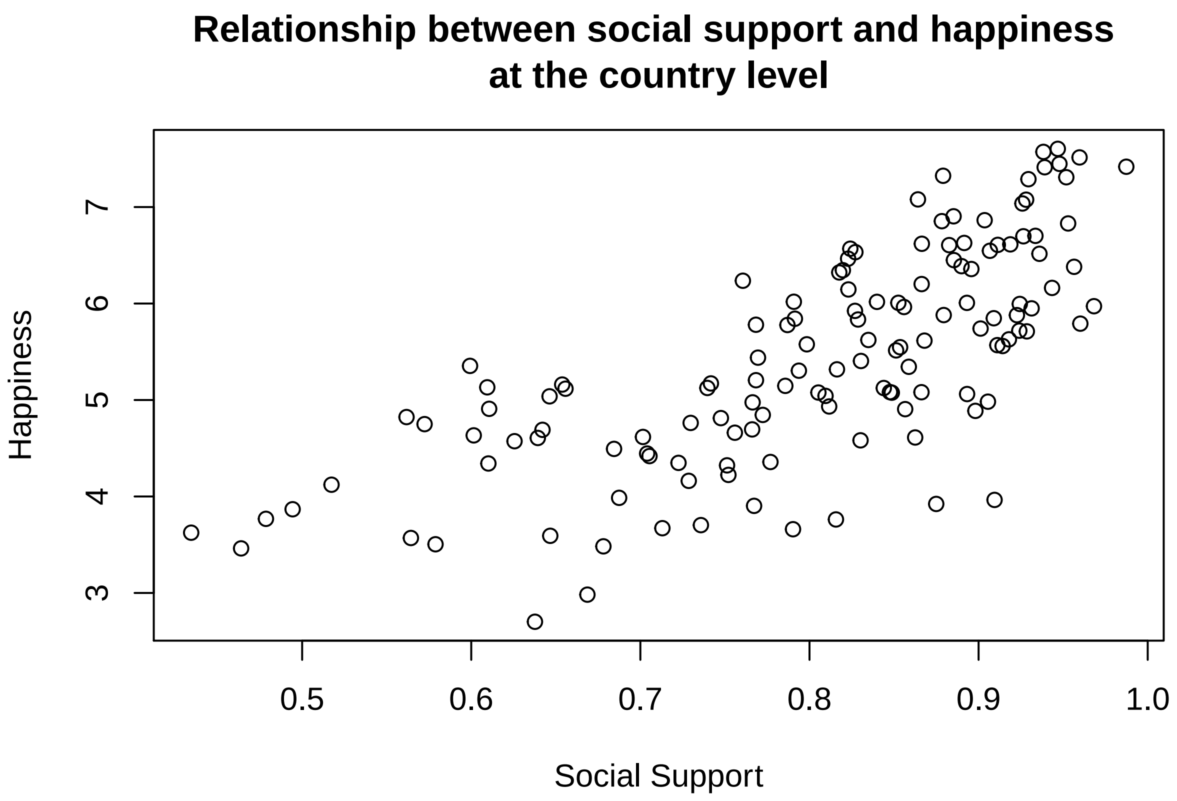
Not surprisingly, we see that there is a strong positive association between happiness and social support.

***Note the use of `use = "pairwise.complete.obs"`. Because there is missing data in these variables, this argument is used to tell R to only use observations (i.e. rows in the data frame) for which there is no missing data for either variable.

Scatterplots

If we want to see this relationship graphically, we can create a *scatterplot*.

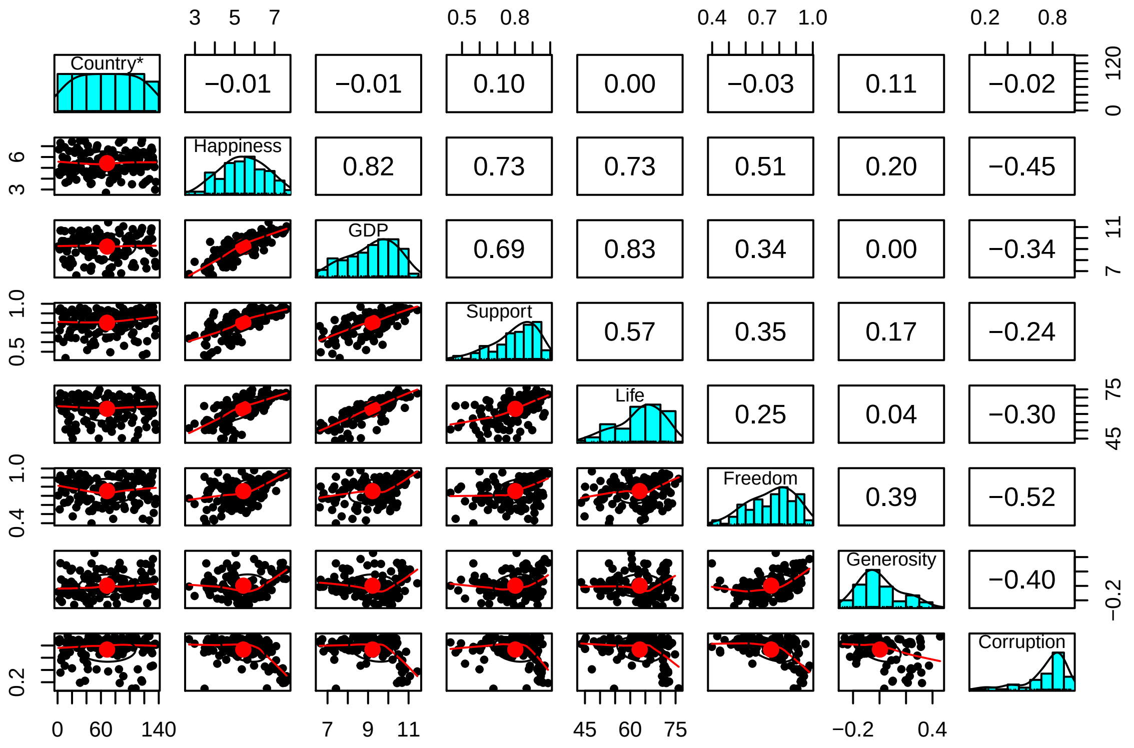
```
plot(world_happiness$Support, world_happiness$Happiness, xlab = "Social Support", ylab = "Happiness", ma
```



`pairs.panels()` function

If we have a dataset with a reasonably small number of variables, we can use `pairs.panels()` (also from the `{psych}` package) to generate a matrix that contains scatterplots between all of the variables below the diagonal and Pearson correlations between all of the variables above the diagonal.

```r
pairs.panels(world_happiness)
```



Again, note that we can't interpret the results for `Country` because it is a non-numeric character variable.

Covariance

Lastly, if we want to find the covariance between two variables, we use the `cov()` function.

```
cov(world_happiness$Happiness, world_happiness$Support, use = "pairwise.complete.obs")
```

```
## [1] 0.09892871
```

NOTE: You can also generate a covariance matrix for a whole dataframe by passing a dataframe to `cov()` instead of only two specific variables. The same goes for generating a correlation matrix using `cor()`.

Minihacks

Feel free to work in groups of 2-3, and ask for help when you get stuck!

Minihack 1: Find the mode

1. Install and load the `{lsr}` package, the companion R package to your textbook, Learning Statistics with R.
2. Check out the help documentation for the function `modeOf()`.
3. Use `modeOf()` to find the mode of `simple_vector`. Make sure that the answer you get is 2. Save your result to an object called `mode`. If you don't already have `simple_vector` loaded in your workspace, here it is again:

```
simple_vector <- c(1, 1, 1, 2, 2, 2, 2, 2, 3, 5, 5, 6, 7)
```

4. Use the function `maxFreq()` to find the frequency of the modal value (i.e. how many time it occurs). Save your result to an object called `modal_freq`.

Minihack 2: Outliers

1. Download this file and import it into R. Save it as an object called `corruption_std`. This contains a reduced form of the World Happiness dataset with three variables:

-`Country` -`Corruption` -`Corruption_scaled` (a standardized, i.e. z-scored, version of the `Corruption` variable)

2. Create a histogram of `Corruption`. What do you notice about the shape of this distribution? How would you describe the skewness?
3. View `corruption_std`. Based on the z-scores (i.e. `Corruption_scaled`), which countries would be considered outliers on the `Corruption` variable?
4. After identifying which countries are outliers, use what you learned in last week's lab about indexing data frames to remove these rows from the data frame.

-Hint #1: you can use the minus sign (`-`) for de-selecting. -Hint #2: pay attention to rows `124` and `125`

Minihack 3: Descriptive plots

For this minihack, we're going to work with a slightly different version of the `world_happiness` data. You can download the new version here. This version has two new variables:

-`War` represents whether or not a country was an active participant in WWII or was invested in by the US or Soviet Union just after WWII

-`Service` represents the proportion of the population that works in the service industry. ***Note: this variable has been simulated and is completely made up!***

1. Import the new version of the data and save it to an object called `world_happiness_sim`.
2. Create a histogram of the `Service` variable. Set the number of bins to `30`. What do you notice about the shape of this distribution?
3. Look up the help documentation for `psych::describeBy()`. Use this function to get descriptive statistics for the `Service` variable for countries involved/not involved in WWII. (That is use `War` as your grouping variable). What are the means and standard deviations for the two groups?
4. Based on what you observe about these group means, how might this explain the shape of the distribution of the `Service` variable?

Minihack 4: Correlations

For this minihack, we're going to use the `cor()` function from `{psych}` to create a correlation matrix of the entire `world_happiness` dataset. Be sure to use the original `world_happiness` data frame without the added variables from Minihack #3.

1. First remove the `Country` variable because this variable is not numeric and will not yield meaningful correlations with the other variables in the dataset. Save the resulting dataframe to a new object called `world_happiness_num`.
2. Use `cor()` to generate a correlation matrix for all the variables in `world_happiness_num`. Use listwise deletion by specifying `use = "complete.obs"`.
3. What is the correlation between `Freedom` and `Happiness`? Between `Freedom` and `Corruption`? Do these correlations make intuitive sense?
4. Now generate another correlation matrix, but this time use pairwise deletion by specifying `use = "pairwise.complete.obs"`.
5. Compare your two correlation matrices. Whey do they have different numbers? What is the difference between pairwise and listwise (aka “casewise”) deletion? Hint: look at the help documentation for `cor()`.