

LAB 1: R 语言的基本操作和数据预处理

R 是什么？

R 是一种通常用于统计学、数据科学和可视化的编程语言。它也可视作是一个运行环境，汇集了许多特性，提供强大的功能。R 的优点：

- 能有效处理大数据，并进行数据处理、整理和存储等操作。
- 集成工具的合集，提供多种统计方法和图形技术。
- 简单易学的编程语言，易于在不同平台上安装和使用。
- 免费开源，拥有不断增长的用户社区。

安装 R: <https://mirrors.tuna.tsinghua.edu.cn/CRAN/>

RStudio 是什么？

RStudio 是一款免费的开源集成开发环境（IDE），使 R 的编写和使用更加容易。

RStudio 的优点：

- 图形用户界面，易学易管理。
- 对学术社区开源，可以免费使用。
- 支持多种操作系统和平台。

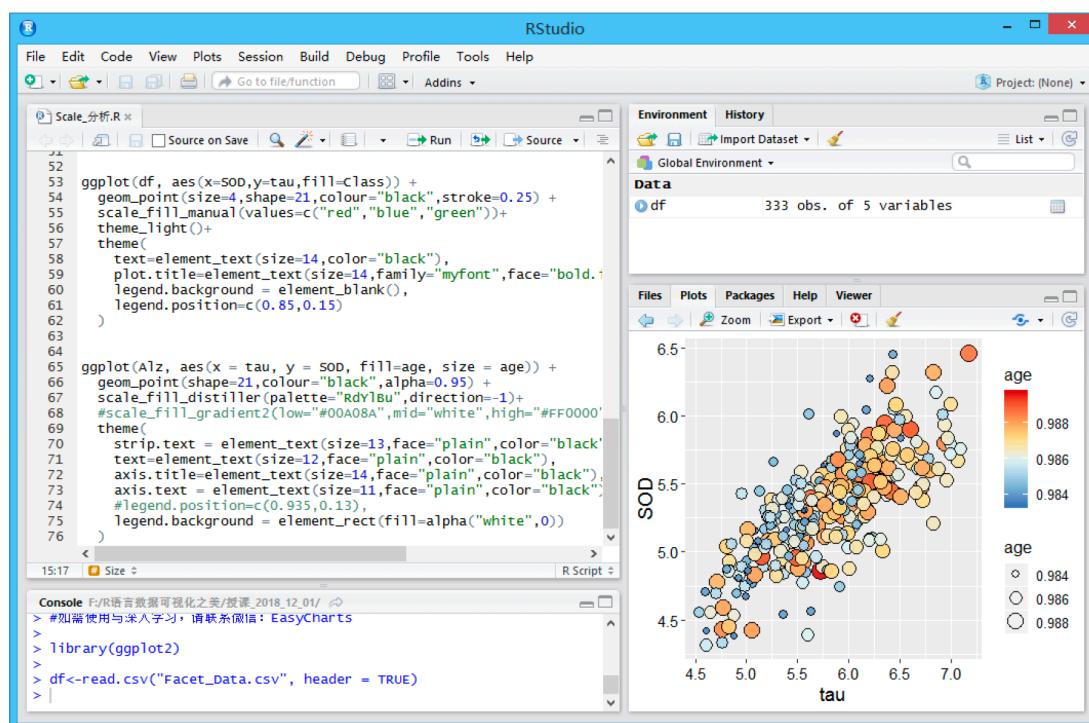
安装 RStudio: <https://posit.co/download/rstudio-desktop/>

安装时，建议把默认语言设置为英文。

RStudio 的界面

RStudio 的通用界面如下， 默认分为四部分：

- 左下是控制台：可以在其中键入命令并查看输出。
- 左上是脚本编辑器：在其中键入代码命令并保存到文件。
- 右上是环境/历史记录：环境显示所有活动对象，历史记录跟踪控制台中运行的所有命令，也显示引入和激活的数据集和方程。
- 右下是文件/绘图/包/帮助等辅助模块。



安装包（library）

在 R 里，许多功能可以调用现有的包来实现。其中，经过社区审核的安装包被存放在 CRAN 网站，对于这些安装包，最简单的方法是单击 RStudio 界面右下角写有“Packages”的选项卡，然后在弹出的对话框中输入包的名称。或者直接在左下角的控制台输入安装命令。例如要安装 `ggplot2`，就输入：

```
install.packages("ggplot2")。
```

此外，还有诸多没有经过 CRAN 社区审核的安装包，它们一般被存放在 GitHub 等网站，其对应的 Repository（代码仓库）里通常有基于 devtools 的安装指导。需要注意的是，这些没有经过 CRAN 审核的安装包质量可能会参差不齐，需谨慎使用。

任务：安装以下常用包

```
ggplot2, readr, astrochron, colorRamps, dplyr, ggrepel, gtable, gridExtra,  
cowplot, tidyverse, stringr, devtools
```

包安装好之后，需要加载才能使用，主要有两种函数可供选择：library()或者 require()。比如：

```
library(ggplot2)。
```

创建 R 脚本文件

创建一个工作文件夹，并在工作文件夹内再创建一个命名为 data 的文件夹，用于储存数据（将本节课的 csv 文件放入这个 data 文件夹里）。

在 RStudio 的菜单栏中，点击 File→New File→R Script，会得到一个新的脚本文件。这时，点击 File→Save，找到你的工作文件夹，将脚本存储为后缀为.R 的文件。

在菜单栏中，点击 Session→Set Working Directory→To Source File Location，可以把工作环境指向脚本文件所在的工作文件夹，有利于之后导入和导出数据以及储存图片。

R 语言的基本操作

Hello World

```
myString <- "Hello, World!"  
print ( myString)
```

运行某一行代码，可将光标置于相应的行，然后点击快捷键：

- Windows 系统: Ctrl 加回车。
- MacOS 系统: cmd 加回车。

打开说明文档: ?+名称, 比如运行 ?ggplot2 语句, 会打开 ggplot2 包的说明文档。

数据类型

与 C++ 和 Java 等其他编程语言不同, 在 R 语言中, 变量不声明为某种数据类型。变量是用 R 对象分配的, R 对象的数据类型就是变量的数据类型。R 对象有多种类型。常用的 R 对象有以下几个:

Vectors、Lists、Matrices、Arrays、Factors 和 Data Frames

这些 R 对象包含不同类别的元素, 包括:

逻辑类元素 (True 和 False)、数值类元素 (包括整数和浮点类型)、整数类元素 (数字后面加 L 表示)、字符类元素等。

向量:

```
# Create a vector.  
animal <- c('dog','cat',"bird")  
print(animal)  
  
# Get the class of the vector.  
print(class(animal))
```

矩阵:

```
# Create a matrix.  
M = matrix( c(7,5,5,2,3,7), nrow = 2, ncol = 3, byrow = TRUE)  
print(M)
```

数据框:

```
# Create the data frame.  
students <- data.frame(  
  name = c("Tommy", "Richard", "Anne"),  
  gender = c("Male", "Male", "Female"),  
  height = c(175, 179, 165),  
  age = c(21, 22, 20),  
  major = c("statistics", "geology", "physics"))  
print(students)
```

数据框的基本操作：

```
#获得数据框中的一列
names = students$name
print(names)

#数据框的行数和列数
print(nrow(students)) #行数
print(ncol(students)) #列数

#把第五列改名为subject
colnames(students)[5] ="subject"
#把名为age的一列改名为years_old
colnames(students)[colnames(students) == "age"] ="years_old"
#一次性把以上两列的名称改回去
library("dplyr")
students <- students %>%
  rename("age"="years_old",
        "major" = "subject")
```

数据框的清理。运行下列代码，观察其作用：

```
df <- data.frame(values = c("A", "A", "B", "C", "C", "D", "E", "F"),
                  data = c(1, 3, 4, NA, 5, 2, 3, NA))
df2 <- df[which(!duplicated(df$values)),]
print(df2)
df3 <- df[which(!is.na(df$data)),]
print(df3)
```

数据框的合并(merge)。运行下列代码，观察其作用：

```
df1 = data.frame(name = c("A", "B", "C", "D", "E", "F"),
                  data1 = c(1, 2, 3, 4, 5, 6))
df2 = data.frame(name = c("B", "C", "D", "E", "G", "H"),
                  data2 = c(2, 3, 4, 5, 6, 7))
df3 = merge(x = df1, y = df2, by = "name", all.x = TRUE, all.y = TRUE)
print(df3)
df4 = merge(x = df1, y = df2, by = "name", all.x = TRUE, all.y = FALSE)
print(df4)
df5 = merge(x = df1, y = df2, by = "name", all.x = FALSE, all.y = TRUE)
print(df5)
df6 = merge(x = df1, y = df2, by = "name", all.x = FALSE, all.y = FALSE)
print(df6)
```

R 语言的数学运算符及常用函数

输出以下代码中的 a、b、c、d、e、f、g、h，并观察运算符的写法。需要注意的是，目前版本的 R 语言中，大多数情况下，等号 (=) 和传统的 R 赋值符号 (<-) 已经可以通用。

```
a <- 1 + 1
b <- 7 - 9 * 6
c <- 3 * (5 + 7) / 2
d <- 2 ** 3
e <- 2 ^ 3
f <- sqrt(4) - 4 ^ (0.5)
g <- 103 %% 10
h <- 51 %% 5
```

输出以下代码的结果，并思考计算过程：

```
a <- c(1,2,3)
b <- c(2,3,4.5)
print(a + b)

c <- c(1,2,3)
d <- c(2,3,4.5)
print(c %% d)

e <- c(1,2,3)
f <- c(2,3,4.5)
print(e ^ f)
```

输出以下代码的结果，并观察代码的功能：

```
print(seq(1,10,0.5))
print(100)
print(toString(100))
print(abs(-1))
print(ceiling(1.3))
print(floor(1.3))
print(max(1,2,3))
print(min(1,2,3))

str <- "Hello World!"
grepl("H", str)
grepl("Hello", str)
grepl("X", str)

str1 <- "Hello"
str2 <- "World"
paste(str1, str2)
paste(str1, str2, sep="%")
```

输出以下代码的结果，并观察代码的功能：

```
a = NA
print(is.na(a))

a = "Hello World!"
print(is.character(a))
print(is.numeric(a))

a = 100
print(is.numeric(a))
print(is.integer(a))

a = 100L
print(is.numeric(a))
print(is.integer(a))

a = FALSE
print(is.logical(a))
print(is.character(a))
```

条件语句

运行以下代码并观察输出结果：

代码 1

```
a = 5
if(a <= 4){
  print("dog")
} else if (a <= 5){
  print("cat")
} else {
  print("bird")
}
```

代码 2

```
x <- 5L
if(is.integer(x)) {
  print("X is an Integer")
} else {
  print("X is NOT an Integer")
}
```

代码 3

```
animal <- c("dog", "cat", "bird")  
  
if("dog" %in% animal) {  
  print("dog is found")  
} else {  
  print("dog is not found")  
}  
  
if("pig" %in% animal) {  
  print("pig is found")  
} else {  
  print("pig is not found")  
}
```

循环语句

运行以下 `for` 循环代码并观察输出结果，思考 `print` 和 `cat` 命令的区别

代码 1

```
animal <- c("dog", "cat", "bird")  
  
for (a in animal){  
  print(a)  
}  
  
for (a in animal){  
  cat(a)  
}  
  
for (a in animal){  
  cat("The animal is", a, "\n")  
}
```

代码 2

```
cities <- c("Beijing", "Chongqing", "Lanzhou")  
  
for(i in 1:length(cities)){  
  cat("The city is ", cities[i], "!! \n", sep = "")  
}
```

运行以下 `while` 循环代码并观察输出结果

```
max = 5
count = 0

while(count < max){
  cat("Count:", count, "\n")
  count = count + 1
}
```

自定义函数

如下所示，定义一个求和的函数。

```
sum <- function(num1, num2){
  return(num1 + num2)
}

a = sum(3,5)
print(a)
```

基本的统计命令

运行以下代码并观察结果：

```
data <- c(1,5,2,6,2,7,7,7,7,3,4,8,3,2,10,15,2,4,7,3,3,3,7,2,8,2,7)
max(data) #最大值
min(data) #最小值
mean(data) #平均值
median(data) #中位数
sd(data) #标准差
quantile(data, c(0.25,0.75,0.9)) #百分位数

#R没有预设的寻找众数的函数，可以用以下两种方法找众数
#方法一：把数字转化为字符串后，按相同字符串的数量排序
#然后返回排名第一的字符串，再转换回数字
as.numeric(names(sort(-table(data)))[1])
#方法二：自定义函数
mode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
mode(data)
```

快速输出常见基本统计量：

```
#示例数据集
data <- data.frame(
  values = c(1,3,6,6,2,7,2,1,7,4,7)
)

#基本统计量
summary(data$values)
```

统计出现次数：

```
#示例数据集
data <- data.frame(
  animals = c("dog", "dog", "cat", "pig", "dog", "cat", "cat", "bird", "bird", "cat")
)

#统计每种动物的出现次数
table(data)

#可以直接汇总为一个数据框
animal_freq <- data.frame(table(data))
```

三角函数

运行以下代码并观察结果：

```
x = 30
sin(x)
cos(x)
tan(x)
```

根据结果可知，R 中的三角函数功能默认的输入值不是角度，而是弧度。因此在使用三角函数时，需先

把角度转换为弧度，公式为：

$$\text{弧度} = (\text{角度} * \pi) / 180$$

运行以下代码并观察结果：

```
deg2rad <- function(deg) {
  return(deg * pi / 180)
}
x = 30
sin(deg2rad(x))
cos(deg2rad(x))
tan(deg2rad(x))
```

同样，R 中的反三角函数输出值为弧度，如果要得到角度，需进行转换。

运行以下代码并观察结果：

```
rad2deg <- function(rad){
  return(rad * 180 / pi)
}
x = 0.5
rad2deg(asin(x))
x = 0.8660254
rad2degacos(x))
x = 0.5235988
rad2degatan(x))
```

导入和导出 CSV 数据

最简单的方法是调用 `readr` 包中的 `read_csv` 功能，从文件夹中读取 CSV 数据

```
library(readr)
example1 <- read_csv("data/lab1/example1.csv")
```

在右上方的环境栏里点击 `example1` 数据集并观察数据。该数据集有几行、几列？

最后一列中的 `good` 和 `bad` 分别有多少个？

用代码实现：

```
print(nrow(example1)) #输出行数
print(ncol(example1)) #输出列数
table(example1$category) #统计最后一列的类别
summary(example1$x) #对“x”这一列进行简单统计
```

新增一列“`z`”并使其等于 `x` 和 `y` 的和。

```
example1$z = example1$x + example1$y
print(example1$z)
```

其他的常用操作：

```
#找到x和y相等的行
example1[example1$x == example1$y,]
#返回所有x > 20的行数
which(example1$x > 20)
#找到category为bad的子集，并将其定义为新的数据框bad
bad <- example1[which(example1$category == "bad"),]
#找到x > 20且category为good的行，并将其定义为新的数据框df
df <- example1[which(example1$x > 20 & example1$category == "good"),]
```

导出数据框到磁盘并储存为 CSV 文件：

```
#把上一步生成的数据库df存到工作文件夹里
write.csv(df, "data/lab1/good_data.csv")
```

数据预处理

任务：导入 example2.csv 并命名为 gradebook

检查：gradebook 数据框中的数据有哪些问题？

	first_name	last_name	age	major	gender	midterm1	midterm2	midterm3	final
1	Frank	Smith	20	computer science	male	85	83	79	80
2	Dan	Williams	19	statistics	male	82	80	77	82
3	Arthur	Philips	21	statistics	male	83	88	74	90
4	Tony	Taylor	~19	statistics	Male	77	90	73	79
5	Allan	Holmes	19	computer science	male	NA	91	89	86
6	Robert	Peters	20	computer science	male	69	NA	NA	68
7	Emma	Cole	"20"	geography	female	83	85	90	82
8	Maria	Keller	18	geography	Female	90	83	92	84
9	Ann	Susanna	21	computer science and geography	female	88	70	77	71
10	Rayford	Clark	20	statistics	male	NA	70	73	75
11	Urban	Meyer	22	statistics and geography	male	65	66	69	71
12	Les	Miles	19 years old	computer science	Male	82	80	80	80
13	Bill	Lohmann	18	statistics	male	70	69	81	81
14	Annie	Bowen	20	statistics	female	85	72	69	66
15	Helen	Dorothy	20	computer science	female	90	86	83	71
16	Jim	Hope	21	geography	Male	91	88	90	85

问题 1：年龄（age）一列的数字中出现了非数字的符号，导致该列的类别是字符串，无法进行平均值等计量操作。

解决方法

```
library(tidyverse)
#利用mutate变动函数，操作“age”一列中的所有数据
#利用readr::parse_number，来drop掉所有非数值的字符
gradebook <- gradebook %>% mutate(across(c("age"), readr::parse_number))
```

R 语言中的管道操作符 %>%: <https://www.jianshu.com/p/c65dbce983dd> (本课不要求但很有用的方法，可作为扩展内容自主学习)

问题 2：性别（gender）一列中首字母大小写不统一，不利于做分类。

解决方法

```

library(stringr)
#利用stringr包中的函数进行大小写格式转换
gradebook$gender = str_to_title(gradebook$gender)

#尝试运行以下代码并观察stringr中不同函数的功能
string = "this is a Test sentence."
print(str_to_title(string))
print(str_to_sentence(string))
print(str_to_upper(string))
print(str_to_lower(string))

```

问题 3：三次期中考试 (midterm) 都有缺失值。如果缺失值代表缺考，记 0 分，则需要把缺失处填上“0”。

解决方法：

```

gradebook$midterm1[is.na(gradebook$midterm1)] <- 0
gradebook$midterm2[is.na(gradebook$midterm2)] <- 0
gradebook$midterm3[is.na(gradebook$midterm3)] <- 0

```

接下来，我们要统计每个专业 (major) 的学生人数。我们可以发现，有的学生为双专业（比如 computer science and geography），因此简单的 table 函数无法满足需求。

解决方法 1：

```

#将major一列提取出来形成一个array
major <- gradebook$major
#将每个含有“and”的要素按照“and”所在的位置分开，并合成新的array
major_split <- unlist(strsplit(major, split = " and ", fixed=TRUE))
table(major_split)

```

解决方法 2：

```

library(tidyverse)
#或者利用tidyverse包里的separate_rows函数直接修改数据框
gradebook %>% separate_rows(major, sep = " and ")

```

现在让我们来计算每个人的最终成绩。

计算方法：3 次期中考试去掉分最低的一次后，算平均分为平时成绩，占 70%。期末考试占 30%。两者相加为最终成绩。

解决方法：

```
library(tidyverse)
#计算去掉一次最低分后的平时成绩得分
gradebook <- gradebook %>%
  rowwise() %>%
  mutate(mean_of_top_two_midterm = mean(c(midterm1, midterm2, midterm3))[order(-c(midterm1, midterm2, midterm3))[1:2]])
#计算最终成绩
gradebook$final_grade = gradebook$mean_of_top_two_midterm * 0.7 + gradebook$final * 0.3
```

我们想要统计期末考试时的成绩分布，从 60 分往上，每 5 分一段的人数。

解决方法：

```
#建立一个新的数据框package
package = data.frame(grade = seq(55,100,5))
#新增一列来接收期末考试人数分布的数据
package$final_dist = 0
#用一个for循环来交叉package的grade一列和gradebook中的final一列
for(t in 1:nrow(package)){
  package$final_dist[t] =
    length(gradebook$final[intersect(which(gradebook$final<=package$grade[t]),
                                     which(gradebook$final>package$grade[t-1]))])
  cat("proc",t,"\n")
}
```

如果要统计每个分数段的总翘课次数（skipping），也可以用类似的方法。

解决方法：

```
#建立一个新的数据框package
package = data.frame(grade = seq(55,100,5))
#新增一列来接收翘课总次数分布的数据
package$skipping_count = 0
#用一个for循环来交叉package的skipping_count一列和gradebook中的skipping一列
for(t in 1:nrow(package)){
  package$skipping_count[t] =
    sum(gradebook$skipping[intersect(which(gradebook$final<=package$grade[t]),
                                     which(gradebook$final>package$grade[t-1]))])
  cat("proc",t,"\n")
}
```

现在我们需要新增一列 output，来储存每个人的学期报告。报告为一句话，内容包括名、姓和期末最终成绩。例如“Frank Smith's score is 82.8.”。

解决方法：

```
#把final_grade一列转换成字符串  
gradebook$final_grade <- lapply(gradebook$final_grade, as.character)  
#将输出报告写进新增的output列里  
gradebook$output <- paste(gradebook$first_name, " ", gradebook$last_name,  
    "'s score is ", gradebook$final_grade,".", sep = "")
```

作业部分

数据

本次作业内容是对古生物（恐龙）的化石数据集进行基本处理。

导入 `china_dino.csv` 数据集，该数据集包含了中国大陆境内发现的恐龙化石，关键字段如下：

- `occurrence_id`: 每一枚化石的唯一识别 (unique id)
- `collection_id`: 每一枚化石所属的化石集 (一般为收藏化石的博物馆或发现化石的化石点)
- `name`: 化石对应物种的名称
- `rank`: 化石分类的精度，包括目 `order`、科 `family`、亚科 `subfamily`、属 `genus`、种 `species` 等级别。
- `max_ma` 和 `min_ma`: 化石对应物种的生存时间，其中 `max_ma` 为最早时间，`min_ma` 为最晚时间，单位是百万年前 (Ma)
- `lon` 和 `lat`: 化石出土地点在现代的经纬度
- `province`: 化石出土地点所在的省级行政区
- `paleo_lon` 和 `paleo_lat`: 在考虑板块活动 (大陆漂移) 后，生物死亡时所在地的经纬度
- `formation`: 化石出土的地层名称
- `quality`: 化石的保存状况
- `collection_method`: 化石的采集方法
- `diet`: 化石对应物种的食物种类，包括肉食 `carnivore`、素食 `herbivore`、杂食 `omnivore`
- `time_bins`: 化石对应的地质年代 (期级)

问题

利用 R 语言分析该数据集，并回答以下问题：

1. 该数据集中，一共有多少枚化石？它们分属于多少个化石集（collection）？平均每个化石集含有多少枚化石？含有化石数量最多的化石集共有几枚化石？
2. 在该数据集中，分类精度为“属”和“种”级的化石一共有多少枚？它们分属于多少个化石集？
3. 出土恐龙化石数量前 10 的省级行政区分别是哪些？它们各自出土了多少枚化石？在已知的情况下，哪个地层出土的恐龙化石数量最多？
4. 该数据集中，恐龙化石的保存状况分为几个等级（不包括未知）？它们各自对应了多少枚化石？哪个省级行政区产出了最多的“excellent”级化石？
5. 哪些省级行政区出土的恐龙化石里，肉食恐龙数量多于素食恐龙？
6. 数据集中包含的化石采集方法共有多少种？每一种分别对应了多少枚化石？注意：有些化石的采集过程使用了不止一种方法。

已知：恐龙生活的中生代分为三迭纪（Triassic）、侏罗纪（Jurassic）和白垩纪（Cretaceous）。

其中三迭纪的起止年代为 251~201.3 (Ma)，侏罗纪的起止年代为 201.3~145 (Ma)，白垩纪的起止年代为 145~66 (Ma)。以上的起止年代包含较新边界，不包含较旧边界（比如白垩纪包含 66Ma，不包含 145Ma）。

7. 中国大陆发现的恐龙化石中，属于三迭纪、侏罗纪和白垩纪的分别有多少个？注意：如果一个化石跨越了不同的纪，那么它应被分别计算在不同纪的统计里。
8. 数据集中的 time_bins 一共出现了多少个“期”（不包括未知“-”）？它们分别属于哪一个纪？（我们默认“期”不会跨越不同的“纪”）

计算地球表面两点间的距离需要用到球面几何知识。

可用 Haversine 公式计算两点间的球面距离。假设两点坐标分别为 $(\text{lat1}, \text{lon1})$ 和 $(\text{lat2}, \text{lon2})$,

则距离 d 的计算公式为：

```
 $\Delta\text{lat} = \text{lat2} - \text{lat1}$ 
 $\Delta\text{lon} = \text{lon2} - \text{lon1}$ 
 $a = \sin^2(\Delta\text{lat}/2) + \cos(\text{lat1}) * \cos(\text{lat2}) * \sin^2(\Delta\text{lon}/2)$ 
 $d = 2 * \arcsin(a^{1/2}) * R$ 
```

其中 R 为地球半径，取 6371 千米。注意：经纬度坐标本质上是角度，需要转换为弧度。

9. 在 R 中编写一个函数 `haversine_dist()`，依次输入两点的经纬度坐标，返回两点间的距离。该

功能可用 `geosphere` 包中的 `distVincentyEllipsoid` 函数轻松实现，但本次作业请勿使用。

10. 在恐龙化石的数据集里，所有化石的经纬度都因板块活动而发生了改变，请问它们的平均移动距离

为多少千米？

提交内容

1. 在 学习通 提交以上 10 个问题的答案。每个问题满分 10 分，共计 100 分。

2. 将所用代码汇总在一个 R 文档里，并写好注释，作为问题 1 的附件上传到学习通。R 文档命名为“Lab1+姓名+学号”，比如 “Lab1 张三 1010101010.R”。

截止：2023 年 11 月 27 日北京时间 23:59。R 代码的参考写法将在 2023 年 11 月 28 日发布。