Ch3_environment_scoping

```r
# 환경(Environment)
library(pryr)
parenvs(all = TRUE)

library(dplyr)

search()


# 환경 트리 구조 확인
parenvs(all=TRUE)

parent.env(globalenv())

as.environment("package:dplyr")


# 함수 범위(Scope) 및 실행 환경(Environment)
nums = 1:5

## 버전 1: 함수 내부에서 nums를 새로 정의
add_numbers = function()
{
  nums = 1:10
  sum_nums = sum(nums)
  return(list(value = sum_nums,
              runtime_env = environment(),
              parent_env = parent.env(environment()),
              objects_env = ls.str(environment())
  ))
}
add_numbers()

environment(add_numbers)
print(nums)



## 버전 2: 전역 환경의 nums를 사용하는 함수
add_numbers_2 = function()
{
  sum_nums = sum(nums)
  return(sum_nums)
}
add_numbers_2()



## 버전 3: 인자로 받은 nums를 수정하여 반환
nums = 1:5
modify_nums = function(nums)
```

```
{
  nums = nums + 1
  return(nums)
}
nums = modify_nums(nums = nums)
nums
```

Ch4_numerical_analysis

```
# 정렬 알고리즘 1
selection_sort = function(vec)
{
  ## 길이가 n인 입력 벡터
  n = length(vec)

  for (i in 1:(n-1))
  {
    min_index = i
    for (j in (i+1):n)
    {
      if (vec[j] < vec[min_index])
      {
        min_index = j
      }
    }
    if (min_index != i)
    {
      temp = vec[i]
      vec[i] = vec[min_index]
      vec[min_index] = temp
    }
  }
  return(vec)
}

vec = c(6, 9, 2, 12, 7, 4)
sorted_vec = selection_sort(vec)
print(sorted_vec)


set.seed(123)
vec = runif(10)
print(vec)
print(selection_sort(vec))



vec = 3
sorted_vec = selection_sort(vec) ## 에러 발생



# 정렬 알고리즘 2
selection_sort = function(vec)
{
  ## 길이가 n인 입력 벡터
  n = length(vec)

  # 입력 벡터 길이가 1인 경우
  if (n < 2) return(vec)
```

```
  for (i in 1:(n-1)) {
    min_index = i
    for (j in (i+1):n) {
      if (vec[j] < vec[min_index]) {
        min_index = j
      }
    }
    if (min_index != i) {
      temp = vec[i]
      vec[i] = vec[min_index]
      vec[min_index] = temp
    }
  }
  return(vec)
}


set.seed(123)
vec = 3
print(selection_sort(vec))


# 가우스 소거법
gaussian_elimination = function(A, b)
{
  n = nrow(A)

  ## 전진 소거
  for (k in 1:(n-1))
  {
    for (i in (k+1):n)
    {
      factor = A[i, k] / A[k, k]
      A[i, k:n] = A[i, k:n] - factor * A[k, k:n]
      b[i] = b[i] - factor * b[k]
    }
  }

  ## 후진 대입
  x = numeric(n)
  for (i in n:1)
  {
    if (i < n)
      sum_ax = sum(A[i, (i+1):n] * x[(i+1):n])
    else
      sum_ax = 0 ## 마지막 변수일 경우 sum_ax는 0
    x[i] = (b[i] - sum_ax) / A[i, i]
  }
  return(x)
}


A = matrix(c(3, 1, 2,
```

```
              2, -3, 4,
              1, 2, -2),
            nrow = 3, byrow = TRUE)
b = c(5, -3, 6)
solution = gaussian_elimination(A, b)
print(solution)
```

Ch5_examples

```r
source("Ch5_modules.R")

# B-스플라인 기저 함수 시각화
## 3차 B-스플라인
degree = 3

## 예시 매듭점 (경계 반복 포함)
tiny = 1e-5
knots = c(rep(0 - tiny, degree), 0, 1, 2, 3, 4, rep(4 + tiny, degree))

## 입력 변수 x 생성
x_values = seq(0, 4, length.out = 200)

## 설계 행렬 생성
design_matrix = create_design_matrix(x_values, knots, degree)

## B-스플라인 기저 함수 시각화
matplot(x_values, design_matrix, type = "l", lty = 1,
        col = rainbow(ncol(design_matrix)),
        xlab = "x", ylab = "B-spline value",
        main = "Cubic B-splines")




# 스플라인 함수 시각화 예제
library(ggplot2)
set.seed(1)

## 임의의 계수 (-2에서 2 사이)
coefficients = runif(ncol(design_matrix), -2, 2)

## 스플라인 함수 계산 (기저 함수와 계수의 선형 결합)
spline_values = design_matrix %*% coefficients

## 데이터 프레임으로 변환
spline_data = data.frame(x = x_values, y = spline_values)

## 3차 B-스플라인 함수 시각화
ggplot(spline_data, aes(x = x, y = y)) +
  geom_line(color = "blue", linewidth = 1.2) +
  labs(title = "Cubic B-spline Function",
       x = "x", y = "Spline Value") +
  theme_bw()




#  스플라인 회귀 적합 예제
```

```r
set.seed(123)
n = 30
x_values = sort(runif(n, 0, 1))
y_values = sin(2 * pi * x_values) + cos(4 * pi * x_values) + rnorm(n, sd = 0.2)

degree = 3
num_knots = 5
tiny = 1e-5
knots = c(rep(0 - tiny, degree), seq(0, 1, length.out = num_knots), rep(1 + tiny, degree))

model = fit_spline(x_values, y_values, knots, degree)
grid_x = seq(min(x_values), max(x_values), length.out = 100)
plot_spline(x_values, y_values, model, grid_x)
```

Ch5_modules

```r
# B-스플라인 기저 함수 값을 계산하는 함수
# b_spline(x, knots, degree, i)
# - x:입력 데이터(스칼라 값)
# - knots: B-스플라인 매듭점(벡터)
# - degree: B-스플라인의 차수(정수)
# - i:해당 기저 함수의 인덱스(정수)
# - 반환값:해당x에서의B-스플라인 기저 함수 값(스칼라 값)
b_spline = function(x, knots, degree, i) {
  # 0차 B-스플라인 기저함수의 경우 (구간별 상수 함수)
  if (degree == 0) {
    return(ifelse(knots[i] <= x & x < knots[i + 1], 1, 0))
  }
  # 재귀적인 경우 (차수 d > 0)
  B_i_d1 = b_spline(x, knots, degree - 1, i)
  B_i1_d1 = b_spline(x, knots, degree - 1, i + 1)
  denom1 = knots[i + degree] - knots[i]
  denom2 = knots[i + degree + 1] - knots[i + 1]
  term1 = if (denom1 == 0) 0 else
    ((x - knots[i]) / denom1) * B_i_d1
  term2 = if (denom2 == 0) 0 else
    ((knots[i + degree + 1] - x) / denom2) * B_i1_d1
  return(term1 + term2)
}




# 데이터를 기반으로 설계 행렬을 생성하는 함수
# create_design_matrix(x_values, knots, degree)
# - x_values: 입력 데이터 (벡터)
# - knots: B-스플라인 매듭점 (벡터)
# - degree: B-스플라인의 차수 (정수)
# - 반환값: B-스플라인 기저 함수로 구성된 설계 행렬 (행렬)
create_design_matrix = function(x_values, knots, degree)
{
```

```
  n = length(x_values) # 샘플의 갯수
  num_basis = length(knots) - degree - 1 # 기저 함수의 갯수
  design_matrix = matrix(0, nrow = n, ncol = num_basis) # 설계 행렬 초기화
  for (j in 1:num_basis)
    for (i in 1:n)
      design_matrix[i, j] = b_spline(x_values[i], knots, degree, j)
  return(design_matrix)
}




# 최소제곱 추정법을 적용하여 회귀 계수를 계산하는 함수
# fit_spline(x_values, y_values, knots, degree)
# - x_values: 입력 데이터 (벡터)
# - y_values: 목표 값 (벡터)
# - knots: B-스플라인 매듭점 (벡터)
# - degree: B-스플라인의 차수 (정수)
# - 반환값: 회귀 계수 및 모델 정보 (리스트)
fit_spline = function(x_values, y_values, knots, degree)
{
  G = create_design_matrix(x_values, knots, degree)
  beta = solve(t(G) %*% G) %*% t(G) %*% y_values
  return(list(beta = beta, knots = knots, degree = degree))
}




# 새로운 데이터에 대한 예측을 수행하는 함수
# predict_spline(model, new_x)
# - model: fit_spline()으로 학습된 스플라인 모델 (리스트)
# - new_x: 예측을 수행할 x 값 (벡터)
# - 반환값: 예측된 y 값 (벡터)
predict_spline = function(model, new_x)
{
  G_new = create_design_matrix(new_x, model$knots, model$degree)
  y_pred = G_new %*% model$beta
  return(y_pred)
}




# 원래 데이터와 적합된 스플라인 회귀 곡선을 시각화하는 함수
# plot_spline(x_values, y_values, model, grid_x)
# - x_values: 원본 데이터 (벡터)
# - y_values: 원본 데이터의 목표 값 (벡터)
# - model: fit_spline()으로 학습된 스플라인 모델 (리스트)
# - grid_x: 예측할 x 값의 범위 (벡터)
```

```r
# - 반환값: ggplot을 이용한 스플라인 회귀곡선 시각화 (플롯)
plot_spline = function(x_values, y_values, model, grid_x)
{
  y_pred = predict_spline(model, grid_x)
  data_plot = data.frame(x = x_values, y = y_values)
  spline_plot = data.frame(x = grid_x, y = y_pred)
  ggplot() +
    geom_point(data = data_plot, aes(x, y), color = "black") +
    geom_line(data = spline_plot, aes(x, y), color = "blue", linewidth = 1) +
    labs(title = "Fitted B-spline Regression", x = "x", y = "y") +
    xlim(c(min(x_values), max(x_values))) +
    theme_minimal()
}
```

Ch6_OOP

```r
# 클래스와 속성
num_vec = 1:5
class(num_vec) = "new_class"
print(num_vec)
class(num_vec)


num_vec2 = structure(1:5, class = "new_class")
print(num_vec2)


df = data.frame(a = 1:4, b = 5:8)
class(df)
attributes(df)
names(df)



# 제너릭 함수
print
methods(print)[1:20]
print.data.frame(df)
print.data.frame


x = 1:5
class(x)
plot(x)


lm_fit = lm(log(mpg) ~ log(disp), data = mtcars)
class(lm_fit)
plot(lm_fit)



# 제너릭 함수와 메서드의 구현
my_function = function(x) UseMethod("my_function")

my_function.default = function(x)
{
  cat("일반 객체입니다:", x, "\n")
}

my_function.person_info = function(x)
{
  cat("이름:", x$name, "나이:", x$age, "\n")
}


p = list(name = "Kwan-Young Bak", age = 35)
class(p) = "person_info"
```

```r
my_function(42)
my_function(p)



# 스플라인 기저 함수 플랏 메서드
## B-스플라인 기저 함수 값을 계산하는 함수
b_spline = function(x, knots, degree, i) {
  if (degree == 0) {
    return(ifelse(knots[i] <= x & x < knots[i + 1], 1, 0))
  }

  B_i_d1 = b_spline(x, knots, degree - 1, i)
  B_i1_d1 = b_spline(x, knots, degree - 1, i + 1)

  denom1 = knots[i + degree] - knots[i]
  denom2 = knots[i + degree + 1] - knots[i + 1]

  term1 = if (denom1 == 0) 0 else
    ((x - knots[i]) / denom1) * B_i_d1
  term2 = if (denom2 == 0) 0 else
    ((knots[i + degree + 1] - x) / denom2) * B_i1_d1

  return(term1 + term2)
}


## 데이터를 기반으로 설계 행렬을 생성하는 함수 (basis 클래스 부여)
create_design_matrix = function(x_values, knots, degree) {
  n = length(x_values) # 데이터 포인트 갯수
  num_basis = length(knots) - degree - 1 # 기저 함수 갯수
  design_matrix = matrix(0, nrow = n, ncol = num_basis) # 설계 행렬 초기화

  for (j in 1:num_basis)
    for (i in 1:n)
      design_matrix[i, j] = b_spline(x_values[i], knots, degree, j)

  attr(design_matrix, "x_values") = x_values # 벡터 x 값 저장
  attr(design_matrix, "knots") = knots # 매듭점 시퀀스 저장
  attr(design_matrix, "degree") = degree # 스플라인 차수 저장
  class(design_matrix) = "basis" # "basis" 클래스 부여

  return(design_matrix)
}


## 원래 데이터와 적합된 스플라인 회귀 곡선을 시각화하는 함수 (S3 메서드 이용)
plot.basis = function(basis_obj, ...) {
  x_values = attr(basis_obj, "x_values")

  if (is.null(x_values))
    stop("x_values attribute is missing in the basis object.")
```

```
  matplot(x_values, basis_obj, type = "l", lty = 1,
          col = rainbow(ncol(basis_obj)),
          xlab = "x", ylab = "B-spline value", ...)
}



## 스플라인 회귀 적합을 위한 데이터 생성
degree = 3
tiny = 1e-5
knots = c(rep(0 - tiny, degree), 0, 1, 2, 3, 4, rep(4 + tiny, degree))
x_values = seq(0, 4, length.out = 200)


## basis 오브젝트 생성
basis_obj = create_design_matrix(x_values, knots, degree)
attributes(basis_obj)$knots
class(basis_obj)

## S3 메서드를 이용한 시각화
plot(basis_obj)
```

Ch7_debug

```r
# 예제 함수
## 로지스틱 곡선 생성 함수
logistic = function(a)
{
  return(1 / (1 + exp(-a)))
}


## 가중 최소제곱법
wls = function(X, y, W = NULL)
{
  if (is.null(W))
    W = diag(1, length(y))
  return(solve(t(X) %*% W %*% X, t(X) %*% W %*% y))
}


## IRLS 알고리즘
glm_irls = function(X, y, max_iter = 1000, epsilon = 1e-8)
{
  p = ncol(X)
  beta_hat = beta_hat_old = rep(1, p)
  diff = 1
  iter = 1

  while((diff > epsilon) & (iter < max_iter))
  {
    iter = iter + 1
    beta_hat_old = beta_hat

    y_hat = logistic(X %*% beta_hat)
    y_hat_prod = y_hat * (1 - y_hat)
    W = diag(as.vector(y_hat_prod))
    z = (y - y_hat) / y_hat_prod

    beta_hat = beta_hat + wls(X, z, W)
    diff = sum((beta_hat - beta_hat_old)^2)
  }
  return(beta_hat)
}


## 데이터 생성
set.seed(923)
n = 1000
x1 = runif(n)
x2 = runif(n)
X = matrix(c(rep(1, n), x1, x2), nrow = n)
beta = c(1, 3, -2)
y = rbinom(n, size = 1, prob = logistic(X %*% beta))

## IRLS 추정치
```

```r
beta_est = glm_irls(X, y)
as.vector(beta_est)

## glm() 함수를 이용한 적합
glm(y ~ x1 + x2, family = binomial())$coefficients



# traceback()을 사용한 디버깅
wls = function(X, y, W = NULL)
{
  if (is.null(W))
    W = diag(1, length(y))

  return(solve(t(X) %*% Wt %*% X, t(X) %*% Wt %*% y))
}


beta_est = glm_irls(X, y)
traceback()


# browser()를 사용한 디버깅
wls = function(X, y, W = NULL)
{
  if (is.null(W))
    W = diag(1, length(y))

  return(solve(t(X) %*% W %*% X, t(X) %*% W %*% y))
}


glm_irls = function(X, y, max_iter = 1000, epsilon = 1e-8)
{
  p = ncol(X)
  beta_hat = beta_hat_old = rep(1, p)

  browser() # 디버깅 모드 진입

  diff = 1
  iter = 1

  while((diff > epsilon) & (iter < max_iter))
  {
    iter = iter + 1
    beta_hat_old = beta_hat

    y_hat = logistic(X %*% beta_hat)
    y_hat_prod = y_hat * (1 - y_hat)
    W = diag(as.vector(y_hat_prod))
    z = (y - y_hat) / y_hat_prod

    beta_hat = beta_hat + wls(X, z, W)
```

```
    diff = sum((beta_hat - beta_hat_old)^2)
  }
  return(beta_hat)
}
glm_irls(X, y, max_iter = 2)



# debug()와 debugonce()를 사용한 디버깅
glm_irls = function(X, y, max_iter = 1000, epsilon = 1e-8)
{
  p = ncol(X)
  beta_hat = beta_hat_old = rep(1, p)

  diff = 1
  iter = 1

  while((diff > epsilon) & (iter < max_iter))
  {
    iter = iter + 1
    beta_hat_old = beta_hat

    y_hat = logistic(X %*% beta_hat)
    y_hat_prod = y_hat * (1 - y_hat)
    W = diag(as.vector(y_hat_prod))
    z = (y - y_hat) / y_hat_prod

    beta_hat = beta_hat + wls(X, z, W)
    diff = sum((beta_hat - beta_hat_old)^2)
  }
  return(beta_hat)
}

debug(glm_irls)
glm_irls(X, y, max_iter = 2)
undebug(glm_irls)



debugonce(glm_irls)
glm_irls(X, y, max_iter = 2) # 디버깅 모드 진입
glm_irls(X, y, max_iter = 2) # 디버깅 과정 없이 코드 실행
```

Ch8_python

```r
library(reticulate)

#use_python("path") : 원하는 Python 경로 지정
#use_virtualenv("myenv") : Python 가상 환경을 사용
#use_condaenv("myenv") : Conda 가상 환경을 사용

use_virtualenv("r-reticulate")
py_install("numpy")
np = import("numpy")
np$arange(10)


# R과 Python을 연동하는 머신러닝 예제
## 데이터 불러오기
data("mtcars")
head(mtcars, 10)



## lm() 함수로 선형 회귀 적합
lm(mpg ~ ., data = mtcars)$coefficients



##  Python의 sklearn.linear_model 모듈을 불러와서 R에서 적합
### 모듈 불러오기
skl_lr = import("sklearn.linear_model")

### 설명 변수와 반응 변수 설정
x = as.matrix(mtcars[, -1])
y = as.numeric(mtcars[, 1])

### 파이썬 객체(numpy 배열)로 변환
np = import("numpy")
pyx = np$array(x)
pyy = np$array(y)

### 선형 회귀 모델 생성
lr = skl_lr$LinearRegression()
lr$fit(pyx, pyy)

### 계수와 절편 저장 및 출력
coefficients = lr$coef_
intercept = lr$intercept_
coefficients
cat("Intercept: ", intercept, "\n")


## py_run_string() 함수를 사용해 python 코드 직접 실행
### 설명 변수와 반응 변수 설정
x = as.matrix(mtcars[, -1])
```

```
y = as.numeric(mtcars[, 1])

### 파이썬 객체(numpy 배열)로 변환
np = import("numpy")
pyx = np$array(x)
pyy = np$array(y)

### 파이썬 코드 문자열 생성
python_code = "
from sklearn.linear_model import LinearRegression

# 데이터 준비
X = r.pyx
y = r.pyy

# 모델 생성 및 학습
lr = LinearRegression()
lr.fit(X, y)

# 계수와 절편 추출
coefficients = lr.coef_
intercept = lr.intercept_

# 결과 출력
print('Coefficients:', coefficients)
print('Intercept:', intercept)
"

### 파이썬 코드 실행
py_run_string(python_code)
```

Ch8_rcpp

```
library(Rcpp)

# cppFunction()을 사용하여 C++ 코드 실행
cppFunction('
int add(int x, int y)
{
return x + y;
}')
add(2, 3)


# sourceCpp() 함수를 이용하여 .cpp 파일 실행
sourceCpp("my_code.cpp") ## 필요한 경우 .cpp 파일 경로 설정
multiply(3, 4)


# Rcpp 코드 디버깅
add(3, 4)
```

```r
# 유클리드 거리 함수 예제
## R 코드
euclidean_distance_r = function(mat)
{
  n = nrow(mat)
  dist_mat = matrix(0, n, n)

  for (i in 1:n)
    for (j in 1:n)
      dist_mat[i, j] = sqrt(sum((mat[i, ] - mat[j, ])^2))

  return(dist_mat)
}


## cpp 코드
cppFunction('
NumericMatrix euclidean_distance_cpp(NumericMatrix mat) {
int n = mat.nrow();
int d = mat.ncol();
NumericMatrix dist_mat(n, n);
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
double sum_sq = 0.0;
for (int k = 0; k < d; k++) {
sum_sq += pow(mat(i, k) - mat(j, k), 2);
}
dist_mat(i, j) = sqrt(sum_sq);
}
}
return dist_mat;
}
')

## 데이터 생성
set.seed(923)
mat = matrix(runif(300 * 10), nrow = 300, ncol = 10)


## 두 함수 비교
dist_mat1 = euclidean_distance_r(mat)
dist_mat2 = euclidean_distance_cpp(mat)

dist_mat1[1:5, 1:5]
dist_mat2[1:5, 1:5]




## microbenchmark()를 이용한 성능 평가 (계산 시간 비교)
library(microbenchmark)
benchmark_results = microbenchmark(
```

```
  R = euclidean_distance_r(mat),
  Rcpp = euclidean_distance_cpp(mat),
  times = 100
)
print(benchmark_results)
```

Ch14_R_basics

```r
#install.packages(c("ggplot2", "palmerpenguins"))

# 경로 관리
getwd()

# R 객체
## 벡터
### 수치형 벡터
odd = c(1, 3, 5, 7, 9)
odd
typeof(odd)


odd_L = c(1L, 3L, 5L, 7L, 9L)
odd_L
typeof(odd_L)


### 문자형 벡터
txt = c("Advanced", "R", "Book")
txt
typeof(txt)


### 논리형 벡터
3.14 > 3
logic = c(FALSE, TRUE)
logic
typeof(logic)



## 행렬
vec = c(1:10)
vec
mat = matrix(vec, nrow = 2, ncol = 5)
mat


mat1 = matrix(vec, nrow = 2, ncol = 5, byrow = TRUE)
mat1


## 배열
arr = array(1:30, dim = c(3, 5, 2))
arr



## 리스트
lst = list(name = "Alice",
```

```r
            age = 25,
            subject = c("math", "science", "english"),
            score = matrix(c(90, 85, 88, 75, 83, 90),
                           nrow = 2,
                           dimnames = list(c("Midterm", "Final"),
                                           c("math", "science", "english"))))
lst



## 데이터 프레임
df = data.frame(Name = c("Alice", "Bob", "Charlie"),
                Age = c(25, 30, 35),
                Score = c(90, 93, 88))
df

df$Age   # Age 열에 접근
df[[2]]  # 두 번째 열에 접근
df[1, ]  # 첫 번째 행에 접근
df[2, 1] # 두 번째 행, 첫 번째 열에 접근




# 반복문과 조건문
## 반복문 기초 문법
for (i in 1:5)
{
  print(i)
}



for (string in c("advanced", "R", "book", "study"))
{
  print(string)
}



i = 1
while (i <= 5)
{
  print(i)
  i = i + 1
}



num = -5
if (num > 0)
{
```

```
  print("positive")
} else if (num < 0)
{
  print("negative")
} else
{
  print("zero")
}


## 조건문 기초 문법
num = -5
if (num > 0)
{
  print("positive")
} else if (num < 0)
{
  print("negative")
} else
{
  print("zero")
}



## 함수 정의 및 호출

add_numbers = function(x, y) ### 함수 정의
{
  sum = x + y
  return(sum)
}
result = add_numbers(3, 5) ### 함수 호출
print(result)




greet = function(name = "사용자") ### 함수 정의
{
  return(paste("안녕하세요,", name, "!"))
}
greet() ### 함수 호출
greet("철수")
```