2019年9月19日 3:35

- **Homework**
  补充了作业中要求的，基于基本理论的操作

2019年9月19日 3:36

HW1

Matrix

1. 用outer创造matrix

   row = c(1:10)
   col = c(1:10)
   m1 <- outer(row+1,col+1, FUN = "/")

   / 表除; pmax, pmin 取max/min

2. sample

   v1<-sample(c(1:30),30,replace = FALSE)， replace == TRUE表示是否不再重复

   m1<-matrix(v1,nrow=5,ncol=6,byrow=TRUE) 从row 开始填充

   norm of matrix

   n1 = norm(m1, type = "O")
   O: L-1 norm, I: L-infinity, F: Frobenius norm

   matrix 某一三角改变符号

   outer_prod2[lower.tri(outer_prod2)] <- -1 * outer_prod2[lower.tri(outer_prod2)]

   取matrix某几列

   t1[,c(which(colnames(t1)!="d"))]

3. Assign to `ca` the matrix corresponding to the upper triangulart of the cholesky decomposition of `a`

   ca<-chol(a,pivot = TRUE)
   pivot <- attr(ca, "pivot")
   oo <- order(pivot)
   ca=ca[, oo]

Tibble

1. 计算tibble每列NA数量

   v1<-colSums(is.na(t1))

   忽略NA后形成新的tibble

   t2<-na.omit(t1)

   按某列排序

   t3 <-t2[order(t2$Temp,t2$Ozone),]

2. 按某一列的特点取tibble

   t2 <- t1[t1$mpg > 18,]

   对tibble中所有entry 使用方程

   t3<-as_tibble(lapply(t2,as.integer))

   合并某两列并移动至最前

   t4$vsam=rowSums(cbind(t4$vs,t4$am))
   t4<-t4[,c(length(t4),1:(length(t4)-1))]

   删除某一列：t2$drat<-NULL

   对调两行内容与名字

   t3 <- t2
   t3[, c("mpg", "qsec")] <- t2[, c("qsec", "mpg")]
   names(t3)[which(names(t3) == "qsec")] <- "mpg"
   names(t3)[1] <- "qsec"

Array

1. 合并两个matrix为array：a1 <- array(c(m1,m2),dim=c(10,10,2))

   对第二个dimension sum: m3<-apply(a1,c(1,3),sum)

List

1. 合并list

   list(a,b)成为2D，c(a,b) b后接a

2. 赋予dim

   dim(l2)<-c(13,4)

Factor

1. ordered factor with rev order: f2 <- ordered(letters, levels = rev(letters))
   ordered factor followed by 4 NA: f3<-factor(c(letters,rep(NA,times=4)),ordered=TRUE,exclude=NULL)

   删除element与相应level: f4<-f3[-c(which(f3=="c")),drop = TRUE]

Missing value

1. set seed

   set.seed(0)
   seed <- .Random.seed

   取某一列整：mtcars_rounded$gear <- round(mtcars_rounded$gear)

   计算每列平均数，并把NA的位置用mean 取代

   mean_mtcars <- as.vector(apply(mtcars_tibble, 2, mean, na.rm = TRUE))
   mtcars_bis <- mtcars_tibble
   mtcars_bis[list_row, list_col] <- mean_mtcars[list_col]

Type coercion

list 转变为 tibble: t1 <- tibble(l[[1]], l[[2]], l[[3]])

取非某些值得行：t2<-t1[-c(which(t1$letters=="b")),]

tibble 转 matrix: m<-as.matrix(t2[c(1,2)])

matrix 相乘: m_prod<- m %*% t(m)

sampling

1. 取sample后录为factor

   random_seq<-sample(c(1:10),200,replace=TRUE)
   random_factor<-factor(random_seq)

   用table count factor: tab<-table(random_factor)

   按数量排序后取前三：

   random_order<-as.integer(names(sort(tab)))
   top3<-top3[c(1:3)]

2. permute 两个list的数，并取名

   sample_space<-expand.grid(c(1:6),c(1:6)) //36*2
   colnames(sample_space)<-c("dice1","dice2")

   把6代为T，别的代为F，然后得到double 6的行为T的list

   tmp<-ifelse(sampled_events==6,TRUE,FALSE)
   double_six<-as.vector(tmp[,"dice1"]&tmp[,"dice2"])

   rle，对list使用回归两个值，$value 与 $lengths ，每一对值表示value重复lengths 次

distribution

285个人五个同生日概率：prob1<-pbirthday(285,coincident=5)

正太分布，$X \leq 3$, mean 0 与 var 2. pnorm(3,0,sqrt(2))

从mean 3 的exp distribution 取10个数：rexp(10,rate=1/3)

2019年9月19日 6:28

HW2

loops

tibble 以某一列牌序：sort(mtcars$disp,decreasing = TRUE)

choices

用case_when 把相应内容替换成固定的别的内容

```
gear<-dplyr::case_when(
  gear == 3 ~ "low gear",
  gear == 4 ~ "medium gear",
  gear == 5 ~ "high gear",
  is.na(gear) ~ "???",
  TRUE ~ as.character(gear)
)
```

用runif 形成11个 1-11之间的数，符合normal distribution: vec2<-runif(11,max=10,min=1)

sorting

自己写一个sorting的函数

```
my_sort<-function(vec){
    i<-1
    newvec<-vector()
    oldvec<-vec
    while(i<=length(vec)){
        pos<-min_position(oldvec)
        newvec<-c(newvec,oldvec[[pos]])
        oldvec<-oldvec[-c(pos)]
        i<-i+1
    }
    newvec
}
```

string

写了几个关于string 分解合并的方程，其中使用到：

strsplit(x, split="") 按split 分解，成为多个element

unlist(x, collapse="")合为一个string.

apply anonymous function

sum: m_row_sum<-apply(m,1,sum) //1表示sum by row, 2 表示对sum by col

choices

简单例子，注意stop表示terminate and gives the error message

```
f3<-function(mass,unit){
    if (mass<0)
        stop("Negative Mass Error")
    if(unit=="lb")
        mass*0.45
```

```
        else if (unit=="kg")
                mass/0.45
        else
                stop("Invalid Unit")
}
```

2019年10月18日 9:50

HW3

triple dot

取任意数的argument，这里意思是把他们合为一个后，放入f2

```
f3<-function(...){
    if(missing(...)){}
    else
        f2(unlist(list(...)))
}
```

实现二分法找根

if(abs(b-mid)<tol | f(mid) ==0)//注意回归条件

对function list 使用 map

```
library(purrr)
list_fun<-c(linear = function(x) x,quadratic = function(x) (x - pi/2)^2 - pi/2,sinusoidal = function(x) sin(x))
integral_values<-map_dbl(list_fun,function(x) integrate(x,-3*pi/4,3*pi/4)$value)
root_values<-map_dbl(list_fun,function(x) uniroot(x,c(-3*pi/4,3*pi/4))$root)
min_values<-map_dbl(list_fun,function(x) optimise(x,c(-3*pi/4,3*pi/4),maximum = FALSE)$objective)
min_position<-map_dbl(list_fun,function(x) optimise(x,c(-3*pi/4,3*pi/4),maximum = FALSE)$minimum)
#注意minimum是位置
```

map(用purrr)

1. 可以pass extra argument： m3<-map_dbl(mtcars,mean,trim = 0.1, byrow=FALSE)
2. 先以某列split(相同的归到一个list),在求个数： m4<-map_int(split(mtcars,mtcars$cyl),nrow)
3. 用keep保留col,再summary: m1<-map(keep(iris,is.numeric),summary)

   可以加extra argument: m2<-map(keep(iris,is.numeric),mean,trim=0.05)

   对不同的row加不同额外argument: m3<-map2_dbl(keep(iris,is.numeric),c(0.01,0.05,0.1,0.2),mean)
4. detect the numeric column with variance greater than 3

   ```
   result<-map_dbl(keep(iris,is.numeric),var)
   select_var<-detect_index(result,function(x) x>3)
   m4<-names(result)[select_var]
   ```
5. 找一类颜色中最贵的价格： max_price<-map(split(diamonds,diamonds$color),function(x) max(x$price))
6. 用modify 把整个tibble用第一列替代： t1<-modify(as_tibble(diamonds), ~.x[[1]]) //modify 保留格式

   modify_if, 符合条件才map: t2<-modify_if(as_tibble(diamonds),is.numeric,mean)

   modify_at，在那个位置才map: mtcars_factor<-modify_at(mtcars,c("cyl","gear"),as.factor)

   mtcars_factor_level<-map_if(mtcars_factor,is.factor,~length(levels(.x)))//可以使用anonymous func, .x

   map2_chr: sentences_list<-map2_chr(name_list,nick_name_list_update,~paste(.x,"can be called",.y))

Reduce

1. 实现factorial：reduce(as.double(1:n),function(.x,.y) .x*.y) //部分code
2. detect回归index, return that [ ]: df1_factor<-detect(df1,is.factor)
3. list of df1 的合并：用样的col会合并
   df_list<-list(df1,df2,df3)
   merged_df<-reduce(df_list,merge,all=TRUE) // all =T 等用于all.x=T, all.y =T.意思就是重复的合为一个

function factory

1. 记得force()

```
mixture_factory<-function(c1, mu1, sigma1, c2, mu2, sigma2){
    force(c(c1,mu1, sigma1, c2, mu2, sigma2))#force只接受一个arg

    function(x) c1*dnorm(x,mu1,sigma1)+c2*dnorm(x,mu2,sigma2)
    }
```

2. poly functions

```
polynomial_factory<-function(a){
    force(a)
    function(x){
        z<-0
        for(i in length(a):1){
            z<-a[[i]]*x^(i-1)+z
        }
        return(z)
    }
}
```

   input_functions<-map(input,polynomial_factory) //input是个List of list.

   v<-map(input_functions,optimise,c(-2,5),maximum=FALSE)//再用求每个极值

3. Use `optimise` to calculate the minimum point `xmin`: xmin<-optimise(f,c(-5,10),maximum=FALSE)$minimum
   // optimise arg为: function, bound. 注意$minimum是位置，$objective才是对应的值

2019年10月18日 11:05

HW4

Compose function

```
compose_2<-function(f,g){
    force(c(f,g))
    function(x){
        f(g(x))
    }
}
compose_3<-function(f,g,h){
    force(c(f,g,h))
    function(...){
        f(compose_2(g,h)(...))
    }
}
```

memoise

记住原来计算过的答案

```
climb2<-memoise(function(n){
    if (n==1) return(1)
    if (n==2) return(2)
    climb(n-1)+climb(n-2)
})
```

Exit handler

```
capture.output2<-function(code,file){
    sink(file)//divert to path at file
    on.exit(sink, add = TRUE,after=TRUE)
    force(code)
    readLines(file)
}
```

Capture error

1. safely

   y<-map(iris,safely(max))

   is_ok<-map_lgl(transpose(y)$error,is_null)// list中元素为list（两个元素，result,error），详情见笔记
2. possibly，error的地方用otherwise后的值替代

   m3<-map_dbl(iris,possibly(max,otherwise=NA_real_))
3. quitely,类似safely,list 元素现有四个

Input/output

1. 取有效数字：format(mean_mpg,digits=4)
2. print na: print(x,digits=3,na.print='N/A')
3. 像C一样输出：

   ```
   wake_up<-function(hour,minute,AMPM){
       sprintf("I wake up at %02.f:%02.f %s.",hour,minute,AMPM)
   }
   ```

Partial

提前prefill 一些argument:

mean_sqrt_fun_narm<-partial(compose(sqrt,mean), na.rm = TRUE)

```
partial_2<-function(f,...){//自己做的partial可以帮助理解
        force(f)
        function(x) f(x,...)
}
```

2019年10月18日 12:22

OOP

```
# HW4: Object Oriented Programming
#'
# In this exercise, we will create a class `shape` and a child class `rectangle`.
#'
# 1. Create a constructor `new_shape(x)` that takes in a string and instantiates
#    an object of class `shape` using `structure`. Don't forget to use `stopifnot` to check the input.
#    Create a helper function `shape(x)` that coerces the input
#    to a string with `as.character` and then calls the constructor.
## Do not modify this line!
new_shape<-function(x=character()){
        stopifnot(is.character(x))
        structure(x,class = "shape")
}
shape<-function(x=character()){
        x<-as.character(x)
        new_shape(x)
}


# 2. Write a new `print` method for class `shape` that prints `"My name is <shape>."` using the function `print`.
#    The `<shape>` should be substituted by the string in the instance of `shape`.
#    Make sure that the method returns the object invisibly by adding `invisible(x)` at the end.
#    Create an instance of `shape` called `s1` of shape `unknown` and then print `s1`.
## Do not modify this line!
print.shape<-function(x,...){#... for extra arg
        print(paste0("My name is ",x,"."))
        invisible(x)
}
s1<-new_shape("unknown")
print.shape(s1)
# 3. Write a generic function called `area(x)` that does nothing. Don't forget to use `UseMethod`.
#    Write a default method `area` that prints `"My area is not defined."` using the function `print`.
#    Test this new method using instance `s1`.
## Do not modify this line!
area<-function(x){
        UseMethod("area")
}
area.default<-function(x){
        print("My area is not defined.")
}
area(s1)
```

```
# 4. Create a constructor `new_rectangle(x, l, w)` that takes as input a string `x`,
#    and two numbers `l` and `w`, and instantiates an object `rectangle` of class `shape`.
#    Use the function `class()`. The constructor should give two attributes to `rectangle` :
#    `length`, equal to `l` and `width`, equal to `w`. Use `attr()` to create the attributes.
#    Use the line of code `rectangle <- function (x, l, w) new_rectangle(x, l, w)` to create a
helper.
## Do not modify this line!
new_rectangle<-function(x,l,w){
        stopifnot(is.character(x))
        stopifnot(is.numeric(l))
        stopifnot(is.numeric(w))
        attr(x,'length')<-l
        attr(x,'width')<-w
        structure(x,class=c("rectangle","shape"))
}
rectangle<-function(x, l, w){
        x<-as.character(x)
        l<-as.numeric(l)
        w<-as.numeric(w)
        new_rectangle(x,l,w)
}

# 5. Write a new method `area` for the class `rectangle` that prints `My area is <area>.` using
`print`.
#    The `<area>` should be substituted by the area of a rectangle of length `length` and
width `width`.
#    Use `attr()` to access these two parameters.
## Do not modify this line!
area.rectangle<-function(x){
        l<-attr(x,"length")
        w<-attr(x,"width")
        print(paste0("My area is ",w*l,"."))
}
my_rec<-new_rectangle("my_rec",2,5)
area(my_rec)

# 6. Write a method `print` for the class `rectangle` that prints `"My name is <shape>.\nI am
a rectangle."`
#    using `print.shape` and `print`. The `<shape>` should be substituted by the string in the
instance of `rectangle`.
#    Make sure that the method returns the object invisibly by adding `invisible(x)` at the
end.
## Do not modify this line!
print.rectangle<-function(x,...){
        print.shape(x)
        print("I am a rectangle.")
        invisible(x)
}
print(my_rec)
```

```
# 7. Create an instance `r1` of `rectangle` with name `rect`, length 10, and width 5.
#    Call `area` and `print` method to instance `r1`.
## Do not modify this line!
r1<-new_rectangle("rect",10,5)
area(r1)
print(r1)


# HW4: Object Oriented Programming
#'
# In this exercise, we will create a class `distance` on top of integers and overload the
function `mean()`.
#'
# 1. Create a constructor `new_distance(x, units)` where:
#    - `x` is a vector of numbers (not a vector of strings like `"ten"`),
#    - `units` is an attribute which can take values in `c("mm", "cm", "m", "km")`, with default
value `"m"`,
#     - the object of class `distance` is instanciated using `structure`.
#    (Hint : Don't forget to use `stopifnot` to check the input type with `is.numeric` and
`is.character`.)
#    Create a helper `distance(x, units)` that checks whether the input `unit` is in the feasible
domain
#    `c("mm", "cm", "m", "km")` and throws an error if not.
#     (Hint : Use `match.arg()`.)
#    Write a new `print` method named `print.distance` for class `distance` that prints
#    `"the distance is <distance> <unit>"` when `<distance>` is a vector of length 1.
#    It should print `"the distances are <distance> <unit>"` if
#    `<distance>` is a vector of at least two elements.
#    For example, `print(distance(1:3, "cm"))` should print `"the distances are 1 2 3 cm"`.
#    Use the functions `print()`, `paste0()`, and the function `attr()` to access the unit of the
distance
#    and use `as.character` to convert the `distance` to a vector of characters,
#    and the function `attr()` to access the unit of the distance.
#    (Note: there is a space between `<distance>` and `<unit>` and no space at the end of the
string.)
#    Make sure that the method returns the object invisibly by adding `invisible(x)` at the
end.
#    The `"<distance>"` should be substituted by the number when instantiating `distance`.
#    The default unit should be `m`.
#    Create an instance of `distance` called `dist1` using `x = 10` and then print `dist1`.
## Do not modify this line!
new_distance<-function(x=double(),units="m"){
        stopifnot(is.numeric(x))
        stopifnot(is.character(units))
        structure(x,class="distance",units=units)
}
distance<-function(x=double(),units="m"){
        units<-match.arg(units,c("mm", "cm", "m", "km"))
        new_distance(x,units=units)
}
print.distance<-function(x,...){
        if(length(x)<1)
```

```
            stop("invalid input.")
        else if(length(x)==1)
            print(paste0("the distance is ",as.character(x), " ",attr(x,"units")))
        else
            print(paste0("the distances are ",as.character(paste0(x,collapse=' ')), "
            ",attr(x,"units")))
        invisible(x)
}
dist1<-new_distance(10)
print(dist1)
# 2. Create a function `to_mm(x)` that takes in a distance and converts it to the equivalent
distance
#    in millimeters (i.e., it returns an object of class `distance`.
#    (Hint: Use `attr()` to access the units and `switch` to convert them.
#    Use `stop("Unknown unit.")` as default to specify the output when the input unit is not
#    in the feasible domain. Do not use `if` and `else`.
#    Use the helper function `distance()`to create the final object.)
#    Create an instance of `distance` called `dist1_converted` by converting `dist1` to
millimeters.
## Do not modify this line!
to_mm<-function(x){
        unit<-attr(x,"units")
        switch(unit,
                "mm" = distance(x,"mm"),
                "cm" = distance(x*10,"mm"),
                "m" = distance(x*1e3,"mm"),
                "km" = distance(x*1e6,"mm"),
                stop("Unknown unit.")
        )
}
dist1_converted<-to_mm(dist1)
# 3. Write a new function `c.distance` for the class `distance` that combines its arguments to
form a vector
#    of distances converted in millimeters.
#    (Hint: note that the argument to the `c()` generic is `...`, and you can use `list(...)`
#    to capture them. You can then use `sapply()` and `to_mm()` to first convert each
distance in `list(...)`
#    the list into millimeters.)
#    Next, instantiate a new distance `dist2` of `20cm`.
#    Concatenate `dist1` and `dist2` together using `c()` and store the distance vector into
`dist_both`.
## Do not modify this line!
c.distance<-function(...){
        distance_data<-list(...)
        data<-sapply(distance_data,to_mm)
        distance(data,"mm")
}
dist2<-new_distance(20,"cm")
dist_both<-c(dist1,dist2)
# 4. Write a new `mean.distance` function for the class `distance`
#    that calculates the average distance of a distance vector in millimeters.
#    (Hint: in the implementation of the function, first convert all elements in the list to
```

```
millimeters.
#    We can do that by simply call `to_mm(x)` because it is well defined for a distance object.
#    Then convert the distance variable to numeric by `as.numeric`.
#    Finally, since all the elements have been converted to numeric values,
#    we can simply calculate the average of the list by calling `mean(x)` and return an object
of class
#    distance by calling the helper `distance()`.)
#    After you define function `mean.distance`, calculate the mean `dist_both` and store it
into `avg`.
## Do not modify this line!
mean.distance<-function(x,...){
  data <- as.numeric(to_mm(x))
  return(distance(mean(data), "mm"))
}
avg<-mean(dist_both)


# HW4: oop weight
#'
# In this exercise, we will create a class `weight`.
#'
# 1. Create a constructor `new_weight(x = double(), units = "kg")` that:
#       - takes in a double `x` of length one,
#       - a `units` attribute,
#       - and initiates an object of class `weight` using `structure`.
#    Don't forget to use `stopifnot` to check that `x` has the correct type and length.
#    Because `units` might include `"lbs"` and `"kg"`, use `match.arg` to check the
#    validy of the second argument.
## Do not modify this line!
new_weight<-function(x=double(),units="kg"){
        stopifnot(is.numeric(x)&length(x)==1)
        units<-match.arg(units,c("lbs","kg"))
        structure(x,class="weight",units=units)
}


# 2. Create a helper function `weight` for `new_weight`,
#    which will convert `x` into a double using `as.double`.
## Do not modify this line!
weight<-function(x=double(),units="kg"){
        x<-as.double(x)
        new_weight(x,units=units)
}


# 3. Write a new `print` method for class `weight` that prints
#    `"The weight is <weight> <units>"` using `print`.
#    The `print` method should return the input invisibly.
#    Create an instance of `weight` called `d1` using `x = 50`,
#    `units = "lbs"` and then print `d1`.
## Do not modify this line!
print.weight<-function(x,...){
```

```
        units<-attr(x,"units")
        print(paste0(c("The weight is",x,units),collapse=" "))
        invisible(x)
}
d1<-weight(50,"lbs")
print(d1)
# 4. Write a generic function called `convert(x)` :
#    If the units of `x` is `"lbs"`, it should return a weight object using `weight()`
#    corresponding to `x` converted to `"kg"`.
#    If the units of `x` is `"kg"`, it should return a weight object using `weight()`
#    corresponding to `x` converted to `"lbs"`.
#    For this exercise, assume that 1 kilogram corresponds to 2.2 pounds.
#    Call this new method to instance `d1`, save the return object into `d2`.
## Do not modify this line!
convert<-function(x){
        UseMethod("convert")
}

convert.weight<-function(x){
        if(attr(x,"units")=="kg")
                weight(2.2*x,"lbs")
        else if(attr(x,"units")=="lbs")
                weight(x/2.2,"kg")
        else
                stop("do not support")
}

d2<-convert(d1)
# 5. Write a `+` method for the `weight` class.
#    Note that arithmetic operators take two arguments: `x` and `y`.
#    Assume that the return units is the unit of `x`.
#    In other words, if `x` is in `"kg"`, then `x + y` should be an object of class
#    weight in `"kg"`. Take care of this by converting `y` to the right units if
#    necessary.
#    In this exercise, after `y` as been converted if necessary,
#    you should NOT use `unclass()`. Instead, delegate the work to the `+`
#    operators of the base type (`double`) using `NextMethod("+")`.
#'
## Do not modify this line!
'+.weight'<-function(x,y){#意思是说+的method和之前的其实一样，别忘记' '即可
        if(attr(x,"units")==attr(y,"units")){
                weight(x[1]+y[1],attr(x,"units"))#可以替代为NextMethod('+')
        }
        else{
                y<-convert(y)
                NextMethod('+')#也可以空格
        }

}

# d3<-weight(90,"lbs")
```

```
# d4<-weight(100,"kg")
# d3+d4

# HW4: Author and Book OOP
#'
# In this exercise we will create an `author` class to model a book's author.
#'
# 1. Create a constructor `new_author` for an `author` class to model a book's
# author.
# It should take as input two strings: `name`, `email`.
# It should check that the inputs are both strings (using `stopifnot`).
# Also check that the `email` contains a `@` sign (using `stopifnot` and
# `grepl`).
# The constructor should then create a named list with the two strings (first
# element is the name - called `"name"` and second element is the email -
# called `"email"`) and use `structure` with this list to create an object of
# class `author`.
## Do not modify this line!
new_author<-function(name,email){
	stopifnot(is.character(name)&is.character(email)&grepl('@',email))
	named_list<-list("name"=name,"email"=email)
	structure(named_list,class="author")
}

# 2. Create a helper `author` that wraps `new_author`. It should have default
# values for `name` (`"John Doe"`) and `email` (`"unknown@unknown"``), return
# an error if the length of the arguments is not 1 and cast them to characters
# before calling `new_author` using `as.character`.
# Create an author object using the `author` helper with name `Susan Barker`
# and email `susan.barker@mail.com` and store it in variable `author_example`.
## Do not modify this line!
author<-function(name="John Doe",email="unknown@unknown"){
	stopifnot(length(name)==1&length(email)==1)
	new_author(as.character(name),as.character(email))
}
author_example<-author(name="Susan Barker",email="susan.barker@mail.com")

# 3. Create 2 functions `get_name` and `get_email` which take an `author`
# object named `author_object` as input: `get_name` should return the name
# of the author, `get_email` should return the email address of the author.
# Then, create a generic `change_email` which takes an `author` object named
# `author_object` along with a string `email` as inputs. Implement the method
# for objects of the class author (i.e., write `change_email.author`) so that
# the method returns a new `author` object with the name of the input `author_object`
# and the input email address.
## Do not modify this line!
get_name<-function(author_object) author_object$name
get_email<-function(author_object) author_object$email
change_email<-function(author_object,email){
	UseMethod("change_email")
}
change_email.author<-function(author_object,email){
```

```
        new_author(get_name(author_object),email)
}
# 4. Change the email address of `author_example` to `s.barker@mail.com`
## Do not modify this line!
author_example<-change_email(author_example,"s.barker@mail.com")


# 5. Create a method `print.author` that prints the string
# `"Author <name>, e-mail: <email>"` where `<name>` and `<email>` are the
# strings corresponding to name and email of the `author` object. You should
# use the functions `print` and `paste0` to do so. Note that:
#    - the arguments of print are `x` and `...`, but `...` won't be used
#    in the body of `print.author`,
#    - print.author` method should return the first argument invisibly.
#'
## Do not modify this line!
print.author<-function(x,...){
        print(paste0("Author ",get_name(x),", e-mail: ",get_email(x)))
        invisible(x)
}
print(author_example)

# hw4_oop4
#'
# In this exercise, we will create a class `shakeshack_order`.
#'
# 1. Create a constructor `new_shakeshack_order(names, prices)` that:
#     - takes in a vector of `names`
#     - a vector of `price` attribute whose type is double.
#     - instanciates an object of class `shakeshack_order` using `structure`.
#     - and it should be a list with 2 elements: `names` and `prices`.
# Note: Use `stopifnot` to check the input.
# Use `new_shakeshack_order(names, prices)` to create a helper function
`shakeshack_order`
# that coerces the arguments `names` and `prices` respectively to string and numeric
# using `as.character` and `as.double`.
## Do not modify this line!
new_shakeshack_order<-function(names,prices){
        stopifnot(is.character(names)&is.numeric(prices))
        named_list<-list("names"=names,"prices"=prices)
        structure(named_list,class="shakeshack_order")
}
shakeshack_order<-function(names,prices){
        names<-as.character(names)
        prices<-as.double(prices)
        new_shakeshack_order(names,prices)
}

# 2. Write a new `sum(..., na.rm = FALSE)` method for the class `shakeshack_order` that
# returns the sum of the prices in a given order. Note that:
#    - the `sum` generic can take more than one argument via `...`, and you can capture
#    it using `list(...)`.
```

```
#    - the `na.rm` argument should be used to provide a way to sum
#    when some prices are not available.
# For instance, the following code should work without error:
# ```
# o <- shakeshack_order(c("shack burger", "fries"), c(5, 2))
# o2 <- shakeshack_order(c("fries", "coke"), c(2, NA))
# sum(o)
# sum(o, o2)
# sum(o, o2, na.rm = TRUE)
# ```
# The first sum should be equal to 7, the second to `NA`, and the third to 9.
# Do NOT use a `for`, `while` or `repeat` loop!
# (Hint: a nice solution could use a combination of `map` and `reduce`.)
## Do not modify this line!
library(purrr)
sum.shakeshack_order<-function(..., na.rm=FALSE){
    data<-list(...)#用这个来把输入做成一个list
    reduce(map(data,~.x$prices),sum,na.rm=na.rm,.init=0)
}

# 3. Write a new `print` method for the class `shakeshack_order` that prints
# `"Your order is <names>. The total price is sum(<prices>)."` using `print`.
# If `length(names)` is larger than one (e.g., 3), the function should print
# `"Your order is <names[1]>, <names[2]>, <names[3]>. The total price is sum(<prices>)."`
# For instance, printing the order `o` describe above should output
# `"Your order is shack burger. The total price is $5.29."`.
# Note that:
#    - The `print` method should return the input invisibly.
#    - The arguments of print are `x` and `...`, but `...` won't be used in the
#    body of `print.shakeshack_order`.
## Do not modify this line!
print.shakeshack_order<-function(x,...){
    if(length(x$name)==1)
        print(paste0("Your order is ", x$names,". The total price is $",sum(x$prices),"."))
    else
        print(paste0("Your order is ", paste0(x$names, collapse=", "),". The total price is
        $",sum(x$prices),"."))
    invisible(x)
}
# 4. Now, you need to create a combine operator for the class `shakeshack_order`.
# For example, `c(o, o2)` should equal
# `shakeshack_order(names = c('shack burger', 'fries', 'fries', 'coke'), prices = c(5, 2, 2, NA))`.
# Similarly as for `sum.shakeshack_order`, the `...` argument of `c.shakeshack_order`
# can be captured using `list(...)`.
# Do NOT use a `for`, `while` or `repeat` loop!
# (Hint: a nice solution could use a combination of `map2` and `reduce`.)
#'
## Do not modify this line!
o <- shakeshack_order(c("shack burger", "fries"), c(5, 2))
o2 <- shakeshack_order(c("fries", "coke"), c(2, NA))
str(o)
c.shakeshack_order<-function(...){
```

```
        data<-list(...)
        names<-reduce(map(data,~.x$names),c)
        prices<-reduce(map(data,~.x$prices),c)
        new_shakeshack_order(names,prices)
}


# HW4: OOP_account
#'
# In this exercise, we will create a class `account`.
#'
# 1. Create a constructor `new_account(number)` that takes in a length 2 numeric
#    vector and initiates an object of class `account` using `structure`.
#    Class `account` should have an attribute `units` which is always the character vector
`c('USD', 'EUR')`.
#    The value should be the amount of money in each currency.
#    Don't forget to use `stopifnot` to check if `number` is numeric or not and if the length of
number is different from 2.
#    For example, `new_account(c(5, 0))` will create an account with 5 USD.
#    `new_account(c(0, 5))` will create an account with 5 EUR.
#    `new_account(c(5, 10))` will create an account with 5 USD and 10 EUR.
## Do not modify this line!
new_account<-function(number){
        stopifnot(is.numeric(number)&length(number)==2)
        structure(number,class="account",units=c('USD', 'EUR'))
}


# 2. Use `pmatch` to reate a function `get_unit_index(units)` to get the index of input `units`
in `c('USD, 'EUR')`.
#    You are supposed to use this function in the following questions.
#    For example, `get_unit_index('EUR') = 2`. `get_unit_index('US') = 1`.
#    `get_unit_index('U','E') = c(1, 2)`.
## Do not modify this line!
get_unit_index<-function(units){
        pmatch(units,c('USD', 'EUR'))
}

# 3. Create a helper function `account(number, units)` that takes in a scalar or vector
`number`
#    with corresponding `units` which initiates an object of class `account`.
#    `units` might include `EUR`, `USD`.
#    For example, `account(5, 'USD')` will create an account with 5 USD.
#    `account(5, 'EUR')` will create an account with 5 EUR.
#    `account(c(5, 10), c('EUR', 'USD'))` will create an account with 5 EUR and 10 USD.
#    Create an account with 100 USD and 100 EUR. Save it as `my_account`.
## Do not modify this line!
account<-function(number,units){
        data<-rep(0,length(c('EUR','USD')))
        data[get_unit_index(units)]<-number
        new_account(data)
}
```

```
my_account<-account(c(100,100),c('EUR','USD'))

# 4. Creat two generic functions `deposit(account, number, units)` and `withdraw(account,
number, units)`
#    that takes in a scalar or vector `number` with corresponding `units`.
#    `units` might include `EUR`, `USD`.
#    Methods `deposit.account` and `withdraw.account` should return an object of class
`account` with correct amounts.
#    Deposit 50 USD and withdraw 50 EUR for `my_account`.
## Do not modify this line!
deposit<-function(account,number,units){
        UseMethod("deposit")
}
withdraw<-function(account,number,units){
        UseMethod("withdraw")
}
deposit.account<-function(account,number,units){
        data<-rep(0,length(c('EUR','USD')))
        data[get_unit_index(units)]<-number
        data<-account+data
        new_account(data)
}
withdraw.account<-function(account,number,units){
        data<-rep(0,length(c('EUR','USD')))
        data[get_unit_index(units)]<-number
        data<-account-data
        new_account(data)
}
my_account<-deposit(my_account,50,"USD")
my_account<-withdraw(my_account,50,"EUR")
# 5. Write a new `summary` method for class `account`. `summary(account, units)` should
return a named vector
#    which has the same length and names as `units`.
#    The reurn value should be the total amount of the `account` in `units`.
#    For simplicity, 1 EUR = 1.1 USD.
#    For example, `summary(my_account, c('USD', 'EUR'))` should return a named vector with
values `c(150, 50)`.
#    `summary(my_account, 'USD')` should return a named vector with value `205`.
## Do not modify this line!
summary.account<-function(account,units){
        index<-get_unit_index(units)
        if(length(index)==1){
                amounts<-as.numeric(account[1]+account[2]*1.1)
                amounts<-ifelse(units=="USD",amounts,amounts/1.1)
                names(amounts)<-units
        }
        else{
                amounts<-as.numeric(account(account,units))
                names(amounts)<-units
        }
        amounts
}
```

```
# 6. Now let's consider a subclass `minimum_balance_account` which should maintain a
pre-determined minimum balance.
#    Create a constructor `new_minimum_balance_account(number, minimum)` that takes in
legnth 2 numeric vectors and
#    initiates an object of subclass `minimum_balance_account` using `structure`.
#    Subclass `minimum_balance_account` should have an attribute `units` which is always
#    the character vector `c('USD', 'EUR')` and an attribute `minimum` which is a numeric
vector.
#    The value should be the amount of money in each currency.
#    Don't forget to use `sopifnot` to check if `number` and `minimum` are numeric or not.
#    For example, `new_minimum_balance_account(c(5, 0), c(1, 0))` will create an account
#    with 5 USD and minimum balance 1 USD.
## Do not modify this line!
new_minimum_balance_account<-function(number, minimum){
        stopifnot(is.numeric(number)&is.numeric(minimum))
        stopifnot(length(number)==2&length(minimum)==2)
        structure(
                number,
                class=c("minimum_balance_account","account"),
                units=c('USD', 'EUR'),
                minimum=minimum
                        )
}

# 7. Create a helper function `minimum_balance_account(number, units, minimum,
minimum_units)`
#    that takes in a scalar or vector `number` and `minimum`
#    with corresponding `units` and `minimum_units` which initiates an object of subclass
`minimum_balance_account`.
#    `units` and `minimum_units` might include `EUR`, `USD`.
#    For example, `minimum_balance_account(5, 'USD', 1, 'USD')` will create an account
#    with 5 USD and minimal balance 1 USD.
## Do not modify this line!
minimum_balance_account<-function(number,units,minimum,minimum_units){
        stopifnot(is.numeric(number)&is.numeric(minimum))
        stopifnot(is.numeric(get_unit_index(units))&is.numeric(get_unit_index(minimum_unit
        s)))
        data1<-rep(0,length(c('EUR','USD')))
        data1[get_unit_index(units)]<-number
        data2<-rep(0,length(c('EUR','USD')))
        data2[get_unit_index(minimum_units)]<-minimum
        new_minimum_balance_account(data1,data2)
}


# 8. Add method for `withdraw` and modify `deposit.account` if needed such that:
#    a. The returned value should be the same class of input `account`.
#    b. Function `deposit` can accpet `minimum_balance_account` and return correct object.
#    c. If withdraw will cause the balance lower than the minimum balance,
#       report an error 'Minimum balance must be maintained.'.
#    d. Use `withdraw.account` in `withdraw.minimum_balance_account`.
```

```
## Do not modify this line!
c1<-new_minimum_balance_account(c(5, 0), c(1, 0))
# deposit.minimum_balance_account<-function(account,number,units){
#         data<-rep(0,length(c('EUR','USD')))
#         data[get_unit_index(units)]<-number
#         data<-account+data
#         new_minimum_balance_account(data,attr(account,"minimum"))
# }
c1
deposit(c1,10,"EUR")
withdraw.minimum_balance_account<-function(account,number,units){
      tmp_account<-withdraw.account(account,number,units)
      data<-tmp_account
      if(data[1]<attr(account,"minimum")[1]|data[2]<attr(account,"minimum")[2])
            stop('Minimum balance must be maintained.')
      new_minimum_balance_account(data,attr(account,"minimum"))
}
withdraw(c1,2,"USD")
```

2019年10月18日 12:26

HW5

下列的函数都有第一个未显出argument（由于使用了pipe >%>），即tibble本身。

1. filter取符合里面的row: filter(length>0.6)

   select选符合名字的col: select('sex', 'diameter', 'height', 'rings')

   arrange 按里面的argu排序，decs表降序，多个表示先依次subrow排序：
   arrange(sex,desc(rings))

   summarize()用于collapse a df to a single row,单独作用不大，但是可以与group_by 配合使用，那就成了对每个group summarize:
   group_by(sex)%>%
   summarise(weight_mean=mean(shucked_weight),weight_max=max(shucked_weight),
   weight_min=min(shucked_weight)) //=前是名字，后面是函数

   rename(期待的名字=原名字)

   mutate()在df末尾加上想要的列，transmute()只保留后来计算的列

2. select 可以用A:B选之前所有列（inclusive），-B表示除去B列

   用pivot_longer，col names成为了新的cases列中的值，col values 成了新的comments 列的值：

   ```
   pivot_longer(Q13:Q13_F,names_to = "cases",values_to =
   "comments",values_drop_na = TRUE)
   ```

   separate 根据sep 一列分成多列：
   separate(cases,into=c("case_name","category",sep='_'))

   把NA替代为A：replace_na(list(category='A'))

   unite，separate的反操作

3. filter(region %in% c('East Asia & Pacific','North America'))可以用这种易读的形式

   gather()第一个arg为转换的col,所有col name成为新col var的值，col value成为score的值

   ```
   act_gathered<-activities%>%
     gather(-c(id,trt),key="var",value="score")
   ```

   找到NA值位置：col_ix<-which(is.na(act_spread),arr.ind = TRUE)[2]//arr.ind = TRUE表示以array 的坐标的形式给出，否则为一个int,表示整个没有dim时的list的index值。

   利用fill来填相应的col中的NA: act_filled<-fill(act_spread,col_ix,.direction='up')

4. 用某一列mean来填其中的NA

   ```
   airquality%>%
     replace_na(list(Ozone=mean(airquality$Ozone,na.rm=TRUE)))
   ```

   drop_na(), drop 某一列中的NA

   使用case_when根据另一列的值改变某一列的值

   ```
   airquality_filtered%>%
   ```

```
 mutate(Wind=case_when(Wind>=1&Wind<9~7.4,Wind>=9&Wind<20~
11.5,TRUE~21))
```

5. 除掉所有NA

```
warpbreaks%>%
 filter_all(all_vars(!is.na(.)))//all_vars中需要用 .
```

pivot_wider,names_from表示tension里的值成为了新col name, values_from 表示break里的值成为了新col value. 但如果新values数量不对，比如这里wool 和tension一个组合对应多个break values,那么需要对其进行计算，values_fn = 表示需要什么计算。

```
t1%>%
 pivot_wider(names_from = tension,values_from = breaks,values_fn = list(breaks
= sum))
```

6. pivot_longer选col时可以用starts_with 筛选。names_prefix表示新的week列中值去掉相应的prefix

```
pivot_longer(cols = starts_with('wk'),names_to = "week",names_prefix = 'wk',
            values_to = "rank",values_drop_na=TRUE)
t2%>%
 group_by(artist,track,date.entered)%>%
 mutate(rank_increase = lag(rank)-rank)%>%
```

//这个计算会限制在group中，故而不会有后一个group使用前一个group值的情况

```
 filter(!is.na(rank_increase))%>%
 summarize(highest_rank_increase=max(rank_increase))
```

7. 对于某些列使用某些方程后的summarize,所有方程都会对，每个列使用

```
mobility%>%
 group_by(State)%>%
 summarize_at(vars(Commute,Mobility), funs(mean, sd))
```

8. 用一个字符代替string中所有想被替代的字符

```
library(stringr)
names(trips)<-str_replace_all(names(trips),' ','_')
```

这个包中另一个方程可以结合filter找到所有包含某一substring的row

```
broadway<-trips%>%

filter(str_detect(start_station_name,'Broadway')|str_detect(end_station_name,'
Broadway'))
```

head(df, n)表示只保留前n行

9. summarize中，如果用function list需要single argument的函数（因为写不了exrtra arg），但有的函数必然有，则可以自己写一个:

```
library(stats)
quantile_1<-function(x)(
 quantile(x,0.25)
)
quantile_3<-function(x)(
 quantile(x,0.75)
)
summary_stats<-iris%>%
```

```
  summarize_if(is.numeric,list('Min.'=min,'1st Qu.'=quantile_
1,'Median'=median,'Mean'=mean,
                                '3rd Qu.' = quantile_3,'Max.'=max))
```

pivot_longer需要选择所有列时，使用everything()

10. mutate根据某一列的值取值

```
fu_tidy%>%
  mutate(score=if_else(team_type=='home_team',home_score,away_score))
```

HW6

```
# HW6: ggplot1
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# The purpose of this task is to get familiar with ggplot.
# Throughout the exercise, please use `theme_light()` as your base theme.
# For graphs with titles, make the format as
# `theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))`.
#'
# 1. Load the `ggplot2`, `readr` and `tidyr` packages.
#   Use `read_csv` to load the `abalone.csv` data set from `data` folder and
#   use `drop_na()` to drop NAs, then assign it to a tibble `abalone`.
#   To check your solution, `abalone` prints to:
#   # A tibble: 4,077 x 9
#   sex   length diameter height whole_weight shucked_weight viscera_weight shell_weight
#   <chr>  <dbl>    <dbl>  <dbl>       <dbl>         <dbl>         <dbl>       <dbl>
#   1 M      0.455    0.365  0.095       0.514        0.224         0.101       0.15
#   2 M      0.35     0.265  0.09        0.226        0.0995        0.0485      0.07
#   3 F      0.53     0.42   0.135       0.677        0.256         0.142       0.21
#   4 M      0.44     0.365  0.125       0.516        0.216         0.114       0.155
#   5 I      0.33     0.255  0.08        0.205        0.0895        0.0395      0.055
#   6 I      0.425    0.3    0.095       0.352        0.141         0.0775      0.12
#   7 F      0.53     0.415  0.15        0.778        0.237         0.142       0.33
#   8 F      0.545    0.425  0.125       0.768        0.294         0.150       0.26
#   9 M      0.475    0.37   0.125       0.509        0.216         0.112       0.165
#   10 F     0.55     0.44   0.15        0.894        0.314         0.151       0.32
#   # … with 4,067 more rows, and 1 more variable: rings <dbl>
#'
## Do not modify this line!
library(ggplot2)
library(readr)
library(tidyr)
abalone<-read_csv("data/abalone.csv")%>%drop_na()%>%as_tibble()

# 2. Draw a density plot of `rings`, colored by `sex`.
#   To do this, you can use:
#   - `stat_density()` to draw a bar plot of `rings` and set `color` as `sex`. Set `geom` as `line`
#   and `position` as `identity`.
#   - `labs()` to name the title as:
#   `Male and female abalone have similar modes in rings around 10,\n`
#   `while infant abalone have lower mode in rings around 8`,
#   name the x-axis as: `"Number of rings"`,
```

```
#    name the y-axis as: `"Density"`.
#    - `scale_color_discrete` to name the legend as `"Sex"`.
#    Store the plot into a variable `g1`.
## Do not modify this line!
g1<-ggplot(data=abalone)+stat_density(mapping =
aes(rings,color=sex),geom="line",position="identity")+
  labs(title = "Male and female abalone have similar modes in rings around 10,\nwhile infant
abalone have lower mode in rings around 8",
       x = "Number of rings",
       y = "Density"
       )+
  scale_color_discrete(name="Sex")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))#标题居中
?theme_light
# 3. Draw a histogram of `diameter` with `binwidth` = 0.05,
#    filled by `sex` and also faceted by `sex` in grids.
#    To do this, you can use:
#    - `geom_histogram()` to draw a histogram of `diameter` with `binwidth` as 0.05,
#     and set `fill` also as `sex`.
#    - `facet_grid()` to facet the graph by `sex`.
#    - `labs()` to name the title as:
#    `"Female and male abalone tend to have more left skewed distributions of diameter,\n`
#    `while the infant abalone\'s diameter distribution looks more normal"`,
#    name the x-axis as `"Diameter of abalone (mm)"`,
#    name the y-axis as `"Count of abalone"`.
#    - `scale_fill_discrete()` to name the legend as `"Sex"`.
#    Store the plot into a variable `g2`.
## Do not modify this line!
g2<-ggplot(data=abalone)+geom_histogram(mapping = aes(diameter,fill=sex),binwidth =
0.05)+
  facet_grid(~sex)+
  labs(
    title = "Female and male abalone tend to have more left skewed distributions of diameter,
\nwhile the infant abalone\'s diameter distribution looks more normal",
    x="Diameter of abalone (mm)",
    y="Count of abalone"
  )+
  scale_fill_discrete(name="Sex")+
  theme_light()


# 4. Draw a boxplot of `diameter` vs. `sex`. The ordering of boxes from left to right
#    should be in descending order of median of `diameter`.
#    To do this, you can use:
#    - `geom_boxplot()` to draw a boxplot of `diameter` vs. `sex`(hint: use `reorder` to
#    organize the order on the x-axis).
#    - `labs()` to name the title as:
#    `"Male and female abalone have similar median,\n`
#    `while infant abalone have smaller median"`,
#    name the x-axis as `"Sex"`,
#    name the y-axis as `"Diameter of abalone (mm)"`.
```

```
#    Store the plot into a variable `g3`.
## Do not modify this line!
g3<-ggplot(data=abalone)+geom_boxplot(mapping = aes(x=reorder(sex,-
diameter,median),y=diameter))+
 labs(title="Male and female abalone have similar median,\nwhile infant abalone have
smaller median",
      x="Sex",y="Diameter of abalone (mm)"
      )+
 theme_light()+
 theme(plot.title = element_text(hjust = 0.5))
#reorder 第一个arg是要排序的目标，第二个表按什么排
?reorder
# 5. Draw a point plot of `diameter` vs. `rings`, faceted by `sex` into wraps.
#    Draw a smooth curve that passes through points using `loess` method.
#    Name the title as `"Diameter is increasing in rings"`,
#    subtitle as `"But the increase mostly stops around 10 rings"`.
#    To do this, you can use:
#    - `geom_point()` to draw a point plot of `diameter` vs. `rings`.
#    - `geom_smooth()` to draw a smooth curve using method as `loess`.
#    - `facet_wrap()` to facet the graph by `sex`.
#    - `labs()` to name the title as `"Diameter is increasing in rings"`, subtitle as
#    `"But the increase mostly stops around 10 rings"`,
#    name the x-axis as `"Number of rings"`,
#    name the y-axis as `"Diameter of abalone (mm)"`.
#    Store the plot into a variable `g4`.
## Do not modify this line!

g4<-ggplot(data=abalone,mapping = aes(x=rings,y=diameter))+geom_point()+
 geom_smooth(method="loess",color="red")+
 facet_wrap(~sex)+
 labs(
   title="Diameter is increasing in rings",
   subtitle = "But the increase mostly stops around 10 rings",
   x="Number of rings",
   y="Diameter of abalone (mm)"
 )+
 theme_light()+
 theme(plot.title = element_text(hjust = 0.5))+
 theme(plot.subtitle = element_text(hjust = 0.5))

# 6. Draw a violin plot of `whole_weight` vs. `length`, faceted by `sex` in grids.
#    To do this, you can use:
#    - `geom_violin()` to draw a violin plot of `whole_weight` vs. `length`.
#    - `facet_wrap()` to facet the graph by `sex`.
#    - `labs()` to name the title as:
#    `"Male and female abalone have similar distribution in whole weight and length,\n`
#    `while infant ablone's distribution skew to the higher weights"`,
#    name the x-axis as `"Length of abalone (mm)"`,
#    name the y-axis as `"Whole weight of abalone (grams)"`.
#    Store the plot into a variable `g5`.
#'
#'
```

```
## Do not modify this line!
g5<-ggplot(data=abalone)+geom_violin(mapping = aes(x=length,y=whole_weight))+
  facet_wrap(~sex)+
  labs(
    title = "Male and female abalone have similar distribution in whole weight and length,
       while infant ablone's distribution skew to the higher weights",
    x = "Length of abalone (mm)",
    y = "Whole weight of abalone (grams)"
  )+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

# HW6: ggplot2
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# The purpose of this task is to get familiar with ggplot.
# Throughout the exercise, please use `theme_light()` as your base theme.
# For graphs with titles, make the format as
# `theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))`.
#'
# 1. Load the `ggplot2`, `dplyr`, `purrr`, `ggrepel` and `readr` packages.
#    Use `read_csv()` to load the `autompg.csv` data set from `data` folder and
#    assign it to a tibble `autompg`.
#    Create a tibble `efficiency_by_manufacturer` of dimension 398 x 11 on the base of
`autompg`.
#    The dataset should transfer the `model_year` into a factor variable. It should also
#    Create a new column called `car_made` that extracts the car manufacturer from
`car_name`.
#    The dataset should Create a new column called `efficiency`.
#    The formula of `efficiency` is `mpg / weight`.
#    To do that, you can use:
#    - `mutate()` and `as.factor()` to transfer the `model_year` into a factor variable.
#    - `map_chr()` and `strsplit` to extract the car manufacturer from `car_name`.
#    - `mutate()` to create a new column called `car_made` and store the list of car
manufacturer names.
#    - `mutate()` to build a new column called `efficiency` and use the formula above to assign
values.
#    To check your solution, `efficiency_by_manufacturer` prints to:
#    # A tibble: 398 x 11
#    mpg cylinders displacement horsepower weight acceleration model_year origin
car_name
#    <dbl>    <dbl>       <dbl> <chr>      <dbl>       <dbl> <fct>      <dbl> <chr>
#    1    18       8         307 130         3504        12   70           1 chevrol…
#    2    15       8         350 165         3693        11.5 70           1 buick s…
#    3    18       8         318 150         3436        11   70           1 plymout…
#    4    16       8         304 150         3433        12   70           1 amc reb…
#    5    17       8         302 140         3449        10.5 70           1 ford to…
```

```
#   6    15       8        429 198        4341       10   70           1 ford ga…
#   7    14       8        454 220        4354        9   70           1 chevrol…
#   8    14       8        440 215        4312        8.5 70           1 plymout…
#   9    14       8        455 225        4425       10   70           1 pontiac…
#   10    15       8         390 190        3850        8.5 70            1 amc amb…
#   # … with 388 more rows, and 2 more variables: car_made <chr>, efficiency <dbl>
## Do not modify this line!
library(ggplot2)
library(dplyr)
library(ggrepel)
library(readr)
library(purrr)
autompg<-read_csv("data/autompg.csv")
efficiency_by_manufacturer<-autompg%>%
  mutate(model_year=as.factor(model_year))%>%
  mutate(car_made=map_chr(car_name,function(x) strsplit(x,split = " ")[[1]][1]))%>%
  mutate(efficiency=mpg/weight)

# 2. Draw a horizontal boxplot of `efficiency` vs. `car_made`. The ordering of boxes from top
to bottom
#   should be in descending order of median of `efficiency`.
#   To do this, you can use:
#   - `geom_boxplot()` to draw a boxplot of `efficiency` vs. `car_made`(hint: use `reorder` to
#   organize the order on the x-axis).
#   - `labs()` to name the title as:
#   `"Cars made by European and Japanese manufacturers are generally more efficient"`,
#   subtitle as:
#   `"Cars made by American manufacturers are less efficient"`,
#   name the x-axis as `"Car manufacturer"`,
#   name the y-axis as `"Efficiency (mpg / kg)"`.
#   - `coord_flip()` to flip x and y.
#   Store the plot into a variable `g1`.
## Do not modify this line!
g1<-ggplot(efficiency_by_manufacturer)+geom_boxplot(mapping =
aes(x=reorder(car_made,efficiency,median),y=efficiency))+
  labs(
    title = "Cars made by European and Japanese manufacturers are generally more efficient",
    subtitle = "Cars made by American manufacturers are less efficient",
    x="Car manufacturer",
    y="Efficiency (mpg / kg)"
  )+coord_flip()+theme_light()


# 3. Draw a point plot of `mpg` vs. `weight`, colored by `model_year`.
#   To do this, you can use:
#   - `geom_point()` to draw a point plot of `diameter` vs. `rings` with colored by
`model_year`.
#   Set `size` as 2, `shape` as 1 and `stroke` as 1.25,
#   - `labs()` to name the x-axis as `"Weight of a car (kg)"`,
#   name the y-axis as `"Miles per gallon"`.
#   - `scale_color_discrete()` to name the legend as `"Model year of a car \n(1970 - 1982)"`.
#   Store the plot into a variable `g2`.
```

```
## Do not modify this line!
g2<-ggplot(efficiency_by_manufacturer,mapping = aes(x=weight,y=mpg))+
 geom_point(mapping = aes(color=model_year),size=2,shape=1,stroke=1.25)+
 scale_color_discrete("Model year of a car
                      (1970 - 1982)")+
 labs(
  x="Weight of a car (kg)",
  y="Miles per gallon"
 )+
 theme_light()


# 4. Draw a smooth curve on the `g2` using `loess` funnction.
#    To do this, you can use:
#    - `geom_smooth()` to draw a smooth curve with `linetype` as `r2`. Set method as `loess`,
#    `se` as `TRUE` and `color` as `red`.
#    - `guides()` to remove the legend of `linetype`.
#    Store the plot into a variable `g3`.
## Do not modify this line!
g3<-g2+geom_smooth(method="loess",se=TRUE,color="red")+guides(legend=NULL)


# 5. Build a tibble `best_efficiency_by_manufacturer` of dimension 37 x 11.
#    The dataset extracts the largest `efficiency` of each manufacturer from
`efficiency_by_manufacturer`.
#    To do this, you can use:
#    - `group_by()` to group the `car_made`
#    - `filter()` to filter out the rows with max `efficiency` of each car manufacturer.
#    To check your solution, `best_efficiency_by_manufacturer` prints to:
#    # A tibble: 37 x 11
#    # Groups:   car_made [37]
#    mpg cylinders displacement horsepower weight acceleration model_year origin
car_name
#    <dbl>    <dbl>       <dbl> <chr>      <dbl>       <dbl> <fct>      <dbl> <chr>
#    1    25        4         104 95          2375        17.5 70             2 saab 99e
#    2    26        4         121 113         2234        12.5 70             2 bmw 2002
#    3     9        8         304 193         4732        18.5 70             1 hi 1200d
#    4    28        4         116 90          2123        14   71             2 opel 19…
#    5    30        4          79 70          2074        19.5 71             2 peugeot…
#    6    35        4          72 69          1613        18   71             3 datsun …
#    7    23        4         120 97          2506        14.5 72             3 toyouta…
#    8    16        6         250 105         3897        18.5 75             1 chevroe…
#    9    25        4         140 92          2572        14.9 76             1 capri ii
#    10    30        4         111 80          2155        14.8 77             1 buick o…
#    # … with 27 more rows, and 2 more variables: car_made <chr>, efficiency <dbl>
## Do not modify this line!
best_efficiency_by_manufacturer<-efficiency_by_manufacturer%>%
 group_by(car_made)%>%
 filter(efficiency==max(efficiency))


# 6. Draw manufacturer names onto `g3` and
```

```
#    - add a title onto the graph and name title as:
#    `"Mpg is decreasing in weights"`.
#    - name the subtitle as:
#    `"How are cars performing in best efficiency by different car manufacturers?"`.
#    - move all the legends on the top in one row.
#    To do this, you can use:
#    - `geom_text_repel()` to draw every `car_made` name onto the graph from
`best_efficiency_by_manufacturer`.
#    - `labs()` to name the title as:
#    `"Mpg is decreasing in weights"`, subtitle as:
#    `"How are cars performing in best efficiency by different car manufacturers?"`.
#    - `guides()` and `guide_legend()` to make all legend in one row.
#    - `theme()` to set `legend.position` as `top`, `legend.direction` as `horizontal` and
#    `legend.justification` as 0.1.
#'
## Do not modify this line!
g4<-g3+
  geom_text_repel(aes(x = weight,y = mpg,label = car_made),data =
best_efficiency_by_manufacturer)+
  labs(title ="Mpg is decreasing in weights",
       subtitle = "How are cars performing in best efficiency by different car manufacturers?")+
  guides(color = guide_legend(nrow = 1),linetype = FALSE)+
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5),
        legend.position = "top",legend.direction = "horizontal",legend.justification = 0.1)
#g3+meom_text_repel会出错，但是没关系

# HW6: ggplot_iris
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# The purpose of this task is to get familiar with ggplot.
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
#    - Do not print the plot.
# Let's consider `diamonds` dataset in the R base environment.
#'
# 1. Load the `ggplot2` and `scales` package.
#    Plot a stacking histogram for Price. Fill in the histogram with `cut` i.e.
#    each cut reprent a different color in the histogram.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_histogram()` to plot a histogram for Price, such that
#      - `binwidth = 500` to set the binwidth
#    - `scale_x_continuous(label=comma)` to change x-axis scale
#    - `scale_y_continuous(label=comma)` to change y-axis scale
#    - `labs()` to format the labels such that:
#      - `title = "Counts go down as price goes up"`
#      - `subtitle = "Ideal cuts account for nearly half of the diamonds"`
```

```
#     - `x = "Price (USD)"`
#     - `y = "Count (n)"`
#    Store the plot into a variable `g1`.
## Do not modify this line!
library(ggplot2)
library(scales)
g1<-ggplot(data=diamonds,mapping=aes(x=price,fill=cut))+geom_histogram(binwidth =
500)+
  scale_x_continuous(label=comma)+
  scale_y_continuous(label=comma)+
  labs(title = "Counts go down as price goes up",
       subtitle = "Ideal cuts account for nearly half of the diamonds",
       x = "Price (USD)",
       y = "Count (n)"
       )+
  theme_light()

# 2. We can see that if we plot the count using different colors in the same
#    bin, it is hard to compare between different color groups(i.e. cut).
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_freqpoly()` to generate the frequency polygons for price
#    split by different cuts(split in color as well)
#      -`binwidth = 500` to set the binwidth
#    - `labs()` to format the labels such that:
#      - `title = "Max count is reached when price is around 1000 USD"`
#      - `x = "Price (USD)"`
#      - `y = "Count (n)"`
#    Store the plot into a variable `g2`.
## Do not modify this line!
g2<-ggplot(data=diamonds,mapping=aes(x=price,color=cut))+
  geom_freqpoly(binwidth=500)+
  labs(title = "Max count is reached when price is around 1000 USD",
       x = "Price (USD)",
       y = "Count (n)"
  )+
  theme_light()

# 3. Generate a scatter plot of Price versus Carat,
#    split by Cut (each color represents one cut).
#    Add smoothing curve to the plot for each cut category.
#    Do not include the confidence interval.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_point()` to add scatter plot
#    - `geom_smooth()` to add a smoothing regression curve such that
#      - `se = FALSE`, to not include the confidence interval
#    - `labs()` to format the labels such that:
#      - `title = "In general price go up as carat goes up"`
#      - `subtitle = "Ideal cuts have highest increasing rate"`
#      - `x = "Carat"`
#      - `y = "Price (USD)"`
```

```
#    Store the plot into a variable `g3`.
## Do not modify this line!

g3<-ggplot(data=diamonds,mapping=aes(x=carat,y=price,color=cut))+
  geom_point()+
  geom_smooth(se=FALSE)+
  labs(
    title = "In general price go up as carat goes up",
    subtitle = "Ideal cuts have highest increasing rate",
    x = "Carat",
    y = "Price (USD)"
  )+
  theme_light()

# 4. Create a grid of plots using, Clarity on x-axis and Color
#    on y-axis for the entire grid.
#    For each plot in the grid, plot Price versus
#    Carat and split by Cut, with each color representing one cut.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_point()` to add scatter plot
#    - `facet_grid()` to create the grid
#    - `labs()` to format the labels such that:
#      - `title = "In general price go up as carat goes up"`
#      - `subtitle = "I1 clarity seems to have the lowest increasing rate"`
#      - `x = "Carat"`
#      - `y = "Price (USD)"`
#    Store your new plot into a variable `g4`.
## Do not modify this line!
g4<-ggplot(data=diamonds,mapping=aes(x=carat,y=price,color=cut))+geom_point()+
  facet_grid(color~clarity)+
  labs(
    title = "In general price go up as carat goes up",
    subtitle = "I1 clarity seems to have the lowest increasing rate",
    x = "Carat",
    y = "Price (USD)"
  )+
  theme_light()


# 5. Generate a boxplot for price for each color,
#    i.e. color on x-axis, then generate boxplot for price within each color.
#    scale the y axis to log10 scale.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_boxplot()` to add box plot for price within each color group
#    - `scale_y_log10()` to rescale the y-axis
#    - `labs()` to format the labels such that:
#      - `title = "Color J has the highest price"`
#      - `x = "Color"`
#      - `y = "Price (USD)"`
#    Store your new plot into a variable `g5`.
```

```
## Do not modify this line!
g5<-ggplot(data = diamonds,mapping = aes(x=color,y=price))+
  geom_boxplot()+
  scale_y_log10()+
  labs(
    title = "Color J has the highest price",
    x = "Color",
    y = "Price (USD)"
  )+
  theme_light()


# 6. Now generate a similar plot with price against color, but change the box
#    plot into a violin plot. This time, also generate a facet grid split by Clarity.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. Set its arguments `data` and `mapping`
#    - `geom_violin()` to add violin plot
#    - `scale_y_log10()` to rescale the y-axis
#    - `facet_wrap()` to generate the grid for each clarity
#    - `labs()` to format the labels such that:
#      - `title = "Diffrent clarities and colors show various distribution in price"`
#      - `x = "Color"`
#      - `y = "Price (USD)"`
#    Store your new plot into a variable `g6`.
#'
## Do not modify this line!
g6<-ggplot(data=diamonds,mapping = aes(x=color,y=price))+
  geom_violin()+
  scale_y_log10()+
  facet_wrap(~clarity)+
  labs(
    title = "Diffrent clarities and colors show various distribution in price",
    x = "Color",
    y = "Price (USD)"
  )+
  theme_light()

# HW6: ggplot_iris
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# The purpose of this task is to construct a complex plot using ggplot.
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
#    - Do not print the plot.
# Let's consider `iris` dataset in the R base environment.
#'
# 1. Load the `ggplot2` package.
```

```
#   Plot Petal Length versus Sepal Length split by Species, i.e. different
#   colors representing different Species.
#   To do that, use:
#   - `ggplot()` to initialize a ggplot object. Set its argument `data`.
#   - `geom_point()` to generate the scatter plot. Set its arguments
#   `mapping` and 'color' to plot Petal Length versus Sepal Length split by Species.
#   - `labs()` to format the labels such that:
#     - `title = "Sepal length by petal length"`
#     - `x = "Sepal Length (cm)"`
#     - `y = "Petal Length (cm)"`
#   Store the plot into a variable `g1`.
## Do not modify this line!
library(ggplot2)
g1<-ggplot(data=iris,mapping = aes(x=Sepal.Length,y=Petal.Length,color=iris
$Species))+geom_point()+
 labs(
   title = "Sepal length by petal length",
   x = "Sepal Length (cm)",
   y = "Petal Length (cm)"
 )+
 theme_light()

# 2. We can see that the instead of in the middle of the plot, the title is
#   on the left corner, and the title of the label is `iris$Species`
#   instead of `Sepecies`.
#   Change the title to the middle of the plot and change the title of the label.
#   To do that, use:
#   - `theme()` to change the title to the middle of the plot. Set its argument
#   `plot.title` using `element_text()`
#   - `scale_color_hue()` to change the title of the label.
#   All the changes are made to `g1`.
#   Store your new plot into a variable `g2`.
## Do not modify this line!
g2<-g1+theme(plot.title = element_text(hjust=0.5))+
 scale_color_hue(name="Species")


# 3. Now we want to see the linear relationship between Sepal length and
#   Petal length.
#   To do that, use:
#   - `geom_smooth()` to add to `g2` a linear regression
#   line between Petal length and Sepal length for different Sepecies, such that
#     - `method = "lm"`, to add a linear regression line
#     - `se = FALSE`, to not include the confidence interval for the regression line
#     - `fullrange = TRUE`, to lot the regression line on full range of x-axis.
#   - `labs()` to format the labels such that:
#     - `title = "Petal length is increasing in sepal length"`
#     - `subtitle="Except for the setosa species"`
#   - `theme()` to change the subtitle to the middle of the plot as well. Set its argument
#   `plot.subtitle` using `element_text()`
#   Store your new plot into a variable `g3`.
## Do not modify this line!
```

```
g3<-g2+geom_smooth(method = "lm",se=FALSE,fullrange=TRUE)+
labs(
  title = "Petal length is increasing in sepal length",
  subtitle="Except for the setosa species"
)+
  theme_light()+  theme(plot.title = element_text(hjust=0.5),plot.subtitle =
element_text(hjust=0.5))


# 4. Let's say we want to emphasize a point, say the 15th point in the iris
#    dataset.
#    To do that, use:
#    - `geom_point()` to add this point to your `g3`,
#      - `pch = 17` , to plot the points as a triangle
#      - `col = "black"`, to change the color into black
#      - `cex = 3`, to change the point size to 3
#    Store your new plot into a variable `g4`.
## Do not modify this line!
g4<-g3+geom_point(data=iris[15,],pch=17,col="black",cex=3)


# 5. We want to add a text to annotate the previous point.
#    To do that, use:
#    - `geom_text()` to add the coordinates for the previouly drawn point
#    in question4 onto your `g4`, such that
#    - `label = "This is the iris with the longest sepal\n among the setosa species.")`, to change
the output of the text
#    - `col = "black"`, to change the color into black
#    - `nudge_x = 1` to make the  x position is 1 greater than the original point
#    Store your new plot into a variable `g5`.
#'
## Do not modify this line!
g5<-g4+geom_text(mapping=aes(iris$Sepal.Length[15],iris$Petal.Length[15]),label = "This is
the iris with the longest sepal\n among the setosa species.",
col = "black",
nudge_x = 1
)
#注意mapping选择的点位


# HW6: ggplot_nations
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# In this exercise, you will familiarize yourself with data visualization using
# ggplot.
#'
# Throughout the exercise:
```

```
#   - Use `theme_light()` for the plots.
#   - Set `alpha = 0.5` and `stoke = 0` for `geom_point()` to avoid overplotting.
#   - For good practice, make sure every graph is complete. Follow the
#     instructions for labelling exactly to get the correct answer.
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#'
# 1. Load the package `tidyverse`.
#   Use `read_csv()` to read the dataset `nations.csv` (located in directory
#   `data/`) along with:
#     - `drop_na()` to drop all missing values.
#     - `mutate()` and `fct_reorder()` to make the column `region` sorted by
#       median of `life_expect`.
#   Store the corresponding dataframe into a tibble `nations`.
#   To check your result, `nation` prints to:
#   # A tibble: 4,635 x 11
#    iso2c iso3c country  year gdp_percap life_expect population birth_rate
#    <chr> <chr> <chr>   <int>     <dbl>       <dbl>     <int>      <dbl>
#    1 AE    ARE   United…  1994     71493.        72.7   2328686       21.3
#    2 AE    ARE   United…  2007     74698.        75.8   6044067       13
#    3 AE    ARE   United…  1996     75977.        73.2   2571020       19.2
#    4 AE    ARE   United…  1993     69085.        72.4   2207405       22.4
#    5 AE    ARE   United…  2005     82206.        75.4   4579562       14.0
#    6 AE    ARE   United…  1991     70642.        71.8   1970026       24.8
#    7 AE    ARE   United…  1992     70496.        72.1   2086639       23.6
#    8 AE    ARE   United…  1995     74045.        72.9   2448820       20.2
#    9 AE    ARE   United…  2004     85090.        75.2   4087931       14.5
#   10 AE    ARE   United…  2003     82571.        75.0   3741932       15.0
#   # … with 4,625 more rows, and 3 more variables: neonat_mortal_rate <dbl>,
#   #   region <fct>, income <chr>
#   We will explore how `life_expect` depends on (1) `region` in questions 2-3
#   and (2) `gdp_percap` in questions 4-6.
## Do not modify this line!
library(tidyverse)
nations<-read_csv("data/nations.csv")%>%
  drop_na()%>%
  mutate(region=fct_reorder(region,life_expect))

# 2. Draw multiple horizontal boxplots for `life_expect` by `region`.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_boxplot()` to draw multiple boxplots.
#     - `coord_filp()` to make the boxplots horizontal.
#     - `labs()` to format the labels such that:
#       - `title = "Life Expectancy by Region"`
#       - `x = "Region"`
#       - `y = "Life expectancy (years)"`
#   Store the plot into a `ggplot` object `g1`.
## Do not modify this line!
g1<-ggplot(data=nations,mapping = aes(x=region,y=life_expect))+
  geom_boxplot()+
  coord_flip()+
```

```
 labs(
  title = "Life Expectancy by Region",
  x = "Region",
  y = "Life expectancy (years)"
 )+
 theme_light()


# 3. Use `median()` to compute the overall median of `life_expect`.
#   Store the result into a variable `med`.
#   Add a line on top of `g1` in question 3 to show the overall median,
#   with a subtitle labelling the value of the overall median.
#   To do this, you can use:
#     - `geom_hline()` to add a red horizontal line for `med` on top of `g1`.
#     - `labs()` to format the labels such that:
#       - `subtitle = paste0("Overall median = ", round(med), " years (shown in red)")`
#   Store the plot into a `ggplot` object `g2`.
## Do not modify this line!
med<-median(nations$life_expect)
g2<-g1+geom_hline(yintercept = med,color="red")+
 labs(
  subtitle = paste0("Overall median = ", round(med), " years (shown in red)")
 )

# 4. Draw a scatterplot for `life_exp` vs. `gdp_percap`.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_point()` to draw a scatterplot.
#     - `labs()` to format the labels such that:
#       - `title = "Life Expectancy Increases with GDP Per Capita"`
#       - `x = "GDP per capita ($)"`
#       - `y = "Life expectancy (years)"`
#   Store the plot into a `ggplot` object `g3`.
## Do not modify this line!
g3<-ggplot(data=nations,mapping = aes(x=gdp_percap,y=life_expect))+geom_point(alpha=
0.5,stroke=0)+
 labs(
  title = "Life Expectancy Increases with GDP Per Capita",
  x = "GDP per capita ($)",
  y = "Life expectancy (years)"
 )+
 theme_light()


# 5. Observe from `g3` in question 4 that most of the countries were plotted in
#   tight cluster of points in the upper left corner of the graph.
#   For a better view, we will show a log-scale of `gdp_percap` and add a
#   smoothed conditional mean on top of `g3`.
#   To do this, you can use:
#     - `scale_x_log10()` to plot a log-scale of `gdp_percap` on top of `g3`.
#       (Note that this is better than applying direct transformation to
#       `gdp_percap`, as the axes would be labelled in original data scale
```

```
#       instead of a log-transformed scale, which is hard to interpret.)
#     - `geom_smooth()` to add a smoothed conditional mean
#       (set `se = FALSE`).
#   Store the plot into a `ggplot` object `g4`.
## Do not modify this line!
g4<-g3+scale_x_log10()+geom_smooth(se=FALSE)


# 6. To examine if the relationship between `life_exp` and `gdp_percap` differs
#   by `region`, use `facet_wrap()` to facet the scatterplot `g4` in question
#   5 by `region`.
#   Store the plot into a `ggplot` object `g5`.
## Do not modify this line!
g5<-g4+facet_wrap(~region)

# HW6: ggplot_taxi
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# In this exercise, you will familiarize yourself with data visualization using
# ggplot.
#'
# Throughout the exercise:
#   - Use `theme_light()` for the plots.
#   - Set `alpha = 0.5` and `stoke = 0` for `geom_point()` to avoid overplotting.
#   - For good practice, make sure every graph is complete. Follow the
#     instructions for labelling exactly to get the correct answer.
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#'
# 1. Load the packages `tidyverse` and `viridis`.
#   Use `read_csv()` to read the dataset `nyc_taxi` (located in directory `data/`).
#   Store the corresponding dataframe into a tibble `taxi`.
#   In the following exercises, we will explore how `tip_amount` depends on
#   other features.
## Do not modify this line!
library(tidyverse)
library(viridis)
taxi<-read_csv("data/nyc_taxi.csv")


# 2. Draw a scatterplot for `tip_amount` vs. `fare_amount`.
#   For a better view, we will remove the nagative fare values and some outliers
#   by zooming in. You can use any of the three methods discussed in class.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_point()` to draw the scatterplot.
#     - `coord_cartesian()` with `xlim = c(0, 60)` and `ylim = c(0, 15)` to
```

```
#       keep `fare_amount` in [0, 60] and `tip_amount` in [0, 15].
#     - `labs()` to format the labels such that:
#       - `title = "Tip Amount Increases with Fare Amount Generally"`
#       - `subtitle = "Some people gave fixed tips regardless of fares."`
#       - `x = "Fare amount ($)"`
#       - `y = "Tip amount ($)"`
#   Store the plot into a `ggplot` object `g1`.
#   Some explanations for the pattern:
#     - Prominent diagonal lines: tip amount increases with fare amount.
#       People generally tip at a certain percentage of the fare, which is
#       very likely due to the tip choices provided when people pay with
#       credit card.
#     - Prominent horizontal lines: a group of people give fixed tips
#       regardless of fares.
#     - Vertical line at $52 fare: tip choices vary a lot at this fare.
## Do not modify this line!
g1<-ggplot(data = taxi,mapping = aes(x=fare_amount,y=tip_amount))+
  geom_point(alpha=0.5,stroke=0)+
  coord_cartesian(xlim = c(0, 60),ylim = c(0, 15))+
  labs(title = "Tip Amount Increases with Fare Amount Generally",
       subtitle = "Some people gave fixed tips regardless of fares.",
       x = "Fare amount ($)",
       y = "Tip amount ($)"
       )+theme_light()


# 3. Heatmaps are useful for visualizing frequency counts and identifying clusters.
#   Draw a square heatmap for `tip_amount` vs. `fare_amount`, using `viridis`
#   for a perceptually uniform colormap.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_bin2d()` to draw a square heatmap (set `binwidth = c(2, 1)`).
#     - `coord_cartesian()` with `xlim = c(0, 60)` and `ylim = c(0, 15)` to
#       zoom in the figure as previous.
#     - `scale_fill_viridis()` to set `viridis` as the colormap.
#     - `labs()` to format the labels such that:
#       - `title = "Tip Amount Increases with Fare Amount Generally"`
#       - `subtitle = "Most commonly, trips had fares under $10 and tips around $2."``
#       - `x = "Fare amount ($)"`
#       - `y = "Tip amount ($)"`
#   Store the plot into a `ggplot` object `g2`.
## Do not modify this line!
g2<-ggplot(data = taxi,mapping = aes(x=fare_amount,y=tip_amount))+
  geom_bin2d(binwidth = c(2, 1))+coord_cartesian(xlim = c(0, 60),ylim = c(0, 15))+
  scale_fill_viridis()+
  labs(title = "Tip Amount Increases with Fare Amount Generally",
       subtitle = "Most commonly, trips had fares under $10 and tips around $2.",
       x = "Fare amount ($)",
       y = "Tip amount ($)"
       )+theme_light()
```

```
# 4. Draw a scatterplot for `tip_amount` vs. `trip_distance`, with points
#    colored by `payment_type`.
#    To do this, you can use:
#      - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#      - `geom_point()` to draw a scatterplot colored by `payment_type`.
#      - `coord_cartesian()` with `xlim = c(0, 25)` and `ylim = c(0, 15)` to
#        zoom in the figure as previous.
#      - `labs()` to format the labels such that:
#        - `title = "Tip Amount Increases with Trip Distance Generally"`
#        - `subtitle = paste("Trips with tips were almost always paid in card.",
#                            "Trips paid in cash were rarely tipped.", sep="\n")`
#        - `x = "Trip distance (mile)"`
#        - `y = "Tip amount ($)"`
#      - `theme()` to place the legend at the bottom of the figure.
#        (Note that this should be done AFTER setting `theme_light()` to
#        override the legend settings in the theme).
#    Store the plot into a `ggplot` object `g3`.
## Do not modify this line!
g3<-ggplot(data = taxi, mapping = aes(y=tip_amount,x=trip_distance))+
  geom_point(aes(color=payment_type),alpha=0.5,stroke=0)+
  coord_cartesian(xlim = c(0, 25),ylim = c(0, 15))+
  labs(title = "Tip Amount Increases with Trip Distance Generally",
       subtitle = paste("Trips with tips were almost always paid in card.",
                                          "Trips paid in cash were rarely tipped.", sep="\n"),
       x = "Trip distance (mile)",
       y = "Tip amount ($)"
       )+theme_light()+
  theme(legend.position = "bottom")


# 5. Make a new tibble `taxi_new` of size 10,000 x 2 with two columns,
#    `tip_amount` and `pickup_time`. `pickup_time` is a new column indicating
#    when a ride starts in hour.
#    To do this, you can use:
#      - `mutate()` with `format()` to take the hour component from
#        `tpep_pickup_datetime`.
#      - `dplyr::select()` to select the two columns of interest.
#        (Note: we need to enforce the use of `dplyr` to resolve function
#        conflicts with other packages such as `MASS`.)
#    To check your result, the tibble `taxi_new` prints to:
#    # A tibble: 10,000 x 2
#      tip_amount pickup_time
#           <dbl> <chr>
#    1       2.66 10
#    2       3.85 11
#    3       5    07
#    4       0    08
#    5       1.7  08
#    6       3.32 01
#    7       2.15 07
#    8       2.16 00
#    9       2.16 12
```

```
#    10      3.25 02
#    # … with 9,990 more rows
## Do not modify this line!
select<-dplyr::select
taxi_new<-taxi%>%
  mutate(pickup_time = strftime(tpep_pickup_datetime,format="%H"))%>%
  select(tip_amount,pickup_time)

# 6. Draw a summary statistic plot for minimum, maximum and median of `tip_amount`
#    by `pickup_time`.
#    To do this, you can use:
#      - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#      - `stat_summary()` to plot the `min`, `max` and `median` of `tip_amount`.
#      - `coord_cartesian()` with `ylim = c(0, 50)` to zoom in the figure as previous.
#      - `labs()` to format the labels such that:
#        - `title = "Min/Max/Median Tip Amount by Pickup Time"`
#        - `subtitle = paste("The highest tip was paid to a trip at 4AM.",
#        "9PM had the highest median tip whereas 8PM/10PM had the lowest.",
#        sep="\n")`
#        - `x = "Pickup Time (hour)"`
#        - `y = "Tip amount ($)"`
#    Store the plot a `ggplot` object `g4`.
## Do not modify this line!
g4<-ggplot(data=taxi_new,mapping = aes(x=pickup_time,y=tip_amount))+
  stat_summary(
    fun.y = median,fun.ymin = min, fun.ymax = max
  )+coord_cartesian(ylim = c(0, 50))+
  labs(title = "Min/Max/Median Tip Amount by Pickup Time",
       subtitle = paste("The highest tip was paid to a trip at 4AM.",
                        "9PM had the highest median tip whereas 8PM/10PM had the lowest.",
                        sep="\n"),
       x = "Pickup Time (hour)",
       y = "Tip amount ($)")+theme_light()

# HW6: ggplot wine
#'
# In this exercise, you will familiarize yourself
# with Cleveland dot plot using ggplot.
# You may refer to `https://uc-r.github.io/cleveland-dot-plots` to
# get a better idea of what Cleveland dot plot is.
#
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
#    - Do not change the default position of the plot title.
#    - Do not print the plot.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
```

```
# 1. Load the `tidyverse`, `pgmm`, and `tidyr` packages.
#    Use function `data()` to load `wine` data set from `pgmm` package.
#    Then use `as_tibble()` to turn `wine` into a tibble .
## Do not modify this line!
library(tidyverse)
library(pgmm)
library(tidyr)
data(wine)
wine<-as_tibble(wine)

# 2. Create a tibble `mean_value` with column `Property` and `Value`.
#    Each row contains one property and its mean value.
#    Column `Property` should be factors and its levels should be
#    ordered by `Value`. This will make the dots in plot ordered by `Value`.
#    To do that, you can use:
#    - `select()` to deselect the `Type` column.
#    - `dplyr::summarize_all()` to compute the mean value for each column.
#    - `pivot_longer()` to transform the result into long form.
#    - `dplyr::rename()` to rename the column name to `"Property"`
#      and `"Value"`.
#    - `dplyr::mutate()`, `fct_reorder()`, and `as.factor()`
#      to turn `Property` column into factors.
#    The first few lines of `mean_value` should look like:
#    # A tibble: 27 x 2
#      Property              Value
#      <chr>                 <dbl>
#    1 Alcohol                13.0
#    2 Sugar-free Extract     25.3
#    3 Fixed Acidity          85.6
#    4 Tartaric Acid          2.00
#    5 Malic Acid             2.34
#    6 Uronic Acids           0.915
#    7 pH                    3.30
#    8 Ash                   2.37
#    9 Alcalinity of Ash      19.5
#   10 Potassium              881.
#    # … with 17 more rows
## Do not modify this line!

mean_value<-wine%>%
  select(-Type)%>%
  dplyr::summarize_all(mean)%>%
  pivot_longer(everything(),names_to = "Property",values_to = "Value")%>%
  dplyr::mutate(Property=fct_reorder(as.factor(Property),Value))

# 3. Create a Cleveland dot plot showing the mean value
#    for each of the 27 chemical and physical properties of the wines
#    using `mean_value`.
#    The x axis is the value of the properties labelled as `"Value"`
#    with ticks `0`, `250`, `500`, `750`(default setting).
#    The y axis is the name of the properties labelled as `"Property"`
#    with ticks `"Potassium"`, `"Proline"`, ...,
```

```
#   `"Non-flavanoid Phenols"` from top to bottom.
#   To do that, you can use:
#   - `ggplot()` to initialize a ggplot object.
#   - `geom_point()` to plot the box plot.
#   - `scale_x_log10()` to use the logarithm scale for x axis.
#   - `ggtitle()` to set title to `"Most properties have value below 100"`.
#   - `theme_light()` to set light theme.
#   Save the plot into `mean_value_plot`.
## Do not modify this line!
mean_value_plot<-ggplot(data=mean_value,mapping=aes(x=Value,y=Property))+
  geom_point()+scale_x_log10()+ggtitle("Most properties have value below
100")+theme_light()



# 4. Create a tibble `mean_value_by_type` with column `Type`, `Property`, and `Value`.
#   For each type and property, it has one row that contains its mean value.
#   Column `Property` should be factors and its levels should be
#   ordered by the maximum value of all types.
#   This will make the dots in plot ordered by `Value`.
#   To do that, you can use :
#   - `group_by()` to group by `Type`.
#   - `dplyr::summarize_all()` to compute the mean value for each column.
#   - `pivot_longer()` to rename the column name.
#   - `fct_relevel()` to turn `Property` column into factors
#     and relevel the levels.
#   To use `fct_relevel()`, you need to pass it a vector containing
#   the properties in order of `Value` (the first `Property` in the vector
#   should be the one that corresponds to the lowest `Value`). You may compute it
#   using  `group_by()` to group by `Property`, `summarize` and `max` to compute
#   the maximum value, `arrange()` to compute the order of the levels according
#   to `Value`, `dplyr::select()` to get the column `Property` and `pull()`
#   to extract the vector.
#   The first few lines of `mean_value_by_type` should look like:
#   # A tibble: 81 x 3
#     Type Property          Value
#     <dbl> <fct>             <dbl>
#   1     1 Alcohol           13.7
#   2     1 Sugar-free Extract   26.8
#   3     1 Fixed Acidity       76.7
#   4     1 Tartaric Acid       1.64
#   5     1 Malic Acid          2.01
#   6     1 Uronic Acids        0.811
#   7     1 pH                3.33
#   8     1 Ash               2.46
#   9     1 Alcalinity of Ash    17.0
#  10     1 Potassium          898.
#   # … with 71 more rows
## Do not modify this line!
vec<-wine%>%group_by(Type)%>%dplyr::summarize_all(mean)%>%
  pivot_longer(-Type,names_to = "Property",values_to = "Value")%>%
group_by(Property)%>%summarise(max=max(Value))%>%arrange(max)%>%
```

```
  pull(Property)

mean_value_by_type<-wine%>%group_by(Type)%>%dplyr::summarize_all(mean)%>%
  pivot_longer(-Type,names_to = "Property",values_to = "Value")%>%
  mutate(Property= fct_relevel(as.factor(Property),vec))



# 5. Create a Cleveland dot plot with multiple dots to
#    show mean value of each of the 27 properties by Type
#    using `mean_value_by_type`.
#    That is, each property–such as Ash–should have three different
#    colored dots, one for each Type. The color should be decided
#    by parameters in `geom_point()` using default setting.
#    The tibble we just built in problem 4 enables the properties
#    to be sorted (highest on top) by the maximum value of all types.
#    The x axis is the value of the properties labelled as `"Value"`
#    with ticks `0`, `250`, `500`, `750`(default setting).
#    The y axis is the name of the properties labelled as `"Property"`
#    with ticks `"Proline_mean"`, `"Potassium"`, ...,
#    `"Non-flavanoid Phenols"` from top to bottom.
#    To do that, you can use:
#    - `ggplot()` to initialize a ggplot object.
#    - `geom_point()` to plot the box plot, setting
#      `color` to `factor(Type)`.
#    - `scale_x_log10()` to use the logarithm scale for x axis.
#    - `labs()` to set the legend name to `"Type"`.
#    - `ggtitle()` to set title to `"Mean Proline values vary significantly among three types"`.
#    - `theme_light()` to set light theme.
#    Save the plot into `mean_value_plot2`.
## Do not modify this line!
mean_value_plot2<-
ggplot(data=mean_value_by_type,mapping=aes(x=Value,y=Property))+geom_point(mappin
g = aes(color=factor(Type)))+
  scale_x_log10()+
  labs(color="Type")+
  ggtitle("Mean Proline values vary significantly among three types")+theme_light()

# HW6: ggplot FIFA
#'
# In this exercise, you will familiarize yourself with data visualization
# using ggplot. To be specific, you will expore the FIFA data set.
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
#    - Do not change the default position of the plot title.
#    - Do not print the plot.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
```

```
#'
# 1. Load the `tidyverse`, `readr`, and `scales` packages.
#    Use the function `read_csv()` to read the dataset `data/fifa.csv`.
#    Store the corresponding dataframe into a dataset `fifa`.
#    The dimension should be 17,955 x 62.
## Do not modify this line!
library(tidyverse)
library(readr)
library(scales)
fifa<-read_csv("data/fifa.csv")

# 2. Plot the density histogram and density curve for `Age` column.
#    The x axis should be the value of age labelled as `"Age"`
#    with ticks 20, 30, 40 (default setting).
#    The y axis should be density labelled as `"Density"`
#    with ticks 0.00, 0.02, ..., 0.08 (default setting).
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object. You can set its arguments
#      `data` and `mapping` to plot the `Age` column of the dataset.
#      Use `aes` to set parameters `mapping`.
#    - `geom_histogram()` to plot the density histogram.
#      Please set the bin wigth to 2 (you can use `binwidth`) and color to `"white"`.
#      Hint : Set `y=..density..` to draw the density instead of count.
#    - `geom_density()` to plot the density curve.
#      Please set color to `"blue"`.
#    - `labs()` to format the labels such that:
#      - `title = "The majority of players are 20 to 30 years old"`
#      - `y = "Density"`
#    - `theme_light()` to set light theme (i.e. a light backgroung).
#    Save the plot into `age_plot`.
#    Please note that the density curve should be above the histogram.
#    The order of your commands matters.
## Do not modify this line!
age_plot<-ggplot(data = fifa, mapping = aes(x=Age))+geom_histogram(mapping =
aes(y=..density..),binwidth=2,color="white")+
  geom_density(color="blue")+labs(title = "The majority of players are 20 to 30 years old",
                                  y = "Density"
                                  )+theme_light()


# 3. Filter the `fifa` dataset to select players that belong to club
#    `"FC Barcelona"`, `"Juventus"`, `"Manchester City"`, `"Real Madrid"`,
#    or `"Manchester United"`. Convert `Club` column into factors and
#    order the levels by `Wage`. Store the result into tibble `club_wage`.
#    You can use :
#      - `filter()` to get the players in the 5 correpsondinng clubs
#      - `mutate()` and `fct_reorder()` so that the players in each clubs
#        are ordered in decreasing `Wage`.
#    The first rows of its print should be :
#    # A tibble: 157 x 62
#    ID Name    Age Photo Nationality Flag  Overall Potential Club
#    <dbl> <chr> <dbl> <chr> <chr>       <chr>   <dbl>     <dbl> <fct>
```

```
#    1 158023 L. M…    31 http… Argentina   http…      94       94 FC B…
#    2  20801 Cris…    33 http… Portugal    http…      94       94 Juve…
#    3 193080 De G…    27 http… Spain       http…      91       93 Manc…
#    4 192985 K. D…    27 http… Belgium     http…      91       92 Manc…
#    5 177003 L. M…    32 http… Croatia     http…      91       91 Real…
#    6 176580 L. S…    31 http… Uruguay     http…      91       91 FC B…
#    7 155862 Serg…    32 http… Spain       http…      91       91 Real…
#    8 182521 T. K…    28 http… Germany     http…      90       90 Real…
#    9 168542 Davi…    32 http… Spain       http…      90       90 Manc…
#    10 211110 P. D…   24 http… Argentina   http…      89       94 Juve…
#    # … with 147 more rows, and 53 more variables: `Club Logo` <chr>,
## Do not modify this line!
club_wage<-fifa%>%
  filter(Club %in% c("FC Barcelona", "Juventus", "Manchester City", "Real Madrid",
                     "Manchester United"))%>%
  mutate(Club=fct_reorder(as.factor(Club),Wage))


# 4. Plot the boxplot of `Wage` by `Club` using dataset `club_wage`.
#    The boxplots should be horizontal and ordered by median wage
#    from highest (on top of the figure) to lowest.
#    The x axis should be `"Wage"` without label
#    with ticks `"€0"`, `"€200,000"`, `"€400,000"`.
#    The y axis should be the name of clubs
#    with ticks `"Juventus"`, `"FC Barcelona"`, `"Real Madrid"`,
#    `"Manchester City"`, `"Manchester United"` from top to bottom.
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object.
#      Set the `mapping` parameter correctly.
#    - `geom_boxplot()` to plot the box plot.
#    - `scale_y_continuous()` and `dollar_format` to add the Euro sign prefix
#      and thousands comma to wage.
#    - `labs()` to format the labels such that:
#      - `title = "Top 5 highest median wage clubs"`
#      `subtitle = "The medians are remarkably similar,
#      but the higher quantiles show more variation."`
#      - `y = "Wage"`
#      - `x = ""`
#    - `theme_light()` to set light theme.
#    - `coord_flip()` to flip the coordinate.
#    After the flip, the x-axis should now be `"Wage"`.
#    Save the plot into `wage_plot`.
## Do not modify this line!
wage_plot<-ggplot(data=club_wage,mapping = aes(x=Club,y=Wage))+
  geom_boxplot()+
  scale_y_continuous(labels = dollar_format(prefix = "€"))+
  coord_flip()+
  labs(title = "Top 5 highest median wage clubs",
       subtitle = "The medians are remarkably similar, but the higher quantiles show more
variation.",
       y = "Wage",
       x = ""
```

```
    )+
 theme_light()

# 5. Create a tibble `fifa_height_weight` that contains all players whose
#    position is either `"GK"` or `"CM"`.
#    You can use `filter()` to filter the dataset given the two positions.
#    It should print to :
#    # A tibble: 3,366 x 62
#    ID Name    Age Photo Nationality Flag  Overall Potential Club  `Club Logo`  Value   Wage
#    <dbl> <chr> <dbl> <chr> <chr>       <chr>   <dbl>     <dbl> <chr> <chr>       <dbl>  <dbl>
#    1 193080 De G…    27 http… Spain       http…      91       93 Manc… https://cd… 7.20e7
260000
#    2 200389 J. O…    25 http… Slovenia    http…      90       93 Atlé… https://cd… 6.80e7
94000
#    3 192448 M. t…    26 http… Germany     http…      89       92 FC B… https://cd… 5.80e7
240000
#
## Do not modify this line!
fifa_height_weight<-fifa%>%
 filter(Position=="GK"|Position=="CM")


# 6. Create a scatter plot of `Weight_kg` against `Height_m` in
#    `fifa_height_weight` colored by `Position` (either `"GK"` or `"CM"`).
#    The x axis should be the height labelled as `Height (m)`
#    with ticks `1.6`, `1.7`, `1.8`, `1.9`, `2.0` (default setting).
#    The y axis should be the weight labelled as `Weight (kg)`
#    with ticks `"60"`, `"70"`, `"80"`, `"90"`, `"100"` (default setting).
#    To do that, use :
#    - `ggplot()` to initialize a ggplot object.
#    - `geom_point()` to plot the scatter plot.
#      You can set `mapping` correctly to plot `Weight_kg` against `Height_m`
#      and set `col` to `Position` to get the right colors.
#      The two colors should be salmon and turquoise.
#      Set `alpha` to `0.8`.
#    - `labs()` to format the labels such that:
#      - `title = "GK tends to be heavier and taller"`
#      - `y = "Weight (kg)"`
#      - `x = "Height (m)"`
#    - `theme_light()` to set light theme.
#    Save the plot into `height_weight_plot`.
## Do not modify this line!
height_weight_plot<-ggplot(data=fifa_height_weight,mapping =
aes(x=Height_m,y=Weight_kg))+
 geom_point(mapping=aes(color=Position),alpha=0.8)+
 labs(
   title = "GK tends to be heavier and taller",
   y = "Weight (kg)",
   x = "Height (m)"
 )+theme_light()
```

```
# 7. Create a tibble `agility` containing the players whose
#    `Preferred Foot` is either in `"Left"` or `"Right"`. You can use
#    `filter()` to filter given the two `Preferred Foot`.
#    It should print to :
#    # A tibble: 17,907 x 62
#    ID Name    Age Photo Nationality Flag  Overall Potential Club  `Club Logo`  Value   Wage
#    <dbl> <chr> <dbl> <chr> <chr>       <chr>   <dbl>     <dbl> <chr> <chr>       <dbl>  <dbl>
#    1 158023 L. M…    31 http… Argentina   http…      94        94 FC B… https://cd… 1.10e8
565000
#    2  20801 Cris…    33 http… Portugal    http…      94        94 Juve… https://cd… 7.70e7
405000
#    3 190871 Neym…    26 http… Brazil      http…      92        93 Pari… https://cd… 1.18e8
290000
#
## Do not modify this line!
agility<-fifa%>%
  filter(`Preferred Foot`=="Left"|`Preferred Foot`=="Right")

# 8. Plot the histogram of `Agility` facted by `Preferred Foot`.
#    There should be 2 subplots `Left` (on the left of the figure)
#    and `Right` (on the right)..
#    The x axis should be `Agility` labelled as `"Agility"`
#    with ticks `"25"`, `"50"`, `"75"`, `"100"` for both
#    subplots (default setting).
#    The y axis should be frequency labelled as `"Count (n)"`
#    with ticks `"0"`, `"200"`, `"400"` for `Left` subplot and
#    `"0"`, `"500"`, `"1000"`, `"1500"`, `"2000"` for `Right` subplot.
#    (There are much more right footed players than left footed)
#    To do that, use:
#    - `ggplot()` to initialize a ggplot object.
#    - `geom_histogram()` to plot the histogram plot.
#      You can use `mapping` to draw the correct variables,
#      and set `bins` (number of bins) to `20`.
#      Set color to `#FFFFFF` i.e. white.
#    - `facet_wrap()` to facet by `Preferred Foot`
#      and set `scales` to `free` so that
#      we can compare the shape of the two distributions.
#    - `labs()` to format the labels such that:
#      - `title = "Distribution of agility is similar regardless of preferred foot"`
#      - `y = "Count (n)"`
#    - `theme_light()` to set light theme.
#    Save the plot into `agility_plot`.
## Do not modify this line!
agility_plot<-ggplot(data=agility,mapping = aes(x=Agility))+
  geom_histogram(bins=20,color="#FFFFFF")+
  facet_wrap(~`Preferred Foot`,scales ="free")+
  labs(
    title = "Distribution of agility is similar regardless of preferred foot",
    y = "Count (n)"
  )+theme_light()
```

```
?facet_wrap

# HW6: ggplot9
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions, in the same order.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Do the following:
#   - load the `readr` and `dplyr` package
#   - then load the `/course/data/employee.csv` file using `read_csv()`, containing
#   salary and overtime information aggregated over different organizational
#   groups.
#   - then, transform the `Year` column to a factor column.
#   Hint: you can use `mutate()` and `factor()` to do so.
#   - assign the resulting tibble to the variable `employee`
#   - use `head()` on `employee` to see the first lines and assign the output
#   tibble to `employee_head`
#   `employee_head` should look like:
#   # A tibble: 6 x 4
#     Year  organization                    Salaries Overtime
#     <fct> <chr>                          <dbl>    <dbl>
#   1 2013  Public Protection                 123841.   76854.
#   2 2014  Public Works, Transportation & Commerce   61138.    7341.
#   3 2016  Public Works, Transportation & Commerce   41193.      0
#   4 2015  Public Works, Transportation & Commerce   66994.   26634.
#   5 2013  Public Works, Transportation & Commerce   74261.      0
#   6 2015  Public Works, Transportation & Commerce  141778.      0
## Do not modify this line!
library(readr)
library(dplyr)
employee<-read_csv("/course/data/employee.csv")
employee<-employee%>%
  mutate(Year=factor(Year))
employee_head<-head(employee)

# 2. Load the `ggplot2` and `scales` packages. Generate a boxplot of salaries
#   per year and assign the plot to variable `salary_boxplots`.
#   To do so, you can:
#   - use `ggplot()`, `geom_boxplot()` to create the boxplot from `employee`
#   - then add `scale_y_continuous(label = comma)`
#     Note: `scale_y_continuous(label = comma)` makes the y-labels more
#     readable, forcing a comma notation for numbers (e.g. 5e+05 becomes
#     500,000)
#    - use `labs()` to format the labels such that:
#        - `title = "Salaries are getting a longer tail with time"`
#        - `y = "Yearly salaries (USD)"`
#   - finally add the light theme using `theme_light()`
#   (adding each element to the plot using `+`)
#   The x-axis name should be `Year` and the y-axis should be
```

```
#    `"Yearly salaries (USD)"`.
## Do not modify this line!
library(ggplot2)
library(scales)
salary_boxplots<-ggplot(data = employee,mapping =
aes(x=Year,y=Salaries))+geom_boxplot()+
  scale_y_continuous(label = comma)+labs(
    title = "Salaries are getting a longer tail with time",
    y = "Yearly salaries (USD)"
  )+theme_light()


# 3. Change all salaries higher than `250,000` to `250,000` and assign the
#    transformed tibble to `employee_trunc`
#    Hint: you can use `mutate` and `ifelse` to do so
#    `employee_trunc` should look like:
#    # A tibble: 213,116 x 4
#       Year  organization                        Salaries Overtime
#       <fct> <chr>                              <dbl>    <dbl>
#     1 2013  Public Protection                   123841.   76854.
#     2 2014  Public Works, Transportation & Commerce    61138.    7341.
#     3 2016  Public Works, Transportation & Commerce    41193.       0
#     4 2015  Public Works, Transportation & Commerce    66994.   26634.
#     5 2013  Public Works, Transportation & Commerce    74261.       0
#     6 2015  Public Works, Transportation & Commerce   141778.       0
#     7 2015  Public Works, Transportation & Commerce    51152.   25725.
#     8 2015  Public Protection                   27352.       0
#     9 2013  Human Welfare & Neighborhood Development   72115.       0
#    10 2015  General Administration & Finance           391.     323.
#    # … with 213,106 more rows
## Do not modify this line!
employee_trunc<-employee%>%
  mutate(Salaries=ifelse(Salaries>250000,250000,Salaries))

# 4. Generate histograms of salaries faceted on years - ie. one histogram per
#    year - and assign the plot to variable `salary_histograms`.
#    To do so, you can:
#    - create the plot by calling `ggplot()` on `employee_trunc`
#    - adding `geom_histogram()` with a `binwidth` of `5,000`
#    - then faceting on years using `facet_wrap()`
#    - then forcing comma notation for both axes (as you did for the y-axis in
#      question 2.)
#     - `labs()` to format the labels such that:
#         - `title = "Salaries have a slowly changing bimodal distribution"`
#         - `x = "Yearly salaries (USD)"`
#         - `y = ""`
#    - then add the light theme using `theme_light()`
#    - finally add a layer to rotate the x-labels by 45 degrees to make them
#      easier to  read to the plot using `theme()`, setting `axis.text.x` to
#      `element_text(angle = 45, hjust = 1)`.
#      Setting the parameter `hjust` of `element_text()` to `1` lowers the
#      labels so that they don't overlap with the figure.
```

```
#    The x-axis should read `Yearly salaries (USD)` and the y-axis should have no name.
## Do not modify this line!
salary_histograms<-ggplot(data = employee_trunc,mapping = aes(x=Salaries))+
  geom_histogram(binwidth = 5000)+
  facet_wrap(~Year)+scale_y_continuous(label = comma)+scale_x_continuous(label =
comma)+
  labs(title = "Salaries have a slowly changing bimodal distribution",
       x = "Yearly salaries (USD)",
       y = ""
       )+theme_light()+theme(axis.text.x = element_text(angle = 45, hjust = 1))


# 5. Plot overlapping density curves of the salaries, one curve per year, on a
#    single set of axes. Assign the plot to the variable `salary_densities`.
#    Each curve should be a different color.
#    To do so, you can:
#    - use `ggplot()` setting with the `colour` parameter in the aesthetic to
#      `Year`
#    - add `geom_density()` to plot the densities of `Salaries`
#    - then force comma notation for the x-axis (as you did for the y-axis in
#      question 2.)
#     - `labs()` to format the labels such that:
#         - `title = "The salaries modes are becoming less acute with time"`
#         - `x = "Yearly salaries (USD)"`
#         - `y = "Densities for each year"`
#    - finally add the light theme using `theme_light()`
#    The x-axis should read `"Yearly salaries (USD)"` and the y-axis should be
#    titled `"Densities for each year"`. There should also be a colour legend
#    titled `Year`.
#    Note that the densities of the more recent years are plotted over the
#    older ones. (Both the legend and the order of plots correspond to default
#    setting if you use the suggested functions.)
## Do not modify this line!
salary_densities<-ggplot(data = employee_trunc,mapping = aes(x=Salaries,color=Year))+
  geom_density()+scale_x_continuous(label = comma)+
  labs(title = "The salaries modes are becoming less acute with time",
       x = "Yearly salaries (USD)",
       y = "Densities for each year")+theme_light()


# 6. Keep only the employees in `Culture & Recreation` from `employee` with a
#    positive salary and assign the result to variable `employee_culture`.
#    Hint: you can use `filter()`
#    `employee_culture` should look like:
#    # A tibble: 19,569 x 4
#       Year  organization        Salaries Overtime
#       <fct> <chr>               <dbl>    <dbl>
#     1 2014  Culture & Recreation   29282.      0
#     2 2015  Culture & Recreation   26871.      0
#     3 2014  Culture & Recreation   17744.      0
#     4 2013  Culture & Recreation   76945.     694.
#     5 2014  Culture & Recreation    9362.      0
```

```
#     6 2015  Culture & Recreation    8304.      0
#     7 2013  Culture & Recreation   12992       0
#     8 2013  Culture & Recreation   16208.      0
#     9 2013  Culture & Recreation   59124.      0
#    10 2017  Culture & Recreation    2164.      0
#    # … with 19,559 more rows
## Do not modify this line!
employee_culture<-employee%>%
  filter(organization=="Culture & Recreation"&Salaries>0)


# 7. Plot a QQ plot for the logarithm of salaries of the `Culture & Recreation`
#    employees and assign the result to variable `log_salaries_qq`.
#    To do so, you can:
#    - create the plot from `employee_culture` using `ggplot()` and setting
#      `sample` (argument of the aesthetic) to the logarithms of `Salaries`
#      (using `log10()`)
#    - add `geom_qq()` to create the QQPlot
#    - add the line to the plot using `geom_qq_line()`
#     - `labs()` to format the labels such that:
#         - `title = "The log salaries severely diverge from a normal distribution"`
#         - `x = "Theoretical quantiles of log(salaries)"`
#         - `y = "Sample quantiles of log(salaries)"`
#    - finally add the light theme using `theme_light()`
## Do not modify this line!
log_salaries_qq<-ggplot(data = employee_culture,mapping =
aes(sample=log10(Salaries)))+geom_qq()+geom_qq_line()+
  labs(title = "The log salaries severely diverge from a normal distribution",x = "Theoretical
quantiles of log(salaries)",
       y = "Sample quantiles of log(salaries)")+theme_light()

?aes

# HW6: ggplot10
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Do the following:
#    - load the `readr` and `dplyr` package
#    - then load the `data/taxi.csv` file using `read_csv()`, containing some
#    fare and tip amounts of NYC yellow cab rides in June 2018, Assign it to
#    the variable `taxi`
#    - then keep only the rows corresponding to trips with `tip_amount < 15`
#    and `0 <= fare_amount <= 50`
#    Hint: you can use `filter()` and `between()` to do so.
#    - assign the resulting tibble of the last two steps to variable `taxi`
#    - use `head()` to see the first lines and assign the resulting tibble to
#    `taxi_head`.
```

```
#   `taxi_head` should look like:
#   # A tibble: 6 x 3
#         X1 fare_amount tip_amount
#      <dbl>      <dbl>     <dbl>
#   1 7832414        7        1.65
#   2 3247452       10.5      2.36
#   3 1026979        5        0.87
#   4 2003753       18.5      4.05
#   5 5884122        6        2
#   6 2373845        3.5      0.96
## Do not modify this line!
library(readr)
library(dplyr)
taxi<-read_csv("data/taxi.csv")
taxi<-taxi%>%filter(tip_amount < 15& fare_amount <= 50&fare_amount >=0 )
taxi_head<-head(taxi)
# 2. Load the `scales` and `ggplot2`package. Plot a histogram of the tip
#   amounts and assign the plot to `tip_histogram`.
#   To do so, you can:
#   - call `ggplot()` and `geom_histogram()` setting the `binwidth` to `1`
#   - add `scale_y_continuous(label = comma)` to make the plot more
#   reader-friendly
#   Note: `scale_y_continuous(label = comma)` makes the y-labels more
#   readable, forcing a comma notation for numbers (e.g. 5e+05 becomes
#   500,000)
#   - use `labs()` to format the labels such that:
#       - `title = "The tip amounts are skewed to the right"`
#       - `x = "Tip Amount (USD)"`
#       - `y = ""`
#   - finally add the light theme using `theme_light()`
#   The x-axis should read `Tip Amount (USD)` and the y-axis should have no
#   name
## Do not modify this line!
library(scales)
library(ggplot2)
tip_histogram<-ggplot(data = taxi,mapping =
aes(x=tip_amount))+geom_histogram(binwidth = 1)+
  scale_y_continuous(label = comma)+scale_y_continuous(label = comma)+
  labs(title = "The tip amounts are skewed to the right",
       x = "Tip Amount (USD)",
       y = "")+theme_light()

# 3. Create a `fare_bins` column in `taxi` mapping each `fare_amount` to one of
#   `("0-10", "11-20", "21-30", "31-40", "41-50")` depending on which interval
#   the fare amounts falls in and assign the transformed tibble to variable
#   `taxi_binned`.
#   Hint: you can use `mutate()`,`cut()` with 5 breaks.
#   `taxi_binned` should look like:
#   # A tibble: 96,469 x 4
#         X1 fare_amount tip_amount fare_bins
#      <dbl>      <dbl>     <dbl> <fct>
#   1 7832414        7        1.65 0-10
```

```
#   2 3247452       10.5      2.36 11-20
#   3 1026979        5        0.87 0-10
#   4 2003753       18.5      4.05 11-20
#   5 5884122        6        2    0-10
#   6 2373845        3.5      0.96 0-10
#   7 3418614       13        2.96 11-20
#   8  603784        6        2.19 0-10
#   9  404232        6        0    0-10
#   10 1381237       17.5      5.79 11-20
#   # … with 96,459 more rows
## Do not modify this line!
taxi_binned<-taxi%>%

mutate(fare_bins=cut(fare_amount,breaks=c(0,10,20,30,40,50),include.lowest=TRUE,labels
=c("0-10", "11-20", "21-30", "31-40", "41-50"),right=TRUE))


# 4. Plot tip histograms faceted on 10 dollar bins of fare amounts and assign
#   the plot to `tip_facet_hist`.
#   To do so, you can:
#   - create the plot from `taxi_binned` with `ggplot()`
#   - then add  `geom_histogram()` with a `binwidth`` of `1`
#   - then forcing comma notation for the y-axis (as you did in question 2.)
#   - and after that, faceting on the bins using `facet_grid()`
#   - use `labs()` to format the labels such that:
#       - `title = "The tip skew becomes less severe as fares increase"`
#       - `x = "Tip Amount (USD)"`
#       - `y = ""`
#   - finally add the light theme using `theme_light()`
#   The x-axis should read `Tip Amount (USD)` and the y-axis should have no
#   name
## Do not modify this line!
tip_facet_hist<-ggplot(data = taxi_binned,mapping = aes(tip_amount))+
  geom_histogram(binwidth = 1)+scale_y_continuous(label = comma)+
  facet_grid(~fare_bins)+labs(title = "The tip skew becomes less severe as fares increase",
                              x = "Tip Amount (USD)",
                              y = "")+theme_light()



# 5. Generate a hexagonal heatmap of the scatter plot and assign the plot to
#   variable `hex_heatmap`.
#   To do so:
#   - create the plot using `ggplot()` (fares on the x-axis and tips on the
#     y-axis)
#   - add a `geom_hex()` with `binwidth`` of `2.5` of fare amounts and `0.5` of
#     tip amounts
#   - then change the colours of the bins using `scale_fill_gradient`. The
#     colours should range from `"yellow"` (`low`) to `"darkgreen"` (`high`)
#     and the gradients should follow a `log10` transformation
#     (`trans = log10`)
#   - use `labs()` to format the labels such that:
```

```
#        - `title = "There is a preference of tipping proportionately to the fare"`
#        - `x = "Fare Amount (USD)"`
#        - `y = "Tip Amount (USD)"`
#    - finally add the light theme using `theme_light()`
#    The x-axis should read `"Fare Amount (USD)"` and the y-axis should be named
#    `"Tip Amount (USD)"`.
## Do not modify this line!
hex_heatmap<-ggplot(data = taxi_binned,mapping = aes(x=fare_amount,y=tip_amount))+
  geom_hex(binwidth = c(2.5, 0.5))+
  scale_fill_gradient(low='yellow',high='darkgreen',trans="log10")+
  labs(title = "There is a preference of tipping proportionately to the fare",
       x = "Fare Amount (USD)",
       y= "Tip Amount (USD)")+theme_light()


?geom_hex()

# HW6: Wine
#'
# In this problem, we will investigate the quality of red wine.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Load necessary packages : `ggplot2`, `viridis`, `lubridate` and
#    `tidyverse`. Load the file `data/winequalityN.csv` and save it
#    into tibble `wine`. Filter out the na values using `drop_na()`.
#    The tibble `wine` should print to:
#    # A tibble: 6,463 x 13
#    type  `fixed acidity` `volatile acidi… `citric acid`
#    <chr>         <dbl>          <dbl>       <dbl>
#    1 white           7           0.27        0.36
#    2 white           6.3          0.3        0.34
#    3 white           8.1          0.28        0.4
#    4 white           7.2          0.23        0.32
#    5 white           7.2          0.23        0.32
#    6 white           8.1          0.28        0.4
#    7 white           6.2          0.32        0.16
#    8 white           7           0.27        0.36
#    9 white           6.3          0.3        0.34
#    10 white           8.1           0.22        0.43
#    # … with 6,453 more rows, and 9 more variables: `residual
#    #   sugar` <dbl>, chlorides <dbl>, `free sulfur dioxide` <dbl>,
#    #   `total sulfur dioxide` <dbl>, density <dbl>, pH <dbl>,
#    #   sulphates <dbl>, alcohol <dbl>, quality <dbl>
## Do not modify this line!
library(ggplot2)
library(viridis)
library(lubridate)
```

```
library(tidyverse)
wine<-read_csv("data/winequalityN.csv")
wine<-wine%>%
  drop_na()


# 2. Set random seed to 0 and save the random seed vector to variable `seed`.
#    (Hint: use the command `seed <- .Random.seed`)
#    Draw the scatter plot of wine quality and fixed acidity.
#    To do that, use:
#     - `ggplot` to initialize the plot object with
#       `aes(x = `fixed acidity`, y = quality)`
#       to select the variables interested.
#     - `geom_point` to draw the scatter plot.
#     - `labs()` to format the labels such that:
#         - `title = "The quality does not seem to be correlated with acidity"`
#         - `x = "Fixed Acidity"`
#         - `y = "Quality"`
#    Save this plot to `plot1_1`. You will find the scattered plots to be mostly
#    horizontal lines because the wine quality are all integers.
#    We can draw a plot to see how fixed acidity will change if the
#    quality is not integer by adding jitter to the above plot.
#    To do that, use:
#     - `ggplot` to initialize the plot object with
#       `aes(x = `fixed acidity`, y = quality)`
#       to select the variables interested.
#     - `geom_point` to draw the scatter plot.
#     -  set argument `position = position_jitter()` in `geom_point`.
#     - `labs()` to format the labels such that:
#         - `title = "The pattern is easier to interpret with jitter"`
#         - `x = "Fixed Acidity"`
#         - `y = "Quality"`
#    Save the new plot as `plot1_2`.
## Do not modify this line!
set.seed(0)
seed <- .Random.seed
plot1_1<-ggplot(data = wine,aes(x = `fixed acidity`, y = quality))+
  geom_point()+labs(
    title = "The quality does not seem to be correlated with acidity",
    x = "Fixed Acidity",
    y = "Quality"
  )
plot1_2<-ggplot(data = wine,aes(x = `fixed acidity`, y = quality))+
  geom_point(position = position_jitter())+labs(
    title = "The pattern is easier to interpret with jitter",
    x = "Fixed Acidity",
    y = "Quality"
  )


# 3. We found from previous figure that most wines have fixed acidity
#    between 6 and 8, especially the high quality ones. To check that,
```

```
#   we can draw a histogram of fixed acidity for each quality level.
#   (7 histograms in total, as `quality` goes from 3 to 9)
#   To draw the plot, use:
#    - `ggplot()` to initialize the plot object with `aes(x = `fixed acidity`)`
#      to specify the relevant variables.
#    - `geom_histogram()` to plot histogram, please set argument `color` to `black`,
#      `fill` to `orange`, and `binwidth` to 0.5.
#    - `facet_wrap()` to generate plots for each quality level.
#    - `labs()` to format the labels such that:
#      - `title = "Fixed acidity distribution for different quality levels"`
#      - `x = "Fixed Acidity"`
#      - `y = "Count (n)"`
#   Please save the `ggplot` object to `plot2`.
## Do not modify this line!
plot2<-ggplot(data = wine,aes(x = `fixed acidity`))+
 geom_histogram(color='black',fill='orange',binwidth = 0.5)+
 facet_wrap(~quality)+
 labs(title = "Fixed acidity distribution for different quality levels",
     x = "Fixed Acidity",
     y = "Count (n)")


# 4. Load package `GGally`.
#   Draw pair plot of columns "density", "volatile acidity", "citric acid"
#   to investigate pairwise correlations of these variables
#   in two different type of wines respectively.
#   (This will generate a 3 x 3 plot matrix).
#   To do that, use:
#   - `ggpairs()` to draw, please set the following parameters :
#    - `mapping` to `aes(color = type)` to set
#      different colors to different type of wines.
#    - `columns=c("density", "volatile acidity", "citric acid")`
#      to select the columns.
#    - `lower=list(continuous=wrap())` to set the lower triangle
#      plots we want to see about pairwise variables.
#      In this case, we want to see a scatter plot with a smooth regression line.
#      To do that, specify the plot inside the `wrap` function:
#       - Set `"smooth"` to be the first argument of `wrap`.
#       - Set transparency parameter `alpha = 0.3`.
#       - Set point size `size = 0.1`.
#    - `upper = list(combo = "box")` to display the correlation of pairwise
#      variables shown in upper triangle.
#    - `diag` to `list(continuous = wrap())` to set the plot on diagonals.
#      In this case, we want the density distribution of these variables.
#      To do that, specify the plot inside the `wrap` function:
#       - Set `"densityDiag"` to be the first argument of `wrap`
#       - Set `alpha = 0.7`.
#    - Set `axisLabel` to `"show"`.
#   - `theme_bw()` to use the black and white theme with argument
#     `base_size` set to 0.5 (it is the font size).
#   Save the `ggplot` object to `plot3`.
## Do not modify this line!
```

```
library(GGally)
plot3<-ggpairs(data=wine,mapping=aes(color = type),columns=c("density", "volatile acidity",
"citric acid"),
      lower=list(continuous=wrap("smooth",alpha = 0.3,size = 0.1)),upper = list(combo =
"box"),
      diag = list(continuous = wrap("densityDiag",alpha = 0.7,axisLabel="show"))
      )+theme_bw(base_size=0.5)

# 5. From `plot3`, we can see there is an obvious positive correlation between
#    `"density"` and `"citric acid"` in red wine.
#    We want to fit the linear relationship between "density" and "citric acid"
#    in red wines at different quality levels separately.
#    To do that, please first do data manipulations as follows:
#    Divide the wine quality into three ranges: `"bad"` (2,5], `"average"` (5,7]
#    and `"good"` (7,10], and store these results in column `quality_label`.
#    To do that, you can use:
#     - `filter()` to select all the red wines.
#     - `cut()` and `mutate()` to add a column called `quality_label` which
#       represents the quality range of this wine ((2,5], (5,7], (7,10]).
#     - `mutate()` and `forcats::fct_recode()`to recode the level
#       of column `quality_label` to c("bad", "average", "good").
#       (set `(2,5]` to `"bad"`, `(5,7]` to `"average"`
#       and `(7,10]` to `"good"`)
#    Save the new tibble into `red`. `red` should look like the following:
#    # A tibble: 1,593 x 14
#    type  `fixed acidity` `volatile acidi… `citric acid`
#    <chr>         <dbl>          <dbl>       <dbl>
#    1 red            7.4          0.7          0
#    2 red            7.8          0.88         0
#    3 red            7.8          0.76         0.04
#    4 red           11.2          0.28         0.56
#    5 red            7.4          0.7          0
#    6 red            7.4          0.66         0
#    7 red            7.9          0.6          0.06
#    8 red            7.3          0.65         0
#    9 red            7.8          0.580        0.02
#    10 red            7.5          0.5          0.36
#    # … with 1,583 more rows, and 10 more variables: `residual
#    #   sugar` <dbl>, chlorides <dbl>, `free sulfur dioxide` <dbl>,
#    #   `total sulfur dioxide` <dbl>, density <dbl>, pH <dbl>,
#    #   sulphates <dbl>, alcohol <dbl>, quality <dbl>,
#    #   quality_label <fct>
## Do not modify this line!
red<-wine%>%filter(type=="red")%>%
  mutate(quality_label=cut(quality,breaks = c(2,5,7,10)))%>%

mutate(quality_label=fct_recode(quality_label,"bad"="(2,5]","average"="(5,7]","good"="(7,
10]"))


# 6. Then draw a scatter plot with smooth regression line of `"density"`
#    against `"citric acid"` of the `red` tibble just created. In this plot, you have
```

```
#   to show the fit line and the scattered points labeled in different colors
#   according to their quality labels (good, bad, or average).
#   To do that, use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_smooth()` to fit the relationship and set `method` argument to `lm`.
#     - `geom_point()` with argument `aes(color = quality_label)` to add data points
#       colored in their quality labels.
#     - `scale_color_brewer()` with argument `type` set to `qual` to scale the color
#         according to the value of quality.
#    - `labs()` to format the labels such that:
#      - `title = "Positive correlation of Density and Fixed Acidity"`
#      - `x = "Fixed Acidity"`
#      - `y = "Density"`
#   Save the `ggplot` object to `plot4`.
## Do not modify this line!
plot4<-ggplot(data=red,mapping = aes(y=density,x=`citric acid`))+
  geom_smooth(method = 'lm')+
  geom_point(aes(color = quality_label))+
  scale_color_brewer(type = 'qual')+
  labs(title = "Positive correlation of Density and Fixed Acidity",
       x = "Fixed Acidity",
       y = "Density"
  )

# HW6: football
#'
# In this problem, we will continue to play with the football dataset from HW5.
# Throughout the exercise:
#   - Use `%>%` to structure your operations.
#   - Do NOT use `for`, `while` or `repeat` loops.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Load the following packages: `ggplot2`, `viridis`, `lubridate` and
#   `tidyverse`. Load the file `data/football.csv` and save it into tibble `fu`.
#   Draw a box plot of total home goals from 2001 to 2019 of team England,
#   Germany, France, Belgium, Italy and Spain.
#   To do that, first create a tibble `fu1` containing three columns
#   `home_team`, `year` and `total`. You can use :
#     - `year()` to extract year from `date` and `filter()` to filter years
#     from 2001 to 2019, and to only keep the 6 countries in `home_team` column.
#     - `group_by()` and `dplyr::summarise()` to count the total goals of each team
#     in each year.
#   `fu1` should print to :
#   # A tibble: 114 x 3
#   # Groups:   home_team [6]
#   home_team  year total
#   <chr>     <dbl> <dbl>
```

```
#    1 Belgium    2001    16
#    2 Belgium    2002     6
#    3 Belgium    2003    11
#    4 Belgium    2004     3
#    5 Belgium    2005    14
#    6 Belgium    2006     8
#    7 Belgium    2007     7
#    8 Belgium    2008     7
#    9 Belgium    2009    12
#    10 Belgium    2010     6
#    # … with 104 more rows
## Do not modify this line!
library(ggplot2)
library(viridis)
library(tidyverse)
library(lubridate)
fu<-read_csv("data/football.csv")
fu1<-fu%>%mutate(year=year(date))%>%
  filter(year>=2001&year<=2019)%>%
  filter(home_team%in%c("England", "Germany", "France", "Belgium", "Italy","Spain"))%>%
  select(home_team,year,home_score)%>%
  group_by(home_team,year)%>%
  dplyr::summarise(total=sum(home_score))

# 2. Then, to draw the plot of `fu1`, use:
#     - `ggplot` to initialize a ggplot object and specify the variables to plot.
#     -  use `fill` to fill the color of boxes to `home_team`.
#     - `geom_boxplot` to plot boxplot, set argument `alpha` to 0.6.
#       It will make the color lighter.
#     - `scale_fill_viridis` to use viridis color scale.
#       Set `discrete` to TRUE.
#     - `labs()` to format the labels such that:
#      - `title = "Six European National Teams Home Goals from 2000 to 2019"`
#      - `x = "Teams"`
#      - `y = "Goals Overall (n)"`
#    Save the `ggplot` object to `plot1`.
## Do not modify this line!
plot1<-ggplot(data=fu1,mapping = aes(x=home_team,y=total))+
  geom_boxplot(aes(fill=home_team),alpha=0.6)+
  scale_fill_viridis(discrete=TRUE)+
  labs(title = "Six European National Teams Home Goals from 2000 to 2019",
       x = "Teams",
       y = "Goals Overall (n)")


# 3. Before plotting, first create a tibble `fu2` containing the relevant data.
#    You can use:
#      - `select()` to select the relevant columns.
#        Make sure you use `dplyr::select`.
#      - `mutate()` and `year()` to change the column `date` to its
#        corresponding year
#      - `filter()` to filter years from 2013~2019 and `home_team`
```

```
#       as England.
#    `fu2` should look like the following:
#    # A tibble: 38 x 4
#    home_team home_score date       year
#    <chr>        <dbl> <date>     <dbl>
#    1 England          1 2014-03-05  2014
#    2 England          3 2014-05-30  2014
#    3 England          1 2014-06-14  2014
#    4 England          1 2014-09-03  2014
#    5 England          5 2014-10-09  2014
#    6 England          3 2014-11-15  2014
#    7 England          4 2015-03-27  2015
#    8 England          2 2015-09-08  2015
#    9 England          2 2015-10-09  2015
#    10 England          2 2015-11-17  2015
#    # … with 28 more rows
## Do not modify this line!
fu2<-fu%>%
  select(c('home_team','home_score','date'))%>%
  mutate(year=year(date))%>%
  filter(between(year,2014,2019)&home_team=='England')


# 4. Load the package `ggridges` for plotting.
#    Draw a ridge line plot to analyze the distribution of goals of England in
#    home games from 2013 to 2019.
#    To draw the plot, use:
#     - `ggplot` to initialize a ggplot object and specify the variables to plot.
#       You should plot `home_score` against `year`, group by `home_score`,
#       and set `fill` to `home_score`.
#     - `geom_density_ridges` to plot ridge lines, with argument `alpha` set
#       to 0.6. This will make the color lighter.
#     - `labs()` to format the labels such that:
#      - `title = "England National Team Home Goals Distribution
#                in International Contests from 2013 to 2019"`
#      - `subtitle = "They've crossed the low valley and kept high level recently"`
#      - `x = "Year"`
#      - `y = "England Goals Overall (n)"`
#    Save the `ggplot` object to `plot2`.
## Do not modify this line!
library(ggridges)
plot2<-ggplot(data = fu2,mapping =
aes(x=year,y=home_score,group=home_score,fill=home_score))+
  geom_density_ridges(alpha=0.6)+
  labs(title = "England National Team Home Goals Distribution

       in International Contests from 2013 to 2019",
       subtitle = "They've crossed the low valley and kept high level recently",
       x = "Year",
       y = "England Goals Overall (n)")
#这题中用到了group参数
# 5. Draw a bar plot representing the number of games that
```

```
#   team England, Germany, France, Belgium, Italy and Spain
#   participated as home team in the following contests:
#   'FIFA World Cup', 'UEFA Euro', 'UEFA Nations League', 'Nations Cup'
#   and 'Kirin Cup' since 2000.
#   Before plotting, please do data manipulations to create a tibble `fu3`
#   containing the relevant data.
#   To do that, you can use:
#    - `select()` to select the three columns `"home_team", "tournament", "date"`.
#    - `mutate()` and `factor()` to convert the `tournament` column
#      into factors with given levels.
#    - `drop_na()` to drop the NA values.
#    - `filter()` to get only the relevant teams and years since 2000.
#   It should print to :
#   # A tibble: 89 x 3
#   home_team tournament    date
#   <chr>    <fct>        <date>
#   1 France    FIFA World Cup 2002-05-31
#   2 England   FIFA World Cup 2002-06-02
#   3 Italy     FIFA World Cup 2002-06-03
#   4 France    FIFA World Cup 2002-06-06
#   5 Italy     FIFA World Cup 2002-06-08
#   6 Belgium   FIFA World Cup 2002-06-14
#   7 England   FIFA World Cup 2002-06-21
#   8 France    UEFA Euro      2004-06-13
#   9 England   UEFA Euro      2004-06-17
#   10 Italy    UEFA Euro      2004-06-18
#   # … with 79 more rows
## Do not modify this line!
fu3<-fu%>%
 select(c("home_team","tournament","date"))%>%
 mutate(tournament=factor(tournament,levels = c('FIFA World Cup', 'UEFA Euro', 'UEFA
Nations League', 'Nations Cup','Kirin Cup')))%>%
 drop_na()%>%
 filter(tournament%in%c('FIFA World Cup', 'UEFA Euro', 'UEFA Nations League', 'Nations
Cup','Kirin Cup')&year(date)>=2002
       &home_team%in%c('England', 'France', 'Belgium', 'Italy', 'Spain','Germany'))


# 6. In the bar plot, each group of columns represent a team
#   and different column for different tournament should be
#   shown in different colors.
#   Then, create the plot using :
#    - `ggplot()` to initialize a ggplot object and specify the `home_team`
#      to plot and setting `fill` to tournament.
#    - `geom_bar()` to plot the bar plot, setting argument `position` to
#      `dodge` and `alpha` to `0.75`.
#    - `scale_fill_viridis(discrete=TRUE)` to use the viridis color scale.
#    - `theme()` to set the x-axis label rotate 45 degrees.
#      (Hint : Use `element_text` to set `axis.text.x` correctly)
#    - `labs()` to format the labels such that:
#     - `title = "Six European National Team International Home Game Participation
#       Distribution from 2013 to 2019"`
```

```
#     - `subtitle = "Germany stands out in game numbers in World Cup participation"`
#     - `x = "Teams"`
#     - `y = "Number of Home Games"`
#    Save the `ggplot` object to `plot3`.
## Do not modify this line!
plot3<-ggplot(data=fu3,mapping = aes(x=home_team,fill=tournament))+
  geom_bar(position = "dodge",alpha=0.75)+scale_fill_viridis(discrete=TRUE)+
  theme(axis.text.x=element_text(angle=45))+
  labs(title = "Six European National Team International Home Game Participation

       Distribution from 2013 to 2019",
       subtitle = "Germany stands out in game numbers in World Cup participation",
       x = "Teams",
       y = "Number of Home Games")




# 7. Draw a histogram of the national teams with more than 8 attendances
#    in World Cup history.
#    (Note: you need to calculate the attendance of these teams in all World Cup)
#    Before plotting, please do data manipulations to create a tibble `fu4`.
#    containing the relevant data.
#    To do that, you can use :
#     - `filter()` and `str_detect` to filter by `World Cup` tournaments.
#       Note : Don't take into account the `qualifications` games.
#     - `pivot_longer()` to stretch the game records so that each row is
#       team-wise record.
#     - `group_by()` and `distinct()` to count the attendance of the team
#       in each World Cup year.
#       (Note: it is counted only one attendance no matter how many
#        games the team played in that year)
#     - `dplyr::mutate()` to get the total number of attendances as column `attendance`.
#     - `filter()` to only get the teams with more than 8 attendances.
#     - `arrange()` and `desc()` to order the result in descending order of `attendance`.
#    `fu4` should print to :
#    # A tibble: 244 x 3
#    # Groups:   team [18]
#    year team   attendance
#    <dbl> <chr>      <int>
#    1  1930 Brazil        21
#    2  1934 Brazil        21
#    3  1938 Brazil        21
#    4  1950 Brazil        21
#    5  1954 Brazil        21
#    6  1958 Brazil        21
#    7  1962 Brazil        21
#    8  1966 Brazil        21
#    9  1970 Brazil        21
#    10  1974 Brazil        21
#    # … with 234 more rows
## Do not modify this line!
```

```
fu4<-fu%>%
 filter(str_detect(tournament,'World Cup')&!str_detect(tournament,'qualification'))%>%
 pivot_longer(c(home_team,away_team),names_to = "tmp",values_to = "team")%>%
 mutate(year=year(date))%>%
 select(year,team)%>%
 group_by(team)%>%distinct(year)%>%
 dplyr::mutate(attendance=n())%>%
 filter(attendance>8)%>%arrange(desc(attendance))

# 8. Then, create the histogram. You can use :
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_bar()` to plot histogram plot, setting argument `fill` to
#       `"lightsalmon1"`, `color` to `"black"` and `stat` to `"count"`.
#       This will give boundary to each column. Plus, please
#       set `bins` to 10. (You can change the parameters to see differences).
#     - `coord_flip()` to flip the x,y axis so country names can be shown properly.
#     - `labs()` to format the labels such that:
#       - `title = "National Team Attendance Distribution in World Cup History"`
#       - `x = "Teams"`
#       - `y = "Attendance in World Cup"`
#    Save the `ggplot` object to `plot4`.
## Do not modify this line!
plot4<-ggplot(data = fu4,mapping = aes(x=team))+
 geom_bar(fill="lightsalmon1",color="black",stat="count",bins=10)+coord_flip()+
 labs(title = "National Team Attendance Distribution in World Cup History",
      x = "Teams",
      y = "Attendance in World Cup")
```

2019年12月6日 15:19

HW7

```
# HW7: Dates Manipulation on CitiBike data
#'
# 1. Load the packages `readr`.
#   Use `read_csv()` to create a `tibble` from the csv file at
#   `/course/data/citybike.csv` taken from sampled data from
#   [Citibike usage data](https://www.citibikenyc.com/system-data) and
#   assign it to variable `trips`. The file contains data corresponding to
#   every bike trip taken by users in August 2016.
#   Peek at the first rows of `trips` using `head()` to get a feel of
#   what your data is like and assign the result to tibble `head_trips`.
#   `head_trips` should print to something like
#  # A tibble: 6 x 15
#    tripduration starttime stoptime `start station … `start station …
#           <dbl> <chr>     <chr>              <dbl> <chr>
#  1        5945 8/5/2016… 8/5/201…            228 E 48 St & 3 Ave
#  2        1494 8/3/2016… 8/3/201…            460 S 4 St & Wythe …
#  3         826 8/30/201… 8/30/20…           3301 Columbus Ave & …
#  4         278 8/15/201… 8/15/20…           3256 Pier 40 - Hudso…
#  5         729 8/8/2016… 8/8/201…            347 Greenwich St & …
#  6         502 8/22/201… 8/22/20…            382 University Pl &…
#  # … with 10 more variables: `start station latitude` <dbl>, `start station
#  #   longitude` <dbl>, `end station id` <dbl>, `end station name` <chr>,
#  #   `end station latitude` <dbl>, `end station longitude` <dbl>,
#  #   bikeid <dbl>, usertype <chr>, `birth year` <dbl>, gender <dbl>
## Do not modify this line!
library(readr)
trips<-read_csv("/course/data/citybike.csv")
head_trips<-head(trips)

# 2. Load the `dplyr` package.
#   Modify the `gender` column to a factor column with levels ordered from 0
#   to 1 and the labels (`"Unknown"` for 0, `"Male"` for 1, `"Female"` for 2).
#   Assign the resulting tibble to `trips_with_genders`.
#   Hint: you can use `mutate()` and `factor()`, and judiciously use the
#   `levels` and `labels` arguments of `factor()`.
#   The `gender` column of `trips_with_genders` should print to:
#   # A tibble: 100,000 x 1
#      gender
#      <fct>
#    1 Male
#    2 Male
#    3 Male
#    4 Female
#    5 Female
#    6 Male
#    7 Male
```

```
#    8 Male
#    9 Male
#   10 Male
#   # … with 99,990 more rows
#   And the rest of the tibble should be the same as in question 1.
## Do not modify this line!
library(dplyr)
trips_with_genders<-trips%>%
  mutate(gender=factor(gender,levels = c(0,1,2),labels=c("Unknown","Male","Female")))

# 3. Load the `lubridate` package.
#   Starting with `trips`, modify the `starttime` and `stoptime` columns to
#   contain dates in the format month-day-year_hour-minute-second.
#   Assign the resulting tibble to `trips_with_dates`.
#   Hint: you can use `mutate()` and `mdy_hms()`.
#   The `starttime` and `stoptime` columns of `trips_with_dates` should print
#   to:
#   # A tibble: 100,000 x 2
#     starttime           stoptime
#     <dttm>              <dttm>
#    1 2016-08-05 14:15:11 2016-08-05 15:54:16
#    2 2016-08-03 22:56:34 2016-08-03 23:21:28
#    3 2016-08-30 07:41:07 2016-08-30 07:54:54
#    4 2016-08-15 20:39:47 2016-08-15 20:44:26
#    5 2016-08-08 17:40:31 2016-08-08 17:52:40
#    6 2016-08-22 07:26:03 2016-08-22 07:34:26
#    7 2016-08-31 16:32:42 2016-08-31 16:40:08
#    8 2016-08-10 19:40:36 2016-08-10 19:59:15
#    9 2016-08-24 11:29:54 2016-08-24 11:32:12
#   10 2016-08-30 12:16:21 2016-08-30 12:21:09
#   # … with 99,990 more rows
#   And the rest of the tibble should be the same as in question 1.
## Do not modify this line!
library(lubridate)
trips_with_dates<-trips%>%
  mutate(starttime=mdy_hms(starttime),stoptime=mdy_hms(stoptime))


# 4. Create a tibble containing the `starttime` column from `trips_with_dates`
#   as well as two additional columns `start_ymd` and `start_hour` containing
#   the starting time day and hour of each trip.
#   To do so, you need to:
#   - starting with `trips_with_dates`, keep only the `starttime` column,
#     (you can use `select()` for this)
#   - then create the two columns from `starttime`, (you can use `mutate()`
#     along with `floor_date()` and `hour()` for this)
#   - assign the result to tibble `trips_start_times`
#   # A tibble: 100,000 x 3
#     starttime           start_ymd           start_hour
#     <dttm>              <dttm>                   <int>
#    1 2016-08-05 14:15:11 2016-08-05 00:00:00         14
#    2 2016-08-03 22:56:34 2016-08-03 00:00:00         22
```

```
#     3 2016-08-30 07:41:07 2016-08-30 00:00:00         7
#     4 2016-08-15 20:39:47 2016-08-15 00:00:00        20
#     5 2016-08-08 17:40:31 2016-08-08 00:00:00        17
#     6 2016-08-22 07:26:03 2016-08-22 00:00:00         7
#     7 2016-08-31 16:32:42 2016-08-31 00:00:00        16
#     8 2016-08-10 19:40:36 2016-08-10 00:00:00        19
#     9 2016-08-24 11:29:54 2016-08-24 00:00:00        11
#    10 2016-08-30 12:16:21 2016-08-30 00:00:00        12
#    # … with 99,990 more rows
## Do not modify this line!
trips_start_times<-trips_with_dates%>%select(starttime)%>%
  mutate(start_ymd = floor_date(starttime,unit = "days"),start_hour=hour(starttime))


# 5. Compute 1) the total number of trips, 2) the number of days in which there
#    was at least one trip and 3) the average number of trips that started
#    for each hour of the day over the whole month. Assign the resulting
#    tibble to `trips_per_hour`.
#    To do so, you need to:
#    - starting with `trips_start_times`, group by the hour of the day, (you
#      can use `group_by()` for this)
#    - then compute the three statistics listed above (you can use
#      `summarize()` for this)
#    Hint:  The function `n()` returns the number of rows, and
#    `n_distinct(column)` returns the number of distinct values in `column`.
#    `trips_per_hour` should print to:
#    # A tibble: 24 x 4
#       start_hour num_trips num_days mean_trips
#            <int>     <int>    <int>      <dbl>
#    1          0       986       31      31.8
#    2          1       506       31      16.3
#    3          2       322       31      10.4
#    4          3       193       30       6.43
#    5          4       195       31       6.29
#    6          5       587       31      18.9
#    7          6      2262       31      73.0
#    8          7      4784       31     154.
#    9          8      7958       31     257.
#   10          9      6528       31     211.
#    # … with 14 more rows
## Do not modify this line!
trips_per_hour<-trips_start_times%>%group_by(start_hour)%>%
summarize(num_trips=n(),num_days=n_distinct(start_ymd),mean_trips=num_trips/num_da
ys)


# HW7: Strings and Dates
#'
# In this exercise, you will manipulate a dataset with strings and factors.
#'
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
```

```
#    - Do not change the default position of the plot title.
#    - Do not print the plot.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Load the `tidyverse`, `rattle`,  `ggplot2` and `lubridate` packages.
#    Use `data(weather)` to read the dataset `weather` from the
#    `rattle` package.
#    Store it in a tibble `weather` using `as_tibble()`.
## Do not modify this line!
library(tidyverse)
library(rattle)
library(ggplot2)
library(lubridate)
data(weather)
weather<-as_tibble(weather)



# 2. Create two tibbles `weather_9am` and `weather_3pm`.
#    `weather_9am` should contain all the variables which names end with
#    `"9am"` and the column `Date`. `"9am"` should be removed from the
#    names of its variables and the format of `Date`` should be changed as follow :
#    Instead of `"Y-M-D"`, add `"Y-M-D h:m:s AECT"` where `"AECT"` is the time zone
#    code for Australian Eastern Time and `h:m:s` is `09:00:00` (9 am).
#    For example, the Date column first value will be `"2007-11-01 09:00:00 AEDT"`.
#    `weather_3pm` should contain all the variables which names end with
#    `"3pm"` and the column `Date`. `"3pm"` should be removed from the
#    names of its variables and the format of `Date`` should be changed as follow :
#    Instead of `"Y-M-D"`, add `"Y-M-D h:m:s AECT"` where `"AECT"` is the time zone
#    code for Australian Eastern Time and `h:m:s` is `15:00:00` (3 pm).
#    For example, the Date column first value will be `"2007-11-01 15:00:00 AEDT"`.
#    To do so, you can use :
#      - `select()` and `ends_with()` to select the right columns
#      - `rename_all()` and `str_remove()` to rename the columns correctly
#      - `mutate()`, `ymd_hms()` and `paste()` to change the format of `Date`
#        Hint : If you use  `ymd_hms()`, set `tz` to `"Australia/Canberra"`
#    `weather_9am` should print to :
#    # A tibble: 366 x 7
#    Date                WindDir WindSpeed Humidity Pressure Cloud  Temp
#    <dttm>              <ord>      <dbl>   <int>    <dbl> <int> <dbl>
#    1 2007-11-01 09:00:00 SW           6      68    1020.     7  14.4
#    2 2007-11-02 09:00:00 E            4      80    1012.     5  17.5
#    3 2007-11-03 09:00:00 N            6      82    1010.     8  15.4
#    4 2007-11-04 09:00:00 WNW         30      62    1006.     2  13.5
#    5 2007-11-05 09:00:00 SSE         20      68    1018.     7  11.1
#    6 2007-11-06 09:00:00 SE          20      70    1024.     7  10.9
#    7 2007-11-07 09:00:00 SE          19      63    1025.     4  12.4
```

```
#   8 2007-11-08 09:00:00 SE          11      65    1026.     6  12.1
#   9 2007-11-09 09:00:00 E           19      70    1026.     7  14.1
#   10 2007-11-10 09:00:00 S           7      82    1024.     7  13.3
#   # … with 356 more rows
#'
## Do not modify this line!
weather_9am<-weather%>%
 select(Date,ends_with("9am"))%>%
 rename_all(~str_remove(.,"9am"))%>%
 mutate(Date=ymd_hms(paste(Date,"9:0:0"),tz="Australia/Canberra"))
weather_3pm<-weather%>%
 select(Date,ends_with("3pm"))%>%
 rename_all(~str_remove(.,"3pm"))%>%
 mutate(Date=ymd_hms(paste(Date,"15:0:0"),tz="Australia/Canberra"))
# 3. Join the tibbles `weather_9am` and `weather_3pm` to create a tibble
#   `weather_arrange` that contains the data in both `weather_9am` and
#   `weather_3pm`. However, its `Date` should change to become the
#   corresponding day of the year, and be named `"Day of the year"`.
#   For example, `"2007-11-23 15:00:00 AEDT"` will become `23`.
#   Then, order the tibble from the earliest `Day of the year` to the latest.
#   To do so, you can use :
#     - `full_join()` to join the datasets
#     - `mutate()` and `yday()` to change the `Date`
#     - `rename()` to change the name of `Date`
#     - `arrange()` to reorder the tibble
#   `weather_arrange` should print to :
#   # A tibble: 732 x 7
#   `Day of the year` WindDir WindSpeed Humidity Pressure Cloud  Temp
#   <dbl> <ord>      <dbl>    <int>    <dbl> <int> <dbl>
#   1             1 ESE          2      50    1016.     7  21.9
#   2             1 W           11      20    1012.     6  31.8
#   3             2 ESE          2      43    1013.     0  23
#   4             2 WSW          9      14    1009.     1  33.6
#   5             3 ESE         20      69    1017.     8  19.2
#   6             3 ESE         24      55    1016.     7  22.3
#   7             4 ESE         24      56    1016.     6  20.3
#   8             4 ESE         26      45    1013      7  23.9
#   9             5 SSE          7      69    1012.     7  18.6
#   10            5 E           15      40    1007.     2  26.8
#   # … with 722 more rows
#'
## Do not modify this line!
weather_arrange<-full_join(weather_9am,weather_3pm)%>%
 mutate(Date=yday(Date))%>%
 rename(`Day of the year` = Date)%>%
 arrange(`Day of the year`)


# 4. Plot both the smoothed temperature `Temp` multiplied by 3,
#   and `Humidity` as function of the date.
#   To do that, you can use :
#     - `ggplot()` to initialize a ggplot object. You can set its arguments
```

```
#       `data` and `mapping` to plot from `weather_arrange`, with
#       `Day of the year` as its x-axis
#     - `geom_smooth()` to plot the smoothed `Humidity`, with `col` set
#       to `green`, `method` set to 'loess', and `formula` to `'y ~ x'`
#       (These are the approximation methods for the smoothing process)
#     - `geom_smooth()` to plot the smoothed `3*Temp`, with `col` set
#       to `yellow`, `method` set to 'loess', and `formula` to `'y ~ x'`
#     - `labs()` to set :
#       - `title` to `"Temperature and Humidity are negatively correlated in Canberra"`
#       - `x` to `"Day of the year"`
#       - `y` to `"Humidity (in green) and rescaled Temperature (in yellow)"`
#     - `theme_light()`
#   Save the plot into `temp_humidity_plot`.
## Do not modify this line!
temp_humidity_plot<-ggplot(weather_arrange,mapping = aes(x=`Day of the
year`,y=Humidity))+geom_smooth(color = "green",method = "loess",formula = 'y ~ x')+
 geom_smooth(mapping = aes(y=3*Temp),color="yellow",method = "loess",formula = 'y ~
x')+
 labs(title = "Temperature and Humidity are negatively correlated in Canberra",
     x="Day of the year",
     y="Humidity (in green) and rescaled Temperature (in yellow)")+theme_light()


# 5. Create a tibble `weather_wind` that contains the wind direction `WindDir`,
#   the temperature `Temp`, and the main wind direction `WindMainDir`, for each
#   date and time (we have two recordings a day at 9am and 15pm.)
#   It should be ordered from lowest temperature to highest, and not contain
#   any `NA` value. `WindMainDir` should correspond to the first letter of
#   `WindDir` and be converted to `factor`. For example, `SW` main direction is `S`.
#   To do that, please create two tibbles `weather_wind_9am` and `weather_wind_3pm`
#   that contain the columns `WindDir` and `Temp` from `weather_9am` ans
#   `weather_wind_3pm`. You can use `select()`.
#   `weather_wind_9am` should print to :
#   # A tibble: 366 x 2
#   WindDir  Temp
#   <ord>   <dbl>
#   1 SW      14.4
#   2 E       17.5
#   3 N       15.4
#   4 WNW     13.5
#   5 SSE     11.1
#   6 SE      10.9
#   7 SE      12.4
#   8 SE      12.1
#   9 E       14.1
#   10 S      13.3
#   # … with 356 more rows
#   `weather_wind_3pm` has the same columns, but with different values.
#   Then, join these two tibbles into a tibble `weather_wind` using `full_join()`.
#   You can then :
#     - use `drop_na()` to drop the `NA` values
#     - use `mutate()`, `factor()` and `str_sub()` to create the column `WindMainDir`.
```

```
#      - use `arrange()` to reorder the dataset.
#    `weather_wind` should print to :
#    # A tibble: 667 x 3
#    WindDir  Temp WindMainDir
#    <ord>   <dbl> <fct>
#    1 N        0.1 N
#    2 E        0.8 E
#    3 SSE      1   S
#    4 SE       1.2 S
#    5 SE       1.4 S
#    6 NNW      1.4 N
#    7 NW       1.8 N
#    8 N        2.1 N
#    9 ESE      2.6 E
#    10 SE      2.7 S
#    # … with 657 more rows
## Do not modify this line!

weather_wind_9am<-weather_9am%>%select(WindDir,Temp)
weather_wind_3pm<-weather_3pm%>%select(WindDir,Temp)
weather_wind<-full_join(weather_wind_9am,weather_wind_3pm)%>%drop_na()%>%
  mutate(WindMainDir=factor(str_sub(WindDir,1,1)))%>%arrange(Temp)
# 6. Plot the facetted histogram of the temperature `Temp` for each main
#    wind direction `WindMainDir`. Add a vertical line for temperature
#    29 degrees Celsius to see what winds bring hot temperatures.
#    To do this, you can use :
#      - `ggplot()` to initialize a ggplot object. You can set its arguments
#        `data` and `mapping` to plot from `weather_arrange`, with
#        `Temp` as its x-axis
#      - `geom_histogram()` to create the histograms of `Temperature`, with
#        `color` set as `black`, `fill` as `orange` and `binwidth` as `0.5`.
#      - `geom_vline()` to draw the vertical line with `aes(xintercept = 29)`
#        and `color` set to `green`
#      - `facet_wrap()` to facet over `WindMainDir`
#      - `labs()` to set :
#        - `title` to `"The winds going west and north bring the highest temperatures"`
#        - `x` to `"Temperature (Celsius)"`
#      - `theme_light()`
#   Save this `ggplot` object to `wind_histogram`.
## Do not modify this line!
wind_histogram<-ggplot(weather_wind,mapping = aes(x=Temp))+
  geom_histogram(color='black',fill='orange',binwidth = 0.5)+geom_vline(aes(xintercept =
29),color='green')+facet_wrap(~WindMainDir)+
  labs(title="The winds going west and north bring the highest temperatures",
       x="Temperature (Celsius)")+theme_light()



# HW7: Reproducing a paper figure
#'
# This exercise was inspired by exercise 6 in Chapter 2 of
# [Bit By Bit: Social Research in the Digital Age]
```

(https://www.bitbybitbook.com/en/1st-ed/observing-behavior/observing-activities/)
by Matt Salganik.
#'
"In a widely discussed paper, Michel and colleagues
([2011](https://doi.org/10.1126/science.1199644)) analyzed the content of
more than five million digitized books in an attempt to identify long-term
cultural trends. The data that they used has now been released as the Google
NGrams dataset, and so we can use the data to replicate and extend some of
their work.
#'
In one of the many results in the paper, Michel and colleagues argued that we
are forgetting faster and faster. For a particular year, say “1883,” they
calculated the proportion of all terms published in each year between 1875
and 1975 that were “1883”. They reasoned that this proportion is a measure of
the interest in events that happened in that year. In their figure 3a, they
plotted the usage trajectories for three years: 1883, 1910, and 1950. These
three years share a common pattern: little use before that year, then a
spike, then decay."
#'
They noticed the rate of decay for each year mentioned seemed to increase
with time and they argued that this means that we are forgetting the past
faster and faster.
#'
The full paper can be found
[here](https://aidenlab.org/papers/Science.Culturomics.pdf), and you are
going to replicate part of figure 3a.
#'
To do so we will focus on the mention of terms that can represent years
(strings like "1765", "1886", "1897", "1937"...). The raw data was fetched
for you from the [Google Books NGram Viewer website]
(http://storage.googleapis.com/books/ngrams/books/datasetsv2.html) and
preprocessed into two files:
- `data/mentions_yearly_counts.tsv` contains the number of mentions of
different terms per year and the number of books retrieved where the term
appeared each year
(one row per term per year)
- `data/yearly_total_counts.csv` contains the total number of mentions of all
terms per year as well as the number of pages and books retrived each year
(one row per year)
#'
1. Load the `readr` package.
Read in `data/mentions_yearly_counts.tsv` using `read_tsv()` and assign
the resulting tibble to `terms_mentions`.
Set the parameters of `read_tsv()` in order to make sure of the following:
- column names should be, in order: `term`, `year`, `n_mentions`,
`book_count`
- column types should be, in order: `character`, `integer`, `integer`,
`integer`
Hint: you can use parameters `col_names` and `col_types` to achieve this.
`terms_mentions` should print to:
A tibble: 53,393 x 4
term   year n_mentions book_count

```
#     <chr> <int>    <int>    <int>
#   1 1817   1524       31        1
#   2 1817   1575       17        1
#   3 1817   1607        3        1
#   4 1817   1637        2        1
#   5 1817   1662        1        1
#   6 1817   1675        5        1
#   7 1817   1693        8        1
#   8 1817   1705        1        1
#   9 1817   1708        1        1
#  10 1817   1713        1        1
#  # … with 53,383 more rows
## Do not modify this line!
library(readr)
terms_mentions<-read_tsv('data/mentions_yearly_counts.tsv',col_names=FALSE,col_types
= cols('c','i','i','i'))
colnames(terms_mentions)<-c("term", "year", "n_mentions","book_count")

# 2. Read in `data/yearly_total_counts.csv` using `read_csv()` and assign the
#   resulting tibble to `total_mentions`.
#   Set the parameters of `read_csv()` in order to make sure of the following:
#   - column names should be, in order: `year`, `total_mentions`,
#     `total_page_count`, `total_book_count`
#   - column types should be, in order: `integer`, `double`, `integer`,
#     `integer`
#   Hint: you can use parameters `col_names` and `col_types` to achieve this.
#   Note: the reason you should read in the `total_mentions` as a `double`
#   column is that it contains very large integers that don't fit within the
#   bounds of numbers represented by the `integer` type in R. Using a
#   double-precision number is our only recourse.
#   `total_mentions` should print to:
#   # A tibble: 425 x 4
#      year total_mentions total_page_count total_book_count
#     <int>        <dbl>         <int>         <int>
#   1  1505        32059           231             1
#   2  1507        49586           477             1
#   3  1515       289011          2197               1
#   4  1520        51783           223             1
#   5  1524       287177          1275               1
#   6  1525         3559            69             1
#   7  1527         4375            39             1
#   8  1541         5272            59             1
#   9  1563       213843           931               1
#  10  1564        70755           387             1
#  # … with 415 more rows
## Do not modify this line!
total_mentions<-read_csv('data/yearly_total_counts.csv',col_names = FALSE,col_types =
cols('i','d','i','i'))
colnames(total_mentions)<-c("year",'total_mentions', 'total_page_count',
'total_book_count')
```

```
# 3. Load the `dplyr` package. In order to join the `total_mentions`
#    Left join the `total_mentions` on `terms_mentions` by `year` and assign
#    the resulting tibble to `mentions`.
#    Hint: you can use `left_join()`.
#    `mentions` should print to:
#    # A tibble: 53,393 x 7
#       term   year n_mentions book_count total_mentions total_page_count
total_book_count
#      <chr> <int>     <int>      <int>        <dbl>            <int>
<int>
#    1 1817   1524        31          1       287177             1275
1
#    2 1817   1575        17          1       186706             1067
1
#    3 1817   1607         3          1       381763             1600
2
#    4 1817   1637         2          1       681719             2315
3
#    5 1817   1662         1          1       239762             1471
3
#    6 1817   1675         5          1      1644156             8918
14
#    7 1817   1693         8          1      1038415             7426
16
#    8 1817   1705         1          1      4908749            28840
60
#    9 1817   1708         1          1      6481151            37416
70
#   10 1817   1713         1          1      4720647            25961
77
#    # … with 53,383 more rows
## Do not modify this line!
library(dplyr)
mentions<-terms_mentions%>%
  left_join(total_mentions,by = 'year')


# 4. Check that your join was successful by using `anti_join()` to drop all
#    observations in `terms_mentions` that have a match in `mentions` and
#    assign the resulting tibble to `diagnosis`. If the join went as expected
#    `diagnosis` should be an empty tibble and print to:
#    # A tibble: 0 x 4
#    # … with 4 variables: term <chr>, year <int>, n_mentions <int>,
#    # book_count <int>
## Do not modify this line!
diagnosis<-anti_join(terms_mentions,mentions)


# 5. Do the following:
#    - starting with `mentions`, add a column `frac_total` that computes the
#    frequency of mentions of each term per year (divides the number of
#    mentions of each term per year by the total number of mentions of all
#    terms that year),
#    - select only columns `term`, `year`, `n_mentions`, `total_mentions` and
#    `frac_total`.
#    Assign the resulting tibble to `relative_mention_counts`.
#    Hint: you can use `mutate()` and `select()`.
#    `relative_mention_counts` should print to:
#    # A tibble: 53,393 x 5
#       term   year n_mentions total_mentions  frac_total
#      <chr> <int>     <int>        <dbl>      <dbl>
#    1 1817   1524        31       287177 0.000108
#    2 1817   1575        17       186706 0.0000911
```

```
#    3 1817   1607         3        381763 0.00000786
#    4 1817   1637         2        681719 0.00000293
#    5 1817   1662         1        239762 0.00000417
#    6 1817   1675         5       1644156 0.00000304
#    7 1817   1693         8       1038415 0.00000770
#    8 1817   1705         1       4908749 0.000000204
#    9 1817   1708         1       6481151 0.000000154
#   10 1817   1713         1       4720647 0.000000212
#   # … with 53,383 more rows
## Do not modify this line!
relative_mention_counts<-mentions%>%
mutate(frac_total=n_mentions/total_mentions)%>%
select(term,year,n_mentions,total_mentions,frac_total)


# 6. Load the `forcats` package.
#   To prepare the tibble to build the figure with:
#   - keep only the terms `"1883"`, `"1910"` and `"1950"`,
#   - transform the terms from characters to a factor in which the levels
#     are in reversed alphabetical order ("1950", "1910" and "1883")
#   Assign the result to `examples_mention_counts`.
#   Hint: you can use `filter()`, `mutate()` and `fct_rev()` to reverse the
#   order of levels of a factor.
#   Note: the order matters to us to reproduce the same colors as the original
#   figure without setting them explicitely when generating the plot.
#   `examples_mentions_counts` should print to:
#   # A tibble: 825 x 5
#      term   year n_mentions total_mentions frac_total
#     <fct> <int>      <int>         <dbl>      <dbl>
#    1 1883   1515         1        289011 0.00000346
#    2 1883   1520         1         51783 0.0000193
#    3 1883   1524        15        287177 0.0000522
#    4 1883   1574         4         62235 0.0000643
#    5 1883   1575         3        186706 0.0000161
#    6 1883   1584         1        151925 0.00000658
#    7 1883   1607         2        381763 0.00000524
#    8 1883   1637         5        681719 0.00000733
#    9 1883   1643         1        177489 0.00000563
#   10 1883   1644         3       1018174 0.00000295
#   # … with 815 more rows
## Do not modify this line!
library(forcats)
examples_mention_counts<-relative_mention_counts%>%filter(term%in%
c(1883,1950,1910))%>%
  mutate(term=fct_rev(term))


# 7. Load the `ggplot2` and `scales` packages.
#   Generate a plot to reproduce the large window of figure 3a and assign the
#   result to `paper_figure`.
#   To do so, you can, in the following order:
#   - create a plot from `examples_mention_counts` using `ggplot()` with the
```

```
#    appropriate columns assigned in the aesthetic's `x`, `y` and `color`
#    arguments,
#    - add the lines using `geom_line()`,
#    - add `scale_y_continuous(label = percent)` to set the y-axis ticks to
#      the percent format,
#    - limit the coordinates to show only the mentions across the timeframe
#      `1850`-`2012` using `coord_cartesian()` and its argument `xlim`,
#    - use `labs()` to:
#        - set `title` to `"Are we forgetting the past faster?"`
#        - set `x` to `"Year"`
#        - set `y` to `"Frequency of mention of each term"`
#        - set `color` to `Term`
#    - finally add `theme_light()` for a clear plot
## Do not modify this line!
library(ggplot2)
library(scales)
paper_figure<-ggplot(examples_mention_counts,mapping =
aes(x=year,y=frac_total))+geom_line(aes(color=term))+scale_y_continuous(label =
percent)+
  coord_cartesian(xlim = c(1850,2012))+
  labs(title="Are we forgetting the past faster?",
       x="Year",
       y="Frequency of mention of each term",
       color='Term')+theme_light()


# HW7: Relational data
#'
# In this exercise, you will familiarize yourself with
# relational data and its manipulation.
#'
# Throughout the exercise:
#    - Use `theme_light()` for the plots.
#    - Do not change the default position of the plot title.
#    - Do not print the plot.
#'
# In this exercise, you have to recreate the figures found at
# the left of the instructions.
# We suggest functions you can use to create the plots, but
# you are free to use the method you are the most comfortable with.
# Make sure that the figures look exactly like the ones you are supposed to create.
#'
# 1. Load the `tidyverse`, `readr`, and `lubridate` packages.
#    Use the function `read_csv()` to read the dataset
#    `booking.csv`, `guest.csv`, `rate.csv`, and `room.csv`
#    from path `data/guest_house/`.
#    Store the corresponding dataframe into a tibble
#    `booking`, `guest`, `rate`, and `room`.
## Do not modify this line!
library(tidyverse)
library(readr)
library(lubridate)
```

```
booking<-read_csv('data/guest_house/booking.csv')
guest<-read_csv('data/guest_house/guest.csv')
rate<-read_csv('data/guest_house/rate.csv')
room<-read_csv('data/guest_house/room.csv')
# 2. Create a tibble `room_earning` from table `booking` and `rate`
#    using `left_join()`.
#    For each `booking_date` and `room_no`, there should be a column
#    `earning` containing the earning of this room.
#    The column `booking_date` should be class `"Date"`.
#    To do that, you can use:
#    - `left_join()` to join tables `booking` and `rate`.
#      Note that you should join by two columns.
#    - `mutate()` to create a new column `earning` that
#      is the multiplication of `nights` and `amount`.
#    - `mdy()` to transform the `booking_date` to class `"Date"`.
#    - `arrange()` to order the tibble.
#    - `select()` to select the columns.
#    The first rows of its print should be :
#    # A tibble: 347 x 5
#      booking_date room_no guest_id nights earning
#      <date>       <dbl>   <dbl> <dbl>  <dbl>
#    1 2016-11-03     101    1027     7    336
#    2 2016-11-03     102    1179     2    112
#    3 2016-11-03     103    1106     2    144
#    4 2016-11-03     104    1238     3    168
#    5 2016-11-03     105    1540     7    588
#    6 2016-11-03     106    1021     3    168
#    7 2016-11-03     107    1623     3    168
#    8 2016-11-03     108    1136     1     56
#    9 2016-11-03     109    1585     4    288
#   10 2016-11-03      201    1613     6    288
#    # … with 337 more rows
## Do not modify this line!
room_earning<-left_join(booking,rate,by =
c('room_type_requested'='room_type','occupants'='occupancy'))%>%
  mutate(earning=nights*amount,booking_date = mdy(booking_date))%>%
arrange(booking_date)%>%select(booking_date,room_no,guest_id,nights,earning)


# 3. Create a tibble `room_earning_filtered` from `room_earning`.
#    This tibble will only have data whose `room_no` is
#    smaller than `200` and `booking_date` between `2016-11-15`
#    and `2019-12-15`(including both).
#    To do that, you can use:
#    - `filter()` to filter the `room_no` that is smaller than
#      `200`.
#    - `filter()` to filter the date.
#    - `mutate()` and `as.character()` to transform the `room_no`
#      from `"numeric"` to `"character"`.
#    The first rows of its print should be :
#    # A tibble: 76 x 5
#      booking_date room_no guest_id nights earning
```

```
#    <date>      <chr>    <dbl> <dbl>  <dbl>
#   1 2016-11-15  101       1344     1    48
#   2 2016-11-15  105       1127     5   280
#   3 2016-11-15  103       1041     4   288
#   4 2016-11-15  109       1624     1    72
#   5 2016-11-15  102       1598     2   144
#   6 2016-11-15  106       1208     5   280
#   7 2016-11-16  101       1185     1    48
#   8 2016-11-16  109       1249     3   216
#   9 2016-11-17  101       1187     5   240
#  10 2016-11-17  104        1477     2    144
#   # … with 66 more rows
## Do not modify this line!

room_earning_filtered<-room_earning%>%filter(room_no<200)%>%
filter(booking_date>='2016-11-15'&booking_date<='2019-12-15')%>%
  mutate(room_no=as.character(room_no))%>%arrange(booking_date)

# 4. Plot line plot: the earnings of room by date
#    using `room_earning_filtered`.
#    The x axis should be date labelled as `"Booking Date"`
#    with ticks `"11/15"`, `"12/01"`, `"12/15"`.
#    The y axis should be earning labelled as `"Earning (dollars)"`
#    with ticks `100`, `200`, `300` (default setting).
#    To do that, you can use:
#    - `ggplot()` to initialize a ggplot object. You can set its arguments
#      `data` and `mapping` to plot the `booking_date` and `earning` column
#      of the `room_earning_filtered`.
#      Use `aes` to set parameters `mapping`.
#    - `geom_line()` to plot the line plot.
#    - `scale_x_date()` to set the ticks of x axis to
#      `"11/15"`, `"12/01"`, `"12/15"`.
#    - `labs()` to format the labels such that:
#      - `title = "Most rooms have earning around 100 to 300"`
#      - `x = "Booking date"`
#      - `y = "Earning (dollars)"`
#    - `facet_wrap` to be faceted by `room_no`.
#    - `theme_light()` to set light theme (i.e. a light backgroung).
#    Save the plot into `room_earning_plot`.
## Do not modify this line!
room_earning_plot<-ggplot(room_earning_filtered,mapping =
aes(x=booking_date,y=earning))+geom_line()+
  scale_x_date(breaks = c(ymd("2016-11-15"), ymd("2016-12-01"),ymd("2016-12-15")),
               date_labels = '%m/%d')+
  labs(title = "Most rooms have earning around 100 to 300",
       x = "Booking date",
       y = "Earning (dollars)")+facet_wrap(~room_no)+theme_light()
?scale_x_date
#注意这个breaks的写法

# 5. Create a tibble `guest_spending` that stores the total spending and
#    total number of nights staying for each guest by their full name
```

```
#   from tibble `guest` and `room_earning` using `right_join`.
#   You can use :
#    - `mutate()` and `paste()` to concatenate `first_name` and
#      `last_name` in `guest`.
#    - `right_join()` to join the aforementioned tibble with
#      `room_earning`.
#    - `group_by()` to group by name.
#    - `summarize()` to compute the total spending `spending`
#      and total number of nights staying `nights`.
#    - `ungroup()` to ungroup the tibble.
#    - `top_n()` to select 10 rows with top spendings.
#    - `arrange()` to order the tibble by descending order of
#      total spending.
#    - `fct_reorder()` to set `name` to factor and order the
#      level by ascending order of total spending.
#   The first rows of its print should be :
#   # A tibble: 10 x 3
#     name               nights spending
#     <fct>               <dbl>    <dbl>
#   1 Sir Edward Garnier       11      780
#   2 Robert Halfon            10      768
#   3 Angela Rayner            14      744
#   4 Karin Smyth              11      696
#   5 Sir Alan Haselhurst      11      680
## Do not modify this line!
guest_spending<-right_join(guest,room_earning,by = c("id"="guest_id"))%>%
mutate(name=paste(first_name,last_name))%>%group_by(name)%>%
 summarize(nights=sum(nights),spending=sum(earning))%>%top_n(10)%>%
arrange(desc(spending))%>%
 mutate(name=fct_reorder(name,spending))

# 6. Plot the bar plot of `spending` by `name` using
#   dataset `guest_spending`.
#   The bars should be horizontal and ordered by total spending
#   from highest (on top of the figure) to lowest.
#   The x axis should be `"Spending (dollars)"` without label
#   with ticks `0`, `200`, `400`, `600`, `800`(default setting).
#   The y axis should be  `"Name"` with ticks
#   `"Sir Edward Garnier"`, `"Robert Halfon"`, `"Angela Rayner"`,
#   ..., `"Craig Tracey"` from top to bottom(default setting).
#   To do that, use:
#   - `ggplot()` to initialize a ggplot object.
#     Set the `date` and `mapping` parameter correctly.
#   - `geom_col()` to plot the box plot.
#   - `coord_flip()` to flip the axises.
#   - `labs()` to format the labels such that:
#     - `title = "Top 10 guests all spent more than 600"`
#     - `y = "Spending (dollars)"`
#     - `x = "Name"`
#   - `theme_light()` to set light theme.
#   Save the plot into `guest_spending_plot`.
## Do not modify this line!
```

```
guest_spending_plot<-ggplot(guest_spending,mapping =
aes(x=name,y=spending))+geom_col()+coord_flip()+
  labs(title = "Top 10 guests all spent more than 600",y = "Spending (dollars)",x =
"Name")+theme_light()


# HW7: relational data
#'
# In this exercise, we will walk you through a complete process of manipulating
# relational data. We suggest the functions you can use to create the tibbles
# and the plots, but you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#'
# Throughout the exercise:
#    - Do NOT use `for`, `while` or `repeat` loops.
#    - Use `%>%` to structure your operations.
#    - Use `theme_light()` for the plots.
#'
# 1. Load the packages `tidyverse`, `lubridate` and `GGally`.
#    Use `read_csv()` to read the datasets:
#      - `data/department_info.csv` into a tibble `department`.
#      - `data/employee_info.csv` into a tibble `employee`.
#      - `data/student_counselling_info.csv` into a tibble `s_admission`.
#      - `/course/data/student_performance_info.csv` into a tibble `s_performance`.
#    All datasets are located in the directory `data/` with the exception of the last one.
#    Background:
#      - `department` has information about departments in the university,
#        including `Department_ID`, `Department_Name` and `DOE` (date of
#        establishment).
#      - `employee` has information about professors, including `Employee_ID`,
#        `DOB` (date of birth), `DOJ` (date of university joining) and
#        `Department_ID` where the professors belong to.
#      - `s_admission` has information about admission of students, including
#        `Student_ID`, `DOA` (date of admission), `DOB`, `Department_Choices`
#        (choices student made during counselling) and `Department_Admission`
#        (department offered to student).
#      - `s_performance` has information about performance details of a student,
#        including `Student_ID`, `Semester_Name`, `Paper_ID`, `Paper_Name` and
#        `Marks` (marks scored in examination).
#    After preprocessing all the tibbles, we will have a final tibble `engie`,
#    with integrated information of students at the Engineering School.
## Do not modify this line!
library(tidyverse)
library(lubridate)
library(GGally)
department<-read_csv("data/department_info.csv")
employee<-read_csv("data/employee_info.csv")
s_admission<-read_csv("data/student_counselling_info.csv")
s_performance<-read_csv("/course/data/student_performance_info.csv")
# 2. We will start by cleaning and aggregating some new columns for our interest.
#    Create a tibble `employee_new` of size 1000 x 5 with columns `Employee_ID`,
#    `Department_ID`, `age`, `seniority` and `seniority_level`, where `age` is
```

```
#   the age of the professor, `seniority` is the year of teaching, and
#   `seniority_level` shows whether a professor is junior (`seniority` <= 10)
#   or senior (`seniority` > 10).
#   To do this, you can use:
#     - `mutate()` to create the columns
#       - `age` and `seniority` from `DOB` and `DOJ` repectively by
#         - Creating an interval with `%--%`, `today()` and `dyears()`.
#         - Rounding the years to integers with `round()`.
#       - `seniority_level` to categorize `seniority` by
#         - Cutting `seniority` with `cut()`, setting
#           - `breaks = min(seniority), 10, max(seniority)`;
#           - `labels = c("junior", "senior")`;
#           - `include.lowest = TRUE`.
#     - `dplyr::select()` to select the desired columns.
#   To check your result, `employee_new` prints to:
#   # A tibble: 1,000 x 5
#     Employee_ID Department_ID   age seniority seniority_level
#     <chr>       <chr>         <dbl>    <dbl> <fct>
#    1 IU196557    IDEPT4938        37       10 junior
#    2 IU449901    IDEPT2357        34       10 junior
#    3 IU206427    IDEPT4670        48       11 senior
#    4 IU688905    IDEPT2601        46       18 senior
#    5 IU634582    IDEPT7626        28       20 senior
#    6 IU138624    IDEPT3778        40       13 senior
#    7 IU932068    IDEPT4132        34        6 junior
#    8 IU572796    IDEPT1142        42       12 senior
#    9 IU134670    IDEPT7626        39       18 senior
#   10 IU393717    IDEPT1825        47       10 junior
#   # … with 990 more rows
## Do not modify this line!
employee_new<-employee%>%
  mutate(age=round(DOB%--%today()/dyears(1)),
       seniority=round(DOJ%--%today()/dyears(1)),
       seniority_level=cut(seniority,breaks = c(min(seniority), 10, max(seniority)),labels =
c("junior", "senior"),
           include.lowest = TRUE))%>%
  select(-DOJ,-DOB)

# 3. Create a tibble `s_admission_new` of size 4000 x 4 with columns `Student_ID`,
#   `Department_Admission`, `age` and `school_year`, where `age` is the age of
#   the student and `school_year` is the class standing, which can be "graduate",
#   "senior", "junior", sophomore" or "freshman".
#   To do this, you can use:
#     - `mutate()` to create the columns
#       - `age` from `DOB` by
#         - Creating an interval with `%--%`, `today()` and `dyears()`.
#         - Rounding the years to integers with `round()`.
#       - `school_year` that categorizes `DOA` by
#         - Making `DOA` a factor with `factor()`.
#         - Collapsing the levels "2013-07-01" and "2014-07-01" into
#           "graduate" with `fct_collapse()`.
#         - Recoding the other levels with `fct_recode()` to make
```

```
#           - "2015-07-01" into "senior";
#           - "2016-07-01" into "junior";
#           - "2017-07-01" into "sophomore";
#           - "2018-07-01" into "freshman".
#     - `dplyr::select()` to select the desired columns.
#   To check your result, `s_admission_new` prints to:
#   # A tibble: 4,000 x 4
#     Student_ID  Department_Admission   age school_year
#     <chr>      <chr>              <dbl> <chr>
#    1 SID20131143 IDEPT7783            24 graduate
#    2 SID20131151 IDEPT6347            24 graduate
#    3 SID20131171 IDEPT1836            24 graduate
#    4 SID20131176 IDEPT8473            24 graduate
#    5 SID20131177 IDEPT5528            24 graduate
#    6 SID20131184 IDEPT5109            24 graduate
#    7 SID20131189 IDEPT8473            24 graduate
#    8 SID20131191 IDEPT3115            24 graduate
#    9 SID20131208 IDEPT6347            23 graduate
#   10 SID20131220 IDEPT2601             24 graduate
#   # … with 3,990 more rows
## Do not modify this line!
s_admission_new<-s_admission%>%
  mutate(age=round(DOB%--%today()/dyears(1)),
       school_year=fct_collapse(factor(DOA),
                          graduate=c("2013-07-01","2014-07-01"),
                          senior=  "2015-07-01",
                          junior="2016-07-01",
                          sophomore="2017-07-01",
                          freshman="2018-07-01")
       )%>%select(-c(DOA:Department_Choices))

# 4. Create a tibble `s_performance_new` of size 3819 x 4 with columns
#   `Student_ID`, `mean_score`, `min_score` and `max_score`. `mean_score`,
#   `min_score` and `max_score` are aggregated across semesters and papers.
#   To do this, you can use:
#     - `group_by()` to group the data by `Student_ID`.
#     - `summarize()` to calculate the mean, minimum and maximum of `Marks`.
#   To check your result, `s_performance_new` prints to:
#   # A tibble: 3,819 x 4
#     Student_ID  mean_score min_score max_score
#     <chr>          <dbl>    <int>    <int>
#    1 SID20131143      71.7       22      100
#    2 SID20131151      72.6       40      100
#    3 SID20131171      70.9       20       99
#    4 SID20131176      70.9       42      100
#    5 SID20131177      70.3       29      100
#    6 SID20131184      66.3       19       99
#    7 SID20131189      64.3       40       99
#    8 SID20131191      69.8       21      100
#    9 SID20131208      68.9       40      100
#   10 SID20131220      69.1       21       98
#   # … with 3,809 more rows
```

```
## Do not modify this line!
s_performance_new<-s_performance%>%group_by(Student_ID)%>%
  summarize(mean_score=mean(Marks),min_score=min(Marks),max_score=max(Marks))



# 5. Filter the tibble `department` for the Engineering School where
#    `Department_Name` has a match with the string "Engineering" (case insensitive).
#    Store the result into a tibble `department_engie` of size 17 x 2 with columns
#    `Department_ID` and `Department_Name`.
#    To do this, you can use:
#      - `filter()` to filter `Department_Name` for the Engineering School.
#        - `str_detect()` to detect the string `"Engineering"`
#          - `fixed()` to modify the matching pattern so that case differences
#            would be ignored.
#      - `dplyr::select()` to select the desired columns.
#    To check your result, `department_engie` prints to:
#    # A tibble: 17 x 2
#      Department_ID Department_Name
#      <chr>         <chr>
#    1 IDEPT4670     Aerospace Engineering
#    2 IDEPT5528     Biosciences and Bioengineering
#    3 IDEPT3115     Chemical Engineering
#    4 IDEPT4938     Civil Engineering
#    # … with 13 more rows
## Do not modify this line!
department_engie<-department%>%
filter(str_detect(Department_Name,fixed("Engineering"))|str_detect(Department_Name,fix
ed("engineering")))%>%select(Department_ID,Department_Name)
?str_detect()
?fixed
# 6. Join the tibbles `employee_new` and `department_engie` to create a tibble
#    `employee_joined` of size 422 x 6 with columns `Employee_ID`, `Department_ID`,
#    `age`, `seniority`, `seniority_level` and `Department_name`.
#    To do this, you can use `inner_join()` with the key `"Department_ID"`.
#    (Note that inner joins are used in this exercise to preserve integrity. For
#    instance, there is no employee record for three engineering departments.
#    Using some other types of joins will force NA's here.)
#    To check your result, `employee_joined` prints to:
#    # A tibble: 422 x 6
#      Employee_ID Department_ID   age seniority seniority_level Department_Name
#      <chr>       <chr>         <dbl>    <dbl> <fct>           <chr>
#    1 IU196557    IDEPT4938        37       10 junior          Civil Engineering
#    2 IU449901    IDEPT2357        34       10 junior          Energy Science and Engineering
#    3 IU206427    IDEPT4670        48       11 senior          Aerospace Engineering
#    4 IU572796    IDEPT1142        42       12 senior          Centre for Aerospace Systems Design
and Engi…
#    5 IU393717    IDEPT1825        47       10 junior          Mechanical Engineering
#    6 IU826824    IDEPT8313        31       12 senior          Metallurgical Engineering & Materials
Science
#    7 IU917848    IDEPT2357        43       17 senior          Energy Science and Engineering
#    8 IU423398    IDEPT4938        29       13 senior          Civil Engineering
```

```
#    9 IU710285    IDEPT3115       49        9 junior         Chemical Engineering
#   10 IU740065    IDEPT1423       30       20 senior         Computer Science & Engineering
#   # … with 412 more rows
## Do not modify this line!
employee_joined<-inner_join(employee_new,department_engie)


# 7. With department and faculty information joined, we can investigate the
#    faculty make-up of each department. To facilitate comparison, we would also
#    like to know the size of the departments. From `employee_joined`, create
#    a tibble `department_size` of size size 14 x 2 with columns `Department_Name`
#    and `n`, where `n` is the department size in ascending order.
#    To do this, you can use:
#      - `dplyr::count()` to count `Department_Name` (the number of rows for
#        each department is the number of faculty).
#      - `arrange()` to sort the department size (i.e., `n`).
#    To check your result, `department_size` prints to:
#    # A tibble: 14 x 2
#      Department_Name                                  n
#      <chr>                                        <int>
#    1 Centre of Studies in Resources Engineering (CSRE)           23
#    2 Centre for Urban Science and Engineering (C-USE)            24
#    3 Centre for Aerospace Systems Design and Engineering (CASDE)    26
#    4 Centre for Environmental Science and Engineering (CESE)       28
#    # … with 10 more rows
## Do not modify this line!
department_size<-employee_joined%>%dplyr::count(Department_Name)%>%arrange(n)


# 8. We will use `department_size` in the previous question as a helper tibble
#    for sorting the levels of `Department_Name` in `employee_joined`. Update
#    the original `employee_joined` tibble so that `Department_Name` becomes a
#    `factor` whose levels are sorted by department size.
#    To do this, you can use:
#      - `mutate()` to create (update) the column `Department_Name` by
#        - Making `Department_Name` a factor with `factor()`.
#          - Set `levels` of `Department_Name` as `Department_Name` in
#            `department_size` with `pull()`.
#    To check your result, the new `employee_joined` prints to:
#    # A tibble: 422 x 6
#      Employee_ID Department_ID   age seniority seniority_level Department_Name
#      <chr>       <chr>         <dbl>    <dbl> <fct>           <fct>
#    1 IU196557    IDEPT4938        37       10 junior          Civil Engineering
#    2 IU449901    IDEPT2357        34       10 junior          Energy Science and E…
#    3 IU206427    IDEPT4670        48       11 senior          Aerospace Engineering
#    4 IU572796    IDEPT1142        42       12 senior          Centre for Aerospace…
#    5 IU393717    IDEPT1825        47       10 junior          Mechanical Engineeri…
#    6 IU826824    IDEPT8313        31       12 senior          Metallurgical Engine…
#    7 IU917848    IDEPT2357        43       17 senior          Energy Science and E…
#    8 IU423398    IDEPT4938        29       13 senior          Civil Engineering
#    9 IU710285    IDEPT3115        49        9 junior          Chemical Engineering
#   10 IU740065    IDEPT1423        30       20 senior          Computer Science & E…
```

```
#   # … with 412 more rows
## Do not modify this line!
employee_joined<-employee_joined%>%
  mutate(Department_Name=factor(Department_Name,levels =
pull(department_size,Department_Name)))


# 9. One important aspect of faculty make-up is seniority. Using `employee_joined`,
#   draw horizontal stacked bar charts for `Department_Name` with distribution
#   of `seniority_level` for each department.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#     - `geom_bar()` to draw bar charts.
#     - `coord_flip()` to make the bar charts horizontal.
#     - `labs()` to format the labels such that:
#       - `title = "Number of Senior and Junior Professors by Department"`
#       - `subtitle = paste("Most departments have more senior professors.",
#                           "Exceptions are CASDE and Chemical Engineering.",
#                           sep = "\n")`
#       - `x = "Department's Name"`
#       - `fill = "Seniority Level"`
#       - `y = "Count (n)"`
#   Store the plot into a `ggplot` object `g1`.
## Do not modify this line!
g1<-ggplot(employee_joined,mapping =
aes(x=Department_Name))+geom_bar(aes(fill=seniority_level))+coord_flip()+
  labs(title = "Number of Senior and Junior Professors by Department",
       subtitle = paste("Most departments have more senior professors.","Exceptions are
CASDE and Chemical Engineering.",sep = "\n"),
       x = "Department's Name",
       fill = "Seniority Level",
       y = "Count (n)"
       )+theme_light()



# 10. Join the tibbles `s_admission_new`, `s_performance_new` and `department_engie`
#    to create a new tibble `student_joined` of size 1740 x 8 with columns
#    `Student_ID`, `Department_Admission`, `age`, `school_year`, `mean_score`,
#    `min_score` and `max_score` and `Department_Name`.
#    To do this, you can use:
#      - `inner_join()` with the key `"Student_ID"` to join `s_admission_new`
#        with `s_performance_new`.
#      - `inner_join()` with the key `"Department_Admission"` in the left table
#        and `"Department_ID"` in the right table to join `department_engie`.
#    To check your result, `student_joined` prints to:
#    # A tibble: 1,740 x 8
#      Student_ID Department_Admi…   age school_year mean_score min_score
#      <chr>      <chr>          <dbl> <chr>          <dbl>     <int>
#    1 SID201311… IDEPT7783         24 graduate        71.7        22
#    2 SID201311… IDEPT8473         24 graduate        70.9        42
#    3 SID201311… IDEPT5528         24 graduate        70.3        29
```

```
#     4 SID201311… IDEPT8473         24 graduate        64.3       40
#     5 SID201311… IDEPT3115         24 graduate        69.8       21
#     6 SID201312… IDEPT1825         24 graduate        70.0       40
#     7 SID201312… IDEPT5564         24 graduate        66.9       24
#     8 SID201313… IDEPT2054         23 graduate        68.8       40
#     9 SID201313… IDEPT2054         24 graduate        72.5       42
#    10 SID201314… IDEPT3115          24 graduate         70.2       40
#    # … with 1,730 more rows, and 2 more variables: max_score <int>,
#    #   Department_Name <chr>
## Do not modify this line!
student_joined<-inner_join(s_admission_new,s_performance_new,by = "Student_ID")%>%
  inner_join(department_engie,by = c("Department_Admission"="Department_ID"))


# 11. The tibble `student_joined` contains complete information about students
#     at the Engineering School. For initial exploration, draw horizontal
#     boxplots for `mean_score` vs. `school_year`. The boxplots should be sorted
#     by descending medians.
#     To do this, you can use:
#       - `ggplot()` to initialize a ggplot object and specify the variables to plot.
#         - `fct_reorder()` to order the levels of `school_year` by `mean_score`.
#       - `geom_boxplot()` to draw boxplots.
#       - `coord_flip()` to make the boxplots horizontal.
#       - `labs()` to format the labels such that:
#         - `title = "Student Academic Performance by Class Standing at Engineering School"`
#         - `subtitle = "Distribution is similar across years."`
#         - `x = "Class Standing"`
#         - `y = "Mean Paper Score"`
#     Store the plot into a `ggplot` object `g2`.
## Do not modify this line!

g2<-ggplot(student_joined,mapping =
aes(x=fct_reorder(school_year,mean_score),y=mean_score))+geom_boxplot()+coord_flip()+
  labs(title = "Student Academic Performance by Class Standing at Engineering School",
       subtitle = "Distribution is similar across years.",
       x = "Class Standing",
       y = "Mean Paper Score")+theme_light()


# 12. We would like to explore what could affect a student's academic performance
#     from the faculty aspect, particularly in seniority of professors. Create
#     a tibble `employee_senior` from `employee_joined` of size 14 x 2 with
#     columns `Department_ID` and `seniority`, where `seniority` is the median
#     seniority for each department.
#     To do this, you can use:
#       - `group_by()` to group data by `Department_ID`.
#       - `summarize()` to compute an overall median of `seniority`.
#     To check your result, `employee_senior` prints to:
#     # A tibble: 14 x 2
#       Department_ID seniority
#       <chr>           <dbl>
#     1 IDEPT1142        10.5
```

```
#     2 IDEPT1423        14
#     3 IDEPT1825        13
#     4 IDEPT2054        14.5
#    # … with 10 more rows
## Do not modify this line!
employee_senior<-employee_senior<-employee_joined%>%group_by(Department_ID)%>%
  summarize(seniority=median(seniority))


# 13. Join the tibbles `student_joined` and `employee_senior` to create a final
#     `school_year`, `mean_score`, `min_score`, `max_score`, `Department_Name`
#     and `seniority`.
#     To do this, you can use:
#       - `inner_join()` with the key `"Department_Admission"` in the left table
#         and `"Department_ID"` in the right table to join `employee_senior`.
#       - `dplyr::select()` to select the desired columns.
#     To check your result, `engie` prints to:
#     # A tibble: 1,429 x 8
#       Student_ID   age school_year mean_score min_score max_score
#       <chr>     <dbl> <fct>          <dbl>    <int>     <int>
#     1 SID201311…    24 graduate        71.7       22       100
#     2 SID201311…    24 graduate        70.3       29       100
#     3 SID201311…    24 graduate        69.8       21       100
#     4 SID201312…    24 graduate        70.0       40       100
#     5 SID201313…    23 graduate        68.8       40       100
#     6 SID201313…    24 graduate        72.5       42       100
#     7 SID201314…    24 graduate        70.2       40        99
#     8 SID201314…    24 graduate        69.4       23        98
#     9 SID201315…    24 graduate        69.8       41       100
#    10 SID201315…    24 graduate         63.3       20        97
#    # … with 1,419 more rows, and 2 more variables: Department_Name <chr>,
#    #   seniority <dbl>
## Do not modify this line!
engie<-inner_join(student_joined,employee_senior,by =
c("Department_Admission"="Department_ID"))%>%select(-Department_Admission)


# 14. With the final tibble `engie`, we are interested in the relationships
#     between professor seniority and student academic performance.
#     Draw pair plots of the variables `seniority`, `mean_score`, `min_score`
#     and `max_score`.
#     To do this, you can use:
#       - `ggpairs()` to draw pair plots for columns of interests
#       - `labs()` to format the labels such that:
#         - `title = "Correlation between Professor Seniority and Student Academic
Performance"`
#         - `subtitle = "No significant correlation is found."`
#     Store the plot into a `ggplot` object `g3`.
## Do not modify this line!

g3<-ggpairs(engie,columns=c("seniority","mean_score","min_score","max_score"))+
  labs(title = "Correlation between Professor Seniority and Student Academic
```

```
Performance",subtitle = "No significant correlation is found.")+theme_light()
g3
?ggpairs

# HW7: relational + regex + ggplot
#'
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - For graphs with titles, make the format as
# `theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))`.
#'
# 1. Load the packages `tidyverse` and `lubridate`.
#   Use `read_csv()` to read the datasets from data folder:
#     - `movies.csv` into a tibble `movie`.
#     - `ratings.csv` into a tibble `ratings`.
#   To check your solution, `movies` prints to:
#   # A tibble: 27,254 x 3
#   movieId title                        genres
#   <dbl> <chr>                        <chr>
#   1       1 Toy Story (1995)              Adventure|Animation|Children|Comedy|Fanta..
#   2       2 Jumanji (1995)                Adventure|Children|Fantasy
#   3       3 Grumpier Old Men (1995)         Comedy|Romance
#   4       4 Waiting to Exhale (1995)        Comedy|Drama|Romance
#   5       5 Father of the Bride Part II (1995) Comedy
#   6       6 Heat (1995)                   Action|Crime|Thriller
#   7       7 Sabrina (1995)                Comedy|Romance
#   8       8 Tom and Huck (1995)             Adventure|Children
#   9       9 Sudden Death (1995)             Action
#   10      10 GoldenEye (1995)               Action|Adventure|Thriller
#   # … with 27,244 more rows
#   `ratings` prints to:
#   # A tibble: 50,000 x 4
#   userId movieId rating  timestamp
#   <dbl>   <dbl>  <dbl>     <dbl>
#   1  36660     280    4    834049053
#   2  91867    2657    5    971578971
#   3 107259    2566    2    944169497
#   4 129338    4161    2.5 1137405482
#   5 128693    2355    4    984620012
#   6  97984    4052    3.5 1112061639
#   7  97700    3949    4   1274039326
#   8  10443   94839    2   1420951497
#   9  81462    4105    3   1092108684
#   10   6031     593    4    834163850
#   # … with 49,990 more rows
## Do not modify this line!
library(tidyverse)
library(lubridate)
movies<-read_csv('data/movies.csv')
ratings<-read_csv('data/ratings.csv')
```

```
# 2. Turn `timestamp` in the `ratings` dataset into normal format
#    (e.g. year-month-day).
#    To do this, you can use:
#      - `as.POSIXct()` to turn `timestamp` into
#        normal format, and specify the argument `origin = "1970-01-01"`.
#      - `with_tz()`to change the time zone to UTC ,by passing
#        `tzone = "UTC"`.
#    Store the returned dataset into `ratings`.
#    To check your result, `ratings` prints to:
#    # A tibble: 50,000 x 5
#    userId movieId rating  timestamp date
#    <dbl>   <dbl>  <dbl>      <dbl> <dttm>
#    1  36660     280    4    834049053 1996-06-06 08:17:33
#    2  91867    2657    5    971578971 2000-10-15 03:02:51
#    3 107259    2566    2    944169497 1999-12-02 21:18:17
#    4 129338    4161    2.5 1137405482 2006-01-16 09:58:02
#    5 128693    2355    4    984620012 2001-03-15 01:33:32
#    6  97984    4052    3.5 1112061639 2005-03-29 02:00:39
#    7  97700    3949    4   1274039326 2010-05-16 19:48:46
#    8  10443   94839    2   1420951497 2015-01-11 04:44:57
#    9  81462    4105    3   1092108684 2004-08-10 03:31:24
#    10   6031     593    4    834163850 1996-06-07 16:10:50
#    # … with 49,990 more rows
## Do not modify this line!
ratings<-ratings%>%mutate(date=with_tz(as.POSIXct(timestamp,origin="1970-01-01"),tzone
= "UTC"))

?with_tz()
# 3. Create a tibble `average_rating_in_different_years` of dimension 20x2.
#    The procedure should first create a new column called `year_of_rating` that
#    gets the year of the rating and converted into a factor variable. Then we
#    group the dataset by `year_of_rating` and summarize the dataset such that
#    we calculate the average ratings of
#    each year and store averages in a column called `average_rating_of_year`.
#    To do that, you can use:
#      - `mutate()` and `as.factor()` to coerce the year of `date` into a
#        factor variable.
#      - `group_by()` to group by `year_of_rating`.
#      - `summarize` to summarize the mean of ratings of different year and
#        store the value into `average_rating_of_year`.
#    To check your result, `average_rating_in_different_years` prints to:
#    # A tibble: 20 x 2
#    year_of_rating average_rating_of_year
#    <fct>                  <dbl>
#    1 1996                    3.56
#    2 1997                    3.56
#    3 1998                    3.49
#    4 1999                    3.59
#    5 2000                    3.59
#    6 2001                    3.52
#    7 2002                    3.51
```

```
#   8 2003                    3.50
#   9 2004                    3.42
#   10 2005                   3.41
#   # … with 10 more rows
## Do not modify this line!
average_rating_in_different_years<-ratings%>%
mutate(year_of_rating=as.factor(year(date)))%>%
 group_by(year_of_rating)%>%
 summarize(average_rating_of_year=mean(rating))


# 4. Draw a point plot of `average_rating_of_year` vs. `year_of_rating`.
#   Name the title as `"2004 and 2005 have lower average ratings"`,
#   subtitle as `"While 2014 has highest average ratings"`.
#   To do this, you can use:
#    - `geom_point()` to draw a point plot of `average_rating_of_year`
#      vs. `year_of_rating`.
#    - `labs()` to name the title as
#      `"2004 and 2005 have lower average ratings"`,
#      the subtitle as `"While 2014 has highest average ratings"`,
#      the x-axis as `"Year"`,
#      the y-axis as `"Average rating"`.
#   Store the plot into a variable `g1`.
## Do not modify this line!
g1<-ggplot(average_rating_in_different_years,mapping =
aes(x=year_of_rating,y=average_rating_of_year))+
 geom_point()+labs(title="2004 and 2005 have lower average ratings",subtitle = "While
2014 has highest average ratings",
                  x="Year",y="Average rating")+theme_light()+theme(plot.title =
element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))


# 5. Extract the year of movies from `title` column in the `movies` dataset.
#   Store the values into a new column called `year` and convert values into numeric.
#   To do this, you can use:
#    - `mutate()`, `map_chr` and `as.numeric()` to generate a new column called `year`
#      and transform values into numeric.
#      (hint: pay attention to `substr()` to extract the characters that we want.)
#   Store the returned dataset into `movies`.
#'
#   *Optional material*: The `title` column format is relatively clean here. What if
#   we have 4-digit years and 2-digit years mix (e.g. 1995, 96, 02, 2005, etc.)?
#   Then the method above will not work and how should we modify the code or use other
methods?
#   Way 1: Locate the index of `(` and `)` then use `substr` to extract years.
#   Way 2: Use regular expression. For instance, you can combine `gsub()` with
#   `pattern = ".+\\(([0-9]+)\\)"`, `replacement = "\\1"`, and `x = title` in order to
#   extract the year.
#   Here, `gsub()` matches to argument pattern within each element of a character
#   vector, and then replace the designed pattern.
#   The pattern is `".+\\(([0-9]+)\\)"`.
#   `.` means the string starts with a character and `.+` means there are one or more
```

```
characters.
#   `\\(` and `\\)` means there really exist parenthese. `\\` is just the syntax for escaping.
#   `([0-9]+)` means there exists a string of characters that can be converted to numeric.
#   The replacement is `\\1`.
#   `\\1` means we only need to keep the characters that can be converted to numeric.
#   If interested, please go to the following website for more detials:
#   https://stringr.tidyverse.org/articles/regular-expressions.html.
#'
#   To check your result, `movies` prints to:
#   # A tibble: 27,254 x 4
#   movieId title                  genres                      year
#   <dbl> <chr>                  <chr>                       <dbl>
#   1      1 Toy Story (1995)            Adventure|Animation|Children|Comedy|Fa…  1995
#   2      2 Jumanji (1995)              Adventure|Children|Fantasy           1995
#   3      3 Grumpier Old Men (1995)       Comedy|Romance                    1995
#   4      4 Waiting to Exhale (1995)      Comedy|Drama|Romance              1995
#   5      5 Father of the Bride Part II (1… Comedy                          1995
#   6      6 Heat (1995)                 Action|Crime|Thriller               1995
#   7      7 Sabrina (1995)              Comedy|Romance                      1995
#   8      8 Tom and Huck (1995)         Adventure|Children                  1995
#   9      9 Sudden Death (1995)         Action                              1995
#   10     10 GoldenEye (1995)           Action|Adventure|Thriller           1995
#   # … with 27,244 more rows
## Do not modify this line!
movies<-movies%>%mutate(year = as.numeric(map_chr(title, ~substr(.x, nchar(.x)-4,
nchar(.x)-1))))

# 6. Convert the `genres` column in `movies` dataset into characters.
#   Then separate the genres of a same movie such that the same movie is split
#   into several entries with different kinds of genres.
#   To do this, you can use:
#    - `mutate()` and `as.character` to convert `genres` into characters.
#    - `separate_rows` to split up the `genres`.
#   Store the returned dataset into `movies`.
#   To check your result, `movies` prints to:
#   # A tibble: 54,374 x 4
#   movieId title                  genres    year
#   <dbl> <chr>                  <chr>     <dbl>
#   1      1 Toy Story (1995)      Adventure  1995
#   2      1 Toy Story (1995)      Animation  1995
#   3      1 Toy Story (1995)      Children   1995
#   4      1 Toy Story (1995)      Comedy     1995
#   5      1 Toy Story (1995)      Fantasy    1995
#   6      2 Jumanji (1995)        Adventure  1995
#   7      2 Jumanji (1995)        Children   1995
#   8      2 Jumanji (1995)        Fantasy    1995
#   9      3 Grumpier Old Men (1995) Comedy     1995
#   10     3 Grumpier Old Men (1995) Romance    1995
#   # … with 54,364 more rows
## Do not modify this line!
movies<-movies%>%mutate(genres = as.character(genres))%>%separate_rows(genres,sep =
"\\|")
```

```
# 7. Join the `movies` and `ratings` together by `movieId`. Remove `movieId`, `userId`
#    and `timestamp` columns after join.
#    To do this, you can use:
#      - `inner_join()` to join two dataset.
#      - `select()` to remove specified columns.
#    Store returned dataset to `ratings_of_movies`.
#    To check your result, `ratings_of_movies` prints to:
#    # A tibble: 132,725 x 5
#    title          genres     year rating date
#    <chr>          <chr>     <dbl>  <dbl> <dttm>
#    1 Toy Story (1995) Adventure  1995    3.5 2005-03-17 20:22:12
#    2 Toy Story (1995) Adventure  1995    3   1999-07-06 18:41:32
#    3 Toy Story (1995) Adventure  1995    3   2009-08-18 18:31:29
#    4 Toy Story (1995) Adventure  1995    4.5 2009-08-20 06:36:02
#    5 Toy Story (1995) Adventure  1995    5   1997-01-26 14:23:23
#    6 Toy Story (1995) Adventure  1995    5   2001-01-18 17:22:03
#    7 Toy Story (1995) Adventure  1995    3.5 2004-03-06 07:59:24
#    8 Toy Story (1995) Adventure  1995    4   1996-10-24 22:12:42
#    9 Toy Story (1995) Adventure  1995    4   1997-11-20 12:59:43
#    10 Toy Story (1995) Adventure  1995    4   2005-01-27 20:49:25
#    # … with 132,715 more rows
## Do not modify this line!
ratings_of_movies<-inner_join(movies,ratings,by = "movieId")%>%select(-movieId,-userId,-
timestamp)


# 8. Draw a horizontal boxplot of `rating` vs. `genres`.
#    To do this, you can use:
#      - `geom_boxplot()` to draw a boxplot of `rating` vs. `genres`.
#      - `labs()` to name the title as:
#        `"Most movie genres have median ratings from 3.5 to 4"`,
#        name the x-axis as `"Genres"`,
#        name the y-axis as `"Ratings"`.
#      - `coord_flip()` to flip x and y.
#    Store the plot into a variable `g2`.
## Do not modify this line!
g2<-ggplot(ratings_of_movies,mapping = aes(x=genres,y=rating))+geom_boxplot()+
  labs(title="Most movie genres have median ratings from 3.5 to 4",
       x="Genres",
       y="Ratings")+coord_flip()+theme_light()+theme(plot.title = element_text(hjust = 0.5),
plot.subtitle = element_text(hjust = 0.5))


# 9. Group by the `genres` columns in `ratings_of_movies`. Then summrize the mean of
#    `rating` and number of `rating` of different `genres`. Store average of rating
#    values into a column `average_rating`, number of `rating`
#    into a column `number_of_ratings`.
#    Finally convert the `genres` into a factor variable.
#    To do this, you can use:
```

```
#     - `group_by` to group by `genres`.
#     - `summarize` to calculate the mean of
#       `rating` and number of `rating` of different `genres`. Store average of
#       rating values into a column `average_rating`, number of `rating`
#       into a column `number_of_ratings`.
#     - `mutate()` and `as.factor` to convert `genres` into a factor variable.
#   Store the returned dataset into `average_rating_of_genres`.
#   To check your result, `average_rating_of_genres` prints to:
#   # A tibble: 20 x 3
#   genres          average_rating number_of_ratings
#   <fct>                    <dbl>          <int>
#   1 (no genres listed)         4.5              1
#   2 Action                  3.43          13970
#   3 Adventure                3.49          10875
#   4 Animation                3.59           2807
#   5 Children                 3.38           4133
#   6 Comedy                   3.42          18701
#   7 Crime                   3.68           8265
#   8 Documentary               3.74            588
#   9 Drama                   3.67          22331
#   10 Fantasy                 3.49           5252
#   # … with 10 more rows
## Do not modify this line!
average_rating_of_genres<-ratings_of_movies%>%group_by(genres)%>%
summarize(average_rating=mean(rating),number_of_ratings=n())%>%
mutate(genres=as.factor(genres))


# 10.Draw a point plot of `average_rating` vs. `number_of_ratings`, colored by `genres`.
#   Draw a smooth curve that passes through points using `loess` method.
#   Name the title as
#   `"When number of ratings is over 5000, the average ratings start to be constant around
3.5"`.
#   To do this, you can use:
#     - `geom_point()` to draw a point plot of `average_rating` vs. `number_of_ratings`,
#       color by `genres`.
#     - `geom_smooth()` to draw a smooth curve using method as `loess`.
#     - `labs()` to name the title as
#       `"When number of ratings is over 5000, the average ratings start to be constant
around 3.5"`.
#       the x-axis as `"Number of ratings"`,
#       the y-axis as `"Average rating"`,
#       the color legend as `"Genres"`.
#   Store the plot into a variable `g3`.
## Do not modify this line!
g3<-
ggplot(data=average_rating_of_genres,mapping=aes(x=number_of_ratings,y=average_ratin
g))+geom_point(mapping=aes(color = genres))+geom_smooth(method =  "loess")+
  labs(title = "When number of ratings is over 5000, the average ratings start to be constant
around 3.5",x="Number of ratings",y="Average rating",color="Genres")+
  theme_light()+theme(plot.title = element_text(hjust = 0.5), plot.subtitle =
element_text(hjust = 0.5))
```

```
# 11.Calculate the interval of time passed between the `date` corresponding the
#    rating and the release `year` from `ratings_of_movies` dataset,
#    and store this interval into a column called `interval`. You can assume
#    that a film released in a given year was actually released on January 1st.
#    Then, convert the units of `interval` into years and store its
#    values into a new column called `years_passed`.
#    Finally, filter out rows with `years_passed` less or equal to 60.
#    To do this, you can use:
#     - `mutate()` to create two required columns.
#       (hint 1: pay attention to `make_datetime()` to convert the release year
#       into a date and remembers that dates/datetimes can
#       be substracted to create intervals)
#       (hint 2: intervals can be divided by durations, and the `dyears()`
#       function can help your compute the number of years that have passed)
#     - `filter` to filter out rows with `years_passed` less or equal to 60.
#       Store the returned dataset to `ratings_over_time`.
#    To check your result, `ratings_over_time` prints to:
#    # A tibble: 130,087 x 7
#    title          genres     year rating date                interval      years_passed
#    <chr>          <chr>     <dbl>  <dbl> <dttm>              <drtn>             <dbl>
#    1 Toy Story (199… Adventure  1995    3.5 2005-03-17 20:22:12 3728.8488 da…       10.2
#    2 Toy Story (199… Adventure  1995    3   1999-07-06 18:41:32 1647.7788 da…       4.51
#    3 Toy Story (199… Adventure  1995    3   2009-08-18 18:31:29 5343.7719 da…       14.6
#    4 Toy Story (199… Adventure  1995    4.5 2009-08-20 06:36:02 5345.2750 da…       14.6
#    5 Toy Story (199… Adventure  1995    5   1997-01-26 14:23:23  756.5996 da…       2.07
#    6 Toy Story (199… Adventure  1995    5   2001-01-18 17:22:03 2209.7236 da…       6.05
#    7 Toy Story (199… Adventure  1995    3.5 2004-03-06 07:59:24 3352.3329 da…       9.18
#    8 Toy Story (199… Adventure  1995    4   1996-10-24 22:12:42  662.9255 da…       1.82
#    9 Toy Story (199… Adventure  1995    4   1997-11-20 12:59:43 1054.5415 da…       2.89
#    10 Toy Story (199… Adventure  1995    4   2005-01-27 20:49:25 3679.8677 da…       10.1
#    # … with 130,077 more rows
## Do not modify this line!
ratings_over_time<-ratings_of_movies%>%mutate(interval=date-
make_datetime(year=year),
                    years_passed = interval/dyears())%>%filter(years_passed<=60)


# 12.Draw a "straight" line plot of `rating` vs. `years_passed`, colored by `genres`.
#    Name the title as `"The average rating usually increases with time"`,
#    subtitle as `"One exception is animation movies"`,
#    x-axis as `"Years between release and rating"`,
#    y-axis as `"Average rating"`,
#    legend as `"Genres"`.
#    To do this, you can use:
#     - `ggplot()` to setup your plot `rating` vs. `years_passed`, colored by `genres`.
#     - `geom_smooth()` to draw a "straight" line plot using method as `"lm"`.
#     - `labs()` to name the title `"The average rating usually increases with time"`,
#       subtitle as `"One exception is animation movies"`,
#       x-axis as `"Years between release and rating"`,
```

```
#       y-axis as `"Average rating"`,
#       color legend as `"Genres"`.
#    Store the plot into a variable `g4`.
#
## Do not modify this line!
g4<-ggplot(ratings_over_time,aes(x=years_passed,y=rating,color=genres))+
  geom_smooth(method = "lm",se=FALSE)+labs(title = "The average rating usually increases
with time",
                         subtitle="One exception is animation movies",
                         x="Years between release and rating",
                         y="Average rating",
                         color="Genres")+theme_light()+theme(plot.title = element_text(hjust =
0.5), plot.subtitle = element_text(hjust = 0.5))


# HW7: NYC stock analysis
#'
# In this exercise, you will conduct complete data analysis on NYC stock price.
#'
# Dataset consists of following files:
# `prices.csv`: raw, as-is daily prices. Most of data spans from 2010 to the end 2016,
#           for companies new on stock market date range is shorter.
#           There have been approximmately 140 stock splits in that time,
#           this set doesn't account for that.
# `securities.csv`: general description of each company with division on sectors
# `fundamentals.csv`: metrics extracted from annual SEC 10K fillings (2012-2016),
#            should be enough to derive most of popular fundamental indicators.
#'
# 1. Do the following:
#     - load the `readr`, `dplyr` and `tidyr` package
#     - load `/course/data/prices.csv` using `read_csv()` and save it to `raw`.
#     - load `data/securities.csv` using `read_csv()` and save it to `sectors`.
#     - load `data/fundamentals.csv` using `read_csv()` and save it to `fund`.
#    `raw` should look like:
#    # A tibble: 851,264 x 7
#    date                symbol  open close   low  high  volume
#    <dttm>              <chr>  <dbl> <dbl> <dbl> <dbl>   <dbl>
#    1 2016-01-05 00:00:00 WLTW    123.  126.  122.  126. 2163600
#    2 2016-01-06 00:00:00 WLTW    125.  120.  120.  126. 2386400
#    3 2016-01-07 00:00:00 WLTW    116.  115.  115.  120. 2489500
#    4 2016-01-08 00:00:00 WLTW    115.  117.  114.  117. 2006300
#    5 2016-01-11 00:00:00 WLTW    117.  115.  114.  117. 1408600
#    6 2016-01-12 00:00:00 WLTW    116.  116.  114.  116. 1098000
#    7 2016-01-13 00:00:00 WLTW    116.  113.  113.  117.  949600
#    8 2016-01-14 00:00:00 WLTW    114.  114.  110.  115.  785300
#    9 2016-01-15 00:00:00 WLTW    113.  113.  112.  115. 1093700
#    10 2016-01-19 00:00:00 WLTW    114.  110.  110.  116. 1523500
#    # … with 851,254 more rows
#    `securities` should look like:
#    # A tibble: 505 x 8
#    `Ticker symbol` Security `SEC filings` `GICS Sector`
#    <chr>           <chr>    <chr>         <chr>
```

```
#   1 MMM          3M Comp… reports      Industrials
#   2 ABT          Abbott … reports      Health Care
#   3 ABBV         AbbVie   reports      Health Care
#   4 ACN          Accentu… reports      Information …
#   5 ATVI         Activis… reports      Information …
#   6 AYI          Acuity … reports      Industrials
#   7 ADBE         Adobe S… reports      Information …
#   8 AAP          Advance… reports      Consumer Dis…
#   9 AES          AES Corp reports      Utilities
#   10 AET         Aetna I… reports      Health Care
#   # … with 495 more rows, and 4 more variables: `GICS Sub
#   #   Industry` <chr>, `Address of Headquarters` <chr>, `Date first
#   #   added` <date>, CIK <chr>
#   `fund` should print to:
#   # A tibble: 1,781 x 79
#   X1 `Ticker Symbol` `Period Ending` `Accounts Payab…
#   <dbl> <chr>         <date>                 <dbl>
#   1     0 AAL          2012-12-31          3068000000
#   2     1 AAL          2013-12-31          4975000000
#   3     2 AAL          2014-12-31          4668000000
#   4     3 AAL          2015-12-31          5102000000
#   5     4 AAP          2012-12-29          2409453000
#   6     5 AAP          2013-12-28          2609239000
#   7     6 AAP          2015-01-03          3616038000
#   8     7 AAP          2016-01-02          3757085000
#   9     8 AAPL         2013-09-28          36223000000
#   10    9 AAPL         2014-09-27          48649000000
#   # … with 1,771 more rows, and 75 more variables: `Accounts
#   #   Receivable` <dbl>, `Add'l income/expense items` <dbl>, `After Tax
#   #   ROE` <dbl>, etc...
## Do not modify this line!
library(readr)
library(dplyr)
library(tidyr)
raw<-read_csv("/course/data/prices.csv")
securities<-read_csv("data/securities.csv")
fund<-read_csv("data/fundamentals.csv")

# 2. Load the `stringr` and `lubridate` packages.
#   Currently, the date in `raw` are in `UTC` time. We want to convert them
#   to New York time zone.
#   To do so, you can:
#     - use `mutate()` to convert the date column
#     - use `force_tz()` to format the date, with argument
#     ` tz` set to `America/New_York`.
#  Save the generated tibble into `raw_time`.
#  `raw_time$date` should be in this format:
#  [1] "2016-01-05 EST" "2016-01-06 EST" "2016-01-07 EST" "2016-01-08 EST"..
#  [6] "2016-01-12 EST" "2016-01-13 EST" "2016-01-14 EST" "2016-01-15 EST"..
## Do not modify this line!
library(stringr)
library(lubridate)
```

```
raw_time<-raw%>%mutate(date=force_tz(date,tz="America/New_York"))


# 3. Load the package `forcats`.
#   In `securities`, keep the companies belonging to the top 6 sectors
#   (by frequency of occurence), as well as those whose `GICS Sub Industry`
#   falls into `"Gold"` or `"Real Estate`" (i.e., `GICS Sub Industry` contains
#   either `"Gold"` or `"REITs"`).
#   You need to do it in three steps:
#    - First, use `mutate()` and `factor()` to convert the `GICS Sector`
#      variable of `securities` from character to a factor. Its levels should
#      be the `unique()` values of `GICS Sector`.
#    - Second, create a tibble named `securities_sectored` that contain
#      only the companies that do not belong to those that you want (see below).
#      Note that `securities_sectored` should contain an additional column
#      `GICS Sector truncated` that contains the top 6 factors in `GICS Sector`
#      and all the others lumped into an additional level `"Other"`.
#    - Third, use `anti_join()` on `securities` and `securities_sectored` to
#      create `securities_selected`, which contains only the rows that actually
#      meet the requirements above by deleting the rows from `securities`
#      that are in `securities_sectored`.
#   To achieve the second step, you can use:
#     - `mutate()` along with `fct_infreq()` and `fct_lump()` to
#       reorder the sectors by frequency of occurence and lump
#       all except the top 6 into a single level `"Other"`.
#     - `filter()` to select the sectors that do not belong to the
#       top 6 (i.e., the ones with the level `"Other"`).
#     - `filter()`  along with `str_detect()` to additionally filter out
#        the observations whose `GICS Sub Industry` contains neither
#        `"Gold"` nor `"REITs"`. In the pattern, you can use a
#        regular expression with or (represented by the alternation
#        symbol `"|"`) to do that.
#   To help you, `securities` is as in part 1, except that the `GICS Sector`
#   column is a factor whose levels print to
#   `Levels: Industrials ... Telecommunications Services`.
#   `securities_sectored` should print to:
#   # A tibble: 94 x 9
#   `Ticker symbol` Security `SEC filings` `GICS Sector`
#   <chr>          <chr>    <chr>       <fct>
#   1 AES           AES Corp reports      Utilities
#   2 APD           Air Pro… reports      Materials
#   3 ALB           Albemar… reports      Materials
#   4 LNT           Alliant… reports      Utilities
#   5 AEE           Ameren … reports      Utilities
#   6 AEP           America… reports      Utilities
#   7 AWK           America… reports      Utilities
#   8 APC           Anadark… reports      Energy
#   9 APA           Apache … reports      Energy
#   10 T            AT&T Inc reports      Telecommunic…
#   # … with 84 more rows, and 5 more variables: `GICS Sub
#   #   Industry` <chr>, `Address of Headquarters` <chr>, `Date first
#   #   added` <date>, CIK <chr>, `GICS Sector truncated` <fct>
```

```
#   `securities_selected` should print to:
#   # A tibble: 411 x 8
#   `Ticker symbol` Security `SEC filings` `GICS Sector`
#   <chr>         <chr>    <chr>       <fct>
#   1 MMM           3M Comp… reports      Industrials
#   2 ABT           Abbott … reports      Health Care
#   3 ABBV          AbbVie   reports      Health Care
#   4 ACN           Accentu… reports      Information …
#   5 ATVI          Activis… reports      Information …
#   6 AYI           Acuity … reports      Industrials
#   7 ADBE          Adobe S… reports      Information …
#   8 AAP           Advance… reports      Consumer Dis…
#   9 AET           Aetna I… reports      Health Care
#   10 AMG          Affilia… reports      Financials
#   # … with 401 more rows, and 4 more variables: `GICS Sub
#   #   Industry` <chr>, `Address of Headquarters` <chr>, `Date first
#   #   added` <date>, CIK <chr>
## Do not modify this line!
library(forcats)
securities<-securities%>%mutate(`GICS Sector`=factor(`GICS Sector`,levels = unique(`GICS
Sector`)))
securities_sectored<-securities%>%
  mutate(`GICS Sector truncated`=fct_lump(fct_infreq(`GICS Sector`),n=6))%>%
  filter(`GICS Sector truncated`=="Other")%>%
  filter(!str_detect(`GICS Sub Industry`,"Gold")&!str_detect(`GICS Sub Industry`,"REITs"))
securities_selected<-anti_join(securities,securities_sectored)

#   4. Convert the column name of `fund` from `Ticker Symbol`
#   to `Ticker symbol`. (This makes sure there is consistency between
#   column names of the different tables!).
#   Then create new column `Period Ending Year` to extract the year from
#   `Period Ending`. Then Drop NA values of `fund`. Select columns `Ticker symbol`,
#   `Period Ending Year` and `Gross Margin`.
#   Save the new tibble as `fund_time`.
#   To do that, you can use:
#     - `dplyr::rename()` to convert the column name.
#     - `mutate()` to create column `Period Ending Year`, inside `mutate()`:
#       - use `str_replace_all()` to first convert "/" to "-".
#       - `mdy()` with `tz` set to `"America/New_York"` to convert the date
#         to EDT time zone.
#       - `year()` to extract the year.
#       (You can use pipe %>% on `Period Ending` inside `mutate()`)
#     - `drop_na()` to drop the rows that contain `NA` values.
#     - `dplyr::select()` to select the intested columns.
#   `fund_time` should look like:
#   # A tibble: 1,299 x 3
#   `Ticker symbol` `Period Ending Year` `Gross Margin`
#   <chr>                  <dbl>        <dbl>
#   1 AAL                    2012           58
#   2 AAL                    2013           59
#   3 AAL                    2014           63
#   4 AAL                    2015           73
```

```
#   5 AAP                    2012          50
#   6 AAP                    2013          50
#   7 AAP                    2015          45
#   8 AAP                    2016          45
#   9 AAPL                    2013          38
#   10 AAPL                     2014          39
#   # … with 1,289 more rows
## Do not modify this line!
fund_time<-fund%>%dplyr::rename(`Ticker symbol`= `Ticker Symbol`)%>%
  mutate(`Period Ending Year`=year(mdy(str_replace_all(`Period
Ending`,"/","-"),tz="America/New_York")))%>%drop_na()%>%
  select(`Ticker symbol`, `Period Ending Year`, `Gross Margin`)


# 5. Select the following columns from `securities_selected`:
#      `Ticker symbol`, `Security`, `GICS Sector`
#   Join the two tibbles `securities_selected` and `fund_time` by `Ticker symbol`.
#   Drop the `fund_time` rows with `NA` if the corresponding `Ticker symbol`
#   in `securities_selected` is not in `fund_time`.
#   To do that, you can use:
#     - `dplyr::select()` to select correponding columns.
#     - `inner_join()` to automatically drop rows in a tibble when not matched
#       with other tibble. Set the argument `by` to `Ticker symbol`.
#   Save the concatenated tibble to `securities_fund`.
#   `securities_fund` should print to:
#   # A tibble: 988 x 5
#   `Ticker symbol` Security `GICS Sector` `Period Ending …
#   <chr>         <chr>    <fct>                <dbl>
#   1 MMM           3M Comp… Industrials           2013
#   2 MMM           3M Comp… Industrials           2014
#   3 MMM           3M Comp… Industrials           2015
#   4 ABT          Abbott … Health Care           2012
#   5 ABT          Abbott … Health Care           2013
#   6 ABT          Abbott … Health Care           2014
#   7 ABT          Abbott … Health Care           2015
#   8 ABBV          AbbVie   Health Care           2013
#   9 ABBV          AbbVie   Health Care           2014
#   10 ABBV          AbbVie   Health Care            2015
#   # … with 978 more rows, and 1 more variable: `Gross Margin` <dbl>
## Do not modify this line!
securities_fund<-securities_selected%>%select(`Ticker symbol`,`Security`,`GICS Sector`)%>%
inner_join(fund_time,by = 'Ticker symbol')


# 6. Load the library `ggplot2`.
#   Generate histograms of `Gross Margin` in different sectors in different periods
#   and assign the plot to variable `gross_margin`.
#   To do so, you can:
#     - create the plot by calling `ggplot()` on `securities_fund`,
#       with `mapping = aes()`, in which argument `fill` should set to `Period
#       Ending Year` and `x` should set to `Gross Margin`.
#     - adding `geom_histogram()` with `binwidth` set to `10`, `color` set to
```

```
#       `black` and `fill` set to `orange`.
#     - then facet on `GICS Sector` using `facet_wrap()`.
#     - `labs()` to format the labels such that:
#         - `title = "Gross margin distributed differently in different sectors"`
#         - `x = "Gross Margin (%)"`
#         - `y = "Count (n)"`
#     - then add the light theme using `theme_light()`.
## Do not modify this line!
library(ggplot2)
gross_margin<-ggplot(securities_fund,mapping = aes(x=`Gross Margin`,fill=`Period Ending
Year`))+geom_histogram(binwidth = 10,color="black",fill="orange")+
  facet_wrap(~`GICS Sector`)+labs(title = "Gross margin distributed differently in different
sectors",
                          x = "Gross Margin (%)",
                          y = "Count (n)")+theme_light()



# 7. Join the two tibbles `raw_time` with `securities_fund` to get each
#    company's stock price trend in each year with its corresponding
#    `Gross Margin` in that year.
#    (Note: the column you want to join the tables by is `Ticker symbol`,
#    make sure they have the exact same name before joining).
#    To do that, you can do the following:
#      - first convert the column name `symbol` of `raw_time` to `Ticker symbol`.
#        (Hint: use `dplyr::rename()`).
#      - use `mutate()`, and `year()` to extract the year of the price happening.
#      - use `dplyr::select()` to select only the following columns:
#          `Ticker symbol`, `close`, `open`, `date`,`year`
#      - use `left_join` with argument `by` set to `Ticker symbol`.
#      - use `filter()` to delete the rows where the `Perior Ending Year` is not
#        corresponding to the date of the price.
#    Save the generated tibble into `full_stock`.
#    `full_stock` should print to:
#    # A tibble: 246,615 x 9
#    `Ticker symbol`  close   open date                year Security
#    <chr>           <dbl>  <dbl> <dttm>              <dbl> <chr>
#    1 AAL             5.12   5.2  2012-01-03 00:00:00  2012 America…
#    2 AAP            69.1   71.1  2012-01-03 00:00:00  2012 Advance…
#    3 ABT            56.7   56.6  2012-01-03 00:00:00  2012 Abbott …
#    4 ADS           103.   103.   2012-01-03 00:00:00  2012 Allianc…
#    5 AKAM            32.9   33.0  2012-01-03 00:00:00  2012 Akamai …
#    6 ALK            73.9   76.4  2012-01-03 00:00:00  2012 Alaska …
#    7 AME             42.2   43.3  2012-01-03 00:00:00  2012 AMETEK …
#    8 AMT             58.8   60.5  2012-01-03 00:00:00  2012 America…
#    9 APH             46.0   46.5  2012-01-03 00:00:00  2012 Ampheno…
#    10 ARNC             9.23   8.94 2012-01-03 00:00:00  2012 Arconic…
#    # … with 246,605 more rows, and 3 more variables: `GICS
#    #   Sector` <fct>, `Period Ending Year` <dbl>, `Gross Margin` <dbl>
## Do not modify this line!
full_stock<-raw_time%>%dplyr::rename(`Ticker symbol`=symbol)%>%
  mutate(year=year(date))%>%dplyr::select(`Ticker symbol`, `close`, `open`, `date`,
```

```
`year`)%>%
  left_join(securities_fund,by = "Ticker symbol")%>%filter(`Period Ending Year`==year)


# 8. Generate the stock close price trend plot in 2010~2016 of the
#    following company:
#      "Aetna Inc", "Amazon.com Inc", "Facebook", "Whole Foods Market",
#      "FedEx Corporation", "Boeing Company", "The Walt Disney Company".
#    To do that, please do data manipulations first and create
#    a tibble `filtered_company`. You can use:
#      - `filter()` and `%in%` to filter the selected companies.
#    `filtered_company` should print to:
#    # A tibble: 5,292 x 9
#    `Ticker symbol` close  open date                year Security
#    <chr>          <dbl> <dbl> <dttm>              <dbl> <chr>
#    1 AMZN           257.  256.  2013-01-02 00:00:00  2013 Amazon....
#    2 BA            77.1  76.6 2013-01-02 00:00:00  2013 Boeing …
#    3 DIS           51.1  50.8 2013-01-02 00:00:00  2013 The Wal…
#    4 FB            28    27.4 2013-01-02 00:00:00  2013 Facebook
#    5 FDX           94.2  93.5 2013-01-02 00:00:00  2013 FedEx C…
#    6 WFM            92.0  93.1 2013-01-02 00:00:00  2013 Whole F…
#    7 AMZN           258.  257.  2013-01-03 00:00:00  2013 Amazon....
#    8 BA            77.5  77.0 2013-01-03 00:00:00  2013 Boeing …
#    9 DIS           51.2  51.0 2013-01-03 00:00:00  2013 The Wal…
#    10 FB            27.8  27.9 2013-01-03 00:00:00  2013 Facebook
#    # … with 5,282 more rows, and 3 more variables: `GICS Sector` <fct>,
#    #   `Period Ending Year` <dbl>, `Gross Margin` <dbl>
## Do not modify this line!
filtered_company<-full_stock%>%filter(Security%in%c("Aetna Inc", "Amazon.com Inc",
"Facebook", "Whole Foods Market","FedEx Corporation", "Boeing Company", "The Walt
Disney Company"))


# 9. Generate the stock close price trend plot in 2016 of the following company:
#    "Aetna Inc", "Amazon.com Inc", "Facebook", "Extra Space Storage",
#    "FedEx Corporation", "JPMorgan Chase & Co.", "Oracle Corp.".
#    To do that, use:
#      - `ggplot()` on `filtered_company`, with `mapping = aes()`,
#        in which argument `y` should set to `close` and `x` should set to `date`.
#      - `geom_line(aes())` in which `color` is set to `Security`.
#      - `labs()` to format the labels such that:
#          - `title = "Six company daily stock close price from 2010 ~ 2016"`
#          - `x = "date"`
#          - `y = "Daily close price (USD)"`
#      - use `theme_light()` to set the theme.
## Do not modify this line!
trend<-ggplot(filtered_company,mapping =
aes(x=date,y=close))+geom_line(aes(color=Security))+
  labs(title = "Six company daily stock close price from 2010 ~ 2016",x = "date",
       y = "Daily close price (USD)")+theme_light()

# 10. Caculate the annual "Rate of Return" (RoR) on the securities in `full_stock`.
```

```
#    "Rate of Return" is defined as the net gain or loss on an investment
#     over a specified time period, calculated as a percentage of
#     the investment's initial cost.
#     (Namely, RoR = (current value - initial value) / initial value)
#     To calculate this index on securities in `full_stock`, you can:
#       - group the stock prices by `Period Ending Year` and `Ticker symbol`
#         using `group_by()`
#       - select the record of start of the year and end of the year
#         by using `filter()` to select `date` equal to `min(date)` or
#         `max(date)`
#       - `mutate()` the `date` to `"open"` if it is equal to `min(date)`,
#         otherwise `"close"`. (use `ifelse()` inside `mutate()`)
#       - use `pivot_longer()` to extract `"open"` and `"close"` from `date`.
#         To do that, inside `pivot_longer()`, set `c("open", "close")`
#         as first argument, and then `names_to` as `"status"`, `values_to`
#         as `"price"`. This will add two columns recording the opening
#         and closing price for each row.
#       - use `filter()` to select the right `status` for each row by
#         condition `date == status`.
#       - use`summarize()` to calculate the `RoR` for each stock in
#         each year. Inside `summarize()`, use `diff()` to calculate the
#         price difference of open price and close price and divide the
#         difference by `price[1]` which represents the open price.
#       - use `left_join()` to add the annual `RoR` to `securities_fund`
#         by joining with `securities_fund` on `Ticker symbol` and
#         `Period Ending Year`.
#         (Set `by` to `c("Ticker symbol", "Period Ending Year")`.)
#       - use `droplevels()` drop the unselected sectors.
#    Save the generated tibble into `return_stock`.
#    `return_stock` should print to:
#    # A tibble: 980 x 6
#    # Groups:   Period Ending Year [5]
#    `Period Ending … `Ticker symbol`  Return Security `GICS Sector`
#    <dbl> <chr>           <dbl> <chr>    <fct>
#    1          2012 AAL            1.60   America… Industrials
#    2          2012 AAP            0.0170 Advance… Consumer Dis…
#    3          2012 ABT            0.158  Abbott … Health Care
#    4          2012 ADS            0.406  Allianc… Information …
#    5          2012 AKAM           0.241  Akamai … Information …
#    6          2012 ALK           -0.436  Alaska … Industrials
#    7          2012 AME           -0.133  AMETEK … Industrials
#    8          2012 AMT            0.278  America… Real Estate
#    9          2012 APH            0.390  Ampheno… Information …
#    10         2012 ARNC          -0.0291 Arconic… Industrials
#    # … with 970 more rows, and 1 more variable: `Gross Margin` <dbl>
## Do not modify this line!
return_stock<-full_stock%>%group_by(`Period Ending Year`,`Ticker symbol`)%>%
filter(date==min(date)|date==max(date))%>%
  mutate(date=ifelse(date==min(date),"open","close"))%>%
  pivot_longer(c("open", "close"),names_to = "status",values_to = "price")%>%
  filter(date == status)%>%summarize(Return=diff(price)/price[1])%>%
left_join(securities_fund,by = c("Ticker symbol", "Period Ending Year"))%>%
```

```
 droplevels()



# 11. Calculate the mean, 0.25 quantile and 0.75 quantile of `Return` for
#    each `GICS Sector`.
#    To do that, use:
#       - `group_by()` to group the stocks by `GICS Sector`.
#       - `summarize()` to calculate `mean_return` using `mean()`,
#         25% quantile `q1` using `quantile()` with `probs` set to `0.25`,
#         75% quantile `q2` using `quantile()` with `probs` set to `0.75`.
#       - `mutate()` to reorder the factor `GICS Sector` using `fct_reorder()`
#         according to `mean_return`.
#    Save the generated tibble into `summary_stock`.
#    The first four lines of `summary_stock` should print to:
#    # A tibble: 8 x 4
#    `GICS Sector`        mean_return       q1     q2
#    <fct>                <dbl>     <dbl>  <dbl>
#    1 Industrials             0.153  -0.0291     0.315
#    2 Health Care              0.192   0.0325     0.323
#    3 Information Technology     0.196  -0.00665    0.350
#    4 Consumer Discretionary     0.121  -0.0513     0.294
## Do not modify this line!
summary_stock<-return_stock%>%group_by(`GICS Sector`)%>%
  summarize(mean_return=mean(Return),q1=quantile(Return,probs = 0.25),q2
=quantile(Return,probs = 0.75))%>%
  mutate(`GICS Sector`=fct_reorder(`GICS Sector`,mean_return))



# HW7: email
#'
# In this exercise, you will perform data analysis with emails within 184 people
# from year 1998 to 2001.
# 1. Let's first read in the required datasets.
#    Load the `readr` package.
#     - Use `read_csv()` to load the `people.csv` data set from `data` folder
#       and assign it to a tibble `people`. This data set contains information
#       about the 184 people who sent emails between each other.
#     - Use `read_csv()` to load the `email.csv` data set from `data` folder
#       and assign it to a tibble `email`. This data set contains information about
#       each email sent: time, sender and receiver.
## Do not modify this line!
library(readr)
people<-read_csv("data/people.csv")
email<-read_csv("data/email.csv")



# 2. `onset` variable in `email` is the time when the email is sent, but we can
#    see that it is shown in a weird way. It turns out that it represents how
#    many seconds from the start time. Let's convert it to normal
#    time stamps.
#    Note: `1998-01-01` is encoded as 883612800 in `onset`.
```

```
#    - Load the `dplyr` package.
#    - Load the `lubridate` package
#    - Use `mutate()` to create a new variable `time` in `email` dataset.
#   We can obtain `time` by following the next three steps:
#    - Substract the `onset` by 883612800 to get the seconds difference with
#      `1998-01-01`.
#    - Use `as.POSIXct()` to change the seconds difference into normal timestamp.
#      Specify the argument `origin = "1998-01-01"`.
#    - Use `with_tz()`to change the time zone to UTC ,by passing
#      `tzone = "UTC"`.
#   Save your generated tibble into `email_w_time` whose first few rows should
#   look like:
#   A tibble: 38,131 x 9
#   onset terminus  tail  head onset.censored terminus.censor… duration edge.id
#   <dbl>   <dbl> <dbl> <dbl> <lgl>          <lgl>            <dbl>   <dbl>
#   9.58e8  9.58e8    30    30 FALSE          FALSE                0       1
#   9.59e8  9.59e8    30    30 FALSE          FALSE                0       1
#   9.59e8  9.59e8    30    30 FALSE          FALSE                0       1
#   9.64e8  9.64e8    30    30 FALSE          FALSE                0       1
#   9.70e8  9.70e8    30    30 FALSE          FALSE                0       1
#   9.70e8  9.70e8    30    30 FALSE          FALSE                0       1
#   9.73e8  9.73e8    30    30 FALSE          FALSE                0       1
#   9.74e8  9.74e8    30    30 FALSE          FALSE                0       1
#   9.79e8  9.79e8    30    30 FALSE          FALSE                0       1
#   9.85e8  9.85e8    30    30 FALSE          FALSE                0       1
#   … with 38,121 more rows, and 1 more variable: date <dttm>
## Do not modify this line!
library(dplyr)
library(lubridate)
email_w_time<-email%>%mutate(date=with_tz(as.POSIXct(onset-883612800,origin =
"1998-01-01"),tzone = "UTC"))


# 3. Now let's take a look into the `people` dataset.
#   We can see some missing values in `person_name` column, but we can get a
#   person's name using his/her `email_id`.
#   For example, Albert Meyers's email ID is just `"albert.meyers"`.
#   To fill in missing values with email ID, let's first create a function
#   `email_id_to_name` to tranform `"albert.meyers"` into `"Albert Meyers"`.
#   You need to:
#    - Load the `stringr` package.
#    - Load the `purrr` package.
#    - Create a function  function `email_id_to_name` that take a string input
#   named `email_id` and returns the name extracted from the email id.
#   To do that, you can use:
#    - `str_split()` to extract the first and last name from `email_id`,
#      split by `"\\."`.
#    - `map_char` and `paste0()` to combine the first and last name using
#      `collapse = " "`.
#    - `str_trim()` to remove whitespace from start and end of string.
#    - `str_to_title()` to capitalize each word.
#   Name your.
```

```
## Do not modify this line!
library(stringr)
library(purrr)
email_id_to_name<-function(email_id){
  str_to_title(str_trim(map_chr(str_split(email_id,"\\."),~paste0(.,collapse = " "))))
}


# 4. Now let's implement the function you just created to our `people` tibble.
#    If `person_name` is not missing, do not modify it. If it is missing, change
#    `person_name` to the output of `email_id_to_name` by taking `email_id` as
#    input, you can use `email_id %>% email_id_to_name`.
#    To do that, you can use:
#      - `mutate()` and `ifelse()` to change the column `person_name`.
#      - `is.na()` to check if the `person_name` is missing.
#    Save your generated tibble into `people_new`, whose first few rows should
#    look like:
#    A tibble: 184 x 5
#    vertex.names email_id       person_name     role         dept
#    <dbl>       <chr>          <chr>           <chr>        <chr>
#    1           albert.meyers   Albert Meyers    Employee      Specialist
#    2           a..martin       Thomas Martin    Vice President NA
#    3           andrea.ring     Andrea Ring      NA            NA
#    4           andrew.lewis    Andrew Lewis     Director      NA
#    5           andy.zipper     Andy Zipper      Vice President Enron Online
#    6           a..shankman     Jeffrey Shankman President     Enron Global Mkts
#    7           barry.tycholiz  Barry Tycholiz   Vice President NA
#    8           benjamin.rogers Benjamin Rogers  Employee      Associate
#    9           bill.rapp       Bill Rapp        NA            NA
#    10           bill.williams   Bill Williams    NA            NA
#    … with 174 more rows
## Do not modify this line!
people_new<-people%>%
mutate(person_name=ifelse(is.na(person_name),email_id_to_name(email_id),person_nam
e))


# 5. We still have `NA` in `role` and `dept`. This time, we want to fill in the
#    missing values in `role` with `Employee` and missing values in `dept` as
#    `General`. Let's create a new tibble `people_new2` that fills such gaps...
#    and more!
#    First, load the `tidyr` package and create a vector `role_order =`
#    `c("Employee", "Trader", "Manager", "Managing Director", "Director",`
#    `"In House Lawyer", "Vice President", "President", "CEO")`
#    Now, can fill the missing roles and make it a factor using the levels
#    in `role_order`. To do that, you can use:
#      - `replace_na()` to fill in the missing values in these two columns.
#        Remember the missing values in `role` should now become `Employee`
#        and the missing values in `dept` should now become `General`.
#      - `mutate()` to change `role` into a factor with `factor()` and
#        specify `levels = role_order` to change `role` into a factor according
#        to our order.
```

```
#   Save your generated tibble into `people_new2` whose first few rows should
#   look like:
#   # A tibble: 184 x 5
#   vertex.names email_id       person_name     role          dept
#   <dbl>       <chr>           <chr>           <fct>         <chr>
#   1           albert.meyers   Albert Meyers   Employee      Specialist
#   2           a..martin       Thomas Martin   Vice President General
#   3           andrea.ring     Andrea Ring     Employee      General
#   4           andrew.lewis    Andrew Lewis    Director      General
#   5           andy.zipper     Andy Zipper     Vice President Enron Online
#   6           a..shankman     Jeffrey Shankman President    Enron Global Mkts
#   7           barry.tycholiz  Barry Tycholiz  Vice President General
#   8           benjamin.rogers Benjamin Rogers Employee      Associate
#   9           bill.rapp       Bill Rapp       Employee      General
#   10          bill.williams   Bill Williams   Employee      General
#   … with 174 more rows
## Do not modify this line!
library(tidyr)
role_order =c("Employee", "Trader", "Manager", "Managing Director", "Director","In House
Lawyer", "Vice President", "President", "CEO")
people_new2<-people_new%>%
mutate(role=replace_na(role,"Employee"),dept=replace_na(dept,"General"))%>%
  mutate(role=factor(role,levels = role_order))

# 6. Let's combine the two datasets `email_w_time` and `people_new2` together,
#   we want to keep the information about every email and add the name, email ID,
#   role and department for the sender as well as for the receiver.
#   The numbers in `tail` and `head` represent different people. The key to join
#   this two tibbles are `tail` and `head` from `email_w_time`, and `vertex.names`
#   from `people_new2`. In other words, you need to use two `left_join()` to
#   add first the information of the receiver and then of the sender.
#   You will also need to update the names of the columns added by the joins.
#   To add information of a receiver, you can use:
#     - `left_join()` to combine `email_w_time` and `people_new2`, specify
#       `by = c("tail"="vertex.names")` because `tail` represents the receiver
#       of an email.
#     - `rename()` to change the column names to specify that they're the
#       receiver information: change `email_id` to `receiver_email`, `person_name`
#       to `receiver`, `role` to `receiver_role` and `dept` to `receiver_dept`.
#   Then, to add information of a sender, you can similarly use:
#     - `left_join()` to combine with `people_new` again, this time with
#       `by = c("head"="vertex.names")` .
#     - `rename()` to change the column names to specify that they're the
#       sender information: change `email_id` to `sender_email`, `person_name`
#       to `sender`, `role` to `sender_role` and `dept` to `sender_dept`.
#   Finally, you can use:
#      -  `select()` to only keep the `date`, sender information and receiver
#         information (each has 4 columns with email ID, name, role and department).
#     - `starts_with()` to select all the four columns of sender / receiver
#       information.
#   Save your generated tibble into `t1`, which should have 9 columns, in the order of
#   `date`, `sender_email` `sender`, `sender_role`, `sender_dept`, `receiver_email`,
```

```
#   `receiver`, `receiver_role`, `receiver_dept`.
#   The first fews rows of `t1` should look like:
#   A tibble: 38,131 x 9
#   date                sender_email sender sender_role sender_dept receiver_email
#   <dttm>              <chr>        <chr>  <fct>       <chr>       <chr>
#   2000-05-15 08:35:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-05-18 04:15:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-05-24 02:58:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-07-19 07:09:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-09-28 02:45:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-09-28 02:52:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-10-27 04:38:00 debra.perli… Debra… Employee    General     debra.perling…
#   2000-11-10 02:52:00 debra.perli… Debra… Employee    General     debra.perling…
#   2001-01-05 03:17:00 debra.perli… Debra… Employee    General     debra.perling…
#   2001-03-23 02:02:00 debra.perli… Debra… Employee    General     debra.perling…
#   … with 38,121 more rows, and 3 more variables: receiver <chr>,
#   receiver_role <fct>, receiver_dept <chr>
## Do not modify this line!
t1<-email_w_time%>%left_join(people_new2,by = c("tail"="vertex.names"))%>%

rename(receiver_email=email_id,receiver=person_name,receiver_role=role,receiver_dept=
dept)%>%
  left_join(people_new2,by = c("head"="vertex.names"))%>%

rename(sender_email=email_id,sender=person_name,sender_role=role,sender_dept=dept)
%>%
  select(date,starts_with("sender"),starts_with("receiver"))




# 7. We noticed that there are emails that one person sent to him/her self
#   with the same `send_email` and `receiver_email`. We do not care about these
#   emails and want to filter them out.
#   - Use `filter` to filter out rows in `t1` whose `sender_email` is exactly
#   the same as `receiver_email`.
#   To simply our analysis with time the email is sent, we want to further
#   parse the information in `date` by creating new columns named `year`, `month`,
#   `day` and `hour`. To do that, you can use:
#     - `mutate()` to create the new columns `year`, `month`, `day` and `hour`.
#     - `year()` to extract `year` in `date`.
#     - `month()` to extract `month` in `date`.
#     - `day()` to extract `day` in `date`.
#     - `hour()` to extract `hour` in `date`.
#   Save your generated tibble into `t2`. The first few rows should look like:
#   A tibble: 34,427 x 13
#   date                sender_email sender sender_role sender_dept receiver_email
#   <dttm>              <chr>        <chr>  <fct>       <chr>       <chr>
#   2001-03-15 02:43:00 jeffrey.sha… Jeffr… President   Enron Glob… greg.whalley
#   2001-04-02 13:44:00 jeffrey.sha… Jeffr… President   Enron Glob… greg.whalley
#   2001-06-05 22:40:00 jeffrey.sha… Jeffr… President   Enron Glob… greg.whalley
#   2001-06-11 05:20:00 jeffrey.sha… Jeffr… President   Enron Glob… greg.whalley
```

```
#   2001-03-08 02:52:00 kim.ward    Kim W… Employee   General    jason.williams
#   2001-03-28 07:40:00 kim.ward    Kim W… Employee   General    jason.williams
#   2001-03-28 18:40:00 kim.ward    Kim W… Employee   General    jason.williams
#   2001-04-02 07:53:00 kim.ward    Kim W… Employee   General    jason.williams
#   2001-04-02 17:53:00 kim.ward    Kim W… Employee   General    jason.williams
#   2001-04-03 07:24:00 kim.ward    Kim W… Employee   General    jason.williams
#   … with 34,417 more rows, and 7 more variables: receiver <chr>,
#   receiver_role <fct>, receiver_dept <chr>, year <dbl>, month <dbl>, day <int>,
#   hour <int>
#   Now we have all the information we want. Let's try to answer some insteresting
#   questions in the following exercises:
## Do not modify this line!
t2<-t1%>%filter(sender_email!=receiver_email)%>%
  mutate(year=year(date),
         month=month(date),
         day=day(date),
         hour=hour(date))


# 8. Who are the top 3 people who sent emails the most? What are their roles and
#   departments?
#   To answer this, you can use:
#     - `group_by()` to group by `sender`, `sender_role` and `sender_dept`
#     - `summarize` by specify `count=n()` to count the number of emails
#     - `arrange()` to sort the tibble by count of emails in decreasing order
#     - `head()` to extract the first three rows.
#   Save your generated tibble into `p1` which should have the following structure:
#   A tibble: 3 x 4
#   sender        sender_role   sender_dept      count
#   <chr>         <fct>         <chr>            <int>
## Do not modify this line!
p1<-t2%>%group_by(sender,sender_role,sender_dept)%>%summarize(count=n())%>%
arrange(desc(count))%>%head(3)


# 9. During which period of day are people tend to send emails?
#   Use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`.
#     - `geom_histogram()` to plot a histogram for `hour`.
#       Specify `bins = 24` to set the bins
#     - `labs()` to format the labels such that:
#       - `title = "People send more emails during noon"`.
#       - `x = "Hour"`.
#       - `y = "Count(n)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the subtitle to the middle of the plot as well.
#       Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#   Save your plot to `g1`.
## Do not modify this line!
library(ggplot2)
g1<-ggplot(t2,mapping=aes(x=hour))+geom_histogram(bins = 24)+
```

```
 labs(title = "People send more emails during noon",x = "Hour",y =
"Count(n)")+theme_light()+
 theme(plot.title = element_text(hjust = 0.5),plot.subtitle = element_text(hjust = 0.5))



# 10. What is the trend of using emails? Do people use it more frequently in 1999 or 2001?
#    Let's first create a tibble to store the information.
#    Use:
#      - `filter()` to only keep the emails sent before `2002/01/01`.
#      - `group_by()` to group the data by `year`, `month` `sender` and
#       `sender_role`
#      - `summarize()` such that
#        - `date=min(date)` keep record of the earliest date in each group.
#        - `count=n()` the number of emails sent in that period.
#      - `arrange()` to order the rows by `date`.
#    Save your generated tibble into `p2`.
## Do not modify this line!
p2<-t2%>%filter(date<"2002/01/01")%>%group_by(year,month,sender_role)%>%
 summarize(date=min(date),count=n())%>%arrange(date)



# 11.Then let's visualize it. We want to plot the `count` against `date`
#    with colors spliting by `sender_role`.
#    Use:
#      - `ggplot()` to initialize a ggplot object.
#        Set its arguments `data`, `mapping`.
#      - `geom_point()` to plot a histogram for hour.
#      - `geom_smooth` to add a smoothing regression line.
#      - `labs()` to format the labels such that:
#        - `title = "People are using emails more frequently in 2001 than 1999"`.
#        - `x = "Date"`.
#        - `y = "Count(n)"`,
#        - `color = "Sender Role"`
#      - `theme_light()` to change the theme of plots.
#      - `theme()` to change the subtitle to the middle of the plot as well.
#        Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#    Save your plot to `g2`.
#'
## Do not modify this line!

g2<-ggplot(p2,mapping =
aes(x=date,y=count,color=sender_role))+geom_point()+geom_smooth()+
 labs(title = "People are using emails more frequently in 2001 than 1999",
      x = "Date",
      y = "Count(n)",
      color = "Sender Role")+theme_light()+  theme(plot.title = element_text(hjust =
0.5),plot.subtitle = element_text(hjust = 0.5))
```

HW8

```
# HW8: Fit different polynomial models
#
# 1. Load the `readr` and `dplyr` package.
#    Read in the file at `"data/polyfit.tsv"` and add an `id` column that
#    contains each row's number and assign the resulting tibble to variable
#    `data_with_id`.
#    To do so, you can do the following:
#    - read in the file at `"data/polyfit.tsv"` using `read_tsv`,
#    - call `mutate()` to create the `id` column using the `row_numbers()`
#      function.
#    `data_with_id` should print to:
#    # A tibble: 200 x 3
#    x      y    id
#    <dbl>  <dbl> <int>
#    1 0.897  0.855     1
#    2 0.266  7.79      2
#    3 0.372  4.52      3
#    4 0.573  3.08      4
#    5 0.908 -1.09      5
#    6 0.202  9.20      6
#    7 0.898 -1.21      7
#    8 0.945  1.63      8
#    9 0.661  2.75      9
#    10 0.629  4.98     10
#    # … with 190 more rows
## Do not modify this line!
library(readr)
library(dplyr)
data_with_id<-read_tsv("data/polyfit.tsv")%>%
  mutate(id = row_number())


# 2. We will now separate the data into two datasets: the train dataset and the
#    validation dataset. The train dataset will be used to fit linear models
#    to our data and the validation dataset will be used to compute the root
#    mean square error (RMSE) and assess the model's ability to generalize to
#    unseen data (unseen in the sense that the data points in the validation
#    dataset were not used to compute the optimal coefficients of the fitted
#    model).
#    To create these datasets, do the following:
#    - first, set the random seed to `0` using `set.seed()` to reproduce
#      identical deterministic results in the following, and save the seed to
#      a vector `seed` using `seed <- .Random.seed`,
#    - then, compute the `train_data` tibble by (randomly!) sampling 80 % of
#      the data in `data_with_id` and add a `split` column containing the
#      string value `"train"` for all rows. To do so, you can:
```

```
#       - use `sample_frac()` to shuffle the data and keep only 80 % of the
#       rows,
#       - then use `mutate()` to create the constant `split` column with
#       value `"train"` for all rows
#   - then, compute the `test_data` tibble, containing all the remaining data
#     (20 % of the original rows) and a `split` column with constant value
#     `"test"`. The `id` column computed in question 3. now comes in handy.
#     You can do the following:
#       - start with `data_with_id`,
#       - use `anti_join` on the `train_data` tibble by the `id` key to keep
#         only rows that have `id`s not in `train_data`'s `id`s,
#       - then use `mutate()` to create the constant `split` column with
#         value `"test"` for all rows
#   - finally, concatenate `train_data` and `test_data` row-wise, in this
#     order, and assign the resulting tibble to `all_data`. (you can use
#     `bind_rows()` to do so.)
#   `train_data` should print to:
#   # A tibble: 160 x 4
#   x     y    id split
#   <dbl> <dbl> <int> <chr>
#   1 0.605 1.58    180 train
#   2 0.861 0.180    53 train
#   3 0.347 5.42     74 train
#   4 0.732 1.34    113 train
#   5 0.741 1.94    179 train
#   6 0.724 1.64     40 train
#   7 0.391 4.21    175 train
#   8 0.191 8.99    183 train
#   9 0.454 2.96    127 train
#   10 0.640 4.72    121 train
#   # … with 150 more rows
#   `test_data` should print to:
#   # A tibble: 40 x 4
#   x      y    id split
#   <dbl>  <dbl> <int> <chr>
#   1 0.897  0.855     1 test
#   2 0.908 -1.09      5 test
#   3 0.992  0.495    19 test
#   4 0.380  5.01     20 test
#   5 0.935  0.205    22 test
#   6 0.267 10.1      26 test
#   7 0.382  3.57     29 test
#   8 0.870  0.207    30 test
#   9 0.477  3.32     49 test
#   10 0.732  1.22     50 test
#   # … with 30 more rows
#   `all_data` should print to:
#   # A tibble: 200 x 4
#   x     y    id split
#   <dbl> <dbl> <int> <chr>
#   1 0.605 1.58    180 train
#   2 0.861 0.180    53 train
```

```
#    3 0.347 5.42     74 train
#    4 0.732 1.34    113 train
#    5 0.741 1.94    179 train
#    6 0.724 1.64     40 train
#    7 0.391 4.21    175 train
#    8 0.191 8.99    183 train
#    9 0.454 2.96    127 train
#    10 0.640 4.72    121 train
#    # … with 190 more rows
## Do not modify this line!
set.seed(0)
seed <- .Random.seed
train_data<-data_with_id%>%sample_frac(0.8)%>%mutate(split = "train")
test_data<-data_with_id%>%anti_join(train_data,by = "id")%>%mutate(split = "test")
all_data<-bind_rows(train_data,test_data)
# 3. Load the `modelr` package.
#    Fit a quadratic model to the `train_data` and assign the result to
#    `model_quad`.
#    You should use `lm()` and the `poly()` function, setting its `degree` to
#    `2`.
#    Add the residuals of `model_quad` to `all_data` using `add_residuals()`
#    and assign the resulting tibble to `data_with_quad_resid`.
#    `data_with_quad_resid` should print to:
#     # A tibble: 200 x 5
#     x     y    id split  resid
#     <dbl> <dbl> <int> <chr>  <dbl>
#     1 0.605 1.58    180 train -2.03
#     2 0.861 0.180    53 train -0.255
#     3 0.347 5.42     74 train  0.338
#     4 0.732 1.34    113 train -0.910
#     5 0.741 1.94    179 train -0.197
#     6 0.724 1.64     40 train -0.713
#     7 0.391 4.21    175 train -0.743
#     8 0.191 8.99    183 train  3.85
#     9 0.454 2.96    127 train -1.73
#     10 0.640 4.72    121 train  1.44
#     # … with 190 more rows
## Do not modify this line!
library(modelr)
model_quad<-lm(y~poly(x,2,raw = TRUE),data = train_data)
data_with_quad_resid<-all_data%>%add_residuals(model_quad)
?poly

# 4. Load the `ggplot2` package.
#    Use `theme_set()` to set the theme to `theme_light()`.
#     It will automatically apply `theme_light()` to every plot you draw.
#    Generate a residual scatter plot (the x-axis should be `x` and the y-axis
#    should be `resid`) with a horizontal line of equation `resid = 0` and
#    assign it to `quad_resid_plot`.
#    To do so, you should do the following:
#    - call `ggplot()` on `data_with_quad_resid` using the `x` and `resid`
#      columns
```

```
#   - add a horizontal line at 0 (using `geom_ref_line()` for example),
#   - add a call to `geom_point()` to add the residuals
#   - use `labs()` to set the following labels:
#     - `y = "Residuals"`,
#     - `title = "The residuals show a secondary pattern that is not captured"`
#   - assign the result to `quad_resid_plot`
#   You should notice that the residuals are visibly not independent of
#   another, and that there is a clear pattern in their distribution according
#   to `x` that is far from uniform noise around 0.
#   We will find a better model to fit the data to capture this pattern in the
#   following.
## Do not modify this line!
library(ggplot2)
theme_set(theme_light())
quad_resid_plot<-ggplot(data_with_quad_resid,mapping=aes(x=x, y=resid)) +
  geom_ref_line(h=0)+
  geom_point(mapping=aes(x=x, y=resid))+
  labs(y = "Residuals",
       title = "The residuals show a secondary pattern that is not captured")

# 5. Load the `purrr` and `tidyr` package.
#   Create a `my_model()` function that takes in an argument `k` and returns a
#   fitted linear model on top of polynomials of `x` up to degree `k` with
#   respect to `y`, on the `train_data`.
#   You can use `lm()` and `poly(x, k, raw = TRUE)`
#   Then generate a tibble containing the entire original data, as well as
#   predictions and residuals for `k`-degree polynomial models for `k`
#   between 1 and 10 (both included).
#   To do so, you can:
#   - create a tibble with a column `k = 1:10`,
#   - add a `model` column containing the `k` order model (computed using
#     `map()` and `my_model()`) using `mutate()`,
#   - add a `data` column that contains the data, predictions and residuals
#     for the model of the same row, computed using `map()`,
#     `add_predictions()` and `add_residuals()` - using `mutate()` again,
#   - unnest the `data` column using `unnest()`
#   - keep all columns except for `model` using `select()`
#   `model_pred_resid` should print to:
#    # A tibble: 2,000 x 7
#          k     x     y    id split  pred    resid
#       <int> <dbl> <dbl> <int> <chr> <dbl>    <dbl>
#     1     1 0.605 1.58    180 train  2.84 -1.26
#     2     1 0.861 0.180    53 train  1.19 -1.01
#     3     1 0.347 5.42     74 train  4.50  0.920
#     4     1 0.732 1.34    113 train  2.02 -0.680
#     5     1 0.741 1.94    179 train  1.96 -0.0235
#     6     1 0.724 1.64     40 train  2.07 -0.436
#     7     1 0.391 4.21    175 train  4.22 -0.00553
#     8     1 0.191 8.99    183 train  5.50  3.49
#     9     1 0.454 2.96    127 train  3.81 -0.857
#    10     1 0.640 4.72    121 train  2.61  2.10
#    # … with 1,990 more rows
```

```
## Do not modify this line!
library(purrr)
library(tidyr)
my_model <- function(k){
 lm(y ~ poly(x, k, raw=TRUE), data=train_data)
}
model_pred_resid<-tibble(k = 1: 10) %>% mutate(model = map(k,my_model))%>%
 mutate(data=map(model, ~add_residuals(add_predictions(all_data,.),.)))%>%
 unnest(data)%>%select(-model)
#这里有个trick，add_predictions(all_data,.)仍是个tibble,所以可以用add_residuals

# 6. Create a function `mse` that computes the Mean Squared Error between two
#    vectors `y` and `pred`.
#    Create a tibble `model_mse` that contains the MSE for each of the
#    polynomial models, on each split (`train` and `test`).
#    To do so, you can:
#    - group `model_pred_resid` by `split` and `k` using `group_by()`,
#    - compute the MSE on each group using `mse()` and `summarize()`
#    `model_mse` should print to:
#     # A tibble: 20 x 3
#     # Groups:   split [2]
#       split     k   mse
#       <chr> <int> <dbl>
#     1 test      1 5.42
#     2 test      2 4.73
#     3 test      3 2.81
#     4 test      4 1.90
#     5 test      5 1.82
#     6 test      6 1.36
#     7 test      7 1.02
#     8 test      8 1.04
#     9 test      9 0.992
#    10 test     10 0.994
#    11 train     1 4.97
#    12 train     2 4.17
#    13 train     3 2.95
#    14 train     4 2.01
#    15 train     5 1.78
#    16 train     6 1.42
#    17 train     7 1.02
#    18 train     8 1.01
#    19 train     9 0.987
#    20 train    10 0.987
## Do not modify this line!
mse<-function(y,pred){
 mean((y-pred)^2)
}
model_mse<-model_pred_resid%>%group_by(split,k)%>%
 summarize(mse = mse(y,pred))

# 7. Create a plot showing how the train and test error vary with the
#    polynomial degree of the fitted model and assign it to variable
```

```
#    `plot_mse`.
#    To do so, you can do the following - in the same order:
#    - call `ggplot()` on `model_mse` with aesthetic containing `k` for
#      x-values, `mse` for y-values and color by `split`
#    - add a line plot  using `geom_line()`,
#    - enforce the x-axis to show ticks at integer values from 1 to 10
#      using `scale_x_continuous()` and setting its `breaks` argument to
#     `1:10`,
#    - use `labs()` to add the following labels:
#      - `x = "Polynomial Degree"`,
#      - `y = "RMSE"`,
#      - `title = "Train and test error vary with the polynomial degree"`,
#      - `subtitle = "Test error first decreases, then increases"`
## Do not modify this line!
plot_mse<-ggplot(model_mse,mapping = aes(x =k,y=mse,color=split))+
  geom_line()+
  scale_x_continuous(breaks = 1:10)+labs(
    x = "Polynomial Degree",
    y = "RMSE",
    title = "Train and test error vary with the polynomial degree",
    subtitle = "Test error first decreases, then increases"
  )


# 8. Find the degree with the smallest validation error and assign it to
#    variable `best_deg`.
#    To do so you can modify `model_mse` and do the following:
#    - filter only the rows containing validation errors and in which the
#      error is equal to the minimum error using `filter()` on the `split`
#      column to keep the errors on the validation set,
#    - filter on `mse` column, with `min()` used in the filtering condition
#      on `mse`,
#    - then, keep only the value of `k` using `pull()`
## Do not modify this line!
best_deg<-model_mse%>%filter(mse==min(mse))%>%filter(split == "test") %>% pull(k)


# 9. Generate a `residual_plot` figure by plotting the residuals of the best
#    polynomial model in terms of lowest test error with a horizontal reference
#    line at 0.
#    To do so you can:
#    - filter on `k` to keep only residuals for `k == best_deg` using
#      `filter()` on `model_pred_resid`,
#    - call `ggplot()` with `x` on the x-axis and `resid` on the y-axis,
#    - add points using `geom_point()`,
#    - add a horizontal line at 0 using `geom_hline()` (make its size be 1),
#    - use `labs()` to add the following labels:
#      - `y = "Residuals"`,
#      - `title = "No clear pattern in the residuals"`,
#      - `subtitle = "The model did its job!"`
## Do not modify this line!
```

```
residual_plot<-model_pred_resid%>%filter(k==best_deg)%>%
ggplot(mapping=aes(x=x,y=resid))+
  geom_point()+geom_hline(yintercept = 0,size=1)+labs(y = "Residuals",
                                title = "No clear pattern in the residuals",
                                subtitle = "The model did its job!")


# 10. Generate a scatter plot of all the data with two models - the best degree
#     and `k=7` - overlayed as lines and assign it to variable
#     `prediction_plot`.
#     To do so, you can:
#     - filter on `k` to keep only residuals for the two models using
#       `filter()` on `model_pred_resid`,
#     - transform `k` to a factor column using `mutate()` and `factor()`,
#     - call `ggplot()` with the aesthetic `x = x` and `y = y`,
#     - add the scatter plot with `geom_point()`,
#     - add the prediction line with a new aesthetic with `y = pred` and
#       `color = k`, with lines of `size = 1` using `geom_line()`
#     - use `labs()` to add the following labels:
#      - `color = "Polynomial Degree"`,
#      - `title = "Both models fit the data quite well"`,
#      - `subtitle = "Except close to the boundaries, is the model with k = 9 really better?"`
## Do not modify this line!
prediction_plot<-model_pred_resid%>%filter(k==7|k==best_deg)%>%
mutate(k=factor(k))%>%ggplot(mapping = aes(x=x,y=y))+
  geom_point()+geom_line(mapping = aes(y = pred,color=k),size=1)+labs(
    color = "Polynomial Degree",
    title = "Both models fit the data quite well",
    subtitle = "Except close to the boundaries, is the model with k = 9 really better?"
  )



# 11. We want to force the model to choose the smallest degree possible
#     among the models that have similar performances.
#     Create a function `mse_penalized()` with arguments `y`, `pred`, `k` and
#     `lambda` (`lambda` should have a default value of `0`) to compute a
#     penalized MSE that adds returns the `MSE(y,pred) + lambda * k`.
#     You can assume that `k` is an integer, and that `y` and `pred`
#     are numeric vectors of same length.
#     To calculate the MSE, you can calculate the mean of the squared difference.
#     Then, create a function `my_mse_penalized()` with arguments `df`, `k` and
#     `lambda` that returns a tibble with two columns:
#     - `loss` should contain the output of `mse_penalized()` computed
#       on the `y` and `pred` columns of `df`, and `k` and `lambda`,
#     - `lambda` should contain the value of `lambda`
#     You can assume that `df` is a dataframe containing two columns `y` and
#     `pred` of same length.
#     Use `my_mse_penalized()` to create a tibble `model_mse_penalized` from
#     `model_pred_resid` with values of the penalized penalized for each value
#     of `k`, as well as for each value of `lambda` in `(0, 0.1, 0.4)`.
#     To do so, you can:
```

```
#    - group `model_pred_resid` by `k` and `split`,
#    - nest the groups using `nest()`,
#    - add an `errors` column using `map()` on `data` to apply
#      `my_mse_penalized()` with the values of `lambda` given above and the
#      value of `k` of each group, using `mutate()`,
#    - unnest `errors` using `unnest()`,
#    - keep all columns except `data` using `select()`
#    `model_mse_penalized` should print to:
#    # A tibble: 60 x 4
#    # Groups:   k, split [20]
#          k split  loss lambda
#      <int> <chr> <dbl>  <dbl>
#     1     1 train  4.97    0
#     2     1 train  5.07    0.1
#     3     1 train  5.37    0.4
#     4     1 test   5.42    0
#     5     1 test   5.52    0.1
#     6     1 test   5.82    0.4
#     7     2 train  4.17    0
#     8     2 train  4.37    0.1
#     9     2 train  4.97    0.4
#    10     2 test   4.73    0
#    # … with 50 more rows
## Do not modify this line!
mse_penalized<-function(y,pred,k,lambda=0){
  mse(y,pred) + lambda * k
}
my_mse_penalized=function(df,k,lambda){
  tibble(loss=mse_penalized(df$y, df$pred, k, lambda), lambda=lambda)
}
#my_mse_penalized(model_mse_penalized$data[[1]],model_mse_penalized$k,lambda
=c(0, 0.1, 0.4))
model_mse_penalized=model_pred_resid%>% group_by(k,split) %>% nest()%>%
  mutate(errors=map(data, ~my_mse_penalized(.x,k,c(0,0.1,0.4))))%>%
  unnest(errors)%>%
  select(-data)


# 12. Create a `mse_penalized_plot` from `model_mse_penalized` that shows the
#    variation of the `loss` with `k` for each value of `lambda` on each
#    `split`.
#    To do so, you can:
#    - transform the `lambda` column of `model_mse_penalized` to a factor
#      column,
#    - call `ggplot()` with an aesthetic that maps `k` to the x-axis, `loss`
#      to the y-axis, `split` to the color and `lambda` to `linetype`,
#    - add lines using `geom_line()`,
#    - use `labs()` to add the following labels:
#     - `color = "Split"``,
#     - `x = "Polynomial Degree"``,
#     - `y = "Loss"`,
#     - `linetype = expression(lambda)`,
```

```
#     - `title = "A stronger penalization implies a smaller optimal degree"`,
#     - `subtitle = "expression(paste(lambda, " = 0 corresponds to the unpenalized case"))"`.
## Do not modify this line!

mse_penalized_plot<-model_mse_penalized%>%mutate(lambda=factor(lambda))%>%
 ggplot(mapping = aes(x=k,y=loss,color = split,linetype = lambda))+geom_line()+
 labs(
  color = "Split",
  x = "Polynomial Degree",
  y = "Loss",
  linetype = expression(lambda),
  title = "A stronger penalization implies a smaller optimal degree",
  subtitle = expression(paste(lambda, " = 0 corresponds to the unpenalized case"))
 )


# 13. Find the best degree according to penalized loss on the `test` split for
#     each value of the penalization parameter (`lambda`, which is in
#     `(0, 0.1, 0.4)`) and assign the tibble containing rows from
#     `model_mse_penalized` with the best degree to `best_deg_penalized`.
#     To do so, you can:
#     - keep only the `test` rows of `model_mse_penalized` using `filter()`,
#     - group the result by `lambda` using `group_by()`,
#     - keep only the rows with minimal loss using `filter()` and `min()`
#     Join `best_deg_penalized` to `model_pred_resid` to keep the data,
#     predictions and residuals for the polynomial models of degrees selected
#     by the penalized models and assign the resulting tibble to
#     `model_pred_resid_best`.
#     To do so, you can:
#     - left join `best_deg_penalized` to `model_pred_resid_best` on `k`,
#     - remove NA values using `drop_na()`
#     `best_deg_penalized` should print to:
#     # A tibble: 3 x 4
#     # Groups:   lambda [3]
#         k split  loss lambda
#       <int> <chr> <dbl>  <dbl>
#     1     4 test  3.50     0.4
#     2     7 test  1.72     0.1
#     3     9 test  0.992    0
#     `model_pred_resid_best` should print to:
#     # A tibble: 600 x 10
#         k     x     y    id split.x  pred   resid split.y  loss lambda
#       <int> <dbl> <dbl> <int> <chr>   <dbl>   <dbl> <chr>   <dbl>  <dbl>
#     1     4 0.605 1.58    180 train    2.25 -0.675  test     3.50    0.4
#     2     4 0.861 0.180    53 train    1.26 -1.08   test     3.50    0.4
#     3     4 0.347 5.42     74 train    6.37 -0.944  test     3.50    0.4
#     4     4 0.732 1.34    113 train    1.69 -0.347  test     3.50    0.4
#     5     4 0.741 1.94    179 train    1.67  0.265  test     3.50    0.4
#     6     4 0.724 1.64     40 train    1.70 -0.0656 test     3.50    0.4
#     7     4 0.391 4.21    175 train    5.55 -1.34   test     3.50    0.4
#     8     4 0.191 8.99    183 train    7.38  1.62   test     3.50    0.4
#     9     4 0.454 2.96    127 train    4.36 -1.40   test     3.50    0.4
```

```
#    10     4 0.640 4.72    121 train    2.00  2.71   test     3.50    0.4
#    # … with 590 more rows
## Do not modify this line!
best_deg_penalized<-model_mse_penalized%>%filter(split == "test")%>%
 group_by(lambda)%>%filter(loss==min(loss))

model_pred_resid_best<-model_pred_resid%>%left_join(best_deg_penalized,by = "k")%>%
drop_na()


# 14. Generate residual plots for the three different values of `k` retained
#    by the penalized criterion with reference lines at 0 and assign the plot
#    to `residual_plot_best`.
#    To do so, you can:
#    - use `ggplot()` on `model_pred_resid_best`, with `x` on the x-axis and
#      `"Residuals"` on the y-axis
#    - add points using `geom_point()`,
#    - add a horizontal line at 0 using `geom_hline()` (make its size be 1),
#    - facet the plot by `k` using `facet_wrap()`
#    - use `labs()` to add the following labels:
#      - `y = "Residuals"`,
#      - `title = "No clear pattern in the residuals, except maybe for k = 6"`,
#      - `subtitle = "Models with k = 7 and k = 9 seem to have done their job!"`
#
## Do not modify this line!
residual_plot_best<-ggplot(model_pred_resid_best,mapping =
aes(x=x,y=resid))+geom_point()+geom_hline(yintercept = 0,size=1)+
 facet_wrap(~k)+labs(y = "Residuals",
                title = "No clear pattern in the residuals, except maybe for k = 6",
                subtitle = "Models with k = 7 and k = 9 seem to have done their job!")



# HW8: temperature
#
# For this assignment, you will use temperature data
# provided by [Berkeley Earth](http://berkeleyearth.org/data/).
# More specifically, you will use time series of average
# monthly air temperatures over land in every country
# between 1743 and today.
#
# Throughout the exercise:
# - Use `theme_light()` for the plots.
# - Do not change the default position of the plot title.
# - Do not print the plot.
#
# 1. Load the `tidyverse`, `maps`, `modelr`, and `lubridate` packages.
#    Use the function `read_csv()` to read the dataset
#    `temperature.csv` from path `/course/data/`.
#    Store it into a tibble `temperature_original`.
## Do not modify this line!
library(tidyverse)
```

```
library(maps)
library(modelr)
library(lubridate)
temperature_original<-read_csv("/course/data/temperature.csv")
theme_set(theme_light())

# 2. Add two columns `year` and `month` to `temperature_original`.
#    `year` is the year part of the `date` column and
#    its class is numeric.
#    `month` is the month part of `date` column and
#    its class is factor. The levels are `"Jan"`, `"Feb"`,
#    ..., `"Dec"`.
#    Futhermore, filter you data to focus on the
#    20th and 21th centuries and drop `NA` values in all columns.
#    Save the altered tibble to `temperature`.
#    To do that, you can use:
#    - `mutate()` to add columns.
#    - `lubridate::year()` to extract year from `date`
#    - `lubridate::month()` to extract month from `date`.
#      Set the `label` parameter to `TRUE` to display
#      the month as a character string.
#    - `factor()` to transform `month` into factor.
#      Set the `levels` parameter to `"Jan"`, `"Feb"`,
#      ..., `"Dec"`. Make sure the levels are ordered.
#    - `filter()` to filter out years that are smaller
#      than 1900.
#    - `drop_na()` to drop NA values.
#    `temperature` should print to :
#    # A tibble: 271,355 x 6
#      date       temperature country     region  year month
#      <date>           <dbl> <chr>       <chr>  <dbl> <fct>
#    1 1900-01-01       -3.43 Afghanistan Asia    1900 Jan
#    2 1900-02-01        1.23 Afghanistan Asia    1900 Feb
#    3 1900-03-01       10.5  Afghanistan Asia    1900 Mar
#    4 1900-04-01       13.4  Afghanistan Asia    1900 Apr
#    5 1900-05-01       20.3  Afghanistan Asia    1900 May
#    6 1900-06-01       24.4  Afghanistan Asia    1900 Jun
#    7 1900-07-01       27.3  Afghanistan Asia    1900 Jul
#    8 1900-08-01       24.9  Afghanistan Asia    1900 Aug
#    9 1900-09-01       20.8  Afghanistan Asia    1900 Sep
#    10 1900-10-01      14.0  Afghanistan Asia    1900 Oct
#    # … with 271,345 more rows
## Do not modify this line!
temperature<-temperature_original%>%
  mutate(year = lubridate::year(date),month = lubridate::month(date,label = TRUE))%>%
  mutate(month = factor(month,levels = unique(month),ordered = FALSE))%>%
  filter(year>=1900)%>%
  drop_na()

# 3. Compute the change in temperature for all countries
#    in `temperature` between March 1910 and March 2010.
#    Save the result in `temperature_change`. It should has
```

```
#   two columns `country` and `temperature`.
#   To do that, you can use:
#   - `filter()` to filter the data.
#   - `group_by()` to group by `country`.
#   - `arrange()` to arrange the data by `year`.
#   - `summarize()` to compute the difference in temperature.
#   `temperature_change` should print to :
#   # A tibble: 199 x 2
#     country       temperature
#     <chr>             <dbl>
#   1 Afghanistan        6.17
#   2 Albania            1.63
#   3 Algeria            3.85
#   4 American Samoa     2.24
#   5 Andorra            0.164
#   6 Angola             0.962
#   7 Anguilla           2.88
#   8 Argentina          3.39
#   9 Armenia            3.80
#  10 Aruba              3.36
#   # … with 189 more rows
## Do not modify this line!
temperature_change<-temperature%>%filter(year == 1910 | year == 2010)%>%
 filter(month == "Mar")%>%
 group_by(country)%>%
 summarize(temperature = diff(temperature))


# 4. Produce a worlwide map of the temperature change between
#   March 1910 and March 2010 using `temperature_change`.
#   To do that, you can use:
#   - `ggplot()` to initialize a ggplot object setting `map_id`
#     to `country` in `aes()`.
#   - `geom_map()` to draw the map.
#     Set `fill` to `temperature` in `aes()`.
#     Set `map` to `map_data("world")`.
#   - `expand_limits()` to set `x` from `-180` to `180`,
#     and `y` from `-90` to `90`.
#   - `scale_fill_gradient()` to set the legend color from
#     `"blue"` to `"red"` so that blue stands for cold and red
#     stands for hot.
#   - `labs()` to set:
#     - `title` to `"Worldwide temperature change between March 1910 and March 2010"`.
#     - `subtitle` to `"Most countries experience an increase"`.
#     - `x` to `"Longitude"`.
#     - `y` to `"Latitude"`.
#     - `fill` to `"Temperature (°C)"`.
#   - `theme_light()` to set light theme (i.e. a light background).
#   Save the ggplot object into `temperature_map`.
## Do not modify this line!
temperature_map<-ggplot(data = temperature_change, mapping = aes(map_id = country))+
 geom_map(mapping = aes(fill = temperature), map = map_data("world"))+
```

```
 expand_limits(x = c(-180,180),y=c(-90,90))+
 scale_fill_gradient(low="blue",high = "red")+
 labs(title="Worldwide temperature change between March 1910 and March 2010",
     subtitle="Most countries experience an increase",
     x="Longitude",
     y="Latitude",
     fill="Temperature (°C)")


# 5. Filter the data from `temperature` to keep all data
#    corresponding to `"Switzerland"`. Save the result in `sw`.
#    To do that, you can use `filter()` to filter the data.
#    `sw` should print to :
#    # A tibble: 1,364 x 6
#      date       temperature country     region  year month
#      <date>           <dbl> <chr>       <chr>  <dbl> <fct>
#    1 1900-01-01      -0.720 Switzerland Europe  1900 Jan
#    2 1900-02-01       1.10  Switzerland Europe  1900 Feb
#    3 1900-03-01      -0.560 Switzerland Europe  1900 Mar
#    4 1900-04-01       5.34  Switzerland Europe  1900 Apr
#    5 1900-05-01       9.59  Switzerland Europe  1900 May
#    6 1900-06-01      14.8   Switzerland Europe  1900 Jun
#    7 1900-07-01      17.3   Switzerland Europe  1900 Jul
#    8 1900-08-01      14.7   Switzerland Europe  1900 Aug
#    9 1900-09-01      13.8   Switzerland Europe  1900 Sep
#    10 1900-10-01      8.10  Switzerland Europe  1900 Oct
#    # … with 1,354 more rows
## Do not modify this line!
sw<-temperature%>%filter(country == "Switzerland")


# 6. Plot the box plot of temperature for each month in Switzerland
#    using tibble `sw` and draw the mean temperature to the plot
#    using red dots.
#    To do that, you can use:
#    - `ggplot()` to to initialize a ggplot object, setting `aes()`
#      correctly to plot `temperature` vs `date`.
#    - `geom_boxplot()` to draw the box plot.
#    - `stat_summary()` to draw the mean temperature.
#      Set `fun.y` to `mean`, `colour` to `red`, `size` to `2`, and
#      `geom` to `point`.
#    - `labs()` to set:
#      - `title` to `"Temperature in Switzerland"`.
#      - `subtitle` to `"Winters are colder, summers are warmer (mean in red)"`.
#      - `x` to `"Month"`.
#      - `y` to `"Temperature (°C)"`.
#    - `theme_light()` to set light theme (i.e. a light background).
#    Save the ggplot into `switzerland_temperature_plot`.
## Do not modify this line!
switzerland_temperature_plot<-ggplot(sw,mapping=aes(x=month,y=temperature))+
 geom_boxplot()+
 stat_summary(fun.y="mean",color="red",size=2,geom="point")+
```

```
  labs(title="Temperature in Switzerland",
       subtitle = "Winters are colder, summers are warmer (mean in red)",
       x="Month",
       y = "Temperature (°C)")


# 7. Fit a simple linear model: monthly temperature
#    vs. month for Swizerland using `sw`. Because there is a strong
#    seasonal effect, it is unclear whether the temperature is
#    increasing over the years.We want here to approximate the
#    monthly evolution of temperature.
#    To do that, please use `lm()` to fit the linear regression
#    model for `temperature` against `month`.
#    Save the model to `sw_mod`.
#    `sw_mod %>% broom::tidy()` should print :
#    # A tibble: 12 x 5
#    term        estimate std.error statistic   p.value
#    <chr>          <dbl>    <dbl>    <dbl>     <dbl>
#    1 (Intercept)    -1.84    0.152    -12.1  5.20e- 32
#    2 monthFeb        1.18    0.215      5.49 4.82e-  8
#    3 monthMar        4.78    0.215     22.2  1.35e- 93
#    4 monthApr        8.27    0.215     38.5  3.08e-219
#    5 monthMay       12.9     0.215     60.1  0.
#    6 monthJun       16.3     0.215     75.6  0.
#    7 monthJul       18.4     0.215     85.4  0.
#    8 monthAug       17.9     0.215     83.0  0.
#    9 monthSep       14.6     0.216     67.6  0.
#    10 monthOct       9.62    0.216     44.6  3.79e-268
#    11 monthNov       4.51    0.216     20.9  1.56e- 84
#    12 monthDec       1.05    0.216      4.89 1.15e-  6
## Do not modify this line!
sw_mod<-lm(temperature~month,data=sw)


# 8. Use model `sw_mod` to predict temperature in 1910 and add
#    the predictions to `sw`. Name the prediction column
#    `mean temperature` and save the result into `sw_seasonal`.
#    To do that, you can use:
#    - `add_predictions()` to add a column that contains
#      the predicted values using `sw_mod`.
#    - `filter()` to select data that is in 1910.
#    `sw_seasonal` should print to :
#    # A tibble: 12 x 7
#      date       temperature country     region  year month `mean temperature`
#      <date>           <dbl> <chr>       <chr>  <dbl> <fct>            <dbl>
#    1 1910-01-01     -1.14  Switzerland Europe  1910 Jan             -1.84
#    2 1910-02-01      0.107 Switzerland Europe  1910 Feb             -0.658
#    3 1910-03-01      2.71  Switzerland Europe  1910 Mar              2.94
#    4 1910-04-01      5.41  Switzerland Europe  1910 Apr              6.43
#    5 1910-05-01      9.34  Switzerland Europe  1910 May             11.1
#    6 1910-06-01     14.4   Switzerland Europe  1910 Jun             14.4
#    7 1910-07-01     14.2   Switzerland Europe  1910 Jul             16.5
```

```
#    8 1910-08-01     14.9   Switzerland Europe  1910 Aug            16.0
#    9 1910-09-01     10.4   Switzerland Europe  1910 Sep            12.7
#    10 1910-10-01      8.17  Switzerland Europe  1910 Oct             7.78
#    11 1910-11-01      1.27  Switzerland Europe  1910 Nov             2.67
#    12 1910-12-01      0.881 Switzerland Europe  1910 Dec            -0.785
## Do not modify this line!
sw_seasonal<-sw%>%add_predictions(sw_mod,"mean temperature")%>%filter(year==
1910)


# 9. Plot the predicted monthly temperature in Swizerland in 1910
#    using `sw_seasonal` based on top of `switzerland_temperature_plot`.
#    To do that, you can use:
#    - Call `switzerland_temperature_plot`.
#    - `geom_point()` to draw the points.
#      Set `color` to `green`, `size` to `2`, and `pch` to `3`.
#    - `labs()` to set:
#      - `subtitle` to `"Winters are colder, summers are warmer (mean/fitted in red/green)"`.
#    Save the ggplot in `sw_seasonal_plot`.
## Do not modify this line!
sw_seasonal_plot<-
switzerland_temperature_plot+geom_point(data=sw_seasonal,color="green",size=2,pch=
3)+
  labs(subtitle="Winters are colder, summers are warmer (mean/fitted in red/green)")


# 10. Add a residual column to `sw` and save the result in
#     `sw_residuals`.
#     To do that, you can use `add_residuals()`.
#     `sw_residuals` should print to :
#     # A tibble: 1,364 x 7
#       date       temperature country     region  year month  resid
#       <date>           <dbl> <chr>       <chr>  <dbl> <ord>  <dbl>
#     1 1900-01-01     -0.720 Switzerland Europe  1900 Jan    1.12
#     2 1900-02-01      1.10  Switzerland Europe  1900 Feb    1.75
#     3 1900-03-01     -0.560 Switzerland Europe  1900 Mar   -3.50
#     4 1900-04-01      5.34  Switzerland Europe  1900 Apr   -1.10
#     5 1900-05-01      9.59  Switzerland Europe  1900 May   -1.49
#     6 1900-06-01     14.8   Switzerland Europe  1900 Jun    0.342
#     7 1900-07-01     17.3   Switzerland Europe  1900 Jul    0.794
#     8 1900-08-01     14.7   Switzerland Europe  1900 Aug   -1.36
#     9 1900-09-01     13.8   Switzerland Europe  1900 Sep    1.11
#     10 1900-10-01      8.10  Switzerland Europe  1900 Oct    0.313
#     # … with 1,354 more rows
## Do not modify this line!
sw_residuals<-sw%>%add_residuals(sw_mod)


# 11. Plot the residuals from the model using `sw_residuals`
#     and plot a smoothing line.
#     To do that, you can use:
#     - `ggplot()` to to initialize a ggplot object, setting `aes()`
```

```
#      correctly to plot `resid` vs `year`.
#    - `geom_point()` to draw the dots.
#    - `geom_smooth()` to draw the smoothing line.
#      Set `method` to `lm`, `col` to `red`, `size` to `2`, and
#      `se` to `FALSE`.
#    - `labs()` to set:
#      - `title` to `"Residuals from the model"`.
#      - `subtitle` to `"We can notice a small yearly increase"`.
#      - `x` to `"Year"`.
#      - `y` to `"Residuals (°C)"`.
#    - `theme_light()` to set light theme (i.e. a light background).
#    Save the ggplot into `residuals_plot`.
## Do not modify this line!
residuals_plot<-ggplot(sw_residuals,mapping = aes(y=resid,x=year))+geom_point()+
  geom_smooth(method = "lm",color="red",size=2,se=F)+
  labs(title = "Residuals from the model",
       subtitle = "We can notice a small yearly increase",
       x="Year",
       y="Residuals (°C)")



# 12. Improve the previous model by adding a linear effect
#    for the temperature as a function of the year using `sw`.
#    To do that, you can use `lm()` to fit the linear regression
#    model for `temperature` against `year` and `month`.
#    Save the model in `sw_improved_mod`.
#    `sw_improved_mod %>% broom::tidy()` should print to :
#    # A tibble: 13 x 5
#    term        estimate std.error statistic   p.value
#    <chr>          <dbl>    <dbl>    <dbl>     <dbl>
#    1 (Intercept) -24.3      2.55       -9.53 7.04e- 21
#    2 year          0.0115   0.00130     8.82 3.33e- 18
#    3 monthFeb      1.18     0.209       5.64 2.03e-  8
#    4 monthMar      4.78     0.209       22.9  4.97e- 98
#    5 monthApr      8.27     0.209       39.5  8.68e-228
#    6 monthMay     12.9      0.209       61.7  0.
#    7 monthJun     16.3      0.209       77.8  0.
#    8 monthJul     18.4      0.209       87.8  0.
#    9 monthAug     17.9      0.209       85.4  0.
#    10 monthSep    14.6      0.210       69.5  0.
#    11 monthOct     9.63     0.210       45.9  4.51e-278
#    12 monthNov     4.52     0.210       21.6  8.77e- 89
#    13 monthDec     1.06     0.210        5.05 5.00e-  7
## Do not modify this line!
sw_improved_mod<-lm(temperature~year+month,sw)



# 13. Compare the `sw_mod` and `sw_improved_model` using `anova()`.
#    The analysis of variance `anova` model tells us what model
#    explains more variance in the data.
#    Save the result to `sw_anova`.
```

```
#    `sw_anova %>% broom::tidy()` should print to :
#    # A tibble: 2 x 6
#      res.df   rss    df sumsq statistic   p.value
#       <dbl> <dbl> <dbl> <dbl>     <dbl>     <dbl>
#    1   1352 3564.    NA   NA       NA   NA
#    2   1351 3370.     1  194.      77.9  3.33e-18
## Do not modify this line!
sw_anova<-anova(sw_mod,sw_improved_mod)


# 14. Use nested data and list-columns to fit the same model
#    to every country in the dataset, as well as to add predictions
#    and residuals for each fitted model using `temperature`.
#    Save the result in `by_country`.
#    To do that, you can :
#    - create a function `country_model <- function(df) {#your code}`
#      that takes as input a dataset or tibble df and returns
#      a fitted linear regression model of `temperature`
#      against `year` and month. You can assume that `df` contains
#      variables `temperature`, `year` and month.
#      For instance, `country_model(sw)` should return `sw_improved_mod`.
#    - use `group_by()` to group by `country` and `region`.
#    - use `nest()` to nest the data.
#    - use `mutate()` to add columns.
#    - use `purrr::map()` to apply `country_model()` to each row of `data`.
#      Note that `purrr::map()` conflicts with `map::map()`.
#    - use `map2()` to apply `add_residuals()` and `add_predictions` to
#      `data` and the models we just made.
#    The first rows of `by_country` print should be :
#    # A tibble: 199 x 6
#    # Groups:   country, region [199]
#      country    region          data model residuals    predictions
#      <chr>      <chr>     <list<df[,4> <lis> <list>       <list>
#    1 Afghanist… Asia       [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    2 Albania    Europe     [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    3 Algeria    Africa     [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    4 American … Other      [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    5 Andorra    Europe     [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    6 Angola     Africa     [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    7 Anguilla   Other      [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    8 Argentina  South Am…  [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    9 Armenia    Asia       [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    10 Aruba     South Am…  [1,364 × 4] <lm>  <tibble [1,… <tibble [1,36…
#    # … with 189 more rows
## Do not modify this line!
country_model<-function(df){
  lm(temperature~year+month,data=df)
}
by_country<-temperature%>%group_by(country,region)%>%nest()%>%
  mutate(model= purrr::map(data,country_model),residuals = map2(data, model,
                                          add_residuals),predictions = map2(data, model,
add_predictions))
```

```
# 15. Filter the data that is in `"Iceland"`, `"Ireland"`, `"Spain"`,
#    `"Sweden"`, and `"Haiti"` in 1910 using `by_country`.
#    To do that, you can use:
#    - `filter()` to filter the countries.
#    - `unnest()` to unnest `predictions`.
#    - `filter()` to filter year.
#    The first rows of `by_country` print should be :
#    # A tibble: 60 x 10
#    # Groups:   country, region [5]
#      country region       data model residuals date       temperature  year
#      <chr>   <chr>  <list<df[,> <lis> <list>    <date>         <dbl> <dbl>
#    1 Haiti   North… [1,365 × 4] <lm>  <tibble … 1910-01-01      23.4  1910
#    2 Haiti   North… [1,365 × 4] <lm>  <tibble … 1910-02-01      23.9  1910
#    3 Haiti   North… [1,365 × 4] <lm>  <tibble … 1910-03-01      24.1  1910
#    4 Haiti   North… [1,365 × 4] <lm>  <tibble … 1910-04-01      25.0  1910
#    5 Haiti   North… [1,365 × 4] <lm>  <tibble … 1910-05-01      25.9  1910
## Do not modify this line!
monthly_pattern<-by_country%>%filter(country %in% c("Iceland", "Ireland", "Australia",
"Brazil","Haiti"))%>%
  unnest(predictions)%>%filter(year==1910)


# 16. Plot the seasonal pattern for the monthly temperature
#    in `"Iceland"`, `"Ireland"`, `"Australia"`, `"Brazil"`, and
#    `"Haiti"` in 1910 using `monthly_pattern`.
#    To do that, you can use:
#    - `filter()` to filter countries and year.
#    - `unnest()` to unnest the prediction column.
#    - `ggplot()` to to initialize a ggplot object.
#      Set `x`, `y` in `aes()` to plot `pred` against `month`.
#      Set `color`, and `group` parameters to `country`.
#      `aes()` correctly.
#    - `geom_point()` to draw the points.
#    - `geom_line()` to draw the line.
#    - `labs()` to set:
#      - `title` to `"The seasonal patterns in the northern/southern hemispheres are
inverted"`.
#      - `subtitle` to `"Iceland/Haiti has the lowest/highest temperature"`.
#      - `x` to `"Month (in 1910)"`.
#      - `y` to `"Temperature (°C)"`,
#      - `color` to `"Country"`.
#    - `theme_light()` to set light theme (i.e. a light background).
#    Save the ggplot in `monthly_pattern_plot`.
## Do not modify this line!
monthly_pattern_plot<-
ggplot(monthly_pattern,mapping=aes(x=month,y=pred,color=country,group=country))+geo
m_point()+geom_line()+
  labs(title="The seasonal patterns in the northern/southern hemispheres are inverted",
       subtitle="Iceland/Haiti has the lowest/highest temperature",x="Month (in
1910)",y="Temperature (°C)",color="Country")
```

```
# 17. Find the 10 countries for which the model is the worst using
#    `by_country`.
#    To do that, you can use:
#    - `mutate()` to add a `glance` column which corresponds to
#      the mapping of function `broom::glance()` to `model` column
#      using `purrr::map()`
#    - `unnest()` to unnest `glance`.
#    - `select()` to select `country`, `region`, `adj.r.squared`.
#    - `arrange()` to arrange by `adj.r.squared`.
#    Save the tibble into `worst_predict`, It should print to :
#    # A tibble: 10 x 3
#   # Groups:   country, region [10]
#     country                   region       adj.r.squared
#     <chr>                    <chr>             <dbl>
#   1 Rwanda                     Africa            0.452
#   2 Colombia                   South America       0.530
#   3 Burundi                   Africa            0.533
#   4 Samoa                     Oceania           0.562
#   5 American Samoa                Other             0.601
#   6 Kiribati                  Other             0.611
#   7 Democratic Republic of the Congo Other              0.618
#   8 Papua New Guinea              Oceania           0.627
#   9 Ecuador                   South America       0.627
#  10 Venezuela                   South America       0.637
#
## Do not modify this line!
worst_predict<-by_country%>%mutate(glance=purrr::map(model,broom::glance))%>%
unnest(glance)%>%
 select(country,region,adj.r.squared)%>%
 arrange(adj.r.squared)%>%ungroup()%>%filter(rank(adj.r.squared)<=10)

# HW8: City Weather Prediction
#
# This exercise is based on historical hourly weather data 2012-2017 obtained from
# [Kaggle]
# (https://www.kaggle.com/selfishgene/historical-hourly-weather-
data#wind_direction.csv).
# The dataset contains approximatelyl 5 years of high temporal resolution
# data of various weather attributes, such as temperature, humidity,
# air pressure, etc. This data is available for 30 US and Canadian Cities,
# as well as 6 Israeli cities.
# Each attribute has its own file and is organized such that the rows are
# the time axis (it's the same time axis for all files), and the columns are
# the different cities (it's the same city ordering for all files as well).
#
# 1. Load the packages `tidyverse`, `lubridate`.
#    Use `read_csv` to load the file `temperature_cities.csv` located in the
#    folder `/course/data`.
#    Store the data into a tibble `temperature`.
#    `temperature` should print to:
#    # A tibble: 45,253 x 5
```

```
#   datetime            Vancouver Phoenix Miami `New York`
#   <dttm>                <dbl>   <dbl> <dbl>     <dbl>
#   1 2012-10-01 12:00:00      NA     NA    NA        NA
#   2 2012-10-01 13:00:00     285.    297.  300.      288.
#   3 2012-10-01 14:00:00     285.    297.  300.      288.
#   4 2012-10-01 15:00:00     285.    297.  300.      288.
#   5 2012-10-01 16:00:00     285.    297.  300.      288.
#   6 2012-10-01 17:00:00     285.    297.  300.      288.
#   7 2012-10-01 18:00:00     285.    297.  300.      289.
#   8 2012-10-01 19:00:00     285.    297.  300.      289.
#   9 2012-10-01 20:00:00     285.    297.  300.      289.
#   10 2012-10-01 21:00:00     285.    297.  300.      289.
# … with 45,243 more rows
## Do not modify this line!
library(tidyverse)
library(lubridate)
temperature<-read_csv("/course/data/temperature_cities.csv")
theme_set(theme_light())

# 2. As you can see, each city is represented by each column of the tibbles
#   (i.e., humidity variable is spread across multiple columns).
#   Our goal is to create a tibble `temperature_full` with only one column
#   that contains the data from all cities.
#   We need to tidy it. You can use :
#     - `pivot_longer()` to make rowise record for each city.
#     - `mutate()` to add the following extra columns:
#       - `temperature`: convert `temperature` from Kelvin to Celsius
#         scale. (Kelvin - 273.15 = Celsius)
#       - `yday` using `yday()` that contain the day of the corresponding
#         date in `datetime`.
#       - `hour` using `hour()` that contain the hour of the corresponding
#         date in `datetime` and `factor()` to make it a factor variable.
#       - `yearfrac` that represent the fraction of this day in this year.
#         (Use `yday(make_date(year(datetime), 12, 31))` to calculate the
#         total days in this year. The fraction is just the `yday` divided
#         by total days in this year.)
#     - `drop_na()` to drop `NA` values of this tibble.
#   `temperature_full` should print to:
#   # A tibble: 178,616 x 6
#   datetime            city      temperature  yday yearfrac hour
#   <dttm>              <chr>           <dbl> <dbl>    <dbl> <fct>
#   1 2012-10-01 13:00:00 Vancouver       11.5   275    0.751 13
#   2 2012-10-01 13:00:00 Phoenix         23.5   275    0.751 13
#   3 2012-10-01 13:00:00 Miami           26.6   275    0.751 13
#   4 2012-10-01 13:00:00 New York        15.1   275    0.751 13
#   5 2012-10-01 14:00:00 Vancouver       11.5   275    0.751 14
#   6 2012-10-01 14:00:00 Phoenix         23.5   275    0.751 14
#   7 2012-10-01 14:00:00 Miami           26.6   275    0.751 14
#   8 2012-10-01 14:00:00 New York        15.1   275    0.751 14
#   9 2012-10-01 15:00:00 Vancouver       11.5   275    0.751 15
#   10 2012-10-01 15:00:00 Phoenix        23.5   275    0.751 15
#   # … with 178,606 more rows
```

```
#   Filter New York temperature and store the tibble into `temperature_ny`.
#   (You can use `filter()`).
#   `temperature_ny` should print to:
#   # A tibble: 44,460 x 6
#   datetime            city     temperature  yday yearfrac hour
#   <dttm>              <chr>          <dbl> <dbl>    <dbl> <fct>
#   1 2012-10-01 13:00:00 New York       15.1   275    0.751 13
#   2 2012-10-01 14:00:00 New York       15.1   275    0.751 14
#   3 2012-10-01 15:00:00 New York       15.2   275    0.751 15
#   4 2012-10-01 16:00:00 New York       15.3   275    0.751 16
#   5 2012-10-01 17:00:00 New York       15.3   275    0.751 17
#   6 2012-10-01 18:00:00 New York       15.4   275    0.751 18
#   7 2012-10-01 19:00:00 New York       15.5   275    0.751 19
#   8 2012-10-01 20:00:00 New York       15.6   275    0.751 20
#   9 2012-10-01 21:00:00 New York       15.7   275    0.751 21
#   10 2012-10-01 22:00:00 New York       15.7   275    0.751 22
#   # … with 44,450 more rows
## Do not modify this line!
temperature_full<-temperature%>%pivot_longer(-datetime,names_to = "city",values_to =
"temperature")%>%
  mutate(temperature=temperature-273.15, yday=yday(datetime),
         yearfrac = yday/yday(make_date(year(datetime), 12, 31)), hour =
factor(hour(datetime)))%>%drop_na()
temperature_ny<-temperature_full%>%filter(city == "New York")


# 3. Plot New York temperature from 2012 to 2018.
#   To do that, you can use:
#     - `ggplot()` to initialize the plot object and set `aes()` so that
#       the plot will show `temperature` on y axis and `datetime` on `x`
#       axis.
#     - `geom_line()` to draw a line plot
#     - `labs()` to set `title` as
#         `"New York temperature trend from 2012 to 2018"`,
#         `y` as `"Temperature (C)"` and `x` as `"Date"`.
#     - `theme_light()` to set the light theme.
#   Save the generated plot into `ny_temp`.
## Do not modify this line!
ny_temp<-
ggplot(temperature_ny,aes(x=datetime,y=temperature))+geom_line()+labs(title="New York
temperature trend from 2012 to 2018",
                                                y="Temperature (C)",x="Date")


# 4. Load package `modelr`. Let's fit the temperature using a polynomial
#   model first.
#    - Create a linear model object using `lm()`:
#     - set the relationship of variables  to be `temperature ~ poly(yday, 5)`
#     - set argument `data` to be `temperature_ny`.
#   Save the generated model object to `model0`.
#    - Use `add_predictions()` and `add_residuals()` to add two
#     new columns to `temperature_ny_warming` representing the prediction
```

```
#     result of `model0` we defined.
#   Save the generated tibble into `temperature_ny_model0`.
#   `temperature_ny_model0` should print to:
#   # A tibble: 44,460 x 8
#   datetime            city     temperature  yday yearfrac hour   pred resid
#   <dttm>              <chr>          <dbl> <dbl>    <dbl> <fct> <dbl> <dbl>
#   1 2012-10-01 13:00:00 New York       15.1   275    0.751 13     17.2 -2.08
#   2 2012-10-01 14:00:00 New York       15.1   275    0.751 14     17.2 -2.06
#   3 2012-10-01 15:00:00 New York       15.2   275    0.751 15     17.2 -1.98
#   4 2012-10-01 16:00:00 New York       15.3   275    0.751 16     17.2 -1.90
#   5 2012-10-01 17:00:00 New York       15.3   275    0.751 17     17.2 -1.82
#   6 2012-10-01 18:00:00 New York       15.4   275    0.751 18     17.2 -1.74
#   7 2012-10-01 19:00:00 New York       15.5   275    0.751 19     17.2 -1.66
#   8 2012-10-01 20:00:00 New York       15.6   275    0.751 20     17.2 -1.58
#   9 2012-10-01 21:00:00 New York       15.7   275    0.751 21     17.2 -1.50
#   10 2012-10-01 22:00:00 New York       15.7   275    0.751 22     17.2 -1.42
#   # … with 44,450 more rows
## Do not modify this line!
library(modelr)
model0<-lm(temperature ~ poly(yday, 5),data=temperature_ny)
temperature_ny_model0<-temperature_ny%>%add_predictions(model0)%>%
add_residuals(model0)

# 5. Visualize our model performance, showing the original data points,
#   the fitted line, the residual and scale the color according to residual.
#   Specifically, you can do the following:
#   First, draw the scatter plot of the original dataset.
#     To do that, you can use:
#       - `ggplot()` to set the variables of interest: `datetime` and
#         `temperature`.
#       - `geom_point()` to plot the original data points. Set arguments
#         `size` to be 0.2 to shrink the points and `alpha` to be 0.2.
#       - `geom_point()` with `aes(y=pred)` to draw the predicted data points.
#         Set arguments `size` to be 1 and `col` to be `"red"`.
#       - `labs()` to add titles and labels. Inside `labs()`, set `title` to
#         `"The seasonality is accounted for"`, `subtitle` to
#         `"But polynomials in red miss the cyclicality (boundary effect)"`,
#         `y` to `Temperature (C)`, and `x` to `"Date"`.
#       - `theme_light()` to set the light theme.
#   Save the generated plot object to `ny_temp_fitted`.
## Do not modify this line!
ny_temp_fitted<-ggplot(temperature_ny_model0,aes(x=datetime,y=temperature))+
  geom_point(size=0.2,alpha=0.2)+geom_line(aes(y=pred),size=1,col="red")+
  labs(title="The seasonality is accounted for",
       subtitle = "But polynomials in red miss the cyclicality (boundary effect)",
       y="Temperature (C)",x="Date")



# 6. We now want to be able to capture the cyclicality.
#   Fit a linear model of `temperature` against the cyclical transformation
#   of `yday`.
```

```
#   First, create the following function :
#   `cyclic <- function(yearfrac) mgcv::cSplineDes(yearfrac, seq(0, 1, 0.2))[, -5]`
#   Then fit linear model on the transformed variables using `lm()` and
#   `cyclic()`, that is regress the `temperature` and `cyclic(yearfrac)`.
#   Save the fitted model in `model1`.
#   Then:
#    - Use `add_predictions()` and `add_residuals()` to add two
#      new columns to `temperature_ny` representing the prediction
#      result of `model1` we defined.
#   Save the generated tibble into `temperature_ny_model1`.
#   `temperature_ny_model1` should print to :
#   # A tibble: 44,460 x 8
#   datetime            city     temperature  yday yearfrac hour   pred resid
#   <dttm>              <chr>          <dbl> <dbl>    <dbl> <fct> <dbl> <dbl>
#   1 2012-10-01 13:00:00 New York       15.1   275    0.751 13     17.3 -2.22
#   2 2012-10-01 14:00:00 New York       15.1   275    0.751 14     17.3 -2.19
#   3 2012-10-01 15:00:00 New York       15.2   275    0.751 15     17.3 -2.11
#   4 2012-10-01 16:00:00 New York       15.3   275    0.751 16     17.3 -2.03
#   5 2012-10-01 17:00:00 New York       15.3   275    0.751 17     17.3 -1.95
#   6 2012-10-01 18:00:00 New York       15.4   275    0.751 18     17.3 -1.87
#   7 2012-10-01 19:00:00 New York       15.5   275    0.751 19     17.3 -1.79
#   8 2012-10-01 20:00:00 New York       15.6   275    0.751 20     17.3 -1.71
#   9 2012-10-01 21:00:00 New York       15.7   275    0.751 21     17.3 -1.64
#   10 2012-10-01 22:00:00 New York      15.7   275    0.751 22     17.3 -1.56
#   # … with 44,450 more rows
## Do not modify this line!
cyclic <- function(yearfrac) mgcv::cSplineDes(yearfrac, seq(0, 1, 0.2))[, -5]
model1<-lm(temperature~cyclic(yearfrac),data=temperature_ny)
temperature_ny_model1<-temperature_ny%>%add_predictions(model1)%>%
add_residuals(model1)


# 7. Now, let's compare the predictions of `model0` and `model1`.
#   To do that, you can :
#     - add a plot `geom_line()` to previous plot of `ny_temp_fitted`.
#       Inside `geom_line()`, set `data` to `temperature_ny_model1`,
#       with `aes(y = pred)` and  `color` to `"green"` and `size` to 1.
#     - `labs()` to set `subtitle` to
#         `"Polynomials in red miss the cyclicality, but cyclic`
#         `splines in green capture it properly"`
#     - `theme_light()` to set a light background.
#   Save the ggplot object into `ny_temp_fitted2`.
## Do not modify this line!
ny_temp_fitted2<-ny_temp_fitted+geom_line(data=temperature_ny_model1,aes(y =
pred),color="green",size=1)+
  labs(subtitle="Polynomials in red miss the cyclicality, but cyclic splines in
       green capture it properly")


# 8. Use boxplot to visualize the residuals over each hour of the day.
#   To do that, you can use:
#     - `ggplot()` to set the `x` axis to be the hour of the time
```

```
#       and `y` axis to be the residual.
#     - `geom_boxplot()` to draw the plot.
#     - `labs()` to set `title` to
#         `"There is a time-of-day effect in the residuals"`,
#         `x` to be `"Hour"` and `y` to be `"Residuals (C)"`.
#    Save the generated plot to `ny_temp_residual`.
## Do not modify this line!
ny_temp_residual<-ggplot(data=temperature_ny_model1,aes(x=hour,y=resid))+
  geom_boxplot()+
  labs(title="There is a time-of-day effect in the residuals",
       x="Hour",y="Residuals (C)")


# 9. We now want to be able to learn the local variations of the temperature
#    by adding `hour` as an intercept in the regression.
#    The goal is to account for the variations between day and night.
#    To do that, declare a new model called `model2` using `lm()` and `cyclic()`.
#    Save the model to `model2`.
#    Combine the prediction result of `model1` and `model2` in the following steps:
#     - use `gather_predictions()` to get the prediction results from both
#       `model1` and `model2`.
#     - use `gather_residuals()` to get the prediction residuals from both
#       `model1` and `model2`.
#     - use `left_join()` to join the above two tibbles.
#     - use `mutate()` and `factor()` to convert `model` column into a factor
#       with `levels()` set to `c("model1", "model2")`.
#    Save the generated tibble into `temperature_ny_model2`.
#    `temperature_ny_model2` should print to:
#    # A tibble: 88,920 x 9
#    model datetime           city  temperature  yday yearfrac hour
#    <fct> <dttm>             <chr>       <dbl> <dbl>    <dbl> <fct>
#    1 mode… 2012-10-01 13:00:00 New …       15.1   275    0.751 13
#    2 mode… 2012-10-01 14:00:00 New …       15.1   275    0.751 14
#    3 mode… 2012-10-01 15:00:00 New …       15.2   275    0.751 15
#    4 mode… 2012-10-01 16:00:00 New …       15.3   275    0.751 16
#    5 mode… 2012-10-01 17:00:00 New …       15.3   275    0.751 17
#    6 mode… 2012-10-01 18:00:00 New …       15.4   275    0.751 18
#    7 mode… 2012-10-01 19:00:00 New …       15.5   275    0.751 19
#    8 mode… 2012-10-01 20:00:00 New …       15.6   275    0.751 20
#    9 mode… 2012-10-01 21:00:00 New …       15.7   275    0.751 21
#    10 mode… 2012-10-01 22:00:00 New …       15.7   275    0.751 22
#    # … with 88,910 more rows, and 2 more variables: pred <dbl>,
#    #   resid <dbl>
## Do not modify this line!
model2<-lm(temperature~cyclic(yearfrac)+hour,data=temperature_ny)
t1<-temperature_ny%>%gather_predictions(model1,model2)
t2<-temperature_ny%>%gather_residuals(model1,model2)
temperature_ny_model2<-left_join(t1,t2)%>%mutate(model=factor(model,levels =
c("model1", "model2")))

# 10. Let's plot the residual comparison of `model1` and `model2`.
#     To do that, you can use :
```

```
#     - `ggplot()` to set the `x` axis to be the hour of the time
#       and `y` axis to be the residual.
#     - `geom_boxplot()` to draw the plot.
#     - `facet_wrap()` to draw individual plot for each model.
#     - `labs()` to set `title` to
#         `"Including the hour captures the time-of-day effect"`,
#         `x` to be `"Hour"` and `y` to be `"Residuals (C)"`.
#     Save the ggplot object into `ny_temp_residual2`.
#     We notice that the splines seem to learn local variations in the data,
#     without overfitting.
## Do not modify this line!
ny_temp_residual2<-ggplot(data=temperature_ny_model2,aes(x=hour,y=resid))+
  geom_boxplot()+facet_wrap(~model)+
  labs(title="Including the hour captures the time-of-day effect",
       x="Hour",y="Residuals (C)"
       )


# 11. Use `anova()` to compare the variance of `model1` and `model2`. Save
#     the result to `compare_models`.
#     `compare_models` should print to :
#     Analysis of Variance Table
#
#     Model 1: temperature ~ cyclic(yearfrac)
#     Model 2: temperature ~ cyclic(yearfrac) + hour
#     Res.Df     RSS Df Sum of Sq      F    Pr(>F)
#     1  44455 1184509
#     2  44432  944015 23    240495 492.15 < 2.2e-16 ***
#       ---
#       Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
## Do not modify this line!
compare_models<-anova(model1,model2)



# 12. Now, we choose `model2` as our best model to fit temperature trend.
#     We want to deploy it to model multiple cities at once.
#     To do that, please do the following:
#       - create a function `my_model` that takes tibble
#         `df` as argument and return a fitted `model2` on the tibble.
#         (use the same formula as `model2`, `temperature ~ cyclic(yearfrac)`).
#         You can assume that `df` has columns `yearfrac` and `temperature`.
#       - Use `temperature_full`, fit `model2` on each city by doing
#         the following:
#         - use `group_by()` and `nest()` to group tibbles according to
#           city.
#         - use `mutate()` and `purrr::map()` to fit the model to each city
#           in `data` using `my_model` function.
#     Save the defined function as `my_model` and the generated tibble as
#     `by_city`. It should print to:
#     # A tibble: 4 x 3
#     # Groups:   city [4]
```

```
#    city     data                                  model
#    <chr>    <S3: vctrs_list_of>                        <list>
#    1 Vancouv… 1.349096e+09, 1.349100e+09, 1.349104e+09, 1.349107… <S3: l…
#    2 Phoenix  1.349096e+09, 1.349100e+09, 1.349104e+09, 1.349107… <S3: l…
#    3 Miami    1.349096e+09, 1.349100e+09, 1.349104e+09, 1.349107… <S3: l…
#    4 New York 1.349096e+09, 1.349100e+09, 1.349104e+09, 1.349107… <S3: l…
## Do not modify this line!
my_model<-function(df){
 lm(temperature~cyclic(yearfrac)+hour,data=df)
}
by_city<-temperature_full%>%group_by(city)%>%nest()%>%
 mutate(model=purrr::map(data,my_model))


# 13. Load package `broom`. Use `broom` to output organized summary of our
#    models.
#    To do that, do the following:
#      - create new column `glance` using `mutate()`, and `map()`
#        to apply `glance()` to `data`.
#      - `unnest()` the column `glance`.
#      - `dplyr::select()` every column except `data` and `model`.
#      - `arrange()` the order the records by `adj.r.squared`.
#    Save the generated tibble into `by_city_stats`.
#    `by_city_stats` should print to:
#    # A tibble: 4 x 12
#    # Groups:   city [4]
#    city  r.squared adj.r.squared sigma statistic p.value    df  logLik
#    <chr>    <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>   <dbl>
#    1 Miami     0.611         0.611  2.61     2588.       0    28 -1.06e5
#    2 New …     0.797         0.797  4.61     6451.       0    28 -1.31e5
#    3 Vanc…     0.798         0.798  2.99     6501.       0    28 -1.12e5
#    4 Phoe…     0.848         0.848  3.87     9337.       0    28 -1.25e5
#    # … with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
#    #   df.residual <int>
## Do not modify this line!
library(broom)
by_city_stats<-by_city%>%mutate(glance=purrr::map(model,glance))%>%
unnest(glance)%>%
 dplyr::select(-data,-model)%>%arrange(adj.r.squared)


# 14. Now, we want to test the model performance in four seasons using simulated data.
#    We will generate our test data, which consists of `yearfrac` ranging from
#    0 to 0.75 with interval 0.25 and hour pulled from `temperature_full`.
#    To do that, you can use:
#      - `crossing` in which we set:
#        - `yearfrac` to be a vector generated from `seq(0, 0.75, 0.25)`.
#        - `hour` using `pull(hour)` on `temperature_full` and then
#          `unique()` to select unique hours.
#    Save the tibble into `test_df`.
#    `test_df` should print to:
#    # A tibble: 96 x 2
```

```
#   yearfrac hour
#   <dbl> <fct>
#   1       0 0
#   2       0 1
#   3       0 2
#   4       0 3
#   5       0 4
#   6       0 5
#   7       0 6
#   8       0 7
#   9       0 8
#   10       0 9
#   # … with 86 more rows
## Do not modify this line!

test_df<-crossing(yearfrac=seq(0, 0.75, 0.25),hour=unique(pull(temperature_full,hour)))

# 15. Apply our model in `by_city` to `test_df`.
#    To do that, please do the following:
#      - use `mutate()` to create column `pred`:
#        - first, use `map()` to apply `predict()` to `model` with
#          argument `newdata` set to `test_df`.
#        - secondly, use `map()` to apply `enframe()` to the `pred` column
#        - thirdly, use `map()` and `bind_cols()` to concatenate the predicted
#          temperature with `test_df`.
#      - use `unnest()` get unnested tibble.
#      - use `dplyr::select()` to get all the columns except `data`, `model`
#        and `name`.
#      - use `mutate()` to formulate the columns:
#        - use `factor` to convert `yearfrac` to factor and `fct_recode()` to
#          rename the factor name as `"Winter"` for `yearfrac` equal to `0`,
#          `"Spring"` for `yearfrac` equal to `0.25`, `"Summer"` for
#          `yearfrac` equal to `0.5` and `"Autumn"` for `yearfrac` equal to
#          `0.75`.
#        - convert `hour` back to a number using `as.numeric()` (don't forget
#          to substract 1).
#   Save the generate tibble into `by_city_pred`.
#   `by_city_pred` should print to:
#   # A tibble: 384 x 4
#   # Groups:   city [4]
#   city      pred yearfrac  hour
#   <chr>    <dbl> <fct>    <dbl>
#   1 Vancouver  5.47 Winter      0
#   2 Vancouver  5.12 Winter      1
#   3 Vancouver  4.56 Winter      2
#   4 Vancouver  4.01 Winter      3
#   5 Vancouver  3.26 Winter      4
#   6 Vancouver  2.58 Winter      5
#   7 Vancouver  2.16 Winter      6
#   8 Vancouver  1.68 Winter      7
#   9 Vancouver  1.36 Winter      8
#   10 Vancouver  1.14 Winter      9
```

```
#   # … with 374 more rows
## Do not modify this line!
by_city_pred<-by_city%>%mutate(pred = purrr::map(model,predict,newdata=test_df),
             pred= purrr::map(pred,enframe),
             pred = purrr::map(pred,bind_cols,test_df))%>%unnest(pred)%>%
 dplyr::select(-data,-model,-name)%>%
 mutate(yearfrac=factor(yearfrac))%>%
 mutate(yearfrac = fct_recode(yearfrac,
                        "Winter"="0","Spring" = "0.25","Summer"="0.5","Autumn"="0.75"))%>%
 mutate(hour=as.numeric(hour)-1)%>%
 rename(pred=value)


# 16. Plot the predicted temperature from `by_city_pred` of the four seasons
#    in city `Miami`, `Phoenix` and `Vancouver`.
#    To do that, you can use:
#     - `filter()` to select the three cities.
#     - `ggplot()` to set the `x` axis to be the hour of the day
#       and `y` axis to be the prediction
#     - `geom_point()` to plot the predicted temperature.
#     - `facet_grid()` to draw plots for each `yearfrac`.
#     - `labs()` to set `title` to
#        `"Our model can predict the temperature for different seasons!"`,
#        `x` to `"Hour"` and `y` to `"Prediction (C)"`.
#   Save the generated tibble into `temperature_per_hour_season_plot`.
#
## Do not modify this line!
temperature_per_hour_season_plot<-by_city_pred%>%filter(city%in%
c("Miami","Phoenix","Vancouver"))%>%
 ggplot(aes(x=hour,y=pred))+geom_point()+facet_grid(city~yearfrac)+
 labs(title = "Our model can predict the temperature for different seasons!",
     x="Hour",y="Prediction (C)")

# HW8: college
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `my_theme` that you will create in the problem 1
#   for all the plots.
#   - Please specify `dplyr::` when you use `select()`.
#
#
# 1. Load the packages `tidyverse`, `modelr` and `broom`.
#   Use `read_csv()` to read the datasets from data folder:
#     - `college.csv` into a tibble `college`.
#   To check your solution, `college` prints to:
#   # A tibble: 777 x 18
#   Private  Apps Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad
#   <chr>   <dbl>  <dbl>  <dbl>    <dbl>    <dbl>      <dbl>      <dbl>
#   1 Yes     1660   1232   721       23       52       2885        537
#   2 Yes     2186   1924   512       16       29       2683       1227
```

```
#   3 Yes     1428   1097    336      22      50      1036       99
#   4 Yes      417    349    137      60      89       510       63
#   5 Yes      193    146     55      16      44       249      869
#   6 Yes      587    479    158      38      62       678       41
#   7 Yes      353    340    103      17      45       416      230
#   8 Yes     1899   1720    489      37      68      1594       32
#   9 Yes     1038    839    227      30      63       973      306
#   10 Yes     582    498    172      21      44       799       78
#   # … with 767 more rows, and 10 more variables: Outstate <dbl>,
#   #   Room.Board <dbl>, Books <dbl>, Personal <dbl>, PhD <dbl>,
#   #   Terminal <dbl>, S.F.Ratio <dbl>, perc.alumni <dbl>, Expend <dbl>,
#   #   Grad.Rate <dbl>
#
#   Create a theme called `my_theme` that combine `theme_light()` and
#   adjust the titles by adding `theme()`
#   with arguments:
#     - `plot.title = element_text(hjust = 0.5)`
#     - `plot.subtitle = element_text(hjust = 0.5))`.
#
## Do not modify this line!
library(tidyverse)
library(modelr)
library(broom)
college<-read_csv("data/college.csv")
my_theme<-theme_light()+theme(plot.title = element_text(hjust = 0.5),
                      plot.subtitle = element_text(hjust = 0.5))

# 2. Create a tibble of dim 777 x 4. Replace all the `.` with `_`.
#   The dataset should sum up `Outstate`, `Room_Board`, `Books`,
#   `Personal` and `Expend` costs, and store the sum into a new
#   column called `total_cost`.
#   Then transform `Private` into a factor variable.
#   Finally, choose the
#   `total_cost`, `Private`, `Top10perc` and `PhD`columns.
#   To do this, you can use:
#     - `rename_all()` and `str_replace_all()` to replace `.` with `_`.
#     (hint1: `rename_all()` has been vectorized so it can
#     work like `map()`.
#     hint2: in order to escape the `.`, you can use `\\.`.)
#     - `nest()` to combine `Outstate`, `Room_Board`, `Books`,
#     `Personal` and `Expend` into one column called `total_cost`.
#     - `mutate()` to create `total_cost` and transform `Private`
#     into factors.
#     (hint: you can use `map_dbl()` and `sum()` to calculate the
#     total cost of each row.)
#     - `dplyr::select()` to choose the required columns.
#   Store the returned dataset into `df`.
#   To check your solution, `df` prints to:
#   # A tibble: 777 x 4
#   total_cost Private Top10perc   PhD
#   <dbl> <fct>       <dbl> <dbl>
#   1      20431 Yes          23    70
```

```
#   2     31507 Yes          16    29
#   3     25300 Yes          22    53
#   4     38751 Yes          60    92
#   5     24902 Yes          16    76
#   6     27737 Yes          38    67
#   7     29871 Yes          17    90
#   8     31481 Yes          37    89
#   9     32439 Yes          30    79
#   10     25299 Yes          21    40
#   # … with 767 more rows
## Do not modify this line!
df<-college%>%rename_all(~str_replace_all(.,"\\.","_"))%>%
  nest(total_cost=c(Outstate,Room_Board,Books,Personal,Expend))%>%
  mutate(Private = factor(Private),
         total_cost = map_dbl(total_cost,sum))%>%
  dplyr::select(total_cost,Private,Top10perc,PhD)

# 3. Draw a point plot of `total_cost` vs. `Top10perc`.
#   Name the title as `"Total cost seems to have linear relationship with percent of students`
#   `from top 10 percent high schools"`,
#   To do this, you can use:
#     - `geom_point()` to draw a point plot of
#     `total_cost` vs. `Top10perc`.
#     - `geom_smooth()` to draw a straight line with `lm` method,
#     and turn of the `se`.
#     - `labs()` to name the title as
#       `"Total cost seems to have linear relationship with percent of students`
#       `from top 10 percent high schools"`
#       the x-axis as `"Pct. of new students from top 10% H.S."`,
#       the y-axis as `"Total cost (USD)"`.
#   Store the plot into a variable `g1`.
## Do not modify this line!
g1<-ggplot(df,aes(y=total_cost,x=Top10perc))+geom_point()+geom_smooth(method =
"lm",se =F)+
  labs(title = "Total cost seems to have linear relationship with percent of students
  from top 10 percent high schools",
       x="Pct. of new students from top 10% H.S.",
       y="Total cost (USD)")+my_theme


# 4. Fit a linear regression model of `total_cost` vs. `Top10perc`.
#   Store the model to `m0`.
#   The expected output for `m0 %>% broom::tidy()` should be:
#   # A tibble: 2 x 5
#   term        estimate std.error statistic   p.value
#   <chr>          <dbl>    <dbl>    <dbl>     <dbl>
#   1 (Intercept)   16854.     450.      37.4 7.22e-176
#   2 Top10perc       345.     13.8      25.0 9.21e-102
#   Calculate residuals and predictions of `m0`, name them `resid0`
#   and `pred0` accordingly, and add two new columns to back of `df`.
#   To do this, you can use:
#     - `lm()` to build the model.
```

```
#     - `add_residuals()` and `add_predictions()` to add the
#     calculated columns.
#   Store returned dataset to `total_cost_pred`.
#   To check your solution, `total_cost_pred` prints to:
#   # A tibble: 777 x 6
#   total_cost Private Top10perc   PhD resid0  pred0
#   <dbl> <fct>      <dbl> <dbl>  <dbl>  <dbl>
#   1     20431 Yes          23    70 -4347. 24778.
#   2     31507 Yes          16    29  9141. 22366.
#   3     25300 Yes          22    53   867. 24433.
#   4     38751 Yes          60    92  1226. 37525.
#   5     24902 Yes          16    76  2536. 22366.
#   6     27737 Yes          38    67 -2209. 29946.
#   7     29871 Yes          17    90  7160. 22711.
#   8     31481 Yes          37    89  1880. 29601.
#   9     32439 Yes          30    79  5250. 27189.
#   10     25299 Yes          21    40  1210. 24089.
#   # … with 767 more rows
## Do not modify this line!
m0<-lm(total_cost~Top10perc,df)
total_cost_pred<-df%>%add_residuals(m0,"resid0")%>%add_predictions(m0,"pred0")

# 5. Draw a point plot of `resid0` vs. `pred0`.
#   Name the title as `"The residuals have a fanning pattern that spread to the right"`
#   To do this, you can use:
#     - `geom_ref_line` to draw reference line with height = 0.
#     - `geom_point()` to draw a point plot of
#     `resid0` vs. `pred0`.
#     - `labs()` to name the title as
#       `"The residuals have a fanning pattern that spread to the right"`,
#       subtitle as `"The data seems to display residual heteroskedasticity"`,
#       the x-axis as `"Prediction of total cost (USD)"`,
#       the y-axis as `"Residuals of m0"`.
#   Store the plot into a variable `g2`.
## Do not modify this line!
g2<-ggplot(total_cost_pred,aes(x=pred0,y=resid0))+geom_ref_line(h=0)+geom_point()+
  labs(title = "The residuals have a fanning pattern that spread to the right",
       subtitle = "The data seems to display residual heteroskedasticity",
       x="Prediction of total cost (USD)",
       y = "Residuals of m0")+my_theme


# 6. Create a new column called `log_total_cost` that takes the log
#   of `total` cost in `df`. You can use `mutate` to finish the work.
#   To check your solution, `df` prints to:
#   # A tibble: 777 x 5
#   total_cost Private Top10perc   PhD log_total_cost
#   <dbl> <fct>      <dbl> <dbl>         <dbl>
#   1     20431 Yes          23    70          9.92
#   2     31507 Yes          16    29         10.4
#   3     25300 Yes          22    53         10.1
#   4     38751 Yes          60    92         10.6
```

```
#   5     24902 Yes          16    76        10.1
#   6     27737 Yes          38    67        10.2
#   7     29871 Yes          17    90        10.3
#   8     31481 Yes          37    89        10.4
#   9     32439 Yes          30    79        10.4
#   10     25299 Yes          21    40        10.1
#   # … with 767 more rows
#
#   Create a new column called `log_total_cost` that takes the log
#   of `total` cost in `total_cost_pred`.
#   You can use `mutate` to finish the work.
#   To check your solution, `total_cost_pred` prints to:
#   # A tibble: 777 x 7
#   total_cost Private Top10perc   PhD resid0  pred0 log_total_cost
#   <dbl> <fct>      <dbl> <dbl>  <dbl>  <dbl>         <dbl>
#   1     20431 Yes          23    70 -4347. 24778.         9.92
#   2     31507 Yes          16    29  9141. 22366.        10.4
#   3     25300 Yes          22    53   867. 24433.        10.1
#   4     38751 Yes          60    92  1226. 37525.        10.6
#   5     24902 Yes          16    76  2536. 22366.        10.1
#   6     27737 Yes          38    67 -2209. 29946.        10.2
#   7     29871 Yes          17    90  7160. 22711.        10.3
#   8     31481 Yes          37    89  1880. 29601.        10.4
#   9     32439 Yes          30    79  5250. 27189.        10.4
#   10     25299 Yes          21    40  1210. 24089.        10.1
#   # … with 767 more rows
## Do not modify this line!
df<-df%>%mutate(log_total_cost=log(total_cost))
total_cost_pred<-total_cost_pred%>%mutate(log_total_cost=log(total_cost))

# 7. Here, we want to try if `log_total_cost` vs. `Top10perc` fits better.
#   Fit a linear regression model of `log_total_cost` vs. `Top10perc`.
#   Store the model to `m1`.
#   The expected output for `m1 %>% broom::tidy()` should be:
#   # A tibble: 2 x 5
#   term        estimate std.error statistic  p.value
#   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#   1 (Intercept)   9.82    0.0162      607.  0.
#   2 Top10perc     0.0111  0.000495     22.5 1.08e-86
#
#   Calculate residuals and predictions of `m1`, name them `resid1`
#   and `pred1` accordingly, and add two new columns to
#   back of `total_cost_pred`.
#   To do this, you can use:
#     - `lm()` to build the model.
#     - `add_residuals()` and `add_predictions()` to add the
#     calculated columns.
#   Store returned dataset to `total_cost_pred`.
#   To check your solution, `total_cost_pred` is a tibble of 777 x 9.
#   The first 10 rows of `resid1` and `pred1` print to:
#   resid1 pred1
#   <dbl> <dbl>
```

```
#    1 -0.153  10.1
#    2  0.358  10.00
#    3  0.0723 10.1
#    4  0.0757 10.5
#    5  0.123  10.00
#    6 -0.0138 10.2
#    7  0.294  10.0
#    8  0.124  10.2
#    9  0.232  10.2
#    10  0.0834 10.1
## Do not modify this line!
m1<-lm(log_total_cost~Top10perc,total_cost_pred)
total_cost_pred<-total_cost_pred%>%add_residuals(m1,"resid1")%>%
add_predictions(m1,"pred1")



# 8. Draw a point plot of `log_total_cost` vs. `Top10perc`.
#    Name the title as `"After taking log of total cost, the log of total cost has a`
#    `more linear relationship with the students from top 10% H.S."`,
#    To do this, you can use:
#      - `geom_point()` to draw a point plot of
#      `log_total_cost` vs. `Top10perc`.
#      - `geom_line()` to draw a line of `pred1` vs. `Top10perc`, make the
#      color as `"red"`.
#      - `labs()` to name the title as
#        `"After taking log of total cost, the log of total cost has better`
#        `linear relationship with top 10 percent colleges"`
#        the x-axis as `"Pct. of new students from top 10% H.S."`,
#        the y-axis as `"Log of total cost (log (USD))"`.
#    Store the plot into a variable `g3`.
## Do not modify this line!
g3<-ggplot(total_cost_pred,aes(x = Top10perc,y=log_total_cost))+
  geom_point()+geom_line(aes(y=pred1),color = "red")+
  labs(title = "After taking log of total cost, the log of total cost has a
  more linear relationship with top 10 percent colleges",
       x = "Pct. of new students from top 10% H.S.",
       y = "Log of total cost (log (USD))")+my_theme


# 9. Draw a point plot of `resid1` vs. `Top10perc`.
#    Name the title as title as `"The residuals don't have obvious pattern"`.
#    To do this, you can use:
#      - `geom_ref_line` to draw reference line with height = 0.
#      - `geom_point()` to draw a point plot of
#      `resid1` vs. `Top10perc`.
#      - `labs()` to name the title as
#        `"The residuals don't have obvious pattern"`,
#        subtitle as `"The Top10perc is a good main effect"`,
#        the x-axis as `"Pct. of new students from top 10% H.S."`,
#        the y-axis as `"Residuals of m1"`.
#    Store the plot into a variable `g4`.
```

```
## Do not modify this line!
g4<-ggplot(total_cost_pred,aes(x = Top10perc,y=resid1))+
  geom_ref_line(h=0)+ geom_point()+
  labs(title = "The residuals don't have obvious pattern",
       subtitle = "The Top10perc is a good main effect",
       x = "Pct. of new students from top 10% H.S.",
       y = "Residuals of m1")+my_theme
#geom的顺序有影响

# 10. Draw a boxplot of `log_total_cost` vs. `Private`.
#     Name the title as `"Whether a college is private or not can help to differentiate`
#     `the total cost"`
#     To do this, you can use:
#       - `geom_boxplot()` to draw a boxplot of
#       `log_total_cost` vs. `Private`.
#       - `labs()` to name the title as
#         `"Whether a college is private or not can help to differentiate`
#         `the total cost"`
#         the x-axis as `"Private"`,
#         the y-axis as `"Log of total cost (log (USD))"`.
#     Store the plot into a variable `g5`
## Do not modify this line!
g5<-ggplot(total_cost_pred,aes(x = Private,y=log_total_cost))+
  geom_boxplot()+
  labs(title = "Whether a college is private or not can help to differentiate
  the total cost",
       x = "Private",
       y = "Log of total cost (log (USD))")+my_theme


# 11. Fit a linear regression model of `log_total_cost` vs. interaction
#     of `Top10perc` and `Private`. Store the model to `m2`.
#     The expected output for `m2 %>% broom::tidy()` should be:
#     # A tibble: 4 x 5
#     term              estimate std.error statistic  p.value
#     <chr>                <dbl>    <dbl>    <dbl>    <dbl>
#     1 (Intercept)         9.72     0.0252     385.   0.
#     2 PrivateYes          0.174    0.0305       5.69 1.82e- 8
#     3 Top10perc           0.00720  0.000902     7.98 5.12e-15
#     4 PrivateYes:Top10perc  0.00370  0.00103      3.59 3.52e- 4
#
#     Calculate residuals and predictions of `m2`, name them `resid2`
#     and `pred2` accordingly, and add two new columns to
#     back of `total_cost_pred`.
#     To do this, you can use:
#       - `lm()` to build the model.
#       - `add_residuals()` and `add_predictions()` to add the
#       calculated columns.
#     Store returned dataset to `total_cost_pred`.
#     To check your solution, `total_cost_pred` is a tibble of 777 x 11.
#     The first 10 rows of `resid2` and `pred2` print to:
#     resid2 pred2
```

```
#    <dbl> <dbl>
#    1 -0.224     10.1
#    2  0.285     10.1
#    3  0.000591  10.1
#    4  0.0125    10.6
#    5  0.0502    10.1
#    6 -0.0819    10.3
#    7  0.221     10.1
#    8  0.0556    10.3
#    9  0.162     10.2
#    10  0.0115    10.1
## Do not modify this line!
m2<-lm(log_total_cost~Private*Top10perc,total_cost_pred)
total_cost_pred<-total_cost_pred%>%add_residuals(m2,"resid2")%>%
add_predictions(m2,"pred2")

# 12. Draw a point plot of `resid2` vs. `pred2`.
#    Name the title as `"There's no obvious pattern of residuals"`
#    To do this, you can use:
#      - `geom_ref_line` to draw reference line with height = 0.
#      - `geom_point()` to draw a point plot of
#      `resid2` vs. `pred2`.
#      - `labs()` to name the title as
#        `"There's no obvious pattern of residuals"`,
#        names the subtitle as
#        `"The interaction of Private and Top10perc is a good main effect"`,
#        the x-axis as `"Prediction of log of total cost (log (USD))"`,
#        the y-axis as `"Residuals of m2"`.
#    Store the plot into a variable `g6`.
## Do not modify this line!
theme_set(my_theme)
g6<-ggplot(total_cost_pred,aes(x=pred2,y=resid2))+
  geom_ref_line(h=0)+geom_point()+
  labs(title = "There's no obvious pattern of residuals",
       subtitle = "The interaction of Private and Top10perc is a good main effect",
       x = "Prediction of log of total cost (log (USD))",
       y = "Residuals of m2")


# 13. Select `log_total_cost`, `pred2`, `Private` and `Top10perc` from
#    `total_cost_pred`. Rename `log_total_cost` as `Data`, `pred2` as
#    `Prediction`. Tidying the dataset by combining `Data` and
#    `Prediction` into one column.
#    To do this, you can use:
#      - `dplyr::select()` to choose required columns.
#      - `rename()` to rename the columns.
#      - `pivot_longer()` to tidy the dataset. Make names as `"Type"`
#      and values as `"log_total_cost"`.
#    Store returned dataset to `stats_m2`.
#    To check your solution, `stats_m2` prints to:
#    # A tibble: 1,554 x 4
#    Private Top10perc Type       log_total_cost
```

```
#    <fct>      <dbl> <chr>           <dbl>
#    1 Yes         23 Data             9.92
#    2 Yes         23 Prediction       10.1
#    3 Yes         16 Data             10.4
#    4 Yes         16 Prediction       10.1
#    5 Yes         22 Data             10.1
#    6 Yes         22 Prediction       10.1
#    7 Yes         60 Data             10.6
#    8 Yes         60 Prediction       10.6
#    9 Yes         16 Data             10.1
#    10 Yes          16 Prediction       10.1
#    # … with 1,544 more rows
## Do not modify this line!
stats_m2<-total_cost_pred%>%dplyr::select(log_total_cost,pred2,Private,Top10perc)%>%
  rename(Data = log_total_cost, Prediction = pred2) %>%
  pivot_longer(c(Data,Prediction),names_to = "Type",values_to = "log_total_cost")


# 14. Draw a point plot of `log_total_cost` vs. `Top10perc`, faceted
#    on `Private`.
#    Name the title as `"The m2 model seems to fit well on both public colleges and private
colleges"`
#    To do this, you can use:
#      - `geom_point()` to draw a boxplot of
#      `log_total_cost` vs. `Top10perc`.
#      - `facet_grid()` to facet on `Private`.
#      - `geom_smooth()` to draw smooth curves with linetype as
#      `Type`. Turn of the `se`.
#      - `labs()` to name the title as
#        `"Whether a college is private or not can help to differentiate`
#        `the total cost"`
#        the x-axis as `"Pct. of new students from top 10% H.S."`,
#        the y-axis as `"Log of total cost (log (USD))"`.
#    Store the plot into a variable `g7`
## Do not modify this line!
g7<-ggplot(stats_m2,aes(y = log_total_cost, x = Top10perc)) +
  geom_point()+
  facet_grid(~Private)+geom_smooth(aes(linetype = Type),se = F)+
  labs(title="The m2 model seems to fit well on both public colleges and private colleges",x =
"Pct. of new students from top 10% H.S.", y = "Log of total cost (log (USD))")



# 15. Draw a point plot of `resid2` vs. square of `PhD`.
#    Name the title as `"The log of total cost has better`
#    `linear relationship with percent of faculty with PhD"`,
#    To do this, you can use:
#      - `geom_point()` to draw a point plot of
#      `log_total_cost` vs. square of `PhD`.
#      - `geom_smooth` to draw a straight line with `lm` method,
#      and turn of the `se`.
#      - `labs()` to name the title as
```

```
#       `"The residuals of m2 seem to have`
#       `linear relationship with squre of pct. of faculty with PhD"`,
#       subtitle as `"Squre of pct. of faculty with PhD is a good predictor after removing the
main effect"`,
#       the x-axis as `"Square of pct. of faculty with PhD"`,
#       the y-axis as `"Residuals of m2"`.
#    Store the plot into a variable `g8`.
## Do not modify this line!
g8<-ggplot(total_cost_pred,aes(x = PhD^2,y = resid2))+geom_point() +
geom_smooth(method = "lm",se=F) +
  labs(title="The log of total cost has better
linear relationship with percent of faculty with PhD",
     subtitle="Squre of pct. of faculty with PhD is a good predictor after removing the main
effect",
     x="Square of pct. of faculty with PhD",
     y="Residuals of m2")



# 16. Fit a linear regression model of `log_total_cost` vs. interaction
#    of `Top10perc` and `Private` plus square of `PhD`.
#    Store the model to `m3`.
#    The expected output for `m3 %>% broom::tidy()` should be:
#    # A tibble: 6 x 5
#    term              estimate std.error statistic  p.value
#    <chr>                <dbl>    <dbl>    <dbl>    <dbl>
#    1 (Intercept)         9.78     0.0222      440.   0.
#    2 PrivateYes          0.270    0.0276       9.79 2.16e-21
#    3 Top10perc           0.00376  0.000812     4.63 4.33e- 6
#    4 poly(PhD, 2)1       3.12     0.232       13.4  4.85e-37
#    5 poly(PhD, 2)2       2.07     0.193       10.8  2.75e-25
#    6 PrivateYes:Top10perc  0.00210  0.000904     2.33 2.02e- 2
#
#    Calculate residuals and predictions of `m3`, name them `resid3`
#    and `pred3` accordingly, and add two new columns to
#    back of `total_cost_pred`.
#    To do this, you can use:
#      - `lm()` to build the model.
#      - `add_residuals()` and `add_predictions()` to add the
#      calculated columns.
#    Store returned dataset to `total_cost_pred`.
#    To check your solution, `total_cost_pred` is a tibble of 777 x 13.
#    The first 10 rows of `resid3` and `pred3` print to:
#    resid3 pred3
#    <dbl> <dbl>
#    1 -0.187    10.1
#    2  0.292    10.1
#    3  0.116    10.0
#    4 -0.0423   10.6
#    5 -0.00478  10.1
#    6  0.0541   10.2
#    7 -0.0173   10.3
```

```
#    8 -0.0659   10.4
#    9  0.144    10.2
#    10  0.108    10.0
## Do not modify this line!
m3<-lm(log_total_cost~Private*Top10perc+poly(PhD,2),total_cost_pred)
total_cost_pred<-total_cost_pred%>%add_residuals(m3,"resid3")%>%
add_predictions(m3,"pred3")
#这里写作squre,但lm中需要poly
# 17. Draw a point plot of `resid3` vs. `pred3`.
#    Name the title as `"There's no obvious pattern of in the residuals"`
#    To do this, you can use:
#      - `geom_ref_line` to draw reference line with height = 0.
#      - `geom_point()` to draw a point plot of
#      `resid3` vs. `PhD^2`.
#      - `labs()` to name the title as
#        `"There's no obvious pattern of in the residuals"`,
#        names the subtitle as
#        `"The m3 fits well"`,
#        the x-axis as `"Prediction of log of total cost (log (USD))"`,
#        the y-axis as `"Residuals of m3"`.
#    Store the plot into a variable `g9`.
## Do not modify this line!
g9<-ggplot(total_cost_pred,aes(x=pred3,y=resid3))+
  geom_ref_line(h=0)+
  geom_point()+
  labs(title="There's no obvious pattern of in the residuals",
       subtitle = "The m3 fits well",
       x="Prediction of log of total cost (log (USD))",
       y="Residuals of m3")


# 18. Create a list of `m1`, `m2` and `m3`. Store the list to `model_list`.
#    Calculate R-squared and adjusted r-squared of each model. Finally,
#    tidy the dataset.
#    To do this, you can use:
#      - `map_dfr()` and `glance()` to get the summary of each model.
#      - `dplyr::select()` to get R-squared and adjusted r-squared.
#      - `mutate()` to create a new column called `group` and assign
#      `c('m1', 'm2', 'm3')` to the column.
#      - `pivot_longer()` to combine r-squared and adjusted r-squared
#      values into the same column. Set the names as `Type`, values as
#      `Value`.
#    Store the returned dataset to `r_squared_and_adjusted`.
#    To check your solution, `r_squared_and_adjusted` prints to:
#    # A tibble: 6 x 3
#    group Type         Value
#    <chr> <chr>        <dbl>
#    1 m1    adj.r.squared 0.394
#    2 m1    r.squared     0.395
#    3 m2    adj.r.squared 0.539
#    4 m2    r.squared     0.541
#    5 m3    adj.r.squared 0.653
```

```
#    6 m3    r.squared     0.655
## Do not modify this line!
model_list<-list(m1,m2,m3)
r_squared_and_adjusted<-map_dfr(model_list,glance)%>%
dplyr::select(adj.r.squared,r.squared)%>%mutate(group = c('m1', 'm2', 'm3') )%>%
  pivot_longer(c(adj.r.squared,r.squared),names_to = "Type" , values_to = "Value")

# 19. Draw a barplot of `Value` vs. `group`, colored by `Type`.
#    Name the title as `"R-squared and adjusted r-squared both increase as models`
#    `becoming more complex"`
#    To do this, you can use:
#      - `geom_col()` to draw a barplot of
#      `Value` vs. `group`, colored by `Type`,
#      with position as `dodge`.
#      - `labs()` to name the title as
#        `"R-squared and adjusted r-squared both increase as models`
#        `becoming more complex"`,
#        the x-axis as `"Model"`,
#        the y-axis as `"Value"`.
#    Store the plot into a variable `g10`.
## Do not modify this line!
g10<-ggplot(r_squared_and_adjusted,aes(x=group,y=Value))+
  geom_col(aes(fill=Type),position = "dodge")+labs(title="R-squared and adjusted r-squared
both increase as models
 becoming more complex",x="Model",y="Value")



# 20. Fit a linear regression model of `total_cost` vs. `1`.
#    Store the model to `mnull`.
#    The expected output for `mnull %>% broom::tidy()` should be:
#    # A tibble: 1 x 5
#    term        estimate std.error statistic p.value
#    <chr>          <dbl>    <dbl>    <dbl>   <dbl>
#    1 (Intercept)     10.1    0.0112      904.       0
#
#    Use `anova()` to test anova of `mnull`, `m1`, `m2` and `m3`.
#    Store the returned values to `anova_models`.
#    To help make sure that you are correct, check that the
#    output of `anova_models %>% broom::tidy()` is
#    # A tibble: 4 x 6
#    res.df   rss    df sumsq statistic    p.value
#    <dbl> <dbl> <dbl> <dbl>     <dbl>      <dbl>
#    776  75.7    NA NA         NA  NA
#    775  45.8     1 29.9      884.  4.94e-130
#    773  34.8     2 11.0      163.  1.00e- 59
#    771  26.1     2  8.67     128.  8.73e- 49
## Do not modify this line!
mnull<-lm(log_total_cost~1,total_cost_pred)
anova_models<-anova(mnull,m1,m2,m3)
```

```
# HW8: Credit
#
# In this exercise, you will analyze a credit data set with
# a person's basic information and his/her balance. The goal is to predict
# the person's balance by using the other variables.
#
# 1. Let's first read in the required datasets.
#    Load the `readr` package.
#      - Use `read_csv()` to load the `credit.csv` data set from the `data`
#        folder and assign it to a tibble `credit`.
## Do not modify this line!
library(readr)
credit<-read_csv("data/credit.csv")


# 2. Change `Student`, `Married` and `Ethnicity` to factor variables instead of
#    characters. Further, add a new variable `Limit_per_card` which equals the
#    value of `Limit` over `Cards` and sort the tibble by `Balance` in
#    ascending order. To do that, you need to :
#      - Load the `dplyr` package.
#    You can use:
#      - `mutate()` and `factor()` to change the three columns into factor.
#      - `mutate()` to add the new variable.
#      - `arrange()` to sort the tibble.
#    Save your new tibble into `credit_new` whose first few rows should look
#    like:
#    # A tibble: 400 x 12
#    Income Limit Rating Cards   Age Education Gender Student Married Ethnicity
#    <dbl> <dbl>  <dbl> <dbl> <dbl>     <dbl> <fct>  <chr>   <fct>   <fct>
#    15.0  1311    138     3    64        16 Male   No      No      Caucasian
#    20.1  2525    200     3    57        15 Female No      Yes     African …
#    53.6  3714    286     3    73        17 Female No      Yes     African …
#    20.1  2631    213     3    61        10 Male   No      Yes     African …
#    10.7  1757    156     3    57        15 Female No      No      Caucasian
#    28.9  2733    210     5    43        16 Male   No      Yes     Asian
#    31.4  1829    162     4    30        10 Male   No      Yes     Caucasian
#    20.2  2646    199     2    25        14 Female No      Yes     Asian
#    44.5  2252    205     6    72        15 Male   No      Yes     Asian
#    15.3  1499    138     2    47         9 Female No      Yes     Asian
#    … with 390 more rows, and 2 more variables: Balance <dbl>,
#    Limit_per_card <dbl>
#
#    We want to perform a linear regression on `Balance` for people. Let's
#    first try to find some patterns with visualization.
## Do not modify this line!
library(dplyr)
credit_new<-credit%>%
mutate(Gender=factor(Gender),Ethnicity=factor(Ethnicity),Married=factor(Married),
            Limit_per_card = Limit/Cards)%>%
            arrange(Balance)
```

```
# 3. Create a facet for different `Ethnicity` group, within each plot,
#    draw a boxplot of `Balance` aginst `Student`. To do that, you need to :
#     - Load the `ggplot2` package.
#    You can use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`.
#     - `geom_boxplot()` to plot a boxplot for `Balance` against `Student`
#     - `facet_wrap()` to create a facet for `Ethnicity`
#     - `labs()` to format the labels such that:
#       - `title = "Students tend to have more balance than nonstudents"`.
#       - `subtitle = "This difference is more obvious in African American group"`.
#       - `x = "Student"`.
#       - `y = "Balance (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#    Save your plot to `g1`.
## Do not modify this line!

library(ggplot2)
my_theme<-theme_light()+
  theme(plot.title=element_text(hjust = 0.5),
        plot.subtitle=element_text(hjust = 0.5))
theme_set(my_theme)

g1<-ggplot(credit_new,aes(x=Student,y=Balance))+geom_boxplot()+
  facet_wrap(~Ethnicity)+
    labs(
      title = "Students tend to have more balance than nonstudents",
      subtitle = "This difference is more obvious in African American group",
      x = "Student",
      y = "Balance (USD)"
    )



# 4. Create a plot for `Balance` against `Limit_per_card` using color split by
#    `Student`. Describe their relationship.
#    To do that, you can use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`. Don't forget to specify
#       `color = Student`.
#     - `geom_point()` to do a scatter plot for `Balance`
#        against `Limit_per_card`.
#     - `geom_smooth()` to add the regression line.
#     - `labs()` to format the labels such that:
#       - `title = "The balance goes higher as per card limit goes up"`.
#       - `subtitle = "This relationship is nonlinear"`.
#       - `x = "Limit per card (USD)"`.
#       - `y = "Balance (USD)"`.
#     - `theme_light()` to change the theme of plots.
```

```
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#    Save your plot to `g2`.
## Do not modify this line!
g2<-ggplot(data = credit_new, aes(x=Limit_per_card,y = Balance,color=Student))+
  geom_point()+geom_smooth()+
  labs(title = "The balance goes higher as per card limit goes up",
       subtitle = "This relationship is nonlinear",
       x = "Limit per card (USD)",
       y = "Balance (USD)")


# 5. We can see the relationship between `Balance` and `Limit_per_card` is
#    nonlinear, but we are trying to fit a linear model, so let's try some
#    transformation on the variable.
#    Create a new variable `log_limit_per_card` in `credit_new`, which equals
#    the logarithm of `Limit_per_card`.
#    Also remove all the observations if the balance is less or equal to 0,
#    because those mess with our regression.
#    To do that, you can use:
#     - `mutate()` and `log()` to create the new variable.
#     - `filter()` to filter out the observations with balance less
#       or equal to 0
#    Then plot `Blance` against `log_limit_per_card` now, still split by
#    `Student`. What is your finding now?
#    To do that, you can use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`. Don't forget to specify
#       `color = Student`.
#     - `geom_point()` to do a scatter plot for `Balance`
#       against `log_limit_per_card`.
#     - `geom_smooth()` to add the regression line.
#     - `labs()` to format the labels such that:
#       - `title = "The balance goes higher as log of per card limit goes up"`.
#       - `subtitle = "This relationship looks more linear after transformation"`.
#       - `x = "Log of Limit per card (USD)"`.
#       - `y = "Balance (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#    Save your plot to `g3`.
## Do not modify this line!
credit_new<-credit_new%>%mutate(log_limit_per_card = log(Limit_per_card))%>%
  filter(Balance>0)
g3<-ggplot(data=credit_new,mapping = aes(x=log_limit_per_card,y=Balance,color =
Student))+geom_point()+geom_smooth()+
    labs(title = "The balance goes higher as log of per card limit goes up",
         subtitle = "This relationship looks more linear after transformation",
         x = "Log of Limit per card (USD)",
         y = "Balance (USD)")
```

```
# 6. Now let's fit a simple linear regression model `m1` on `Balance` use
#   `log_limit_per_card`, `Student` and their second order iteration.
#   Based on `credit_new`, save your prediction residuals into a column
#   `resid1`, save your predictions into a column `pred1` and save this new
#   tibble into `credit_pred`.
#   To do this, you need to:
#    - Load `modelr` package
#   You can use:
#    - `lm()` to fit a linear regression, save the model to `m1`.
#    - `add_residuals` to add residuals to the new tibble `credit_pred`.
#    - `add_predictions` to add predictions to the new tibble `credit_pred`.
#   The expected output for `m1 %>% broom::tidy()` should be:
#   # A tibble: 4 x 5
#   term                    estimate std.error statistic  p.value
#   <chr>                      <dbl>    <dbl>    <dbl>    <dbl>
#   (Intercept)               -1223.     268.     -4.56 7.49e- 6
#   log_limit_per_card          245.      35.2     6.96 2.09e-11
#   StudentYes                -1113.     665.     -1.67 9.51e- 2
#   log_limit_per_card:StudentYes   192.     89.2     2.16 3.17e- 2
#   The first few lines for `credit_pred` looks like:
#   # A tibble: 310 x 15
#   Income Limit Rating Cards   Age Education Gender Student Married Ethnicity
#   <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl> <fct>  <chr>   <fct>   <fct>
#   16.3  1160    126    3    78       13 Male   Yes     Yes     African …
#   44.5  3500    257    3    81       16 Female No      No      African …
#   88.8  4952    360    4    86       16 Female No      Yes     Caucasian
#   20.9  1233    128    3    47       18 Female Yes     Yes     Asian
#   33.0  3180    224    2    28       16 Male   No      Yes     African …
#   34.5  3271    250    3    57       17 Female No      Yes     Asian
#   35.0  3327    253    3    54       14 Female No      No      African …
#   15.5  2762    215    3    60       18 Male   No      No      Asian
#   26.5  2910    236    6    58       19 Female No      Yes     Caucasian
#   17.4  2748    228    3    32       14 Male   No      Yes     Caucasian
#   … with 300 more rows, and 5 more variables: Balance <dbl>,
#   Limit_per_card <dbl>, log_limit_per_card <dbl>, resid1 <dbl>, pred1 <dbl>
## Do not modify this line!
library(modelr)
m1<-lm(Balance~log_limit_per_card*Student,credit_new)
credit_pred<-credit_new%>%add_residuals(model=m1,var="resid1")%>%
add_predictions(model=m1,var="pred1")

# 7. Let's check if our model captures the relationship we observe in `g3`.
#   To do this, you need to:
#    - Load `tidyr` package.
#   You can start from `credt_pred` and use:
#    - `rename` to change the column name of `Balance` to `Data`,
#      `pred1` to `Predictions`
#    - `pivot_longer` to make the true values `Data` and predicted values
#      in `Predictions` into one column named `Balance` and use `Type` to
#      characterize if it is true or predicted.
```

```
#      - set `names_to = "Type`
#      - set `values_to = "Balance"`
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`. Don't forget to specify
#       `color = Student`.
#     - `geom_point()` to do a scatter plot for `Balance`
#       against `log_limit_per_card`.
#     - `geom_smooth()` to add the regression line for each Student group
#       and for both true and predicted values
#       - set `linetype = Type` in `aes()`
#       - specify `se = FALSE`
#     - `labs()` to format the labels such that:
#       - `title = "In general the model captures the pattern between
#                 Balance and log of limit per card"`.
#       - `subtitle = "The fit is not ideal for nonstudent group"`.
#       - `Log of Limit per card (USD)`.
#       - `y = "Balance (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#   Save your plot to `g32`.
## Do not modify this line!
library(tidyr)
g32<-credit_pred%>%rename(Data=Balance,Predictions=pred1)%>%
pivot_longer(c(Data,Predictions),names_to = "Type",values_to = "Balance")%>%
ggplot(aes(x=log_limit_per_card,y=Balance,color =
Student))+geom_smooth(aes(linetype=Type),se=F)+geom_point(data=credit_pred,aes(log_li
mit_per_card,Balance))+
 labs(title = "In general the model captures the pattern between
     Balance and log of limit per card",
     subtitle = "The fit is not ideal for nonstudent group",
     x="Log of Limit per card (USD)",
     y="Balance (USD)")
#注意顺序和aes的改变也会导致结果不通过
# 8. Now we want to visualize the distribution of residuals to check our model
#   performance.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`.
#     - `geom_ref_line()` to add a reference line.
#       - specify `h=0` to use the x-axis as the reference line.
#     - `geom_point()` to add the scatter plot of residuals.
#     - `labs()` to format the labels such that:
#       - `title = "The residuals goes up as balance goes up"`.
#       - `subtitle = "The range of residuals is pretty wide"`.
#       - `x = "Balance (USD)"`.
#       - `y = "Residuals (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
```

```
#    Save your plot to `g4`.
## Do not modify this line!
g4<-ggplot(data = credit_pred, mapping = aes(x = Balance, y = resid1))+geom_ref_line(h=0)+
  geom_point()+labs(
    title = "The residuals goes up as balance goes up",
    subtitle = "The range of residuals is pretty wide",
    x = "Balance (USD)",
    y = "Residuals (USD)"
  )


# 9. Now let's visualize the predictions and true balance against other
#    variables(e.g. `Rating`) to see if the model is doing well.
#    To do that, you can use:
#      - `ggplot()` to initialize a ggplot object.
#        Set its arguments `data`.
#      - `geom_point()` to do a scatter plot for `Balance` againt `Rating`
#      - `geom_smooth()` to add the regression line for `Balance` and `Rating`
#        - specify `se = FALSE`
#        - use `color="Data"` as your color label
#      - `geom_smooth()` to add the regression line for `Balance` and `pred1`
#        - specify `se = FALSE`
#        - use `color="Predictions"` as your color label
#      - `labs()` to format the labels such that:
#        - `title = "The model did not capture the pattern between Rating and Balance"`.
#        - `x = "Rating"`.
#        - `y = "Balance (USD)"`
#        - `color = "Label"`.
#      - `theme_light()` to change the theme of plots.
#      - `theme()` to change the title and subtitle to the middle of the plot.
#        - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#    Save your plot to `g5`.
## Do not modify this line!
g5<-ggplot(credit_pred)+
  geom_smooth(aes(y=Balance,x=Rating,color="Data"),se=F)+
  geom_smooth(aes(y=pred1,x=Rating,color="Predictions"),se=F)+
  geom_point(aes(y=Balance,x=Rating))+
  labs(title = "The model did not capture the pattern between Rating and Balance",
       x = "Rating",y = "Balance (USD)",
       color = "Label"
       )


# 10. Now let's add one more predictor `Rating` in our model.
#    Fit another linear model `m2` using these variables:
#    `log_limit_per_card`, `Student`, their second order iteration, and `Rating`.
#    Again, save your prediction residuals into a column
#    `resid2`, save your predictions into a column `pred2` into `credit_pred`.
#    Then visualize the distribution of residuals.
#    To do this, you can use:
#      - `lm()` to fit a linear regression, save the model to `m2`.
#      - `add_residuals` to add residuals to the new tibble `credit_pred`.
```

```
#     - `add_predictions` to add predictions to the new tibble `credit_pred`.
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`.
#     - `geom_ref_line()` to add a reference line.
#       - specify `h=0` to use the x-axis as the reference line.
#     - `geom_point()` to add the scatter plot of residuals.
#     - `labs()` to format the labels such that:
#       - `title = "The residual goes up as balance goes up"`.
#       - `subtitle = "But the slope and the range residuals has decreased a lot"`.
#       - `x = "Balance (USD)"`.
#       - `y = "Residuals (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#   Save your plot to `g6`.
#   And your expected output for `m2 %>% broom::tidy()` shoule be:
#   # A tibble: 5 x 5
#   term                    estimate std.error statistic  p.value
#   <chr>                      <dbl>    <dbl>    <dbl>    <dbl>
#   (Intercept)              -207.     163.       -1.27 2.05e- 1
#   log_limit_per_card        -25.0    23.5       -1.06 2.88e- 1
#   StudentYes               -502.     391.       -1.28 2.01e- 1
#   Rating                    2.52     0.104     24.2  8.14e-73
#   log_limit_per_card:StudentYes   120.     52.4       2.29 2.30e- 2
## Do not modify this line!
m2<-lm(Balance~log_limit_per_card*Student+Rating,credit_pred)
credit_pred<-credit_pred%>%add_residuals(m2,"resid2")%>%add_predictions(m2,"pred2")
g6<-ggplot(data = credit_pred, mapping = aes(x=Balance,y=resid2))+geom_ref_line(h=0)+
  geom_point()+labs(
    title = "The residual goes up as balance goes up",
    subtitle = "But the slope and the range residuals has decreased a lot",
    x = "Balance (USD)",
    y = "Residuals (USD)"
  )

# 11. After including `Rating` as a predictor, let's do exercise 8 again
#     and check if our new model captures the relationship between `Rating`
#     and `Balance` and compare the new model with our previous model.
#   To do that, you can use:
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data`.
#     - `geom_point()` to do a scatter plot for `Balance` againt `Rating`
#     - `geom_smooth()` to add the regression line for `Balance` and `Rating`
#       - specify `se = FALSE`
#       - use `color="Data"` as your color label
#     - `geom_smooth()` to add the regression line for `Balance` and `pred1`
#       - specify `se = FALSE`
#       - use `color="Predictions 1"` as your color label
#     - `geom_smooth()` to add the regression line for `Balance` and `pred2`
#       - specify `se = FALSE`
#       - use `color="Predictions 2"` as your color label
```

```
#     - `labs()` to format the labels such that:
#       - `title = "The 2nd model did much better!"`.
#       - `x = "Rating"`.
#       - `y = "Balance (USD)"`
#       - `color = "Label"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#   Save your plot to `g7`.
## Do not modify this line!
g7<-ggplot(data = credit_pred) + geom_point(aes(x=Rating,y=Balance))+
  geom_smooth(aes(x=Rating,y=Balance,color = "Data"),se=F)+
  geom_smooth(aes(x=Rating,y=pred1,color="Predictions 1"),se=F)+
  geom_smooth(aes(x=Rating,y=pred2,color="Predictions 2"),se=F)+
  labs(title = "The 2nd model did much better!",
       x = "Rating",
       y = "Balance (USD)",
       color = "Label")


# 12. By iterating the process above, we can include more variables which
#   improves our model.
#   Let's fit our final model `m3` using these variables:
#   `log_limit_per_card`, `Student`, their second order interaction,
#   `Rating`, `Income`, `Limit` and `Cards`.
#   Again, save your prediction residuals into a column
#   `resid3`, save your predictions into a column `pred3` into `credit_pred`.
#   Then visualize the distribution of residuals.
#   To do this, you can use:
#     - `lm()` to fit a linear regression, save the model to `m3`.
#     - `add_residuals` to add residuals to the new tibble `credit_pred`.
#     - `add_predictions` to add predictions to the new tibble `credit_pred`.
#     - `ggplot()` to initialize a ggplot object.
#       Set its arguments `data` and `mapping`.
#     - `geom_ref_line()` to add a reference line.
#       - specify `h=0` to use the x-axis as the reference line.
#     - `geom_point()` to add the scatter plot of residuals.
#     - `labs()` to format the labels such that:
#       - `title = "The residuals are randomly distributed around x-axis"`.
#       - `subtitle = "Our model is pretty good!"`.
#       - `x = "Balance (USD)"`.
#       - `y = "Residuals (USD)"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#   Save your plot to `g8`.
#   And your expected output for `m3 %>% broom::tidy()` should be:
#   # A tibble: 8 x 5
#   term                  estimate std.error  statistic   p.value
#   <chr>                    <dbl>    <dbl>     <dbl>     <dbl>
#   (Intercept)              -807.     50.6     -15.9     5.92e- 42
```

```
#   log_limit_per_card           7.48     6.54      1.14    2.54e- 1
#   StudentYes                 503.      38.2       13.2     1.41e- 31
#   Rating                    -0.190    0.112     -1.70    9.06e- 2
#   Income                    -10.1      0.0567  -179.      2.97e-308
#   Limit                     0.338    0.00782   43.3     1.48e-131
#   Cards                     27.9      2.35      11.9     5.37e- 27
#   log_limit_per_card:StudentYes    0.0195   5.10      0.00382 9.97e- 1
## Do not modify this line!
m3<-lm(Balance~log_limit_per_card*Student+Rating+Income+Limit+Cards,credit_pred)
credit_pred<-credit_pred%>%add_residuals(m3,"resid3")%>%add_predictions(m3,"pred3")
 g8<- ggplot(credit_pred,mapping = aes(x=Balance,y=resid3))+geom_ref_line(h=0)+
 geom_point()+
 labs(title = "The residuals are randomly distributed around x-axis",
      subtitle = "Our model is pretty good!",
      x = "Balance (USD)",
      y = "Residuals (USD)")

# 13. Let's fit the previous model within every `Ethnicity` group, and save
#   the model information into `credit_model`.
#   To do this, you need to:
#     - Load the `purrr` package.
#     - Create a function `f1` that takes an input tibble `df` and perform a
#     linear regression of `Balance` against  `log_limit_per_card`, `Student`,
#     their second order interaction, `Rating`, `Income`, `Limit`, and `Cards`
#     within the input `df`. In other words, `f1(credit_new)`
#     should return `m3`.
#   You can use:
#     - `lm()` to fit a linear regression with formula `Balance ~
#       log_limit_per_card * Student + Rating + Income + Limit + Cards`.
#     - `group_by()` to group `credit_new` by `Ethnicity`
#     - `nest()` to let each row represents a group
#     - `map()` to map the model `m2` to every group
#     - `mutate()`, `map2()` and `add_residuals()` to add the prediction
#      residuals to each group
#     - `unnest()` to unnest the residuals
#   Your final output `credit_model` is a tibble containing the model
#   information for each group and the first few rows should look like:
#   A tibble: 310 x 16
#   Groups:   Ethnicity [3]
#   Ethnicity      data model Income Limit Rating Cards   Age Education Gender
#   <fct>     <list<df> <lis>  <dbl> <dbl>  <dbl> <dbl> <dbl>     <dbl> <fct>
#   African … [78 × 12] <lm>   16.3  1160   126    3    78       13 Male
#   African … [78 × 12] <lm>   44.5  3500   257    3    81       16 Female
#   African … [78 × 12] <lm>   33.0  3180   224    2    28       16 Male
#   African … [78 × 12] <lm>   35.0  3327   253    3    54       14 Female
#   African … [78 × 12] <lm>   50.7  3977   304    2    84       17 Female
#   African … [78 × 12] <lm>   54.7  4116   314    2    70        8 Female
#   African … [78 × 12] <lm>   58.1  4221   304    3    50        8 Male
#   African … [78 × 12] <lm>   29.7  3536   270    2    52       15 Female
#   African … [78 × 12] <lm>   19.1  3291   269    2    75       13 Female
#   African … [78 × 12] <lm>   26.1  3388   266    4    74       17 Female
#   … with 300 more rows, and 6 more variables: Student <chr>, Married <fct>,
```

```
#    Balance <dbl>, Limit_per_card <dbl>, log_limit_per_card <dbl>, resid <dbl>
## Do not modify this line!
library(purrr)
 f1<-function(df) lm(Balance~log_limit_per_card*Student+Rating+Income+Limit+Cards,df)
 credit_model<-credit_new%>%group_by(Ethnicity)%>%nest()%>%
mutate(model=map(data,f1))%>%
   mutate(resid=map2(data,model,add_residuals))%>%unnest(resid)


# 14. Create a residual plot of `resid` against `Balance`
#     for all the three models. Use different colors
#     to show residuals from different models.
#     To do this, you can use:
#      - `ggplot()` to initialize a ggplot object.
#        Set its arguments `data` and `mapping`.
#      - `geom_point()` to add the scatter plot of residuals.
#        - set `color = Ethnicity` in `aes()`
#      - `geom_smooth()` to add a regression line for residuals for
#        different groups
#        - set `color = Ethnicity` in `aes()`
#        - specify `se = FALSE`
#        - specify `method = "lm"`
#      - `labs()` to format the labels such that:
#        - `title = "For all three groups, the residuals are randomly
#          distributed around x-axis"`.
#        - `subtitle = "Our model is pretty good!"`.
#        - `x = "Balance (USD)"`.
#        - `y = "Residuals (USD)"`.
#      - `theme_light()` to change the theme of plots.
#      - `theme()` to change the title and subtitle to the middle of the plot.
#        - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#        - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#    Save your plot to `g9`.
## Do not modify this line!
g9<-ggplot(data = credit_model,mapping = aes(x=Balance,y=resid))+
  geom_point(aes(color=Ethnicity))+
  geom_smooth(aes(color = Ethnicity),se=F,method = "lm")+
  labs(title = "For all three groups, the residuals are randomly
 distributed around x-axis",
       subtitle = "Our model is pretty good!",
       x = "Balance (USD)",
       y = "Residuals (USD)"
  )
# 15. We want to evaluate the three models by adjusted r-squared.
#    Create a tibble `credit_model_glance` with the adjusted r-squared for
#    each model and sort them by ascending order.
#    To do this, you need to:
#      - Load the `broom` package.
#    You can use:
#      - `nest()` to nest the residuals and feature information back into
#        `resid` in the `credit_model` tibble.
#      - `glance()` to calculate the summary statistics for each model .
```

```
#     - `mutate()` and `map()` to save the summary statistics.
#     - `unnest()` to unnest the all the summary statistics.
#     - `arrange()` to sort the tibble.
#   Save your output to `credit_model_glance` whose first few rows should
#   look like:
#   A tibble: 3 x 15
#    Groups:   Ethnicity [3]
#   Ethnicity      data model      resid r.squared adj.r.squared sigma statistic
#   <fct>     <list<df[> <lis> <list<df[>     <dbl>         <dbl> <dbl>     <dbl>
#   Caucasian [158 × 12] <lm>  [158 × 13]     0.997         0.997  20.7     7955.
#   African …  [78 × 12] <lm>   [78 × 13]     0.998         0.998  21.0     5023.
#   Asian      [74 × 12] <lm>   [74 × 13]     0.998         0.998  18.2     5721.
#   … with 7 more variables: p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>,
#   BIC <dbl>, deviance <dbl>, df.residual <int>A tibble: 3 x 15
## Do not modify this line!
library(broom)
credit_model_glance<-credit_model%>%nest(resid=c(Income:resid))%>%
  mutate(tm=map(model,glance))%>%unnest(tm)%>%arrange(r.squared)


# HW8: King County House Sales
#
# In this exercise, we will walk you through a complete process of modeling data.
# We suggest the functions you can use to create the tibbles and the plots, but
# you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - We will not check the comparision figures created by `grid.arrange()`
#     but we hope they will be helpful for you to see the difference directly.
#
# 1. Load the packages `tidyverse`, `GGally`, `gridExtra`, `modelr`, `broom`
#   and `lubridate`.
#   Use `read_csv()` to read the dataset `kc_house_data.csv` (located in the
#   directory `data/`) into a tibble `data`.
#   This dataset has information about house sales in King County between
#   May 2014 and May 2015. You can find a detailed dictionary for the dataset
#   at https://rdrr.io/cran/moderndive/man/house_prices.html. In this dataset,
#   each observation is a house sale, identified by `id` and `date`.
#   In this homework, we are going to explore how `price` of a house depends on
#   other factors.
## Do not modify this line!
library(tidyverse)
library(GGally)
library(gridExtra)
library(modelr)
library(broom)
library(lubridate)
data <-read_csv("data/kc_house_data.csv") %>%
```

```
 as_tibble()


# 2. We will start with data cleaning and aggregation.
#   (1) Check that the dataset has no missing value for each column.
#     To do this, you can use:
#       - `map_lgl()` with `is.na()` and `any()`.
#     Store the result into a variable `check_na`.
#     Your output should be a named logical vector corresponding to the 21
#     columns, with all values as `FALSE`.
#   (2) Create a new tibble `data_clean` of size 21613 x 20 with:
#       - All columns except `yr_built`, `yr_renovated`, `id` and `date`
#         from `data`.
#       - `waterfront` recoded as `factor`, where `"0"` is coded as `"No"`
#         and `"1"` as `"Yes"`.
#       - Three aggregated columns:
#         - `quarter`, the quarter when the house was sold.
#         - `age`, the age of the house in years calculated by
#           `2019 - yr_built`.
#         - `renovated`, an indicator of whether the house has been
#           renovated, i.e., whether `yr_renovated != 0`.
#     To do this, you can use:
#       - `mutate()` to create the new columns
#         - `quarter()` to get the quarter of `date`.
#         - `factor()` to create the desired factors.
#         - `row_number()` to generate the row number.
#       - `dplyr::select()` to select the desired columns.
#     To check your result, `data_clean` prints to:
#     # A tibble: 21,613 x 20
#       price bedrooms bathrooms sqft_living sqft_lot floors waterfront
#       <dbl>   <int>    <dbl>      <int>   <int>  <dbl> <fct>
#     1 221900       3     1         1180    5650     1 No
#     2 538000       3     2.25      2570    7242     2 No
#     3 180000       2     1          770   10000     1 No
#     4 604000       4     3         1960    5000     1 No
#     5 510000       3     2         1680    8080     1 No
#     # … with 2.161e+04 more rows, and 13 more variables: view <int>,
#     #   condition <int>, grade <int>, sqft_above <int>,
#     #   sqft_basement <int>, zipcode <int>, lat <dbl>, long <dbl>,
#     #   sqft_living15 <int>, sqft_lot15 <int>, quarter <fct>, age <dbl>,
#     #   renovated <fct>
## Do not modify this line!
check_na<-map_lgl(data,~any(is.na(.)))#结合is.na, any才能对df 使用
data_clean<-data%>%mutate(quarter=factor(quarter(date)),
                          age = 2019 - yr_built,
                          waterfront = factor(ifelse(waterfront==0,"No","Yes")),
                          renovated = factor(ifelse(yr_renovated==0,"No","Yes")))%>%
  select(-c(yr_built, yr_renovated, id,date))
# 3. Before getting deep, we will do some visualization to get a feel for how
#   features are correlated. First, draw and store horizontal boxplots for
#   `price` vs. `waterfront` as `g1` and `price` vs. `renovated` as `g2`.
#   To do this, you can use:
```

```
#     - `ggplot()` to initialize a ggplot object and specify variables to plot.
#     - `geom_boxplot()` to plot the boxplots.
#     - `coord_flip()` to make the boxplots horizontal.
#     - `labs()` to format the labels such that:
#       - For `g1`:
#         - `title = "House Price Increases With A View To Waterfront"`
#         - `x = "View to waterfront"`
#         - `y = "Price ($)"`
#       - For `g2`:
#         - `title = "House Price Increases With A Renovation"`
#         - `x = "Renovation"`
#         - `y = "Price ($)"`
#   To facilitate comparison, you can use `grid.arrange()` to arrange the plots
#   side-by-side.
## Do not modify this line!
theme_set(theme_light())
g1<-ggplot(data_clean,aes(x=waterfront,y=price))+geom_boxplot()+coord_flip()+
  labs(title = "House Price Increases With A View To Waterfront",
       x = "View to waterfront",
       y = "Price ($)")
g2<-ggplot(data_clean,aes(x=renovated,y=price))+geom_boxplot()+coord_flip()+
  labs(title = "House Price Increases With A Renovation",
       x = "Renovation",
       y = "Price ($)")


# 4. Create a pair plot for `price`,`sqft_lot`, `sqft_above`, `sqft_living` and
#   `sqft_basement`.
#   To do this, you can use:
#     - `ggpairs()` to initialize a ggplot object and specify variables to plot.
#     - `labs()` to format the labels such that:
#       - `title = "House Price Is Correlated With Areas"`
#       - `subtitle = "Collinearity of sqft_living and sqft_above/sqft_basement is seen."`
#   Store the plot into a `ggplot` object `g3`.
#   Remark:
#     - We will exclude `sqft_lot`, `sqft_above` and `sqft_basement` from
#       modeling, for either low correlation or collinearity.
#     - Note that the distribution of `price` is highly skewed. Thus, we will
#       apply a log-transformation on `price` when fitting the model.
## Do not modify this line!
g3 <- ggpairs(data_clean,columns =
            c("price","sqft_lot","sqft_above","sqft_living","sqft_basement"))+
  labs(title = "House Price Is Correlated With Areas",
       subtitle = "Collinearity of sqft_living and sqft_above/sqft_basement is seen.")

# 5. We may now start to fit a multiple linear regression model.
#   (1) For our interest, create a new tibble `data_model` of size 21613 x 13
#     to exclude columns `quarter`, `zipcode`, `sqft_above`, `sqft_basement`,
#     `sqft_living15` and `sqft_lot15` from `data_clean`, using methods such
#     as `dplyr::select()`.
#   (2) Use `lm()` to create a linear model `m1` for `log10(price)` against all
#     other features in `data_model`. (hint: use `.` to include all features).
```

```
#       To check your result, `m1 %>% broom::tidy()` prints to:
#       # A tibble: 13 x 5
#         term           estimate  std.error statistic   p.value
#         <chr>             <dbl>     <dbl>     <dbl>     <dbl>
#       1 (Intercept)   -20.8      0.776       -26.8  4.98e-156
#       2 bedrooms      -0.00520   0.00105      -4.97 6.59e-  7
#       3 bathrooms      0.0289    0.00178      16.2  1.31e- 58
#       4 sqft_living    0.0000781 0.00000170   45.9  0.
#       5 floors         0.0246    0.00178      13.8  2.50e- 43
#       6 waterfrontYes  0.157     0.00964      16.3  3.85e- 59
#       7 view           0.0289    0.00116      25.0  6.95e-136
#       8 condition      0.0282    0.00129      21.8  1.33e-104
#       9 grade          0.0782    0.00112      69.7  0.
#      10 lat            0.589     0.00576     102.   0.
#      11 long           0.0206    0.00612      3.37 7.56e-  4
#      12 age            0.00143   0.0000400    35.9  1.68e-273
#      13 renovatedYes   0.0279    0.00405      6.88 5.95e- 12
#   (3) Create a tibble `fit_m1` of size 21613 x 2 with columns `resids` and
#       `preds`, which are the residuals and predictions after fitting `m1`
#       on `data_model`.
#       To do this, you can use:
#         - `add_residuals()` to add residuals.
#         - `add_predictions()` to add predictions.
#         - `dplyr::select()` to select the desired columns.
#       To check your result, `fit_m1` prints to:
#       # A tibble: 21,613 x 2
#           resids preds
#            <dbl> <dbl>
#       1   -0.129   5.48
#       2   -0.0694  5.80
#       3   -0.280   5.54
#       4    0.147   5.63
#       5    0.0657  5.64
#       # … with 2.161e+04 more rows
## Do not modify this line!
data_model <- data_clean %>%
  dplyr::select(-quarter,-zipcode,-sqft_above,-sqft_basement,-sqft_living15,-sqft_lot15,-
sqft_lot)
m1<-lm(log10(price)~.,data_model)
fit_m1 <- data_model %>%
  add_residuals(m1,"resids") %>%
  add_predictions(m1,"preds") %>%
  dplyr::select(resids,preds)
# 6. We will check model assumptions and quality by some diagnostic plots.
#    Draw a scatterplot of `resids` vs. `preds` with a horizontal line of
#    `y = 0` as a reference.
#    To do this, you can use:
#       - `ggplot()` to initialize a ggplot object and specify variables to plot.
#       - `geom_point()` to draw a scatterplot, setting `alpha = 0.3`.
#       - `geom_hline()` to add the line `y = 0`, setting `color = "red"`.
#       - `labs()` to format the labels such that:
#         - `title = "Residuals Almost Have No Pattern"`
```

```
#       - `subtitle = "Potential outliers spotted."`
#       - `x = "Fitted Prices ($)"`
#       - `y = "Residuals ($)"`
#   Store the plot into a ggplot object `g4`.
## Do not modify this line!
g4<-ggplot(fit_m1,aes(x=preds,y=resids))+
 geom_point(alpha = 0.3)+
 geom_hline(yintercept = 0, color = "red")+
 labs(title = "Residuals Almost Have No Pattern",
     subtitle = "Potential outliers spotted.",
     x = "Fitted Prices ($)",
     y = "Residuals ($)")


# 7. Draw a qq-plot of `resids` to check the normality assumption, with a diagonal
#   qq-line as a reference.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify variables to plot.
#     - `stat_qq()` to draw a qq-plot.
#     - `stat_qq_line()` to add a qq-line.
#     - `labs()` to format the labels such that:
#       - `title = "Residuals Are Almost Normal"`
#       - `subtitle = "Potential outliers are seen at the heavy tail."`
#       - `x = "Theoretical Quantiles"`
#       - `y = "Sample Quantiles"`
#   Store the plot into a ggplot object `g5`.
## Do not modify this line!
g5<-ggplot(fit_m1)+
 stat_qq(aes(sample=resids))+
 stat_qq_line(aes(sample = resids))+labs(title = "Residuals Are Almost Normal",
                            subtitle = "Potential outliers are seen at the heavy tail.",
                            x = "Theoretical Quantiles",
                            y = "Sample Quantiles")

# 8. Overall, our model has satisfied the assumptions for linear regression.
#   To further improve, we would like to remove the outliers from the dataset.
#   (1) Identify the outliers using Cook's distance with a threshold of `4/n`,
#     where `n` is the total number of observations. Create a tibble `outliers`
#     of size 1249 x 1 with a column `key`, which is a new surrogate key
#     column for the outliers.
#     To do this, you can use:
#       - `augment()` to get `.cooksd` from `m1`.
#       - `mutate()` to generate a `key` column by `row_number()`.
#       - `filter()` to filter points with `.cooksd >= 4 / nrow(.)`.
#       - `dplyr::select()` to select the desired columns.
#     To check your result, `outliers` prints to:
#     # A tibble: 1,249 x 1
#        key
#      <int>
#     1    6
#     2   22
#     3   37
```

```
#      4    50
#      5    66
#      # … with 1,244 more rows
#    (2) Create a function `remove_outliers()` that
#      - Takes the data frames `df` and `outliers` as input.
#      - Modifies `df` by:
#        - Creating a new surrogate `key` column using row numbers.
#        - Removing the records for outliers.
#        - Removing the `key` column.
#      To do this, you can use:
#        - `mutate()` to generate a `key` column by `row_number()`.
#        - `anti_join()` to join `df` with `outliers` by `key`.
#        - `dplyr::select()` to select the desired columns.
#    (3) Create a tibble `data_no_outlier` of size 20364 x 13 by applying the
#      function `remove_outliers()` to `data_model`.
#      To check your result, `data_no_outlier` prints to:
#      # A tibble: 20,364 x 13
#        price bedrooms bathrooms sqft_living floors waterfront  view
#        <dbl>    <int>     <dbl>      <int>  <dbl> <fct>      <int>
#      1 221900       3     1          1180      1 No           0
#      2 538000       3     2.25       2570      2 No           0
#      3 180000       2     1           770      1 No           0
#      4 604000       4     3          1960      1 No           0
#      5 510000       3     2          1680      1 No           0
#      # … with 2.036e+04 more rows, and 6 more variables: condition <int>,
#      #   grade <int>, lat <dbl>, long <dbl>, age <dbl>, renovated <fct>
## Do not modify this line!
outliers <- data_clean %>%
  mutate(key = row_number())%>%
  filter(augment(m1)$.cooksd >= 4/nrow(.)) %>%#augment(model) 意思就是对原df添加
apply model后的数据，里面有一项叫.cooksd
  dplyr::select(key)
remove_outliers<-function(df,outliers){
  df%>%mutate(key = row_number())%>%
    anti_join(outliers,by = "key")%>%#注意join要key
    dplyr::select(-key)
}
data_no_outlier<-remove_outliers(data_model,outliers)

# 9. We will add an interaction term into our model to fit on the new dataset.
#    (1) Create a model `m2` for `log10(price)` against all other features,
#      plus an interaction term of `lat*long` in `data_no_outlier`.
#      To check your result, `m2 %>% broom::tidy()` prints to:
#      # A tibble: 14 x 5
#        term           estimate  std.error statistic   p.value
#        <chr>             <dbl>     <dbl>     <dbl>     <dbl>
#        1 (Intercept)   3104.       226.           13.7  8.44e- 43
#        2 bedrooms       -0.00751    0.000996      -7.54 4.99e- 14
#        3 bathrooms       0.0331     0.00163       20.3  8.83e- 91
#        4 sqft_living     0.0000834  0.00000161    51.8  0.
#        5 floors          0.0253     0.00160       15.8  1.21e- 55
```

```
#         6 waterfrontYes    0.161      0.0128        12.6  3.18e- 36
#         7 view             0.0282     0.00111       25.4  2.39e-140
#         8 condition        0.0253     0.00116       21.8  5.60e-104
#         9 grade            0.0758     0.00102       73.9  0.
#         10 lat            -65.1       4.75         -13.7  1.49e- 42
#         11 long            25.6       1.85          13.8  2.02e- 43
#         12 age             0.00152    0.0000366     41.6  0.
#         13 renovatedYes    0.0253     0.00410        6.16 7.43e- 10
#         14 lat:long       -0.538      0.0389       -13.8  2.69e- 43
#    (2) Follow the same procedure in question 5 to generate a tibble `fit_m2`
#       of size 20364 x 2 to store the residuals and predictions by `m2`.
#       To check your result, `fit_m2` prints to:
#       # A tibble: 20,364 x 2
#         resids preds
#          <dbl> <dbl>
#       1  -0.122   5.47
#       2  -0.0843  5.82
#       3  -0.282   5.54
#       4   0.155   5.63
#       5   0.0738  5.63
#       # … with 2.036e+04 more rows
## Do not modify this line!
m2<-lm(log10(price)~.+lat*long,data_no_outlier)
fit_m2 <- data_no_outlier %>%
  add_residuals(m2,"resids") %>%
  add_predictions(m2,"preds") %>%
  dplyr::select(resids,preds)

# 10. We will observe the improvements in two ways:
#    (1) A higher R-squared.
#       Create a tibble `glance` of size 2 x 12, with columns of statistics
#       obtained from `glance()` and an additional column indicating the model.
#       To do this, you can use:
#         - `glance()` to draw model statistics from `m1` and `m2`.
#         - `bind_rows()` to bind the two tibbles after calling `glance()`,
#           setting `.id = "model"`.
#       To check your result, `glance` prints to:
#       # A tibble: 2 x 12
#         model r.squared adj.r.squared  sigma statistic p.value    df logLik
#         <chr>     <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <int>  <dbl>
#       1 1         0.761         0.761 0.112      5730.       0    13 16682.
#       2 2         0.794         0.794 0.0955     6045.       0    14 18934.
#       # … with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
#       #   df.residual <int>
#    (2) Better residual plots.
#       Reproduce the residual vs. fitted plot and the qq-plot following the same
#       procedure in question 6 and 7. Store the corresponding plots into ggplot
#       objects `g6` and `g7`, respectively. Modify the labels such that:
#         - For `g6`:
#           - `title = "Residuals Have No Pattern"`
#           - `subtitle = "No outliers are found."`
#         - For `g7`:
```

```
#          - `title = "Residuals Are Normal"`
#          - `subtitle = "No more deviations at tails."`
#       To facilitate comparison, you can use `grid.arrange()` to compare `g4`
#       to `g6` and `g5` to `g7` side-by-side.
## Do not modify this line!
glance<-bind_rows(glance(m1),glance(m2),.id = "model")
g6<-ggplot(fit_m2,aes(x=preds,y=resids))+
  geom_point(alpha = 0.3)+
  geom_hline(yintercept = 0, color = "red")+
  labs(title = "Residuals Have No Pattern",
       subtitle = "No outliers are found.",
       x = "Fitted Prices ($)",
       y = "Residuals ($)")
g7<-ggplot(fit_m2)+
  stat_qq(aes(sample=resids))+
  stat_qq_line(aes(sample = resids))+
  labs(title = "Residuals Are Normal",
       subtitle = "No more deviations at tails.",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles")
# 11. So far, we haven't taken the date of sale (`quarter`) into account. Will the
#     model still capture the relationship between `price` and other features
#     for different quarters?
#     To answer this question, create a tibble `by_quarter` of size 4 x 2 with
#     columns `quarter` and `data`, where `quarter` is the quarter when the house
#     was sold and `data` is a list-column for the other features. This dataset
#     should contain only the columns of interest in question 5 and rows without
#     outliers.
#     To do this, you can use:
#       - `dplyr::select()` to exclude columns `zipcode`, `sqft_above`,
#         `sqft_basement`, `sqft_living15` and `sqft_lot15` from `data_clean`.
#       - `remove_outliers()` to remove outliers from the dataset.
#       - `group_by()` to group the data by `quarter`.
#       - `arrange()` to sort the data by ascending `quarter`.
#       - `nest()` to nest the other columns.
#     To check your result, `by_quarter` prints to:
#     # A tibble: 4 x 2
#     # Groups:   quarter [4]
#       quarter          data
#       <fct>   <list<df[,13]>>
#     1 1       [3,836 × 13]
#     2 2       [6,427 × 13]
#     3 3       [5,612 × 13]
#     4 4       [4,489 × 13]
## Do not modify this line!
by_quarter<-data_clean%>%dplyr::select(-zipcode,-sqft_above,-sqft_basement,-
sqft_living15,-sqft_lot15,-sqft_lot)%>%
  remove_outliers(outliers)%>%group_by(quarter)%>%arrange(quarter)%>%nest()


# 12. Create a function `f` that takes an input data frame `df` and fits
#     the same linear model as `m2` (i.e., a regression of the `log10(price)`
```

```
#    on all the other variables plus the interaction between `lat` and
#    `long`) to `df`.
#    To check your result, `f(data_no_outlier)` outputs `m2`.
#    Next, update the tibble `by_quarter` to have size 4 x 5 with three
#    additional list-columns `model`, `resids` and `preds` after fitting `f`.
#    To do this, you can use:
#      - `mutate()` to create the columns
#        - `model` that maps `f()` to `data`.
#        - `resids` that maps `add_residuals()` to `data` and `model`.
#        - `preds` that maps `add_predictions()` to `data` and `model`.
#    To check your result, `by_quarter` prints to:
#    # A tibble: 4 x 5
#    # Groups:   quarter [4]
#      quarter          data model  resids           preds
#      <fct>   <list<df[,13]>> <list> <list>          <list>
#    1 1         [3,836 × 13] <lm>   <tibble [3,836 × … <tibble [3,836 × …
#    2 2         [6,427 × 13] <lm>   <tibble [6,427 × … <tibble [6,427 × …
#    3 3         [5,612 × 13] <lm>   <tibble [5,612 × … <tibble [5,612 × …
#    4 4         [4,489 × 13] <lm>   <tibble [4,489 × … <tibble [4,489 × …
## Do not modify this line!
f<-function(df){
  lm(log10(price)~.+lat*long,df)
}
by_quarter<-by_quarter%>%
mutate(model=map(data,f),resids=map2(data,model,add_residuals),preds=map2(data,mod
el,add_predictions))


# 13. Create a new tibble `by_quarter_fit` of size 20364 x 4 with columns `quarter`,
#    `key`, `resid` and `pred` to store the residuals and predictions, where
#    `key` is a surrogate key column for each record.
#    Note that due to runtime issues, we can not unnest `resids` and `preds`
#    from `by_quarter` in a single command. Instead, we will create two smaller
#    tibbles to unnest `resids` and `preds` one at each time and join them.
#    The two tibbles you need to create, each of size 20364 x 3, are:
#      - `by_quarter_resid`, with columns `quarter`, `key` and `resid`.
#      - `by_quarter_pred`, with columns `quarter`, `key` and `pred`.
#    To do this, you can use:
#      - `unnest()` to unnest `resids`/`preds`.
#      - `mutate()` to generate a `key` column by `row_number()`.
#      - `dplyr::select()` to select the desired columns.
#    Lastly, join `by_quarter_resid` and `by_quarter_pred` as `by_quarter_fit`,
#    using methods such as `inner_join()` with keys `c("quarter", "key")`.
#    To check your result,
#      - `by_quarter_resid` prints to:
#        # A tibble: 20,364 x 3
#        # Groups:   quarter [4]
#          quarter   key   resid
#          <fct>   <int>   <dbl>
#        1 1         1 -0.294
#        2 1         2  0.0641
#        3 1         3  0.0679
```

```
#         4 1         4  0.0389
#         5 1         5 -0.0127
#         # … with 2.036e+04 more rows
#       - `by_quarter_pred` prints to:
#         # A tibble: 20,364 x 3
#         # Groups:   quarter [4]
#           quarter   key   resid
#           <fct>   <int> <dbl>
#         1 1         1  5.55
#         2 1         2  5.64
#         3 1         3  5.40
#         4 1         4  5.47
#         5 1         5  5.74
#         # … with 2.036e+04 more rows
#       - `by_quarter_fit` prints to:
#         # A tibble: 20,364 x 4
#         # Groups:   quarter [4]
#           quarter   key   resid  pred
#           <fct>   <int>   <dbl> <dbl>
#         1 1         1 -0.294   5.55
#         2 1         2  0.0641  5.64
#         3 1         3  0.0679  5.40
#         4 1         4  0.0389  5.47
#         5 1         5 -0.0127  5.74
#         # … with 2.036e+04 more rows
## Do not modify this line!
by_quarter_resid<-by_quarter%>%select(quarter,resids)%>%unnest(resids)%>%
  mutate(key=row_number())%>%dplyr::select(quarter,key,resid)
by_quarter_pred<-by_quarter%>%select(quarter,preds)%>%unnest(preds)%>%
  mutate(key=row_number())%>%dplyr::select(quarter,key,pred)
by_quarter_fit<-inner_join(by_quarter_resid,by_quarter_pred,key = c("quarter", "key"))

# 14. Finally, generate the residuals plots for each of the `quarter`s.
#     To do this, you can use:
#       - `ggplot()` to initialize a ggplot object and specify variables to plot.
#       - `geom_point()` to draw a scatterplot, setting `alpha = 0.3`.
#       - `geom_hline()` to add a line `y = 0`, setting `color = "red"`.
#       - `facet_wrap()` to facet the plots by `quarter`.
#       - `labs()` to format the labels such that:
#         - `title = "Residuals Have No Pattern"`
#         - `subtitle = "All quarters have good fit and safe model assumptions."`
#         - `x = "Fitted Prices ($)"`
#         - `y = "Residuals ($)"`
#     Store the plot into a ggplot object `g8`.
## Do not modify this line!
g8<-ggplot(by_quarter_fit,aes(pred,resid))+
  geom_point(alpha=0.3)+
  geom_hline(yintercept = 0,color = "red")+
  facet_wrap(~quarter)+
  labs(title = "Residuals Have No Pattern",
       subtitle = "All quarters have good fit and safe model assumptions.",
       x = "Fitted Prices ($)",
```

```
y = "Residuals ($)")
```

2019年12月6日 15:27

HW9

```
# HW9: Differential evolution
#
# In this exercise, you will show that for problems with many global optima, optim()
# and other gradient-based methods do not work. In this case, differential evolution
# algo can help. Here, you don't need to actually write an algo from scratch,
# but we hope the exercise can teach something.
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - For graphs with titles, make them use `theme()` with
#     `plot.title = element_text(hjust = 0.5)` and
#     `plot.subtitle = element_text(hjust = 0.5))`.
#
# 1. Create a function called `f_bowl()` which takes in `x` and `y` and then
#   calculate `((0.5*x)^2 + (0.5*y)^2)`.
#
#   Create a function called `f_ackley()` that takes in `x` and `y` and then
#   calculate the formula in the following link:
#   [https://en.wikipedia.org/wiki/Ackley_function].
#
#   Now, maybe you are curious about what 3D shapes of `bowl` and `ackley`
#   look like, so let's draw them.
#   Following visualizations using the `plotly` package.
#   ```
#   library(plotly)
#   grid <- seq(-4, 4, 1e-1)
#   bowl <- outer(grid, grid, f_bowl)
#   ackley <- outer(grid, grid, f_ackley)
#   p1 <- plot_ly() %>%
#     add_surface(x = ~grid, y = ~grid, z = ~bowl) %>%
#     layout(title = "3D shape of bowl")
#   p2 <- plot_ly() %>%
#     add_surface(x = ~grid, y = ~grid, z = ~ackley) %>%
#     layout(title = "3D shape of ackley")
#   ```
## Do not modify this line!
f_bowl <- function(x, y){
  return((0.5*x)^2 + (0.5*y)^2)
}
f_ackley = function(x, y){
  -20 * exp(-0.2 * sqrt(0.5 * (x^2+y^2))) -
    exp(0.5*(cos(2*pi*x)+cos(2*pi*y))) + exp(1) + 20
}
library(plotly)
```

```
grid <- seq(-4, 4, 1e-1)
bowl <- outer(grid, grid, f_bowl)
ackley <- outer(grid, grid, f_ackley)
p1 <- plot_ly() %>%
  add_surface(x = ~grid, y = ~grid, z = ~bowl) %>%
  layout(title = "3D shape of bowl")
p2 <- plot_ly() %>%
  add_surface(x = ~grid, y = ~grid, z = ~ackley) %>%
  layout(title = "3D shape of ackley")



# 2. Create a function called `obj_bowl()` that takes in `par` and calculates
#    values of `f_bowl()`. Similarly, create a function called `obj_ackley()`
#    that takes in `par` and calculates values of `f_ackley()`.
## Do not modify this line!
obj_bowl = function(par){
  f_bowl(par[1], par[2])
}
obj_ackley = function(par){
  f_ackley(par[1], par[2])
}


# 3. Let's find the minizers of those functions using Newton's method.
#    Use:
#      - `set.seed()` to set your seed to `100`.
#      -`.Random.seed()` and store it to `seed1`.
#      - `runif()` to generate 2 random numbers and save values to `par0`.
#      - `optim()` with `method = "BFGS"` to find a minimizer of `obj_bowl()`
#        (DO NOT specify any other parameters in `optim()`).
#    Store the return output into `newton_bowl`. Then, repeat those steps
#    (starting with the `set.seed` to `100`) but
#    store the random seed into `seed2`, use `obj_ackley()` instead of
#    `obj_bowl()` and store the output into `newton_ackley`.
## Do not modify this line!

set.seed(100)
seed1<-.Random.seed
par0<- runif(2)
newton_bowl <- optim(par0,obj_bowl, method="BFGS")
set.seed(100)
seed2<-.Random.seed
par1<- runif(2)
newton_ackley <-optim(par1,obj_ackley, method="BFGS")
# 4. Now let's use differential evolution to the minizers of those two
#    functions. Load the `DEoptim` package.
#    Then use:
#      - `set.seed()` to set your seed to `100`.
#      -`.Random.seed()` and store it to `seed3`
#      - `DEoptim()` to find the minimer of `obj_bowl()` in the region
#       `[-4, 4]x[-4, 4]`.
```

```
#      (DO NOT specify any other parameters in `DEoptim()`)
#    Store the return output into `de_bowl`. Then, repeat those steps
#    (starting with the `set.seed` to `100`) but
#    store the random seed into `seed4`, use `obj_ackley()` instead of
#    `obj_bowl()` and store the output into `de_ackley`.
#    When you look at the results of two methods, you can spot the difference
#    and DE does a better job here!
## Do not modify this line!

library(DEoptim)
set.seed(100)
seed3 <-.Random.seed
de_bowl <-DEoptim(obj_bowl,lower=c(-4,-4),upper=c(4,4))
set.seed(100)
seed4 <-.Random.seed
de_ackley <-DEoptim(obj_ackley,lower=c(-4,-4),upper=c(4,4))


# 5. Load the `tidyverse` package.
#    Create a tibble called `value_of_params` that contains the
#    best set of parameters found at each iteration of the optimization.
#    and also the value of `f_ackley` for those points.
#    To do this, you can use:
#      - Subsetting to get the best member at each iteration from
#      `de_ackley`. Note that it corresponds to the `bestmemit` element from
#      `de_ackley`'s `member`. You can take a peak at the structure of
#      `de_ackley` using `str(de_ackley)`.
#      - `as_tibble()` to transform it into a tibble.
#      - `rename()` to change the names into `x` and `y`.
#      - `mutate()` to generate a new column called `iteration` that record the
#      index of the iteration and another column called `objective` that calculate
#      the value of `f_ackley()`.
#    To check your solution, `value_of_params` prints to:
#    # A tibble: 200 x 4
#    x       y iteration objective
#    <dbl>   <dbl>     <int>     <dbl>
#      1  1.000  -0.0936        1     2.87
#    2  1.000  -0.0936        2     2.87
#    3  1.000  -0.0936        3     2.87
#    4  0.0667  0.151         4     1.06
#    5 -0.116  -0.0936        5     0.932
#    6  0.114  -0.0750        6     0.824
#    7  0.0477  0.0613        7     0.374
#    8  0.0477  0.0613        8     0.374
#    9  0.0477  0.0613        9     0.374
#    10  0.0477  0.0613       10     0.374
#    # … with 190 more rows
## Do not modify this line!
library(tidyverse)
df<-as_tibble(de_ackley$member$bestmemit)
value_of_params<-df%>%rename(x=par1,y=par2)%>%mutate(iteration=
1:nrow(df),objective=f_ackley(x,y))
```

```
# 6. Use `value_of_params` to draw a line plot of `y` vs. `x`.
#    To do this, you can use:
#      - `geom_line()` to draw a line plot.
#      - `scale_color_gradient2()` to scale the colors.
#      - `labs()` to name the title as
#        `"We can see that x and y both converge to 0"`.
#    Store the returned graph to `g1`.
## Do not modify this line!
g1<-
ggplot(value_of_params,aes(x,y))+geom_line(aes(color=objective))+scale_color_gradient2()
+labs(title = "We can see that x and y both converge to 0")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))


# 7. Use `pivot_longer()` to pivot all columns except `iteration` from
#    `value_of_params`. Store the returned tibble to `value_of_params_pivot`.
#
#    Then, use `value_of_params_pivot` to draw a plot of `value` vs.
#    `iteration`, colored by `name`.
#    To do this, you can use:
#      - `geom_line()` to draw a line plot.
#      - `scale_x_log10()` to scale the x-axis.
#      - `geom_hline()` to draw a horizontal line in which intercept of y should
#      take the best value of `f_ackley`. Let the `linetype` be 2
#      - `labs()` to name the
#        - title as
#          `"Evolution of parameters on 200 iterations"`,
#        - subtitle as
#          `"The algorithm approximates more and more to the best parameters and the best
value as more iterations are executed"`,
#        - x-axis as `"Iterations"`,
#        - y-axis as `"Value of parameters"`,
#        - color as `"Name"`.
#    Store the returned graph to `g2`.
#
## Do not modify this line!
value_of_params_pivot<-value_of_params%>%pivot_longer(-iteration,names_to =
"name",values_to = "value")

  g2<-
ggplot(value_of_params_pivot,aes(x=iteration,y=value,color=name))+geom_line()+scale_x_l
og10()+geom_hline(yintercept
=min(subset(value_of_params_pivot,name=="objective")$value),linetype=2)+
  labs(title="Evolution of parameters on 200 iterations",
       subtitle="The algorithm approximates more and more to the best parameters and the
best value as more iterations are executed",
       x="Iterations",
       y="Value of parameters",
       color="Name")+
```

```
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
#这里hline需要的是objective 最小的值


# HW9: Distribution
#
# In this exercise, you will use a simulated data, estimate the parameters
# and compare the result of method of moments and maximum likelihood estimator.
# We will focus on the beta distribution[https://en.wikipedia.org/wiki/Beta_distribution]
#
# 1. Set your seed to `5206`, and generate 1000 samples from a beta distribution
#    with parameters shape1($\alpha$) = 5, shape2($\beta$) = 10.
#    Save your simulation into `x`.
## Do not modify this line!
set.seed(5206)
seed <-.Random.seed
x <- rbeta(1000,shape1 = 5,shape2 = 10 )
#rgamma, rbeta这些都是生成相应distribution 的sample 用的

# 2. We want to estimate the parameters from these simulated points.
#    For beta distribution, we have closed form for the method-of-moments estimates of
#    the parameters. You can find the equations through the wikipedia link above (in
#    section Method of moments).
#    Use the closed form to estimate $\alpha$ and $\beta$. Save the estimated $\alpha$
#    into `alpha0` and $\beta$ into `beta0`.
## Do not modify this line!
m <- c(mean(x), var(x))
alpha0=m[1]*(m[1]*(1-m[1])/m[2]-1)
beta0=(1-m[1])*(m[1]*(1-m[1])/m[2]-1)
#照写即可

# 3. Now instead of using the closed form equations, we will try a numerical method.
#    Let's first create a function `mom_beta` that takes an input `par`, a
#    vector of length 2, representing respectively $\alpha$ and $\beta$ in the
#    beta distribution. The function will output a vector of length 2,
#    representing respectively the mean and variance of the beta distribution.
#    You can find formulas on your probability text book or on the website above.
#    Then create a function factory `obj_beta_factory` that takes an input `x`
#    representing the data, and return a function which takes an input `par` as
#    parameter and beta distribution and returns the sum of squared error
#    between the true mean and variance of `x` and the beta distribution
#    specified by `par`.
## Do not modify this line!
mom_beta <- function(par){
  c(par[1]/(par[1]+par[2]), par[1]*par[2]/(((par[1]+par[2])^2)*(par[1]+par[2]+1)))
}
obj_beta_factory <- function(x){
  moments <- c(mean(x), var(x))
  function(par){
```

```
  differences <- mom_beta(par)-moments
  return(sum(differences^2))
 }

}
```
#按课件gamma改
```
# 4. Then create a function `par_beta` to estimate the parameters from the data.
#   The function takes an input `x` representing the data, and returns the
#   estimated parameters.
#   To do this, you can use:
#     - `obj_beta_factory()` you just created to create an objective function for
#       optimization for our sampled data `x`.
#     - `optim()` to optimize the objective function
#       - choose startpoint as $\alpha=1$ and $\beta=1$
#     - Then extract and return the optimal parameters
#   Implement your `par_beta` on the simulated data `x` in question #1.
#   Save your estimated parameter $\alpha$ into `alpha1` and $\beta$ into `beta1`.
## Do not modify this line!
par_beta <- function(x){
 obj_beta_x <- obj_beta_factory(x)
 optim(c(1,1), obj_beta_x)$par
}
alpha1 <- par_beta(x)[1]
beta1 <- par_beta(x)[2]


# 5. After using the method of moments, let's try maximum likelihood estimators.
#   Create a function `nll_beta` to calculate the negative loglikelihood
#   of our data under the parameters. The function will take two inputs `par`
#   as parameters(a vector of length two) and `x` as data. It returns the
#   calculated negative loglikelihood.
#   To do this, you can use:
#     - `dbeta()` to calculate the likelihood
#       - specify `log = TRUE` to calculate the loglikelihood.
#     - `optim()` and `nll_beta()` you just created
#       - choose startpoint as $\alpha=1$ and $\beta=1$
#       - specify `hessian = T`
#       - specify `lower = 0`
#       - specify `method = "L-BFGS-B"`
#     - `solve()` to get the inverse of hessian matrix
#     - `sqrt()` and `diag()` to get the standard error of each variable
#   Save your estimated parameter $\alpha$ into `alpha2` and $\beta$ into `beta2`.
#   Save the estimated lower and upper bound confidence interval into `lower` and
#   `upper` respectively, they should both be vector of length 2.
#   You can use 1.96 for the 0.975 quantile of the standard normal distribution.
## Do not modify this line!
nll_beta <- function(par, x){
 return(-sum(dbeta(x, par[1], par[2], log=T)))#注意negative
}
fit <- optim(c(1,1),nll_beta,x=x,hessian = T, lower = 0,method = "L-BFGS-B")
alpha2 <- fit$par[1]
```

```
beta2 <- fit$par[2]
se <- sqrt(diag(solve(fit$hessian)))#这里算standar error 然后算bound
lower <- fit$par-se*1.96
upper <- fit$par+se*1.96
# 6. Compare the result of MOM (use `alpha1` and `beta1`) and MLE by ploting the
#    estimated density curves together on the same plot.
#    You should first load `ggplot2` and then add a histogram and the estimated
#    density of the original `x`, then add the two estimated density curves.
#    To do this, you can use:
#      - `as.data.frame()` to tranform `x` into a dataframe.
#      - `ggplot()` to initialize a ggplot object.
#        Set its arguments `data` and `mapping`.
#      - `geom_histogram()` to add a hitogram with density scale.
#        - specify `bins=20`.
#      - `geom_density()` to add the density curve.
#        - specify `color = "Data"`.
#      - `stat_function` to add the three curves for MOM, MLE and True Value.
#        - specify `color = "MOM"`, `color = "MLE"` and `color = "True Value"`
#          for each curve.
#      - `labs()` to format the labels such that:
#        - `title = "Both of the methods seem to do a good job"`.
#        - `x = "Simulated x"`.
#        - `y = "Density"`.
#      - `theme_light()` to change the theme of plots.
#      - `theme()` to change the title and subtitle to the middle of the plot.
#        - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#    Save your figure into `g1`.
## Do not modify this line!
library(purrr)
library(ggplot2)
x_beta_density1 <- partial(dbeta, shape1=alpha1, shape2=beta1)
x_beta_density2 <- partial(dbeta, shape1=alpha2, shape2=beta2)
x_beta_density_T <- partial(dbeta, shape1=5, shape2=10)
x <- as.data.frame(x)
g1 <-ggplot(data=x, mapping=aes(x=x))+
  geom_histogram(aes(y=..density..), bins=20)+#画density的方式
  geom_density(aes(color='Data'))+
  stat_function(fun=x_beta_density1, aes(color='MOM'))+
  stat_function(fun=x_beta_density2, aes(color='MLE'))+
  stat_function(fun=x_beta_density_T, aes(color='True Value'))+
  labs(title = "Both of the methods seem to do a good job",
       x = "Simulated x",
       y = "Density")+
  theme_light()+
  theme(plot.title=element_text(hjust = 0.5))

# HW9: distribution_fish
#
# In this exercise, we will walk you through a complete process of fitting
# distributions.
# We suggest the functions you can use to create the tibbles and the plots, but
```

```
# you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - When defining functions, the "create" parts are not mandatory. We suggest
#     creating intermediate variables to increase the readability of your code,
#     and hopefully to decrease the chance of mistakes.
#
# 1. Load the packages `tidyverse` and `gamlss`.
#   Use `read_csv()` to read the dataset `fish.csv` (located in the directory
#   `data/`) into a tibble `data`.
#   This dataset has information about the number of fish caught by visitors
#   (`count`) at a state park, whose distribution will be explored in this
#   homework.
#   To facilitate fitting, pull `count` out of `data` and store it as a vector
#   `cnt` of length 250, using methods such as `pull()`.
## Do not modify this line!
library(tidyverse)
library(gamlss)
data<-read_csv("data/fish.csv")
cnt<-pull(data,count)

# 2. Before getting deep, we will do some visualization to get a feel for how
#   `count` is distributed. To begin with, plot a histogram for `count`.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify variables to plot.
#     - `geom_histogram()` to draw a histogram.
#     - `labs()` to format the labels such that:
#       - `title = "Count of Fish Caught"`
#       - `subtitle = "A great many visitors caught no fish at all."`
#       - `x = "Fish Caught (n)"`
#       - `y = "Count (n)"`
#   Store the plot into a ggplot object `g1`.
## Do not modify this line!
g1<-ggplot(data = data)+
  geom_histogram(aes(x=count))+
  labs(title = "Count of Fish Caught",
       subtitle = "A great many visitors caught no fish at all.",
       x = "Fish Caught (n)",
       y = "Count (n)")+
  theme_light()


# 3. Plot the empirical cumulative distribution (ECDF) for `count`.
#   To do this, you can use:
#     - `ggplot()` to initialize a ggplot object and specify variables to plot.
#     - `stat_ecdf()` to draw the ECDF.
#     - `labs()` to format the labels such that:
#       - `title = "Empirical Distribution of Fish Caught"`
```

```
#       - `subtitle = "Distribution is right-skewed, suggesting a Poisson distribution with a low
mean."`
#       - `x = "Fish Caught (n)"`
#       - `y = "Count (n)"`
#    Store the plot into a ggplot object `g2`.
## Do not modify this line!
g2<-ggplot(data = data , aes(x=count))+
  stat_ecdf()+
  labs(title = "Empirical Distribution of Fish Caught",
       subtitle = "Distribution is right-skewed, suggesting a Poisson distribution with a low
mean.",
       x = "Fish Caught (n)",
       y = "ECDF")+
  theme_light()


# 4. The histogram and ECDF plots for `count` suggest a zero-inflated Poisson
#    (ZIP) distribution, which concerns a random event with excess zero-count
#    data in unit time. In our case, the number of fish caught was zero-inflated
#    by visitors who did not fish and thus caught no fish at all.
#    The ZIP model employs two parameters, pi and lambda. While pi corresponds
#    to a binary distribution that generates structural zeros, lambda corresponds
#    to the Poisson distribution that generates counts.
#    To find the ZIP model that best fits our data, we will derive the method
#    of moments estimators (MME) and maximum likelihood estimators (MLE) for
#    the two parameters.
#    Let's start with the MME.
#    (1) Define a function `par_pois()` that
#       - Takes an input vector `x`.
#       - Creates a vector `m` of `c(mean(x), var(x))`.
#       - Returns a vector of `c(lambda, pi)`, where `lambda` and `pi` are the
#         MMEs defined in equations 2-3 and 2-4 at
#         https://projecteuclid.org/download/pdf_1/euclid.involve/1513733747.
#         The two elements of the output should be named `lambda` and `pi`.
#    (2) Apply your function `par_pois()` to `cnt` and store the result into
#       a vector `cnt_pois_mme`.
#       To check your result, `round(cnt_pois_mme, 2)` prints to:
#           lambda     pi
#             4.31   0.56
## Do not modify this line!
par_pois<-function(x){
  m<-c(mean(x),var(x))
  lambda<-m[1]+(m[2]/m[1])-1
  pi <- (m[2] - m[1])/(m[1]^2 + m[2] - m[1])
  return(c(lambda = lambda, pi = pi))
}

cnt_pois_mme<-par_pois(cnt)


# 5. Unlike the MMEs, we cannot find the explicit expressions for the MLEs.
#    Instead, we will solve the MLEs by optimization.
```

```
#   Define a function `nll_pois_factory()` that
#     - Takes an input vector `x`.
#     - Creates
#       - An integer `n`, which is the length of `x`.
#       - An integer `y`, which is the number of elements in `x` taking the
#         value 0.
#       - A float `xbar`, which is the mean of `x`.
#     - Returns a function that
#       - Takes an input vector `par` of `c(lambda, pi)`.
#       - Returns a value for the negative log-likelihood function defined
#         in equation 2-8 at
#         https://projecteuclid.org/download/pdf_1/euclid.involve/1513733747.
#         - You can leave out the last term involving the factorial of `x`,
#           as it is not relevant when taking the gradients w.r.t. the two
#           parameters.
#   To check your result, `nll_pois_factory(cnt)(c(4, 0.5))` prints to:
#   [1] -48.84884
## Do not modify this line!
nll_pois_factory<-function(x){
 n=length(x)
 y= length(which(x==0))
 xbar = mean(x)
 function(par){
  l<-y*log(par[2]+(1-par[2])*exp(-par[1]))+(n-y)*log(1-par[2])-(n-
y)*par[1]+n*xbar*log(par[1])
  return(-l)
 }
}


# 6. Define a function `par_pois2()` that
#     - Takes an input vector `x`.
#     - Creates
#       - A vector `par0` by applying `par_pois()` to `x`.
#       - A function `nll_pois_x` by applying `nll_pois_factory()` to `x`.
#       - A list `fit` that stores the result of the optimizing `nll_pois_x`
#         over `par0`.
#         - To do this, you can use `optim()` with
#           - `method = "L-BFGS-B"`;
#           - `hessian = TRUE`;
#           - `lower = c(0.001, 0)`;
#           - `upper = c(Inf, 0.999)`.
#       - A Fisher Information matrix `fisher_info` from `fit$hessian` using
#         `solve()`.
#       - A vector `se` for the standard errors, which are the square roots
#         of the diagonal elements of `fisher_info`.
#     - Returns a tibble of size 2 x 5 with columns
#       - `parameter`: `lambda` and `pi`;
#       - `value`: the MLEs (i.e., `fit$par`);
#       - `se`: the standard errors;
#       - `lower` and `upper`: the bounds for the 95% confidence intervals.
#         You can use `1.96` for the 0.975% quantile of the standard normal
```

```
#         distribution.
#    Apply your function `par_pois2()` to `cnt` and store the result into a
#    tibble `fit_mle`.
#    To check your result, `fit_mle %>% mutate_at(2:5, round, 1)` prints to:
#    # A tibble: 2 x 5
#      parameter value    se lower upper
#      <chr>     <dbl> <dbl> <dbl> <dbl>
#    1 lambda      4.3   0.2   3.9   4.7
#    2 pi          0.6   0     0.5   0.6
## Do not modify this line!
par_pois2<-function(x){
  par0<-par_pois(x)
  nll_pois_x<-nll_pois_factory(x)
  fit<-optim(nll_pois_x,par=par0,method = "L-BFGS-B",hessian = TRUE,lower =
c(0.001,0),upper = c(Inf,0.999))
  fisher_info<-solve(fit$hessian)
  se<-sqrt(diag(fisher_info))
  lower<-c(fit$par[1]-1.96*se[1],fit$par[2]-1.96*se[2])
  upper<-c(fit$par[1]+1.96*se[1],fit$par[2]+1.96*se[2])
  tibble("parameter"=c("lambda","pi"),"value"=fit
$par,"se"=se,"lower"=lower,"upper"=upper)
}
fit_mle<-par_pois2(cnt)

# 7. We can see that the standard errors are small, meaning that the predictions
#    are pretty accurate. To confirm the result, we will check the goodness of
#    fit by the qq-plot.
#    First, create a vector `cnt_pois_mle` of length 2 for the MLEs from
#    `fit_mle`, using methods such as `pull()`.
#    Then, create a list `dparams_mle` of two components, `mu` and `sigma`,
#    where `mu` corresponds to `cnt_pois_mle[0]` (i.e., `lambda`) and
#    `sigma` corresponds `cnt_pois_mle[1]` (i.e., `pi`).
#    Lastly, plot a qq-plot of actual vs. fitted distributions of `count` in
#    `data`.
#    To do this, you can use:
#      - `ggplot()` to initialize a ggplot object and specify variables to plot.
#      - `stat_qq()` to draw the qq-plot, setting
#        - `distribution = qZIP`.
#        - `dparams = dparams_mle`.
#      - `stat_qq_line()` to draw a qq-line for reference, setting
#        - `distribution = qZIP`.
#        - `dparams = dparams_mle`.
#        - `color = "red"`.
#      - `labs()` to format the labels such that:
#        - `title = "Good Fit Using MLEs"`
#        - `subtitle = "Distributions of sample and theoretical quantiles agree."`
#        - `x = "Theoretical Quantiles"`
#        - `y = "Sample Quantiles"`
#    Store the plot into a ggplot object `g3`.
#    We can see that the MLEs have pretty good fit for our data.
## Do not modify this line!
cnt_pois_mle<-pull(fit_mle,value)
```

```
dparams_mle<-list(mu=cnt_pois_mle[1],sigma=cnt_pois_mle[2])
g3<-ggplot(data = data,mapping = aes(sample=count))+#stat_qq需要sample aes
 stat_qq(distribution = qZIP,dparams = dparams_mle)+
 stat_qq_line(distribution = qZIP,dparams = dparams_mle,color = "red")+
 labs(title = "Good Fit Using MLEs",
     subtitle = "Distributions of sample and theoretical quantiles agree.",
     x = "Theoretical Quantiles",
     y = "Sample Quantiles")+
 theme_light()

# HW9: Inverse Gaussian distribution
#
# In this exercise, we will generate random data, estimate the parameters
# and compare the result of method of moments and maximum likelihood estimators.
# We will focus on the inverse gaussian distribution.
# For details, here is the link: [https://en.wikipedia.org/wiki/Inverse_Gaussian_distribution]
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - For graphs with titles, make the format as
# `theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))`.
#
# 1. Load the packages `statmod`, `ggplot2` and `tidyverse`.
#   Set your seed to `100`. Store the seed to `seed1`.
#   Generate 1000 samples from a inverse gaussian
#   distribution with parameters `mean` = 2, `shape` = 3.
#   To do this, you should use:
#     - `set.seed()` to set random seed
#     - `rinvgauss()` to generate random variables from inverse gaussian
#   distribution
#   Save your simulation into `random_data`.
## Do not modify this line!
library(statmod)
library(ggplot2)
library(tidyverse)
set.seed(100)
seed1<-.Random.seed
random_data<-rinvgauss(1000,mean = 2,shape = 3)


# 2. Create a function `mom_invgauss()` that takes an input `par`. The function
#   should finally return the moments of mean and variance of inverse gassian
#   distribution represented by parameters. You can find formulas on the website
#   presented above.
#   To do this, you can follow the steps:
#     - create a vector of length 2, use two parameters stored in the
#       `par` to represent the mean and variance of the inverse gaussian
#       distribution.
#     - Store the returned vector to `moments` and return it.
#   Example: `mom_invgauss(c(1,1))` would return `[1] 1 1`.
```

```
#
#   Create a function `obj_invgauss()` that takes in `x` which represents
#   the data, and an input `par` as parameters of inverse gaussian distribution.
#   The funtion should return the squared error between the true mean and
#   variance of `x` and the inverse gaussian distribution specified by `par`.
#   To do this, you can follow the steps:
#     - create a vector called `moments` that stores mean and
#       variance of `x`.
#     - use `mom_invgauss()` that we just implemented to
#       generate estimated moments.
#     - calculate the differences between estimated moments
#       and true moments, and store returned value to `differences`.
#     - return the sum of `differences` as the sum of square errors.
#   Example: `obj_invgauss(c(1,1),c(1,2,3))` would return `[1] 1`.
#
#   Create a function `par_invgauss1()` that take `x` as an input and returns the
#   estimated parameters.
#   To do this, you can use:
#     - `optim()` to optimize the objective function
#       - choose startpoint as `mean` = 1 and `shape` = 1.
#     - Then extract and return the optimal parameters.
#   Example: `par_invgauss1(c(1,2,3))` would return `[1] 2.00 8.00`
#   printed in 2 digits.
## Do not modify this line!
mom_invgauss <- function(par){
  moments <- c(par[1], (par[1]^3)/par[2])
  return(moments)
}
obj_invgauss<-function(par,x){
  moments<-c(mean(x),var(x))
  differences<-mom_invgauss(par)-moments
  sum(differences^2)
}
par_invgauss1<-function(x){
  optim(c(1,1),obj_invgauss, x=x)$par
}

# 3. Use `par_invgauss1()` to estimate the parameters of `random_data`. Store
#   returned estimated parameters to `par1`.
#   Create a list that owns two parameters in `par1`, store returned list
#   to `dparams`.
#   Use `tibble()` to transform `random_data` into a tibble, store the
#   returned data frame to `random_data_tbl`.
#   Draw a qq-plot to plot the estimated distribution vs. true value.
#   To do this, you can use:
#     - `ggplot()` to plot the `random_data`.
#     - `geom_qq()` to draw a inverse gaussian distribution with parameters
#       of `dparams`.
#     - `stat_qq_line()` to draw a straight line that takes inverse gaussian
#       distribution with parameters of `dparams`, make the `color` as red.
#     - `labs()` to name the title as `"The sample seems to distribute along the straight line"`,
#       subtitle as `"The MOM estimators fit well"`,
```

```
#       x-axis as `"Theoretical values"`,
#       y-axis as `"Actual values"`.
#    Store returned graph to `g1`.
## Do not modify this line!
par1<-par_invgauss1(random_data)
dparams<-list(par1[1],par1[2])
random_data_tbl<-tibble(random_data)
g1 <- ggplot(data=random_data_tbl, aes(sample=random_data))+
  geom_qq(distribution=qinvgauss,dparams = dparams)+#注意ditribution名字
  stat_qq_line(distribution=qinvgauss,dparams = dparams, color="red")+
  labs(title="The sample seems to distribute along the straight line",
       subtitle="The MOM estimators fit well",
       x="Theoretical values",
       y="Actual values")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))
# 4. Create a function `nll_invgauss` to calculate the negative loglikelihood
#    of our data under the parameters. The function will take two inputs `par`
#    as parameters(a vector of length two) and `x` as data. It returns the
#    calculated negative loglikelihood.
#    To do this, you can use:
#      - `dinvgauss()` to calculate the likelihood
#        - specify `log = TRUE` to calculate the loglikelihood.
#
#    Set your seed to `100`. Store the seed to `seed2`.
#    To do this, you should use:
#      - `set.seed()` to set random seed
#
#    Now we will use some package to calculate MLE.
#    To do this, you can use:
#      - `optim()` and `nll_invgauss()` you just created. Inside the function,
#      you should:
#        - choose startpoint as `mean` = 1 and `shape` = 1.
#        - specify `hessian = TRUE`
#        - specify `lower = 0`
#        - specify `method = "L-BFGS-B"`
#        - specify `x` as `random_data`
#        - DON'T sepcify any other parameters in `optim()`
#    Store the returned object to `fit`.
#
#    Use `solve()` to get the inverse of hessian matrix from `fit`, then use
#    `sqrt()` and `diag()` to get the standard error of each variable.
#    Store returned standard error to `se`.
#
#    Extract parameters of fit and save them to `par2`. Then, calculate lower
#    and upper bound confidence interval. Save the estimated lower and upper bound
#    confidence interval into `lower` and`upper` respectively, they should both be
#    vector of length 2.
#    You can use 1.96 for the 0.975 quantile of the standard normal distribution.
## Do not modify this line!
nll_invgauss <- function(par,x){
  return(-sum(dinvgauss(x, par[1], par[2], log = T)))
```

```
}
set.seed(100)
seed2 <- .Random.seed
fit <- optim(c(1, 1),
          nll_invgauss,
          hessian = T,
          lower = 0,
          method = "L-BFGS-B",
          x = random_data,)
se<-sqrt(diag(solve(fit$hessian)))
par2<-fit$par
lower <- par2-se*1.96
upper <- par2+se*1.96
# 5. Follow the link given above to find the formulas of mu hat and shape hat.
#    Use those formulas to estimate parameters directly and save returned value to
#    `mu_hat` and `shape_hat` accordingly. Here, the results should be same as `par2`.
## Do not modify this line!
mu_hat=mean(random_data)
shape_hat =1/(sum(1/random_data-1/mu_hat)/length(random_data))


# 6. Use `ks.test()` to test if parameters from MLE fit well. The argument
#    should take in `random_data`, `pinvgauss` and two parameters in `par2`.
## Do not modify this line!
ks.test(random_data,pinvgauss, par2[1], par2[2] )


# 7. Compare the result of MOM and MLE by ploting the estimated density curves
#    together on the same plot.
#    You should add a histogram and the estimated density of
#    the original `random_data`, then add the two estimated density curves.
#    To do this, you can use:
#      - `ggplot()` to take in `random_data_tbl` and set `x` as `random_data`.
#      - `geom_histogram()` to add a hitogram with density scale, specify `bins=15`.
#      - `geom_density()` to add the density curve, specify `color = "Data"`.
#      - `stat_function` to add the three curves for MOM, MLE and True Value.
#        - specify `color = "MOM"`, `color = "MLE"` and `color = "True Value"`
#          for each curve.
#      - `labs()` to names the title as
#        `"Both MOM and MLE did good jobs to estimate true distribution"`.
#        x-axis as `"Random data"`,
#        y-axis as`"Density"`.
#        Store returned graph to `g2`.
#
## Do not modify this line!
g2 <- ggplot(data=random_data_tbl, aes(x=random_data))+
  geom_histogram(aes(y=..density..), bins=15)+
  geom_density(aes(color="Data"))+
  stat_function(aes(color="MOM"),fun=partial(dinvgauss, mean = par1[1], shape = par1[2]))+
  stat_function(aes(color="MLE"),fun = partial(dinvgauss, mean = par2[1], shape = par2[2]))+
  stat_function(aes(color="True Value"),fun = partial(dinvgauss, mean = 2, shape = 3))+
  labs(title = "Both MOM and MLE did good jobs to estimate true distribution",
```

```
     x = "Random data", y = "Density") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))

# HW9: Optimization
#
# In this exercise, you will use linear programming to solve a problem by
# plotting the feasible region of solution and by using programming tools.
#
# 1. A company makes two products (X and Y) using two machines (A and B).
#    Each unit of X that is produced requires 50 minutes processing time on
#    machine A and 30 minutes processing time on machine B. Each unit of Y that
#    is produced requires 24 minutes processing time on machine A and 33 minutes
#    processing time on machine B. At the start of the current week there are 30
#    units of X and 90 units of Y in stock. Available processing time on machine
#    A is forecast to be 40 hours and on machine B is forecast to be 35 hours.
#    The demand for X in the current week is forecast to be 75 units and for Y
#    is forecast to be 95 units. Company policy is to maximize the combined sum
#    of the units of X and the units of Y in stock at the end of the week.
#    Formulate the problem of deciding how much of each product to make in the
#    current week as a linear program.
#    Let's first solve this linear program graphically by plotting out the
#    feasible region.
#    To do this, you can:
#      - Load `ggplot2` package.
#      - Create a sequence `x` from 30 to 50, with step size 0.1.
#      - Use `as.data.frame()` to change `x` to a data frame.
#      - Use `ggplot()` to initialize a ggplot object.
#        Set its arguments `data` and `mapping`.
#      - Determine the constraints of the maximum running time of machine A and B,
#        represent the constraints by two lines(functions).
#      - Use `stat_function()` to draw the two lines.
#        - Use `annotate()` to add annotation for the line indicating constraint for machine A
#          - specify `x = 35`, `y = 20` and `label = "Constraint of Machine A"`
#        - Use `annotate()` to add annotation for the line indicating constraint for machine B
#          - specify `x = 40`, `y = 32` and `label = "Constraint of Machine B"`
#      - Determine what are the minimum numbers of production for X and Y
#      - Use `geom_hline()` to plot the minimum production for Y
#        - Use `annotate()` to add annotation for the line indicating constraint for Y
#          - specify `x = 40`, `y = 3` and `label = "Constraint of Product Y"`
#      - Use `geom_vline()` to plot the minimum production for X
#        - Use `annotate()` to add annotation for the line indicating constraint for X
#          - specify `x = 48`, `y = 12` and `label = "Constraint of Product X"`
#      - Determine the feasible region and indicate the region by its vertex,
#        use `geom_point()` to point out the vertex in red
#      - `labs()` to format the labels such that:
#        - `title = "Feasible region of Linear Programming"`.
#      - `theme_light()` to change the theme of plots.
#      - `theme()` to change the title and subtitle to the middle of the plot.
## Do not modify this line!
library(ggplot2)
x<-seq(30,50,0.1)
```

```
x <- as.data.frame(x)
g1 <- ggplot(data=x, mapping=aes(x=x)) +
  stat_function(fun=function(x) (40*60-50*x)/24) + #意思是x,y数量的所需时间为40*60,用x表示y就得到这个式子
  annotate("text", x=35, y=20, label = "Constraint of Machine A") +
  stat_function(fun=function(x) (35*60-30*x)/33) +#同上
  annotate("text", x=40, y=32, label = "Constraint of Machine B") +
  geom_hline(yintercept=95-90) + #结束时需要的减去原来的即可
  annotate("text", x=40, y=3, label = "Constraint of Product Y") +
  geom_vline(xintercept=75-30) +
  annotate("text", x=48, y=12, label = "Constraint of Product X") +
  geom_point(aes(x = 45,y = 5),color = "red")+#三个点通过方程解得
  geom_point(aes(x = 45,y = 6.25),color = "red")+
  geom_point(aes(x = 45.6,y = 5),color = "red")+
  labs(title = "Feasible region of Linear Programming") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))

# 2. The goal is to maximize the combined sum of the units of X and the units of
#    Y in stock at the end of the week, which is the same as to maximize the
#    combined sum of the units produced during this week.
#    For the three vertex in the `g1`, plot three parallel lines with slope -1 by
#    using `stat_function()`.
#    Can you tell what is the optimal point for this linear programming problem?
#    Save your plot into `g2` and save your answer of the optimal `x` and `y` into
#    `x1`, `y1`.
## Do not modify this line!
g2 <- g1 +
  stat_function(fun=function(x) -x+50.6, color="blue")+
  stat_function(fun=function(x) -x+50, color="blue")+
  stat_function(fun=function(x) -x+51.25, color="blue")#过每个点的-1slope线即可
x1 <- 45#选取x+y最大的函数
y1<- 6.25

# 3. Now let's load the `lpSolve` package to solve the above problem using the
#    computational tools.
#    To do this, you should follow these steps:
#    - Create a matrix representing the constraints, the dimension should be 4*2.
#      Two rows corresponding to the constraints on machines and two rows corresponding
#      to the constraints on products.
#    - Use `lp` to solve the linear programming.
#      - specify `direction = "max"`
#      - use `>=` for all constraints in `const.dir`
#    Save the output of `lp` into `s1`.
## Do not modify this line!
library(lpSolve)
C <- c(1,1)#表示x+y
A <- matrix(c(-50, -24, #每行就是上面的constraint方程。头两条取负值是因为最后要
```

用>=

```
        -30, -33,
        1, 0,
        0, 1), nrow=4, byrow=TRUE)
B <- c(-40*60, -35*60, 45, 5)
s1 <- lp(direction = "max", objective.in = C, const.mat = A, const.dir = c(">=", ">=", ">=",
">="), const.rhs = B)

# 4. For the above problem, we only consider two variables, thus we can solve it
#   graphically, but if we have more than 3 variables to consider, the
#   graphical way is no longer straightforward.
#   Let's use the `lp` function to solve the following problem:
#   The facility has four machines of type 1, five of type 2, three of type 3
#   and seven of type 4. Each machine operates 40 hours per week. The company
#   has 5 products in total, which generate different profits and have
#   different processing time on different machines. The relationship is
#   shown in the following table:
#   Machine     Quantity   Product1  Product2  Product3  Product4  Product5
#   M1            4        1.2       1.3       0.7       0         0.5
#   M2            5        0.7       2.2       1.6       0.5       1
#   M3            3        0.9       0.7       1.3       1         0.8
#   M4            7        1.4       2.8       0.5       1.2       0.6
#   Unit ptofit $           18        25        10        12        15
#   The units of products can only be integers.The company wants to maximize
#   the profit. Solve this linear programming problem and save the output of
#   `lp` into `s2`.
#   To do this, you should follow these steps:
#   - Create a matrix representing the constraints, the dimension should be 4*5.
#     Each column represents a product and each row represents the constraint on
#     a machine.
#   - Use `lp` to solve the linear programming.
#     - specify `direction = "max"`
#     - use `<=` for all constraints in `const.dir`
#
#
## Do not modify this line!
const.mat <-matrix(c( 1.2,1.3,0.7 , 0 ,0.5,
                      0.7, 2.2 ,1.6, 0.5,1,
                      0.9 ,0.7,1.3 , 1,0.8,
                      1.4, 2.8, 0.5,1.2,0.6),ncol = 5, byrow = TRUE)

s2 <-lp(direction = "max",
        objective.in =c(18, 25, 10, 12, 15),
        const.mat = const.mat,
        const.dir =c("<=","<=","<=","<="),
        const.rhs =c(4*40, 5*40, 3*40,7*40),
        all.int=T)
#rhs就是限制，这里是以时间为唯一限制，objective 函数即是profit*数量
```

```
# HW9: Optimization: Logistic Regression with Newton's Method
#
# In this exercise, we will walk you through the process of implementing Newton's
# method to solve the logistic regression problem.
# We suggest the functions you can use to create the tibbles and the plots, but
# you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#
# Throughout the exercise:
#   - Use `%>%` to structure your operations.
#   - Do NOT use `for`, `while` or `repeat` loops for parts other than
#     question 5.
#
# 1. Load the packages `tidyverse`, `numDeriv`, `MASS` and `broom`.
#   Use `read_csv()` to read the dataset `admission.csv` (located in the directory
#   `data/`) into a tibble `data`.
#   In this homework, we would like to fit a logistic regression model on `admit`
#   vs. `gre` and `gpa`. Instead of using the in-built optimization method to
#   find out the coefficient estimates, we will write up our own implementation
#   of Newton's Method to do so.
#   To facilitate analysis, create
#   (1) A matrix `X` of size 400 x 2 for `gre` and `gpa` in `data`.
#       To do this, you can use `dplyr::select()` with `as.matrix()`.
#   (2) A vector `y` of length 400 for `admit` in `data`.
#       To do this, you can use `pull()`.
## Do not modify this line!
library(tidyverse)
library(numDeriv)
library(MASS)
library(broom)
library(readr)
data <- read_csv("data/admission.csv")
x <- data %>% dplyr::select(gre, gpa)
X <- as.matrix(x)
y <- data %>% pull(admit)
# 2. We will use the formulas for the hypothesis function, the gradient, the
#   Hessian and the cost function at https://stanford.io/36RbPAj to build some
#   helper functions for our algorithm.
#   First, define a function `sigmoid()` that
#       - Takes an input matrix `x`.
#       - Returns a matrix for S(x) = 1 / (1 + e^(-x)).
#   To check your result, `sigmoid(X[1:5,])` prints to:
#            gre      gpa
#       [1,]   1 0.9736607
#       [2,]   1 0.9751565
#       [3,]   1 0.9820138
#       [4,]   1 0.9604562
#       [5,]   1 0.9493097
## Do not modify this line!

sigmoid<-function(x){
  S = 1 / (1 + exp(-x))
```

```
 return(S)
}

# 3. Define a function `find_grad()` that
#       - Takes an input matrix `X`, a vector `y` and a vector `theta`.
#       - Returns a matrix for the gradient defined in the Newton's Method
#         section at https://stanford.io/36RbPAj.
#         - `m` is the length of `y`.
#         - You will need the `sigmoid()` function you defined previously.
#         - You may need to reorder the terms in the formula as we are doing
#           matrix multiplications (`%*%`) here. Also, it is always helpful to
#           print out the dimensions of the components before gluing them up.
#    To check your result, `find_grad(X, y, rep(0, 2))` prints to:
#             [,1]
#    gre 97.350000
#    gpa  0.587125
## Do not modify this line!
find_grad<-function(X,y,theta){
 m<-length(y)
 ysx<-X%*%theta
 h<-sigmoid(ysx)
 gradient<-(1/m)*t(h-(y))%*%(X)
 return(t(gradient))
}
#也是照着公式写，但是h(x^i)那个部分其实已经计算出，所以可以直接使用

# 4. Define a function `find_cost()` that
#       - Takes an input matrix `X`, a vector `y` and a vector `theta`.
#       - Returns a float for the cost defined in the Newton's Method
#         section at https://stanford.io/36RbPAj.
#         - `m` is the length of `y`.
#         - You will need the `sigmoid()` function you defined previously.
#         - Again, pay attention to matrix multiplications.
#    To check your result, `find_cost(X, y, rep(0, 2))` prints to:
#    [1] 0.6931472
## Do not modify this line!
find_cost<-function(X,y,theta){
 m<-length(y)
 ysx<-X%*%theta
 h<-sigmoid(ysx)
 cost<-(1/m)*(-y*log(h)-(1-y)*log(1-h))
 return(sum(cost))
}
#同样照着公式写，问题不大
# 5. Define a function `newton()` that
#       - Takes an input matrix `X`, a vector `y`, a float `threshold`
#         defaulted to `1e-5`, and an integer `max_t` defaulted to `50`.
#       - Implements the algorithm.
#       - Returns a matrix `theta` for the final estimates (`theta` should
#         contain a bias estimate as well).
#    The skeleton codes for the function follow as below:
```

```
#
#   newton <- function(X, y, threshold = 1e-5, max_t = 50) {
#     # Append a bias column of ones to the left of X
#     X <- cbind(rep(...), X)
#
#     # Initialize theta to be a vector of zeros with the same length as ncol(X).
#     theta <- rep(...)
#
#     t <- 0
#     while (t < max_t) {
#       t <- t + 1
#
#       # Compute gradients and the Hessian
#       g <- find_grad(...)
#       H <- hessian(find_cost, theta, method = "complex", X = X, y = y)
#
#       # Break the algorithm when gradient change (2-norm of g) < threshold
#       if (... < threshold) {
#         break
#       }
#
#       # Update theta using the update rule in the Newton's Method section
#       # at https://stanford.io/36RbPAj.
#       # Hint: use `ginv()` to find the inverse of the Hessian.
#       theta <- ...
#     }
#     return(theta)
#   }
#
#   Apply your function `newton()` to find the coefficient estimates for `y` vs.
#   `X`. Store the result into a variable `estimate_newton`.
#   To check your result, `round(estimate_newton, 2)` prints to:
#         [,1]
#   [1,] -4.95
#   [2,]  0.00
#   [3,]  0.75
## Do not modify this line!
  newton <- function(X, y, threshold = 1e-5, max_t = 50) {
    # Append a bias column of ones to the left of X
    X <- cbind(rep(1,nrow(X)), X)

    # Initialize theta to be a vector of zeros with the same length as ncol(X).
    theta <- rep(0,ncol(X))#注意这个地方*要符合规则

    t <- 0
    while (t < max_t) {
      t <- t + 1

      # Compute gradients and the Hessian
      g <- find_grad(X,y,theta)
      H <- hessian(find_cost, theta, method = "complex", X = X, y = y)
```

```
    # Break the algorithm when gradient change (2-norm of g) < threshold
    if (norm(g,type=c("2")) < threshold) {
      break
    }

    # Update theta using the update rule in the Newton's Method section
    # at https://stanford.io/36RbPAj.
    # Hint: use `ginv()` to find the inverse of the Hessian.
    theta <- theta-ginv(H)%*%(g)
  }
  return(theta)
}
#还是照着公式写
estimate_newton=newton(X,y,1e-5,50)
# 6. Finally, let's check if our algorithm is correct by comparing it against
#   the build-in method.
#   Create an `lm` object `logit` that fits `admit` against `gre` and `gpa`
#   using `glm(family = "binomial")`.
#   Create a data frame `estimates` of size 3 x 3 with columns `term`, `estimate`
#   and `estimate_newton`, where
#     - `term` categorizes the coefficients.
#     - `estimate` are the estimates in `logit`.
#     - `estimate_newton` are the estimates in `estimate_newton`.
#   To do this, you can use:
#     - `broom::tidy()` to collect the coefficients in `logit`.
#     - `dplyr::select()` to select the desired columns.
#     - `cbind()` to bind the result with `estimate_newton`.
#   To check your result, `estimates %>% mutate_at(c(2, 3), round, 2)` prints to:
#          term estimate estimate_newton
#   1 (Intercept)    -4.95         -4.95
#   2       gre     0.00          0.00
#   3       gpa     0.75          0.75
#   We can see that our algorithm is correct.
## Do not modify this line!
  logit<-glm(admit~gre+gpa,data=data,family = "binomial")
  estimates=broom::tidy(logit)%>%dplyr::select(term,estimate)%>%
cbind(estimate_newton)


# HW9: Spotify song dataset
#
# In this exercise, we will explore the distribution of song attributes
# in different popularity groups.
#
# 1. Load the packages `tidyverse`, `lubridate`.
#   Use `read_csv()` to load the file `spotify.csv`
#   located in the folder `data`.
#   Add popularity label to these songs by doing the following:
#    - use `mutate()` to add column `popularity_label`:
#     - use `cut()` to popularity and set `breaks` to `c(0, 30, 70, 100)`.
#     - use `fct_recode()` to set popularity labels to different intervals:
#       `unpopular = "(0,30]"`,
```

```
#       `fair = "(30,70]"`,
#       `popular = "(70,100]"`.
#    - drop `NA` values uysing `drop_na()`.
#   `spotify` should print to:
#   # A tibble: 51,334 x 18
#   artist_name track_id track_name acousticness danceability
#   <chr>      <chr>    <chr>           <dbl>       <dbl>
#   1 Marlon Wil… 48LQlnC… Can I Cal…       0.174        0.523
#   2 Colorblind… 6VPo6yS… Old Cadil…       0.0203       0.811
#   3 Armin van … 6F5JDmp… A State O…       0.152        0.511
#   4 Johann Seb… 6hWhZGZ… Goldberg …       0.977        0.122
#   5 Frdric C…   2CXfRXB… Scherzo N…       0.992        0.318
#   6 Daze        6pss1TM… Oh!              0.0392       0.722
#   7 Kane Brown  3TclTDy… Live Fore…       0.687        0.709
#   8 Daniela Ba… 2lcdI3W… Por el Bo…       0.108        0.912
#   9 Harvey      5VdRbSK… Flow             0.0838       0.233
#   10 The Sheepd… 59jWpoY… You Got t…      0.17         0.642
#   # … with 51,324 more rows, and 13 more variables: duration_ms <dbl>,
#   #   energy <dbl>, instrumentalness <dbl>, key <dbl>, liveness <dbl>,
#   #   loudness <dbl>, mode <dbl>, speechiness <dbl>, tempo <dbl>,
#   #   time_signature <dbl>, valence <dbl>, popularity <dbl>,
#   #   popularity_label <fct>
## Do not modify this line!
library(readr)
library(tidyverse)
library(lubridate)
data <- read_csv("data/spotify.csv")
spotify <- data %>% mutate(popularity_label=cut(popularity, breaks = c(0, 30, 70, 100)))%>%

mutate(popularity_label=fct_recode(popularity_label,"unpopular"="(0,30]","fair"="(30,70]",
"popular"="(70,100]"))%>% drop_na()


# 2. Load `ggplot2`.
#   Draw histogram of danceability to see the different distributions of
#   danceability among different level of popularity
#   To do that, you can use :
#    - `ggplot()` to initialize plot object.
#    - `geom_histogram()` to add the histogram plot:
#      - set `aes` to `aes(x = danceability, y = ..density..)`
#      - set `binwidth` to 0.04.
#      - set `color` to `black` and `fill` to `orange`.
#    - `geom_density()` to add kernel density estimation fit curve:
#      - set `aes` to `aes(x = danceability)`.
#      - set `kernel` to `gaussian`.
#      - set `bw` to 0.06.
#      - set `color` to black.
#    - facet it by `popularity_label`.
#    - inside `labs`, set `x` to `"Danceability"` and `y` to `"Density"`,
#      `title` to `"Danceability distribution varies with popularity"`,
#      `subtitle` to `"Popular songs are more skewed to have high danceability"`.
#    - `theme_light()` to use light theme.
```

```
#   Save the plot to `danceability_dist`.
## Do not modify this line!
library(ggplot2)
danceability_dist <- ggplot(data=spotify)+
 geom_histogram(aes(x = danceability, y = ..density..),binwidth=0.04, color="black",
fill="orange")+
 geom_density(aes(x = danceability), kernel="gaussian", bw=0.06, color="black")+
 facet_wrap(~popularity_label)+
 labs(x="Danceability",
     y="Density",
     title="Danceability distribution varies with popularity",
     subtitle="Popular songs are more skewed to have high danceability")+
 theme_light()

# 3. Filter the songs to keep the ones by artist `BTS` using `filter()`
#   and save to `spotify_bts`. It should print to:
#   # A tibble: 46 x 18
#   artist_name track_id track_name acousticness danceability
#   <chr>      <chr>    <chr>           <dbl>       <dbl>
#   1 BTS        0YguyyC… Go Go - J…      0.222         0.837
#   2 BTS        6u1ILVD… "\x8f\xc1…      0.0704        0.61
#   3 BTS        4Bkh5uD… DNA - Jap…      0.00335       0.574
#   4 BTS        4a8guR4… Spring Da…      0.104         0.572
#   5 BTS        3N6te5x… MIC Drop …      0.0136        0.664
#   6 BTS        6aJj9Ol… Not Today…      0.00191       0.598
#   7 BTS        6yGKx4J… Best Of M…      0.0366        0.659
#   8 BTS        6ePyl2n… Intro: Si…      0.554         0.775
#   9 BTS        0qrPZ27… OUTRO : C…      0.864         0.729
#   10 BTS       0usLRFL… "Trivia \…      0.0907        0.723
#   # … with 36 more rows, and 13 more variables: duration_ms <dbl>,
#   #   energy <dbl>, instrumentalness <dbl>, key <dbl>, liveness <dbl>,
#   #   loudness <dbl>, mode <dbl>, speechiness <dbl>, tempo <dbl>,
#   #   time_signature <dbl>, valence <dbl>, popularity <dbl>,
#   #   popularity_label <fct>
## Do not modify this line!
spotify_bts <- spotify %>% filter(artist_name=="BTS")


# 4. We now want to observe the tempo distribution of hiphop artist. We will
#   fit a normal distribution.
#   Define function `calc_normal_param <- function(data)`, in which `data`
#   should be a vector. `calc_normal_param` should return a vector consisting
#   of the two normal parameters `mean` and `sigma` of `data`:
#    - to get the first parameter mean, use `mean(x)`
#    - to calculate the estimated variance, use formula `E[x^2] - E[x]^2`.
#      (Note: `E[x^2]`` can be calculated by `mean(data ** 2)`)
#   For example, `calc_normal_param(c(0, 1, 2))` should return `[1] 1 0.6666667`.
#   Calculate the parameters of `tempo` of `spotify_bts` and save the result
#   to `tempo_bts`.
## Do not modify this line!
calc_normal_param <- function(data){
 mean <- mean(data)
```

```
 var <- (mean(data^2)-mean(data)^2)
 return(c(mean, var))
}
tempo_bts <-calc_normal_param(spotify_bts$tempo)

# 5. Visualize the performance of our parameterized fit.
#    To do that:
#    - Create a list `dparams_tempo`, with element `mean` set to `tempo_bts[1]`
#      and element `sd` set to `sqrt(tempo_bts[2])`.
#    - Draw qq-plot to see how well the above normal distribution fit to BTS tempo:
#      - `ggplot()` on data `spotify_bts` with `aes(sample = tempo)`.
#      - `geom_qq()` to plot the QQ-plot, with `distribution` set to `qnorm`
#        and `dparams` set to dparams_tempo.
#      - `stat_qq_line()` to plot fitted line with `distribution` set to `qnorm`,
#        `dparams` set to `dparams_tempo` and `color` to `"red"`.
#      - inside `labs()`, set `title` to `"Normal distribution fit for BTS tempo"`,
#        `subtitle` to `"Moment parameter estimation fits tempo well"`,
#        `x` to `"normal distributed value"` and `y` to `"sample value"`.
#      - `theme_light()` to use light theme.
#    Save the generated plot object to `qq_plot_tempo`.
## Do not modify this line!
dparams_tempo <- list(mean=tempo_bts[1], sd=sqrt(tempo_bts[2]))
qq_plot_tempo <- ggplot(data=spotify_bts, aes(sample = tempo))+
 geom_qq(distribution=qnorm,dparams=dparams_tempo)+
 stat_qq_line(distribution=qnorm,dparams=dparams_tempo,color="red")+
 labs(title="Normal distribution fit for BTS tempo",
      subtitle="Moment parameter estimation fits tempo well",
      x="normal distributed value",
      y= "sample value")+
 theme_light()

# 6. Next, we will fit a gamma distribution to `liveness` of popular songs.
#    First, create the following functions :
#    - Moment parameter estimation function:
#      `calc_gamma_param <- function(data)`, in which `data` is a vector,
#      and returns the estimated parameters for the gamma distribution
#      (shape and scale) using the moments from the data.
#      Note that the output should be a vector with names `a` and `s`.
#      For example, `calc_gamma_param(c(1, 2))` should return:
#              a         s
#       4.5000000 0.3333333
#      (referenced formula: shape = mean^2 / var, scale = var / mean)
#    - Log likelihood function:
#       `nll_gamma_factory <- function(x)`, in which `x` is a vector, and
#       returns the sum of log likelihood function of all the gamma-distributed
#       data point in `x`. The returned likelihood function should take `par` as input,
#       which is a vector of the two parameters for gamma distribution and
#       return the calculated log likelihood of input data `x`.
#       Inside the function:
#       - declare `a <- par[1]` as the shape parameter.
#       - declare `s <- par[2]` as the scale parameter.
#       - calculate `n` to be length of the input vector using `length()`.
```

```
#      - calculate `sx` to be sum of the input vector using `sum()`.
#      - calculate `slx` to be sum of the log of each data point in input vector
#        using `sum()` and `log()`.
#      - declare `function(par)` which calculates the log likelihood of data by
#        the following formula:
#        - `(a-1) * slx + n * lgamma(a) + n * a * log(s) + sx / s`
#    - MLE function:
#      `par_gamma` takes data and return the parameters optimized by maximizing
#      the log likelihood of the data.
#      Inside the function:
#      - initialize the parameter `par0` by calling `calc_gamma_param(x)`.
#      - define likelihood function `nll_gamma_x` by calling `nll_gamma_factory(x)`
#      - use `optim()` to optimize the parameter, and set `par0` as first argument,
#        `nll_gamma_x` as second argument, `lower` to 0 and `method` to ""L-BFGS-B".
#      - return the optimized parameter by `optim(..)$par`.
#       For example, `par_gamma(c(1, 2))` should return:
#             a        s
#      8.6502954 0.1734083
## Do not modify this line!
calc_gamma_param <- function(x){
  mean <- mean(x)
  var <- var(x)
  return (c("a" = mean^2/var, "s" = var/mean))
}
nll_gamma_factory <- function(x){
  function(par){
    force(x)
    a <- par[1]
    s <- par[2]
    n <- length(x)
    sx <- sum(x)
    slx <- sum(log(x))
    l=-(a-1) * slx + n * lgamma(a) + n * a * log(s) + sx / s
    return (l)
  }
}
par_gamma <- function(x){
  par0 <- calc_gamma_param(x)
  nll_gamma_x <- nll_gamma_factory(x)
  return (optim(par0, nll_gamma_x, lower = 0, method = "L-BFGS-B")$par)
}
# 7. Now, let's fit the predictions using the above methods.
#    To do that, you can :
#     - filter songs of `spotify` with `popularity_label = "popular"`
#       and save the resulted tibble to `spotify_pop`.
#     - calculate parameter `liveness_pop1` using `par_gamma` (mle estimate).
#     - calculate parameter `liveness_pop2` using `calc_gamma_param`.
#     - create density function `liveness_pop_density1`using
#       `partial(dgamma, shape =., scale=.)` setting `shape` to
#       `liveness_pop1[1]` and `scale` to `liveness_pop1[2]`.
#     - create density function `liveness_pop_density2`using
#       `partial(dgamma, shape =., scale=.)` setting `shape` to
```

```
#     `liveness_pop2[1]` and `scale` to `liveness_pop2[2]`.
#   `spotify_pop` should print to:
#   # A tibble: 1,758 x 18
#   artist_name track_id track_name acousticness danceability
#   <chr>      <chr>    <chr>           <dbl>       <dbl>
#   1 Zé Neto & … 1fyhyOy… Status Qu…       0.59         0.580
#   2 Drake       3mvYQKm… After Dar…       0.0414       0.686
#   3 Julia Mich… 2OvV4Nj… Jump (fea…       0.25         0.654
#   4 Meghan Tra… 7fCNUWi… No Excuses       0.0224       0.827
#   5 Jorge & Ma… 2U94QDS… Terra Sem…       0.357        0.676
#   6 Rels B      5I2I0xd… Buenos Ge…       0.486        0.804
#   7 Capo Plaza  093RgZ7… Tesla (fe…       0.316        0.854
#   8 Mc Davi     1R1Hmdu… Bonita, L…       0.306        0.712
#   9 Tedua       1Oou7m2… Vertigini        0.164        0.579
#   10 Gemitaiz    1Lq5Apq… Davide (f…       0.136        0.797
#   # … with 1,748 more rows, and 13 more variables: duration_ms <dbl>,
#   #energy <dbl>, instrumentalness <dbl>, key <dbl>, liveness <dbl>,
#   #loudness <dbl>, mode <dbl>, speechiness <dbl>, tempo <dbl>,
#   #time_signature <dbl>, valence <dbl>, popularity <dbl>,
#   #popularity_label <fct>
## Do not modify this line!
spotify_pop <- spotify%>%filter(popularity_label == "popular")
liveness_pop1 <- par_gamma(spotify_pop$liveness)
liveness_pop2 <-calc_gamma_param(spotify_pop$liveness)
liveness_pop_density1 <- partial(dgamma, shape =liveness_pop1[1],
scale=liveness_pop1[2])
liveness_pop_density2 <- partial(dgamma, shape =liveness_pop2[1],
scale=liveness_pop2[2])


# 8. Visualize the fit of gamma distribution to liveness of songs.
#    - `ggplot()` on tibble `spotify_pop`.
#    - `geom_histogram()` to draw liveness distribution:
#      - set `aes` to `aes(x = liveness, y = ..density..)`.
#      - set `binwidth` to 0.02, `color` to "black" and `fill` to "orange".
#    - `geom_density()` to draw fitted kernel density estimation:
#      - set `kernel` to "gaussian".
#      - set `bw` to 0.06.
#      - set `color` to "red" and `fill` to "red".
#      - set `alpha` to 0.5.
#    - Draw two parameterized fitted curve using `stat_function`:
#      - set mle curve with `color` set to `"yellow"`.
#      - set moments curve with `color` set to `"green"`.
#    - `labs()` to set `x` to "Liveness", `y` to `"Density"``,
#      `title` to  `"Liveness distribution and density estimates"` and `subtitle` to
#      `"MLE fits better than moments method"`.
#      - `theme_light()` to set a light background.
#   Save the ggplot object into `liveness_gamma_plot`.
## Do not modify this line!
liveness_gamma_plot <- ggplot(data=spotify_pop,aes(x=liveness))+
  geom_histogram(aes(y = ..density..), binwidth = 0.02, color="black", fill="orange")+
  geom_density(kernel="gaussian", bw=0.06, color="red", fill="red")+
```

```
 stat_function(fun = liveness_pop_density1, color = "yellow")+
 stat_function(fun = liveness_pop_density2, color = "green")+
 labs(x="Liveness",
     y="Density",
     title="Liveness distribution and density estimates",
     subtitle="MLE fits better than moments method")+
 theme_light()


# 9. Visualize the performance of our mle fit.
#    To do that:
#    - Create a list `dparams_liveness`, with element `shape` set to
#      `liveness_pop1[1]` and element `scale` set to `liveness_pop1[2]`).
#    - Draw qq-plot to see how well the above normal distribution fit to BTS tempo:
#      - `ggplot()` on tibble `spotify_pop` with variables `aes(sample=liveness)`.
#      - `geom_qq()` to plot the QQ-plot, with `distribution` set to `qgamma`
#        and `dparams` set to dparams_liveness
#      - `stat_qq_line()` to plot fitted line with `distribution` set to `qgamma`,
#        `dparams` set to `dparams_liveness` and `color` to "red".
#      - inside `labs()`, set `title` to "Gamma distribution fit for song liveness",
#        `subtitle` to `"Fits liveness well only for small values"`,
#        `x` to `"gamma distributed value"` and `y` to`"sample value"`.
#      - `theme_light()` to use light theme.
#    Save the generated plot object to `qq_plot_liveness`.
## Do not modify this line!
dparams_liveness <- list(shape=liveness_pop1[1],scale=liveness_pop1[2])
qq_plot_liveness <- ggplot(data=spotify_pop,aes(sample=liveness))+
 geom_qq(distribution="qgamma",dparams= dparams_liveness)+
 stat_qq_line(distribution="qgamma",dparams= dparams_liveness,color="red")+
 labs(title="Gamma distribution fit for song liveness",
     subtitle="Fits liveness well only for small values",
     x="gamma distributed value",
     y="sample value")+
 theme_light()

# HW9: Quadratic Programming
#
# Suppose we have selected 10 stocks from which to build a portfolio.
# We want to determine how much of each stock to include in our portfolio.
#
# 1. Load the `tidyverse` and `quadprog` packages.
#    Load the monthly stock return data during 2012 and 2013 for stocks
#    `"AAPL"`, `"XOM"`, `"GOOG"`, `"MSFT"`, `"GE"`, `"JNJ"`, `"WMT"`, `"CVX"`,
#    `"PG"`, `"WF"` from path `data/stocks_data.csv` using `read_csv()`.
#    Store the result in tibble `stocks_data`.
#    The first rows of `stocks_data` should be:
#    # A tibble: 24 x 11
#      date        AAPL     XOM    GOOG    MSFT     GE     JNJ
#      <date>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
#    1 2012-01-31  0.127   -0.0120  -0.102    0.138    0.0447   0.00503
#    2 2012-02-29  0.188    0.0330   0.0657   0.0748   0.0182  -0.0126
#    3 2012-03-30  0.105    0.00266  0.0372   0.0164   0.0535   0.0135
```

```
#    4 2012-04-30 -0.0260  -0.00450 -0.0567  -0.00744 -0.0244  -0.0130
#    5 2012-05-31 -0.0107  -0.0893  -0.0397  -0.0884  -0.0250  -0.0410
#    6 2012-06-29  0.0109   0.0883  -0.00136  0.0480   0.0917   0.0822
#    7 2012-07-31  0.0458   0.0150   0.0912  -0.0366  -0.00432  0.0246
#    8 2012-08-31  0.0892   0.00518  0.0823   0.0458  -0.00193 -0.0259
#    9 2012-09-28  0.00280  0.0475   0.101   -0.0344   0.0966   0.0219
#   10 2012-10-31 -0.108   -0.00306 -0.0983  -0.0410  -0.0727   0.0277
#    # … with 14 more rows, and 4 more variables: WMT <dbl>, CVX <dbl>,
#    #   PG <dbl>, WF <dbl>
## Do not modify this line!
library(tidyverse)
library(quadprog)
stocks_data <- read_csv("data/stocks_data.csv")


# 2. Transform `stocks_data` into long form so that it can be
#    used to draw the plot.
#    You can use `pivot_longer()` to transform the data.
#    The new column names are `stock` and `return`.
#    Store the result into tibble `stocks_data_longer`.
#    The first rows of its print should be:
#    # A tibble: 240 x 3
#      date       stock   return
#      <chr>      <chr>    <dbl>
#    1 2012-01-31 AAPL   0.127
#    2 2012-01-31 XOM   -0.0120
#    3 2012-01-31 GOOG  -0.102
#    4 2012-01-31 MSFT   0.138
#    5 2012-01-31 GE     0.0447
#    6 2012-01-31 JNJ    0.00503
#    7 2012-01-31 WMT    0.0268
#    8 2012-01-31 CVX   -0.0308
#    9 2012-01-31 PG    -0.0550
#   10 2012-01-31 WF     0.219
#    # … with 230 more rows
## Do not modify this line!
stocks_data_longer <- stocks_data %>%pivot_longer(cols=2:11,names_to="stock",
values_to="return")


# 3. Draw the monthly stock return data for 10 stocks
#    using `stocks_data_longer` and save it in `stock_return_plot`.
#    Please use:
#    - `ggplot()` to initialize a ggplot object. You can set its arguments
#      `data` and `mapping` to plot the `return` column against `date`.
#      of dataset `stocks_data_longer`, setting `group` to `stock` in `aes()`.
#    - `geom_line()` to draw the lines.
#      - Set `color` to `stock` in `aes()`.
#    - `labs()` to set:
#      - `title` to `"10 stocks monthly return rates"`.
#      - `subtitle` to `"There is significant fluctuation in return rates"`.
#      - `x` to `""`.
```

```
#     - `y` to `"Monthly Return Rate"`.
#    - `theme_light()` to set light theme (i.e. a light background).
#    - `theme()` to rotate the element text of x axis by 90 degrees.
## Do not modify this line!
stock_return_plot <-ggplot(data=stocks_data_longer, aes(x=date, y=return))+
  geom_line(aes(color=stock))+
  labs(title="10 stocks monthly return rates",
       subtitle="There is significant fluctuation in return rates",
       x="",
       y="Monthly Return Rate")+
  theme_light()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))


# 4. To consider the risk of deviations in our portfolio return,
#    define the quadratic form `Q(w) = w^T C w`,
#    where
#    - `C` is the covariance matrix of the returns `r_i`.
#    - `w_i` is the fraction of the portfolio value allocated to asset `i`.
#    To solve the portfolio allocation problem, we'll try to determine
#    the weights `w = (w_1,...,w_{10})` so that the risk function
#    `Q(w)` is minimized.
#    The constraints are:
#    - Normalized `w` i.e.
#      - The sum of `w` should be 1.
#      - Every element of `w` should be between 0 and 1.
#    - A minimum expected monthly return of 1% i.e.
#      the sum of `w_i * r_i` should be greater or equal than 0.01 with `r_i` the
#      average monthly return rate on asset `i`,
#      computed from the data from `stocks_data`.
#    You can use `solve.QP()` to solve this quadratic programming problem
#    and save the result into `sol`.
#    Save the parameters of `solve.QP()` into matrix/vector `Dmat`, `dvec`, `Amat`,
#    and `bvec`.
#    Note that:
#    - The order of the variable i.e. the order of the columns of `Dmat` and `Amat`
#    is `"AAPL"`, `"XOM"`, `"GOOG"`, `"MSFT"`, `"GE"`, `"JNJ"`, `"WMT"`, `"CVX"`,
#    `"PG"`, `"WF"` which is the original order of the `stocks_data`.
#    - `Dmat` should be a `10 * 10` covariance matrix of `stocks_data$return`.
#    - `dvec` should be a zero vector of length 10.
#    - `Amat` should be a `22 * 10` matrix.
#      - The first row makes sure the sum of the variables equals to one.
#      - The second row makes sure the expected monthly return is greater than or equal to
0.01.
#      - The 3rd to 12th rows make sure each variable is greater than or equal to 0.
#      - The 13th to 22nd rows make sure each variable is less than or equal to 1.
#        Remember that the symbol of the equation should be `greater than or equal to`.
#    - `bvec` should be a vector of length 22. It should be set according to `Amat`.
#    Apply the parameters to `solve.QP()` and set `meq = 1`. Remember to use `t(Amat)`.
## Do not modify this line!
Dmat = cov(stocks_data[2:11])
dvec = rep(0, 10)
```

```
Amat = rbind(rep(1, 10), colMeans(stocks_data[2:11]),
          diag(10), -diag(10))
bvec = c(1, 0.01, rep(0,10), rep(-1,10))
sol = solve.QP(Dmat, dvec, t(Amat), bvec, meq = 1)
```
#与课件不同的是colMeans(stocks_data[2:11])，这里没有直接提供

```
# 5. Create a tibble `portfolio` to store the proportion of the stocks.
#    You can use:
#    - `tibble()` to create the tibble.
#      - The fist column `stock`  stores the names of 10 stocks
#      - The second column `proportion` stores the proportion of the
#        according stocks.
#        This data comes from `sol[[1]]` and should be rounded to
#        3 digits using `round()`.
#        Note: Use `sol[[1]]]` instead of `sol$solution` to aviod being
#        identified as cheating.
#    - `mutate()` and `fct_reorder()` to reorder the `stock` column
#      according to `proportion` column.
#    It should print to :
#    # A tibble: 10 x 2
#      stock proportion
#      <fct>      <dbl>
#    1 AAPL       0.01
#    2 XOM        0.162
#    3 GOOG       0
#    # … with 8 more rows
## Do not modify this line!
portfolio<-tibble(stock=names(stocks_data)[2:11],proportion = round(sol[[1]], 3))%>%
  mutate(stock = fct_reorder(stock, proportion))


# 6. Draw the horizontal bar chart for the proportions and save it
#    into `portfolio_plot` using `portfolio`.
#    Please use:
#    - `ggplot()` to initialize a ggplot object. You can set its arguments
#      `data` and `mapping` to plot the `proportion` column of the dataset.
#      Use `aes()` to set parameters `mapping`.
#    - `geom_bar()` to draw the lines.
#      - Set `stat` to `identity`.
#      - Set `fill` to `blue`.
#    - `labs()` to set:
#      - `title` to `"Portfolio"`.
#      - `subtitle` to `"WMT has the highest proportion"`.
#      - `x` to `""`.
#      - `y` to `"Proportion"`.
#    - `theme_light()` to set light background.
## Do not modify this line!

portfolio_plot<-ggplot(data=portfolio,aes(stock,proportion))+
  geom_bar(stat="identity",fill="blue")+
  coord_flip()+
  labs(title="Portfolio",
```

```
    subtitle="WMT has the highest proportion",
    x="",
    y="Portfolio")+
  theme_light()

# HW9: Fit a truncated Poisson and estimate its parameter
#
# This exercise is based on the results of a well-known social studies paper
# "Independent Discovery in Science and Technology: A Closer Look at the
# Poisson Distribution".
#
# Abstract: Social determinists have argued that the occurrence of independent
# discoveries and inventions demonstrates the inevitability of
# techno-scientific progress. Yet the frequency of such multiples may be
# adequately predicted by a probabilistic model, especially the Poisson model
# suggested by Price. A detailed inquiry reveals that the Poisson distribution
# can predict almost all the observed variation in the frequency distribution
# of multiples collected by Merton, and by Ogburn and Thomas. This study
# further indicates that: (a) the number of observed multiples may be greatly
# underestimated, particularly those involving few independent contributors,
# (b) discoveries and inventions are not sufficiently probable to avoid a large
# proportion of total failures, and hence techno-scientific advance is to a
# large measure indeterminate; (c) chance or 'luck' seems to play such a major
# part that the 'great genius' theory is no more tenable than the social
# deterministic theory.
#
# We will examine some data that contains one point per scientific discovery
# that originated from two or more simultaneous independent discoveries, fit a
# truncated Poisson model, estimate its parameter and evaluate goodness-of-fit.
#
# 1. Load the `readr` and `magrittr` packages.
#    Read in the file `"data/discoveries.txt"`, convert the vector to
#    numeric values and assign the resulting vector to variable `discoveries`.
#    Each value is the number of simultaneous independent studies that lead to
#    a major scientific discovery.
#    To do so, you can do the following:
#    - read in the file at `"data/discoveries.txt"` using `read_lines()`,
#    - call `as_numeric()` to convert the string vector to a numeric vector
#    `discoveries` should print to:
#      [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#     [34] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#     [67] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#    [100] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#    [133] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#    [166] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
#    [199] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4
#    [232] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 6 6 6 6 6 6 6 7 9 9
## Do not modify this line!

library(readr)
library(magrittr)
discoveries<-read_lines("data/discoveries.txt")%>%
```

```
 as.numeric()

# 2. Load the `tibble` and `ggplot2` packages.
#    Create a tibble with one column `"discoveries"` from the `discoveries`
#    vector and call it `discoveries_tibble` using `tibble()`.
#    Plot a histogram of the simultaneous discoveries using the tibble and
#    assign the plot to `discoveries_hist`.
#    To do so, you can:
#    - call `ggplot()` on `discoveries_tibble` with the column in the
#      aesthetic
#    - add a call to `geom_histogram()`, setting `binwidth` to 0.25
#    - use `labs()` for the following labels:
#      - `x = "Multi-disciplinary discovery co-occurrences"``,
#      - `y = "Count"``,
#      - `title = "Multiple simultaneous co-occurrences become rare beyond 2"`
#    - add `theme_light()`
#    `discoveries_tibble` should print to:
#    # A tibble: 264 x 1
#       discoveries
#             <dbl>
#    1           2
#    2           2
#    3           2
#    4           2
#    5           2
#    6           2
#    7           2
#    8           2
#    9           2
#   10           2
#    # … with 254 more rows
## Do not modify this line!
library(tibble)
library(ggplot2)
discoveries_tibble<-tibble(discoveries=discoveries)
discoveries_hist<-ggplot(discoveries_tibble,aes(x=discoveries))+
  geom_histogram(binwidth=0.25)+
  labs(x = "Multi-disciplinary discovery co-occurrences",
       y = "Count",
       title = "Multiple simultaneous co-occurrences become rare beyond 2")+
  theme_light()


# 3. We will fit a truncated Poisson model to this data - here meaning a
#    Poisson model that can only take values starting at 2.
#    Recall that the usual Poisson distribution for variable `X`, with
#    parameter `lambda`, is defined by:
#    For `k >= 0` (k is an integer) : `P(X = k) = exp(-lambda)*lambda^k/(k!)`
#    We will model our truncated Poisson by the distribution:
#    For `k >= 2`: `P(X = k) = correction(lambda) * exp(-lambda)*lambda^k/(k!)`
#    where `correction(lambda)` is a function of `lambda` chosen such that the
#    distribution is a probability distribution.
```

```
#   Do the following:
#   - find the expression of `correction(lambda)` and create the
#     `correction` function with argument `lambda` that returns
#     `correction(lambda)`.
#     Hint: use the condition that the sum of probabilites must add up to 1.
#   - write the truncated density function `dtrunc_poisson()` with arguments
#     `x` and `lambda` that returns the value of the density of the truncated
#     poisson with parameter lambda
#     Hint: use `dpois()` and `correction()`
## Do not modify this line!
correction<-function(lambda){
 as.numeric(1/(1-exp(-lambda)-lambda*exp(-lambda)))
}#即减去x=1,x=2
dtrunc_poisson<-function(x,lambda){
 correction(lambda)*dpois(x,lambda)
}

# 4. Write a function factory `factory_trunc_poisson()` with argument vector
#   `x`. In its body, it should:
#   - assign the number of data points in `x` to `n`,
#   - assign the sum of all data points in `x` to `S`,
#   - return the negative log likelihood function with argument `lambda` that
#     returns the value of the negative log-likelihood of the present data,
#     which will depend on the data only through `S` and `n`, up to a
#     constant. There shouldn't be any term that doesn't depend on lambda.
#     Its form should be: `n * log( g(lambda) ) - S * h(lambda)` where it is
#     up to you to find the expression of `g` and `h`.
#   Assign the function output of `factory_trunc_poisson(discoveries)` to
#   `nlle_trunc_poisson()`.
#   Example : `factory_trunc_poisson(c(3, 2, 4))(2)` should return `[1] -1.800982`
#   and `factory_trunc_poisson(c(3, 2, 4))(10)` should return `[1] 9.275236`.
#   It is more probable to have observations 3, 2, and 4 with mean 2 instead of 10.
## Do not modify this line!
factory_trunc_poisson<-function(x){
 n=length(x)
 S=sum(x)
 function(lambda){
  -(n*log(correction(lambda))+log(lambda)*S-n*lambda)}#truc 的 log-likihoode取负值
}
nll_trunc_poisson<-factory_trunc_poisson(discoveries)

# 5. Load the `dplyr` package.
#   Plot the negative log-likelihood as a function of `lambda` and assign the
#   plot to `nll_trunc_poisson_plot`.
#   To do so, you can:
#   - create a tibble containing a column `lambda` with linearly spaced values
#     from `0.01` to `10`, with a step of `0.01` (using `tibble()` and
#     `seq()`),
#   - add a column containing the negative log-likelihood values for each
#     `lambda` using `mutate()` and `nll_trunc_poisson()` defined in question
#     4,
```

```
#    - feed the tibble to `ggplot()`,
#    - call `geom_line()` to plot the function,
#    - use `labs()` for the following labels:
#      - `y = "Negative loglikelihood"``,
#      - `title = "The negative loglikelihood has a unique global minimum"``,
#      - `subtitle = "The minimum is reached for a lambda between 1 and 2"``
#    - add `theme_light()`
## Do not modify this line!
library(dplyr)
library(dplyr)
nll_trunc_poisson_plot<-tibble(lambda=seq(0.01,10,0.01))%>%
  mutate(value=nll_trunc_poisson(lambda))%>%
  ggplot(aes(x=lambda,y=value))+
  geom_line()+
  labs(y = "Negative loglikelihood",
       title = "The negative loglikelihood has a unique global minimum",
       subtitle = "The minimum is reached for a lambda between 1 and 2")+
  theme_light()



# 6. Find the argmin of the negative log-likelihood and assign the resulting
#    number (ie. the Maximum Likelihood Estimator) to `mle_trunc_poisson`.
#    To do so you can:
#    - use `optimize()` with an appropriate `interval` (the plot in 5 should
#      give you ideas),
#    - fetch the `argmin` using `$minimum`
#    Compute a 95% confidence interval for the estimator centered on
#    `mle_trunc_poisson` using the observed Fisher information. Assign the
#    vector containing the lower and upper bounds of the interval to
#    `ci_mle_trunc_poisson`. To do so, you should:
#    - compute the observed Fisher information (and call it `fisher_info`) -
#      whose expression we give to you at the end of the question,
#    - retrieve the number of data points using `length()` on `discoveries`,
#    - use `qnorm()` with a `0.975` quantile and divide it by the square root
#      of `n * fisher_info` to compute the distance of the bounds to the center
#      (`mle_trunc_poisson`)
#    - create a vector by respectively substracting and adding the value
#      computed in the previous step to the `mle_trunc_poisson` and assign it
#      to `ci_mle_trunc_poisson`
#    $$fisher_info = \frac{(1-exp(-mle))^2 - mle^{2}*exp(-mle)}{mle*(1-exp(-mle) - mle*exp(-
mle))^2}$$
#    where `mle` is the Maximum Likelihood Estimator (ie. `mle_trunc_poisson`
#    in this exercise)
#    Note: the fisher info is equal to the second derivative of the negative
#    log-likelihood evaluated at lambda equal to `mle_trunc_poisson`,
## Do not modify this line!
mle_trunc_poisson<-optimize(nll_trunc_poisson,interval=c(0.01,10))$minimum
fisher_info<-{(1-exp(-mle_trunc_poisson))^2 - mle_trunc_poisson^{2}*exp(-
mle_trunc_poisson)}/{mle_trunc_poisson*(1-exp(-mle_trunc_poisson) -
mle_trunc_poisson*exp(-mle_trunc_poisson))^2}
n<-length(discoveries)
```

```
ci_mle_trunc_poisson<-c(mle_trunc_poisson-
qnorm(0.975)/sqrt(n*fisher_info),mle_trunc_poisson+qnorm(0.975)/sqrt(n*fisher_info))
# 7. We will now fit a truncated Poisson using the MLE and evaluate
#    goodness-of-fit using a Chi-Square Test.
#    Generate a vector of theoretical probabilities for the truncated Poisson
#    for values from 2 to 9 and assign the vector to `trunc_poisson_to_9`.
#    To do so, you can:
#    - call `dtrunc_poisson()` defined in question 3 for values in `2:9` with
#      the MLE
#    Compute the rest of the probability mass (ie. `P(X > 9)`) and assign the
#    probability to `prob_mass_beyond_9`.
#    Hint: you can use  `trunc_poisson_to_9` and the fact that the
#    probabilities sum to 1.
#    Add `prob_mass_beyond_9` to the `trunc_poisson_to_9` vector and name the
#    resulting vector `trunc_poisson`.
#    `trunc_poisson` should print to:
#    [1] 5.924706e-01 2.761689e-01 9.654815e-02 2.700245e-02 6.293339e-03
#    [6] 1.257223e-03 2.197615e-04 3.414588e-05 5.461141e-06
## Do not modify this line!
trunc_poisson_to_9<-dtrunc_poisson(2:9,mle_trunc_poisson)#with MLE,所以就是上面那
项
prob_mass_beyond_9<-1-sum(dtrunc_poisson(2:9,mle_trunc_poisson))
trunc_poisson<-c(trunc_poisson_to_9,prob_mass_beyond_9)


# 8. Compute the counts of each value in `discoveries` and include counts of 8
#    and values higher than 9. Assign the resulting vector to
#    `discoveries_counts`.
#    To do so, you can:
#    - call `count()` on the column of `discoveries_tibble`,
#    - add two rows to the resulting tibble, using `add_row():
#      - one with `discoveries=8` and `n=0` (no observations of 8 simultaneous
#        discoveries)
#      - one with `discoveries=10` and `n=0` (no observations of more than 9
#        simultaneous discoveries)
#    - sort by `discoveries` using `arrange()`,
#    - call `pull(n)` to obtain the count vector
#    `discoveries_count` should print to:
#    [1] 179  51  17   6   8   1   0   2   0
## Do not modify this line!
discoveries_count<-discoveries_tibble%>%
  count(discoveries)%>%
  add_row(discoveries=8,n=0)%>%
  add_row(discoveries=10,n=0)%>%
  arrange(discoveries)%>%
  pull(n)


# 9. Set the random seed to `0` using `set.seed()`.
#    Run a Chi-Square test comparing actual frequencies and the theoretical
#    truncated Poisson distribution using `chisq.test()`, `discoveries_count`
```

```
#    and `trunc_poisson` and setting `simulate.p.value` to `TRUE` (the p-value
#    will be computed by Monte-Carlo simulation - this is where there is
#    stochasticity and why we need to set the seed before the call).
#    Retrieve its p-value and assign it to `chisq_p_value`.
#    You should get a p-value significantly smaller than 0.05. This tells us
#    that we can reject the hypothesis that the distribution fits the data
#    well with high confidence. It seems that the paper's choice of
#    distribution to model the data at hand is maybe not the best to explain
#    its trends. Further goodness-of-fit tests could confirm this intuition.
#
## Do not modify this line!
set.seed(0)
seed<-.Random.seed
chisq_p_value<-chisq.test(x=discoveries_count,p=trunc_poisson,simulate.p.value =
TRUE)$p.value


# HW9: Implement Newton-Raphson and observe on a function
#
# In this exercise, you will implement the Newton-Raphson algorithm from
# scratch and use it to find the roots - values for which the function is
# zero - of a real-valued function.
#
# 1. Implement a function that takes in a real number `x` and returns
#    `exp(x*x/(x*x + 2*x + 5))*tanh(x)*sin(x)/(x*(1 + .5/abs(x)))` and call it
#    `f`.
## Do not modify this line!
f<-function(x){
  exp(x*x/(x*x + 2*x + 5))*tanh(x)*sin(x)/(x*(1 + .5/abs(x)))
}


# 2. Load the `tibble`, `dplyr` and `ggplot2` packages.
#    Plot the graph of `f` for values between `-30` and `30` (points separated
#    by a step of `0.01`) and assign it to `f_plot`.
#    To do so, you can:
#    - create a tibble with `x` points between `-30` and `30` with a step size
#      of `0.01`, using `tibble()` and `seq()`,
#    - add a column with values `f(x)` using `mutate()` and `f()`,
#    - call `ggplot()` on the tibble,
#    - add `geom_line()`,
#    - add a blue vertical line of equation `x = -3` using `geom_vline()`,
#    - add the following labels (using `labs()`):
#      - `y = "f(x)"`,
#      - `title = "Graph of f"`,
#      - `subtitle = "There are many roots, including one near -3"`
#    - add `theme_light()`
## Do not modify this line!
library(tibble)
library(dplyr)
library(ggplot2)
f_plot<-tibble(x=seq(-30,30,0.01))%>%
```

```
 mutate(`f(x)`=f(x))%>%
 ggplot(aes(x=x,y=`f(x)`))+
 geom_line()+
 geom_vline(xintercept = -3,color="blue")+
 labs(y = "f(x)",
     title = "Graph of f",
     subtitle = "There are many roots, including one near -3")+
 theme_light()


# 3. The Newton-Raphson method produces successive approximations to the roots
#    of a real-valued function. The most basic version starts with an initial
#    guess `x_0` and generates successive points `x_i` such that
#    `x_(i+1) = x_i - f(x_i) / f'(x_i)` until a convergence criterion has been
#    met or a number of iterations has been reached. We might not have access
#    to the derivative `f'` in some cases, so we will approximate `f'(x)` by
#    `(f(x + h) - f(x)) / h` for small `h`.
#    Do the following:
#    - load the `purrr` package,
#    - assign value `1e-7` to variable `h`.
#    - write one step of the Newton-Raphson algorithm. It should be a function
#      called `one_step` that takes in arguments, `f`, `x` (for `x_i`) and
#      `h` and returns the next value `x_(i+1)` using the recursive formula
#      and the approximation of the derivative above.
#    - write a function that runs several steps and returns a tibble with the
#      values obtained at each step. It should take in arguments `f`, `x`, `h`
#      and `n_iterations` (the number of times to call `one_step()`). Call it
#      `multi_step()`. To write its body, you can do the following:
#      - call `accumulate()`, on `n_iterations` replicas of `h`, function
#        `one_step()`, `f = f` and `init = x`,
#      - pass the result tibble to `enframe()` to output a tibble with two
#        columns (`step` and `x`) as below.
#    the call `multi_step(f, 1, h, 2)` should return:
#    # A tibble: 3 x 2
#       step     x
#      <int>  <dbl>
#    1     1  1
#    2     2 -0.400
#    3     3 -0.130
## Do not modify this line!
library(purrr)
h<-1e-7
one_step<-function(f,x,h){
  x = x - f(x) / ((f(x + h) - f(x)) / h)
}
multi_step<-function(f,x,h,n_iterations){
  accumulate(rep(h,n_iterations),one_step,f=f,.init=x)%>%
    enframe(name="step",value="x")
}

# 4. Load the `tidyr` package.
#    Create a tibble storing the output of the first 2 iterations of the
```

```
#   Newton-Raphson algorithm starting with initial guesses `seq(-7, 7, 2)`,
#   `f` and `h` defined in previous questions. Assign the resulting tibble to
#   `newton_steps_tibble`.
#   To do so, you can:
#   - create a tibble with columns `start = seq(-7, 7, 2)` and `steps`
#     using `tibble()`. `steps` should contain nested tibbles created by
#     `multi_step()` with `f`, `h` and `n_iterations = 1` (you can create the
#     nested tibbles from the `start` column using `map()`),
#   - unnest `steps` using `unnest()`,
#   - use `mutate()` to:
#     - add a column `f` containing the values `f(x)`,
#     - turn `step` and `start` into factors
#   `newton_steps_tibble` should print to:
#   # A tibble: 24 x 4
#      start step      x           f
#      <fct> <fct> <dbl>       <dbl>
#    1 -7    1     -7    -0.298
#    2 -7    2     -6.00  0.151
#    3 -7    3     -6.28  0.00295
#    4 -5    1     -5     0.608
#    5 -5    2     -7.09 -0.325
#    6 -5    3     -5.84  0.232
#    7 -3    1     -3    -0.124
#    8 -3    2     -3.14 -0.000706
#    9 -3    3     -3.14 -0.0000000627
#   10 -1    1     -1    -0.549
#   # … with 14 more rows
## Do not modify this line!
library(tidyr)
newton_steps_tibble<-tibble(start = seq(-7, 7, 2),
      steps = map(start,multi_step,f=f,h=h,n_iterations = 2))%>%
  unnest(steps)%>%
  mutate(f=f(x),step=factor(step),start=factor(start))

# 5. Create a plot showing the first 2 iterations of the Newton-Raphson
#   algorithm starting with initial guesses `seq(-7, 7, pi)`, `f` and `h`
#   defined in previous questions. Points corresponding to each iteration
#   should have a different color. Assign the resulting plot to
#   `newton_steps_plot`.
#   To do so, you can:
#   - call `ggplot()` on `newton_steps_tibble` with aesthetic `(x, f)`,
#   - add `stat_function` with `fun = f`,
#   - add `geom_point()` to show each point of `size = 2` and color them by
#     `step`,
#   - add `geom_hline()` with an `alpha` of `0.2` to show the reference `y=0`
#     line with a bit of transparency,
#   - facet the plot by `start` using `facet_wrap()`
#   - add the following labels (using `labs()`):
#     - `y = "f(x)"`,
#     - `title = "The points get closer to roots of f as steps increase"``
#   - add `theme_light()`
## Do not modify this line!
```

```
newton_steps_plot<-ggplot(newton_steps_tibble,aes(x,f))+
 stat_function(fun=f)+
 geom_point(size=2,aes(color=step))+
 geom_hline(alpha=0.2,yintercept = 0)+
 facet_wrap(~start)+
 labs(y = "f(x)",
     title = "The points get closer to roots of f as steps increase")+
 theme_light()


# 6. Write the complete `newton_raphson` function with arguments `f`, `x_0`
#    (float initial guess), `h` and `n_iterations` where `h` is passed default
#    value `1e-7` and `n_iterations` is passed default value `5`.
#    Starting with `i = 1`, while `i <= n_iterations`, the function should call
#    `one_step()` and create the successive guesses, returning the final one.
## Do not modify this line!
newton_raphson<-function(f,x_0,h=1e-7,n_iterations=5){
 i=1
 while(i<=n_iterations){
  x_0<-one_step(f,x_0,h)
  i=i+1
 }
 return(x_0)
}


# 7. Use `newton_raphson` on `f` with initial value `-3` and assign the result
#    to `root`.
#    Check that `f(root)` is sufficiently close to zero by checking
#    that the absolute value of `f(root)` is smaller than `1e-8` and assign the
#    result of this evaluation to boolean variable `is_zero` (it should be `TRUE`)
## Do not modify this line!
root<-newton_raphson(f,-3)
is_zero<-abs(f(root))<1e-8


# 8. Add the root to `f_plot` and assign the resulting plot to
#    `plot_with_root`. Modify the scale to visualize it more clearly.
#    To do so, you can:
#    - starting with `f_plot`, add the point corresponding to `root` of using
#      `geom_point()` with `colour = "red"`,
#    - use `coord_cartesian(xlim=...)` to restrict shown x values between -5
#      and 0,
#    - add a blue vertical line of equation `x = -3` using `geom_vline()`,
#    - replace the title from `f_plot` with
#      `"Newton-Raphson finds the closest root to -3 within a few iterations"`
#      and remove the `subtitle`, using `labs()`
#
## Do not modify this line!
plot_with_root<-f_plot+
 geom_point(aes(x=root,y=0),color="red")+
 coord_cartesian(xlim=c(-5,0))+
```

```
  geom_vline(xintercept = -3,color="blue")+
  labs(title="Newton-Raphson finds the closest root to -3 within a few iterations",
       subtitle="")


# HW9: Optimizing Himmelblau’s function
#
# In this exercise, we will use different optimization methods to optimize
# Himmelblau’s function, which is a benchmark optimization function commonly used.
# Himmelblau’s function is defined as follows:
#    f(x,y) = (x^2 + y −11)^2 + (x + y^2 −7)^2
#
# 1. Create a function `himmelblau_helper <- function(x, y)` that takes as input
#    two scalars `x` and `y` and returns the calculated function value given by
#    `f(x,y) = (x^2 + y −11)^2 + (x + y^2 −7)^2`.
#    Example : `himmelblau_helper(3, 4) = 148`.
#    Then, create Himmelblau’s function `himmelblau <- function(para)` that takes a
#    vector `para` of size 2 as argument and return the calculated function value
#    given by `himmelblau(para) = himmelblau_helper(para[1], para[2])`.
#    Example : `himmelblau(c(3, 4)) = 148`.
#    This will help us deal with data frames later.
## Do not modify this line!
himmelblau_helper <- function(x, y){
  (x^2+y-11)^2+(x+y^2-7)^2
}
himmelblau <- function(para){
  himmelblau_helper(para[1], para[2])
}


# 2. Load packages `lattice` and `tidyverse`.
#    Our goal is to visualize the Himmelblau's function in the
#    two dimensional domain [-4, 4] x [-4, 4].
#    To do that, create a tibble `xy` using:
#      - `crossing()` to create tibble `xy`, inside which create column
#        `x` using `seq()` to generate a vector from -4 to 4 with `length`
#        set to `101` and `y` using `seq()` to generate a vector from `-4` to `4`
#        with `length` set to `101` as well.
#      - `mutate()` to create column `fnxy` calculated by `himmelblau_helper`.
#    `xy` should look like:
#    # A tibble: 10,201 x 3
#    x     y  fnxy
#    <dbl> <dbl> <dbl>
#    1    -4 -4    26
#    2    -4 -3.92 20.2
#    3    -4 -3.84 15.4
#    4    -4 -3.76 11.4
#    5    -4 -3.68  8.21
#    6    -4 -3.6   5.80
#    7    -4 -3.52  4.12
#    8    -4 -3.44  3.13
#    9    -4 -3.36  2.77
```

```
#    10    -4 -3.28  3.02
#    # … with 10,191 more rows
## Do not modify this line!
library(lattice)
library(tidyverse)
xy<-crossing(x=seq(-4,4,length=101),y=seq(-4,4,length=101))%>%
  mutate(fnxy=himmelblau_helper(x,y))


# 3. Use `wireframe` to draw 3D plot of the function. You can
#    set the following arguments :
#      - the first argument should be set to `fnxy ~ x * y`
#      - set `data` to `xy`.
#      - set `shade` to be `True`.
#      - set `scales` to be `list(arrows = FALSE)`, this will allow the axis
#        labels show up.
#      - set `screen` to be `list(z=-300, x=-70, y=0)` to set the plot size.
#      - set `xlab` to 'X', `ylab` to 'Y' and `zlab`='Z'.
#    Save the plot to `himmelblau_plot`.
#    We can see the function is decreasing around the four corners of the
#    4 x 4 square.
## Do not modify this line!
himmelblau_plot<-wireframe(fnxy~x*y,data=xy,shade=TRUE,scales=list(arrows =
FALSE),screen=list(z=-300, x=-70, y=0),xlab="X",ylab="Y",zlab="Z")


# 4. Then, we can draw the contour plot of Himmelblau's function.
#    To do that, you can use:
#      - `ggplot()` to initialize plot object over `xy` and set mapping to be
#        `aes(x = x, y = y, z = fnxy)`
#      - `geom_tile()` with `mapping` set to `aes(fill = fnxy)`.
#      - change the color of the contour by adding `scale_fill_viridis`
#        and set `options` to "plasma".
#      - `stat_contour()` to draw the contour plot, and set `breaks` to be
#        `c(0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100)`, which is
#         the value of each line.
#      - inside `labs`, set `x` to "X" and `y` to "Y", `fill` to "Z",
#        `title` to `"Contour of Himmelblau function"`.
#      - `theme_light()` for a light background
#    Save the generate plot to `contour`.
## Do not modify this line!
library(viridis)
contour<-ggplot(xy,aes(x = x, y = y, z = fnxy))+
  geom_tile(aes(fill = fnxy))+
  scale_fill_viridis(option="plasma")+
  stat_contour(breaks=c(0, 1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100))+
  labs(x="X",
       y="Y",
       fill="Z",
       title="Contour of himmelblau function")+
  theme_light()
```

```
# 5. Load package `pracma`.
#    We will now use four different optimization methods to find the minima of
#    the Himmelblau's function inside the area [-4, 4] x [-4, 4].
#    Initialize the start point `x0` to be `c(0, 0)`.
#    Firstly, we will try `gaussNewton`. It solves system of equations applying
#    the Gauss-Newton's method. It is especially designed for minimizing a
#    sum-of-squares of functions and can be used to find a common zero of
#    several function. To use `gaussNewton`, simply call `gaussNewton()` and
#    feed `x0` and our objective function `himmelblau` into it. Save the returned
#    element `xs` to `gaussNewton_minima` as our first minima.
## Do not modify this line!
library(pracma)
x0=c(0, 0)
gaussNewton_minima<-gaussNewton(x0,himmelblau)$xs

# 6. Load package `nloptr`.
#    We will now try `neldermead` method in package `nloptr`. It provides
#    explicit support for bound constraints. Whenever a new point would lie outside
#    the bound constraints the point is moved back exactly onto the constraint.
#    To use Nelder-Mead algorithm, simply call `neldermead()` and set `x0` and
#    `himmelblau` as first two arguments, and set `lower` to be `c(-4, -4)` and
#    `upper` to be `c(4, 4)`. Save the returned element to
#    `neldermead_minima` as our second minima.
## Do not modify this line!
library(nloptr)
neldermead_minima<-neldermead(x0,himmelblau,lower=c(-4,-4),upper=c(4,4))$par

# 7. Next, we will try `softline` method in package `pracma`. It is an inexact
#    line search algorithm that are used to find the most appropriate step size.
#    To use `softline()`, we feed `x0` as first argument, and a vector `d0`
#    representing the direction that we want to minimize. As we start from
#    original point, we want to explore four directions:
#    `(1, 1)`, `(-1, 1)`, `(-1, -1)` and  `(1, -1)`.
#    Use `softline` to get the optimal solutions for different directions.
#    To do that, you can use:
#     - `crossing()` to generate a tibble consisting of combination of
#       `x = c(1, -1)` and `y = c(1, -1)`.
#     - `mutate()` to generate column `d0` which represents the direction
#       we want to take, column `a` to represent the calculated gradient,
#       `optimum` to represent the calculated minima and `method` to
#       represent the direction used.
#       To be specific:
#       - use `map2(x, y, c)` to concatenate the column `x` and `y` to create
#         `d0`.
#       - use `map_dbl()` to conduct `softline()` optimization on each
#         `d0` with `x0` over `himmelblau` and create `a`.
#       - use `map2()` to calculate the predicted minima from `d0` and
#         `a` using the formula: minima = x0 + a * d0 and save it into column
#         `optimum`.
#       - use `paste0s("softline:a=")` to generate method name concatenated with
#         `a` with `round()` to 2 digits.
```

```
#   Save the result to `softline_dir` and it should print to:
#   # A tibble: 4 x 6
#   x     y     d0        a    optimum   method
#   <dbl> <dbl> <list>    <dbl> <list>    <chr>
#   1    -1    -1 <dbl [2]>  1    <dbl [2]> softline: a=1
#   2    -1     1 <dbl [2]>  2.97 <dbl [2]> softline: a=2.97
#   3     1    -1 <dbl [2]>  2.84 <dbl [2]> softline: a=2.84
#   4     1     1 <dbl [2]>  2.63 <dbl [2]> softline: a=2.63
#   We should get a sense that this method may not be as powerful as others since
#   it just select the step size of moving direction.
## Do not modify this line!
softline_dir<-crossing(x=c(1,-1),y=c(1,-1))%>%
  mutate(d0=map2(x, y, c),
         a=map_dbl(d0,softline,x0=x0,f=himmelblau),
         optimum=map2(d0,a,function(d0,a) x0+a*d0),
         method=paste0("softline: a=",round(a,2)))


# 8. Next we will use `optim()` function with initial parameter `x0` and `method`
#    `L-BFGS-B`. Save the returned element `par` as `optim_minima`.
## Do not modify this line!
optim_minima<-optim(x0,himmelblau,method="L-BFGS-B")$par


# 9. Summarize the above optimization results into a tibble by doing the following:
#    - use `tribble()`, inside which we set two columns to be `~optimum`, and
#      `~method` and then assign the corresponding minima with its method:
#      `gaussNewton_minima` to `"gaussNewton"`, `neldermead_minima` to `"neldermead"`,
#      and `optim_minima` to `"optim"`.
#    - use `bind_rows()` to concatenate the `method` and `optimum` columns,
#      using `dplyr::select()` to select the two columns from `softline_dir`.
#    - use `mutate()` to convert `optimum` columnn to tibbles using `map()`
#      to apply `enframe` to all the elements in `optimum`.
#    - separate each element in `optimum` to two columns using `unnest()`
#      before creating two columns containing `value` for each `name` using `spread()`.
#    - `rename()` column `1` to `x` and column `2` to `y`.
#    - use `mutate()` to calculate himmelblau function value column `fnxy`
#      using `himmelblau_helper`on `x` and `y`.
#   Save the result to `pts`.
#   `pts` should look like:
#    # A tibble: 7 x 4
#    method        x     y     fnxy
#    <chr>       <dbl> <dbl>    <dbl>
#    1 gaussNewton  3.00  2.00 2.10e- 6
#    2 neldermead   3.00  2.00 1.41e-11
#    3 optim        3.00  2.00 1.46e-12
#    # … with 4 more rows
#    (Notice that here, the first rows of x and y actually have slightly different
#    values but when printed, they are rounded to `2.00` and `3.00`)
## Do not modify this line!
pts<-tribble(~optimum, ~method,
          gaussNewton_minima,"gaussNewton",
```

```
        neldermead_minima,"neldermead",
        optim_minima,"optim")%>%
 bind_rows(softline_dir%>%dplyr::select(method,optimum))%>%
 mutate(optimum=map(optimum,enframe))%>%
 unnest(optimum)%>%
 spread(name,value)%>%
 rename(x=`1`,y=`2`)%>%
 mutate(fnxy=himmelblau_helper(x,y))
#记住tribble的构建方法和spread的用法

# 10. Add the minima found by each method on top of `contour`.
#    To do that, you can use :
#    - `geom_point()` with data set to `pts`, `mapping` as
#    `aes(color=method)` and `size` set to 2.
#    - `labs()`, setting `title` to `"Methods find different local minima"`
#     and `subtitle` to `"Softline allow to explore different directions"`.
#    - `theme_light()` to set light background.
#    Save the plot to `compare_plot`.
#    We can see that `gaussNewton`, `neldemead` and `optim` can converge to a good
#    local minima in the upper right corner while the `softline` converge worse
#    compared to them. However, `softline` can be useful if we want to implement
#    gradient descent in helping us find suitable step size in each step, so that
#    we can search for the best direction while optimizing how much we want to step
#    in that direction.
## Do not modify this line!
compare_plot<-contour+
 geom_point(data=pts,aes(color=method),size=2)+
 labs(title="Methods find different local minima",
     subtitle="Softline allow to explore different directions")+
 theme_light()


# HW9: Log normal distribution
#
# In this exercise, we will study Log Normal distribution.
# A log-normal distribution is a continuous probability distribution
# of a random variable whose logarithm is normally distributed.
# Thus, if the random variable X is log-normally distributed,
# then Y = ln(X) has a normal distribution.
# Source: https://en.wikipedia.org/wiki/Log-normal_distribution
#
# 1. Set the random seed to zero and save the random seed vector to `seed`.
#    (hint: use the command `seed <- .Random.seed`).
#    Generate 1000 samples from a log-normal distribution
#    with parameters `meanlog = 2`, `sdlog = 1`, add with random
#    noise from a normal distribution with mean 0 and standard deviation 0.05.
#    To do this, you should use:
#     - `set.seed()` to set random seed
#     - `rlnorm()` to generate random variables from log-normal distribution
#     - `rnorm()` to generate random variables from normal distribution
#    Save the generated vector into `x`.
## Do not modify this line!
```

```
set.seed(0)
seed <- .Random.seed
x<-rlnorm(1000,meanlog=2,sdlog=1)+rnorm(1000,mean=0,sd=0.05)


# 2. Now we want to estimate the parameters from these simulated points.
#    Let's create a function `mom_lnorm` that takes an input `par`, a
#    vector of length 2, representing respectively `meanlog` and `sdlog`
#    in the log-normal distribution.
#    The function will output a vector of length 2, representing respectively
#    the expectation and variance of the log-normal distribution.
#    Note: `expectation = exp(meanlog + 0.5 * sdlog^2)`,
#          `variance = exp(2 * meanlog + sdlog^2) * (exp(sdlog^2) - 1)`.
#    Example: `mom_lnorm(c(1, 1))` would return `[1]  4.481689 34.512613`.
#    Then create a function factory `obj_lnorm_factory` that takes an input `x`
#    representing the data, and return a function which takes an input `par` as
#    parameter and returns the sum of squared error between the
#    true mean and variance of `x` and the log-normal distribution
#    specified by `par`.
#    Example: `obj_lnorm_factory(c(0, 1))(c(1, 1))` would return `[1] 1172.712`.
## Do not modify this line!
mom_lnorm<-function(par){
  meanlog=par[1]
  sdlog=par[2]
  expectation = exp(meanlog + 0.5 * sdlog^2)
  variance = exp(2 * meanlog + sdlog^2) * (exp(sdlog^2) - 1)
  c(expectation,variance )
}
obj_lnorm_factory<-function(x){
  function(par){
    sum((mean(x)-mom_lnorm(par)[1])^2,(var(x)-mom_lnorm(par)[2])^2)
  }
}
#和slide不同的写法，但效果一致

# 3. (1) Create a function `par_lnorm` to estimate the parameters (the
#    moments of the distribution) from the data.
#    Goal: estimate the parameters that minimize the squared difference
#    between the true and estimated moments with a confidence interval.
#    Input: `x`, representing the data.
#    Output: # A tibble: 2 x 5
#            parameter value     se lower upper
#            <chr>     <dbl>  <dbl> <dbl> <dbl>
#          1 mu_mom      ...    ...   ...   ...
#          2 sd_mom      ...    ...   ...   ...
#    where
#    `mu_mom` is the estimate for `meanlog`.
#    `sd_mom` is the estimate for `sdlog`.
#    `se` is the standard error of the estimate.
#    `lower` and `upper` are the lower and upper bound
#    of the 95% confidence interval.
#    You can use:
```

```
#    - `optim()` to compute the optimal value.
#      - Set startpoint as `c(1, 1)`.
#      - Set `fn` to `obj_lnorm_factory(x)`.
#      - Set `hessian` to `TRUE`.
#    - `solve()` to compute the inverse of hessian matrix.
#    - `sqrt()` and `diag()` to compute the standard error.
#    - `tibble()` to create a tibble.
#    (2) Use `par_lnorm` with the data `x` simulated in question 1 to estimate
#    the moments. Save the tibble to `result_mom`.
#    Save the estimate of `meanlog` to scalar `mu_mom`.
#    Save the estimate of `sdlog` to scalar `sd_mom`.
## Do not modify this line!
library(tibble)
par_lnorm<-function(x){
  optim<-optim(c(1,1),fn=obj_lnorm_factory(x),hessian=TRUE)
  value<-optim$par
  se<-sqrt(diag(solve(optim$hessian)))
  lower<-value-1.96*se
  upper<-value+1.96*se
  tibble(parameter=c("mu_mom","sd_mom"),
         value=value,
         se=se,
         lower=lower,
         upper=upper)
}
result_mom<-par_lnorm(x)
mu_mom<-result_mom$value[1]
sd_mom<-result_mom$value[2]



# 4. After using the method of moments, let's try maximum likelihood estimators.
#    Create a function `nll_lnorm` to calculate the negative log-likelihood
#    of our data under the parameters. The function will take two inputs `par`,
#    a vector of length two representing the moments, and `x`, the data.
#    It returns the negative loglikelihood.
#    To do this, you can use:
#      - `dlnorm()` to calculate the likelihood
#      - specify `log = TRUE` to calculate the loglikelihood.
#    Then use your `nll_lnorm()` and `optim()` to estimate the parameters
#    (i.e. minimize the negative loglikelihood). Choose your startpoint to be `c(1, 1)`.
#    Save your estimated parameter `meanlog` into `mu_mle` and `sdlog` into `sd_mle`.
## Do not modify this line!
library(purrr)
nll_lnorm<-function(par,x){
  -sum(dlnorm(x,par[1],par[2],log=TRUE))
}
mu_mle<-optim(c(1,1),partial(nll_lnorm,x=x))$par[1]
sd_mle<-optim(c(1,1),partial(nll_lnorm,x=x))$par[2]



# 5. Compare the result of MOM and MLE by ploting the estimated density curves
```

```
#   together on the same plot.
#   You should first load `ggplot2` and then add a histogram and the estimated
#   density of the original `x`, then add the two estimated density curves.
#   To do this, you can use:
#     - `as.data.frame()` to tranform `x` into a dataframe.
#     - `ggplot()` to initialize a ggplot object.
#       Set `aes(x = x)`.
#     - `geom_histogram()` to add an histogram with density scale.
#       - set `y` to `..density..` in the aesthetics and specify `bins=50`.
#     - `geom_density()` to add the density curve.
#       - specify `color = "Data"`.
#     - `stat_function` to add the two curves for MOM and MLE.
#       - specify `color = "MOM"` and `color = "MLE"` for each curve.
#     - `labs()` to format the labels such that:
#       - `title = "Both methods seem to do a good job"`.
#       - `x = "Simulated x"`.
#       - `y = "Density"`.
#     - `theme_light()` to change the theme of plots.
#     - `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#   Save your figure into `compare_plot`.
#
## Do not modify this line!
library(ggplot2)
compare_plot<-x%>%
  as.data.frame()%>%
  ggplot(aes(x=x))+
  geom_histogram(aes(y=..density..),bins=40)+
  geom_density(aes(color="Data"))+
  stat_function(aes(color="MOM"),fun=dlnorm,n=
1000,args=list(meanlog=mu_mom,sdlog=sd_mom))+
  stat_function(aes(color="MLE"),fun=dlnorm,n=
1000,args=list(meanlog=mu_mle,sdlog=sd_mle))+
  labs(title = "Both methods seem to do a good job",
       x = "Simulated x",
       y = "Density")+
  theme_light()+
  theme(plot.title=element_text(hjust = 0.5))
```

2019年12月6日 18:01

HW10

```
# HW10: LCG for generating distributions
#
# In this exercise, we will walk you through the process of using LCGs to generate
# distributions.
# We suggest the functions you can use to create the tibbles and the plots, but
# you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#
# Throughout the exercise:
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - Do NOT use `for`, `while` or `repeat` loops except for question 1.
#   - When defining functions, you do not need to follow the exact process as
#     suggested by the instructions. However, make sure you are using the same
#     parameter settings, inputs and outputs in order to pass the test.
#
# 1. Load the package `tidyverse`.
#   In this exercise, we will create an LCG to simulate a Uniform distribution
#   and utilize it to simulate some other distributions.
#   First, define a function `lcg_unif` to simulate a Uniform Distribution in
#   [0, 1] using the recurrence relation formula, $X_n = (aX_n − 1 + c) mod m$.
#   `lcg_unif` should:
#     - Take inputs
#       - `n`, the number of samples to generate.
#       - `x`, defaulted to `2^32 - 1`.
#       - `m`, defaulted to `2^32`.
#       - `a`, defaulted to `1103515245`.
#       - `c`, defaulted to `12345`.
#     - Create a vector `rng` of length `n`.
#     - Loop over 1 to `n` to update
#       - `x` using the formula.
#       - `rng[i]` by `x / m`.
#     - Return the vector `rng` for the simulated samples.
#   Use `lcg_unif()` to generate 5000 samples from the Uniform(0, 1) and
#   store the result into a vector `unif_rng`.
#   To check your result, `head(unif_rng, 5)` prints to:
#   [1] 0.74307060 0.87848759 0.77028656 0.04326916 0.83987619
## Do not modify this line!

library(tidyverse)
lcg_unif = function(n, x = 2^32-1, m = 2^32, a = 1103515245, c = 12345){
 rng = rep(0, length(n))
 for(i in 1:n){
  x = (a*x + c) %% m
  rng[i] = x/m
 }
```

```
  return (rng)
}
unif_rng = lcg_unif(5000)


# 2. To check if our parameter choices are good, we would like to draw a
#    scatterplot for successive pairs (X[i], X[i + 1]) of the samples generated
#    by our LCG. Ideally, these pairs should be independent and should fill the
#    entire sample space.
#    (1) Create a tibble `unif_pairs` of size 4999 x 2 with columns `X1` and
#        `X2`, where `X1` is `unif_rng[1:4999]` and `X2` is `unif_rng[2:5000]`.
#        To do this, you can use:
#          - `tibble()` to make a tibble for `unif_rng[1:4999]` and
#            `unif_rng[2:5000]`.
#          - `rename()` to rename the columns as `X1` and `X2`.
#        To check your result, `unif_pairs` prints to:
#        # A tibble: 4,999 x 2
#             X1     X2
#          <dbl>  <dbl>
#        1 0.743  0.878
#        2 0.878  0.770
#        3 0.770  0.0433
#        4 0.0433 0.840
#        5 0.840  0.388
#        # … with 4,994 more rows
#    (2) Draw the scatterplot for the successive pairs using `unif_pairs`.
#        To do this, you can use:
#          - `ggplot()` to initialize a ggplot object and specify variables to plot.
#          - `geom_point()` to draw the scatterplot.
#          - `labs()` to format the labels such that:
#            - `title = "Successive Pairs of Simulated X ~ Unif(0, 1)"`
#            - `subtitle = "Good generator as pairs are independent and uniformly distributed."`
#            - `x = "X_n"`
#            - `y = "X_{n+1}"`
#        Store the plot into a ggplot object `g1`.
## Do not modify this line!
unif_pairs = tibble(X1 = unif_rng[1:4999], X2 = unif_rng[2:5000])
g1 = unif_pairs %>%
  ggplot(mapping = aes(x = X1, y = X2)) +
  geom_point() +
  labs(title = "Successive Pairs of Simulated X ~ Unif(0, 1)",
       subtitle = "Good generator as pairs are independent and uniformly distributed.",
       x = "X_n",
       y = "X_{n+1}") +
  theme_light()


# 3. With `lcg_unif`, we would like to first similuate an exponential distribution.
#    Define a function `lcg_exp` that
#      - Takes inputs
#        - `n`, the number of samples to generate.
#        - `rate`, the rate parameter of the exponential distribution,
```

```
#         defaulted to `1.5`.
#      - Creates
#        - A vector `U` of length `n` for samples from Uniform(0, 1) using
#          `lcg_unif()`.
#        - A vector `E` of length `n` for samples from Exponential(rate) using
#          `- 1 / rate * log(U)`.
#      - Returns a tibble of size n x 1 with the column `E`.
#    Use `lcg_exp()` to generate 5000 samples for Exponential(1.5) and store the
#    result into a vector `exp_rng`.
#    To check your result, `exp_rng` prints to:
#    # A tibble: 5,000 x 1
#          E
#      <dbl>
#    1 0.198
#    2 0.0864
#    3 0.174
#    4 2.09
#    5 0.116
#    # … with 4,995 more rows
## Do not modify this line!
lcg_exp = function(n, rate = 1.5){
  U = lcg_unif(n)
  E = -1/rate * log(U)
  return(tibble(E))
}
exp_rng = lcg_exp(5000)

# 4. Draw a histogram with density plot to compare the simulated distribution
#    and the actual one.
#    To do this, you can use:
#      - `ggplot()` to initialize a ggplot object and specify variables to plot.
#      - `geom_histogram()` to draw the density histogram, setting
#          `fill = "white", color = "black"`.
#          Hint: set `y` as density instead of count in the aesthetics.
#      - `geom_density()` to draw the density plot, setting `aes(color = "Simulated")`.
#      - `stat_function()` to add a density curve for Exponential(1.5) using
#          `dexp`, setting `aes(color = "Theoretical")`.
#      - `labs()` to format the labels such that:
#        - `title = "PDF of X ~ Exponential(1.5)"`
#        - `subtitle = "Simulated values are close to actual ones with n = 5000."`
#        - `x = "X"`
#        - `y = "Density"`
#    Store the plot into a ggplot object `g2`.
## Do not modify this line!
g2 = exp_rng %>%
  ggplot(mapping = aes(x = E)) +
  geom_histogram(aes(y = ..density..), fill = 'white', color = 'black', bins = 30) +
  geom_density(aes(color = 'Simulated')) +
  stat_function(aes(color = 'Theoretical'), fun = dexp, args = list(rate = 1.5)) +
  labs(title = "PDF of X ~ Exponential(1.5)",
       subtitle = "Simulated values are close to actual ones with n = 5000.",
       x = "X", y = "Density") +
```

```
  theme_light()
#注意stat_function添加arg的方法


# 5. With `lcg_unif`, we can also simulate a normal distribution. We will use
#    the Box-Muller transformation to generate a pair of independent normal
#    variables (Z1, Z2) by transforming a pair of independent Uniform(0, 1)
#    random variables (U1, U2). The formulas follow:
#      - `Z1 = mean + sd * sqrt(-2 * log(U1)) * cos(2 * pi * U2)`.
#      - `Z2 = mean + sd * sqrt(-2 * log(U1)) * sin(2 * pi * U2)`.
#    With the formulas, define a function `lcg_norm` that
#      - Takes inputs
#        - `n`, the number of samples to generate.
#        - `mean`, the mean parameter of the normal distribution, defaulted to 0.
#        - `sd`, the standard deviation parameter of the normal distribution,
#          defaulted to 1.
#      - Creates
#        - A vector `U1` of length `n` for samples from Uniform(0, 1) using
#          `lcg_unif()` with `x = 2^32 - 1`.
#        - A vector `U2` of length `n` for samples from Uniform(0, 1) using
#          `lcg_unif()` with `x = 2^32 - 2`.
#        - A vector `Z1` of length `n` for samples from Normal(mean, sd) using
#          the formulas above.
#        - A vector `Z2` of length `n` for samples from Normal(mean, sd) using
#          the formulas above.
#      - Returns a tibble of size n x 2 with columns `Z1` and `Z2`.
#    Use `lcg_norm()` to generate 5000 samples for a pair of independent
#    Normal(0, 1) and store the result into a vector `norm_rng`.
#    To check your result, `norm_rng` prints to:
#    # A tibble: 5,000 x 2
#          Z1      Z2
#       <dbl>   <dbl>
#    1 -0.768  0.0670
#    2 -0.237 -0.451
#    3 -0.612  0.384
#    4  0.848 -2.36
#    5 -0.241  0.539
#    # … with 4,995 more rows
## Do not modify this line!

lcg_norm = function(n, mean = 0, sd = 1){
  U1 = lcg_unif(n, x = 2^32-1)
  U2 = lcg_unif(n, x = 2^32-2)
  Z1 = mean + sd * sqrt(-2 * log(U1)) * cos(2 * pi * U2)
  Z2 = mean + sd * sqrt(-2 * log(U1)) * sin(2 * pi * U2)
  return(tibble(Z1 = Z1, Z2 = Z2))
}
norm_rng = lcg_norm(5000)
# 6. Draw a histogram with density plot to compare the simulated distribution
#    and the actual one.
#    To do this, you can use:
#      - `gather()` to collect `Z1` and `Z2` as a `key` column with corresponding
```

```
#       `value`.
#     - `ggplot()` to initialize a ggplot object and specify variables to plot.
#     - `geom_histogram()` to draw the density histogram, setting
#       `fill = "white", color = "black"`.
#       Hint: set `y` as density instead of count in the aesthetics.
#     - `geom_density()` to draw the density plot, setting `aes(color = "Simulated")`.
#     - `stat_function()` to add a density curve for Normal(0, 1) using `dnorm`,
#       setting `aes(color = "Theoretical")`.
#     - `facet_wrap()` to facet the plot by `key`.
#     - `labs()` to format the labels such that:
#       - `title = "PDFs of Two Independent Standard Normal Random Variables Generated by
Box-Muller."`
#       - `subtitle = "Simulated values are close to actual ones with n = 5000."`
#       - `x = "Z"`
#       - `y = "Density"`
#   Store the plot into a ggplot object `g3`.
## Do not modify this line!
g3 = norm_rng %>% gather('key', 'value', Z1, Z2) %>%
  ggplot(mapping = aes(x = value)) +
  geom_histogram(fill = 'white', color = 'black', aes(y = ..density..)) +
  geom_density(aes(color = 'Simulated')) +
  stat_function(aes(color = 'Theoretical'), fun = dnorm, args = list(mean = 0, sd = 1)) +
  facet_wrap(~key) +
  labs(title = "PDFs of Two Independent Standard Normal Random Variables Generated by
Box-Muller.",
       subtitle = "Simulated values are close to actual ones with n = 5000.",
       x = "Z",
       y = "Density") +
  theme_light()


# HW10: Pseudo random generation with Sobol sequences
#
# A low-discrepancy sequence is a sequence with the property that for all
# values of N, its subsequence x1, ..., xN has a low discrepancy.
# Roughly speaking, the discrepancy of a float number sequence is low if the
# proportion of points in the sequence falling into an arbitrary interval I is
# close to proportional to the measure of I (ie. its length), as would happen
# on average (but not for particular samples) in the case of an equidistributed
# sequence.
#
# Such sequences are often used for MC Integration and the method of estimation
# is then called a quasi-Monte Carlo method. The estimation is bounded by both
# a term due to the function to integrate alone and a term related to the
# discrepancy of the sequence used for estimation. Low-discrepancy sequences
# provide both a low bound and a way to avoid re-computation of the whole
# sequence if we decide to augment the estimation with more points and still
# keep low discrepancy.
# Sobol sequences are an example of low-discrepany sequences used for
# quasirandom number generations. We will implement a function to generate
# Sobol sequences and look how the values are distributed across time,
# comparing with other pseudo random distributions.
```

```
#
# The sequence consists of values `v_i = m_i / 2^i` where `m_i` are odd
# positive integers smaller than `2^i` (so that the `v_i` are in `[0, 1]`).
# The sequence requires picking an irreducible polynomial with coefficients in
# `{0, 1}` (meaning the polynomial can't be factorized into smaller degree
# polynomials with coefficients in `{0, 1}` as well) defined by:
# `f(x) = z^p + c_1*z^(p-1) + ... + c_(p-1)*z + c_p` with the `c_i` in `{0, 1}`
# Then one has to choose initial values for the sequence.
# The recurrence relation used to generate the `m_i` from the preceding ones is
# `m_i = 2*c_1*m_(i-1) xor 2^2*c_1*m_(i-1) xor ... xor 2^p*c_1*m_(i-1) xor m_(i-p)`
# where `xor` is the bitwise mutually exclusive or (e.g.
# `3 xor 4 = 011 xor 100 = 111 = 7` using their binary representations).
#
# We will use the polynomial `x^4 + x + 1` (ie. the coefficients `0, 0, 1, 1`).
# We will use starting values `m_1 = 1`, `m_2 = 1`, `m_3 = 3` and `m_4 = 13`.
#
# 1. Load the `binaryLogic` package.
#    Assign the coefficients chosen above i.e. `0, 0, 1, 1` to the vector `coeffs`.
#    Compute the value `m5` from the recurrence formula above.
#    To do so you can use `as.binary()` that converts an integer into a vector
#    of its binary representation, `xor()` and `as.numeric()` to get back an
#    integer from a vector of its binary representation.
## Do not modify this line!
library(binaryLogic)
library(tidyverse)
coeffs = c(0,0,1,1)
previous = c(13,3,1,1)
aux = function(x,k) as.binary(2^k*x)
coeff_x_previous = coeffs * previous
m5 = tibble(coeff_x_previous = coeff_x_previous,
          k = 1:length(coeff_x_previous),
          tmp = map2(coeff_x_previous, k, ~aux(.x,.y))) %>%
  pull(tmp) %>%
  reduce(xor) %>%
  xor(as.binary(previous[length(previous)])) %>%
  as.numeric()
# 2. Load the `tibble`, `dplyr` and `purrr` packages.
#    In this question we will compute `next_in_seq`, a function that takes as
#    input the vector of coefficients of the chosen polynomial `coeffs` and
#    the vector of previous values needed to compute the next value `previous`.
#    Elements of `previous` are to be ordered from newest to olders (ie. to
#    compute `m_i`, the first element of `previous` should be `m_(i-1)`, the
#    second `m_(i-2)`, etc.). `coeffs` and `previous` should have the same
#    length.
#    In the body of the function you can:
#    - define a helper function `aux` that takes in two integer arguments `x`
#      and `k` and returns the binary representation of `2^k*x` using
#      `as.binary()`,
#    - compute the element-wise product of `coeffs` and `previous` and assign
#      it to `coeff_x_previous`,
#    - map `aux()` to `coeff_x_previous` and an integer vector of `k` from 1 to
#      the length of `coeffs` using `map2()`. More precisely, you can:
```

```
#     - construct a tibble with three columns:
#       - `coeff_x_previous` containing the previously computed vector
#       - `k` containing integers 1, 2, ..., `length(coeffs)` which you can
#         compute using `seq_along()` and `coeff_x_previous`
#       - `tmp` containing the result of mapping `aux()` to the two other
#         columns by calling `map2()` on `coeff_x_previous`, `k` and `aux()`,
#     - `pull()` column `tmp` to obtain a vector
#    - reduce the vector using the `xor()` function and `reduce()`,
#    - add a final `xor()` with the result of the previous step and the binary
#      representation of the last element of `previous` - which you can obtain
#      using `as.binary()`
#    - finally return the result as an integer using `as.numeric()` on the
#      resulting binary representation of the previous step to retrieve the
#      associated integer
#    For example, `next_in_seq(c(0, 1, 1), c(3, 1, 1))` should be equal to `13`
## Do not modify this line!
aux = function(x,k) as.binary(2^k*x)
next_in_seq = function(coeffs, previous){
  coeff_x_previous = coeffs * previous
  tibble(coeff_x_previous = coeff_x_previous,
         k = 1:length(coeff_x_previous),
         tmp = map2(coeff_x_previous, k, ~aux(.x,.y))) %>%
    pull(tmp) %>%
    reduce(xor) %>%
    xor(as.binary(previous[length(previous)])) %>%
    as.numeric()
}


# 3. Once the `m_i` will all have been computed, we will need to transform them
#    into the `v_i = m_i / 2^i`.
#    Write a function `post_process()` that takes in a generated sequence of
#    `m_i` called `vec` and returns the associated `v_i` vector.
#    For example, `post_process(c(1, 1, 3, 13))` should be equal to
#    `c(0.5000, 0.2500, 0.3750, 0.8125)`
## Do not modify this line!
post_process = function(vec){
  for(i in 1:length(vec)){
    vec[i] = vec[i]/2^i
  }
  return (vec)
}


# 4. Now we can compute the `sobol` function, that requires arguments `coeffs`,
#    `n` and `init` and returns the sobol sequence with `n` elements, built
#    using the polynomial with coefficients `coeffs` and starting at `init` (
#    the first values of the sequence). The function should:
#    - initialize the `output` vector to be returned with `init`,
#    - initialize the `previous` vector to be used to compute the next value
#      using `init` (change the order so that the `m_1` is last in `previous`,
#      - while it should be first in `output`),
```

```
#    - then call `next_in_seq()` until `output` grows to a size of `n`,
#      updating `previous` after each call
#    - return `post_process(output)`
#    For example, `sobol(c(0, 1, 1), 5, c(1, 1, 3))` should be equal to
#    `c(0.50000 0.25000 0.37500 0.81250 0.15625)`
## Do not modify this line!
sobol = function(coeffs, n, init){
  output = NULL
  output[1:length(init)] = init
  previous = rev(init)
  for(i in (length(init)+1):n){
    m = next_in_seq(coeffs,previous)
    output[i] = m
    previous = append(m, previous)[1:length(coeffs)]
  }
  post_process(output)
}


# 5. Load the `tibble` and `tidyr` packages.
#    Create the `init` vector `c(1, 1, 3, 13)`.
#    Set the seed by doing:
#    - call `set.seed(11)`,
#    - assign `.Random.seed` to variable `seed`,
#    Construct the `simul_tibble` tibble by:
#    - calling `tibble()` to generate a tibble with 4 columns:
#      - `sequence` containing integers from 1 to 50 - the indices of the
#        sequences' elements,
#      - `sobol` containing the output of `sobol(coeffs, 50, init)`,
#      - `normal` containing a sample of size 50 from the normal distribution
#        with mean `0.5` and standard deviation `0.25`,
#      - `uniform` containing a sample of size 50 from the uniform distribution
#        on `[0, 1]`
#    - calling `pivot_longer()` on all columns except `sequence` to create a
#      tidy tibble.
#    `simul_tibble` should print to:
#    # A tibble: 150 x 3
#       sequence type    value
#          <int> <chr>   <dbl>
#    1        1 sobol   0.5
#    2        1 normal  0.539
#    3        1 uniform 0.921
#    4        2 sobol   0.25
#    5        2 normal  0.328
#    6        2 uniform 0.275
#    7        3 sobol   0.375
#    8        3 normal  0.613
#    9        3 uniform 0.901
#   10        4 sobol   0.812
#    # … with 140 more rows
## Do not modify this line!
init = c(1, 1, 3, 13)
```

```
set.seed(11)
seed = .Random.seed
simul_tibble = tibble(sequence = 1:50,
                      sobol = sobol(coeffs, 50, init)[1:50],
                      normal = rnorm(50, 0.5, 0.25),
                      uniform = runif(50, 0, 1)) %>%
  pivot_longer(-sequence, names_to = 'type', values_to = 'value')


# 6. Load the `ggplot2` package.
#    Plot the sequence values for each type of simulation from `simul_tibble`
#    and assign the plot to `simul_plot`.
#    To do so, you can:
#    - call `ggplot()` on `simul_tibble` with the aesthetic `x = sequence`,
#      `y = value`,
#    - add points for each value using `geom_point()`,
#    - facet the plot by type of simulation using `facet_wrap()` and `type`,
#    - add the following title using `ggtitle()`:
#      `"There is a clear non random pattern in the Sobol sequence generation"`
#    - add `theme_light()`
#
## Do not modify this line!
library(ggplot2)
simul_plot = simul_tibble %>%
  ggplot(mapping = aes(x = sequence, y = value)) +
  geom_point() +
  facet_wrap(~type) +
  ggtitle("There is a clear non random pattern in the Sobol sequence generation") +
  theme_light()

# HW10: Inverse Transform Sampling
#
# In this exercise, we will learn to apply the inverse transform method.
# We will focus on the weibull distribution.
# For details, here is the link: [https://en.wikipedia.org/wiki/Weibull_distribution]
#
# Throughout the exercise:
#    - Do NOT use `for`, `while` or `repeat` loops.
#    - Use `%>%` to structure your operations.
#    - Use `theme_light()` for the plots.
#    - For graphs with titles, use `theme()` to set
#      `plot.title = element_text(hjust = 0.5)` and
#      `plot.subtitle = element_text(hjust = 0.5)`.
#
# 1. Let `U` be a uniform random variable over [0, 1]. Let `X` be a weibull
#    random variable with cdf `F` such that `F(X) ~ U[0, 1]`. Based on the
#    link given above, derive the formula that `F^{-1}(u) = x`. Then, create
#    a function `Finv()` that takes an input `u`, `shape`(which is k) and
#    `scale`(which is lamda) and ouput `F^{-1}(u)` from a weibull distribution.
#    Specify the `shape` as 3 and `scale` as 2.
#    For example, Finv(0) = 0,  Finv(0.5, shape = 1, scale = 1) = 0.6931472.
## Do not modify this line!
```

```
Finv = function(u, shape = 3, scale = 2){
  (-log(1-u))^(1/shape)*scale
}
#找到cdf，解出逆函数，再写即可

# 2. Load the `tidyverse` package.
#    Use `set.seed()` to set the random seed to 100. Store the seed to `seed1`.
#    Generate 10000 samples from uniform distribution and save them into a
#    tibble `df`.
#    To do this, you use:
#    - `runif()` to generate random samples from uniform distribution and name
#    the column as `u`.
#    - `Finv()` you created to calculate values and name the column as `x`.
#    To check your answer, the `df` prints to:
#    # A tibble: 10,000 x 2
#    u     x
#    <dbl> <dbl>
#    1 0.308  1.43
#    2 0.258  1.34
#    3 0.552  1.86
#    4 0.0564 0.774
#    5 0.469  1.72
#    6 0.484  1.74
#    7 0.812  2.37
#    8 0.370  1.55
#    9 0.547  1.85
#    10 0.170  1.14
#    # … with 9,990 more rows
## Do not modify this line!
library(tidyverse)
set.seed(100)
seed1 = .Random.seed
df = tibble(u = runif(10000)) %>%
  mutate(x = Finv(u))


# 3. Use `set.seed()` to set the random seed to 100. Store the seed to `seed2`.
#    Use `ks.test()` to test if the values of `x` we calculated fit well in
#    weibull distribution. The argument should take in values of `x` from df,
#    `pweibull` and two parameters we specified in the first problem. Extract
#    p-value of the test and store it to `pvalue_result`.
## Do not modify this line!
set.seed(100)
seed2 = .Random.seed
pvalue_result = ks.test(df$x, pweibull, shape = 3, scale = 2)$p.value

# 4. Load the `ggplot2` package.
#    Draw a histogram of `x` to compare with the distribution of weibull
#    dsitribution.
#    To do this, you can use:
#    - `ggplot()` to plot `x` from `df`.
#    - `geom_histogram()` to add a hitogram with density scale.
```

```
#       - specify `bins = 40`.
#       - specify `y = ..density..`.
#    - `stat_function()` to plot the density of weibull distribution
#       - use `dweibull()` to calculate the density.
#       - specify `color` in `aes()` as `"True value"`.
#    - `scale_colour_manual()` to name the title as `"Legend"` and color the
#      `"True value"` as `"red"`.
#    - `labs()` to format the labels such that:
#       - `title = "The density plot of random generated samples is similar to the true
distribution"`.
#       - `x = "Random generated x"`.
#       - `y = "Density"`.
#    Store the returned plot to `g1`.
## Do not modify this line!
library(ggplot2)
g1 = df %>% ggplot(mapping = aes(x = x)) +
  geom_histogram(bins = 40, aes(y = ..density..)) +
  stat_function(aes(color = 'True value'), fun = dweibull,
                args = list(shape = 3, scale = 2)) +
  scale_colour_manual(values = c('red'), labels = 'True value', name = 'Legend') +
  labs(title = "The density plot of random generated samples is similar to the true
distribution",
       x = "Random generated x",
       y = "Density") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
#scale_colour_manual 中 注意assign的不同argument


# 5. Draw a qq-plot to plot the estimated distribution vs. true value.
#    To do this, you can use:
#      - `ggplot()` to plot the `x` from `df`.
#      - `geom_qq()` to draw a weibull distribution with parameters
#        of `dparams`.
#      - `stat_qq_line()` to draw a straight line that takes weibull
#        distribution with parameters of `dparams`, make the `color` as red.
#      - `labs()` to name the title as `"The sample seems to distribute along the straight line"`,
#        subtitle as `"The samples fit well in the weibull distribution with shape = 3 and scale =
2"`,
#        x-axis as `"Theoretical values"`,
#        y-axis as `"Actual values"`.
#    Store returned graph to `g2`.
## Do not modify this line!
g2 = df %>% ggplot(mapping = aes(sample = x)) +
  geom_qq(distribution = qweibull, dparams = list(shape = 3, scale = 2)) +
  stat_qq_line(distribution = qweibull, dparams = list(shape = 3, scale = 2), color = 'red') +
  labs(title = "The sample seems to distribute along the straight line",
       subtitle = "The samples fit well in the weibull distribution with shape = 3 and scale = 2",
       x = "Theoretical values",
       y = "Actual values") +
  theme_light() +
```

```
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))



# HW10: Inverse Transform Method
#
# In this exercise, we will consider the Cauchy random variable
# X[https://en.wikipedia.org/wiki/Cauchy_distribution] and use inverse
# transformation method to sample the random variables.
#
# 1. Let U be a uniform random variable over [0,1]. We want to find a transformation
#    of U that allows us to simulate a Standard Cauchy random variable X from U.
#    By following the steps of inverse transformation method, we want to first
#    obtain the cumulative distribution function `F`. Create a function `F1` that
#    takes an input `x`, `location` and `scale` and output the calculated
#    cumulative density for a cauchy distribution with speficified location and
#    scale. Set the default location into 0 and default scale into 1.
#    For example, F1(0) = 0.5,  F1(1,location=-1,scale=0.5) = 0.922
#    You should use `atan()` to calculate the arc tangent of an input and you should not
#    directly use `pcauchy()` to calculate the cdf.
## Do not modify this line!
F1 = function(x, location = 0, scale = 1){
  1/pi * atan((x-location)/scale) + 1/2
}



# 2. Calculate the inverse of `F1` and create a function `Finv()` that takes an
#    input `u`, `location` and `scale` and ouput `F^{-1}(u)` from a cauchy distribution
#    with specified location and scale. Again, set the default location
#    into 0 and default scale into 1.
#    You should use `tan()` to calculate the tangent of an input and you should not
#    directly use `pcauchy()` to calculate the cdf.
#    For example, Finv(0.5) = 0,  Finv(0.92, location=-1, scale=0.5) = 0.947
## Do not modify this line!
Finv = function(u, location = 0, scale = 1){
  location + scale*tan(pi*(u - 1/2))
}



# 3. Load the `tidyr` package.
#    Use `set.seed()` to set the random seed to 0 and save your seed into `seed` using
#    `.Random.seed`
#    Generate 1000 samples from standard cauchy distribution and save them into a
#    tibble `df`.
#    To do this, you should:
#    - Use `runif()` to generate random samples from uniform distribution.
#    - Use `Finv()` created before.
#    You `df` should look like:
#    A tibble: 1,000 x 2
#    u      x
#    <dbl>  <dbl>
```

```
#    0.897  2.97
#    0.266 -0.907
#    0.372 -0.425
#    0.573  0.233
#    0.908  3.37
#    0.202 -1.36
#    0.898  3.03
#    0.945  5.70
#    0.661  0.553
#    0.629  0.429
#    … with 990 more rows.
## Do not modify this line!
library(tidyverse)
set.seed(0)
seed = .Random.seed
df = tibble(u = runif(1000)) %>%
  mutate(x = Finv(u))


# 4. Check your simulated distribution by using a histogram.
#    Recall that cauchy distribution is a heavy tail distribution with more probability
#    on the extreme values, which will make our plot hard to interpret. So before
#    we plot, we want to first filter out the extreme values.
#    To do this, you can:
#    - Load the `ggplot2` and `dplyr` package.
#    - Use `filter()` to keep the samples in (-50,50).
#    - Use `ggplot()` to initialize a ggplot object.
#      Set its arguments `data` and `mapping`.
#    - Use `geom_histogram()` to add a hitogram with density scale.
#        - specify `bins=100`.
#        - specify `y = ..density..`.
#    - Use `stat_function()` to plot the density true standard cauchy distribution
#        - specify `color = "red"`.
#        - use `dcauchy()` to calculate the density.
#    - Use `labs()` to format the labels such that:
#        - `title = "The generated samples have same density as the true distribution"`.
#        - `x = "Simulated x"`.
#        - `y = "Density"`.
#    - Use `theme_light()` to change the theme of plots.
#    - Use `theme()` to change the title and subtitle to the middle of the plot.
#        - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#    Save your plot into `g1`.
## Do not modify this line!
library(ggplot2)
g1 = df %>% filter(x < 50, x > -50) %>%
  ggplot(mapping = aes(x = x)) +
  geom_histogram(bins = 100, aes(y = ..density..)) +
  stat_function(color = 'red', fun = dcauchy,
                args = list(location = 0, scale = 1)) +
  labs(title = "The generated samples have same density as the true distribution",
       x = "Simulated x",
       y = "Density") +
```

```
 theme_light() +
 theme(plot.title = element_text(hjust = 0.5),
       plot.subtitle = element_text(hjust = 0.5))



# 5. Check your simulated distribution by using a qqplot.
#    To do this, you can:
#    - Use `ggplot()` to initialize a ggplot object.
#      Set its arguments `data` and `mapping`.
#    - Use `geom_qq()` to add a qqplot
#       - specify `distribution = qcauchy`.
#    - Use `stat_qq_line()` to plot the reference line
#       - specify `color = "red"`.
#       - specify `distribution = qcauchy`.
#    - Use `labs()` to format the labels such that:
#       - `title = "The sampling distribution follows the general pattern of a standard cauchy"`.
#       - `subtitle = "Cauchy distribution is more likely to generate extreme variables"`.
#       - `x = "Theoretical"`.
#       - `y = "Sample"`.
#    - Use `theme_light()` to change the theme of plots.
#    - Use `theme()` to change the title and subtitle to the middle of the plot.
#       - Set its argument `plot.title` using `element_text(hjust = 0.5)`.
#       - Set its argument `plot.subtitle` using `element_text(hjust = 0.5)`.
#    Save your plot into `g2`.
## Do not modify this line!
g2 = df %>%
 ggplot(mapping = aes(sample = x)) +
 geom_qq(distribution = qcauchy, dparams = list(location = 0, scale = 1)) +
 stat_qq_line(distribution = qcauchy, dparams = list(location = 0, scale = 1), color = 'red') +
 labs(title = "The sampling distribution follows the general pattern of a standard cauchy",
      subtitle = "Cauchy distribution is more likely to generate extreme variables",
      x = "Theoretical",
      y = "Sample") +
 theme_light() +
 theme(plot.title = element_text(hjust = 0.5),
       plot.subtitle = element_text(hjust = 0.5))
#不要忘了dparams

# HW10: Rejection sampling for Gamma distribution estimate.
#
# In this exercise, we will go over rejection sampling for estimating
# the gamma distribution Gamma(10, 0.3).
#
# 1. Load the packages `ggplot2`, `tibble` and `tidyverse`.
#    Create a function `f <- function(x)` that takes as input
#    one scalar `x` and returns the gamma density for `shape = 10`
#    and `scale = 0.3`.
#    Example : `f(1) = 0.01664854`.
#    Create a function `g <- function(x)` that takes as input scalar `x` and returns
#    the density of uniform distribution between 0 and 10.
#    Example : `g(2) = 0.1`.
```

```
#   These functions will make the computations easier.
#   Find the maximum of `f`.
#   To do that, use `optimize()` to find where `-f` reaches its minimum and
#   and assign it to `mode`. Then, compute the corresponding value for `f` and
#   assign it to `M_best`.
## Do not modify this line!
library(ggplot2)
library(tibble)
library(tidyverse)
f = function(x){
  dgamma(x, shape = 10, scale = 0.3)
}
g = function(x){
  dunif(x, min = 0, max = 10)
}
mode = optimize(f, c(0,10), maximum = TRUE)$maximum
M_best = optimize(f, c(0,10), maximum = TRUE)[[2]]

# 2. Use `set.seed()` to set seed to `0` and save seed to `seed1` using `.Random.seed`.
#   Create a function `rejection_sampling <- function(M, f, g, seed = 0, n = 1e3)`
#   which takes as input a scalar `M`, two functions `f` and `g`, a seed and a number `n`,
#   and returns samples of `f` between 0 and 10 using rejection sampling.
#   The function should :
#     - set the seed to `seed` using `set.seed()`.
#       Do that inside the function so each call
#       to the function will use the same random seed.
#     - generate `n` samples from `runif(1000, 0, 10)` into `y`
#     - generate `n` samples from `runif(1000, 0, 1)` into `u`
#     - return the values of `y` such that `u < f(y) / (M *  g(y))`.
#   Example: `rejection_sampling(1, f, g, 0, 3)` will return `[1] 2.655087 3.721239`.
## Do not modify this line!
rejection_sampling <- function(M, f, g, seed = 0, n = 1e3){
  set.seed(seed)
  seed1 = .Random.seed
  y = runif(n, 0, 10)
  u = runif(n, 0, 1)
  return(y[u < f(y) / (M *  g(y))])
}
#注意这里选择时实际上是利用index一致的特点，其实并不算太完善
# 3. Set `n` to `1e3`
#   Generate a tibble `result` which gives the approximated acceptance ratio for
#   different constants M. It should ciontain three columns :
#    - `M` that contains a sequence of number from `M_best` to `5` spaced by `0.025`.
#      You can use `seq()` to generate this sequence.
#    - `Y` that contains samples of the distribution when using rejection sampling
#      with constant `M`. You can use `map()` and `rejection_sampling()` to compute it.
#    - `acceptance_rate` that contains the ratio of samples accepted and `n`
#   `result` should print to:
#   # A tibble: 183 x 3
#         M Y          acceptance_rate
#     <dbl> <list>              <dbl>
#   1 0.439 <dbl [482]>         0.482
```

```
#    2 0.464 <dbl [478]>         0.478
#    3 0.489 <dbl [476]>         0.476
#    4 0.514 <dbl [469]>         0.469
#    5 0.539 <dbl [466]>         0.466
#    6 0.564 <dbl [464]>         0.464
#    7 0.589 <dbl [459]>         0.459
#    8 0.614 <dbl [458]>         0.458
#    9 0.639 <dbl [454]>         0.454
#    10 0.664 <dbl [450]>          0.45
#    # … with 173 more rows
## Do not modify this line!
library(tidyverse)
n = 1e3
result = tibble(M = seq(M_best, 5, by = 0.025)) %>%
  mutate(Y = map(M, ~rejection_sampling(.x,f,g,0,n))) %>%
  mutate(acceptance_rate = map_dbl(Y, ~length(.x)/n))
#注意最后的ratio是按照点数除以总数



# 4. Now, let's plot it to see how acceptance ratio changes:
#    - use `ggplot()` to initialize the plot object.
#    - use `geom_line()` to plot the line.
#    - use `labs` with `x` set to `"M"`, `y` set to `"Acceptance rate"` and
#      `title` set to `"The acceptance rate decreases with M"`.
#    - use `theme_light()` to set a light background.
#    Save the plot to `acceptance_plot`.
## Do not modify this line!
acceptance_plot = result %>%
  ggplot(mapping = aes(x = M, y = acceptance_rate)) +
  geom_line() +
  labs(x = "M", y = "Acceptance rate",
       title = "The acceptance rate decreases with M") +
  theme_light()



# 5. As we were using a distribution defined between 0 and 10, it won't help
#    estimating the whole distribution.
#    We now want to sample from a distribution defined on `[0, Inf)`,
#    and we'll use the Gaussian which is supported on `(-Inf, Inf)`.
#    In order to get the smallest `M` value possible, we can decide to sample from
#    a Gaussian that has the same mode as `f`.
#    Create a function `g2(x)` that represents the gaussian density centered
#    at `mode` with standard deviation `sd = 3`.
#    Compute the new value for `M2`. (It is equal to the maximum of `f/g2`,
#    that is to the value of `f/g2` at `mode`.)
#    Set the seed to `1` and save it to `seed` using `.Random.seed`.
#    Generate tibble `result2`, in which :
#     - the first column `y` represents a `n` absolute values of samples from
#       normal distribution with mean `mode` and `sd = 3`. You can use `rnorm()`.
#     - the second column `u` represents `n` samples from uniform distribution
```

```
#      between 0 and 1 using `runif()`,
#    - the third column `likelihood_ratio` is the ratio of `f(y)` and `M2 * g(y)`
#    - the fourth column `selected` is set to boolean `u < likelihood_ratio`.
#   `result2` should print to:
#   # A tibble: 10,000 x 4
#   y     u likelihood_ratio selected
#   <dbl> <dbl>           <dbl> <lgl>
#   1 0.821 0.211      0.0142      FALSE
#   2 3.25  0.115      0.862       TRUE
#   3 0.193 0.145      0.000000296 FALSE
#   4 7.49  0.310      0.00408     FALSE
#   5 3.69  0.150      0.649       TRUE
#   6 0.239 0.527      0.00000168  FALSE
#   7 4.16  0.118      0.423       TRUE
#   8 4.91  0.477      0.179       FALSE
#   9 4.43  0.308      0.319       TRUE
#   10 1.78  0.305      0.533       TRUE
#   # … with 9,990 more rows
## Do not modify this line!
g2 = function(x){
  dnorm(x, mean = mode, sd = 3)
}
M2 = f(mode)/g2(mode)
set.seed(1)
seed = .Random.seed
result2 = tibble(y = rnorm(n, mode, 3), u = runif(n, 0 ,1)) %>%
  mutate(likelihood_ratio = f(y)/(M2*g2(y))) %>%
  mutate(selected = u < likelihood_ratio)

# 6. To visualize it, let's plot the two distributions with the accepted and
#    rejected samples.
#    - Use `ggplot()` to initialize plot object, in which the aesthetics are
#      set to `aes(x = y)`.
#    - Use `geom_point()` to draw sampled points with `data` set to `result2`,
#      and the aesthetics set so that you plot `M2 * u * g2(y)` against `y`,
#      and `color` to `selected`.
#    - Use `stat_function()` to draw the plot of density `f` with `size = 2`.
#    - Use `stat_function()` to draw the plot of density `M2 * g2`
#      setting `color` to `"red"` and  `size = 2`.
#    - Use `labs()` to generate axis labels `x` to `"x"`, `y` to `"Density"`
#      and `title` to `"Acceptance illustration"`
#    - Use `theme_light()`.
#   Save it to `good_rejection_sampling_plot`.
## Do not modify this line!
good_rejection_sampling_plot = result2 %>%
  ggplot(mapping = aes(x = y)) +
  geom_point(aes(y  = M2 * u * g2(y), color = selected)) +
  stat_function(fun = f, size = 2) +
  stat_function(fun = ~M2*g2(.x), color = 'red', size = 2) +
  labs(x = 'x', y = 'Density',
       title = "Acceptance illustration") +
  theme_light()
```

#stat_function中如果要自己写函数就要~

```
# HW10: Acceptance-Rejection Sampling for Dirichlet distribution
#
# In this exercise, we will use Acceptance-Rejection Sampling method
# to sample from the Dirichlet distribution.
#
# The Dirichlet distribution, often denoted `Dir(alpha)`, is a family
# of continuous multivariate probability distributions parameterized by
# a vector `alpha` of positive real numbers.
#
# It is commonly used as prior distributions in Bayesian statistics,
# as the Dirichlet distribution is the conjugate prior of the
# categorical distribution and multinomial distribution.
#
# Source: https://en.wikipedia.org/wiki/Dirichlet_distribution
#
# 1. In this part, we will sample from a Dirichlet distribution
#    using package `MCMCpack`.
#    (1)
#    Load the `tidyverse`, `MCMCpack` and `ggplot2` packages.
#    Set the random seed to zero and save the random seed vector to
#    `seed`. (hint: use the command `seed <- .Random.seed`).
#    (2)
#    Generate 1000 samples from `Dir(2, 2, 2)` using `rdirichlet()`.
#    Save the result into a tibble `X`
#    and rename the columns `x1`, `x2`, and `x3`.
#    The first rows of `X` should be:
#    # A tibble: 1,000 x 3
#      x1     x2     x3
#      <dbl>  <dbl>  <dbl>
#    1 0.660 0.216  0.125
#    2 0.242 0.458  0.300
#    3 0.556 0.279  0.165
#    4 0.243 0.336  0.421
#    5 0.108 0.478  0.414
#    6 0.403 0.371  0.226
#    7 0.497 0.403  0.0997
#    8 0.272 0.343  0.385
#    9 0.200 0.0708 0.730
#   10 0.151 0.467  0.382
#   # … with 990 more rows
## Do not modify this line!
library(MCMCpack)
library(tidyverse)
library(ggplot2)
set.seed(0)
seed = .Random.seed
X = rdirichlet(1000, c(2,2,2))
X = tibble(x1 = X[,1], x2 = X[,2], x3 = X[,3])
```

```
# 2. Draw the density plot of the samples and save it into `density_plot`.
#    In our case, `K = 3`, and `a1 = a2 = a3 = 2`. This distribution is
#    essentially a bivariate one, as `X3 = 1 − X1 − X2`.
#    Therefore, drawing samples from this Dirichlet distribution is equivalent
#    to drawing samples of `(X1, X2)`. We only need to draw the heatmap of the
#    `(X1, X2)`.
#    You can use:
#    - `ggplot()` to initialize a ggplot object.
#      Set `data` to `X`.
#      Set `mapping` to `aes(x1, x2)`.
#    - `stat_density_2d()` to plot the density.
#      - Set `fill` in `aes()` to `..density..`.
#      - Set `geom` to `raster`.
#      - Set `contour` to `FALSE`.
#    - `geom_density_2d()` to add contours.
#    - `scale_x_continuous()` to customize x scale.
#      - Set `expand` to `c(0, 0)`.
#    - `scale_y_continuous()` to customize y scale.
#      - Set `expand` to `c(0, 0)`.
#    - `labs()`
#      - Set `title` to `"Density plot of Dir(2, 2, 2)"`.
#      - Set `subtitle` to
#        `"The samples are distributed in the lower left corner."`.
#    - `theme_light()` to set the theme to light.
#    - `theme()` to remove the legend.
#      - Set `legend.position` to `none`.
## Do not modify this line!
density_plot = X %>%
  ggplot(mapping = aes(x1, x2)) +
  stat_density_2d(aes(fill = ..density..), geom = 'raster',
                  contour =  FALSE) +
  geom_density_2d() +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  labs(title = "Density plot of Dir(2, 2, 2)",
       subtitle = "The samples are distributed in the lower left corner.") +
  theme_light() +
  theme(legend.position = 'none')



# 3. Now, we will use the Acceptance-Rejection Sampling method to simulate
#    from this bivariate distribution of `(X1,X2)`.
#    Recall that `f(x)` is our target distribution, `g(x)` is our proposed
#    distribution. We want `f(x) ≤ M * g(x)` for `M > 1`.
#    For this part, we will use a two-dimensional uniform distribution on
#    `[0, 1] x [0, 1]` as our proposal distribution. Note that sampling from
#    a two-dimensional uniform distribution is just sampling from a
#    one-dimensional twice.
#    (1)
#    Write a function `f(x)` for the target distribution,
#    in this case `Dir(2, 2, 2)`.
```

```
#   It takes in a vector of length 2, representing `x1` and `x2`.
#   The output is the density of the distribution `Dir(2, 2, 2)`.
#   Do not use the functions from `MCMCpack`.
#   For example, `f(c(0, 0)) = 0` and `f(c(0.3, 0.3)) = 4.32`.
#   (2)
#   Find the optimal `M` i.e. the smallest possible value and save it
#   into scalar `M`.
#   Use `f(x)` to compute `M` instead of using functions from `MCMCpack`.
#   Recall that the density of uniform distribution is always one.
#   To ensure for every x, `f(x) ≤ M * g(x)`, we just need to make sure
#   `M` is larger than the highest point of `f(x)`.
#   To check your answer, verify that `round(M, 1)` equals to `4.4`.
#   (3)
#   Write another function `g(x)` for the proposal distribution,
#   in this case a two-dimensional uniform distribution.
#   It takes in a vector of length 2, representing `x1` and `x2`.
#   The output is the density of the uniform distribution.
#   Remember the density should be zero is the input is not
#   within the support.
#   For example, `g(c(0, 0)) = 1` and `g(c(-1, -1)) = 0`.
## Do not modify this line!
f = function(x){
  if(sum(x) > 1)
    return (0)
  x[1]*x[2]*(1-x[1]-x[2])*120
}
#这个是按Dir(2,2,2) 的 pdf来写的，注意gamma functoni is (n-1)!
M<-abs(optim(c(0.1,0.1),function(x) -f(x))$value)
#如果使用optim()不像optimize可以选max/min，所以才需要-f(x)得到结果
g = function(x){
  if(x[1] <= 1 & x[2] <= 1 & x[1] >= 0 & x[2] >= 0)
    return (1)
  return (0)
}
# 4. (1)
#   Use `f(x)`, `g(x)`, and `M` that we just defined to write a
#   sampling function `sample_one()` to draw one sample from
#   `Dir(2, 2, 2)`. There is no input.
#   The output is a tibble with 3 columns:
#   - The fisrt two columns `x1` and `x2` contain the pair of
#     sample `x1` and `x2`.
#   - The third column `count` contains the the number of
#     attempts to obtain this sample.
#   You can use the following skeleton to write your function:
#   ```
#   sample_one <- function() {
#     count <- 0
#     accepted <- FALSE
#     while (!accepted) {
#       ## your code goes here
#     }
```

```
#     ## your code goes here
#   }
#   ```
#   For example, if you set seed to zero, `sample_one()` would return:
#   # A tibble: 1 x 3
#       x1    x2 count
#    <dbl> <dbl> <dbl>
#   1 0.126 0.267     7
#   `(0.126, 0.267, 0.607)` is the sample we draw (with
#   `0.601 = 1 - 0.126 - 0.26`). To obtain this sample, we sampled
#   from `g(x)` 7 times before the sample is accepted.
#   (2)
#   Write a sampling function `sample_n(n, seed)` that
#   draws samples from `Dir(2, 2, 2)`.
#   Use `sample_one()` that we just defined.
#   The input `n` is the number of samples we will draw.
#   The input `seed` is the seed used within the function.
#   The default seed should be set to zero.
#   Remember to use `set.seed()` inside the function.
#   The output is a tibble with 3 columns:
#   - The fisrt two columns `x1` and `x2` contain the pair of
#     sample `x1` and `x2`.
#   - The third column `count` contains the the number of
#     attempts to obtain this sample.
#   For example, `sample_n(100, 100)` would return
#   # A tibble: 100 x 3
#       x1    x2 count
#    <dbl> <dbl> <dbl>
#   1 0.308 0.258     1
#   2 0.280 0.398     3
#   3 0.205 0.358     1
#   4 0.208 0.307     8
#   5 0.236 0.275     1
#   6 0.445 0.358     3
#   7 0.327 0.389     9
#   8 0.397 0.393     5
#   9 0.186 0.348    16
#  10 0.219 0.292     7
#  # … with 90 more rows
## Do not modify this line!
set.seed(0)
sample_one = function(){
 count = 0
 accepted = FALSE
 while(!accepted){
  count = count + 1
  x = runif(2)
  u = runif(2)
  if(max(u) < f(x)/(M*g(x))){
   accepted = TRUE
  }
 }
```

```
  return (tibble(x1 = x[1], x2 = x[2], count = count))
}
#这里最重要的是accepted的条件
sample_n = function(n, seed = 0){
  set.seed(seed)
  k = NULL
  for (i in 1:n){
    k = rbind(k, sample_one())
  }
  return (k)
}

# 5. (1)
#    Draw 1000 samples from `Dir(2, 2, 2)` using `sample_n()` with
#    default seed and save the samples in `samples`.
#    The first rows of `samples` should be:
#    # A tibble: 1,000 x 3
#         x1    x2 count
#      <dbl> <dbl> <dbl>
#    1 0.126 0.267     7
#    2 0.122 0.245    16
#    3 0.208 0.229    10
#    4 0.557 0.329     6
#    5 0.181 0.530     1
#    6 0.269 0.181    12
#    7 0.340 0.262     4
#    8 0.131 0.374     3
#    9 0.143 0.415     3
#   10 0.109 0.382     7
#   # … with 990 more rows
#    (2)
#    Compute the acceptance rate using `count` column in `samples`
#    and save it in scalar `acceptance_rate`.
#    You can see that the acceptance rate is around 15%. It is very
#    low and we would like it to be higher to get a more efficient procedure.
## Do not modify this line!
samples = sample_n(1000)
acceptance_rate = nrow(samples)/sum(samples$count)
#acceptance_rate就是产生的总数/需要的sample次数

# 6. Plot the dot plot of the samples and save it into `sample_plot`.
#    You can use:
#    - `ggplot()` to initialize a ggplot object.
#      Set `data` to `samples`.
#      Set `mapping` to `aes(x1, x2)`.
#    - `geom_point()` to plot the dots.
#      - Set `alpha` to `0.8`.
#    - `labs()`
#      - Set `title` to `"Samples from Dir(2, 2, 2)"`.
#      - Set `subtitle` to `"It is similar to ones that drawn from rdirichlet."`.
#    - `theme_light()` to set the theme to light.
```

```
## Do not modify this line!
sample_plot = samples %>%
  ggplot(mapping = aes(x1, x2)) +
  geom_point(alpha = 0.8) +
  labs(title = "Samples from Dir(2, 2, 2)",
       subtitle = "It is similar to ones that drawn from rdirichlet.") +
  theme_light()
```

```
# 7. The acceptance rate of using Uniform distribution is really low since
#    the shape of these two distribution are not alike.
#    We need to find a distribution that has a peak around the peak of the
#    dirichlet distribution, but that doesn't decay too fast to 0 so that the
#    constant `M` dosn't have to be too high.
#    Let's suppose that we can sample efficiently from a Beta distribution.
#    We can find parameters such that the Beta distribution will have a shape
#    that is closer to the Dirichlet distribution than the uniform distribution.
#    To make the sampling more efficient, we will use the product of two
#    Beta distribution as the proposal distribution.
#    In order to have a peak around the peak of the dirichlet distribution,
#    and not decay too fast ot 0, the alpha and beta will all be 1.1.
#    (It will give only a slightly better convergence rate. We encourage you
#    to play with different alphas and betas once the exercises is done to see
#    how `M` and the acceptance rate evolve.)
#    (1)
#    Write a function `g2(x)` for the product of two Beta distribution.
#    It takes in a vector of length 2, representing `x1` and `x2`.
#    The output is the product of the values of the
#    density of a `Beta(1.1, 1.1)` distribution at points `x1` and `x2`.
#    You can use the `dbeta()` function.
#    For example, `g2(c(0, 0)) = 0` and `g2(c(0.5, 0.5)) = 1.123137`.
#    (2)
#    Set M to `4.06` and save it into `M2`.
#    This value makes sure for all x, `f(x) <= M2 * g2(x)`.
## Do not modify this line!
g2 = function(x){
  dbeta(x[1],1.1,1.1)*dbeta(x[2],1.1,1.1)
}
M2 = 4.06
```

```
# 8. (1)
#    Use `f(x)`, `g2(x)`, and `M2` that we just defined to write a sampling
#    function `sample_one2()` to draw one sample from `Dir(2, 2, 2)`.
#    There is no input.
#    The output is a tibble with 3 columns:
#    - The fisrt two columns `x1` and `x2` contain the pair of
#      sample `x1` and `x2`.
#    - The third column `count` contains the the number of
#      attempts to obtain this sample.
```

```
#   For example, if you set seed to zero, `sample_one2()` would return:
#   # A tibble: 1 x 3
#       x1    x2 count
#     <dbl> <dbl> <dbl>
#   1 0.433 0.313    11
#   (2)
#   Write a sampling function `sample_n2(n, seed)` that
#   draws samples from `Dir(2, 2, 2)`.
#   Use `sample_one2()` that we just defined.
#   The input `n` is the number of samples we will draw.
#   The input `seed` is the seed used within the function.
#   The default seed should be set to zero.
#   Remember to use `set.seed()` inside the function.
#   The output is a tibble with 3 columns:
#   - The fisrt two columns `x1` and `x2` contain the pair of
#     sample `x1` and `x2`.
#   - The third column `count` contains the the number of
#     attempts to obtain this sample.
#   For example, `sample_n2(100, 100)` would return
#   # A tibble: 100 x 3
#         x1     x2 count
#       <dbl>  <dbl> <dbl>
#   1 0.316  0.550     1
#   2 0.218  0.338     8
#   3 0.587  0.134     1
#   4 0.458  0.255     2
#   5 0.401  0.474     9
#   6 0.549  0.248     1
#   7 0.229  0.362    14
#   8 0.279  0.0851    2
#   9 0.0689 0.310     4
#   10 0.143  0.338     7
#   # … with 90 more rows
#   (3)
#   Draw 100 samples from `sample_n2()` with default seed and
#   save the acceptance rate to `acceptance_rate2`. You can see
#   that the acceptance rate is around 16%.
#   It is slightly better and the procedure will be faster.
#   (given that we know how to sample from a beta distribution).
#   Sampling from a Dirichlet distribution can be tricky and expensive.
#   You can try to think of other distributions that would help!
## Do not modify this line!
set.seed(0)
sample_one2 = function(){
  count = 0
  accepted = FALSE
  while(!accepted){
    count = count + 1
    x = rbeta(2, 1.1, 1.1)
    u = runif(2)
    if(max(u) < f(x)/(M2*g2(x))){
      accepted = TRUE
```

```
  }
 }
 return (tibble(x1 = x[1], x2 = x[2], count = count))

}


sample_n2 = function(n, seed = 0){
 set.seed(seed)
 k = NULL
 for (i in 1:n){
  k = rbind(k, sample_one2())
 }
 return (k)
}

samples = sample_n2(100)
acceptance_rate2 = nrow(samples)/sum(samples$count)

# HW10: Inverse Transform Sampling
#
# In this exercise, we will learn to use the bootstrap for bias correction.
# The inspiration is from this link and on p126 of the linked book:
# [https://books.google.com/books?id=MWC1DwAAQBAJ&pg=PA139&lpg=PA139
&dq=Efron+and+Tibshirani+chapter+10.4
&source=bl&ots=_bEGQKn74v&sig=ACfU3U0IFgsMnr3bzhNQNruroDsrwBQLww&hl=en&sa=
X&ved=2ahUKEwiu577p4
_nlAhVLq1kKHelhAnMQ6AEwAnoECAcQAQ#v=onepage&q&f=false]
# The book talks about two methods to use bootstrap to find the bias of
# estimator. We are going to implement the two methods and draw plots to
# compare the two methods here.
#
# Throughout the exercise:
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - For graphs with titles, use `theme()` to set
#     `plot.title = element_text(hjust = 0.5)` and
#     `plot.subtitle = element_text(hjust = 0.5)`.
#
# 1. Load the `tidyverse` package. Use `read_csv()` to read in `"patch.csv"`
#   from the `data` folder. Create a new tibble with two columns `z` and `y`.
#   The formulas are `z = oldpatch - placebo` and `y = newpatch - oldpatch`.
#   To do this, you can use:
#     - `transmute()` to create the new columns.
#   Store returned tibble to `df`.
#   To check your answer, `df` prints to:
#   # A tibble: 8 x 2
#   z     y
#   <dbl> <dbl>
#   1  8406 -1200
#   2  2342  2601
```

```
#   3  8187 -2705
#   # … with 5 more rows
#
#   Let‘s try the first so-called normal method.
#
## Do not modify this line!
library(tidyverse)
df = 'data/patch.csv' %>% read_csv() %>%
  transmute(z = oldpatch - placebo, y = newpatch - oldpatch)



# 2. Create a function called `ratio()` that takes in `df` as argument. The
#   function calculates the ratio of mean of `y` and mean of `z`. Then,
#   apply `ratio()` to `df` and store the returned value to `theta_hat`.
#   (e.g. Let `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#   `ratio(df)` prints to:
#   [1] 1.452135)
#
## Do not modify this line!
ratio = function(df){
  mean(df$y)/mean(df$z)
}
theta_hat = ratio(df)



# 3. Create a function called `boot_df()` that takes in `df` as argument and
#   samples all the columns of `df` with replacement.
#   To do this, you can use:
#     - `sample_frac()` to sample every column with `replace` as `TRUE`.
#   (e.g. set seed as 1, let `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#   `boot_df(df)` prints to:
#   # A tibble: 2 x 2
#   y     z
#   <dbl> <dbl>
#   1  4.06  6.18
#   2  7.06  1.48
#   )
#
#   Create a function called `boot_ratio()` that takes in `nboot` and `df`
#   as arguments. The function should take `nboot` samples from `df` and
#   apply `ratio()` to every sample. Finally the function returns a vector
#   of returned values of samples.
#   To do this, you can use:
#     - `tibble()` to generate three columns:
#       - The first column is called `b` that stores a vector of integers
#         from 1 to `nboot`.
#       - The second column is called `df_b` that maps `boot_df()` to every
#         integer to `b`. This column should store `nboot` samples.
#         You may use `map()` here.
#       - The third column is called `theta_hat_b`. This column stores
#         ratios of each sample in `df_b`.
```

```
#     - `pull()` to pull out `theta_hat_b` from the tibble.
#    (e.g. set seed as 1, let `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#    `nboot = 2`, `boot_ratio(nboot, df)` prints to:
#    [1] 0.6567138 1.4521353
#    )
#
## Do not modify this line!
set.seed(1)
boot_df = function(df){
  sample_frac(df, replace = TRUE)
}
boot_ratio = function(nboot, df){
  theta_hat_b = tibble(b = 1:nboot) %>%
    mutate(df_b = map(b, ~boot_df(df))) %>%
    mutate(theta_hat_b = map_dbl(df_b, ratio)) %>%
    pull()
}
```

#b其实是计数的

```
# 4. Use `set.seed()` to set seed to 1 and store it to `seed1`.
#    Here, it's for the bootstrap for different bootstrap lengths!
#    Create a tibble that has 5 columns:
#      - The first column is called `nboot` and it stores a vector of
#        integers `50, 100, 200, 500, 1000`.
#      - The second column is called `theta_hat_b`. Each entry of the column
#        takes a list of ratios of `nboot` samples. (e.g. In the first row,
#        `nboot` is 50, then the first entry of `theta_hat_b` should be a list
#        of ratios with length 50.)
#        (hint: you may use `map()` here and it takes three arguments)
#      - The third column is called `bias`. It takes mean of each every of
#        `theta_hat_b` and then it substract `theta_hat`.
#      - The fourth column is called `theta_hat_corrected` and it is
#        `theta_hat` minus `bias`.
#      - The fifth column is called `type` and the values of all entries are
#        `"normal"`.
#    Then, drop the `theta_hat_b` column. Store the return tibble to
#    `results`.
#    To check your answer, `results` prints to:
#    # A tibble: 6 x 4
#    nboot    bias theta_hat_corrected type
#    <dbl>   <dbl>              <dbl> <chr>
#    1    50 0.0260            -0.0973 normal
#    2   100 0.0118             -0.0831 normal
#    3   200 0.00755            -0.0789 normal
#    # … with 3 more rows
#
#    Now, let's implement the so-called improved method.
## Do not modify this line!
set.seed(1)
seed1 = .Random.seed
results = tibble(nboot = c(50, 100, 200, 500, 1000)) %>%
  mutate(theta_hat_b = map(nboot, ~boot_ratio(.x, df))) %>%
```

```
  mutate(bias = map_dbl(theta_hat_b, mean), bias = bias - theta_hat) %>%
  mutate(theta_hat_corrected = theta_hat - bias) %>%
  mutate(type = 'normal') %>%
  select(-theta_hat_b)


# 5. Create a function called `rmean()` that takes in a vector of `p` and a
#    vector of `x`. The function should calculate the sum of product of `p`
#    and `x` then divided by length of `x`.
#    (e.g. Let `p = c(0.5, 0.5)`, `x = c(1, 2)`,
#    `rmean(p, x)` prints to:
#    [1] 0.75
#    )
#
#    Create a function called `rratio()` that takes in a vector of `p` and
#    `df`.
#    The function should apply `rmean()` to `y` and `z` columns in `df` and
#    divide the first manipulation by the second manipulation.
#    (e.g. Let `p = c(0.5, 0.5)`,
#    `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#    `rratio(p, df)` prints to:
#    [1] 1.452135
#    )
#
## Do not modify this line!
rmean = function(p, x){
  sum(p*x)/length(x)
}
rratio = function(p, df){
  y = rmean(p, df$y)
  z = rmean(p, df$z)
  y/z
}



# 6. Create a function called `boot_p_ratio()` that takes in `df` and it
#    should return a tibble with a calculated `theta` and a list of
#    calculated `p`.
#    Here, we only have one sample.
#    To do this, you can do as following:
#      - Use `nrow()` to count number of rows of `df` and store it to `n`.
#      - Use `sample()` to generate samples of size `n` with `replace` as
#        `TRUE`.
#      - Use `tabulate()` to count how many times each number between 1 and
#        `n` appears in the sample, then divide all of them by `n`. This is the
#        proportion of each number and store the vector to `p`. Here, the
#        length of `p` should be 8.
#      - Apply `rratio()` to `p` and `df` to calculate the theta
#        and store returned values to `theta`.
#      - Use `tibble()` to create two columns with `theta` and `p`.
#      - Finally return the tibble.
```

```
#   (e.g. set seed as 1, let `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#   `boot_p_ratio(df)` prints to:
#   # A tibble: 1 x 2
#   theta p
#   <dbl> <list>
#   1  4.78 c(0.5, 0.5)
#   )
#
#   Create a function called `mean_p()` that takes in a list of `p` and
#   calculate mean of each index of `p`.
#   To do this, you can use:
#     - `reduce()` to calculate sum of `p` on each index and then divide
#       by the length of `p`.
#   (e.g. let `p` be list(c(0.5, 0.2), c(0.1, 0.3), c(1, 2)),
#   `mean_p(p)` prints to: [1] 0.5333333 0.8333333)
#
## Do not modify this line!
boot_p_ratio = function(df){
  n = nrow(df)
  sample = sample(n, replace = TRUE)
  p = tabulate(sample)/n
  p = c(p, rep(0, n-length(p)))
  theta = rratio(p, df)
  tibble(theta = theta, p = list(p))
}
mean_p<-function(p){
  reduce(p, `+`) / length(p)
}
#使用reduce的原因是p list 中是vector

# 7. Create a function called `boot_ratio2()` that takes in `nboot` and `df`.
#    The function creates `nboot` samples of `df` and calculates the ratios
#    with the improved method described in the book.
#    Here, we have `nboot` samples.
#    To do this, you can use:
#      - `tibble()` to generate two columns:
#        - The first column is called `b` that stores a vector of integers
#          from 1 to `nboot`.
#        - The second column is called `df_b` that maps `boot_p_df()` to
#          every integer to `b`. This column should store `nboot` samples of
#          `theta` and `nboot` lists of `p`.
#          You may use `map()` here.
#      - `unnest()` to unnest samples in `df_b`.
#      - `summarize()` to summarize the statistics:
#        - The first column is called `theta_hat_b` that stores the mean
#          of `theta`.
#        - The second column is called `p_hat_b` that stores the list of mean
#          of each index `p` from all samples. You may apply `mean_p()` here,
#          and the length of `p_hat_b` should be 8.
#
#     (e.g. set seed as 1, let `df` be `c(y = c(7.06, 4.06), z = c(1.48, 6.18))`,
#      `nboot = 3`, `boot_ratio2(nboot, df)` prints to:
```

```
#    # A tibble: 1 x 2
#    theta_hat_b p_hat_b
#    <dbl>      <list>
#    1       1.45 c(0.5, 0.5)
#    )
#
## Do not modify this line!
boot_ratio2 = function(nboot, df){
 tibble(b = 1:nboot) %>%
  mutate(df_b = map(b, ~boot_p_ratio(df))) %>%
  unnest(df_b) %>%
  summarize(theta_hat_b = mean(theta), p_hat_b = list(mean_p(p)))

}

#  8. Use `set.seed()` to set seed to 1 and store it to `seed2`.
#    Here, it's for the bootstrap for different bootstrap lengths!
#    Create a tibble that has 6 columns:
#      - The first column is called `nboot` and it stores a vector of
#        integers `50, 100, 200, 500, 1000`.
#      - The second column is called `theta_hat_p_hat`. Each entry of
#        the column has a value of `theta_hat_b` and a list of values of `p`.
#        (hint: you may use `map()` here and it takes three arguments)
#
#      - Then use `unnest()` to unnest `theta_hat_p_hat` into two columns.
#      - The third column is called `theta_hat`. You may use `map_dbl()`
#        combined with `p_hat_b`, `rratio()` and `df` to calculate the
#        estimated `theta_hat`.
#      - The fourth column is called `bias`. It takes mean of each every of
#        `theta_hat_b` and then it substract `theta_hat`.
#      - The fifth column is called `theta_hat_corrected` and it is
#        `theta_hat` minus `bias`.
#      - The sixth column is called `type` and the values of all entries are
#        `"improved"`.
#    To check your answer, `results2` prints to:
#    # A tibble: 6 x 4
#    nboot    bias theta_hat_corrected type
#    <dbl>   <dbl>             <dbl> <chr>
#    1    50 0.00544           -0.0614 improved
#    2   100 0.00676           -0.0734 improved
#    3   200 0.00685           -0.0808 improved
#    # … with 3 more rows
#
## Do not modify this line!
set.seed(1)
seed2 = .Random.seed
results2 = tibble(nboot = c(50, 100, 200, 500, 1000)) %>%
 mutate(theta_hat_p_hat = map(nboot, boot_ratio2, df)) %>%
 unnest(theta_hat_p_hat) %>%
 mutate(theta_hat = map_dbl(p_hat_b, ~rratio(.x, df))) %>%
 mutate(bias = theta_hat_b - theta_hat) %>%
 mutate(theta_hat_corrected = theta_hat - bias) %>%
```

```
  mutate(type = 'improved') %>%
  select(nboot, bias, theta_hat_corrected, type)
#  9. Draw a line plot to plot the estimated `bias` vs. `nboot`.
#     To do this, you can use:
#       - `bind_rows()` to bind `results2` to `results`
#       - `ggplot()` to plot the `bias` vs. `nboot`, colored by `type`.
#       - `geom_line()` to draw to lines.
#       - `labs()` to name the title as `"The bias of improved method converge faster than that
of the normal method"`,
#         x-axis as `"Nboot"`,
#         y-axis as `"Bias"`,
#         color as `Type`.
#     Store returned graph to `g1`.
## Do not modify this line!

g1 = bind_rows(results, results2) %>%
  ggplot(aes(nboot, bias, color = type))+
  geom_line()+
  labs(title = "The bias of improved method converge faster than that of the normal
method",
       x = "Nboot",
       y = "Bias",
       color = "Type")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))



# HW10: Bootstrap confidence interval
#
# In this exercise, we will use bootstrap to estimate the confidence interval for
# regression coefficients. We will fit linear regression on bootstrap samples of
# `mtcars` dataset and estimate the confidence intervals.
#
# 1. Let's first calculate the estimated intercept and slope on the full dataset
#    without resampling.
#    Load the `tidyverse` and `broom` library.
#    Use `set.seed()` to set the random seed to 0 and save your seed into `seed` using
#    `.Random.seed`.
#    Create a function `model_mtcar` which takes an input `df` and output a linear model
#    of `mpg` regress on `wt` on the `df` dataset.
#    For example, the output of `model_mtcars(mtcars[1:20,]) %>% tidy()` shoule be:
#    # A tibble: 2 x 5
#    term        estimate std.error statistic  p.value
#    <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#    (Intercept)    38.5      2.36      16.3  3.07e-12
#    wt             -5.41     0.664     -8.15 1.88e- 7
#    Implement your function on `mtcars` (the original whole dataset) and save the
#    output linear model into `fit_mtcars`.
#    Then get the confidence interval of this model by using `confint()`, save your
#    confidence interval into `ci_normal`.
## Do not modify this line!
```

```
library(tidyverse)
library(broom)
set.seed(0)
seed = .Random.seed
model_mtcars = function(df){
  lm(mpg ~ wt, data = df)
}
fit_mtcars = model_mtcars(mtcars)
ci_normal = confint(fit_mtcars)


# 2. Now let's get the bootstrap index for permutations.
#   Create a function factory `boot_permute_factory` that takes an input `df`, calculate
#   the number of rows of `df` and returns a function to resample from `df` with
#   replacement. For example, `boot_permute_factory(mtcars)` returns a function that
#   takes no input, but returns the resampled data from `df`.
#   Note that for bootstrap resample, your resampled tibble should have the same
#   row numbers with your original tibble.
#   For instance, if you runs:
#   `myboot <- boot_permute_factory(tibble(x = 1:20)); set.seed(0); myboot()`,
#   then the first few rows of output will be
#   # A tibble: 20 x 1
#   x
#   <int>
#     18
#     6
#     8
#     12
#     19
#     5
#   Now implement your `boot_permute_factory()` on `mtcars` and save the output into
#   `boot_permute_mtcars`.
#   Choose number of bootstrap as 200 and save it to `B`.
#   Use `set.seed()` to set the random seed to 0 and save your seed into `seed`
#   using `.Random.seed`.
#   Generate a tibble `result_boot`, storing all information about the linear model for
#   the bootstrap resampled tibble, to do this, you can use:
#   - `tibble()` to create a tibble including
#       column `b` indicates the row number, from 1 to `B`
#       column `permute` indicates the information of linear regressionin each bootstrap
#   - `map()` to broadcast your operations.
#   - `boot_permute_mtcars()` to get permutations of dataset for each `b`
#   - `model_mtcars()` to fit a linear model on each permutation
#   - `tidy()` to extract the information in the linear model
#   Your first few rows of `result_boot` should look like this:
#   A tibble: 200 x 2
#     b   permute
#   <int> <list>
#     1 <tibble [2 × 5]>
#     2 <tibble [2 × 5]>
#     3 <tibble [2 × 5]>
#     4 <tibble [2 × 5]>
```

```
#     5 <tibble [2 × 5]>
#     6 <tibble [2 × 5]>
#     7 <tibble [2 × 5]>
#     8 <tibble [2 × 5]>
#     9 <tibble [2 × 5]>
#    10 <tibble [2 × 5]>
#    # … with 190 more rows
#    and Your first tibble in permute should be :
#    # A tibble: 2 x 5
#    term        estimate std.error statistic  p.value
#    <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#    (Intercept)    34.1     1.77     19.3  1.89e-18
#    wt             -4.71    0.515    -9.16 3.40e-10
## Do not modify this line!
boot_permute_factory = function(df){
  force(df)
  n = nrow(df)
  function(){
    sample_n(df, n, replace = TRUE)
  }
}
boot_permute_mtcars = boot_permute_factory(mtcars)
B = 200
set.seed(0)
result_boot = tibble(b = 1:B) %>%
  mutate(permute = map(b, ~boot_permute_mtcars())) %>%
  mutate(permute = map(permute, model_mtcars)) %>%
  mutate(permute = map(permute, tidy))
# 3. Now let's change our strategy and use another bootstrap way: the parametric
#    bootstrap[https://en.wikipedia.org/wiki/Bootstrapping_%28statistics%29
#Parametric_bootstrap]
#    Create a function factory `boot_parametric_factory` that takes two inputs `df` and
#    `model`, then fit the `model` on `df`, save the predicted values and residuals
#    using `fitted()` and `resid()`, and in the end returns a function to resample
#    from residuals with replacement, and to return the values of prediction plus the
#    resampled residuals.
#    For example, `boot_parametric_factory(mtcars, model_mtcars)` returns a function that
#    takes no input, but returns the sum of predicted values and resampled residuals.
#    If you run `set.seed(0); myboot <- boot_parametric_factory(tibble(wt = rnorm(10), mpg =
wt + rnorm(10)), model_mtcars); myboot()`,
#    the output should be:
#    # A tibble: 10 x 2
#    wt    mpg
#    <dbl>  <dbl>
#     1.26     0.846
#    -0.326    0.154
#     1.33     1.27
#     1.27     0.164
#     0.415   -0.267
#    -1.54    -1.95
#    -0.929   -1.73
#    -0.295    0.684
```

```
#   -0.00577 -0.247
#    2.40     1.14
#   Use `set.seed()` to set the random seed to 0 and save your seed into `seed`
#   using `.Random.seed`.
#   Now implement your `boot_permute_factory()` on `mtcars` and `model_mtcars` and
save
#   the output into `boot_parametric_mtcars`.
#   Now add a new column `parametric` to `result_boot`, storing all information about
#   linear model for each parametric bootstrap sample.
#   To do this, you can use:
#   - `mutate()` to add the column `parametric`
#   - `map()` to braodcast your operations.
#   - `boot_parametric_mtcars()` to get parametric bootstrap samples
#   - `model_mtcars()` to fit a linear model on each parametric permutation
#   - `tidy()` to extract the information of linear regression
#   Your first few rows of `result_boot` should now look like this:
#   # A tibble: 200 x 3
#   b permute         parametric
#   <int> <list>          <list>
#      1 <tibble [2 × 5]> <tibble [2 × 5]>
#      2 <tibble [2 × 5]> <tibble [2 × 5]>
#      3 <tibble [2 × 5]> <tibble [2 × 5]>
#      4 <tibble [2 × 5]> <tibble [2 × 5]>
#      5 <tibble [2 × 5]> <tibble [2 × 5]>
#      6 <tibble [2 × 5]> <tibble [2 × 5]>
#      7 <tibble [2 × 5]> <tibble [2 × 5]>
#      8 <tibble [2 × 5]> <tibble [2 × 5]>
#      9 <tibble [2 × 5]> <tibble [2 × 5]>
#     10 <tibble [2 × 5]> <tibble [2 × 5]>
#      … with 190 more rows
#   The first tibble in the `parametric` column should be:
#   # A tibble: 2 x 5
#   term        estimate std.error statistic  p.value
#   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#   (Intercept)    36.1      1.47       24.6 1.82e-21
#   wt             -5.30     0.436     -12.2 4.05e-13
## Do not modify this line!
boot_parametric_factory = function(df, model){
  force(df)
  force(model)
  m = model(df)
  fit = fitted(m)
  resid = resid(m)
  function(){
    df$mpg = fit + sample(resid, length(resid), replace=TRUE)
    df
  }
}
#按照描述写函数
set.seed(0)
seed = .Random.seed
boot_parametric_mtcars = boot_parametric_factory(mtcars, model_mtcars)
```

```
result_boot = result_boot %>%
  mutate(parametric = map(b, ~boot_parametric_mtcars())) %>% #对第一步操作和之前一
样
  mutate(parametric = map(parametric, model_mtcars)) %>%
  mutate(parametric = map(parametric, tidy)) %>%
  select(b, permute, parametric)

# 4. Let's extract the information from `result_boot`, create a dataframe `result_boot_tidy`
#    by following these steps:
#    - Use `pivot_longer()` to change the shape of tibble:
#      - Specify `cols = -b` to keep column `b` and transforme on the other columns
#      - Specify `names_to = boot` to differentiate between two bootstrap methods
#    - Use `unnest()` to extract the `term`, `estimate`, `std.error`, `statistic` and
#      `p.value` from each linear model
#    - Use `mutate()` and `ifelse()` to add a column `full` in `result_boot_tidy` which
#      indicate the estimated value of intercept or slope on the whole dataset. Recall
#      that we've fit a model on the original dataset and saved the model into
#      `fit_mtcars`.
#    Your first few rows of `result_boot_tidy` should look like:
#    # A tibble: 800 x 8
#    b     boot      term       estimate std.error statistic  p.value  full
#    <int> <chr>     <chr>         <dbl>    <dbl>    <dbl>    <dbl> <dbl>
#       1 permute    (Intercept)    34.1     1.77     19.3  1.89e-18 37.3
#       1 permute    wt             -4.71    0.515    -9.16 3.40e-10 -5.34
#       1 parametric (Intercept)    39.2     2.04     19.2  2.09e-18 37.3
#       1 parametric wt             -5.76    0.607    -9.48 1.56e-10 -5.34
#       2 permute    (Intercept)    36.3     1.59     22.9  1.56e-20 37.3
#       2 permute    wt             -4.94    0.439   -11.3  2.70e-12 -5.34
#       2 parametric (Intercept)    38.5     1.55     24.9  1.43e-21 37.3
#       2 parametric wt             -5.75    0.462   -12.4  2.23e-13 -5.34
#       3 permute    (Intercept)    33.7     2.08     16.3  2.05e-16 37.3
#       3 permute    wt             -4.44    0.596    -7.45 2.68e- 8 -5.34
#       … with 790 more rows
## Do not modify this line!
result_boot_tidy = result_boot %>%
  pivot_longer(cols = -b, names_to = 'boot') %>%
  unnest(value) %>%
  mutate(full = ifelse(term == 'wt', fit_mtcars$coefficients[2], fit_mtcars$coefficients[1]) )



# 5. Now let's calculate the bootstrap confidence intervals.
#    Save your confidence intervals in a tibble `ci_bootstrap`. The format should look
#    like this:
#    # A tibble: 8 x 5
#    Groups:   boot [2]
#    boot      term       name      lower upper
#    <chr>     <chr>      <chr>     <dbl> <dbl>
#    parametric (Intercept) basic     34.1  41.6
#    parametric (Intercept) percentile 33.0  40.4
#    parametric wt          basic     ...   ...
```

```
#   parametric wt         percentile ...   ...
#   permute    (Intercept) basic      ...   ...
#   permute    (Intercept) percentile ...   ...
#   permute    wt         basic      ...   ...
#   permute    wt         percentile ...   ...
#   To do this, you can use:
#   - `group_by()` to group the `result_boot_tidy` by `boot` and `term`
#   - `summarize()` to add summay columns
#     - `basic` will be the summary that stores the bootstrap confidence intervals using
#       standard bootstrap interval.
#       You can refer to the function `ci_basic_bootstrp()` on page 40 of
#       the lecture slides to get the basic confidence interval
#     - `percentile` will be the summary that stores the bootstrap confidence
#       intervals using the percentile method.
#       You can refer to the function `ci_percentile_bootstrp()` on page 40 of
#       the lecture slides to get the basic confidence interval
#   - `pivot_longer` to save one confidence interval in one row.
#     - Specify `cols = c("basic,"percentile")`
#   - `unnest()` to get the lower and upper bound.
#
#
#
## Do not modify this line!

ci_basic_bootstrap <- function(thetab, thetahat, alpha = 0.95) {
  qb <- quantile(thetab, c((1 - alpha) / 2, (1 + alpha) / 2))
  tibble(
    lower = 2 * thetahat - qb[2],
    upper = 2 * thetahat - qb[1]
  )
}
ci_percentile_bootstrap <- function(thetab, thetahat, alpha = 0.95) {
  qb <- quantile(thetab, c((1 - alpha) / 2, (1 + alpha) / 2))
  tibble(
    lower = qb[1],
    upper = qb[2]
  )
}

ci_bootstrap <- result_boot_tidy %>%
  group_by(boot, term) %>%
  summarize(
    basic = list(ci_basic_bootstrap(estimate, full[1])),
    percentile = list(ci_percentile_bootstrap(estimate, full[1]))
  ) %>%
  pivot_longer(cols = c("basic", "percentile")) %>%
  unnest(value)
#在照抄两个公式后，apply

# HW10: Bootstrap for hypothesis testing
#
# In 1882 Simon Newcomb performed an experiment to measure the speed of
```

```
# light. The numbers below represent the measured time it took for light
# to travel from Fort Myer on the west bank of the Potomac River to a fixed
# mirror at the foot of the Washington monument 3721 meters away.
# In the units in which the data are given, the currently accepted “true”
# speed of light is 33.02. (To convert these units to time in the millionths
# of a second, multiply by 10-3 and add 24.8.)
# The recorded values were :
# 28, -44, 29, 30, 26, 27, 22, 23, 33, 16,
# 24, 29, 24, 40, 21, 31, 34, -2, 25, 19
#
# Does the data support the current accepted speed of 33.02?
#
# 1. (1)
#    Create a vector `speed` and save the values
#    `28, -44, 29, 30, 26, 27, 22, 23, 33, 16, 24, 40, 21, 31, 34, -2, 25, 19`
#    in it (in this order).
#    (2)
#    Draw a histogram using `ggplot()` and save it in `speed_hist`.
#    Note that there are some outliers.
#    You can use:
#    - `ggplot()` to initialize a ggplot object.
#      Set `data` to `tibble(speed = speed)`.
#      Set `mapping` to `aes(speed)`.
#    - `geom_histogram()` to plot the histogram.
#      - Set `bins` to `30`.
#      - Set `color` to `white`.
#    - `labs()`
#      - Set `title` to `"Histogram of speed"`.
#      - Set `subtitle` to `"There are outliers."`.
#      - Set `x` to `Speed`.
#      - Set `y` to `Count`.
#    - `theme_light()` to set the theme to light.
## Do not modify this line!
library(ggplot2)
library(tidyverse)
speed = c(28, -44, 29, 30, 26, 27, 22, 23, 33, 16, 24, 29, 24, 40, 21, 31, 34, -2, 25, 19)
data = tibble(speed = speed)
speed_hist = data %>%
  ggplot(mapping = aes(x = speed)) +
  geom_histogram(bins = 30, color = 'white') +
  labs(title = "Histogram of speed",
       subtitle = "There are outliers.",
       x = "Speed", y = "Count") +
  theme_light()


# 2. Draw a qq plot using `ggplot()` and save it in `speed_qq`.
#    Note that the distribution is not normal.
#    You can use:
#    - `ggplot()` to initialize a ggplot object.
#      Set `data` to `tibble(speed = speed)`.
#      Set `mapping` to `aes(speed)`.
```

```
#   - `stat_qq()` to plot the dots.
#   - `stat_qq_line` to plot the line.
#   - `labs()`
#     - Set `title` to `"QQ plot of speed"`.
#     - Set `subtitle` to `"The data is not normally distributed."`.
#   - `theme_light()` to set the theme to light.
## Do not modify this line!
speed_qq = data %>%
 ggplot(mapping = aes(sample = speed)) +
 stat_qq() +
 stat_qq_line() +
 labs(title = "QQ plot of speed",
      subtitle = "The data is not normally distributed.") +
 theme_light()


# 3. When performing a t-test, we assume that the population of
#   measurements is normally distributed.
#   Since this is not the case here, our tests will at best be approximations.
#   But with this small sample size, and with such a severe departure from
#   normality, we can't guarantee a good approximation.
#   The bootstrap offers one approach.
#   The null and alternative hypotheses are :
#   `H0: mean = 33.02`
#   `Ha: mean is not equal to 33.02`
#   The significant level is 5%.
#   (Note: The significance level is the probability of rejecting the null
#   hypothesis when it is true)
#   We now need the p-value, but to do this we need to know the sampling
#   distribution of our test statistic when the null hypothesis is true.
#   Our approach is to perform a simulation under
#   conditions in which we know the null hypothesis is true.
#   (1)
#   What we'll do is use our data to represent the population, but first
#   we shift it over so that the mean really is 33.02.
#   Write a function factory `boot_null(x, mu0)` to do this.
#   The input `x` is a vector.
#   The input `mu0` is the mean under null hypothesis.
#   This function shifts the mean of input `x` to `mu0` and returns a function that
#   samples from the shifted `x` with replacement.
#   For example, for seed `5206`, `boot_null(speed, 0)()` would return
#    [1] -23.75   2.25   7.25   7.25 -65.75   2.25   7.25   7.25   4.25   5.25
#    [11] -5.75   4.25 -65.75  -5.75 -65.75   8.25   7.25  18.25   0.25  18.25
#   (2)
#   Set `mu0` to `33.02`.
#   Use `boot_null()` to create a function `boot_speed_null()` that has no input,
#   shift the mean of `speed` to `mu0` and sample from it with replacement.
#   (3)
#   Bootstrap using `boot_speed_null()` 1000 times.
#   Set the random seed to zero and save the random seed vector to `seed`.
#   (hint: use the command `seed <- .Random.seed`).
#   Save the bootstrap samples in tibble `bstrap` with 3 columns `b`, `sample_null`,
```

```
#   and `muhat_null`.
#   - `b` indicates the row number, from 1 to 1000.
#   - `sample_null` contains one bootstrap sample in each row.
#   - `muhat_null` is the mean of the bootstrap sample.
#   You can use `tibble()` to create a tibble.
#   - Set column `b` to 1 to 1000.
#   - Use column `b` and `map()` to compute column `sample_null`.
#   - Use column `sample_null` and `map_dbl()` to compute column `muhat_null`.
#   Tibble `bstrap` should print to:
#   # A tibble: 1,000 x 3
#         b sample_null muhat_null
#     <int> <list>          <dbl>
#   1     1 <dbl [20]>       34.5
#   2     2 <dbl [20]>       31.5
#   3     3 <dbl [20]>       32.7
#   4     4 <dbl [20]>       37.7
#   5     5 <dbl [20]>       32.6
#   6     6 <dbl [20]>       37.4
#   7     7 <dbl [20]>       24.0
#   8     8 <dbl [20]>       36.7
#   9     9 <dbl [20]>       35.7
#  10    10 <dbl [20]>       32.8
#  # … with 990 more rows
#   `bstrap[1, 2][[1]]` should print to:
#    [1]  51.27  41.27  33.27  39.27 -32.73  35.27  51.27   9.27  36.27  39.27
#    [11] 27.27  51.27  27.27  33.27 44.27  32.27  37.27  44.27  51.27  37.27
## Do not modify this line!
boot_null <- function(x, mu0) {
  muhat <- mean(x)
  x <- x - muhat + mu0#因为要shift “到” mu0,所以要先减去，再加上
  function() {
    sample(x, replace = TRUE)
  }
}
mu0 = 33.02
boot_speed_null = boot_null(speed, mu0)
set.seed(0)
seed = .Random.seed
bstrap = tibble(b = 1:1000) %>%
  mutate(sample_null = map(b, ~boot_speed_null())) %>%
  mutate(muhat_null = map_dbl(sample_null, mean))

# 4. Draw a histogram for column `muhat_null` in `bstrap` and save it in
#   `bstrap_hist`. You can use:
#   - `ggplot()` to initialize a ggplot object.
#     Set `data` to `bstrap`.
#     Set `mapping` to `aes(muhat_null)`.
#   - `geom_histogram()` to plot the histogram.
#     - Set `bins` to `30`.
#     - Set `color` to `white`.
#   - `labs()`
#     - Set `title` to `"Histogram of bootstrap samples"`.
```

```
#     - Set `subtitle` to `"The distribution is not normal."`.
#     - Set `x` to `Samples`.
#     - Set `y` to `Count`.
#    - `theme_light()` to set the theme to light.
## Do not modify this line!
bstrap_hist = bstrap %>%
  ggplot(mapping = aes(muhat_null)) +
  geom_histogram(bins = 30, color = 'white') +
  labs(title = "Histogram of bootstrap samples",
       subtitle = "The distribution is not normal.",
       x = "Samples", y = "Count") +
  theme_light()



# 5. (1)
#    Write a function `boot_pval(muhat, mu0, muhat_null)` to compute p value.
#    The input `muhat` is the mean of the original samples.
#    The input `mu0` is the mean under null hypothesis.
#    The input `muhat_null` is a vector contains the mean of bootstrap samples.
#    The output is the two sided p value.
#    Note the p value is the proportion of samples that has mean as extreme as or
#    more extreme as the mean of original sample.
#    For example, `boot_pval(30, 31, speed)` should be `0.9`.
#    (2)
#    Compute the p value for our case and save it into `p_value`.
#    That is `muhat` is the mean of `speed`, `mu0` is `33.02`, and `muhat_null`
#    is the column `muhat_null` in `bstrap`.
## Do not modify this line!
boot_pval <- function(muhat=30, mu0=31, muhat_null) {
  mu_diff <- abs(muhat - mu0)
  mean(muhat_null < mu0 - mu_diff) + mean(muhat_null > mu0 + mu_diff)
}#这里的思路是sample中多少百分比的在两个border之外
p_value = boot_pval(mean(speed), 33.02, bstrap$muhat_null)
# 6. (1)
#    Note that there were two extreme observations: -44 and -2.
#    The t-test is notoriously susceptible to being influenced
#    by extreme observations because it depends on the sample average.
#    Let's take those two values out and save the vectors of
#    the values we keep into `newspeed`. (It should be equal to
#    `28, 29, 30, 26, 27, 22, 23, 33, 16, 24, 29, 24, 40, 21, 31, 34, 25, 19`)
#    (2)
#    Draw a qq plot for `newspeed` and save it in `newspeed_qq`.
#    Note that the distribution is not normal.
#    You can use:
#    - `ggplot()` to initialize a ggplot object.
#      Set `data` to `tibble(speed = newspeed)`.
#      Set `mapping` to `aes(speed)`.
#    - `stat_qq()` to plot the dots.
#    - `stat_qq_line` to plot the line.
#    - `labs()`
#      - Set `title` to `"QQ plot of newspeed"`.
```

```
#     - Set `subtitle` to `"The data is now normally distributed."`.
#    - `theme_light()` to set the theme to light.
## Do not modify this line!
newspeed = c(28, 29, 30, 26, 27, 22, 23, 33, 16, 24, 29, 24, 40, 21, 31, 34, 25, 19)
newspeed_qq = tibble(newspeed = newspeed) %>%
  ggplot(mapping = aes(sample = newspeed)) +
  stat_qq() +
  stat_qq_line() +
  labs(title = "QQ plot of newspeed",
       subtitle = "The data is now normally distributed.") +
  theme_light()


# 7. Use `t.test()` to conduct a two-sided t test for `newspeed` to
#    test whether the mean of `newspeed` equals to `33.02`.
#    Save the result into `ttest`. It should be the output of `t.test()`.
#    We can see that the null hypothesis is rejected.
## Do not modify this line!
ttest<-t.test(newspeed,mu=33.02)

# 8. Now we will use bootstrap again.
#    (1)
#    Use `boot_null()` to create a function `boot_speed_null2()` that has no input,
#    shift the mean of `newspeed` to `33.02` and sample from it with replacement.
#    (2)
#    Bootstrap using `boot_speed_null2()` 1000 times.
#    Set the random seed to zero and save the random seed vector to `seed`.
#    (hint: use the command `seed <- .Random.seed`).
#    Save the bootstrap samples in tibble `bstrap2` with 3 columns `b`, `sample_null`,
#    and `muhat_null`.
#    - `b` indicates the row number, from 1 to 1000.
#    - `sample_null` contains one bootstrap sample in each row.
#    - `muhat_null` is the mean of the bootstrap sample.
#    You can use `tibble()` to create a tibble.
#    - Set column `b` to 1 to 1000.
#    - Use column `b` and `map()` to compute column `sample_null`.
#    - Use column `sample_null` and `map_dbl()` to compute column `muhat_null`.
#    Tibble `bstrap2` should print to:
#    # A tibble: 1,000 x 3
#          b sample_null muhat_null
#      <int> <list>          <dbl>
#    1     1 <dbl [18]>       30.0
#    2     2 <dbl [18]>       32.3
#    3     3 <dbl [18]>       31.9
#    4     4 <dbl [18]>       34.7
#    5     5 <dbl [18]>       33.2
#    6     6 <dbl [18]>       30.5
#    7     7 <dbl [18]>       35.0
#    8     8 <dbl [18]>       33.0
#    9     9 <dbl [18]>       30.0
#   10    10 <dbl [18]>       32.2
#    # … with 990 more rows
```

```
#   `bstrap2[1, 2][[1]]` should print to
#    [1] 27.29778 32.29778 29.29778 34.29778 35.29778 35.29778 27.29778 25.29778
#    [9] 34.29778 30.29778 27.29778 30.29778 29.29778 22.29778 37.29778
#    [16] 33.29778 22.29778 27.29778
#   (3)
#   Compute the two sided p value and save it into `p_value2`.
#   We see that this time the p value is much smaller.
## Do not modify this line!
boot_speed_null2 = boot_null(newspeed, 33.02)
set.seed(0)
seed = .Random.seed
bstrap2 = tibble(b = 1:1000) %>%
  mutate(sample_null = map(b, ~boot_speed_null2())) %>%
  mutate(muhat_null = map_dbl(sample_null, mean))
p_value2 = boot_pval(mean(newspeed), 33.02, bstrap2$muhat_null)

# HW10: MC Integration of a power law
#
# This exercise aims at practicing Monte-Carlo Integration and observing the
# benefits of a couple techniques.
#
# 1. In this question:
#   - write a function factory `factory` that takes an input `k` and returns
#     the function `x -> x^k` - don't forget to use `force()` in it,
#   - assign the cubic function to `f_3` by calling `factory(3)`,
#   - assign the value of the integral of `f_3` between 0 and 1 to
#     `integral_f_3` (don't compute it programmatically, you can
#     calculate it directly!)
#   For example, `factory(2)(3)` should be equal to `9` and `f_3(3)` should be
#   equal to `27`
## Do not modify this line!
factory = function(k){
  force(k)
  function(x){
    x^k
  }
}
f_3 = factory(3)
integral_f_3 = f_3(1)^4/4

# 2. Load the `tibble` and `dplyr` packages.
#   Let's sample from a uniform distribution and use the samples to build a
#   MC estimator of the integral. In this question:
#   - assign a sample size of 10,000 to `n`,
#   - call `set.seed(11)`,
#   - assign `.Random.seed` to variable `seed`,
#   - create a tibble `df` using `tibble()`, containing:
#     - a column `s`: a sample of `n` data points from a uniform distribution
#       between 0 and 1 generated using `runif()`,
#     - a column `f_3` obtained by applying `f_3()` to `s`
#   - compute the estimator and confidence interval using `summarize()`
#     from `df` and assign the resulting tibble to `ci_unif`; you will need
```

```
#     to:
#     - compute the MC estimation of the integral of `f_3` between 0 and 1
#       using `mean()` and `f_3` and name it `integral`,
#     - compute the MC standard error using `sqrt()`, `var()`, `f_3` and `n`
#       and assign it to `se`,
#     - use `integral` and `se` to compute a 95% confidence interval centered
#       on `integral` using the tabulated value of `1.96` and assign bounds of
#       the interval to `lower` and `upper`
#     The shape of `ci_unif` will be `1x4` with column names
#     `(integral, se, lower, upper)`.
#   `df` should print to:
#   # A tibble: 10,000 x 2
#           s     f_3
#       <dbl>   <dbl>
#    1 0.277    2.13e- 2
#    2 0.000518 1.39e-10
#    3 0.511    1.33e- 1
#    4 0.0140   2.77e- 6
#    5 0.0647   2.71e- 4
#    6 0.955    8.71e- 1
#    7 0.0865   6.47e- 4
#    8 0.290    2.44e- 2
#    9 0.881    6.83e- 1
#   10 0.123    1.87e- 3
#   # … with 9,990 more rows
## Do not modify this line!
library(tidyverse)
n = 10000
set.seed(11)
seed= .Random.seed
df = tibble(s = runif(n)) %>%
  mutate(f_3 = f_3(s))
ci_unif <- df %>%
  summarize(
    integral = mean(f_3),
    se = sqrt(var(f_3) / n),
    lower = integral - 1.96 * se,
    upper = integral + 1.96 * se
  )
# 3. Now, let's use antithetic examples to reduce the variance of the
#   estimator; using the fact that if `X ~ unif(0, 1)`, `1-X ~ unif(0, 1)`.
#   In this question:
#   - add the column `f_3_anti` applying `f_3()` to antithetic samples to `df`
#     using `mutate()` and assign the resulting tibble to `df2`,
#   - compute the estimator and and confidence interval using `summarize()`
#     from `df2` and assign the resulting tibble to `ci_anti`; you will need
#     to:
#     - compute the MC estimation of the integral of `f_3` between 0 and 1
#       using `mean()`, `f_3` and `f_3_anti` - name it `integral`,
#     - compute the MC standard error using `sqrt()`, `var()`, `f_3`,
#       `f_3_anti` and `n` and assign it to `se`,
#     - use `integral` and `se` to compute a 95% confidence interval centered
```

```
#      on `integral` using the tabulated value of `1.96` and assign bounds of
#      the interval to `lower` and `upper`
#    The shape of `ci_anti` will be `1x4` with column names
#    `(integral, se, lower, upper)`.
#   `df2` should print to:
#   # A tibble: 10,000 x 3
#           s     f_3  f_3_anti
#       <dbl>   <dbl>    <dbl>
#    1 0.277    2.13e- 2 0.378
#    2 0.000518 1.39e-10 0.998
#    3 0.511    1.33e- 1 0.117
#    4 0.0140   2.77e- 6 0.958
#    5 0.0647   2.71e- 4 0.818
#    6 0.955    8.71e- 1 0.0000920
#    7 0.0865   6.47e- 4 0.762
#    8 0.290    2.44e- 2 0.358
#    9 0.881    6.83e- 1 0.00170
#   10 0.123    1.87e- 3 0.674
#   # … with 9,990 more rows
## Do not modify this line!
df2 = df %>% mutate(f_3_anti = f_3(1-s))#antithetic samples的含义
ci_anti <- df2 %>%
 summarize(
  integral = mean((f_3 + f_3_anti)/2),
  se = sqrt(var((f_3 + f_3_anti)/2) / n),
  lower = integral - 1.96 * se,
  upper = integral + 1.96 * se
 )

# 4. Now, let's use control variates to reduce the variance of the estimator in
#   question 2; using the known value of the integral of the square function
#   between 0 and 1. In this question:
#   - use `factory()` to assign the square function to `f_2`,
#   - assign the value of the integral of `f_2` between 0 and 1 to
#     `integral_f_2` (don't compute it programmatically!),
#   - add the column `f_2` applying `f_2()` to `s` in `df` using `mutate()`
#     and assign the resulting tibble to `df3`,
#   - compute the estimator and and confidence interval using `summarize()`
#     from `df3` and assign the resulting tibble to `ci_control`; you will
#     need to:
#     - compute the negative ratio of the covariance of `f_3` and `f_2`, and
#       `var(f_2)` - and name it `c`,
#     - compute the MC estimation of the integral of `f_3` between 0 and 1
#       using `mean()`, `f_3` and `c`, `f_2` and `integral_f_2` - and name it
#       `integral`,
#     - compute the MC standard error using `sqrt()`, `var()`, `f_3`,
#       `f_2`, `integral_f_2` and `n` and assign it to `se`,
#     - use `integral` and `se` to compute a 95% confidence interval centered
#       on `integral` using the tabulated value of `1.96` and assign bounds of
#       the interval to `lower` and `upper`
#     The shape of `ci_control` will be `1x5` with column names
#     `(c, integral, se, lower, upper)`.
```

```
#   `df3` should print to:
#   # A tibble: 10,000 x 4
#           s     f_3  f_3_anti       f_2
#       <dbl>   <dbl>    <dbl>     <dbl>
#    1 0.277    2.13e- 2 0.378    0.0769
#    2 0.000518 1.39e-10 0.998    0.000000269
#    3 0.511    1.33e- 1 0.117    0.261
#    4 0.0140   2.77e- 6 0.958    0.000197
#    5 0.0647   2.71e- 4 0.818    0.00418
#    6 0.955    8.71e- 1 0.0000920 0.912
#    7 0.0865   6.47e- 4 0.762    0.00748
#    8 0.290    2.44e- 2 0.358    0.0841
#    9 0.881    6.83e- 1 0.00170   0.776
#   10 0.123    1.87e- 3 0.674    0.0152
#   # … with 9,990 more rows
## Do not modify this line!
f_2 = factory(2)
integral_f_2 = 1/3
df3 = df2 %>% mutate(f_2 = f_2(s))
ci_control = df3 %>%
  summarize(c = -cov(df3$f_3, df3$f_2)/var(df3$f_2),
            integral = mean(f_3 + c * (f_2 - integral_f_2)),
            se = sqrt(var(f_3 + c * (f_2 - integral_f_2))/n),
            lower = integral - 1.96*se,
            upper = integral + 1.96*se)
#slide的公式，h(x)为f_2, g(x) = f_3
# 5. Load the `tidyr` package.
#   Compute a tibble containing the estimation errors (the difference between
#   the MC estimation and the actual value of the integral) of the three
#   different estimators for each step of the MC sequence generation.
#   For this we first need to compute cumulative statistics of the MC
#   sequences at each step of the generation process.
#   In this question:
#   - compute the `cum_stats` tibble from `df3` using `mutate()` to add the
#     following columns - you can compute all columns only using the existing
#     columns of `df3` and the function `cummean()`:
#     - `cummean_f_3`: the cumulative mean of `f_3`,
#     - `cummean_f_3_anti`: the cumulative mean of `f_3_anti`,
#     - `cummean_f_2`: the cumulative mean of `f_2`,
#     - `cumvar_f_2`: the cumulative variance of `f_2` computed using
#       `cummean(f_2^2)` and `cummean_f_2`,
#     - `cummcov_f_2_f_3`: the cumulative covariance of `f_2` and `f_3`
#       computed using the cumulative mean of `f_2 * f_3` minus the product
#       ot the cumulative means of each,
#     - `cum_c`: the cumulative value of `c` (using only already generated
#       points to compute `c` at each step) - you can use `cumvar_f_2` and
#       `cumcov_f_2_x_f_3` to compute `cum_c`,
#     - filter out rows in which `cum_c` is `NaN` - this happens if the
#       sample variance of `f_2` is `0` (ie. for step 1 when only one point is
#       generated)
#   - compute the `errors` tibble containing the estimation errors from
#     `cum_stats`:
```

```
#    - use `mutate()` to construct the following columns:
#      - `sequence` contains all integers from `1` to `n - 1`,
#      - `unif` contains the difference of the MC estimation of the vanilla
#        uniform samples and the actual integral at each step (you can compute
#        it using `cummean_f_3` and `integral_f_3`),
#      - `antithetic` contains the difference of the MC estimation of the
#        antithetic samples and the actual integral at each step (you can
#        compute it using `cummean_f_3, cummean_f_3_anti` and `integral_f_3`),
#      - `control` contains the difference of the MC estimation using control
#        variates and the actual integral at each step (you can compute it
#        using `cummean_f_3`, `cum_c`, `cummean_f_2`, `integral_f_2` and
#        `integral_f_3`)
#    - keep only the columns you just added using `select()`,
#    - use `pivot_longer()` to tidy the data by placing the different types of
#      estimation in one column and the error values in another as in the
#      following
#   `cum_stats %>% select(-c(s, f_3, f_3_anti, f_2))` should print to:
#   # A tibble: 9,999 x 6
#     cummean_f_3 cummean_f_3_anti cummean_f_2 cumvar_f_2 cumcov_f_2_x_f_3
cum_c
#           <dbl>            <dbl>      <dbl>      <dbl>            <dbl>  <dbl>
#   1      0.0107           0.688      0.0384    0.00148         0.000410 -0.277
#   2      0.0515           0.498      0.113     0.0120          0.00632  -0.528
#   3      0.0386           0.613      0.0844    0.0113          0.00583  -0.514
#   4      0.0309           0.654      0.0684    0.0101          0.00515  -0.510
#   5      0.171            0.545      0.209     0.107           0.103    -0.957
#   6      0.147            0.576      0.180     0.0969          0.0922   -0.952
#   7      0.131            0.549      0.168     0.0858          0.0819   -0.955
#   8      0.193            0.488      0.236     0.113           0.106    -0.940
#   9      0.174            0.507      0.214     0.106           0.0991   -0.937
#  10      0.158            0.512      0.197     0.0989          0.0927   -0.937
#  # … with 9,989 more rows
#   `errors` should print to:
#    # A tibble: 29,997 x 3
#      sequence sample_type   value
#         <int> <chr>         <dbl>
#    1        1 unif        -0.239
#    2        1 antithetic   0.0993
#    3        1 control     -0.158
#    4        2 unif        -0.199
#    5        2 antithetic   0.0246
#    6        2 control     -0.0818
#    7        3 unif        -0.211
#    8        3 antithetic   0.0758
#    9        3 control     -0.0835
#   10        4 unif        -0.219
#    # … with 29,987 more rows
## Do not modify this line!
cum_stats = df3 %>%
  mutate(cummean_f_3 = cummean(f_3),
         cummean_f_3_anti = cummean(f_3_anti),
         cummean_f_2 = cummean(f_2),
```

```
      cumvar_f_2 = cummean(f_2^2) - cummean_f_2^2,
      cumcov_f_2_x_f_3 = cummean(f_2*f_3) - cummean_f_3*cummean_f_2,
      cum_c = -cumcov_f_2_x_f_3/cumvar_f_2) %>%#cum_c公式和上面的c是一样的
  filter(!is.na(cum_c))
errors = cum_stats%>%
  mutate(sequence = 1:(n-1),
      unif = cummean_f_3 - integral_f_3,
      antithetic = (cummean_f_3 + cummean_f_3_anti)/2 - integral_f_3, #这条与下条前半部分来自slides，为相应公式
      control = cummean_f_3  + cum_c * (cummean_f_2 - integral_f_2) - integral_f_3)%>%
  select(sequence:control) %>%
  pivot_longer(-sequence, values_to = 'value', names_to = 'sample_type')



# 6. Load the `ggplot2` package.
#    Generate an error plot `error_plot` from `errors` showing the error for
#    each type of estimation at each step.
#    To do so, you can:
#    - call `ggplot()` from `errors` with the aesthetic `x = sequence`,
#      `y = value` and `colour = sample_type`,
#    - add `geom_line()`,
#    - restrict the shown sequence values up to the 1500th step included using
#      `coord_cartesian(xlim = c(0, 1500))`,
#    - feed the tibble to `ggplot()`,
#    - use `labs()` for the following labels:
#      - `y = "MC error"`,
#      - `colour = "Sample type"`,
#      - `title = "Using antithetic samples and control variates accelerates convergence and
reduces variance"`
#    - add `theme_light()`
#
## Do not modify this line!

library(ggplot2)
error_plot = errors %>%
  ggplot(mapping = aes(x = sequence, y = value, color = sample_type)) +
  geom_line() +
  coord_cartesian(xlim = c(0, 1500)) +
  labs(y = "MC error",
       color = "Sample type",
       title = "Using antithetic samples and control variates accelerates convergence and
reduces variance") +
  theme_light()



# HW10: Monte Carlo Integration
#
# In this exercise, we will walk you through the process of using Monte Carlo
# method to approximate complicated integrals. There are three independent parts
# in this exercise: 1-2, 3-5 and 6-7.
```

```
# We suggest the functions you can use to create the tibbles and the plots, but
# you are free to use the methods you are the most comfortable with.
# Make sure that the outputs look exactly like the ones you are supposed to create.
#
# Throughout the exercise:
#   - Use `%>%` to structure your operations.
#   - Use `theme_light()` for the plots.
#   - Do NOT use `for`, `while` or `repeat` loops.
#   - When defining functions, you do not need to follow the exact process as
#     suggested by the instructions. However, make sure you are using the same
#     parameter settings, inputs and outputs in order to pass the test.
#
# 1. Load the package `tidyverse`.
#   Let X ~ Cauchy(0, 1) be a standard Cauchy random variable.
#   We would like to approximate the probability P(X > 2), i.e., the integral
#   of g(x) = `1 / (pi * (1 + x^2))` over x from 2 to inf.
#   First, let's compute the exact probability as a reference using `pcauchy()`
#   and store the result as a float `p1`.
#   Next, we will use Monte Carlo (MC) method to estimate the integral.
#   Consider f(x) as the p.d.f for Cauchy(0, 1). Then g(x) / f(x) is an
#   indicator function 1(X > 2).
#     - Create an integer `n` as `1e4`.
#     - Set the seed to `0` using `set.seed()`.
#       Store the seed `.Random.seed` as a vector `seed` of length 626.
#     - Draw `n` i.i.d. samples from Cauchy(0, 1) using `rcauchy()`.
#       Store the result as a vector `x_cauc`.
#     - Define a function `g_over_f1_cauc` that
#       - Takes an input vector `x`.
#       - Returns a vector that maps `x[i]` to `TRUE` if `x[i] > 2` and
#         `FALSE` otherwise.
#     - Compute the MC estimator and its standard error using `g_over_f1_cauc`.
#       Store the results as a vector `est1_cauc` of the form c(estimate, se).
#   To check your result,
#     - `g_over_f1_cauc(x1_cauc)[1:5]` prints to:
#       [1] FALSE FALSE  TRUE FALSE FALSE
#     - `round(est1_cauc, 4)` prints to:
#       [1] 0.1500 0.0036
## Do not modify this line!
library(tidyverse)
p1 = 1 - pcauchy(2)
n = 1e4
set.seed(0)
seed = .Random.seed
x1_cauc = rcauchy(n)
g_over_f1_cauc = function(x){
 x > 2
}
est1_cauc = c(mean(g_over_f1_cauc(x1_cauc)), sd(g_over_f1_cauc(x1_cauc))/sqrt(n))
#上面这个mean公式就是slide上g_over_f的形式，se则很自然用后面的公式
# 2. To improve the precision of our estimator, we will use importance sampling.
#   Essentially, we want to decrease cases sampled out of `[2, inf)` to reduce
#   the variance of the estimator.
```

```
#   Consider X ~ Uniform(0, 2) with f(x) = 1 / 2.
#   Given that the Cauchy distribution is symmetric, we can write
#     - P(X > 2) = 1 - P(X < 2)
#                = 0.5 - the integral of g(x) over x from 0 to 2.
#   Then, with f(x) = 1 / 2 and g(x) / f(x) = 2 / (pi * (1 + x^2)):
#     - Set the seed to `0` using `set.seed()`.
#       Store the seed `.Random.seed` as a vector `seed` of length 626.
#     - Draw `n` i.i.d. samples from Uniform(0, 2) using `runif()`.
#       Store the result as a vector `x2_cauc`.
#     - Define a function `g_over_f2_cauc` that
#       - Takes an input vector `x`.
#       - Returns a vector that maps `x[i]` to `g(x[i]) / f(x[i])`.
#     - Compute the MC estimator and its standard error using `g_over_f2_cauc`.
#       Note that you should use 0.5 - mean(g(x) / f(x)) as we have rewritten
#       the integral.
#       Store the results as a vector `est2_cauc` of the form c(estimate, se).
#   To check your result,
#     - `g_over_f2_cauc(x2_cauc)[1:5]` prints to:
#       [1] 0.1509915 0.4965913 0.4096903 0.2752779 0.1480730
#     - `round(est2_cauc, 4)` prints to:
#       [1] 0.1473 0.0017
#   We can see that `est2_cauc` is closer to `p1` with a reduced variance.
## Do not modify this line!
set.seed(0)
seed = .Random.seed
x2_cauc = runif(n, 0, 2)
g_over_f2_cauc = function(x){#改变了g,思路仍相同
  2/(pi*(1+x^2))
}
est2_cauc = c(0.5 - mean(g_over_f2_cauc(x2_cauc)),sd(g_over_f2_cauc(x2_cauc))/sqrt(n))


# 3. Estimate the integral of `g(x) = exp(-x^2)` over x from 0 to 1.
#   First, let's compute the exact probability as a reference. We can show that
#   the integral = sqrt(pi) * P(0 < X < 1) when X ~ N(0, 1 / sqrt(2)).
#   Create a float `p2` to store the exact result using `pnorm()`.
#   Next, we will use MC to estimate the integral.
#   Consider X ~ Uniform(0, 1) with f(x) = 1. Then g(x) / f(x) = g(x).
#     - Create an integer `n` as `1e3`.
#     - Set the seed to `0` using `set.seed()`.
#       Store the seed `.Random.seed` as a vector `seed` of length 626.
#     - Draw `n` i.i.d. samples from Uniform(0, 1) using `runif()`.
#       Store the result as a vector `x_exp`.
#     - Define a function `g_over_f_exp` that
#       - Takes an input vector `x`.
#       - Returns a vector that maps `x[i]` to `g(x[i])`.
#     - Compute the MC estimator and its standard error using `g_over_f_exp`.
#       Store the results as a vector `est1_exp` of the form c(estimate, se).
#   To check your result,
#     - `g_over_f_exp(x_exp)[1:5]` prints to:
#       [1] 0.4475058 0.9319325 0.8706840 0.7202471 0.4383045
#     - `round(est1_exp, 4)` prints to:
```

```
#       [1] 0.7469 0.0064
## Do not modify this line!
p2 = sqrt(pi)*(pnorm(1, 0, 1/sqrt(2)) - 0.5)# 第一个arg是quantiles
n = 1e3
set.seed(0)
seed = .Random.seed
x_exp = runif(n, 0, 1)
g_over_f_exp = function(x){
  exp(-x^2)
}#就是g(x) 本身

est1_exp = c(mean(g_over_f_exp(x_exp)),sd(g_over_f_exp(x_exp))/sqrt(n))
# 4. To improve the precision of our estimator, we will first try antithetic
#    variates.
#    Consider X ~ Uniform(0, 1). Then X and 1 - X are antithetic variates.
#      - Compute the MC estimator and its standard error from `x_exp` and
#        `1 - x_exp` using `g_over_f_exp`.
#        Store the results as a vector `est2_exp` of the form c(estimate, se).
#      - Compute the correlation between g(x) and g(1 - x) using `cor()`.
#        Store the result as a float `cor1_exp`.
#    To check your result,
#      - `round(est2_exp, 4)` prints to:
#        [1] 0.7469 0.0009
#      - `round(cor1_exp, 4)` prints to
#        [1] -0.9595
#    We can see that the new estimator is more accurate with a reduced variance,
#    and the correlation between the antithetic variates is negative as expected.
## Do not modify this line!
est2_exp = c(mean((g_over_f_exp(1-x_exp)+g_over_f_exp(x_exp))/2), sd((g_over_f_exp(1-
x_exp)+g_over_f_exp(x_exp))/2)/sqrt(n))
cor1_exp = cor(g_over_f_exp(x_exp), g_over_f_exp(1-x_exp))
#两个公式都是slide上照抄
# 5. Next, we will try control variates to improve precision.
#    Consider the control variate h(x) = -x^2.
#      - Define a function `h` that
#        - Takes an input vector `x`.
#        - Returns a vector that maps `x[i]` to `h(x[i])`, where `h(x) = -x^2`.
#      - Create a float `c` by `-cov(g(x), h(x)) / var(h(x))`.
#      - Given that the integral of h(x) over x in [0, 1] is `- 1 / 3`, compute
#        the MC estimator and its standard error from `x_exp` using
#        `g_over_f_exp` and `h`.
#        Store the results as a vector `est3_exp` of the form c(estimate, se).
#      - Compute the correlation between g(x) and h(x) using `cor()`.
#        Store the result as a float `cor2_exp`.
#    To check your result,
#      - `h(x_exp)[1:5]` prints to:
#        [1] -0.80406587 -0.07049485 -0.13847620 -0.32816098 -0.82484139
#      - `round(est3_exp, 4)` prints to:
#        [1] 0.7473 0.0008
#      - `round(cor2_exp, 4)` prints to
#        [1] 0.9924
```

```
#   We can see that the new estimator is more accurate with a reduced variance,
#   and the correlation between `exp(-x^2)` and `-x^2` is large as expected.
## Do not modify this line!
h = function(x){
 -x^2
}
c = -cov(g_over_f_exp(x_exp), h(x_exp))/var(h(x_exp))
est3_exp = c(mean(g_over_f_exp(x_exp)+c*(h(x_exp)+1/3)),
sd(g_over_f_exp(x_exp)+c*(h(x_exp)+1/3))/sqrt(n))
cor2_exp = cor(g_over_f_exp(x_exp), h(x_exp))
```

#仍是按照公式照抄

```
# 6. Compute the area of A = {(x, y): x^2 + y^2 <= 1}.
#   In other words, estimate the double integral of g(x, y) = 1 over x and y,
#   when the integration domain is restricted to A.
#   We know that A is a circle centered at the origin with radius = 1, so its
#   area would be pi, but let's perform a MC integration for practice.
#   Consider two independent random variables X ~ Uniform(-1, 1) and
#   Y ~ Uniform(-1, 1), with joint p.d.f.:
#     - f(x, y) = 1 / 4, for -1 <= x <= 1 and -1 <= y <= 1.
#   Then g(x, y) / f(x, y) is 4 * 1(x^2 + y^2 <= 1), where 1(x^2 + y^2 <= 1)
#   is an indicator function.
#     - Create an integer `n` as `1e4`.
#     - Set the seed to `0` using `set.seed()`.
#       Store the seed `.Random.seed` as a vector `seed_x` of length 626.
#     - Draw `n` i.i.d. samples from Uniform(-1, 1) using `runif()`.
#       Store the result as a vector `x_circ`.
#     - Set the seed to `1` using `set.seed()`.
#       Store the seed `.Random.seed` as a vector `seed_y` of length 626.
#     - Draw `n` i.i.d. samples from Uniform(-1, 1) using `runif()`.
#       Store the result as a vector `y_circ`.
#     - Define a function `g_over_f_circ` that
#       - Takes input vectors `x` and `y`.
#       - Returns a vector that maps `(x[i], y[i])` to `4` if
#         `(x[i])^2 + (y[i])^2 <= 1` and `0` otherwise.
#     - Compute the MC estimator and its standard error using `g_over_f_circ`.
#       Store the results as a vector `est_circ` of the form c(estimate, se).
#   To check your result,
#     - `g_over_f_circ(x_circ, y_circ)[1:5]` prints to
#       [1] 4 4 4 4 0
#     - `round(est_circ, 4)` prints to:
#       [1] 3.1068 0.0167
## Do not modify this line!
n = 1e4
set.seed(0)
seed_x = .Random.seed
x_circ = runif(n, -1, 1)
set.seed(1)
seed_y = .Random.seed
y_circ = runif(n, -1, 1)
```

g_over_f_circ <- function(x, y) 4 * (x^2 + y^2 <= 1)#T = 1， F=0。下面的仍是公式运用

```
est_circ <- c(
  mean(g_over_f_circ(x_circ, y_circ)),
  sqrt(var(g_over_f_circ(x_circ, y_circ)) / n)
)


# 7. Our final step is to visualize the MC integration.
#    (1) Create a tibble `circ` of size 10000 x 3 with columns `x_circ`,
#        `y_circ` and `selected`, where `selected` is a boolean column indicating
#        whether the pair (x_circ, y_circ) is in the circle `x^2 + y^2 <= 1`.
#        To do this, you can use:
#          - `tibble()` to create a tibble from `x_circ` and `y_circ`.
#          - `mutate()` to add a column `selected`, by applying `g_over_f_circ`
#            to `x_circ` and `y_circ` and checking if the values equal 4.
#        To check your result, `circ` prints to:
#          x_circ y_circ selected
#          <dbl>  <dbl> <lgl>
#        1  0.793 -0.469 TRUE
#        2 -0.469 -0.256 TRUE
#        3 -0.256  0.146 TRUE
#        4  0.146  0.816 TRUE
#        5  0.816 -0.597 FALSE
#        # … with 9,995 more rows
#    (2) Draw a scatterplot for `y_circ` vs. `x_circ`. Color the scatterplot by
#        `selected`.
#        To do this, you can use:
#          - `ggplot()` to initialize a ggplot object and specify variables to plot.
#          - `geom_point()` to draw the scatterplot, setting `shape = 20`.
#          - `stat_function` to plot
#            - The upper circle `function(x) sqrt(1 - x^2)`, setting `size = 1.5`.
#            - The lower circle `function(x) -sqrt(1 - x^2)`, setting `size = 1.5`.
#          - `coord_fixed()` to fix the aspect.
#          - `labs()` to format the labels such that:
#            - `title = "Monte Carlo Integration of x^2 + y^2 <= 1"`
#            - `x = "x"`
#            - `y = "y"`
#        Store the plot as a ggplot object `g`.
## Do not modify this line!
circ = tibble(x_circ = x_circ, y_circ = y_circ) %>%
  mutate(selected = g_over_f_circ(x_circ, y_circ) == 4)
g = circ %>%
  ggplot(mapping = aes(x_circ, y_circ)) +
  geom_point(shape = 20, aes(color = selected)) +
  stat_function(fun = function(x) sqrt(1 - x^2), size = 1) +
  stat_function(fun = function(x) -sqrt(1 - x^2), size = 1) +
  coord_fixed(xlim = c(-1, 1), ylim = c(-1, 1)) +
  labs(title = "Monte Carlo Integration of x^2 + y^2 <= 1",
       x = "x", y = 'y') +
  theme_light()


# HW10: Importance sampling in Monte Carlo integration
```

```
#
# In this exercise, we will go over a simple application of monte carlo
# estimation of an integral. We will estimate the standard normal CDF
# between 1.5 and 2.5, namely `P(x > 1.5 & x < 2.5)` where `x ~ norm(0, 1)`.
# We can't derive the integral manually, thus we will use monte carlo estimation.
# The idea of Importance Sampling is to sample from a second distribution that
# will generate most of the samples in the desired interval, to have a fast
# convergence and find an appropriate acceptance rate to get the correct
# distribution approximation.
#
# 1. Create a function `in_set <- function(x, minx, maxx)` that takes as input
#    a numeric vector `x` and the limit `minx` and `maxx` and returns a vector
#    of `1` or `0` depending on whether the corresponding element of `x` falls
#    into the interval `[minx, maxx]` os not.
#    (It returns numeric `0` if not in this interval and `1` otherwise)
#    Example : `in_set(c(1, 2), 2, 2.5)` is `c(0, 1)`.
#    Create a function `norm_density <- function(x)` that takes as input
#    one scalar `x` and returns the standard normal density of that value.
#    Example : `norm_density(2) = 0.05399097`.
#    Create a function `g <- function(x)` that takes as input
#    one scalar `x` and returns `in_set(x, 1.5, 2.5) * norm_density(x)`.
#    Example : `g(2) = 0.05399097`.
## Do not modify this line!
in_set = function(x, minx, maxx){
  ifelse(x > minx & x < maxx, 1, 0)
}
norm_density = function(x){
  dnorm(x)
}
g = function(x){
  in_set(x, 1.5, 2.5) * norm_density(x)
}


# 2. Load the `tidyverse`, `tibble` and `ggplot2` packages.
#    Use `set.seed()` to set seed to `0` and save it into `seed1` using
#    `.Random.seed`. Use samples from standard normal to
#    estimate the cumulative distribution function between 1.5 and 2.5.
#    Create a tibble called `data` with three columns:
#      - `x` are 1000 samples generated randomly from standard normal using `rnorm`
#      - `y`, that is equal to 1 if `x` is in `[1.5, 2.5]` and 0 otherwise.
#        (you can use `in_set()`)
#      - `z` the density evaluated at point `x`. (you can use `norm_density`)
#      - inside `labs`, set `x` to "X", `y` to `"Density Function"` and `title`
#        to `"Standard Normal density"`, `subtitle` to
#        `"Integral interval shown in red"`.
#    `data` should print to:
#    # A tibble: 1,000 x 3
#            x     y     z
#        <dbl> <int>  <dbl>
#    1  1.26       0 0.180
#    2 -0.326      0 0.378
```

```
#   3  1.33       0 0.165
#   4  1.27       0 0.178
#   5  0.415      0 0.366
#   6 -1.54       0 0.122
#   7 -0.929      0 0.259
#   8 -0.295      0 0.382
#   9 -0.00577    0 0.399
#   10  2.40      1 0.0221
#   # … with 990 more rows
#   Estimate the integral value by calculating the mean of `data$y` and
#   save the result to `cdf_estimate`. Save the standard deviation of
#   `data$y` to `sd_estimate`.
#   (As we are using the same distribution as the function,
#   the integral is an approximation of `P(1.5 < x < 2.5))`
#   Calculate the true value by using difference of `pnorm()` at 2.5 and 1.5
#   and save it into `cdf_gold`. Compare our estimated `cdf_estimate` and
#   `cdf_gold`. We can notice that the variance of the estimate `sd_estimate`
#   is pretty high.
## Do not modify this line!
library(tidyverse)
library(tibble)
library(ggplot2)
set.seed(0)
seed1 = .Random.seed
data = tibble(x = rnorm(1000)) %>%
  mutate(y = in_set(x, 1.5, 2.5))%>%
  mutate(z = norm_density(x))
cdf_estimate = mean(data$y)
sd_estimate = sd(data$y)
cdf_gold = pnorm(2.5) - pnorm(1.5)

# 3. Plot the density function of standard normal to explore why this would happen.
#   To plot the figure :
#    - use `ggplot()` to initialize the ggplot object on `data`.
#    - use `geom_line()` with `mapping` set to `aes(x = x, y = z)` to draw
#      the full normal plot.
#    - use `geom_line()` with `data` `filter()` by `y == 1`, with `mapping`
#      set to `aes(x = x, y = z)` and `color` set to `red` to highlight the
#      area we want to take integral of.
#    - inside `labs()`, set `x` to `"X"`, `y` to `"Density Function"`,
#      `title` to `"Standard Normal density"` and `subtitle` to
#      `"Integral interval shown in red"`.
#    - use `theme_light()`.
#   Save the plot to `norm_plot`.
#   Then we can see the reason why the variance is high when sampling from
#   `norm(0, 1)` : we have relatively low probability to get samples
#   within `(1.5, 2.5)` range.
#   Thus, the value tend to vary a lot (and it has high standard deviation).
#   We should use another distribution which has a higher concentration over
#   the range `(1.5, 2.5)`.
## Do not modify this line!
norm_plot = data %>%
```

```
 ggplot() +
 geom_line(aes(x = x, y = z)) +
 geom_line(data = data %>% filter(y ==1), aes(x = x, y = z), color = 'red') +
 labs(x = "X", y = "Density Function",
     title = "Standard Normal density",
     subtitle = "Integral interval shown in red") +
 theme_light()


# 4. Now, we will explore this effect by using three different distributions
#    and use Importance Sampling to estimate the integral.
#    - set `n` to 1e4.
#    - Use `set.seed()` to set seed to `0` and save seed to `seed2` using
#    `.Random.seed`.
#    - Generate `n` samples from `uniform(1.5, 2.5)` and save it to `uniform`.
#    - Use `set.seed()` to set seed to `0` and save seed to `seed3` using
#    `.Random.seed`.
#    - Generate `n` samples from `normal(0, 1)` and save them into `original`.
#    - Generate tibble `fit` using `tribble()`, inside which:
#      - set three formula `~x` to represent our samples, `~name` to
#        record the distribution, `~g` to caculate the corresponding density
#        value.
#      - then add these rows by specifying the values:
#            `uniform`, `"uniform"`, `dunif(uniform, 1.5, 2.5)`,
#            `original`, `"original"`, `dnorm(original, 0, 1)`
#    - Use `mutate` to create column `g_over_f` using `map2()` and `g()` on
#      columns `x` and `f`.
#    Save the result into `fit`. It should print to :
#    # A tibble: 2 x 4
#    x            name     f            g_over_f
#    <list>       <chr>    <list>       <list>
#    1 <dbl [10,000]> uniform  <dbl [1]>     <dbl [10,000]>
#    2 <dbl [10,000]> original <dbl [10,000]> <dbl [10,000]>
3 <dbl [1,… uniform(1.… <dbl [1,… <int [1,… <dbl [1… <dbl [1… <dbl [1…
3 <dbl [1,000]> uniform(1.5, 2.… <dbl [1,000… <int [1,000… <dbl [1,000… <dbl [1,000… <dbl
[1,000…
## Do not modify this line!
n = 1e4
set.seed(0)
seed2 = .Random.seed
uniform = runif(n, 1.5, 2.5)
set.seed(0)
seed3 = .Random.seed
original = rnorm(n, 0, 1)
fit = tribble(~x, ~name, ~f,
           uniform, "uniform", dunif(uniform, 1.5, 2.5)[1],
           original, "original", dnorm(original, 0, 1))

fit = fit %>%
 mutate(g_over_f = map2(x, f, function(x,f) g(x)/f))
#注意map中fun要另外写
```

```
# 5. Calculate the expectation of `f(x)/g(x)` over distribution `g` by calculating
#    the mean of column `z` for all of our samples.
#    To do that, use `transmute()` to create new columns:
#      - use `mean()` and `map_dbl()` to generate
#        column `mean` recording the estimated integral under each distribution,
#        and `sd() / sqrt(n)` and `map_dbl()` to calculate column `se` which
#        is the variance of `f(x)/g(x)` under each set of samples.
#      - create column `upper` by setting it to `mean + 1.96 * se` and `lower`
#        by setting to `mean - 1.96 * se`.
#    Save the result tibble into `result` and it should print to:
#    # A tibble: 2 x 4
#        mean      se  lower  upper
#      <dbl>   <dbl>  <dbl>  <dbl>
#    1 0.0607 0.000326 0.0600 0.0613
#    2 0.0613 0.00240  0.0566 0.0660#'
## Do not modify this line!
result = fit %>%
  transmute(mean = map_dbl(g_over_f, mean), se = map_dbl(g_over_f, ~sd(.x)/sqrt(n)))
%>%
  mutate(lower = mean - 1.96*se, upper = mean + 1.96*se)


# 6. We will notice that for sample distribution from `uniform(1.5, 2.5)`,
#    we have smaller variance and closer estimate.
#    To explore the effect of different uniform intervals on estimation variance,
#    we will calculate the estimation for different uniform intervals centered
#    around 2.
#    - generate a sequence of possible uniform interval `width` using `seq()`
#      ranging from 0.1 to 3 with interval 1e-2.
#    - create a function `generate_sample <- function(w, seed = 0, n = 1e4)`,
#      in which `w` represents the uniform interval width, `seed` represents the
#      seed number, and `n` represents the sample size.
#      Inside the function:
#        - first set the seed to `seed` by `set.seed()`.
#        - return `n` samples using `runif()` and set `min` to `2 - w/2`, `max` to
#          `2 + w/2`. (The function will return the corresponding samples from the
#          uniform distribution specified by width `w`).
#    - create tibble `subsamples`, in which each row represents a different sample
#      size:
#      - use `tribble()`, set three formula `~width` to represent our sample size,
#        `~uniform` to record the uniform samples.
#      - then specify each row by `width` and its corresponding samples by `map`
#        and `generate_sample`.
#      - `unnest()` the tibble by `c(width, samples)`.
#      - use `mutate()` and `map2()` to calculate weighted sample values `g_over_f`
#        for each interval `width` and corresponding `samples` by customising
#      `function(width, sample) g(sample) / dunif(sample, 2 - width/2, 2 + with/2)`
#      - calculate the estimation result by `transmute()` to create new columns:
#        - use `mean()` and `map_dbl()` to generate column `mean` recording the
#          estimated integral under each distribution, and `sd() / sqrt(n)` and
#          `map_dbl()` to calculate column `se` which
#          is the variance of `f(x)/g(x)` under each set of samples.
```

```
#       - create column `upper` by setting it to `mean + 1.96 * se` and `lower`
#         by setting to `mean - 1.96 * se`.
#    Save the tibble to `result2`. it should print to:
#    # A tibble: 201 x 4
#       mean      se  lower  upper
#      <dbl>   <dbl>  <dbl>  <dbl>
#    1 0.0607 0.000326 0.0600 0.0613
#    2 0.0606 0.000333 0.0599 0.0612
#    3 0.0606 0.000341 0.0599 0.0613
#    4 0.0605 0.000348 0.0598 0.0612
#    5 0.0604 0.000354 0.0597 0.0611
#    6 0.0604 0.000361 0.0597 0.0611
#    7 0.0604 0.000368 0.0597 0.0611
#    8 0.0604 0.000374 0.0597 0.0611
#    9 0.0603 0.000380 0.0596 0.0611
#    10 0.0604 0.000386 0.0596 0.0612
#    # … with 191 more rows
## Do not modify this line!
width = seq(1, 3, 1e-2)
generate_sample = function(w, seed = 0, n = 1e4){
  set.seed(seed)
  sample = runif(n, min = 2-w/2, max = 2+w/2)
}
subsamples = tribble(~width, ~samples,
                     width, map(width, generate_sample)) %>%
  unnest(c(width, samples)) %>%
  mutate(g_over_f = map2(width, samples, ~g(.y)/dunif(.y, 2-.x/2, 2+.x/2)))

result2 = subsamples %>%
  transmute(mean = map_dbl(g_over_f, mean), se = map_dbl(g_over_f, ~sd(.x)/sqrt(n)))
%>%
  mutate(lower = mean - 1.96*se, upper = mean + 1.96*se)

# 7. Next, we can visualize the variance trend as the interval width changes.
#     - use `ggplot` to initialize the plot object over `result2`. Set `mapping`
#       to `aes(y = mean, x = width)`.
#     - use `geom_line` to plot variance curves.
#     - use `geom_ribbon()` to draw standard deviation shade to the plot,
#       set `mapping` to `aes(ymin = lower, ymax = upper)`, set `alpha`
#       to 0.2 and `fill` to `"orange"`.
#     - add gold line by `geom_line` with `mapping` set to
#       `aes(y = cdf_gold, x = width)` and `color` set to "red".
#     - use `scale_x_reverse()` to reverse the axis.
#     - inside `labs`, set `title` to `"MC estimate for different uniform interval"`,
#       set `subtitle` to `"Variance decreases with the interval width"`,
#       `x` to `"Interval Width"` and `y` to `"MC Estimate"`.
#      - use `theme_light()`.
#    Save the plot into `variance_plot`.
## Do not modify this line!
variance_plot = result2 %>%
  ggplot(mapping = aes(y = mean, x = width)) +
  geom_line() +
```

```
geom_ribbon(mapping = aes(ymin = lower, ymax = upper), alpha = 0.2, fill = 'orange') +
geom_line(mapping = aes(y = cdf_gold, x = width), color = 'red') +
scale_x_reverse() +
labs(title = "MC estimate for different uniform interval",
    subtitle = "Variance decreases with the interval width",
    x = "Interval Width", y = "MC Estimate") +
theme_light()
```