데이터분석기초(R)

Lee Jun Hyeok (wnsx0000@gmail.com)

August 18, 2024

목차

Part I

데이터분석기초(R)

1. 서론

1.1. 서론

1.1.1. 4차 산업혁명과 데이터

정량화될 수 있는 수치들을 데이터라고 함. 데이터는 돈이 되기에 중요함.

인공지능(Artificial Intelligence, AI), 빅데이터(big data), 로봇(robot), 사물 인터넷(IoT, Internet of Things), 생명공학기술(Biotechnology), 3D 프린터(3D printer) 등의 새로운 과학기술이 사회, 경제, 문화 전반에 영향을 미치게 되고, 이러한 변화를 잘 수용하고 가능성을 최대화하는 시대를 4차 산업혁명 시대라고 함.

인공지능과 빅데이터가 4차 산업혁명의 핵심 기술로서 화두에 오름. 빅데이터에서 데이터를 가져와 인공지능을 학습시킨다는 측면에서도 빅데이터는 가치가 있음.

1.1.2. 빅데이터

Definition 1 *빅데이터는 3V로 설명됨.*

큰 크기(Volume)와 다양성(Variety), 빠른 속도(Velocity)를 가지는 데이터를 빅데이터라고 함.

빅데이터는 정형, 반정형, 비정형 데이터로 나뉘는 다양성을 가짐. 정형은 일정한 형식의 데이터(ex. 엑셀)를, 반정형은 일정한 구조는 아니지만 파악이 가능한 구조를 가지는 데이터(ex. HTML)를, 비정형은 정해진 구조가 없는 데이터(ex. 사진)를 의미함.

빅데이터는 쌓이는 속도와 유통되어 활용되는 속도 모두가 빠름.

예를 들면, 의료 분야의 환자 데이터, 대중교통 이용 데이터 등이 있음.

1.1.3. 데이터 분석 과정

데이터 분석 과정은 아래와 같음.

데이터 분석 시 시간을 가장 많이 잡아먹는 것은 정제 및 전처리 단계임.
데이터 탐색 단계에서는 대푯값, 그래프 등으로 전반적인 내용을 파악함.
데이터 분석 단계에서는 고급 분석 기법들과 머신러닝 등이 수행됨.
결과 보고 단계에서는 주로 데이터를 그래프 등으로 시각화함.

전체 데이터 분석 시간에서 60%정도를 데이터 정제 및 전처리에, 19%정도를 데이터 수집에 사용함.

1.1.4. 버전 읽는 법

대체로 버전은 1.2.3꼴로 작성하는데, 각 영역은 major, minor, fetch임.

버그를 발견하고 해결한 경우 해결한 버전을 fetch, 사소한 변화로 인해 디테일을 수정한 버전을 minor, 핵심 요소의 변화 등 큰 변화로 인해 수정한 버전을 major로 지정함.

1.2. R

1.2.1. R

Definition 2 *데이터분석용 인터프리터형 프로그래밍 언어.*

R의 소스 파일의 확장자는 .R임.

1970년대 후반에 벨 연구소에서 개발된 통계분석용 언어 S를 기반으로 로스 이하카, 로버트 젠틀맨 등이 만든 언어가 R임. 1993년 R에 처음 공개되었고, 이후 GNU 소프트웨어로 배포되어 무료로 사용이 가능함.

인터프리터 방식으로, 프로그래밍이 간단하고 확장성이 좋아 통계분석에서 자주 사용됨.

R에는 아주 다양한 라이브러리들이 존재함. R을 잘 다룬다는 것은 다양한 라이브러리와 함수를 알고 사용할 수 있다는 것을 말함.

1.2.2. R studio

R의 가장 대표적인 IDE.

실행하면 기본적으로 화면이 편집 창, 콘솔 창, 환경 창, 파일 창으로 구성되어 있음.

환경 창에서는 변수, 자료구조 등을 확인할 수 있음.

파일 창의 plots에서는 그래프를, packages에서는 패키지를, help에서는 도움말을, viewer에서는 웹브라우저 결과를 확인할 수 있음. packages에서 패키지 수동 설치 가능.

코드를 실행할 때는 화살표 버튼을 누르거나(현재 커서 위치부터 실행됨.) 아래의 단축키를 사용할 수 있음.

Ctrl + Enter : 현재 커서 위치의 한 줄 실행.
코드 드래그 후 Ctrl + Enter : 해당 코드 실행.
Ctrl + Alt + r : 모든 코드 실행.
Ctrl + Shift + p : 직전에 실행한 코드를 다시 실행.

한글이 깨지는 경우 Tools -> Global Options -> Code -> Saving 에서 인코딩 방식을 지정해 해결 가능.

참고로, 파일 경로 자체에 한글이 들어가면 언젠가 오류가 생길 수 있음. 특히 계정명이 한글이면 경로가 Users/이준혁/.... 일 수 있음.

2. R 기본 문법

2.0.1. 문법 형태

R의 문법은 파이썬 문법과 유사한 점들이 있음.

변수 사용 시에 자료형을 명시하지 않고, ;를 사용하지 않음.

2.0.2. 출력

또한 변수나 어떤 데이터를 출력할 때 print() 함수에 넣어서 출력할 수도 있지만, 줄에 식별자 하나만 작성해도 출력이 됨.

데이터 출력 시에 [1] 등이 출력 결과 앞에 붙어 있는데, [] 안 숫자로 몇 번째 정보인지를 알려주는 것임.

cat() 함수를 사용하면 cat('2 *', i,'=', 2*i,'\n')과 같이 여러 문자형 데이터와 값을 쉼표로 구분해서 출력할 수 있음. 또한 print()에서는 '\n'가 개행으로 반영되지 않고 그대로 출력되는데, cat()에서는 개행으로 반영됨.

2.1. 기초 문법

2.1.1. 주석

R에서 주석은 ♯로(한 줄 주석) 표기함.

2.1.2. 함수 인자의 작성

R에서는 함수의 매개변수에 default값을 지정해 놓는 경우가 있는데, 이 경우 인자로 값을 전달하지 않아도 정상적으로 작동함.

인자에 sort(d, decreasing=TRUE)등과 같이 어떤 이름 = 값 꼴로 작성할 때가 있는데, 이는 인자의 값을 알아보기 편하게 하기 위함으로, 생략하고 값만 입력해도 정상 작동하는 경우가 많음.

사용하려는 함수가 어떤 매개변수 목록을 가지고 있고, 어떤 기능을 하는지는 파일 창의 help에서 함수를 검색하여 확인할 수 있음. 다만 최신 함수의 경우 반영이 되어있지 않을 수 있음.

2.1.3. 인덱스 값

R에서는 벡터, 매트릭스 등의 인덱스가 0이 아닌 1부터 시작함.

2.2. 기본 연산

2.2.1. 연산자

1. 산술 연산자
+, -, *, /, %%(나머지), ∧(거듭제곱)

ex. ∧는 5 ∧ 7이면 5의 7제곱인 것.

2. 논리 연산자
<, <=, >, >=, ==, !=, |, &

c에서 ||, &&로 작성하는 것을 R에서는 |와 &로 작성함.

2.2.2. 연산 함수

R의 기본 연산 함수들은 아래와 같음.

함수	의미	사용 예
log( )	로그함수	log(10), log(10, base=2)
sqrt( )	제곱근	sqrt(36)
max( )	가장 큰 값	max(3,9,5)
min( )	가장 작은 값	min(3,9,5)
abs( )	절대값	abs(-10)
factorial( )	팩토리얼	factorial(5)
sin( ), cos( ), tan( )	삼각함수	sin(pi/2)

2.3. 변수

2.3.1. 변수

Definition 3 *R에서는 파이썬에서처럼 변수 사용 시에 자료형을 명시하지 않아도 자동 지정됨. 단순히 필요할 때 대입해서 사용하면 됨. 대입은 아래와 같이 <-로 함.*

```
a <- 10
```

=로도 대입이 가능은 하지만 권장되지 않음.

2.3.2. 식별자 조건

1. 첫 글자는 영문자 또는 마침표(.)로 시작. (대체로 영문자 사용)
2. 두 번째 글자부터 영문자, 숫자, 마침표(.), 밑줄(_) 사용 가능.
3. 대문자와 소문자 구분.
4. 중간에 공백문자 사용 불가.

2.3.3. 자료형

자료형으로는 숫자형, 문자형, 논리형이 있고, 특수값들도 존재함.

R에서는 기본적으로 자료형이 다른 두 데이터의 산술 연산은 불가능한데, 연산을 하면 가능한 경우 자동 형변환이 일어남. 이때 순위는 논리형<숫자형<문자형임. 문자형과의 연산에서 논리형은 "TRUE"등으로, 숫자형은 "123" 등으로 형변환됨. 숫자형과의 연산에서 논리형은 0과 1로 형변환됨.

1. 숫자형 : 정수, 실수를 저장.

as.numeric("123") : 지정한 값을 숫자형으로 변환하여 반환.

2. 문자형 : 문자열을 저장.
문자열은 작은따옴표 또는 큰따옴표로 묶어서 표기. " ' 이런 식으로는 사용할 수 없음.
문자와 문자열을 따로 구분해서 저장하지 않음.

3. 논리형 : 논리값을 저장.
논리값으로는 TRUE와 FALSE가 있고, 각각 T와 F로 줄여서 표기 가능. true, false로는 표기 불가.
논리형끼리의 연산에서 TRUE는 1, FALSE는 0으로 취급됨.

4. 특수값 : 특수한 데이터에 대해서는 변수에 특수값들이 저장됨.
NULL : 정의되지 않음을 의미. 자료형이 없고 길이가 0임.
NA : Not available. 결측값을 의미. 데이터가 깨지는 경우나 값이 없는 경우 등.
NaN : Not a Number. 수학적으로 정의가 불가능한 값을 의미. (ex. sqrt(-3))
Inf, -Inf : 양의 무한대와 음의 무한대를 의미.

2.3.4. 변수에 조건 저장하여 사용하기

R에서는 변수에 조건을 저장하여 사용할 수 있음. 아래는 그 예시임.

```
condi <- d > 5 & d < 8
x[condi]                    # d > 5 & d < 8가 조건으로 사용됨.
```

3. 1차원 자료구조

3.1. 벡터

3.1.1. 벡터

Definition 4 *벡터는 같은 자료형의 데이터들을 저장하는 일차원 배열임.*

숫자형, 문자형, 논리형 벡터가 존재함.

벡터는 전부 같은 자료형의 데이터들만 다룰 수 있기 때문에 다른 자료형의 데이터를 넣으면 자동 변환되거나 오류가 발생함. (ex. c(1, 2, 3, T, F, "a", "b", "c")로 작성하면 1,2,3, T, F는 문자로 형변한되어 "1", "2", "3", "TRUE", "FALSE"으로 적용됨.)

3.1.2. 벡터 생성

다양한 방식으로 벡터를 생성할 수 있음. 각각의 방법들을 서로 혼용하여 사용이 가능함. 이 방법들로 생성한 벡터를 변수에 대입하여 사용함.

1. 기본 방법

```
x <- c(1, 2, 3, 4, 5)
```

2. a:b 사용 : 연속적인 숫자를 생성

```
x <- 4:6                # 4, 5, 6
x <- c(1, 2, 3, 4:6)    # 1, 2, 3, 4, 5, 6
```

3. seq() 사용 : 일정한 간격의 숫자를 생성

```
x <- seq(1, 20, 3)      # 1, 4, 7, 10, 13, 16, 19
x <- seq(1, 5)          # 1, 2, 3, 4, 5
```

4. rep() 사용 : 반복된 숫자를 생성. 숫자 자리에 c()나 a:b, seq()도 사용 가능.

```
x <- rep(1, times=5)        # 1, 1, 1, 1, 1
x <- rep(1:3, times=2)      # 1, 2, 3, 1, 2, 3
x <- rep(c(1,2,3), times=3) # 1, 2, 3, 1, 2, 3, 1, 2, 3
```

3.1.3. 벡터 원소 추출

[]를 사용하여 다양한 방식으로 벡터의 원소를 추출할 수 있음. 인덱스를 넘어서는 값으로 추출하려고 하면 NA가 추출됨.

1. 인덱스로 원소 하나 추출

```
x <- seq(1, 3, 1)           # 1, 2, 3
x[2]                        # 2
```

2. 여러 개의 원소 추출 : c(), a:b, seq(), - 등으로 추출 가능.
인덱스에 작성한 벡터의 원소 값이 인덱스로 하나씩 사용되면서 추출되는 것.
이때 숫자형 벡터를 넣으면 해당 숫자가 인덱스 값으로 들어가는 것으로 이해할 수 있는데, 논리형 벡터의 경우에는 조금 다르게 작동함. n번째 논리형 벡터 원소가 TRUE이면 n번째 원소를 추출하고, FALSE이면 추출하지 않는 방식임.

```
x[c(1, 3, 5)]               # 1, 3, 5번째 값 추출
x[1:3]                      # 1, 2, 3번째 값 추출
x[seq(1,5,2)]               # 1, 3, 5번째 값 추출

x[-2]                       # 2번째 제외하고 모두 추출
x[-c(3:5)]                  # 3, 4, 5번째 제외하고 모두 추출
```

3. 이름으로 추출하기
이름을 인덱스로 사용할 수 있음. 즉, [] 안에 원소에 붙은 이름을 따옴표로 묶어 작성하여 추출할 수 있음.

3.1.4. 벡터 원소에 대입

[]로 벡터 원소를 지정한 후 값을 대입할 수 있음. 이때 여러 개의 인덱스를 지정하고, 벡터를 대입하여 여러 원소를 조작할 수 있음.

```
x[2] <- 3
x[c(1, 3, 5)] <- c(10, 20, 30)  # 1, 3, 5번째에 10, 20, 30 대입.
```

3.1.5. 벡터 원소에 이름 지정

names() 함수로 벡터 원소에 이름을 지정할 수 있음. 아래와 같이 작성함. 원소 추출 시에 해당 이름으로 추출이 가능함. 이때 이름은 문자열이므로 따옴표로 묶어서 작성해야 함. 당연히 여러 개의 원소도 추출이 가능함.

```
x <- seq(1, 3, 1)           # 1, 2, 3
names(x) <- c("First", "Second", "Third")
...
x["Second"]                 # 2
x[c("Second", "Third")]     # 2, 3
```

3.1.6. 벡터의 연산

벡터에 대한 연산은 각 원소들끼리의 연산으로 취급됨.

1. 숫자와 벡터 연산
산술 연산의 경우 각각의 벡터 원소와 숫자의 연산임.
(ex. 3 * d + 4이면 각 원소에 3을 곱하고 4를 더한 값이 반환됨.)
논리 연산의 경우 해당 연산의 결과인 논리값들로 구성된 벡터가 반환됨. 이를 이용해 특정 값보다 큰 원소만 추출하는 등의 처리를 할 수 있음.
(ex. x[x > 5])

2. 벡터와 벡터 연산 : 대응되는 위치의 벡터 원소끼리의 연산임.
이때 두 벡터는 그 길이가 배수 관계에 있어야 함.
길이가 같은 경우 단순 연산이 되지만, 한쪽의 길이가 더 길거나 짧은 경우 짧은 쪽의 벡터가 긴 쪽의 벡터의 길이에 맞도록 원소가 여러 번 사용됨.
(ex. 길이가 각각 4, 2인 경우 2인 벡터는 두 개 이어붙인 것처럼 취급되어 연산됨.)

3.1.7. 벡터 원소의 대푯값 등 관련 함수

벡터 원소들의 대푯값을 구하는 함수들을 아래와 같음. 적당한 벡터를 추출하고 연산한 값의 대푯값을 구할 수 있음.

이때 sort()는 오름차순이 default임. 정렬 방식은 decreasing=값으로 지정함. FALSE는 오름차순, TRUE는 내림차순임. sort는 1차원 데이터만 정렬이 가능함.

length()는 벡터의 길이를 반환함. 문자열의 길이를 반환하는 것이 아님.

함수명	설명
sum( )	벡터에 포함된 값들의 합
mean( )	벡터에 포함된 값들의 평균
median( )	벡터에 포함된 값들의 중앙값
max( ), min( )	벡터에 포함된 값들의 최댓값, 최솟값
var( )	벡터에 포함된 값들의 분산
sd( )	벡터에 포함된 값들의 표준편차
sort( )	벡터에 포함된 값들을 정렬(오름차순이 기본)
range( )	벡터에 포함된 값들의 범위(최솟값~최댓값)
length( )	벡터에 포함된 값들의 개수(길이)

3.2. 펙터

3.2.1. 펙터

Definition 5 *문자형 데이터가 저장된 벡터의 특수한 형태.*

지정한 특정 집합의 문자열만을 대입할 수 있음. 벡터로 펙터를 처음 생성하면 Levels가 지정되는데, 이는 중복되는 것을 제외한 입력의 종류를 저장한 것임. 이후로는 해당 Levels에 해당되는 것들만을 대입할 수 있음.

벡터와 사용법은 동일함. Levels에 해당되는 것만을 대입할 수 있다는 점만 다름.

Levels에 포함되지 않는 값을 대입하면 NA가 대입됨.

3.3. 펙터 생성

펙터는 아래와 같이 factor() 함수에 벡터를 넣어 생성함.

```
a <- seq(1, 10)
b <- factor(a)
```

3.4. 리스트

3.4.1. 리스트

Definition 6 *리스트는 다른 자료형의 데이터들을 저장하는 일차원 배열임.*

3.4.2. 리스트 생성

리스트는 list()를 사용해 이름과 값을 모두 지정하여 생성함. 아래는 그 예시임.

리스트에는 벡터도 넣을 수 있음.

```
x <- list(name="Jun", age="23", status="True")
```

3.4.3. 리스트 원소 추출

리스트 원소 추출 시에는 [], [[]]로 인덱스를 지정하거나 $로 이름을 지정하여 추출함. 아래는 그 예시임.

list에서는 메모리 공간에 각각 이름을 붙여 할당하고, 해당 이름의 메모리 공간 속에서 공간을 다시 할당하여 데이터를 저장함. 그래서 데이터만 꺼낼 때는 [[]]로 두 개 작성함. 하나만 쓰면 이름과 데이터가 모두 추출됨.

```
x <- list(naem="Jun", age="23", status="True")
x[2]            # age, 23
x[[2]]          # 23
x$age           # 23
```

4. 2차원 자료구조

4.1. 매트릭스

4.1.1. 매트릭스

Definition 7 *매트릭스는 같은 자료형의 데이터들을 저장하는 이차원 배열임.*

세로를 열(column), 컬럼, 변수(variable)라 하고, 가로를 행(row), 관측값(observation)이라 함. 행과 열에 의해 정의되는 데이터 저장 공간을 셀(cell)이라고 함.

4.1.2. 매트릭스 생성

매트릭스는 matrix() 함수로 생성함. 이때 값은 기본적으로 열을 기준으로 채워짐. byrow=T로 행을 기준으로 채울 수 있음.

이때 자리가 데이터 개수의 배수여야 함. 자리가 데이터 개수의 배수많큼 존재하면 데이터가 배수 번 만큼 들어감.

```
x <- matrix(1:20, nrow=4, ncol=5)           # 1~20을 행 4개, 열 5개로.
x <- matrix(1:20, nrow=4, ncol=5, byrow=T)  # 행 기준으로.
```

4.1.3. 벡터와 매트릭스의 결합

벡터끼리 붙여서 매트릭스를 만들거나, 매트릭스와 벡터를 붙이거나, 매트릭스와 매트릭스를 붙일 수 있음. cbind()(column)와 rbind()(row)를 사용하는데, cbind()는 열 방향 결합(좌우에 붙임.), rbind()는 행 방향 결합임(위아래에 붙임.).

아래와 같이 사용함.

```
a <- seq(1, 10)
b <- seq(11, 20)
c <- rbind(a,b)     # 위아래로 결합
```

4.1.4. 매트릭스 원소 추출

벡터의 원소를 추출하는 것과 동일한 원리인데, 이차원이므로 인덱스 값을 2개 작성해야 함.

인덱스 값을 작성하지 않으면 해당 행/열 전부를 추출함.

```
x[1, 4]             # 1, 4에 해당하는 원소 추출
x[,4]               # 4번째 열 전부 추출
x[1:3, c(5. 6. 7)]  # 1~3번째 행의 5, 6, 7번째 열 추출
```

4.1.5. 매트릭스 행과 열에 이름 지정

벡터에서 이름을 지정한 것과 유사하지만, 매트릭스에서는 행과 열에 이름을 붙임. 벡터에서와 동일한 방식으로 이름을 사용해 원소를 추출할 수 있음.

```
rownames(score) <- c('John','Tom','Mark','Jane')
colnames(score) <- c('English','Math','Science')
...
score['Tom', 'Math']
```

4.2. 데이터프레임

4.2.1. 데이터프레임

Definition 8 *데이터프레임은 다른 자료형의 데이터들을 저장하는 이차원 배열임.*

4.2.2. 데이터프레임 생성

데이터프레임은 data.frame() 함수로 생성함. 인자로 데이터셋을 작성하면 데이터프레임이 생성됨. 이때 두 개 이상의 인자를 작성하면 벡터/매트릭스들이 cbind() 하는 것처럼 가로로 합쳐져 생성됨.

이름이 지정되어 있었으면 데이터프레임 생성 시 해당 이름이 반영됨. 또한 일차원 자료구조의 경우 원본 벡터가 저장되어 있던 변수의 식별자가 열의 이름으로 지정됨. 이름이 지정되어 있지 않다면 작성 시에 이름=을 작성하여 이름을 지정할 수도 있음(이미 지정되어 있으면 반영되지 않음).

```
b <- data.frame(a)
c <- data.frame(a, b)
e <- data.frame(original=a, new=b)
```

펙터를 사용하여 데이터프레임을 만들면, 펙터에 해당되는 부분은 펙터의 성질을 띄어 Levels가 적용됨.

리스트는 데이터프레임으로 만들면 다른 데이터셋들과는 그 형태가 좀 다름. 그렇다는 것만 알자.

4.2.3. iris 데이터셋

R에서 제공하는 연습용 데이터셋으로, 데이터프레임으로 되어 있음. 5개의 열로 이루어져 있는데, 앞쪽 4개의 열은 숫자형이고 맨 마지막 열은 펙터로 되어 있는 문자형 데이터임.

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
...
```

4.3. 2차원 자료구조 다루기

매트릭스와 데이터프레임을 다루는 문법은 거의 동일함. 여기서는 iris를 예시로 2차원 자료구조를 다루는 함수들을 정리함.

물론, []를 사용하여 접근하고 수정하는 방법도 유효함.

4.3.1. 정보 확인

dim(iris) : 행과 열의 개수 반환.
nrow(iris) : 행의 개수 반환.
ncol(iris) : 열의 개수 반환.
colnames(iris), names(iris) : 열의 이름 반환. 해당 함수에 문자형 벡터를 대입해 열 이름을 지정할 수 있음.
rownames(iris) : 행의 이름 반환. 해당 함수에 문자형 벡터를 대입해 행 이름을 지정할 수 있음.
head(iris) : 데이터셋의 앞부분 6개 행 반환.
tail(iris) : 데이터셋의 뒷부분 6개 행 반환.

unique(iris[,1]) : 데이터셋의 요소들을 중복을 제거하여 반환. 2차원 자료구조에 대해서는 행 단위로 비교를 함.
-> 오타 검수 등에서 유용함.

table(iris[,5]) : 데이터셋 각 요소의 종류별로 등장하는 횟수를 반환. 도수분포표 생성.

str(iris) : 데이터셋의 요약 정보 출력. (structure)
-> 150 obs는 행(observation)이 150개, 5 variables는 열(variable)이 5개임을 나타냄.
-> Sepal.Length num 5.1 4.9 ...은 해당 열의 이름과 자료형, 값을 나타냄. 숫자형은 num, 문자형은 chr, 논리형은 logi임.
-> Species: Factor w/3 levels "setosa",... : 1 1 1...은 3개의 Levels로 이뤄진 펙터이고 그 뒤에는 Levels의 원소들임. 맨 뒤에는 각 원소들에 해당하는 Levels를 숫자로 나타낸 것임.

4.3.2. 행과 열 다루기

1. 열의 추출
iris[,"Species"], iris[,3] : 해당 열을 벡터로 추출. 매트릭스/데이터프레임 모두 사용 가능.
iris["Species"], iris[3] : 해당 열을 데이터프레임으로 추출. 데이터프레임에서만 사용 가능.
iris$Species : 해당 열을 벡터로 추출. 데이터프레임에서만 사용 가능.

subset(iris, 조건) : 조건이 참이 되게 하는 행 추출.
-> 조건 지정 시에는 열의 이름을 사용하는데, 독특하게 이름 지정 시에 이름을 따옴표로 묶지 않음.
-> ex. Species=="setosa"나 Sepal.Length>5.0 & Sepal.Width>4.0 등으로.

2. 행/열에 대한 합계와 평균
colSums(iris[,-5]) : 각 열 별 합계를 반환.
rowSums(iris[,-5]) : 각 행 별 합계를 반환.
colMeans(iris[,-5]) : 각 열 별 평균을 반환.
rowMeans(iris[,-5]) : 각 행 별 평균을 반환.

3. 행/열의 transpose
t(iris) : 행과 열의 방향 전환. (transpose)

4.3.3. 산술 연산

1차원 자료구조에서와 같이, 데이터셋에 대한 산술 연산은 각 요소에 대한 연산으로 적용됨.

4.3.4. 자료구조 조작

class(iris) : 데이터셋의 자료구조를 문자형으로 반환.
is.matrix(iris) : 해당 데이터셋이 매트릭스이면 TRUE, 아니면 FALSE를 반환.
is.data.frame(iris) : 해당 데이터셋이 데이터프레임이면 TRUE, 아니면 FALSE를 반환.

as.matrix(iris[,1:4]) : 데이터프레임을 매트릭스로 변환.
-> 이때 작성한 데이터셋 내부에 다른 자료형들이 존재하면 자동 변환되거나 오류가 발생함. (벡터에서처럼

동작함.)
-> 반대로 매트릭스를 데이터프레임으로 변환하는 것은 당연히 data.frame()으로 데이터프레임을 생성하기만 하면 됨.

5. 제어의 조작

5.1. 제어의 조작

5.1.1. 조건문

if, if else, else문 사용 가능. 이때 {와 }의 위치를 if/else if/else 바로 옆에 써 줘야 함. 다음 줄에 작성하면 인터프리터가 정상적으로 인식하지 못함.

아래의 형태로 작성함.

```
if(조건){
    ...
} else if{
    ...
} else{
    ...
}

if(조건) ...
```

ifelse(조건, 값1, 값2) : 조건이 참이면 값1을, 거짓이면 값2를 반환.
-> 이 함수로 if문을 대체할 수 있음.

5.1.2. 반복문

1. for문
아래와 같이 반복 변수와 데이터셋을 작성하면 매 loop마다 해당 반복 변수에 데이터셋의 값이 순차적으로 들어감.

```
for(반복 변수 in 데이터셋){
    ...
}
```

2. while문
아래와 같이 작성함.

```
while(비교조건){
    ...
}
```

3. break과 next
R에서는 continue의 기능을 next로 수행함.

5.1.3. apply() 함수

반복 작업의 대상이 매트릭스/데이터프레임의 행/열인 경우 반복문을 apply() 함수로 대체할 수 있음.

apply(iris[,1:4], 1, mean) : 데이터셋, 행/열 방향, 함수를 지정하여, 해당 데이터셋의 행/열 단위로 함수를 적용함.
-> 첫 번째 인자로 데이터셋을 작성함.
-> 두 번째 인자로 행/열 방향을 지정함. 1이면 행 방향, 2이면 열 방향으로 함수가 적용됨.
-> 함수 지정 시에는 ()를 제외하고 식별자만 작성함.

5.1.4. 사용자 정의 함수

아래의 형태로 사용자가 함수를 정의하여 사용할 수 있음. 이때 return()에 괄호 씌워줘야 하는 것 유의.

```
함수명 <- function(매개변수 목록){
    ...
    return(반환값)
}
```

매개변수 목록에 (x, y=2) 등으로 초기값을 지정할 수 있음.

호출 시에 인자를 (x=10, y=3) 등으로 작성할 수 있음. 이 경우 인자는 (10,3)으로 전달됨.

여러 개의 반환값을 보내야 하는 경우 주로 데이터셋으로 만들어 반환함.

6. 자료의 탐색

각 함수의 option들은 정리하지 않음.

6.1. 자료의 종류

6.1.1. 자료의 종류

자료는 범주형/연속형 자료인지와 단일변수/다중변수 자료인지 분류할 수 있음. 즉, 자료는 총 4가지로 분류됨. 각 분류에 따른 분석 방법들이 존재함. 여기에서 변수는 연구/조사/관찰하는 대상의 특성을 의미함.

범주형 자료(categorical data) : 범주/그룹으로 구분될 수 있는 값들로 구성된 자료.
-> 범주형 자료 또한 숫자로 표기할 수 있으므로 숫자로 표기된다고 연속형 자료인 것은 아님. -> ex. 혈액형, 성별, 찬성 여부

연속형 자료(numerical data) : 연속적인 수치로 표현되는 값들로 구성된 자료.
-> 대소비교와 대푯값의 계산이 가능함.
-> ex. 몸무게, 키, 온도

단일변수 자료(univariate data) : 하나의 변수로만 구성된 자료. 일변량 자료라고도 함.
-> 벡터에 저장하여 분석함.

다중변수 자료(multivariate data) : 여러 개의 변수로 구성된 자료. 다변량 자료라고도 함. 특별히 두 개의 변수로 구성된 것은 이변량 자료(bivariate data)라고 함. -> 매트릭스/데이터프레임에 저장하여 분석함.
-> 매트릭스/데이터프레임에서 하나의 열이 하나의 변수를 나타냄.

6.2. 단일변수 범주형 자료의 탐색

6.2.1. 분석 방법

범주형 자료에 대한 가장 기본적인 분석은 각 범주에 대한 개수를 세는 것임. 개수를 세서 비율을 계산할 수 있음. 도수분포표, 막대그래프, 원그래프 등으로 분석 가능.

6.2.2. 탐색

도수분포표를 만들고 막대그래프, 원그래프 등을 그릴 수 있음.

1. 도수분포표의 작성
아래와 같이 벡터를 생성하고, table() 함수로 도수분포표를 생성할 수 있음. 전체 개수로 해당 값을 나누면 각 범주에 대한 비율을 알 수 있음.

```
favorite <- c('WINTER', 'SUMMER', 'SPRING', 'SUMMER', 'SUMMER',
'FALL', 'FALL', 'SUMMER', 'SPRING', 'SPRING')
table(favorite)
table(favorite) / length(favorite)
```

2. 막대그래프/원그래프 작성
아래와 같이 table()로 얻은 데이터를 barplot()/pie()에 넣어 막대그래프/원그래프를 그릴 수 있음.

```
d <- table(favorite)
barplot(d)
pie(d)
```

6.3. 단일변수 연속형 자료의 탐색

6.3.1. 분석 방법

연속형 자료에 대해서는 대소비교와 대푯값의 계산 등이 가능하므로 다양한 분석이 가능함. 평균, 중앙값, 절사평균, 사분위수, 산포(분산, 표준편차)을 구하고, 히스토그램과 상자그림으로 나타내는 등으로 분석할 수 있음.

1. 평균, 중앙값, 절사평균
절사평균은 하위 n%와 상위 n%를 제외한 중간 값들만을 가지고 계산한 평균을 말함.

mean(iris$Sepal.Length) : 평균을 반환.
-> 마지막 인자로 trim=0.2 등을 지정하여 절사평균을 구할 수 있음.
-> ex. trim=0.2로 지정하면 상하위 20%를 제외하는 것.

median(iris$Sepal.Length) : 중앙값을 반환.

2. 사분위수
사분위수는 주어진 값들을 크기순으로 나열했을 때 4등분하는 3개의 값들을 말함. 앞에서부터 '제1사분위수(Q1)', '제2사분위수(Q2)', '제3사분위수(Q3)'라고 부르며, 제2사분위수(Q2)는 중앙값과 동일할 수 있음.

quantile(iris$Sepal.Length) : 4분위수와 최대/최솟값을 반환. (0%, 25%, 50%, 75%, 100%)
-> 이때 4분위수로는 데이터의 값에서 선택하는 것이 아니라, 수치적인 계산을 통해 나온 값을 반환함.
-> 마지막 인자로 %에 대한 벡터를 지정할 수 있음. (ex. a <- c(0.3, 0.6, 0.9), quantile(mydata, a)이면 30%, 60%, 90%가 반영됨. quantile(mydata, (0:10)/10) 이런 식으로 할 수도 있음.)

summery(iris$Sepal.Length) : 사분위수, 최대/최솟값, 평균값을 한 번에 반환.

3. 산포
자료의 값들이 퍼져 있는 정도를 산포라고 함. 분산과 표준편차로 산포를 파악할 수 있음. 분산과 표준편차가 클수록 더 퍼져 있는 것.

var(iris$Sepal.Length) : 분산을 반환.
sd(iris$Sepal.Length) : 표준편차를 반환.
range(iris$Sepal.Length) : 값의 범위를 반환.
diff(iris$Sepal.Length) : 값들의 차이를 반환.
-> diff(range(iris$Sepal.Length))로 작성하여 최댓값과 최솟값의 차를 구할 수 있음.

6.3.2. 탐색

히스토그램과 상자그림을 그릴 수 있음.

1. 히스토그램
막대그래프와 유사하게 생겼지만, 연속형 자료의 분포를 시각화할 때 사용함. 막대그래프를 그릴 때는 값의 종류로 분류를 했지만, 연속형 자료의 경우에는 종류로 분류할 수 없기 때문에 구간으로 분류하는 히스토그램을 사용함.

hist(iris$Sepal.Length) : 히스토그램 생성.

2. 상자그림(box plot)
사분위수를 그래프로 나타낸 것.

boxplot(iris$Sepal.Length) : 상자그림 생성.
-> boxplot(Petal.Length~Species, data=iris)와 같이 작성하여 여러 개의 데이터에 대한 상자그림을 나타낼 수 있음. 다중변수에 대한 것이지만 상자그림이므로 정리함.
-> boxplot(iris$Petal.Length, iris$Species)로 작성하면 단순히 각각에 대한 상자그림이 한꺼번에 출력됨.

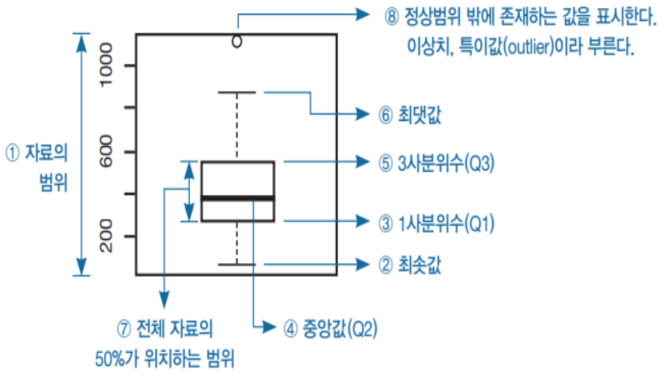


그림 5-9 상자그림의 구성 요소

6.4. 다중변수 자료의 탐색

6.4.1. 산점도

산점도(scatter plot)는 2개의 변수로 구성된 자료의 분포를 나타내는 그래프임.

plot(iris$Sepal.Length, iris$Petal.Length) : 두 변수에 대한 산점도 생성.
pairs(dataset) : 여러 개의 변수들 간의 산점도 생성.
-> 산점도는 2개 변수에 대한 것이므로 여러 개의 산점도를 생성함.
-> 출력되는 변수 이름을 기준으로 위아래로는 해당 변수가 x축, 좌우로는 해당 변수가 y축임. 당연하게도 대각선으로 잘라서 보면 대칭적임.

아래는 pairs에 대한 예시임.

```
vars <- c("mpg","disp","drat","wt")
target <- mtcars[,vars]
head(target)
pairs(target)
```

6.4.2. 상관분석

데이터가 얼마나 선형성을 보이는지 수치상으로 나타내는 것을 상관분석이라고 함. 상관분석을 통해 두 변수가 서로 얼마나 영향을 주고받는지를 알 수 있음.

피어슨 상관계수(Pearson's correlation coerricent)라는 것을 사용하는데, 자세한 것은 몰라도 됨. 피어슨 상관계수는 주로 r로 표기함. $-1 \leq r \leq 1$이고, r이 0보다 크면 양의 상관관계, 작으면 음의 상관관계를 가지는 것임. r이 -1, 1에 가까울수록 상관관계가 높음. 0이면 관련 없음.

선형성은 그래프로 나타냈을 때의 추세선으로 확인해볼 수도 있음. 회귀식을 도출하면 이를 이용해 그래프에 회귀선을 그릴 수 있음. 회귀선은 분산되어 있는 값을 평균적으로 나타내는 그래프이고, 회귀식으로 나타냄.

cor(iris$Sepal.Length, iris$Petal.Length) : 상관상수 반환.
-> 인자로 여러 개의 변수(매트릭스/데이터프레임)를 작성하면 모두에 대한 상관상수를 반환함. (ex. iris[,1:4])

lm(Petal.Length~Species, data=iris) : 회귀식 반환.
abline(회귀식) : 회귀선 생성.
-> plot으로 산점도를 생성한 후에 회귀선을 생성해야 확인할 수 있음.

6.4.3. 선그래프

산점도의 plot()에서 lty=1, lwd=1만 옵션으로 넣어주면 선그래프가 그려짐.

plot()으로 그래프를 생성한 후, lines() 함수로 선을 추가로 넣을 수 있음. lines()는 독립적인 사용이 불가능함.

lines(iris$Sepal.Length, iris$Petal.Length) : 산점도에 선을 삽입함.

7. 기타

7.1. 파일 입출력

7.1.1. .csv 파일

데이터셋을 다룰 때에는 .csv 파일을 주로 사용함. .csv 파일(comma seperated value)은 각 데이터를 쉼표로 구분한 파일임. 파일에 따라 쉼표가 아니라 공백문자나 tab, 기타 특수기호로 되어 있기도 한데, 구분자가 존재하면 .csv라고 함. 여기서는 엑셀 등으로 되어 있는 데이터셋을 .csv 파일로 변환해서 사용함.

7.1.2. 작업 디렉토리 지정

getwd() : 현재 작업 디렉토리(working directory)의 경로를 문자형으로 반환.
setwd("D:/source") : 작업 디렉토리(working directory)를 문자형으로 작성한 경로로 지정함.
-> 파일을 읽고 쓰기 전에 우선 작업 디렉토리를 지정해야 함. 해당 디렉토리 내에서 파일을 읽고 쓸 수 있음.
-> 경로의 맨 앞에는 드라이브명:을 써 줘야 하고, 경로의 디렉토리는 /로 구분함.
-> getwd()로 얻은 경로를 활용하면 편리함.

7.1.3. 파일 읽기/쓰기

read.csv("airquality.csv", header=T) : 작업 디렉토리 내부의 해당 csv 파일을 읽어 반환.
-> 파일로부터 읽어 온 데이터셋은 데이터프레임으로 반환됨.
-> header는 첫 줄을 열 이름으로 할 것인지를 지정함. header=T이면 첫번째 줄의 문자열들이 가져온 데이터의 열 이름으로 지정됨.

write.csv(my.iris, "myiris.csv", row.names=F) : 해당 데이터셋을 지정한 파일에 작성함.
-> row.names=F로 지정하면 작성 시 각 행에 행 번호를 붙이지 않음.

7.1.4. 다른 파일의 사용자 정의 함수 사용하기

작업 디렉토리를 지정하고, 해당 위치에 존재하는 .R 소스 파일을 source() 함수로 가져오면 내부의 사용자 정의 함수를 호출할 수 있음.

source("myfunc.R") : 작업 디렉토리에 있는 해당 .R 소스 파일을 가져옴.

7.2. 위치 찾기

7.2.1. 위치 찾기

아래의 함수들로 데이터셋에서 특정 데이터의 위치를 반환받을 수 있음.

which(조건) : 데이터셋에서 해당 조건이 참이 되게 하는 요소의 인덱스를 반환.
-> 마지막 인자에 arr.ind=TRUE로 지정하면 행과 열의 인덱스 모두를 반환함. FALSE가 기본값이고, 기본적으로 행 인덱스만 반환함.
-> 즉, 조건에 TRUE/FALSE의 집합을 넣으면 TRUE인 인덱스를 반환하는 것.

which.max(iris$Petal.Length) : 해당 데이터셋에서 최댓값을 가지는 요소의 인덱스를 반환.
which.min(iris$Petal.Length) : 해당 데이터셋에서 최솟값을 가지는 요소의 인덱스를 반환.

아래는 그 예시임.

```
idx <- which(iris$Petal.Length > 4.0)
iris.big <- iris$Petal.Length[idx]
```

7.3. 패키지

7.3.1. 패키지 설치/사용

R studio에서 패키지의 설치/사용은 GUI에서 간단히 수행할 수 있지만, 코드로도 가능함.

install.packages("jsonlite") : 패키지 설치
library(jsonlite) : 패키지 사용(include)
data(iris) : 패키지로부터 데이터셋 로드
detach("package:mlbench", unload = TRUE) : 패키지 제외

참고로, 패키지들은 R을 설치했던 CRAN에서 가져와 설치하는 것임.

8. 전처리

8.1. 결측값의 처리

8.1.1. 결측값

Definition 9 *결측값은 데이터 수집 과정에서 데이터를 얻지 못한 경우를 나타냄.*

R에서 결측값은 NA로 나타냄.

결측값이 존재하면 연산이 정상적으로 수행되지 못함. 연산을 시도하면 NA가 반환됨.

결측값은 아래의 두 가지 방법으로 처리할 수 있음. 이때 치환은 까다롭기 때문에 대체로 삭제하는 것으로 처리함.

1. 결측값을 찾아서 제거함.
2. 결측값을 추정하여 적절한 값으로 치환함.

8.1.2. 1차원 데이터셋의 처리

1. 결측값 확인

is.na(x) : 해당 데이터셋의 각 원소들을 NA인지 검사하여 논리형 데이터(NA이면 TRUE)의 집합을 반환. 1차원이든 2차원이든 기존 데이터셋에서 원소만 TRUE/FALSE로 바꾼 형태로 반환함.
sum(x) : 해당 데이터셋의 모든 원소들의 합을 반환. NA가 존재하면 NA가 반한됨.
sum(x, na.rm=TRUE) : NA를 제외한 것들의 합을 반환.

예시.

```
is.na(x)            # FALSE TRUE FALSE 등
sum(is.na(x))       # NA의 개수
sum(x, na.rm=TRUE)  # NA를 제외한 것들의 합
```

2. 결측값 처리
결측값을 삭제함. 0으로 치환할 수도 있지만 이는 적절하지 않음.

na.omit(x) : NA를 제거한 버전의 데이터셋을 반환. 이상한 형태로 반환하는데 이를 벡터 등으로 바꿔줘야 함.

예시.

```
x[is.na(x)] <- 0            # NA를 0으로 치환
x <- as.vector(na.omit(x))  # 결측값 제거
```

8.1.3. 2차원 데이터셋의 처리

1. 행/열별 결측값 확인
for문으로 행/열 별로 is.na()를 사용하거나, 또는 함수를 정의하여 apply로 해당 함수를 열 별로 적용되게 하거나, rowSums()/colSums()(이게 젤 편한듯) 등을 사용할 수 있음.

예시.

```
# 열 별 결측값 확인 방법
for(i in 1:ncol(x)){        # 반복문 이용
```

```
  cat(sum(is.na(x[,i])))
}

myfunc <- function(){
    return(sum(is.na(sum))) # apply() 이용
}
apply(x, 2, myfunc)

colSums(is.na(x))           # colSums() 사용
```

2. 결측값 처리(제거)
complete.cases()를 사용함.

complete.cases(x) : 각 행별로 NA의 존재 여부를 확인하여 TRUE/FALSE로 구성된 데이터셋을 반환함. 즉, 행별로 적용되는 is.na()임.

예시.

```
y <- x[complete.cases(x),]  # NA가 있는 행 삭제
z <- x[!complete.cases(x),] # NA가 있는 행만 가져옴
```

8.2. 특이값의 처리

8.2.1. 특이값

Definition 10 *특이값(이상치)은 잘 입력되었지만 정상적이지 않은 값을 말함.*

특이값을 포함하여 데이터를 분석하면 그 결과가 왜곡될 수 있기 때문에 결측값처럼 삭제하는 것이 합리적임.

8.2.2. 특이값의 처리

1. 특이값 확인
상자그림(boxplot())으로 특이값의 존재를 확인할 수 있음. 해당 부분에 정리되어 있지만, 1차원 데이터셋(열)을 넣으면 해당 열에 대해 상자그림이 그려지고, 2차원 데이터셋 전체를 넣으면 각 열에 대한 상자그림이 그려짐.

boxplot.stats(...)$out으로 특이값의 값을 얻을 수 있음.

boxplot.stats(dataset) : 해당 1차원 데이터셋에서 특이값을 찾아 그 값을 list로 반환함. list 내부의 out에 해당 값이 들어 있음.

예시.

```
boxplot.stats(x$Income)$out     # 특이값들의 집합이 반환됨
```

2. 특이값 제거
특이값의 값을 찾고, %in% 연산자로 해당 값의 위치를 얻고(which()로도 가능하지만 이게 더 간단함), 그 위치의 값을 NA로 지정한 뒤 결측값 제거를 수행함.

%in%는 특별연산자로, 앞에 작성한 데이터셋의 각 요소가 뒤에 작성한 데이터셋의 값 중 하나와 같은지를 비교하여 TRUE/FALSE로 구성된 데이터셋을 반환함.

예시.

```
k <- boxplot.stats(x$Income)$out
j <- x$Income %in% k
x[j,] <- NA

newdata <- x[complete.cases(x),]    # 결측값 제거
```

8.3. 데이터 정렬

8.3.1. 데이터 정렬

1. 1차원 데이터셋의 정렬

order(x) : x를 오름차순으로 정렬할 때 해당 순서에 들어가는 인덱스의 집합을 반환함. 데이터셋을 여러 개 작성하여 여러 기준에 대해 정렬할 수 있음. 이때 데이터셋 앞에 -를 붙이면 현재 방향의 역방향으로 정렬됨.
sort(x) : x를 오름차순으로 정렬함.

둘 다 두 번째 인자로 decreasing=TRUE를 작성하여 내림차순에 대해서 처리할 수 있음. 왜인지는 모르겠는데(아마 인자가 여러 개 들어올 수 있어서인 듯) 이때 decreasing=을 생략하면 에러남.

2. 2차원 데이터셋의 정렬
행/열에 대해서(주로 행) order()를 사용함.

예시.

```
y <- x[order(x$Sepal.Width, decreasing=TRUE),]  # Sepal.Width에 대해 내림차순 정렬
z <- x[order(x$Sepal.Width, -x$Sepal.Length, decreasing=TRUE),]
```

8.4. 데이터 분리/선택/병합

8.4.1. 데이터 분리/선택

1. 데이터 분리
split()으로 데이터셋에서 특정 열이 가지는 값에 대해서 행별로 분리하여 나타낼 수 있음.

split(x, x$Species) : x의 원소(행)들을 x$Speceis의 값을 기준으로 분리하고 각 원소(행) 별로 묶어 list로 반환함. list이므로 $를 사용하여 각 부분에 접근할 수 있음. 이때 이름은 x$Species의 값이고, 각 부분은 데이터프레임으로 저장됨.

예시.

```
x <- split(x, x$Species)
x <- x$setosa                   # Species가 setosa인 행들만 가져옴
```

2. 데이터 선택
이미 등장했던 subset()으로 특정 조건을 만족하는 데이터를 선택하여 가져올 수 있음. subset()에서 특이한 점은 열을 작성할 때 $ 등을 작성하지 않고 단순히 식별자만 작성해도 된다는 것임.

이때 select=c(Petal.Length, Sepal.Width)등으로 해당 열에 대해서만 가져오도록 지정할 수 있음.

예시.

```
y <- subset(x, Sepal.Width > 1.0)
z <- subset(x, Sepal.Length < 2.0, select=c(Sepal.Width, Petal.Length))
```

8.4.2. 데이터 병합

병합은 분리된 두 데이터셋에 대해서 각각의 특정 열을 기준으로 합치는 것을 말함. merge()를 사용하여 두 데이터셋(2차원)을 병합할 수 있음. 이때 all, by를 지정할 수 있는데 x는 첫 번째 데이터셋을 , y는 두 번째 데이터셋을 의미함.

merge(a, b, by=c("name")) : a와 b가 공통된 이름(name)의 열을 가지고 있을 때, 해당 이름의 열을 기준으로 병합함. 유의할 점은 이름을 벡터로 지정한다는 것임.

a, b가 공통된 이름의 열을 가지고 있지 않다면 아래와 같이 각각의 열 이름을 지정해야 함. 해당 두 열을 기준으로 병합되는데, 이때 병합된 결과는 a의 이름을 가지게 됨.

```
merge(a, b, by.x=c("aname"), by.y=c("bname"))
```

특정 위치에 값이 존재하지 않으면 NA로 처리되는데, 기본적으로 NA가 있는 행은 제거됨. 아래와 같이 all을 지정하여 삭제되지 않도록 할 수 있음.

```
merge(a, b, by=c("name"), all.x=T)   # a로부터 온 행들은 삭제되지 않음.
merge(a, b, by=c("name"), all.y=T)   # b로부터 온 행들은 삭제되지 않음.
merge(a, b, by=c("name"), all=T)     # 모든 행들이 삭제되지 않음.
```

8.5. 데이터 샘플링

8.5.1. 데이터 샘플링

Definition 11 *데이터셋에서 임의의 데이터들을 추출하는 것을 데이터 샘플링이라고 함. 해당 데이터셋의 모든 데이터를 확인할 수 없는 경우 등에서 사용함.*

복원추출과 비복원추출이 있음. 여기서는 비복원추출만 다룸.

8.5.2. 추출 방법

1. 1차원 데이터셋의 추출

sample(x, size=10, replace=FALSE) : x에서 size만큼 무작위로 추출하는데, replace가 FALSE이면 비복원추출, TRUE이면 복원추출을 함.

2. 2차원 데이터셋의 추출
행/열의 인덱스에 대해 sample()을 하여 그 인덱스를 사용함.

예시.

```
index <- sample(1:nrow(x), size=20, replace=FALSE)  # 비복원추출
x <- x[index,]
```

8.5.3. seed 지정

set.seed()로 seed를 지정하면 바로 다음에 수행되는 sample()에 대해 seed가 지정되어 해당 seed에 대해 무작위 작업을 수행함. 당연하게도 sample()의 seed()가 같으면 동일한 결과를 도출함.

물론 실제로 시드 값을 정할 때는 여러 데이터가 사용되기 때문에, 같은 정수를 지정해도 컴퓨터를 껏다 킨다던가 며칠 수가 된다던가 하면 다른 값이 나올 수 있음. 또한 너무 큰 값을 넣으면 오작동할 수 있음.

8.5.4. 데이터 조합

combn()을 사용하여 해당 데이터셋(1차원)의 원소를 뽑아 만들 수 있는 조합을 얻을 수 있음.

combn(x, 10) : x에서 10개를 뽑는 조합을 반환함. 이때 반환 결과는 열 단위로 구성되어 있음.

8.6. 데이터 집계

8.6.1. 데이터 집계

집계(aggregation)는 특정 데이터 집합에 대해서 합계, 평균 등을 계산하는 것을 말함. 사실 지금까지 계속 해 온 것인데, aggregate()는 apply(), subset(), split()을 합친 것처럼 동작함.

aggregate(x, by=list(iris$Species), FUN=mean) : x를 list(iris$Species)에 대해서 split하고, 각 데이터 집합에 FUN에 해당하는 함수를 적용하여 그 결과를 반환함.

list에 여러 기준을 작성하여 해당 기준들에 대해서 데이터 집합을 split할 수 있음.

예시.

```
k <- aggregate(x, by=list(cyl=x$cyl, vs=x$vs), FUN=max)
```

반환 결과는 split한 기준 열의 값으로 구성된 열이 가장 왼쪽에 추가되는데, 이때 리스트에 이름을 지정하면 이 열이 해당 이름을 가짐.

```
> agg <- aggregate(iris[,-5], by=list(표준편차=iris$Species),
+                  FUN=sd)
> agg
    표준편차 Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa    0.3524897   0.3790644    0.1736640   0.1053856
2 versicolor    0.5161711   0.3137983    0.4699110   0.1977527
3  virginica    0.6358796   0.3224966    0.5518947   0.2746501
```

9. 시각화

실제로는 시각화 기법보다는 ggplot2를 주로 사용함.

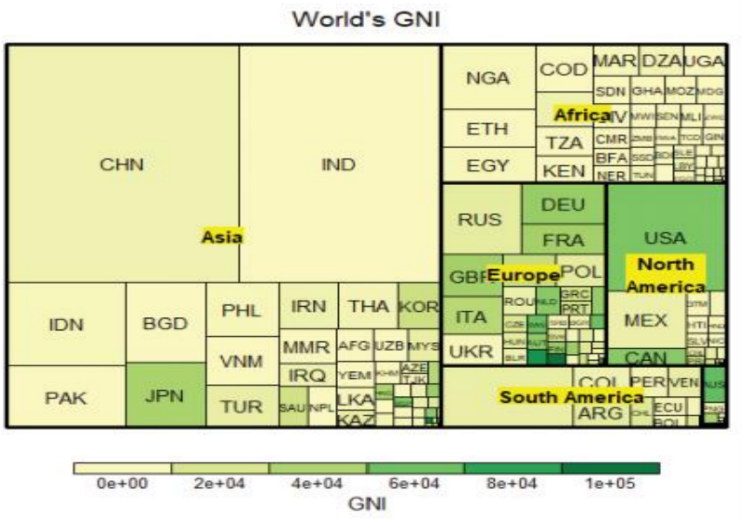
9.1. 데이터 시각화 기법

필수로 작성해야 하는 인자만 정리함.

9.1.1. 트리맵

트리맵은 계층 구조를 가지는 데이터에 대해서 해당 계층 구조에 따라 데이터를 분류하고, 지정한 기준으로 각 원소의 크기와 색을 나타내는 시각화 기법임. treemap()을 사용하려면 treemap 패키지를 사용해야 함.

treemap(x, index=c("continent", "iso3"), vSize="population", vColor="GNI") : x 데이터셋(2차원)에 대해 coninent, iso3 순서로 계층을 구분하며, population을 기준으로 크기가 정해지고, GNI를 기준으로 색이 정해지는 트리맵을 생성함.

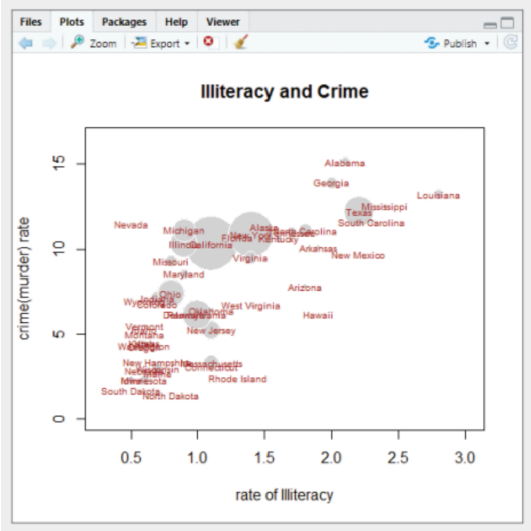


9.1.2. 버블 차트

버블차트는 원으로 나타낸 차트임. symbols()로 버블 차트를 만들 수 있고, text()로 해당 차트 위에 글자를 작성할 수 있음. 이때 text()는 단독으로 사용이 불가능함.

symbols(a, b, circles=x$Population) : a, b 데이터셋(1차원. 열 등을 지정)을 각각 x, y 좌표로 하고, circles로 지정된 값을 반지름으로 하는 원들로 버블차트를 생성함.

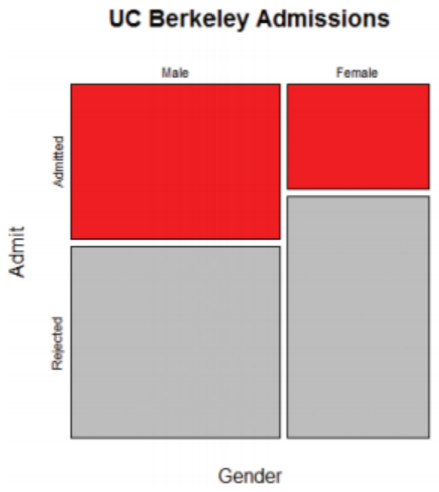
text(a, b, rownames(x)) : a, b 데이터셋을 각각 x, y 좌표로 하고 각 위치에 세 번째 인자에 작성된 데이터셋(1차원, 문자열)을 출력함.



9.1.3. 모자이크 플롯

모자이크 플롯은 다중변수 범주형 데이터에 대해 각 그룹별 비율을 사각형으로 나타낸 것임. mosaicplot()을 사용하는데, 형식이 독특하므로 유의.

mosaicplot(~a+b, data=x) : ~ 뒤에 x축, +뒤에 y축을 작성하고, data에 데이터셋을 지정함.



9.2. ggplot

9.2.1. ggplot

Definition 12 *ggplot은(ggplot2) R에서 사용할 수 있는 가장 대표적이면서 자주 사용되는 시각화 라이브러리임.*

ggplot2 패키지를 설치해야 함.

9.2.2. 기본 구조

ggplot을 사용한 시각화에서는 하나의 ggplot() 함수와 여러 개의 geom_*() 함수를 더하기로 연결하여 작성함. ggplot()에서는 데이터셋을 지정하고 데이터를 그룹화를 하고, geom_*()에서는 그래프의 종류와 디테일한 옵션들을 지정함.

ggplot(data=mydata, aes(x=a, y=b)) : data에 데이터셋을 지정(데이터프레임이어야 함)하고, aes()의 x/y에 x/y축으로 사용할 열의 이름을 지정함. x만을 사용한다면 y는 생략함. aes의 인자로는 $ 등을 생략하고 열 이름만을 작성함.

aes의 인자로 fill/color=Species 등을 작성하여 해당 열의 값들을 기준으로 그룹화하여 내부 색을 채우거나 테두리 색을 칠함. 이때 그래프의 종류에 따라 단순히 색만 칠하기도 하고(point 등), 그래프가 여러 개로 나뉘기도 함(boxplot 등). fill/color로 지정하는 값은 데이터셋의 데이터일 필요는 없고, 데이터셋과 크기만 맞으면 됨.

예시.

```
ggplot(data=x, aes(x=Sepal.Width, fill=Species, color=Species)) +
  geom_histogram() +
  ggtitle("강수량") +
  labs(x="월")
```

9.2.3. 관련 함수들

그래프에 따른 함수들은 아래와 같음.

geom_bar() : 막대그래프를 그림. x/y중 하나만 있으면 됨. 둘 다 있다면 stat="identity"로 높이를 y값으로 지정해야 함.
geom_histogram() : 히스토그램을 그림. x/y중 하나만 있으면 됨.
geom_point() : 산점도를 그림. x/y 둘 다 필요.
geom_boxplot() : 상자그림을 그림. x/y중 하나만 있으면 됨.
geom_line() : 선그래프를 그림. x/y 둘 다 필요.

ggtitle("강수량") : 플롯의 제목 지정.
labs(x="월", y="강수량") : 플롯의 x/y축 레이블 지정.
theme() : 플롯의 색, 폰트 등 지정.
coord_flip() : 플롯을 가로로 출력.

9.2.4. esquisse

esquisse 패키지를 사용하면 ggplot을 GUI로 그리고 옵션을 설정할 수 있음.

먼저 사용할 데이터셋을 준비함. 이후 ggplot2/esquisse 패키지를 추가하고, R studio 왼쪽 상단의 Addins를 누르면 ggplot2 builder를 실행시킬 수 있음. 해당 builder에서 데이터셋을 선택하고 ggplot()에 의한 플롯을 만들면 그 코드를 얻을 수 있음. 또한 데이터셋의 값도 수정할 수 있고, 데이터를 구글시트, URL 등으로도 가져올 수 있음.

9.3. 차원 축소

9.3.1. 차원 축소

Definition 13 *차원 축소는 차원(변수의 개수)을 낮춰 고차원의 데이터 분포를 분석하는 것을 말함.*

차원 축소를 하면 데이터 손실이 발생하므로, 정확한 분석/시각화라기보다는 대략적인 분포 정도를 확인하기 위한 것임.

차원 축소는 Rtsne 패키지의 Rtsne() 함수로 함. 이후 ggplot 등으로 결과를 출력할 수 있음.

9.3.2. 차원 축소 방법

1. 중복 제거
차원 축소는 Rtsne() 함수로 하는데, 이를 위해 우선 데이터셋의 중복 데이터를 삭제해야 함.

duplicated(x) : 데이터셋(2차원)에 대해서 각 행이 중복되는지를 TRUE/FALSE의 집합으로 반환하는데, 값이 중복되는 행이 존재할 경우 뒤쪽 행 위치의 값을 TRUE로 반환함.

예시. 아래와 같이 which()와 사용하여 중복 행의 위치를 찾고, 해당 행을 삭제할 수 있음.

```
idx <- which(duplicated(x))
x <- x[-idx,]
```

2. 차원 축소

Rtsne(x, dims=2, perplexity=10) : 첫 번째로 데이터셋을 작성하고, dims로 축소할 차원을, perplexity로 샘플의 개수를 지정함.

샘플이 너무 많으면 보기에 불편할 수 있어 perplexity로 샘플 개수를 지정하는 것.

Rtsne()의 결과는 list로 반환됨. 반환된 list를 확인해보면 여러 값들이 들어 있는데, 이 중 N(이름)은 기존 데이터의 개수를 가지고 있고, Y에는 축소한 결과 데이터가 들어 있는데 이는 메트릭스임. 결과적으로 Y를 사용하는데 이를 ggplot으로 출력하려면 데이터프레임으로 변환해야 함. 이때 변환 후 각 열의 이름은 X1, X2 등임.

예시.

```
a <- Rtsne(x, dims=2, perplexity=10)
b <- data.frame(a$Y)

ggplot(b, aes(x=X1, y=X2)) +
    geom_point()
```

9.3.3. 3차원 그래프 그리기

기존의 ggplot()에서는 3차원 플롯을 그리는 것을 다루지 않았음. R에서 제공하는 scatter3d() 함수를 쓰면 x,y,z만 지정해도 그래프를 자동으로 그려줌.

scatter3d(x=ds$X1, y=ds$X2, z=ds$X3) : 지정함 x,y,z에 따라 3차원 그래프를 그림. 기본적으로 회귀면을 함께 출력하는데, 마지막 인자에서 surface=FALSE로 지정해 출력하지 않을 수 있음.

10. 문자열 처리

10.1. 문자열 처리

10.1.1. Stringr

문자열 데이터 가공에는 Stringr 패키지를 사용함.

특히 빅데이터 등에서는 문자열 데이터를 가져와서 가공해야 하는 경우가 많음.

10.1.2. 정규표현식

Stringr의 문자열 검색/비교에서는 정규표현식을 사용함.

이 문자열에는 비교에 대한 정규표현식을 작성할 수 있는데, 아래와 같은 형식을 가짐. ∧는 맨 앞을 나타내고, $는 맨 뒤를 나타냄. []는 하나의 문자로 묶는 것임.

"abc" -> 해당 문자열이 존재하면 TRUE.
"∧abc" -> 해당 문자열이 맨 앞에 존재하면 TRUE.
"∧[aAbc]" -> [] 내부의 문자 중 하나가 맨 앞에 존재하면 TRUE.
"∧[aA][bB]" -> 첫번째 문자가 첫번째 [] 내부의 문자이고 두번째 문자가 두번째 [] 내부의 문자이면 TRUE.
"abc$" -> 해당 문자열이 맨 뒤에 존재하면 TRUE.
"[aA][bB]$" -> 뒤에서 첫번째 문자가 두번째 [] 내부의 문자이고 뒤에서 두번째 문자가 첫번째 [] 내부의 문자이면 TRUE.

예시.

```
x <- x[str_detect(x, "^a")]     # a로 시작하는 문자열만 추출
```

10.1.3. Stringr 함수들

1. str_detect(x, "abc") : 문자열 데이터셋 x에서 해당 문자열에 대해 검색/비교를 하여 TRUE/FALSE의 집합을 반환함.

2. str_count(x, "abc") : 문자열 데이터셋 x에서 각 원소에 대해 해당 문자열이 들어간 횟수의 집합을 반환함.

3. str_c(...) : 문자열 여러 개를 인자로 작성하면 해당 문자열들을 순서대로 결합한 문자열을 반환함. 문자열 데이터셋과 문자열을 작성하면 데이터셋의 각 문자열에 대해 결합한 문자열들의 집합을 반환함. 문자열 데이터셋과 collapse="abc"를 작성하면 원소들을 해당 문자열로 연결한 문자열을 반환함.

예시.

```
x <- str_c("a", "bc", "cd")
x <- str_c(x, "!!!")
str_c(x, collapse="-")
```

4. str_length(x) : 해당 문자열의 길이 반환.

단순 length(x)는 x의 데이터 개수를 반환함. length('1')은 1은 반환함.

5. str_replace(x, "abc", "def") : 해당 문자열에서 맨 앞 "abc"를 "def"로 교체하여 반환함.
str_replace_all(x, "abc", "def") : 해당 문자열에서 모든 "abc"를 "def"로 교체하여 반환함.

특히 "4,100"이런 경우 ','를 ”로 바꾸는 식으로 자주 사용됨. 지운 후에 as.numeric()으로 해당 문자열을 숫자로 바꾸는 등.

6. str_split(x, '/') : 해당 문자열을 '/'을 기준으로 쪼개서 문자열의 집합을 반환함.

7. str_sub(x, start=2, end=5) : 해당 문자열에서 2번째~5번째까지만 추출해서 문자열로 반환함. 이때 문자열은 1번째부터 시작하는 것 유의.

8. str_trim(x) : 해당 문자열에서 좌우 양 끝의 공백을 제거한 문자열을 반환함. 마지막 인자로 side="left" 또는 side-"right"를 지정하여 해당 방향만 삭제할 수 있음.

11. API와 공공데이터포털

11.1. API

11.1.1. API

Definition 14 *API(Application Programming Interface)는 어떤 서버의 특정 부분에 접속해서 내부의 데이터와 서비스를 이용할 수 있게 하는 소프트웨어상의 도구를 말함.*

어떤 서버에서 제공하는 정보를 가지고 있는 API 서버가 존재하여, 이 API 서버에 접근하면 특정 데이터를 얻거나 서비스를 이용할 수 있음. API 서버에 key와 함께 request를 보내면 response하는 식으로 동작함.

공개된 API를 open API라고 함.

원래의 API는 응용프로그램-운영체제, 응용프로그램-응용프로그램, 응용프로그램-프레인워크 등의 관계에서의 기능제어를 위한 인터페이스를 의미하지만, 데이터분석 등에서는 이런 의미로도 사용됨.

11.1.2. SOAP vs. REST

API에는 크게 2가지 종류가 존재함. 즉, 접속하고 데이터를 받아오는 데에는 2가지 방법이 있는데 SOAP와 REST가 그것임.

1. SOAP(Simple Object Access Protocol)
-> 프로토콜임. 즉, 하나의 정해진 규약/표준임.
-> 기능 위주로 설계되어 구조화된 데이터를 전송함.
-> 데이터 포멧으로 XML만을 사용함.

-> WS-security와 SSL을 지원함.
-> 개발환경 세팅이 복잡함.

SOAP은 마이크로소프트가 하나의 통일된 규약으로서 만든 방식임. 규약에 맞춰 데이터가 구조화되어야 하고, 하나의 일관된 포멧(XML)을 사용해야 함. 이에 따라 보안이 튼튼함.

2. REST(Representational State Transfer)
-> 아키텍처 원칙 세트임(설계 지침).
-> 데이터 위주로 설계되었고, 기본적으로 데이터 접근을 위해 서버에 접근함.
-> 텍스트, HTML, XML, JSON 등 다양한 데이터 포멧 지원.
-> SSL과 HTTPS를 지원함.
-> HTTP 프로토콜 기반으로 웹에 최적화되어 있음. HTTP URL을 통해 request를 보내므로 개발환경 세팅이 필요없음.

REST는 오픈소스 진영에서 만든 방식임. 각 기관별로 스타일을 정해서 사용함. SOAP만큼은 아니지만 보안이 튼튼함. 개발자 입장에서는 REST가 편리하기에 대부분의 API는 REST임.

아래의 정리들은 REST를 기반으로 함.

11.1.3. request URL 구성

request URL은 protocol, domain name, path, 파라미터로 구성됨.

이때 path는 'API 목록'의 맨 위에 적혀 있는 경로임. path 작성 이후 ?를 작성하고 파라미터를 작성함. 각 파라미터는 &로 구분함. 파라미터로는 page, perPage, serviceKey를 작성함.

perPage는 페이지 당 항목의 개수이고, page는 해당 perPage를 기준으로 했을 때 가져올 page의 번호임.

```
https://api.odcloud.kr/api/apnmOrg/v2/list
?page=1&perPage=10&serviceKey=h%2BAU9SbW%2BxVJwxw0XYKQuWgXq%2BGzI%2FbTBq3ldTJEJLhDQ
aX%2Bvlkv3NmSBXljG%2BbRjyQ6QNVx6YP86x4jtE1eoQ%3D%3D
```

11.1.4. response 형식

XML, JSON, Yaml 등 다양한 포멧을 사용할 수 있지만, 가장 주요한 것은 json임. JSON이 XML보다 light 하고, Yaml은 비교적 등장한지 얼마 되지 않아서 JSON이 가장 많이 사용됨. Yaml은 JSON을 좀 더 읽기 쉽게 나타낸 것임.

JSON은 키-값 쌍으로 이뤄진 데이터 객체의 전달에 사용됨.

물론 포멧의 구조를 알 필요는 없음. R에서 패키지를 사용하면 알아서 읽어 줌.

11.2. 공공데이터포털

11.2.1. 공공데이터포털 사용법

공공데이터포털에서 데이터를 받아올 수 있음.

1. 활용 신청
데이터 활용 신청을 하면 개인별로 문자열로 이뤄진 인증키가 발급됨. 이때 인코딩 키와 디코딩 키가 발급되는데, R에서 소스코드 작성 시에 인코딩 키를 써야 될 수도 있고 디코딩 키를 써야 될 수도 있음. 둘 다 넣어보고 되는 걸로 쓰면 됨.

이 인증키는 토큰인데, 한 번 인증받으면 특정 기간 동안 다시 인증을 받지 않아도 되는 것을 말함.

2. 사용법 설명
공공데이터포털에서는 API에 대한 명세와 사용 설명서가 첨부되어 있음.

3. 포털에서 테스트하기
'개발계정 상세보기'에서 인코딩/디코딩 키를 가지고 request시에 어떤 값이 respond되는지를 테스트할 수 있음. '인증키 설정'에 들어가 인코딩/디코딩 키를 넣고(넣지 않거나 잘못된 값을 넣으면 데이터가 respond 되지 않음) OpenAPI 호출을 하면 자동 생성된 request URL과 respond된 데이터를 확인할 수 있음.

이때 코드가 200인 경우는 정상적으로 인증되어 데이터가 전송된 경우임. 나머지는 어떤 이유로 문제가 발생한 상황들임.

4. respond 데이터 자료형 확인하기
'개발계정 상세보기' 아래쪽의 'models'에서 respond 데이터의 각 변수 자료형을 확인할 수 있음.

11.2.2. 코드

문자열로 URL을 구성하고, fromJSON() 함수 하나만 사용해도 데이터를 전달받을 수 있음.

fromJSON(requestURL) : 해당 문자열(URL)을 request로 전송하고, 받은 respond를 JSON으로 변환하여 반환함. fromJSON()를 사용하려면 jsonlite(json으로의 변환)와 httr(데이터 송수신) 패키지를 사용해야 함.

예시. 물론 paste0() 대신 str_c()를 사용해도 됨. install.packages()로 코드에서 패키지 설치가 가능함.

```
serviceURL <- "https://api.odcloud.kr/api/"
operation <- "apnmOrg/v1/list"
page <- "?page=1"
perPage <- "&perPage=10"
serviceKey <- "&serviceKey=zBdQqRglMM~~~~~~~~~~"

requestUrl = paste0(serviceURL, operation, page, perPage, serviceKey)

install.packages("jsonlite")
library(jsonlite)
install.packages("httr")
library(httr)

repos <- fromJSON(requestUrl)
repos <- data.frame(repos)

str(repos)
names(repos)
write.csv(repos,"data.csv",row.names = TRUE)
```

데이터를 가공하는 것을 파싱이라고 하는데, 여기서는 JSON으로의 1차 파싱, 데이터프레임으로의 2차 파싱이 적용되었음.

12. 군집화

12.1. 군집화

12.1.1. 군집화

Definition 15 *군집화란 주어진 데이터 집합에서 유사성이 높은 데이터끼리 묶는 것을 말함.*

특히 머신러닝에서 군집화는 가장 대표적인 기술 중 하나임. 군집화를 해 놓고 새로운 데이터가 들어왔을 때 어떤 그룹에 속하는지를 판단하는 것 또한 주요한 과제임.

12.1.2. k-평균 군집화

k-평균 군집화는 군집화의 한 종류로, 데이터를 k개의 군집으로 나누는 알고리즘임. 아래의 과정을 거쳐 군집화함.

1. 산점도 상에서 임의의 위치의 k개의 점을 찍음.
2. 각 데이터와 각 점의 거리를 계산해서 각 데이터가 가장 가까운 점의 군집에 속하는 것으로 판정함.
3. 각 점에 대해서 형성된 군집과 기존 점의 위치로 중심점을 계산하여 위치를 갱신.

4. 2~3의 과정을 점의 위치가 변하지 않을 때까지 반복.
5. 군집 확정.

12.1.3. R에서의 k-평균 군집화

1. kmeans()
R에서는 kmeans() 함수 하나만 사용하면 k-평균 군집화를 알아서 해줌.

kmeans(x=mydata, centers=3) : 데이터셋에 대해 k-평균 군집화를 수행하여 결과를 list로 반환함. centers에 k를 지정함.

kmeans()가 반환하는 list에는 다양한 데이터가 있지만, 주요한 것은 cluster와 center임. cluster에는 각 행이 몇 번 군집에 포함되는지가 저장되어 있고, centers에는 군집 별 중심점의 좌표가 저장되어 있음.

```
> fit$cluster
  [1] 1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 1 1
 [34] 1 2 1 1 1 2 1 1 2 2 1 1 2 1 2 1 1 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3
 [67] 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 2 3 3 3 3 2
[100] 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 3 3 3 3
[133] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

> fit$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.175758    3.624242     1.472727   0.2727273
2     4.738095    2.904762     1.790476   0.3523810
3     6.314583    2.895833     4.973958   1.7031250
```

2. clusplot()
군집의 시각화는 clusplot()으로 할 수 있음. cluster 패키지를 사용해야 함.

clusplot(mydata, fit$cluster) : 첫 번째 인자로 원본 데이터셋을, 두 번째 인자로 각 행 별 군집을 나타낸 데이터(cluster)를 작성해 군집을 시각화할 수 있음.

3. 특정 군집 추출
subset()으로 특정 군집을 추출할 수 있음.

예시.

```
x <- subset(x, fit$cluster == 2)
```

12.1.4. 표준화

각 변수의 특성에 따라 거리 계산에 있어 어떤 변수는 영향이 크고 어떤 변수는 영향이 작을 수 있음(ex. 키, 시력). 이에 따라 모든 변수가 거리 계산에 있어 동등하게 영향을 가지게 하기 위해 변수의 값 범위를 0과 1 사이로 표준화함.

어떤 변수 A의 값을 표준화할 때는 (x - min(A))/(max(A) - min(A))의 수식으로 표준화할 수 있음. x가 max(A)보단 항상 작거나 같고, min(A)보다 크거나 같으므로 항상 [0,1]에서 값을 가지게 됨.

코드로 나타내면 아래와 같음.

```
std <- function(x){
    return ((x - min(x)) / (max(x) - min(x)))
}

mydata <- apply(mydata, 2, std)     # 데이터 표준화
```

13. 회귀분석

13.1. 회귀분석

13.1.1. 회귀분석

1. 독립변수와 종속변수
독립변수 : 종속변수의 값을 결정하는 변수
종속변수 : 독립변수에 의해 값이 결정되는 변수

2. 예측모델과 회귀분석
예측모델 : 독립변수와 종속변수의 관계를 분석하여 이를 예측에 활용하는 통계적 방법으로 정리한 것.
회귀분석 : 회귀 이론을 기반으로 독립변수와 종속변수의 관계를 파악하여 예측모델을 도출하는 통계적 방법.
독립변수가 하나인 것은 단순회귀, 독립변수가 2개 이상인 것은 다중회귀라고 함.

회귀분석에서는 기존의 데이터로 회귀선을 그리고, 새로운 데이터가 들어올 위치를 예측함.

13.1.2. 단순선형 회귀분석

1. 단순선형 회귀식
단순선형 회귀식(회귀모델)은 $y = b + Wx$(W,b는 상수)꼴의 1차식 형태를 가짐.

2. lm()
실제로 회귀식을 구하는 것은 복잡하지만, R에서는 lm()함수가 이 작업을 수행해줌.

lm(a∼b, data=x) : a에 종속변수, b에 독립변수를 지정하고, x에 데이터셋을 지정하면 회귀모델에 대한 데이터를 list로 반환함. 변수명으로는 열 이름을 작성함.
coef(model) : 회귀모델을 지정하면 해당 회귀식의 계수(coefficient)를 벡터로 반환함. 계수는 상수 쪽(b,W)부터 들어 있음.
abline(model) : 산점도에 회귀선을 그림.

예시. 물론 이렇게 회귀식으로 구한 dist는 실제 값과 약간의 오차가 있음.

```
plot(dist~speed, data=cars)     # speed에 대해 dist를 나타낸 산점도
model <- lm(dist~speed, data=cars)
abline(model)

b <- coef(model)[1]
W <- coef(model)[2]             # (speed * w + b)로 해당 dist를 구할 수 있음
```

13.1.3. 다중선형 회귀분석

1. 다중선형 회귀식
다중선형 회귀식(회귀모델)은 $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$꼴의 1차식 형태를 가짐.

2. lm()
다중선형 회귀식도 lm()으로 구함. lm(a ∼ b1 + b2 + b3, data=x) 꼴로 작성함. 앞이 종속변수, 뒤가 독립변수들.

summery(model)로 해당 회귀모델에 대한 정보를 확인할 수 있음. 이때 Coefficient 부분을 보면 각 독립변수의 종속변수에 대한 중요도(***, **, *, ., 공백)를 확인할 수 있음. *가 많을수록 더 중요하다는 것. 또한 p-value는 해당 회귀모델이 의미 있는 모델인지를 나타내는데, 숫자가 작을수록 의미있는 것임. R-squared는 모델의 설명력(0∼1)을 나타냄.

3. 독립변수 선별
중요도가 낮은 독립변수는 분석에서 제외하는 것이 좋음. R에서는 setpAIC()로 제외 가능. MASS 패키지를 사용함.

setpAIC(model) : 해당 회귀모델에서 중요도가 낮은 독립변수를 제거한 회귀모델을 반환함.

13.1.4. 로지스틱 회귀분석

1. 로지스틱 회귀분석
지금까지는 종속변수가 숫자형인 데이터에 대해 회귀분석을 했는데, 로지스틱 회귀분석은 종속변수가 범주형인 데이터에 대한 회귀분석임.

2. glm()
범주형 데이터를 정수형으로 변환하고, glm()으로 로지스틱 회귀분석을 할 수 있음.

glm(a∼b1 + b2 + b3, data=x) : lm()과 사용법 동일. 이때 a에는 범주형 데이터만 정수형으로 변환한 데이터셋을 작성함.

예시.

```
x <- iris
x$Species <- as.integer(x$Species)  # 범주형 데이터가 각각 1, 2, 3 등으로 변환됨
model <- glm(Species ~ ., data=x)   # .으로 열 전부를 지정할 수 있음
```

3. 예측
로지스틱 회귀모델로 행 단위로 구성된 데이터에 대해 예측을 수행할 수 있음.

predict(model, unknown) : 해당 회귀모델로 해당 데이터에 대해 행 단위로 예측을 수행하여 그 결과를 소수의 집합으로 반환함. 즉, 범주형 데이터를 정수로 변환했을 때, 해당 소수와 가장 가까운 정수가 예측되는 데이터의 범주인 것.

예시. 아래와 같이 round()와 levels()로 소수를 반올림하고 Species의 level(펙터)에 의한 정수를 확인할 수 있음.

```
unknown <- data.frame(rbind(c(5.1, 3.5, 1.4, 0.2))) # 데이터 생성
unknown2 <- iris[,-5]       # 데이터 개수가 많아져도 동일하게 처리 가능
colnames(unknown) <- colnames(iris)[1:4]            # 이름이 동일해야 수행됨

pre1 <- predict(model, unknown)
pre1 <- round(pre, 0)                # 소수 첫째자리에서 반올림
pre2 <- predict(model, unknown2)
pre2 <- round(pre2, 0)

levels(iris$Species)[pre1]
levels(iris$Species)[pre2]
```