cheatsheet

## Data Structures

### R

R offers quite a few data structures

#### Create a vector

# Create a vector (numeric array)  
vec1 <- c(2,3,5,7)  
vec2 <- c(3,5,1,4)  
vec1

## [1] 2 3 5 7

vec2

## [1] 3 5 1 4

vec1 + vec2

## [1] 5 8 6 11

#### Create a matrix

# Create a matrix  
mtrx1 <- matrix(vec1, nrow = 2, ncol = 2)  
mtrx2 <- matrix(vec2, nrow = 2, ncol = 2)  
mtrx1

## [,1] [,2]  
## [1,] 2 5  
## [2,] 3 7

mtrx2

## [,1] [,2]  
## [1,] 3 1  
## [2,] 5 4

mtrx1 + mtrx2

## [,1] [,2]  
## [1,] 5 6  
## [2,] 8 11

#### Create a list

# Create a character array  
names\_array <- c('Jon','Jane','John','Jean')  
  
# Create a list  
lst <- list(id = vec1, name = names\_array)

#### Create a data-frame

# Create a data.frame  
df <- data.frame(id = vec1, name = names\_array)

#### Create a data-table

# Create a data.table  
dt <- data.table(id = vec1, name = names\_array)

### Python

Python offers quite a few data structures

#### Create a vector

import numpy as np  
# Vector  
v1 = np.array([2, 3, 5, 7])  
v2 = np.array([3, 5, 1, 4])  
print(v1)

## [2 3 5 7]

print(v2)

## [3 5 1 4]

print(v1+v2)

## [ 5 8 6 11]

type(v1)

## <class 'numpy.ndarray'>

#### Create a Matrix

# Matrix  
import numpy as np  
m1 = np.array([2, 3, 5, 7])  
m1 = m1.reshape(2,2)  
m1 = m1.T  
m2 = np.array([[3, 5],[1, 4]])  
m2 = m2.T  
print(m1+m2)

## [[ 5 6]  
## [ 8 11]]

#### Create a List

#List is a collection which is ordered and changeable. Allows duplicate members.  
#Tuple is a collection which is ordered and unchangeable. Allows duplicate members.  
#Set is a collection which is unordered and unindexed. No duplicate members.  
#Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.  
  
# list  
id = [2,3,5,7]  
name = ['Jon','Jane','John','Jean']  
lst = [id,name]  
lst[0][1]

## 3

#### Create a data-frame

import numpy as np  
import pandas as pd  
# Dict  
dct = {  
 'id':id,  
 'name':name  
 }  
dct  
  
# Data Frame

## {'id': [2, 3, 5, 7], 'name': ['Jon', 'Jane', 'John', 'Jean']}

df = pd.DataFrame(dct)  
dt\_r = df

## Data Import and Export

### R

# Check if directory exists, if not, create one  
output\_dir <- file.path(getwd(), 'Data')  
  
if (!dir.exists(output\_dir)){  
 dir.create(output\_dir)  
} else {  
 print("Dir already exists!")  
}

## [1] "Dir already exists!"

#Write  
fwrite(dt,'Data/employees.csv')  
  
#Read  
dt\_r <- fread('Data/employees.csv')  
  
#Print top 2 records   
head(dt\_r, 2)

## id name  
## 1: 2 Jon  
## 2: 3 Jane

### Python

import os  
import pandas as pd  
# Check if directory exists, if not, create one  
output\_dir = os.getcwd() + '/Data'  
if not os.path.exists(output\_dir):  
 os.makedirs(output\_dir)  
else:  
 print("Dir already exists!")  
  
#Write

## Dir already exists!

df.to\_csv('Data/employees.csv', index=False)  
  
#Read  
dt\_r = pd.read\_csv('Data/employees.csv')   
  
#Print top 2 records  
dt\_r.head(2)

## id name  
## 0 2 Jon  
## 1 3 Jane

## Data Binding

### R

# Data Binding  
  
# Bind new columns  
age <- c(30, 25, 35, 29)  
dt\_r <- cbind(dt\_r, age)  
  
height <- c(1.7, 1.8, 1.65, 1.85)  
dt\_r <- cbind(dt\_r, height)  
  
# Bind new rows  
# new row is defined as a new data.table  
new\_row <- data.table(id = 9, name = 'Jen', age = 31, height = 1.6)  
  
# Must be of same shape  
dt\_r <- rbind(dt\_r, new\_row)

### Python

import pandas as pd  
  
# Bind new columns  
age = [30, 25, 35, 29]  
dt\_r['age']=age  
  
height = [1.7, 1.8, 1.65, 1.85]  
dt\_r['height']=height  
  
# Another way  
# age = {'age':[30, 25, 35, 29]}  
# height = {'height':[1.7, 1.8, 1.65, 1.85]}  
# dt\_r = pd.concat([dt\_r, pd.DataFrame(age)], axis=1)  
# dt\_r = pd.concat([dt\_r, pd.DataFrame(height)], axis=1)  
  
# Bind new rows  
# new row is defined as a dict first (each item as a list) and then as pandas dataframe  
new\_row = {  
 'id':[9],  
 'name':['Jen'],  
 'age':[31],  
 'height':[1.6]  
 }  
   
dt\_r = pd.concat([dt\_r, pd.DataFrame(new\_row)], ignore\_index=True)

## Data Wrangling

### R

# Data Wrangling  
  
## Descriptive statistics  
summary(dt\_r)

## id name age height   
## Min. :2.0 Length:5 Min. :25 Min. :1.60   
## 1st Qu.:3.0 Class :character 1st Qu.:29 1st Qu.:1.65   
## Median :5.0 Mode :character Median :30 Median :1.70   
## Mean :5.2 Mean :30 Mean :1.72   
## 3rd Qu.:7.0 3rd Qu.:31 3rd Qu.:1.80   
## Max. :9.0 Max. :35 Max. :1.85

## Removing NULLS  
dt\_r[!is.na(name)]

## id name age height  
## 1: 2 Jon 30 1.70  
## 2: 3 Jane 25 1.80  
## 3: 5 John 35 1.65  
## 4: 7 Jean 29 1.85  
## 5: 9 Jen 31 1.60

## Removing Duplicates  
# Add a duplicate  
dt\_r <- rbind(dt\_r, new\_row)  
dt\_r

## id name age height  
## 1: 2 Jon 30 1.70  
## 2: 3 Jane 25 1.80  
## 3: 5 John 35 1.65  
## 4: 7 Jean 29 1.85  
## 5: 9 Jen 31 1.60  
## 6: 9 Jen 31 1.60

# Remove the duplicate  
dt\_r <- unique(dt\_r)  
dt\_r

## id name age height  
## 1: 2 Jon 30 1.70  
## 2: 3 Jane 25 1.80  
## 3: 5 John 35 1.65  
## 4: 7 Jean 29 1.85  
## 5: 9 Jen 31 1.60

## Select rows/columns  
### Rows  
dt\_r[1:2,]

## id name age height  
## 1: 2 Jon 30 1.7  
## 2: 3 Jane 25 1.8

dt\_r[name=='Jon',]

## id name age height  
## 1: 2 Jon 30 1.7

### Columns  
dt\_r[,1:2]

## id name  
## 1: 2 Jon  
## 2: 3 Jane  
## 3: 5 John  
## 4: 7 Jean  
## 5: 9 Jen

dt\_r[,.(name, id)]

## name id  
## 1: Jon 2  
## 2: Jane 3  
## 3: John 5  
## 4: Jean 7  
## 5: Jen 9

### Rows & Columns  
dt\_r[name=='Jon',.(name, id)]

## name id  
## 1: Jon 2

## Where clause  
## group by  
## order by  
  
weight <- c(75,60,70,65,50)  
dt\_r <- cbind(dt\_r, weight)  
  
gender <- c('M','F','M','F','F')  
dt\_r <- cbind(dt\_r, gender)  
  
dt\_r[weight>60, .N, by = gender][order(-N)]

## gender N  
## 1: M 2  
## 2: F 1

### Python

import pandas as pd  
# Data Wrangling  
  
## Descriptive statistics  
dt\_r.describe(include = 'all')  
  
## Removing NULLS

## id name age height  
## count 5.000000 5 5.000000 5.000000  
## unique NaN 5 NaN NaN  
## top NaN Jon NaN NaN  
## freq NaN 1 NaN NaN  
## mean 5.200000 NaN 30.000000 1.720000  
## std 2.863564 NaN 3.605551 0.103682  
## min 2.000000 NaN 25.000000 1.600000  
## 25% 3.000000 NaN 29.000000 1.650000  
## 50% 5.000000 NaN 30.000000 1.700000  
## 75% 7.000000 NaN 31.000000 1.800000  
## max 9.000000 NaN 35.000000 1.850000

dt\_r[~dt\_r['name'].isnull()]

## id name age height  
## 0 2 Jon 30 1.70  
## 1 3 Jane 25 1.80  
## 2 5 John 35 1.65  
## 3 7 Jean 29 1.85  
## 4 9 Jen 31 1.60

dt\_r.isnull().values.any()  
  
## Removing Duplicates  
# Add a duplicate

## False

dt\_r = pd.concat([dt\_r, pd.DataFrame(new\_row)], ignore\_index=True)  
dt\_r  
  
# Drop duplicate

## id name age height  
## 0 2 Jon 30 1.70  
## 1 3 Jane 25 1.80  
## 2 5 John 35 1.65  
## 3 7 Jean 29 1.85  
## 4 9 Jen 31 1.60  
## 5 9 Jen 31 1.60

dt\_r = dt\_r.drop\_duplicates()  
dt\_r  
  
## Select rows/columns  
### Rows

## id name age height  
## 0 2 Jon 30 1.70  
## 1 3 Jane 25 1.80  
## 2 5 John 35 1.65  
## 3 7 Jean 29 1.85  
## 4 9 Jen 31 1.60

dt\_r.iloc[0:2]

## id name age height  
## 0 2 Jon 30 1.7  
## 1 3 Jane 25 1.8

dt\_r[dt\_r['name']=='Jon']  
  
### Columns

## id name age height  
## 0 2 Jon 30 1.7

dt\_r.iloc[:,0:2]

## id name  
## 0 2 Jon  
## 1 3 Jane  
## 2 5 John  
## 3 7 Jean  
## 4 9 Jen

dt\_r[['name','id']]  
  
### Rows & Columns

## name id  
## 0 Jon 2  
## 1 Jane 3  
## 2 John 5  
## 3 Jean 7  
## 4 Jen 9

dt\_r.loc[dt\_r['name']=='Jon', ['name','id']]  
  
## Where clause  
## group by  
## order by

## name id  
## 0 Jon 2

weight = [75,60,70,65,50]  
dt\_r['weight'] = weight

## /Users/anuj/.virtualenvs/cs\_proj/bin/python:1: SettingWithCopyWarning:   
## A value is trying to be set on a copy of a slice from a DataFrame.  
## Try using .loc[row\_indexer,col\_indexer] = value instead  
##   
## See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy

gender = ['M','F','M','F','F']  
dt\_r['gender'] = gender  
  
# Default sorts on frequencies and order in descending  
dt\_r.loc[dt\_r['weight']>60,'gender'].value\_counts()

## M 2  
## F 1  
## Name: gender, dtype: int64

# Data Transformation

### R

# Data Transformation  
  
# Convert height in metres to inches and save as another column  
  
dt\_r[, height\_inch:=height\*39.37]  
  
# Drop a column  
dt\_r[, height\_inch:= NULL]  
  
# Long form  
dt\_r\_l <- melt(dt\_r, id.vars = 'name', measure.vars = c('id','age','height','weight'))  
  
# Wide form  
dt\_r\_w <- dcast(dt\_r\_l, name~variable, value.var = 'value')  
  
summary(dt\_r\_w)

## name id age height weight   
## Length:5 Min. :2.0 Min. :25 Min. :1.60 Min. :50   
## Class :character 1st Qu.:3.0 1st Qu.:29 1st Qu.:1.65 1st Qu.:60   
## Mode :character Median :5.0 Median :30 Median :1.70 Median :65   
## Mean :5.2 Mean :30 Mean :1.72 Mean :64   
## 3rd Qu.:7.0 3rd Qu.:31 3rd Qu.:1.80 3rd Qu.:70   
## Max. :9.0 Max. :35 Max. :1.85 Max. :75

### Python

import pandas as pd  
  
# Data Transformation  
  
# Convert height in metres to inches and save as another column  
dt\_r['height\_inch'] = dt\_r['height']\*39.37  
  
  
# Drop columns  
del dt\_r['height\_inch']  
  
# or  
# dt\_r = dt\_r.drop(columns=['height\_inch'])  
  
#dt\_r['height\_inch'] = dt\_r['height']\*39.37  
dt\_r['height\_inch'] = dt\_r.height\*39.37  
  
# Long form  
dt\_r\_l = pd.melt(dt\_r, id\_vars=['name'], value\_vars=['id','age','height','weight'])  
  
  
# Wide form  
#pd.crosstab(index=dt\_r\_l['name'], columns=dt\_r\_l['variable'], values=dt\_r\_l['value'], aggfunc='first').reset\_index()  
dt\_r\_w = dt\_r\_l.pivot\_table(values='value', index='name', columns='variable').reset\_index()

#Data Joins ###R

#Data Joins  
  
address\_id <- c(1,2,3,4,5)  
address\_array <- c('1640 Riverside Drive, Hill Valley, California'  
 ,'344 Clinton St., Apt. 3B, Metropolis, USA'  
 ,'12 Grimmauld Place, London, UK'  
 ,'221B Baker Street, London, UK'  
 ,'1313 Webfoot Walk, Duckburg, Calisota')  
  
address\_dt <- data.table(address\_id = address\_id, address = address\_array)  
  
address\_id <- c(1,2,3,5,5)  
dt\_r <- cbind(dt\_r, address\_id)  
  
setkey(dt\_r, address\_id)  
setkey(address\_dt, address\_id)  
  
# RIGHT JOIN  
dt\_r[address\_dt]

## id name age height weight gender address\_id  
## 1: 2 Jon 30 1.70 75 M 1  
## 2: 3 Jane 25 1.80 60 F 2  
## 3: 5 John 35 1.65 70 M 3  
## 4: NA <NA> NA NA NA <NA> 4  
## 5: 7 Jean 29 1.85 65 F 5  
## 6: 9 Jen 31 1.60 50 F 5  
## address  
## 1: 1640 Riverside Drive, Hill Valley, California  
## 2: 344 Clinton St., Apt. 3B, Metropolis, USA  
## 3: 12 Grimmauld Place, London, UK  
## 4: 221B Baker Street, London, UK  
## 5: 1313 Webfoot Walk, Duckburg, Calisota  
## 6: 1313 Webfoot Walk, Duckburg, Calisota

# INNNER JOIN  
dt\_r[address\_dt, nomatch=0]

## id name age height weight gender address\_id  
## 1: 2 Jon 30 1.70 75 M 1  
## 2: 3 Jane 25 1.80 60 F 2  
## 3: 5 John 35 1.65 70 M 3  
## 4: 7 Jean 29 1.85 65 F 5  
## 5: 9 Jen 31 1.60 50 F 5  
## address  
## 1: 1640 Riverside Drive, Hill Valley, California  
## 2: 344 Clinton St., Apt. 3B, Metropolis, USA  
## 3: 12 Grimmauld Place, London, UK  
## 4: 1313 Webfoot Walk, Duckburg, Calisota  
## 5: 1313 Webfoot Walk, Duckburg, Calisota

# LEFT JOIN  
address\_dt[dt\_r]

## address\_id address id name age  
## 1: 1 1640 Riverside Drive, Hill Valley, California 2 Jon 30  
## 2: 2 344 Clinton St., Apt. 3B, Metropolis, USA 3 Jane 25  
## 3: 3 12 Grimmauld Place, London, UK 5 John 35  
## 4: 5 1313 Webfoot Walk, Duckburg, Calisota 7 Jean 29  
## 5: 5 1313 Webfoot Walk, Duckburg, Calisota 9 Jen 31  
## height weight gender  
## 1: 1.70 75 M  
## 2: 1.80 60 F  
## 3: 1.65 70 M  
## 4: 1.85 65 F  
## 5: 1.60 50 F

###Python

import pandas as pd  
#Data Joins  
address\_dt = pd.DataFrame()  
  
address\_id = [1,2,3,4,5]  
address\_array = ['1640 Riverside Drive, Hill Valley, California'  
 ,'344 Clinton St., Apt. 3B, Metropolis, USA'  
 ,'12 Grimmauld Place, London, UK'  
 ,'221B Baker Street, London, UK'  
 ,'1313 Webfoot Walk, Duckburg, Calisota']  
  
address\_dt['address\_id'] = address\_id  
address\_dt['address'] = address\_array  
  
address\_id = [1,2,3,5,5]  
dt\_r['address\_id'] = address\_id  
  
# RIGHT JOIN  
dt\_r.merge(address\_dt, how='right')  
  
# INNNER JOIN

## id name ... address\_id address  
## 0 2.0 Jon ... 1 1640 Riverside Drive, Hill Valley, California  
## 1 3.0 Jane ... 2 344 Clinton St., Apt. 3B, Metropolis, USA  
## 2 5.0 John ... 3 12 Grimmauld Place, London, UK  
## 3 7.0 Jean ... 5 1313 Webfoot Walk, Duckburg, Calisota  
## 4 9.0 Jen ... 5 1313 Webfoot Walk, Duckburg, Calisota  
## 5 NaN NaN ... 4 221B Baker Street, London, UK  
##   
## [6 rows x 9 columns]

dt\_r.merge(address\_dt)  
  
# LEFT JOIN

## id name ... address\_id address  
## 0 2 Jon ... 1 1640 Riverside Drive, Hill Valley, California  
## 1 3 Jane ... 2 344 Clinton St., Apt. 3B, Metropolis, USA  
## 2 5 John ... 3 12 Grimmauld Place, London, UK  
## 3 7 Jean ... 5 1313 Webfoot Walk, Duckburg, Calisota  
## 4 9 Jen ... 5 1313 Webfoot Walk, Duckburg, Calisota  
##   
## [5 rows x 9 columns]

dt\_r.merge(address\_dt, how='left')

## id name ... address\_id address  
## 0 2 Jon ... 1 1640 Riverside Drive, Hill Valley, California  
## 1 3 Jane ... 2 344 Clinton St., Apt. 3B, Metropolis, USA  
## 2 5 John ... 3 12 Grimmauld Place, London, UK  
## 3 7 Jean ... 5 1313 Webfoot Walk, Duckburg, Calisota  
## 4 9 Jen ... 5 1313 Webfoot Walk, Duckburg, Calisota  
##   
## [5 rows x 9 columns]

# String Manipulation

###R

# String Manipulation  
dt\_r[name%like%'o']

## id name age height weight gender address\_id  
## 1: 2 Jon 30 1.70 75 M 1  
## 2: 5 John 35 1.65 70 M 3

dt\_r[,.(name, o\_exists = str\_detect(name, 'o'))]

## name o\_exists  
## 1: Jon TRUE  
## 2: Jane FALSE  
## 3: John TRUE  
## 4: Jean FALSE  
## 5: Jen FALSE

dt\_r[,.(name, first\_letter = str\_sub(name, 1,1), last\_letter = str\_sub(name, -1,-1))]

## name first\_letter last\_letter  
## 1: Jon J n  
## 2: Jane J e  
## 3: John J n  
## 4: Jean J n  
## 5: Jen J n

# Regex  
  
dt\_r[,str\_view\_all(name,'n')]

## PhantomJS not found. You can install it with webshot::install\_phantomjs(). If it is installed, please make sure the phantomjs executable can be found via the PATH variable.

dt\_r[,str\_view\_all(name,'n$')]

dt\_r[,str\_view\_all(name,'^J')]

###Python

import pandas as pd  
dt\_r[dt\_r['name'].str.contains("o")]

## id name age height weight gender height\_inch address\_id  
## 0 2 Jon 30 1.70 75 M 66.9290 1  
## 2 5 John 35 1.65 70 M 64.9605 3

dt\_r['name'].str.contains("o")

## 0 True  
## 1 False  
## 2 True  
## 3 False  
## 4 False  
## Name: name, dtype: bool

dt\_r['name'].str[:1]

## 0 J  
## 1 J  
## 2 J  
## 3 J  
## 4 J  
## Name: name, dtype: object

dt\_r['name']  
  
#Regex

## 0 Jon  
## 1 Jane  
## 2 John  
## 3 Jean  
## 4 Jen  
## Name: name, dtype: object

dt\_r.loc[dt\_r['name'].str.contains('^J'),['name']]

## name  
## 0 Jon  
## 1 Jane  
## 2 John  
## 3 Jean  
## 4 Jen

dt\_r.loc[dt\_r['name'].str.contains('n$'),['name']]

## name  
## 0 Jon  
## 2 John  
## 3 Jean  
## 4 Jen

# Date and Time

###R

# Date and Time  
birth\_date <- c('1989-03-01','1994-09-09','1984-07-15','1990-05-01','1988-06-03')  
  
dt\_r <- cbind(dt\_r, birth\_date)  
summary(dt\_r)

## id name age height weight   
## Min. :2.0 Length:5 Min. :25 Min. :1.60 Min. :50   
## 1st Qu.:3.0 Class :character 1st Qu.:29 1st Qu.:1.65 1st Qu.:60   
## Median :5.0 Mode :character Median :30 Median :1.70 Median :65   
## Mean :5.2 Mean :30 Mean :1.72 Mean :64   
## 3rd Qu.:7.0 3rd Qu.:31 3rd Qu.:1.80 3rd Qu.:70   
## Max. :9.0 Max. :35 Max. :1.85 Max. :75   
## gender address\_id birth\_date   
## Length:5 Min. :1.0 Length:5   
## Class :character 1st Qu.:2.0 Class :character   
## Mode :character Median :3.0 Mode :character   
## Mean :3.2   
## 3rd Qu.:5.0   
## Max. :5.0

dt\_r[,birth\_date:= as.IDate(birth\_date)]  
summary(dt\_r)

## id name age height weight   
## Min. :2.0 Length:5 Min. :25 Min. :1.60 Min. :50   
## 1st Qu.:3.0 Class :character 1st Qu.:29 1st Qu.:1.65 1st Qu.:60   
## Median :5.0 Mode :character Median :30 Median :1.70 Median :65   
## Mean :5.2 Mean :30 Mean :1.72 Mean :64   
## 3rd Qu.:7.0 3rd Qu.:31 3rd Qu.:1.80 3rd Qu.:70   
## Max. :9.0 Max. :35 Max. :1.85 Max. :75   
## gender address\_id birth\_date   
## Length:5 Min. :1.0 Min. :1984-07-15   
## Class :character 1st Qu.:2.0 1st Qu.:1988-06-03   
## Mode :character Median :3.0 Median :1989-03-01   
## Mean :3.2 Mean :1989-06-05   
## 3rd Qu.:5.0 3rd Qu.:1990-05-01   
## Max. :5.0 Max. :1994-09-09

dt\_r[,birth\_date:= as.numeric(birth\_date)]  
summary(dt\_r)

## id name age height weight   
## Min. :2.0 Length:5 Min. :25 Min. :1.60 Min. :50   
## 1st Qu.:3.0 Class :character 1st Qu.:29 1st Qu.:1.65 1st Qu.:60   
## Median :5.0 Mode :character Median :30 Median :1.70 Median :65   
## Mean :5.2 Mean :30 Mean :1.72 Mean :64   
## 3rd Qu.:7.0 3rd Qu.:31 3rd Qu.:1.80 3rd Qu.:70   
## Max. :9.0 Max. :35 Max. :1.85 Max. :75   
## gender address\_id birth\_date   
## Length:5 Min. :1.0 Min. :5309   
## Class :character 1st Qu.:2.0 1st Qu.:6728   
## Mode :character Median :3.0 Median :6999   
## Mean :3.2 Mean :7096   
## 3rd Qu.:5.0 3rd Qu.:7425   
## Max. :5.0 Max. :9017

###Python

import pandas as pd  
# Date and Time  
  
birth\_date = ['1989-03-01','1994-09-09','1984-07-15','1990-05-01','1988-06-03']  
  
# String Object  
dt\_r['birth\_date'] = birth\_date  
  
# Date Object  
pd.to\_datetime(dt\_r['birth\_date']).dt.strftime('%Y-%d-%m')  
  
# Days since epoch

## 0 1989-01-03  
## 1 1994-09-09  
## 2 1984-15-07  
## 3 1990-01-05  
## 4 1988-03-06  
## Name: birth\_date, dtype: object

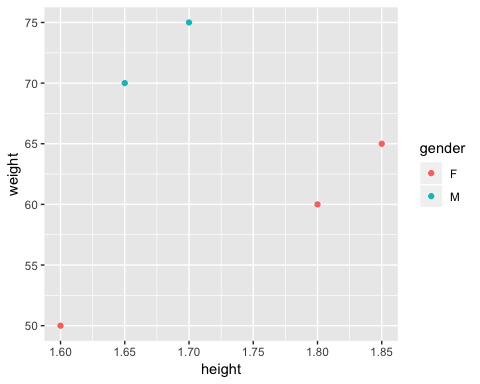
pd.to\_datetime(dt\_r['birth\_date']) - pd.datetime(1970,1,1)

## 0 6999 days  
## 1 9017 days  
## 2 5309 days  
## 3 7425 days  
## 4 6728 days  
## Name: birth\_date, dtype: timedelta64[ns]

# Data Visualization

###R

# Data Visualization  
ggplot(dt\_r, aes(x=height, y=weight, color = gender))+  
 geom\_point()



###Python

import seaborn as sb  
sb.pairplot(x\_vars=["height"], y\_vars=["weight"], data=dt\_r, hue="gender", size=5)

## <seaborn.axisgrid.PairGrid object at 0x1308bc898>  
##   
## /Users/anuj/anaconda3/lib/python3.7/site-packages/seaborn/axisgrid.py:2065: UserWarning: The `size` parameter has been renamed to `height`; pleaes update your code.  
## warnings.warn(msg, UserWarning)