Session 1 - R Basics

Erik Steeb

2/10/2022

Agenda

- 9am - 12pm : Session 1: R Basics
- 12pm - 1pm : Lunch
- 1pm - 3pm : Session 2: Practical Application

Session 1 Agenda

- What is R? R Studio? R Markdown?
- Tidyverse
- Basic R Syntax
- R Datatypes

BREAK

- Dataframes
- Data visualization
- R Functions

BREAK

- R for data Analysis
- Classification

LUNCH

What is R?

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

- Free as in beer
- Free as in speech
- An interpreted language
- Extendable with packages (https://cran.r-project.org/)

Source: https://www.r-project.org/about.html

What is RStudio?

- An R IDE
- Free as in beer
- Free(ish) as in speech
- A company that maintains & promotes major R packages

What is R Markdown?

- Necessary for Fallaw's elective
- Jupyter Notebooks for R
- A way to narrate an analytics process
- Capable of producing slide decks, pdfs, or html documents
- The way you're currently viewing this workshop

Tidyverse Overview

- Collection of packages that "share an underlying design philosophy, grammar, and data structures."
- Tidy data: – A standard method of displaying a multivariate set of data is in the form of a data matrix in which rows correspond to sample individuals and columns to variables, so that the entry in the ith row and jth column gives the value of the jth variate as measured or observed on the ith individual. – Each variable is a column, each observation is a row

R Basics:

"Hello world"

```r
print("Hello World")
```

```
## [1] "Hello World"
```

R Basics: Variables, basic operations

Use normal math operators. Use <- to assign variables

```r
a <- 3
b <- 2
c <- a+b
print(c)
```

```
## [1] 5
```

```
d <- a^b
print(d)
```

```
## [1] 9
```

R Basics: Equals Sign

You can use `=` to assign variables, but `<-` is standard

```
a = 1
print(a)
```

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

Getting in the habit of using `=` can lead to problems when using conditional statements. Conditional statements use "==" to evaluate logic.

```
b = 1
if (a==b) print("Same")
```

```
## [1] "Same"
```

R Basics: Data Types

Like most other programming languages, R has a number of datatypes. We've already looked at numeric data in our previous examples. Other basic data types are: - Character - Integer (special case of numeric) - Logical - Complex (special case of numeric)

R Basics: Data Structures

- Vectors: Most common structure, contains a series of elements – Created using vector() – Also created using c() function (more common)
- Matrix: Vector with dimensions
- List: A collection of items, can contain vectors, matrices, or other lists
- Data Frame: Essentially the R version of a spreadsheet – Can contain a **header row** - the name of each column. These can be set with the colnames() function

Questions?

Take a ten-minute break

Dataframe Example: mtcars

mtcars is a dataset built in to R that

```r
df <- mtcars
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Dataframe Example: mtcars

We can look up specific cells in a dataframe in two ways: By number or by the name of the row and column:

```r
df[2,1]
```

```
## [1] 21
```

```r
# This shows us what is in row 2, column 1
```

```r
df["Mazda RX4 Wag",'disp']
```

```
## [1] 160
```

```r
# This shows us what the displacement for a Mazda RX4 Wag is
```

Manipulating Dataframes

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

The dplyr package is a set of tools designed to manipulate dataframes. As you can see from the warnings, it replaces several functions from the base R language (which are designed for time series). You can still use those functions, however, by specifying which package to call the functions from, i.e. stats::filter.

Manipulating Dataframes

Let's say we only want to look at 8-cylinder cars:

```
eight_cyl_cars <- filter(df,cyl==8)
head(eight_cyl_cars)
```

```
##                      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
## Duster 360          14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.73 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.78 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.25 17.98  0  0    3    4
```

Alternatively, we can write that expression like this:

```
eight_cyl_cars <- df%>%
  filter(cyl==8)
```

Manipulating Dataframes: Pipe Operator

The advantage of the pipe operator (%>%) is that it allows you to chain multiple operations together. For example, instead of writing:

```
eight_cyl_cars <- filter(df, cyl == 8)
eight_cyl_cars <- select(eight_cyl_cars, mpg, hp, disp)
```

We can write it this way:

```
eight_cyl_cars_1 <- df %>%
  filter(cyl == 8)%>%
  select(mpg, hp, disp)

identical(eight_cyl_cars,eight_cyl_cars_1)
```

```
## [1] TRUE
```

Manipulating Dataframes: Pipe Operator

Advantages: - Allows you to extend the same process - Reduce copy-paste errors (leaving the initial variable, instead of using intermediate variables) - Results in cleaner-looking code that's easier to follow.

Disadvantage: - Lack of intermediate variables makes it potentially harder to debug when there's unexpected behavior.

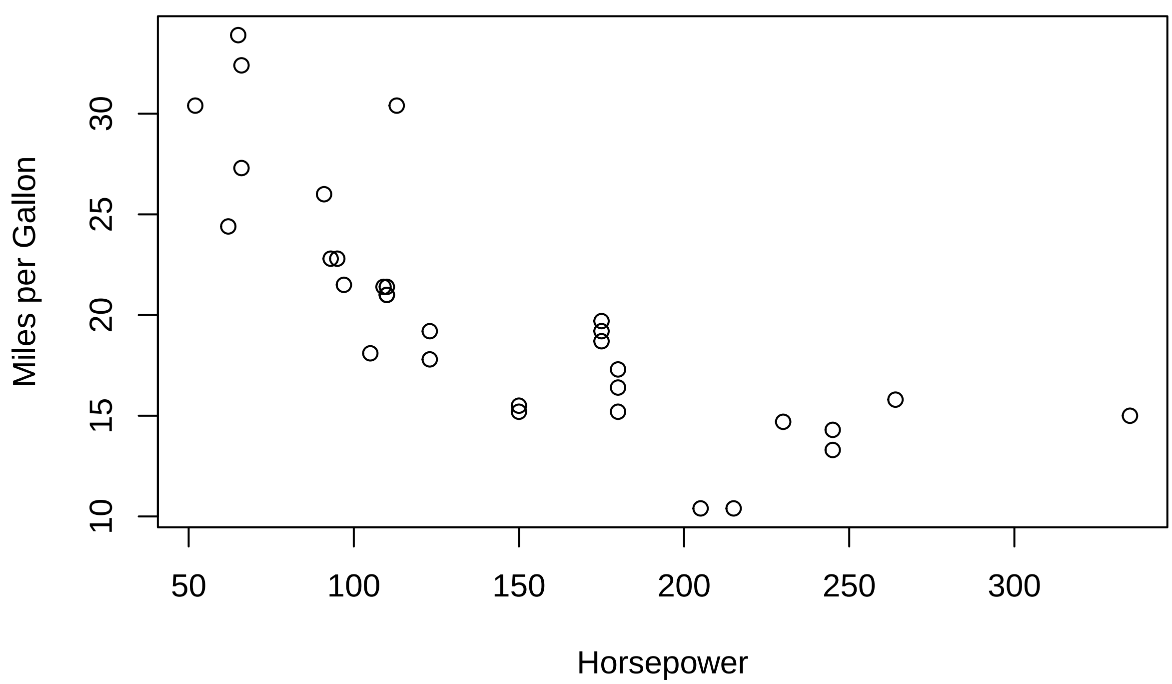
Visualization

Visualizing data can be a very powerful tool to initial get an idea of what your dataset looks like. It can give you an idea if you're looking at linear data, exponential growth, or exponential decay, and potentially highlight any missing or anomalous variables.

Visualization: Base Graphics

R has a built-in graphics package.

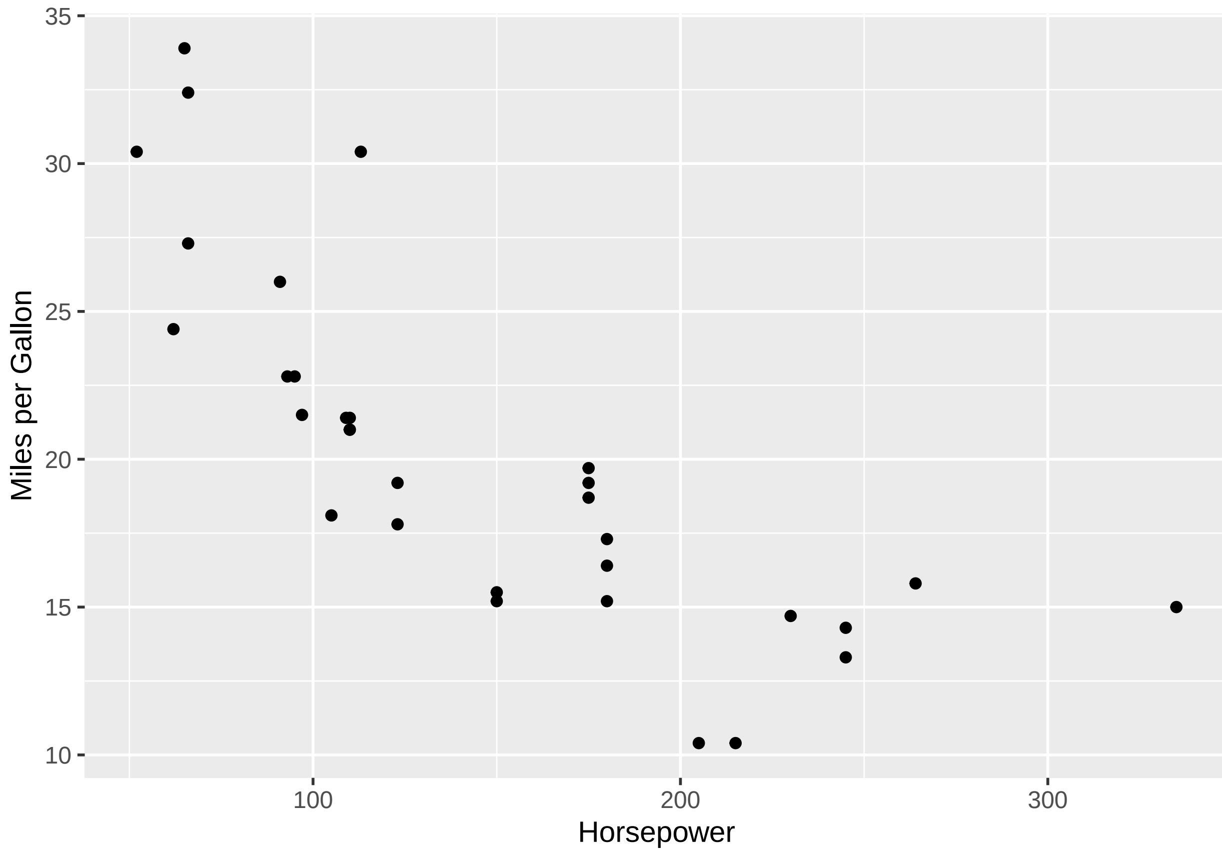
```
plot(y=df$mpg,x=df$hp, ylab = "Miles per Gallon", xlab = "Horsepower")
```



Visualization: ggplot

ggplot is the graphics package that is integrated into the tidyverse. It takes an declarative approach to creating graphics and is based on the book The Grammar of Graphics (which I need to read). Here's an example plot of what we just looked at.

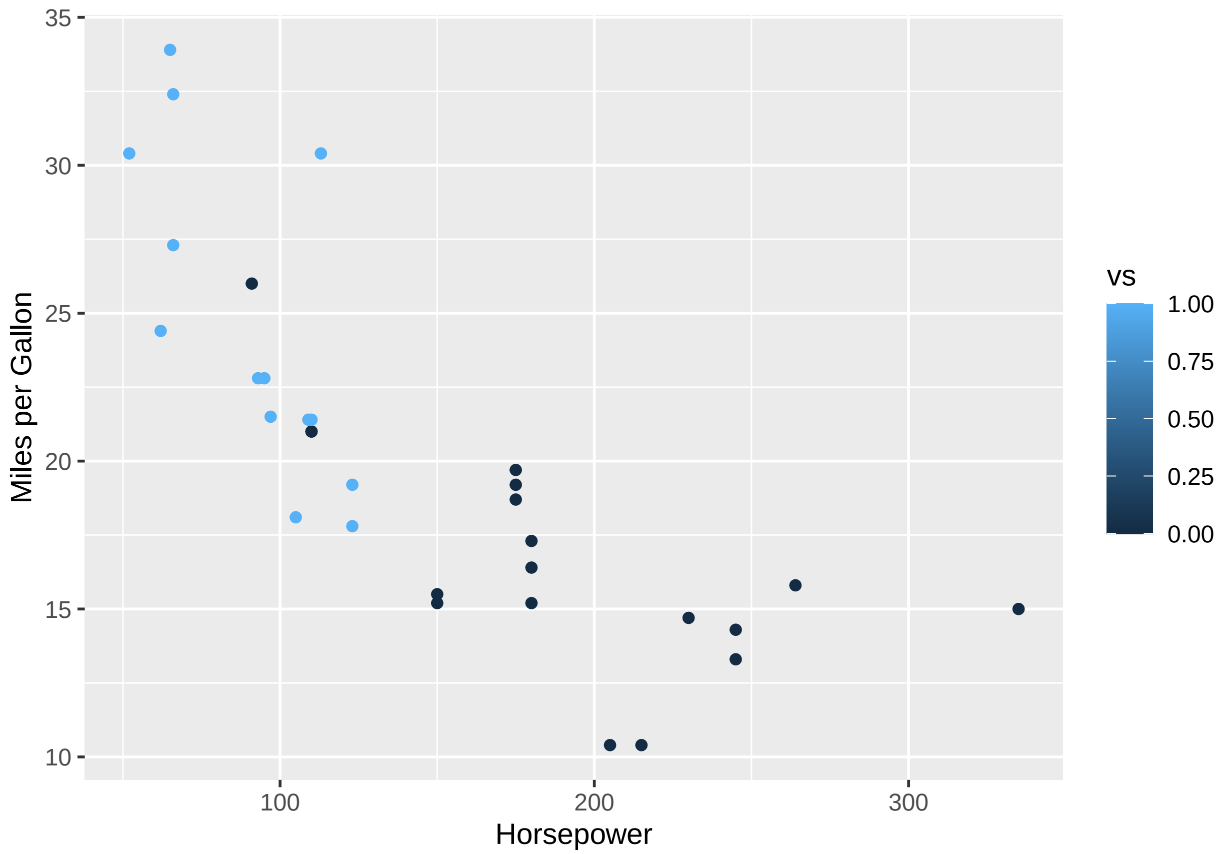
```
library(ggplot2)
p <- ggplot(data = df, mapping = aes(x=hp, y=mpg))+
  geom_point()+
  labs(x="Horsepower",y="Miles per Gallon")
p
```



Visualization: ggplot

We're also able to layer more information on the same plot. For example, color-coding points based on whether a car is manual (1) or automatic (0) transmission

```
p <- ggplot(data = df, mapping = aes(x=hp, y=mpg))+
  geom_point(aes(color=vs))+
  labs(x="Horsepower",y="Miles per Gallon")
p
```



Questions?

R Functions

We've already used a lot of functions, written by other people. But R also allows you to write your own functions.

Functions are an excellent way to avoid having to copy the same piece of code multiple times in the same project. Generally, if you find yourself reusing the same sets of code multiple times, you should write a function.

```
add_multiply <- function(x,y,z){
  var <- x+y
  output <- var*z
  return(output)
}

add_multiply(1,2,3)
```

```
## [1] 9
```

R Functions: Recursive Functions

Functions are also able to call themselves. Example:

```
Factorial <- function(N)
{
if (N == 0)
return(1)
else
return( N * Factorial (N-1))
}
```

Questions?

Take a ten minute break

R for Data Analysis

This is why we're here today.

Data analysis in R breaks down into three main categories: - Data processing - Regression - Classification

R for Data Analysis: Data Processing

Dataset: National-level data from World Bank. Contains the following fields: -iso2c iso3c Two- and Three-letter codes for each country, assigned by the International Organization for Standardization. - country Country name. - year - gdp_percap Gross Domestic Product per capita in current international dollars, corrected for purchasing power in different territories. - life_expect Life expectancy at birth, in years. - population Estimated total population at mid-year, including all residents apart from refugees. - birth_rate Live births during the year per 1,000 people, based on mid-year population estimate. - neonat_mortal_rate Neonatal mortality rate: babies dying before reaching 28 days of age, per 1,000 live births in a given year. - region income World Bank regions and income groups

```
df <- read.csv("nations.csv")
```

R for Data Analysis: Examine Data

Like we did earlier, let's take a look at the structure of the data:

```
str(df)
```

```
## 'data.frame':    5697 obs. of  11 variables:
##  $ iso2c            : chr  "AD" "AD" "AD" "AD" ...
##  $ iso3c            : chr  "AND" "AND" "AND" "AND" ...
##  $ country          : chr  "Andorra" "Andorra" "Andorra" "Andorra" ...
##  $ year             : int  2007 2011 2013 2008 1992 2006 2009 2010 1994 1993 ...
##  $ gdp_percap       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ life_expect      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ population       : num  82683 83751 80788 83861 58888 ...
##  $ birth_rate       : num  10.1 NA NA 10.4 12.1 10.6 9.9 9.8 10.9 11.4 ...
##  $ neonat_mortal_rate: num  1.5 1.3 1.2 1.4 3.6 1.6 1.4 1.3 3.1 3.4 ...
##  $ region           : chr  "Europe & Central Asia" "Europe & Central Asia" "Europe & Central Asia"
##  $ income           : chr  "High income" "High income" "High income" "High income" ...
```

R for Data Analysis: Summary Statistics

And some summary statistics for that data:

```r
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
describe(df)
```

```
##                    vars    n        mean           sd     median     trimmed
## iso2c*                1 5670      105.50        60.63     105.50      105.50
## iso3c*                2 5697      106.00        60.92     106.00      106.00
## country*              3 5697      106.00        60.92     106.00      106.00
## year                  4 5697     2003.00         7.79    2003.00     2003.00
## gdp_percap            5 4918    13529.52     17159.24    6843.80    10041.22
## life_expect           6 5268       68.19         9.73      70.70       69.10
## population            7 5681 30012558.35 120518403.46 5378867.00 10211287.46
## birth_rate            8 5385       23.90        11.78      21.45       22.99
## neonat_mortal_rate    9 5130       18.92        15.03      14.50       17.18
## region*              10 5697        3.54         2.17       3.00        3.42
## income*              11 5697        2.69         1.54       3.00        2.62
##                           mad     min          max        range  skew kurtosis
## iso2c*                  77.84    1.00 2.100000e+02 2.090000e+02  0.00    -1.20
## iso3c*                  78.58    1.00 2.110000e+02 2.100000e+02  0.00    -1.20
## country*                78.58    1.00 2.110000e+02 2.100000e+02  0.00    -1.20
## year                    10.38 1990.00 2.016000e+03 2.600000e+01  0.00    -1.20
## gdp_percap            7835.71  242.00 1.400372e+05 1.397952e+05  2.56     8.86
## life_expect              8.62   27.61 8.542000e+01 5.781000e+01 -0.81    -0.03
## population         7700969.85 9003.00 1.378665e+09 1.378656e+09  8.93    85.51
## birth_rate              13.54    6.90 5.556000e+01 4.866000e+01  0.53    -0.88
## neonat_mortal_rate      14.97    0.60 7.500000e+01 7.440000e+01  0.88     0.01
## region*                  1.48    1.00 7.000000e+00 6.000000e+00  0.61    -1.11
## income*                  2.97    1.00 5.000000e+00 4.000000e+00  0.40    -1.27
##                            se
## iso2c*                   0.81
## iso3c*                   0.81
## country*                 0.81
## year                     0.10
## gdp_percap             244.68
## life_expect              0.13
## population         1598972.43
## birth_rate               0.16
## neonat_mortal_rate       0.21
## region*                  0.03
## income*                  0.02
```

R for Data Analysis: Changing Data Types

If you need to change the data type for any column, use the following functions:

- as.character converts to a text string.
- as.numeric converts to a number that can include decimal fractions.
- as.factor converts to a categorical variable.
- as.integer converts to an integer.
- as.Date converts to a date.
- as.POSIXct converts to a full date and timestamp.

R for Data Analysis: Changing Data Types

Let's change the data type for population and income:

```
#convert population to numeric
df$population <- as.numeric(df$population)

#convert income to factor
df$income <- as.factor(df$income)

#convert region to factor
df$region <- as.factor(df$region)

str(df)
```

```
## 'data.frame':    5697 obs. of  11 variables:
##  $ iso2c            : chr  "AD" "AD" "AD" "AD" ...
##  $ iso3c            : chr  "AND" "AND" "AND" "AND" ...
##  $ country          : chr  "Andorra" "Andorra" "Andorra" "Andorra" ...
##  $ year             : int  2007 2011 2013 2008 1992 2006 2009 2010 1994 1993 ...
##  $ gdp_percap       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ life_expect      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ population       : num  82683 83751 80788 83861 58888 ...
##  $ birth_rate       : num  10.1 NA NA 10.4 12.1 10.6 9.9 9.8 10.9 11.4 ...
##  $ neonat_mortal_rate: num  1.5 1.3 1.2 1.4 3.6 1.6 1.4 1.3 3.1 3.4 ...
##  $ region           : Factor w/ 7 levels "East Asia & Pacific",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ income           : Factor w/ 5 levels "High income",..: 1 1 1 1 1 1 1 1 1 1 ...
```

R for Data Analysis: Selecting Relvant Information

We can filter and sort data to look at the countries with the lowest life expectancy. Later, we'll also see if high income and region is correlated with life expectancy. We'll only look at the most recent year on record (2016) and remove any countries where we don't have life expectancy data.

```
longest_lived <- df%>%
  filter(year == 2016 & !is.na(life_expect))%>%
  select(country, life_expect,gdp_percap,income,region)%>%
  arrange(life_expect)
head(longest_lived)
```

```
##                    country life_expect gdp_percap              income
## 1             Sierra Leone      51.835  1476.2137          Low income
## 2 Central African Republic      52.171   698.7067          Low income
## 3                     Chad      52.903  1990.7267          Low income
## 4                  Nigeria      53.428  5861.0897 Lower middle income
## 5            Cote d'Ivoire      53.582  3693.4369 Lower middle income
## 6                  Lesotho      54.174  2951.0219 Lower middle income
##               region
## 1 Sub-Saharan Africa
## 2 Sub-Saharan Africa
## 3 Sub-Saharan Africa
## 4 Sub-Saharan Africa
## 5 Sub-Saharan Africa
## 6 Sub-Saharan Africa
```

R for Data Analysis: Adding data

We can also calculate each country's ranking in terms of life expectancy by adding a new column:

```
longest_lived_1 <- longest_lived%>%
  mutate(life_rank = rank(desc(life_expect)))%>%
  arrange(life_rank)
head(longest_lived_1)
```

```
##                country life_expect gdp_percap      income                region
## 1 Hong Kong SAR, China    84.22683   58617.97 High income   East Asia & Pacific
## 2                Japan    83.98488   42281.19 High income   East Asia & Pacific
## 3     Macao SAR, China    83.84900  105420.41 High income   East Asia & Pacific
## 4          Switzerland    82.89756   63888.73 High income Europe & Central Asia
## 5                Spain    82.83171   36304.85 High income Europe & Central Asia
## 6            Singapore    82.79512   87832.59 High income   East Asia & Pacific
##   life_rank
## 1         1
## 2         2
## 3         3
## 4         4
## 5         5
## 6         6
```

R for Data Analysis: Grouping and Summarizing

We may also be interested in how life expectancy has changed over time. One potential issue with our data here is that we don't have data for every country, in every year, so we'll want to know how many countries are included. If it changes by a large number, we may need to decide whether that year is a good data point. If one or two countries are added or dropped, this may not be an issue.

```
life_expect_summary <- df %>%
  #Drop all rows with an NA for life_expect
  filter(!is.na(life_expect))%>%
  group_by(year)%>%
  # Start summary
  summarize(countries = n(), # count number of countries per year
```

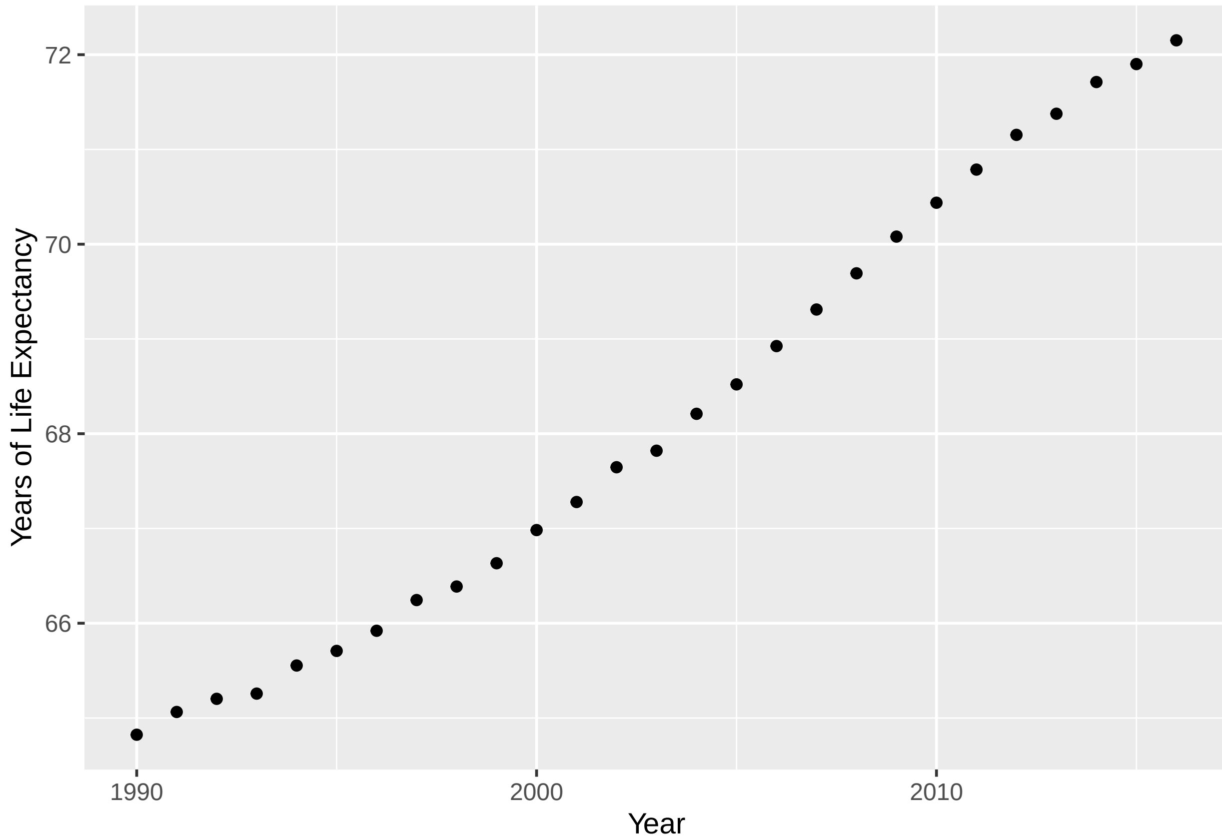
```r
            avg_life_expect = mean(life_expect), #average life expectancy
            max_life_expect = max(life_expect), # maximium life expectancy
            min_life_expect = min(life_expect)  # minimum life expectancy
            )%>%
  mutate(range_life = max_life_expect - min_life_expect)%>%
  arrange(desc(year))
head(life_expect_summary)
```

```
## # A tibble: 6 x 6
##    year countries avg_life_expect max_life_expect min_life_expect range_life
##   <int>     <int>           <dbl>           <dbl>           <dbl>      <dbl>
## 1  2016       195            72.2            84.2            51.8       32.4
## 2  2015       194            71.9            84.3            51.4       32.9
## 3  2014       195            71.7            84.0            50.6       33.4
## 4  2013       195            71.4            83.8            49.8       34.0
## 5  2012       197            71.2            85.4            49.0       36.4
## 6  2011       196            70.8            83.4            48.3       35.1
```

R for Data Analysis: Visualization

Let's take a look at average life expectancy over time.

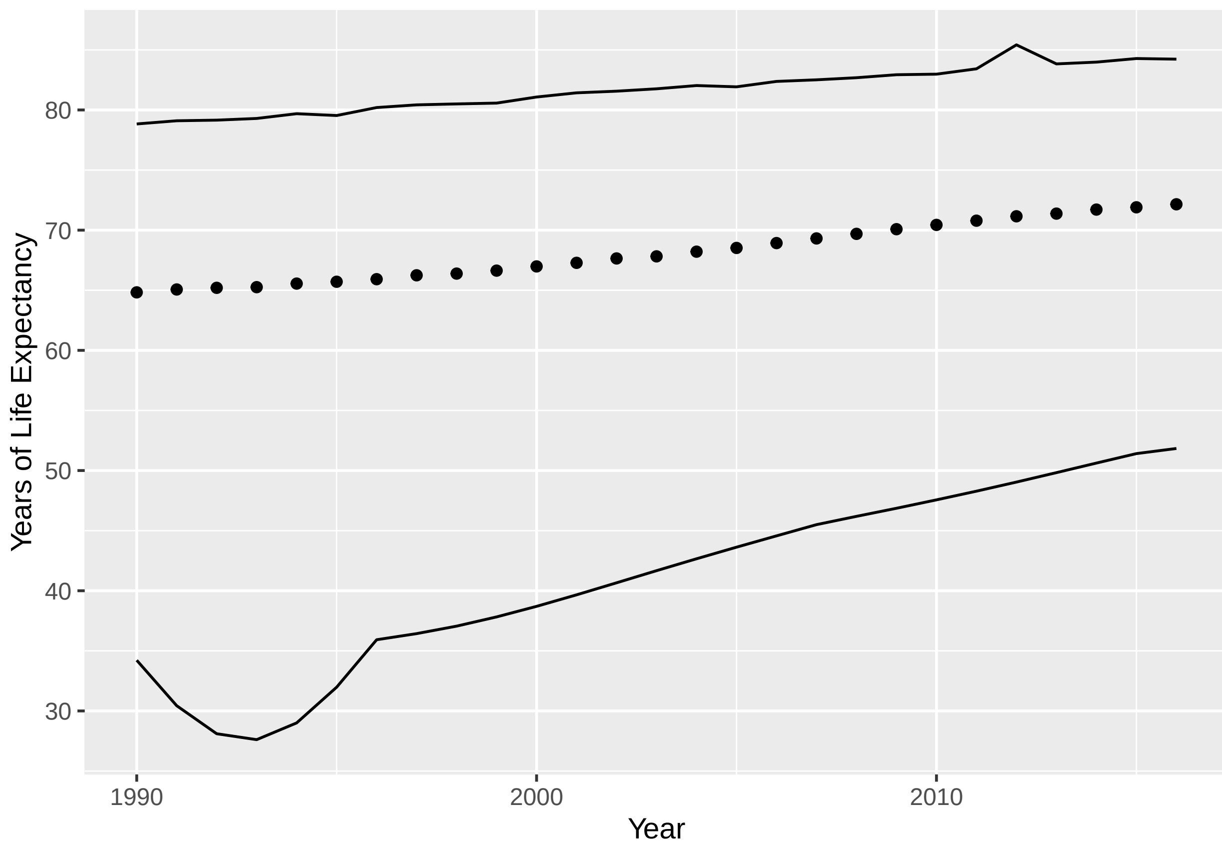
```r
p <- ggplot(data = life_expect_summary)+
  geom_point(mapping = aes(x=year, y=avg_life_expect))+
  labs(x="Year",y="Years of Life Expectancy")
p
```



R for Data Analysis: Visualization

Lets add upper and lower bounds to this plot:

```
p <- ggplot(data = life_expect_summary)+
  geom_point(mapping = aes(x=year, y=avg_life_expect))+
  labs(x="Year",y="Years of Life Expectancy")+
  geom_line(aes(x=year,y=min_life_expect))+
  geom_line(aes(x=year,y=max_life_expect))
p
```



R for Regression

Everyone should be familiar with the math / concepts behind this, so let's just show to do this in R. We first need to remove NAs from the dataset, however. We should also only look at one year, so we'll filter down to 2016 again.

```
df <- df%>%
  filter(!is.na(life_expect) & !is.na(gdp_percap) & year==2016)
life_gdp_regression <- lm(life_expect~gdp_percap,df)
summary(life_gdp_regression)
```

```
##
## Call:
## lm(formula = life_expect ~ gdp_percap, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.855  -3.964   1.954   4.517   8.962
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.728e+01  6.071e-01  110.82   <2e-16 ***
## gdp_percap  2.334e-04  2.055e-05   11.36   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.936 on 173 degrees of freedom
## Multiple R-squared:  0.4273, Adjusted R-squared:  0.424
## F-statistic: 129.1 on 1 and 173 DF,  p-value: < 2.2e-16
```

R for Regression: Exponential data

But wait, per capita GDP is likely to not be linear. There's a chance that taking the log of per capita GDP is going to better describe the data. We can do that directly in the formula.

```
life_log_gdp_regression <- lm(life_expect~log(gdp_percap),df)
summary(life_log_gdp_regression)
```

```
##
## Call:
## lm(formula = life_expect ~ log(gdp_percap), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.9373  -2.0714   0.8925   2.9739   7.1463
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      22.8483     2.6229   8.711 2.36e-15 ***
## log(gdp_percap)   5.2881     0.2802  18.870  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.485 on 173 degrees of freedom
## Multiple R-squared:  0.673,  Adjusted R-squared:  0.6711
## F-statistic: 356.1 on 1 and 173 DF,  p-value: < 2.2e-16
```

R for Regression: Factors

Sometimes, categorical data can explain continuous responses. Recall that our Region grouping is a factor. We need to use a different formula (glm) for this.

```
region_life_regression <- glm(life_expect~region,data = df)
summary(region_life_regression)
```

```
##
## Call:
## glm(formula = life_expect ~ region, data = df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -11.0109   -3.4512    0.0422    3.3312   12.8512
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)                        74.1577     0.9145  81.090  < 2e-16 ***
## regionEurope & Central Asia         3.2513     1.1431   2.844  0.00501 **
## regionLatin America & Caribbean     0.1832     1.2606   0.145  0.88463
## regionMiddle East & North Africa    1.2278     1.4992   0.819  0.41398
## regionNorth America                 6.3376     3.4824   1.820  0.07055 .
## regionSouth Asia                   -3.6232     1.9128  -1.894  0.05992 .
## regionSub-Saharan Africa          -12.6140     1.1617 -10.858  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 22.58095)
##
##     Null deviance: 10642.8  on 174  degrees of freedom
## Residual deviance:  3793.6  on 168  degrees of freedom
## AIC: 1051
##
## Number of Fisher Scoring iterations: 2
```

R for Regression: Multivariate regression

Sometimes, more than one factor may be at play and an important driver of the response variable. For numerical data, we can use the usual lm() function. Let's return to the mtcars dataset.

```r
mpg_regression <- lm(mpg~hp,data=mtcars)
summary(mpg_regression)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

A second variable may help explain mpg better than just one:

```r
mpg_regression <- lm(mpg~hp+wt,data=mtcars)
summary(mpg_regression)
```

```
##
## Call:
```

```
## lm(formula = mpg ~ hp + wt, data = mtcars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.941 -1.600 -0.182  1.050  5.854
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
## hp          -0.03177    0.00903  -3.519  0.00145 **
## wt          -3.87783    0.63273  -6.129 1.12e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.593 on 29 degrees of freedom
## Multiple R-squared:  0.8268, Adjusted R-squared:  0.8148
## F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

R for Regression: Considerations

- Don't always throw more variables at a problem. Add and remove them to see.
- Be wary of overfitting.
- Judge what should be used as a continuous variable v. discrete (factor) variable.
- Correlation v. causation: In the life expectancy example, does a specific region **cause** longer/shorter life, or are there other hidden common factors?