MQIM R BOOTCAMP

Financial Data & R Class, Object and functions

2023 August

Section 1

Bootcamp Week Schedule

Table of Content

Section 2

Getting Financial Data in R

Obtaining Financial Data in R

- Many R packages have built in functionality for downloading financial data from either free sources (Yahoo!Finance, etc.) or commercial vendors (Bloomberg or FactSet, for example).
- We'll primarily use free data sources in this class (or I will provide data sets when free sources won't cover our needs) but for those with Bloomberg access here at UNB or at work, the packages *Rbbg* and *Rblpapi* can be very useful.
- Note that Bloomberg does not suport either of these packages - they appear to be OK with users accessing the API in this way, but will of course offer absolutely no support for Rblpapi users experiencing issues. Do NOT ask the Bloomberg Help Desk for assistance with Rblpapi problems.

Free Sources

- In general, Yahoo!Finance is a pretty good source of equity pricing data, and Quandl is a good source of pricing and fundamentals (both free and non free) for equities and other securities, as well as some economic data.
- Oanda has some FX data, and Google Finance can also be used to obtain some equity pricing.
- The packages we'll use for accessing these sources are *tseries*, *quantmod*, *Quandl*, in addition to *Rblpapi* for Bloomberg data.

tseries

The *tseries* package can download data from Yahoo!Finance or Oanda:

```
> library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

The function *get.hist.quote()* (refer to *?get.hist.quote* for usage details) can download data into a time series object in R - lets get a few years of S&P 500 prices from Yahoo!Finance:

tseries

```
> get.hist.quote("^GSPC","2019-12-31","2020-10-31",
+ provider="yahoo",retclass="zoo")
```

```
## time series ends   2020-10-30

##               Open    High     Low   Close
## 2019-12-31 3215.18 3231.72 3212.03 3230.78
## 2020-01-02 3244.67 3258.14 3235.53 3257.85
## 2020-01-03 3226.36 3246.15 3222.34 3234.85
## 2020-01-06 3217.55 3246.84 3214.64 3246.28
## 2020-01-07 3241.86 3244.91 3232.43 3237.18
## 2020-01-08 3238.59 3267.07 3236.67 3253.05
## 2020-01-09 3266.03 3275.58 3263.67 3274.70
## 2020-01-10 3281.81 3282.99 3260.86 3265.35
## 2020-01-13 3271.13 3288.13 3268.43 3288.13
## 2020-01-14 3285.35 3294.25 3277.19 3283.15
## 2020-01-15 3282.27 3298.66 3280.69 3289.29
## 2020-01-16 3302.97 3317.11 3302.82 3316.81
```

quantmod

quantmod is a package for quantitative modeling of financial data and includes a variety of functions to obtain, process, and plot financial time series from multiple sources. Let's use the *getSymbols* function in the *quantmod* package to get the past 5 years of USDCAD exchange rates:

```
> library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 4.1.3

## Warning: package 'TTR' was built under R version 4.1.2
```

```
> USDCAD <- getSymbols("CAD=X",src = "yahoo",auto.assign = F)
```

```
## Warning: CAD=X contains missing values. Some functions will not w
## contain missing values in the middle of the series. Consider usin
## na.approx(), na.fill(), etc to remove or replace them.
```

```
> tail(USDCAD,2) # show final two data points
```

quantmod

```
> library(quantmod)
> tail(USDCAD,2) # show final two data points
```

```
##            CAD=X.Open CAD=X.High CAD=X.Low CAD=X.Close CAD=X.Volu
## 2023-08-25    1.35868    1.36392    1.3568     1.35868
## 2023-08-26    1.35784    1.36400    1.3565     1.35960
##            CAD=X.Adjusted
## 2023-08-25        1.35868
## 2023-08-26        1.35960
```

Quandl

Quandl has become one of the most full-featured sources of free financial data on the web (www.quandl.com) and have been offering non-free premium data as well. R package titled *Quandl*:

```
> library(Quandl)
```

For limited usage (less than 50 calls per day), just install the package and use it. If you plan to download larger amounts of data, you will need to register for an authentication token (free) and register that in your R session.

Quandl

The official Quandl documentation has the following example to download a time series of oil prices from the NYSE and save it to an *xts* object:

```
> mytimeseries <- Quandl("FRED/GDP", type="xts")
> tail(mytimeseries,5)
```

```
> # Quandl.api_key("NfEvd1uysf11ToVR7dbh")
> # mytimeseries <- Quandl("FRED/GDP")
> # head(mytimeseries,5)
```

Rblpapi

Rblpapi uses the Bloomberg API to allow easy importing of Bloomberg data into R. Load the package with the *library()* command and connect to the Bloomberg API (on a terminal computer) with the *blpConnect* command:

```
> library(Rblpapi)
```

Essentially all of the API functionality (whatever you're familiar with from the Bloomberg Excel addin) should be available in Rblpapi - including *bdp*, *bds*, *bdh* functions. Start a connection to the API with

```
> con <- blpConnect()
```

Rblpapi - bdp()

Get current data points (prices, fundamentals) for one or more tickers with *bdp()*:

```
> tickers <- c("RY CN Equity","TD CN Equity","CM CN Equity","BNS CN
> data.items <- c("PX_LAST","DIVIDEND_YIELD","EQY_BETA")
> bdp(tickers,data.items)
```

```
##                PX_LAST DIVIDEND_YIELD  EQY_BETA
## RY CN Equity    121.02       4.354652 0.8805032
## TD CN Equity     80.37       4.690805 0.8803391
## CM CN Equity     53.98       6.224528 0.9440815
## BNS CN Equity    62.12       6.632325 0.9080062
```

Rblpapi - bdh()

Get historical data with *bdh()*:

```
> # historial prices for the XIU etf since Dec 31, 2019
> my.data <- bdh("XIU CN Equity","PX_LAST",
+               start.date=as.Date("2019-12-31"),
+               end.date=as.Date("2020-11-30"))
> head(my.data)
```

```
##          date PX_LAST
## 1 2019-12-31 25.5600
## 2 2020-01-02 25.6300
## 3 2020-01-03 25.5600
## 4 2020-01-06 25.6508
## 5 2020-01-07 25.7100
## 6 2020-01-08 25.7680
```

Rblpapi - bds()

Download bulk data with *bds()*:

```
> # current constituents of the s&p/tsx 60 index
> tsx.memb <- bds("SPTSX INDEX","INDX_MEMBERS")
> head(tsx.memb)
```

```
##   Member Ticker and Exchange Code
## 1                          AAV CT
## 2                          ABX CT
## 3                           AC CT
## 4                        ACO/X CT
## 5                          AEM CT
## 6                          AGI CT
```

Rblpapi

Other functionality exists, including the *getBars()* and *getTicks* functions, as well as a function for searching available fields (*fieldSearch*):

```
> search.res <- fieldSearch("volatility")
> head(search.res, 2)
```

```
##       Id       Mnemonic       Description
## 1 RK002 VOLATILITY_30D Volatility 30 Day
## 2 RK004 VOLATILITY_90D Volatility 90 Day
```

```
> dim(search.res)
```

```
## [1] 973   3
```

Note that *fieldSearch()* returns a data frame object.

Rblpapi

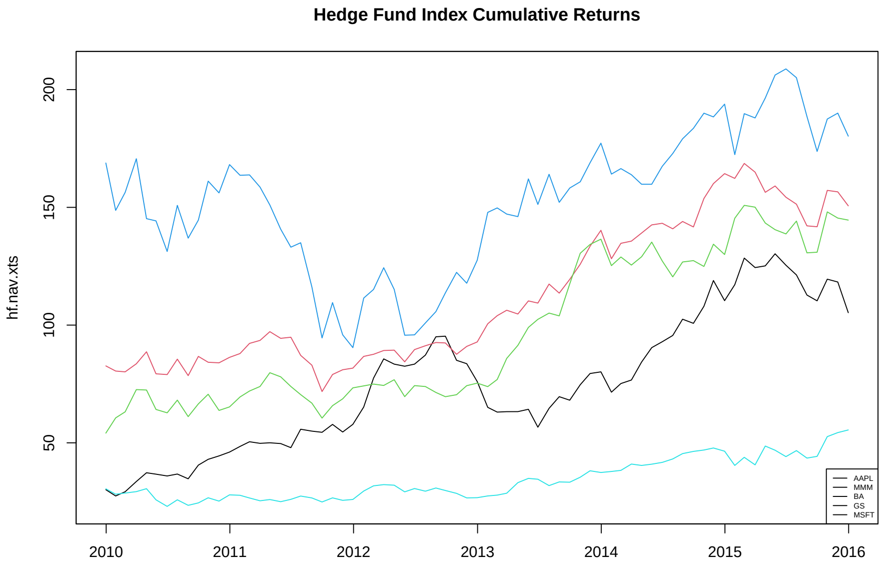
Don't forget to close the connection

```
> blpDisconnect(con)
```

Importing Data from Files

```
> hf.nav <- read.table(file = "data/stock_data.csv",
+                      header=T,sep=",",
+                      stringsAsFactors=FALSE)
> hf.nav.xts <- xts(hf.nav[,-1],
+                   order.by=as.Date(hf.nav[,1],
+                                    format="%Y-%m-%d"))
> plot.zoo(hf.nav.xts,plot.type='single',
+          lty=1,main="Hedge Fund Index Cumulative Returns",
+          col=seq(1:ncol(hf.nav.xts)))
> legend("bottomright", colnames(hf.nav.xts), lty = 1, cex = 0.5)
```

Importing Data from Files



Recap

- R has easy and convenient methods for importing data from both free and non-free online sources
- We can also import files (and interact with databases) to import data that we already own
- Many packages are available to make this process easier - *quantmod* is one of the originals and is very full featured, while *Quandl* allows access to Quandl data sets
- Commercial vendors often provide unofficial support (Bloomberg) or official support (FactSet) for development of R packages for subscribers to access their data

Section 3

R Class, Object and functions

R Objects (frequently used)

In R, all types of data are treated as objects.

- Vectors

Type	Example
Doubles	die <- c(1:6)
Characters	text <- c('R', 'Workshop')
Logicals	logic <- c(TRUE, FALSE, TRUE)

- Matrices

```
> die <- c(1:6)
> mtx <- matrix(die, nrow =2, byrow = TRUE)
```

- Arrarys

```
> ary <- array(mtx, dim=c(2,3,3))
```

R Objects (frequently used)

- Lists

```
> list <- list(die, mtx, ary)
```

- DataFrame is the two-dimensional version of a list. It is a very useful storage structure for data analysis. You can think of a dataframe as R's equivalent to the Excel spreadsheet.

```
> df <- data.frame(face = c("ace","king","queen"),
+                  suit = c("heart","spades","diamonds"),
+                  value = c(1,13,12))
```

R Class

However, objects are not simply collections of data. An object is a data structure having some attributes and methods which act on its attributes.

Class is a blueprint for the object. We can think of class like a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house.

Let's build a simple slot machine, with the *play* class with *symbols* and *score* methods, that you can play by running the class. When you're finished, you'll be able to play it

like this:

```
## [1] "0"   "B"   "BBB"
```

```
> play()
```

```
## [1] "BB" "7"  "0"
```

```
## [1] 0
```

R Class

The **play** function will need to do two things. First, it will need to randomly generate three symbols; and, second, it wil need to calculate a prize based on those symbols.

R function

1. Define your slot machine symbols and then randomly generate three symbols with the **sample** function.

The wheel, or all the possible outcomes are DD, 7,BBB,BB,B,0, now please take a few minute to think about how to write this *get_symbols* function to randomly select 3 symbols from the wheel.

steps: 1. set up the general function format for *get_symbols* 2. need a selection funcion for the output

R function

1. Define your slot machine symbols and then randomly generate three symbols with the **sample** function.

```
> get_symbols <- function(){
+   wheel <- c("DD","7","BBB","BB","B","0")
+   sample(wheel, size = 3, replace = TRUE,
+          prob = c(0.03, 0.03, 0.06 ,0.1, 0.27, 0.51))
+ }
> get_symbols()
```

```
## [1] "B" "0" "0"
```

R function

2. Write a program that can take the output of get_symbols and calculate the correct prize based on the prize rule, the payout scheme is set as:

Combination	Prize
DD DD DD	100
7 7 7	80
BBB BBB BBB	40
BB BB BB	25
B B B	10
Any combo of bars	5

We call this function **score**, such that *score(c("DD","DD","DD"))* will give you $100.

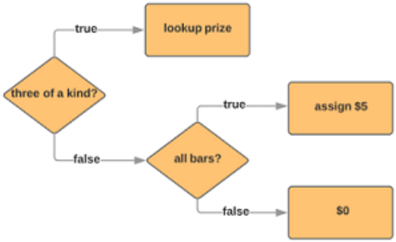
R function

3. Put them together to create the full slot machine, like:

```
> play <- function(){
+   symbols <- get_symbols()
+   print(symbols)
+   score(symbols)
+ }
```

R function

Now let's write the score function, the flow chart below describes the payout scenarios, diamond shape symbolize an *if else* decision.



R function

Please take a few minutes to think about how to write this *score* function.

steps: 1. general function format setup 2. identify case (ifelse statement and test for equality) 3. assign prize for each case

R function

```
> score <- function(symbols){
+   # identify case
+   same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]
+   bars <- symbols %in% c("B","BB","BBB")
+   # get prize
+   if(same){
+     payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25,
+                  "B" = 10, "C" = 10, "0" = 0)
+     prize <- unname(payouts[symbols[1]])
+   } else if(all(bars)){
+     prize <- 5
+   } else {
+     prize <- 0
+   }
+   # return prize
+   prize <- paste0("You Win $", prize)
+   return(prize)
+ }
```

R function

Once the **score** function is defined, the **play** class will work as well:

```
> play <- function(){
+   symbols <- get_symbols()
+   print(symbols)
+   score(symbols)
+ }
> play()
```

```
## [1] "0"  "B"  "DD"
```

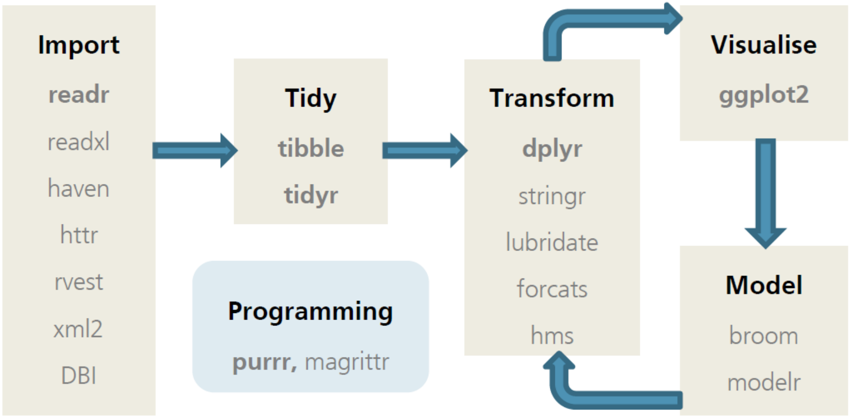
```
## [1] "You Win $0"
```

Section 4

Data manipulation and Visualization

Introduction to *tidyverse*

The *tidyverse* is a set of packages for doing data science, it loads six "core" libraries that provide tools for importing(*readr*), tidying(*tidyr*,*tibble*), manipulation(*dplyr*) and visualizing(*ggplot*) data, as well as support for functional programming(*purrr*).



Data Manipulation (*dplyr*)

key verbs:

- **select** for subsetting variables (columns)
- **mutate** for creating new variables from existing ones
- **filter** for subsetting observations (rows)
- **arrange** for rearranging/ordering observations
- **summarise** for computing various summary statistics

Along with the **group_by**, combining multiple simple pieces with the pipe operator %>% is a powerful way of solving complex problems.

Let's see a simple example by loading *data_df.RData*, which contains information about 40 large-cap companies.

```
> library(tidyverse)
> load("data/data_df.RData")
```

Data Manipulation (*dplyr*)

Suppose we have the following task:

Calculate the mean ROE by sector and 12-month momentum (positive or negative), making sure that there are at least 3 observations per group.

Finally, arrange the data in descending order of ROE.

Data Manipulation (*dplyr*)

What would you do it without advanced package except base R?

```
> data_df$Momentum <- ifelse(data_df$Return12m > 0, "Positive",
+                                 "Negative")
> data_df <- transform(data_df, freq = as.numeric(as.character(
+ ave(Sector, Sector, Momentum, FUN = length))))
> data_df_filtered <- subset(data_df, freq >= 3)
> mean_roe <- aggregate(ROE ~ Sector + Momentum,
+                          data = data_df_filtered,
+ FUN = mean, na.rm = T)
> mean_roe <- mean_roe[order(mean_roe$ROE, decreasing = TRUE), ]
> head(mean_roe,3)
```

```
##                   Sector Momentum      ROE
## 3       Consumer Staples Positive 59.76600
## 2 Consumer Discretionary Positive 39.30333
## 5 Information Technology Positive 25.20818
```

Data Manipulation(*dplyr*)

It's cleaner and straight forward by using *dplyr*

```
> mean_roe <- data_df %>%
+ mutate(Momentum = ifelse(Return12m > 0, "Positive",
+                               "Negative")) %>%
+ group_by(Sector, Momentum) %>%
+ filter(n() >= 3) %>% # n() returns row count in each group
+ summarise(ROE = mean(ROE, na.rm = TRUE)) %>%
+ arrange(desc(ROE))
```

Data Manipulation(*dplyr*)

```
> head(mean_roe,3)
```

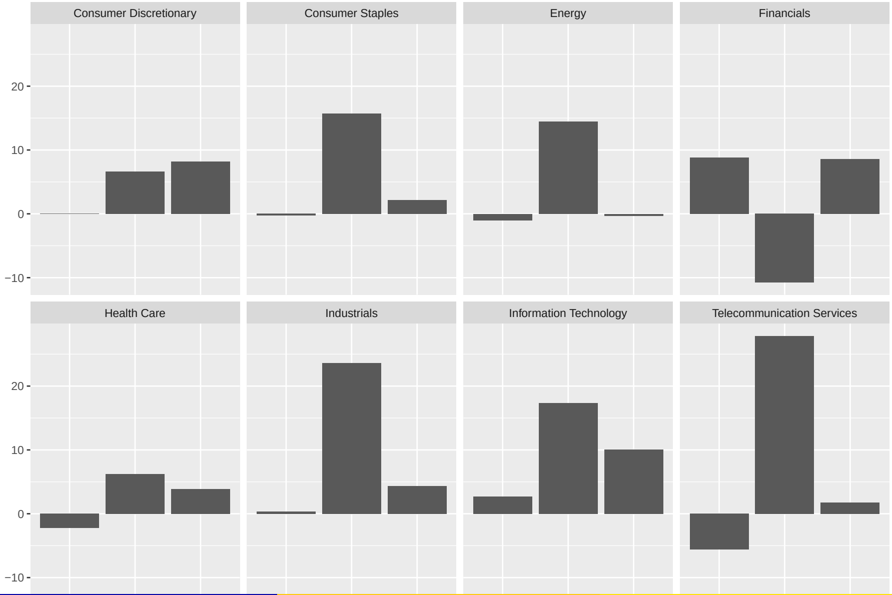
```
## # A tibble: 3 x 3
## # Groups:   Sector [3]
##   Sector                 Momentum   ROE
##   <chr>                  <chr>    <dbl>
## 1 Consumer Staples       Positive  59.8
## 2 Consumer Discretionary Positive  39.3
## 3 Information Technology Positive  25.2
```

Data Visualization(*ggplot*)

Charts are great tools to help you understand the data from the initial data research. If you don't want to spend too much time writing your own data manipulating and plotting functions, there is a nice and quick way to do it by combining *dplyer* and *ggplot*. Here is an example to visualize the sector mean returns of the data:

```
> myplot <- data_df %>%
+   group_by(Sector) %>%
+   summarise_at(vars(contains("Return")), funs(mean)) %>%
+   tidyr::gather(Variable, Return, -Sector) %>%
+   ggplot(aes(x = Variable, y = Return)) +
+   geom_bar(stat = "identity") + ylab("") +
+   facet_wrap(~ Sector, ncol = 4)
```

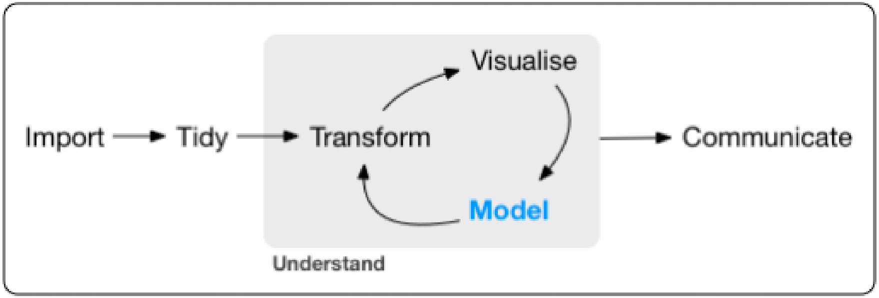
Data Visualization (*ggplot*)



Section 5

Model Building

Model Basics with *modelr*



Model Basics with *modelr*

The goal of a model is to provide a simple low-dimensional summary of a dataset. Ideally, the model will capture true “signals” (i.e., patterns generated by the phenomenon of interest), and ignore “noise” (random variation that you’re not interested in).

There are two parts to a model:

1. Define a *family of models* that express generic pattern that you want to capture. For example, the pattern might be a straight line, or a quadratic curve. You express the model family as an equation like $y = a_1 x + a_2$ or $y = a_1 x^{a_2}$. Here, x and y are known variables from your data, and a_1 and a_2 are parameters that can vary to capture different patterns.
2. Generate a *fitted model* by finding the model from the family that is the closet to your data. this takes the generic model family and makes it specific, like $y = 2x + 8$ or $y = 5x^2$

Note: fitted model only implies that you have the “closest” model based on some criteria, doesn’t imply it’s a true/good model.

Model Basics with *modelr*

Let's begin with a simple model.

load *modelr* package

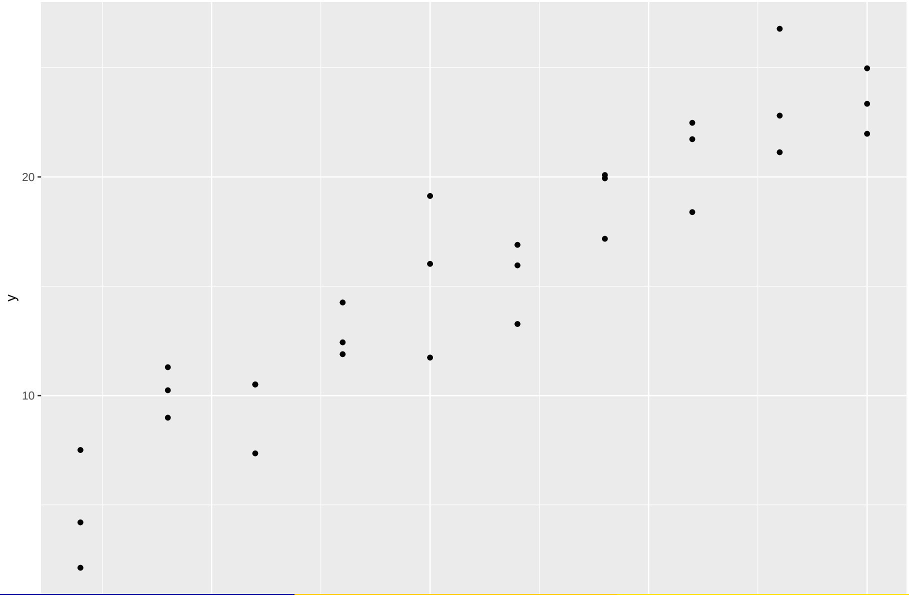
```
> library(tidyverse)
> library(modelr)
```

Model Basics with *modelr*

The simulated dataset *sim1*.

```
> ggplot(sim1, aes(x,y)) +
+   geom_point()
```

Model Basics with *modelr*



Model Basics with *modelr*

You can see a strong pattern in the data. Let's use a model to capture that pattern and make it explicit. It's our job to supply the basic form of the model. In this case, the relationship looks linear, i.e., $y = a_1 + a_2 * x$.

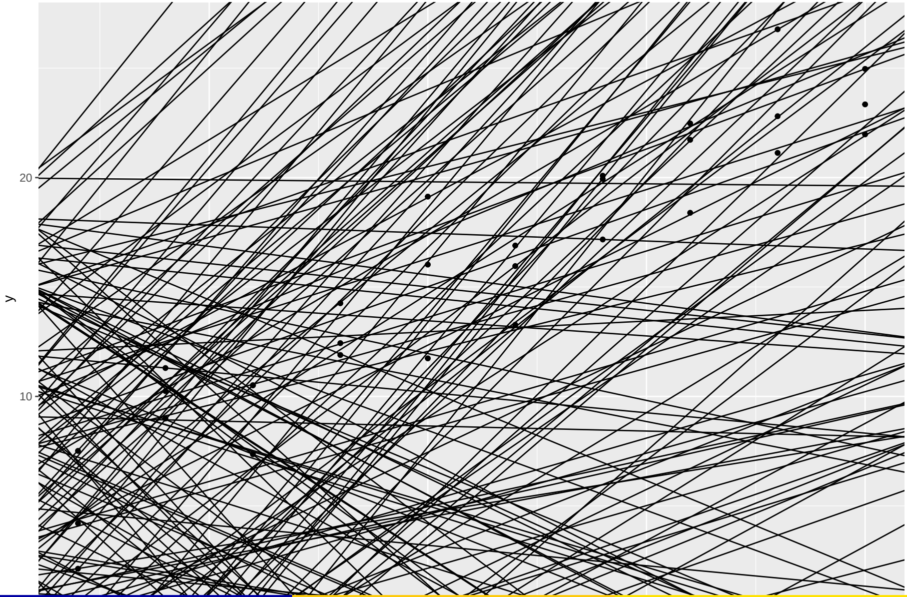
To see what models from that family look like, we can start to generate some random data. Then use the *geom_abline()*, which takes a slope and intercept as parameters.

```
> models <- tibble(
+   a1 = runif(250,-20,20),
+   a2 = runif(250,-5,5)
+ )
```

Model Basics with *modelr*

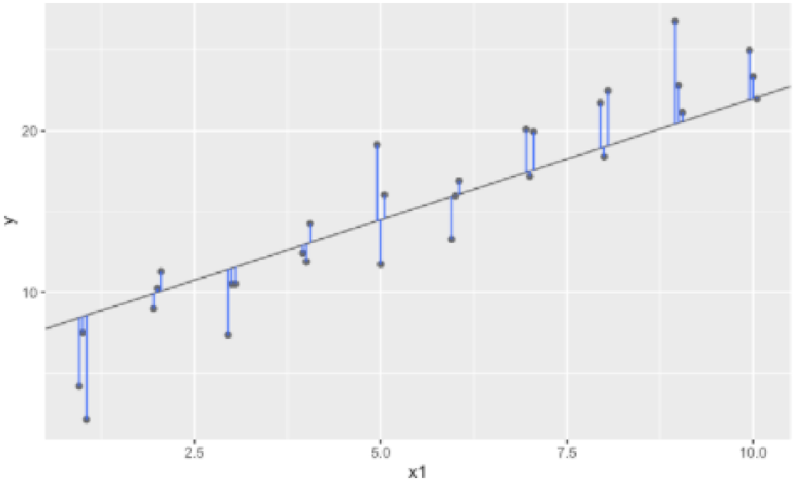
```
> ggplot(sim1, aes(x,y)) +
+   geom_abline(
+     aes(intercept = a1, slope = a2),
+     data = models) +
+   geom_point()
```

Model Basics with *modelr*



Model Basics with *modelr*

There are 250 models on this plot. We need a way to quantify the distance between the data and a model. Then we can fit the model by finding the values of a_1 and a_2 that generate the model with the smallest distance from this data.



Model Basics with *modelr*

One easy place to start is to find the vertical distance between the y value given by the model (the *prediction*), and the actual y value in the data (the *response*).

To compute this distance, need to turn the model family into an R function. This takes the model parameters and the data as inputs, and gives values predicted by the model as output:

```
> model1 <- function(a,data){
+   a[1] + data$x *a[2]
+ }
> model1(c(8,2), sim1)
```

```
##  [1] 10 10 10 12 12 12 14 14 14 16 16 16 18 18 18 20 20 20 22 22
## [26] 26 26 28 28 28
```

Model Basics with *modelr*

```
> meansure_dist <- function(mod, data){
+   diff <- data$y -  model1(mod, data)
+   sqrt(mean(diff^2))
+ }
> meansure_dist(c(8,2), sim1)
```

```
## [1] 4.095278
```

Model Basics with *modelr*

Now if we compute the distance for all the models defined previously.

```
> sim1_dist <- function(a1,a2){
+   meansure_dist(c(a1,a2), sim1)
+ }
>
> models <- models %>%
+   mutate(dist = purrr::map2_dbl(a1,a2,sim1_dist))
```

Model Basics with *modelr*

```
> head(models)
```

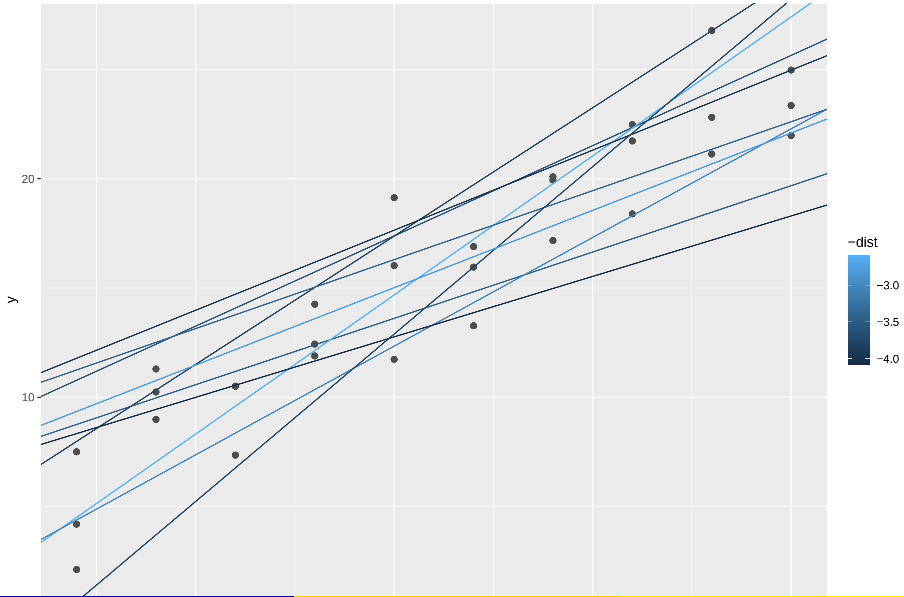
```
## # A tibble: 6 x 3
##        a1      a2  dist
##     <dbl>   <dbl> <dbl>
## 1 -18.6  -3.00   52.7
## 2 -12.3   0.0858 28.0
## 3   8.46 -1.24   16.9
## 4 -14.1   4.92    8.88
## 5   1.09 -0.211  17.0
## 6   9.11 -0.0929  9.49
```

Model Basics with *modelr*

Let's overlay the 10 best models on the the data, and visualize it. The ones with the smallest distance models get the brightest colors:

```
> ggplot(sim1, aes(x,y)) +
+   geom_point(size = 2, color = "grey30") +
+   geom_abline(
+     aes(intercept = a1, slope = a2, color = -dist),
+     data = filter(models, rank(dist) <= 10)
+   )
```

Model Basics with *modelr*

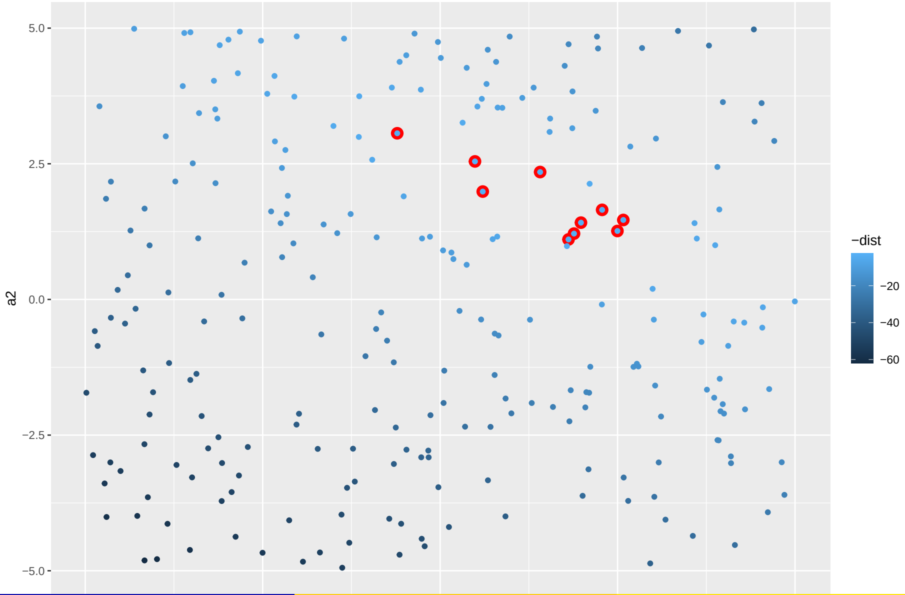


Model Basics with *modelr*

We can also think about these models as observations and visualize them with a scatterplot of a1 vs a2.

```
> ggplot(models, aes(a1,a2)) +
+   geom_point(
+     data = filter(models, rank(dist) <= 10),
+     size = 4, color = "red"
+   ) +
+   geom_point(
+     aes(color = -dist)
+   )
```

Model Basics with *modelr*



Model Basics with *modelr*

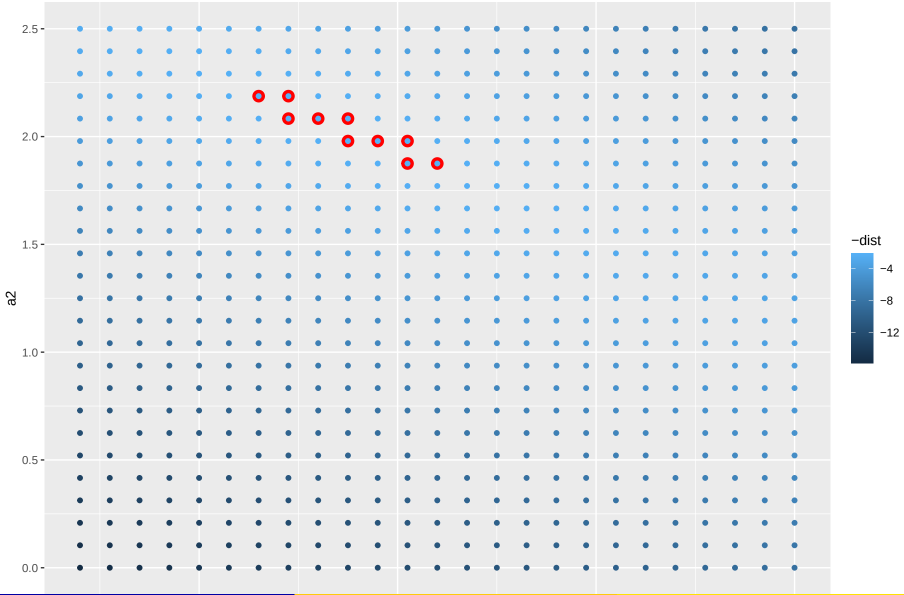
Instead of trying lots of random models, we could be more systematic and generate an evenly spaced grid of points (grid search).

```
> grid <- expand.grid(
+   a1 = seq(1,10, length = 25),
+   a2 = seq(0,2.5, length = 25)) %>%
+   mutate(dist = purrr::map2_dbl(a1,a2,sim1_dist))
```

Model Basics with *modelr*

```
> grid %>%
+   ggplot(aes(a1,a2))+
+   geom_point(
+     data = filter(grid, rank(dist) <=10),
+     size = 4, color = "red"
+   ) +
+   geom_point(aes(color = -dist))
```

Model Basics with *modelr*

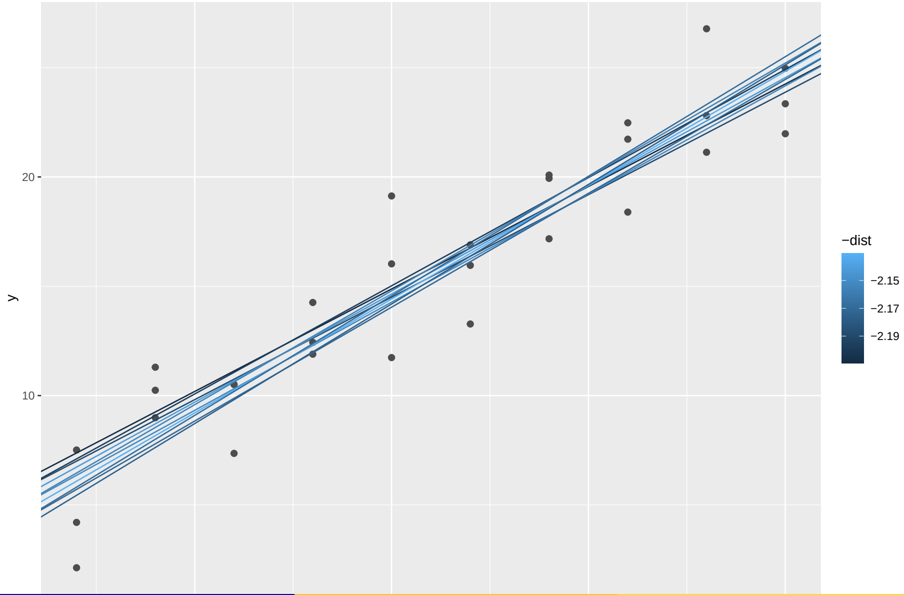


Model Basics with *modelr*

When you overlay the best 10 models back on the original data, they all look pretty good:

```
> ggplot(sim1, aes(x,y))+
+   geom_point(size = 2, color = "grey30") +
+   geom_abline(
+     aes(intercept = a1, slope = a2, color = -dist),
+     data = filter(grid, rank(dist) <= 10)
+   )
```

Model Basics with *modelr*



Model Basics with *modelr*

You could imagine iteratively making the grid finer and finer until you narrowed in on the best model.

But there are other numerical minimization tools to solve the problem. You can pick a starting point and look around for the steepest slope, then ski down that slope a little way, then repeat again and again, until no way down further. In R, we can do that with *optim()*.

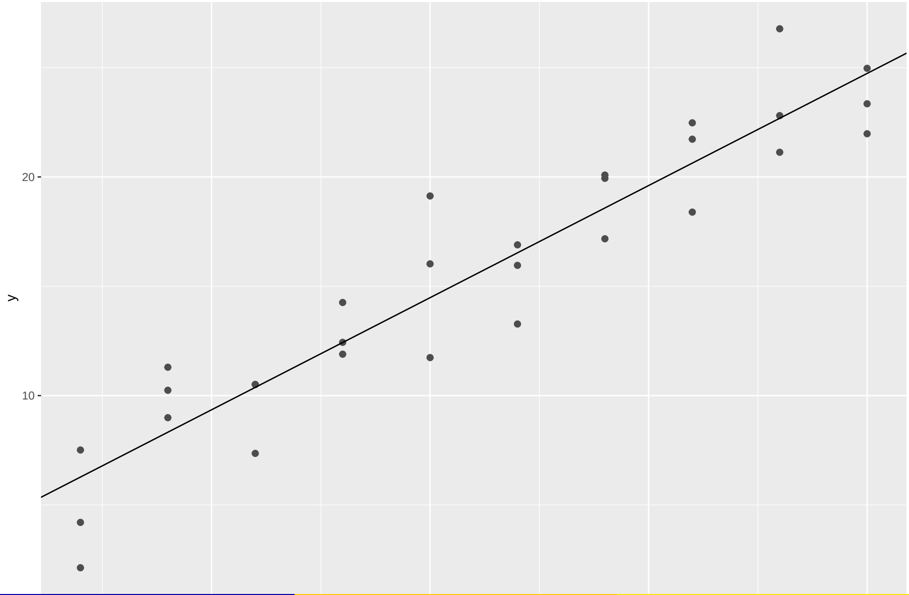
```
> best_guess <- optim(c(0,0), meansure_dist, data = sim1)
> best_guess$par
```

```
## [1] 4.222248 2.051204
```

Model Basics with *modelr*

```
> ggplot(sim1,aes(x,y))+
+   geom_point(size = 2, color ="grey30") +
+ geom_abline(intercept = best_guess$par[1],
+             slope = best_guess$par[2])
```

Model Basics with *modelr*



Model Basics with *modelr*

Linear model with *lm()*

```
> sim1_mod <- lm(y ~x , data = sim1)
> coef(sim1_mod)
```

```
## (Intercept)           x
##    4.220822    2.051533
```

Behind the scenes *lm()* doesn't use *optim()*, but instead takes advantage of the mathematic structure of linear models. *lm()* actually finds the closest model in a single step, the approach is faster and it guarantees that there is a global minimum.

Mode Building

Previously, we have focused on simulated datasets to help you learn about how models work, now we will focus on real data, how you can progresssively build up a model to help you better understand the data.

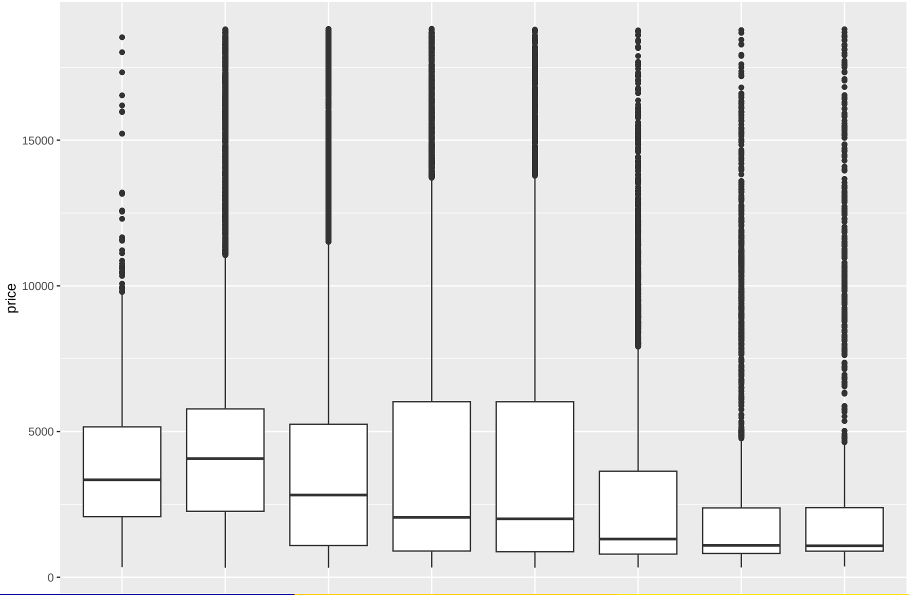
We will use the *diamonds* data set to investigate the relationship between diamonds' quality and it's price.

Mode Building

Firstly, visualize the relationship

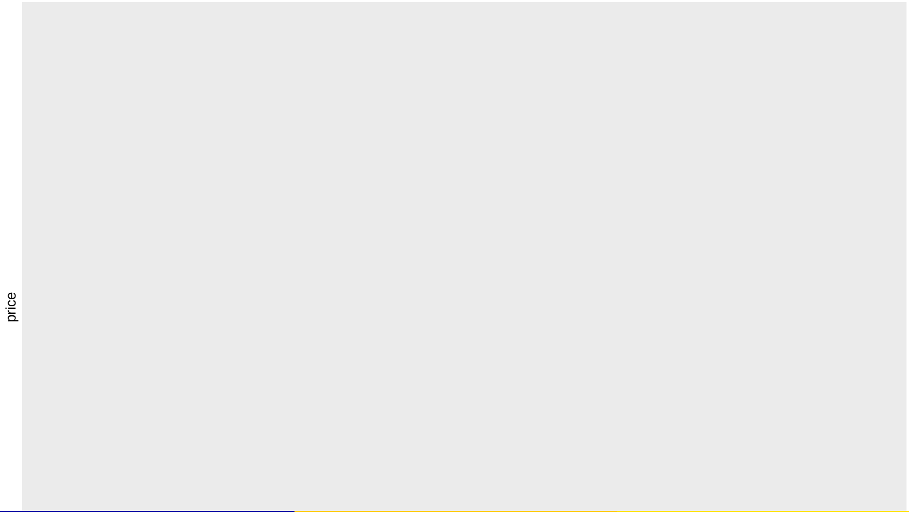
```
> ggplot(diamonds, aes(clarity, price)) +
+ geom_boxplot()
```

Mode Building



Mode Building

It looks like lower-quality diamonds have higher prices because there is an important confounding variable: the weight (*carat*) of the diamond.

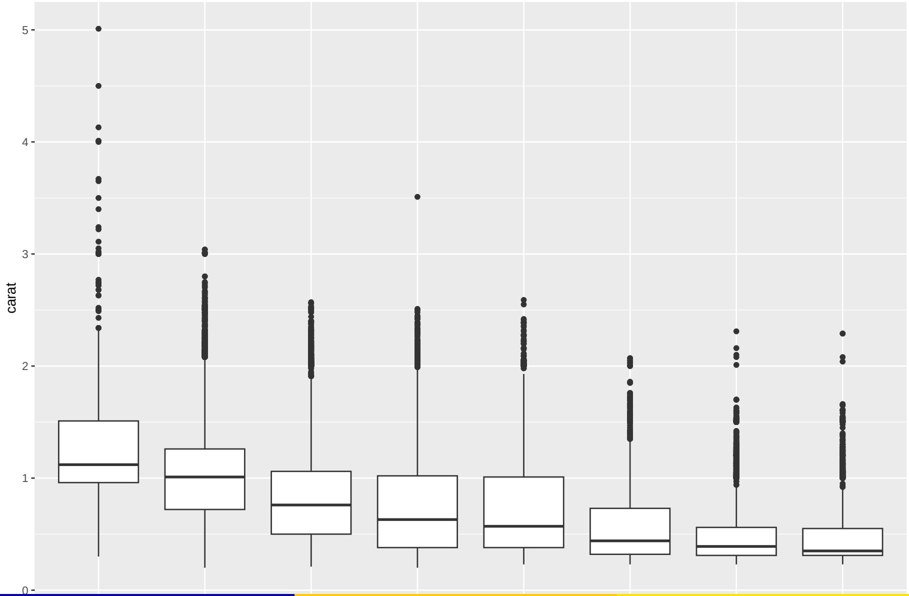


Mode Building

The weight of the diamond is the single most important factor for determining the price of the diamond, and lower quality diamonds tent to be larger.

```
> ggplot(diamonds, aes(clarity, carat)) +
+ geom_boxplot()
```

Mode Building



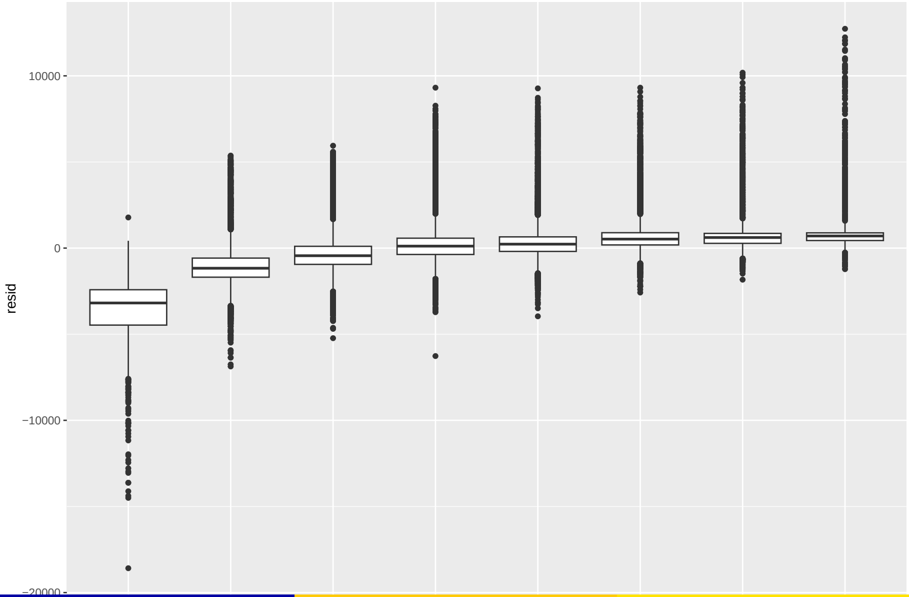
Mode Building

We need to remove the “carat effect” to see how the *clarity* or other attributes of a diamond affect its relative price.

Firstly, we fit a linear model with *price* to *carat* to get the residuals, or get the separate out the effect of *carat*.

```
> mod_diamond <- lm(price ~ carat, data = diamonds)
>
> diamonds2 <- diamonds %>%
+   add_residuals(mod_diamond, "resid")
```

Mode Building



Section 6

R Markdown, R Projects and Github

R Markdown

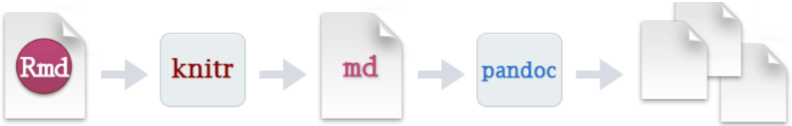
Why R Markdown?

- R Markdown provides various great formats for communicating, presenting as well as publishing rearch results.
- It enables the final report to include code chunks and output (table, plot etc.) while executing the codes.
- In addition, it makes typing complicated formulas and equations much less painful.
- It also ensures reproducibility and consistency. You can keep your code, notes, graphs and relevant links all in one place.
- Of course, a great way to submit your assignment.

R Markdown

How it works?

- create *.Rmd* file, select *File > New File > R Markdown.* in the menubar. RStudio will launch a wizard that you can use to pre-populate your file with useful content that reminds you how the key features of R Markdown work.
- **knit** the document, R Markdown sends the .Rmd file to knitr, http://yihui.name/knitr/. R Markdown executes all of the code chunks and creates a new markdown (.md) document which includes the code and its output.
- The markdown file generated by *knitr* is then processed by **pandoc**, http://pandoc.org/. R Markdown is responsible for creating the final file.



R Markdown

R Markdown formats:

- Documents:
  - html
  - pdf
  - word
- Presentations:
  - ioslides - HTML presentation with ioslides
  - slidy - HTML presentation with W3C Slidy
  - beamer - PDF presentation with LaTeX Beamer.
  - ppt

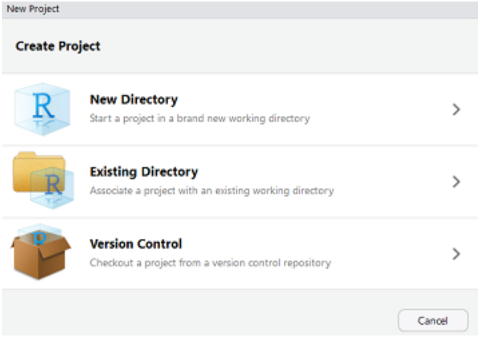
Rstudio Projects

It is a good practice to create a projects for an ongoing research process with kept developing codes. The beauty of using projects are:

- Holds all the files relevant to that particular piece of work in a folder in your computer in a desired project directory. - Set unite working directory to Project directory, save the trouble for typing *setwd("C:/Users/path)* and *rm(list = ls())* for each r script.
- Dedicated R process. File browser pointed at project directory.
- The way to create RStuodio Project is quite flexible, you can create it in a new folder, in a existing folder, or link it to a version control repository (we will talk about GitHub here).

Rstudio Projects

To create a new Rstudio Projects, go to *RStudio -> New Project*, you will see options:


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Click **New Directory** if you'd like to make a new folder as project, or **Existing Directory** to make existing folder into an RStudio Project.

GitHub

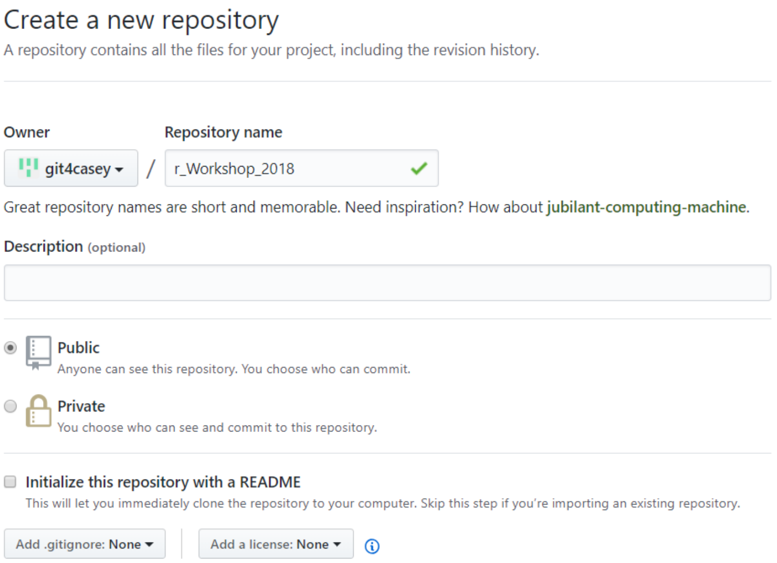
What is it?

- Git is an open-source version control system that was started by Linus Trovalds, whom created Linux.
- When developers create something (an app, for example), they make constant changes to the code, releasing new versions after the first official release.
- Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.
- People who have nothing to do with the development of a project can still download the files and use them.
- "Hub" part in GitHub is https://github.com/

GitHub

- Repository: a repository("repo") is a location where all the files for a particular project are stored. Each project has its own repo, and you can access it with a unique URL.
- Forking a Repo
  - create a new project based off of another project that already exists.
  - fork the repo that you'd like to contribute to, make the changes you like and release the revised project as a new repo.
- You can fork the **R_Workshop** repo from my github to yours to access some R files and data.

GitHub



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: git4casey / Repository name: r_Workshop_2018 ✓

Great repository names are short and memorable. Need inspiration? How about **jubilant-computing-machine**.

Description (optional)

[● Public] Anyone can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

☐ Initialize this repository with a README

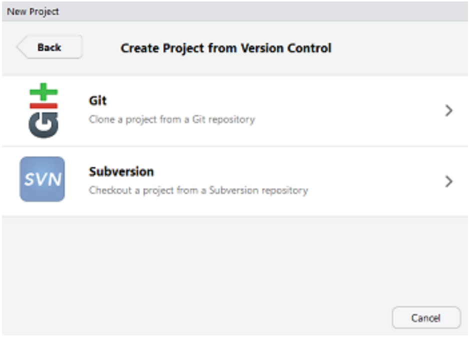
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Link Rstudio Project to Github

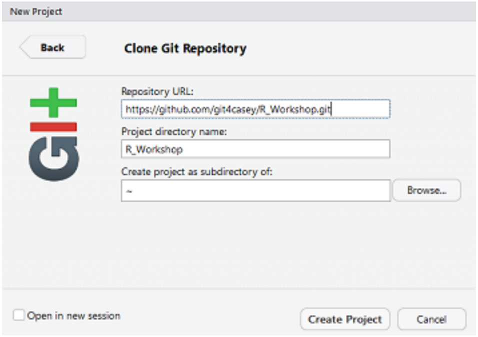
How to use it along with Rstudio Project?

- When creating the Rstudio Project, go to the third option **Version Control**


New Project
Back
Create Project from Version Control
Git
Clone a project from a Git repository
Subversion
Checkout a project from a Subversion repository
Cancel

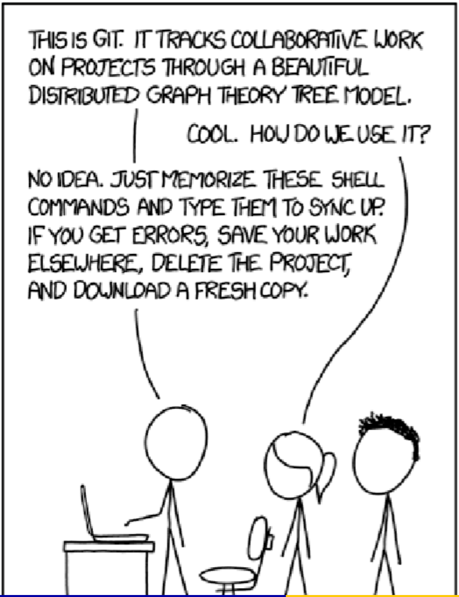
Link Rstudio Project to Github

- Select **Git**, and paste the repo URL you'd like to work on. For example, it can be the forked **R_Workshop** repo on your github.
- Select the preferred the directory in your computer by *Browse...*, then click *Create Project*. Now you should be able to see all the files in the repo from Rstudio, and edit and push to update your github repo.



Repository URL: https://github.com/git4casey/R_Workshop.git
Project directory name: R_Workshop
Create project as subdirectory of: ~

GitHub

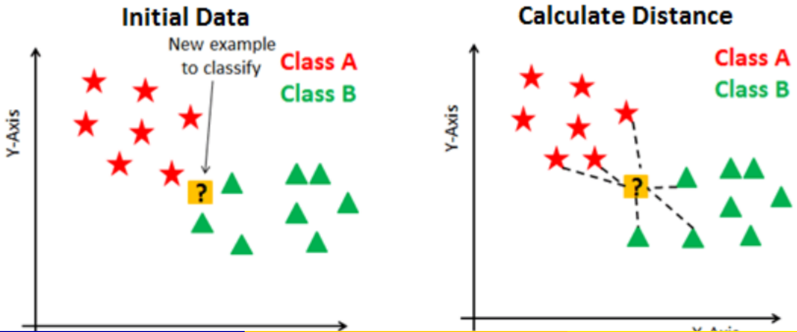

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Section 7

Basic Machine Learning, R for KNN

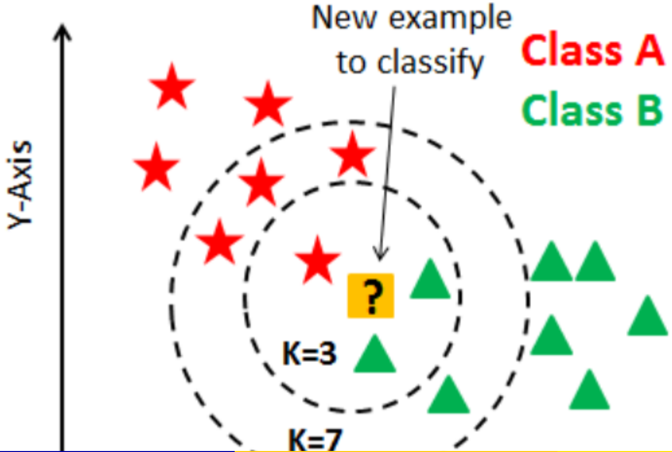
KNN concepts

- The KNN or k-nearest neighbors algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances.
- More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure.
- This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance.



KNN

The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. Look at the case below: The question is how to choose the optimal number of *k* neighbors?



KNN

No unified answer that suits all kind of data sets. In general, requires test on different k and validate the performance. Research has also shown that

- a small number of neighbors are most flexible fit which will have low bias but high variance
- a large number of neighbors will have a smoother decision boundary, which implies lower variance but higher bias.

KNN Iris Example in R

Let's use famous iris dataset to understand how knn works in R. To inspect the data, some simple summary:

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

KNN Iris Example in R

To visualize if there is any correlation between variables.

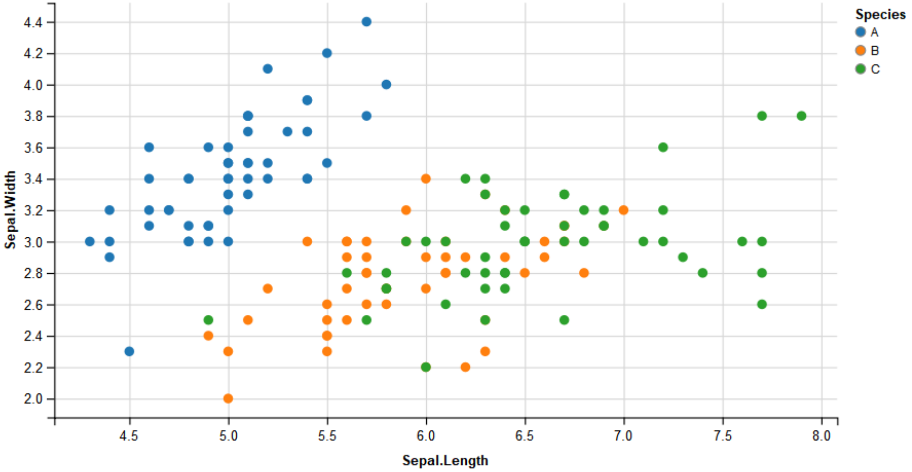


Figure 2: flowchart

KNN Iris Example in R

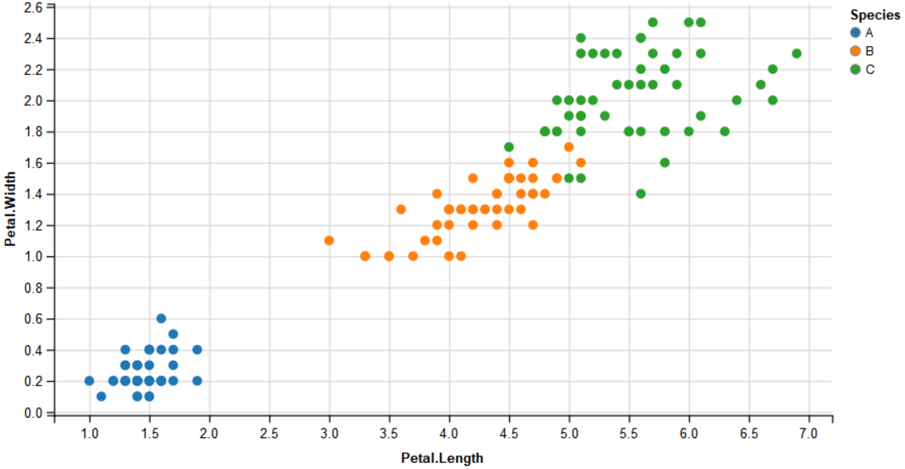


Figure 3: flowchart

KNN Iris Example in R

- Divide the data set into two parts: a training set and a test set. (2/3, 1/3 split)

```
> ind <- sample(2, nrow(iris_new), replace=TRUE, prob=c(0.67, 0.33))
> iris_training <- iris_new[ind==1, 1:4]
> iris_test <- iris_new[ind==2, 1:4]
> iris_trainLabels <- iris_new[ind==1,5]
> iris_testLabels <- iris_new[ind==2, 5]
```

- Using the knn() function, which uses the Euclidian distance measure in order to find the k-nearest neighbors to your new, unknown instance.

```
> library(class)
> iris_pred <- knn(train = iris_training, test = iris_test,
+                  cl = iris_trainLabels, k=3)
> iris_pred
```

```
##  [1] A A A A A A A A A A A A A A A A A A A A B B B B B B B B C C
## [39] B B B B C C C C C C C C
```

KNN Iris Example in R

- Compare the predicted class to the test labels.

```
##   test_lable pred_lable diff
## 1          A          A    1
## 2          A          A    1
## 3          A          A    1
## 4          A          A    1
## 5          A          A    1
## 6          A          A    1
```

```
## [1] "Accuracy: 94%"
```

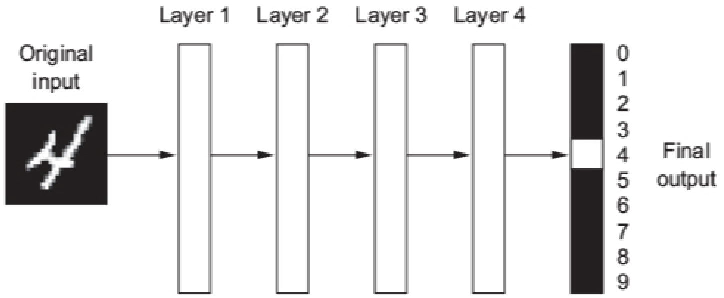
Section 8

Introduction to deep learning (keras library)

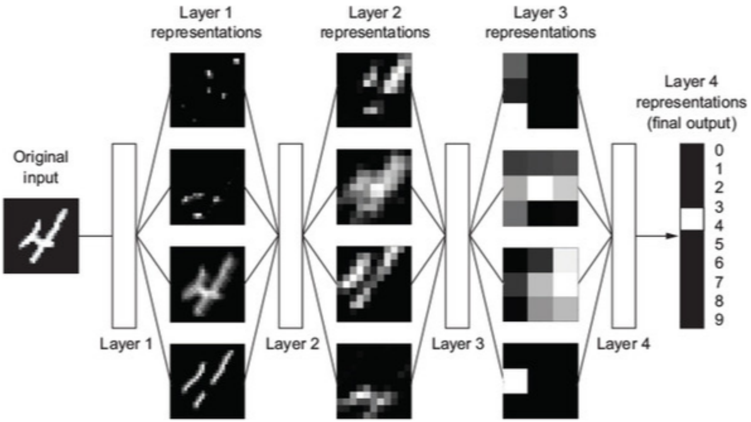
What's the "deep"

- Emphasize on learning through successive *layers* of increasingly meaningful representations.
- these layered representations are (mostly) learned via models called *neural networks*, it structured in layers stacked on top of each other

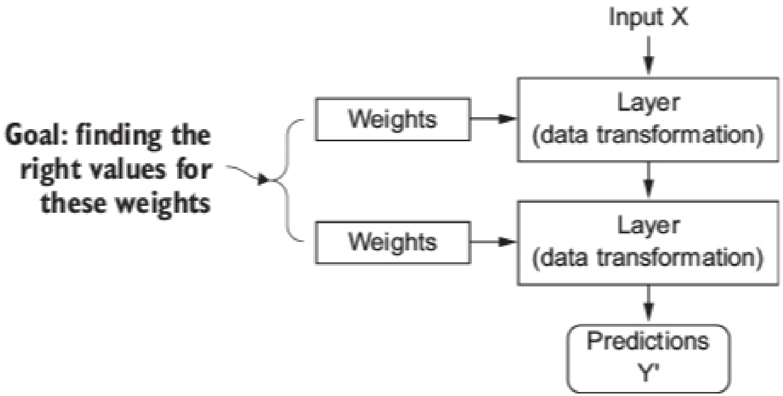
Layers representation



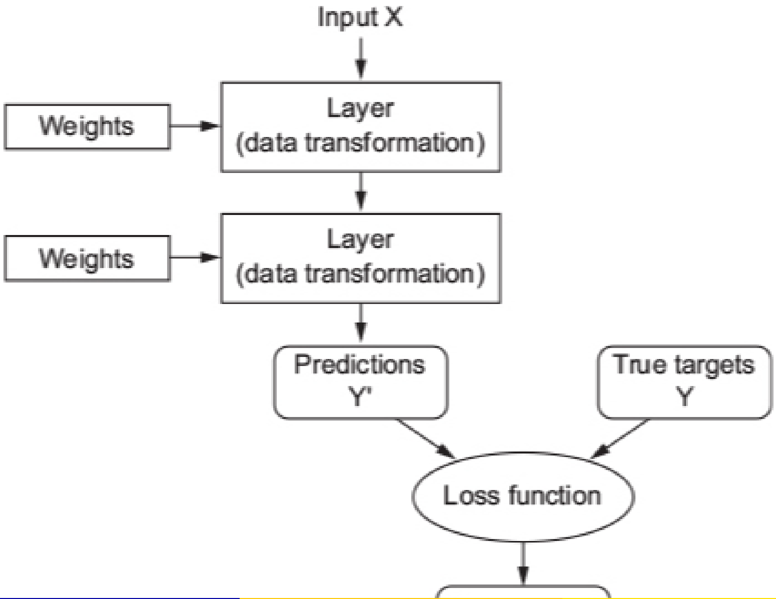
Layers representation



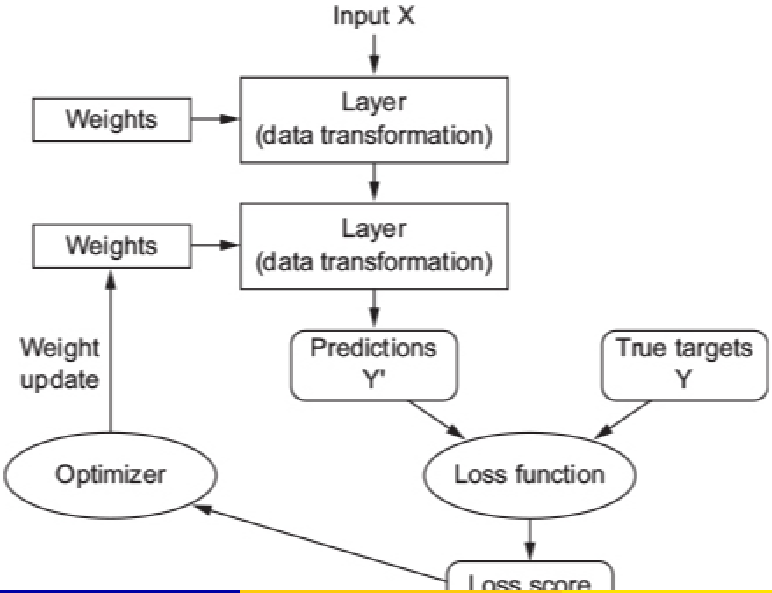
The workflow


Input X
Weights
Layer (data transformation)
Weights
Layer (data transformation)
Predictions Y'
Goal: finding the right values for these weights

The workflow



The workflow



Data structure

In general, all current machine-learning systems use multidimensional arrays as their basic data structure, it also called *tensors*.

Tensors are a generalization of vectors and matrices to an arbitrary number of dimensions.

Tensors

- 0D tensors(Scalars)

A tensor that contains only one number is called a *scalar* or *zero-dimensional tensor*. R doesn't have a *scalar* data type, but you can think R vector with length 1 is conceptually a scalar.

- 1D tensors(Vectors)
- 2D tensors(Matrices)
- 3D tensors and higher-dimensional tensors

Tensors

If you pack 2D tensors(matrices) in a new array, you will get a 3D tensor.

```
> x <- array(rep(0,2*3*2), dim = c(2,3,2))
> dim(x)
```

```
## [1] 2 3 2
```

By packing 3D tensors in an array, u can create a 4D tensor, and so on. In deep learning, you'll generally manipulate tensors up to 5D.

MNIST dataset

Let's look at a concrete example of a neural network that uses the Keras R package to learn to classify handwritten digits. The MNIST dataset comes preloaded in Keras, in the form of *train* and *test* lists, each of which includes a set of images (x) and associated labels (y).

```
> library(keras)
> library(tensorflow)
> #install_tensorflow()
> #mnist <- dataset_mnist()
>
> load("data/mnist.RData")
```

MNIST dataset

The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 X 28 pixels) into their 10 categories ($0 \sim 9$).

```
> train_images <- mnist$train$x
> train_labels <- mnist$train$y
> test_images <- mnist$test$x
> test_labels <- mnist$test$y
```

MNIST dataset

The network architecture

```
> network <- keras_model_sequential() %>%
+   layer_dense(units = 512, activation = "relu",
+               input_shape = c(28 * 28)) %>%
+   layer_dense(units = 10, activation = "softmax")
```

MNIST dataset

The compilation step

```
> network %>% compile(
+   optimizer = "rmsprop",
+   loss = "categorical_crossentropy",
+   metrics = c("accuracy")
+ )
```

MNIST dataset

Preparing the image data

```
> train_images <- array_reshape(train_images, c(60000, 28 * 28))
> train_images <- train_images / 255
>
> test_images <- array_reshape(test_images, c(10000, 28 * 28))
> test_images <- test_images / 255
>
> train_labels <- to_categorical(train_labels)
> test_labels <- to_categorical(test_labels)
```

MNIST dataset

Now ready to train the network, which in Keras is done via a call to the networks's *fit* method - we *fit* the model to its training data:

```
> network %>% fit(train_images, train_labels, epochs = 3,
+                  batch_size = 125)
```

MNIST dataset

Generate predictions for the first 10 samples of the test set:

```
> network %>% predict(test_images[1:10,]) %>% k_argmax()
```

```
## tf.Tensor([7 2 1 0 4 1 4 9 5 9], shape=(10), dtype=int64)
```

MNIST dataset

Now let's evaluate the model performs on the test set

```
> metrics <- network %>% evaluate(test_images, test_labels)
```

Section 9

Matlab

Matlab

- MATLAB is a programming platform designed specifically for engineers and scientists
- Like R, MATLAB has many specialized toolboxes for making things easier for us
- Using MATLAB, you can:
  - Analyze data
  - Develop algorithms
  - Create models and applications

Matlab

- No need for types. i.e.,

~~int a;~~
~~double b;~~
~~float c;~~

- All variables are created with double precision unless specified and they are matrices.

```
Example:
>>x=5;
>>x1=2;
```

- After these statements, the variables are 1x1 matrices with double precision

Matlab

- a vector `x = [1 2 5 1]`

```
x =
     1    2    5    1
```

- a matrix `x = [1 2 3; 5 1 4; 3 2 -1]`

```
x =
     1      2      3
     5      1      4
     3      2     -1
```

- transpose `y = x'`

```
y =
     1
     2
     5
     1
```

Matlab

-
  ```
  t =1:10
  ```

```
t =
     1    2    3    4   5   6    7   8    9    10
```

-
  ```
  k =2:-0.5:-1
  ```

```
k =
     2   1.5   1   0.5   0   -0.5   -1
```

-
  ```
  B = [1:4; 5:8]
  ```

```
x =
     1      2      3     4
     5      6      7     8
```

Matlab

zeros(M,N)	MxN matrix of zeros	`x = zeros(1,3)`
		`x =`
		`0 0 0`

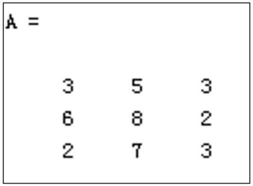
ones(M,N)	MxN matrix of ones	`x = ones(1,3)`
		`x =`
		`1 1 1`

rand(M,N)	MxN matrix of uniformly distributed random numbers on (0,1)	`x = rand(1,3)`
		`x =`
		`0.9501 0.2311 0.6068`

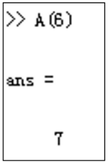
Matlab

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

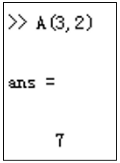
Given:



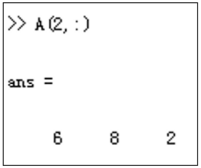
```
A =

     3     5     3
     6     8     2
     2     7     3
```



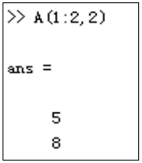
```
>> A(6)

ans =

     7
```



```
>> A(3,2)

ans =

     7
```



```
>> A(2,:)

ans =

     6     8     2
```



```
>> A(1:2,2)

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)
Error: ??? Index exceeds matrix dimensions.

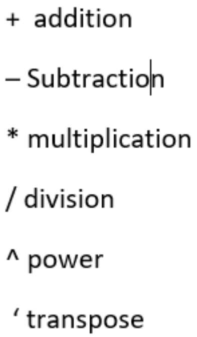
Matlab

-
  ```
  x = [1 2], y = [4 5], z=[ 0 0]

  A = [ x y]

       1    2    4    5

   B = [x ; y]

        1  2
        4  5
  ```

```
      C = [x y ;z]
Error:
??? Error using ==> vertcat CAT arguments dimensions are not consistent.
```

Matlab

Operations:



+ addition

– Subtraction

* multiplication

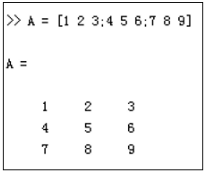
/ division

^ power

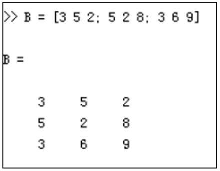
‘ transpose

Matlab

Given A and B:

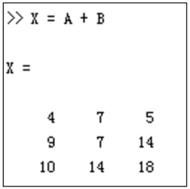


```
>> A = [1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```



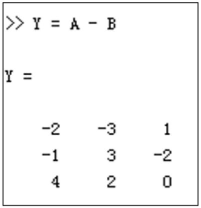
```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition



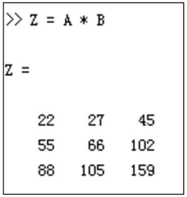
```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction



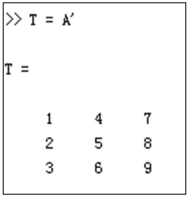
```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product



```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose



```
>> T = A'

T =

     1     4     7
     2     5     8
     3     6     9
```

Section 10

Python

Python

It's free (open source)

- Downloading and installing Python is free and easy
- Source code is easily accessible
- Free doesn't mean unsupported! Online Python community is huge

It's portable

- Python runs virtually every major platform used today
- As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform It's powerful

Dynamic typing

- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, SciPy)
- Automatic memory management

Python - Operations on Numbers

Basic algebraic operations

- Four arithmetic operations: a+b, a-b, a*b, a/b
- Exponentiation: a**b
- Other elementary functions are not part of standard Python, but included in packages like NumPy and SciPy

Comparison operators

- Greater than, less than, etc.: a < b, a > b, a <= b, a >= b
- Identity tests: a == b, a != b

Python - String

Strings are ordered blocks of text

- Strings are enclosed in single or double quotation marks
- Double quotation marks allow the user to extend strings over multiple lines without backslashes, which usually signal the continuation of an expression
- Examples: ‘abc’, “ABC”

Concatenation and repetition

- Strings are concatenated with the + sign:
- Enter: ‘abc’+‘def’
- You get: ‘abcdef’
- Strings are repeated with the * sign:
- Enter ‘abc’*3
- You get: ‘abcabcabc’

Python - Indexing and Slicing

Indexing and slicing

- Python starts indexing at 0. A string s will have indexes running from 0 to len(s)-1 (where len(s) is the length of s) in integer quantities.
- s[i] fetches the i+1 th element in s
- Assume s = ‘string’
- Enter: s[1]
- You get: ‘t’
- s[i:j] fetches elements i (inclusive) through j (not inclusive)

Python - Indexing and Slicing

- Enter: s[1:4]
- You get: ‘tri’
- s[:j] fetches all elements up to, but not including j
- Enter: s[:3]
- You get: ‘str’
- s[i:] fetches all elements from i onward (inclusive)
- Enter: s[2:]
- You get: ‘ring’

Python - List

Basic properties:

- Lists are contained in square brackets []
- Lists can contain numbers, strings, nested subsists, or nothing
- Examples: L1 = [0,1,2,3], L2 = [‘zero’, ‘one’], L3 = [0,1,[2,3],‘three’,[‘four,one’]], L4 = []
- List indexing works just like string indexing
- Lists are mutable: individual elements can be reassigned in place.
- Example: L1 = [0,1,2,3], L1[0] = 4
- Enter: L1
- You get: [4,1,2,3]