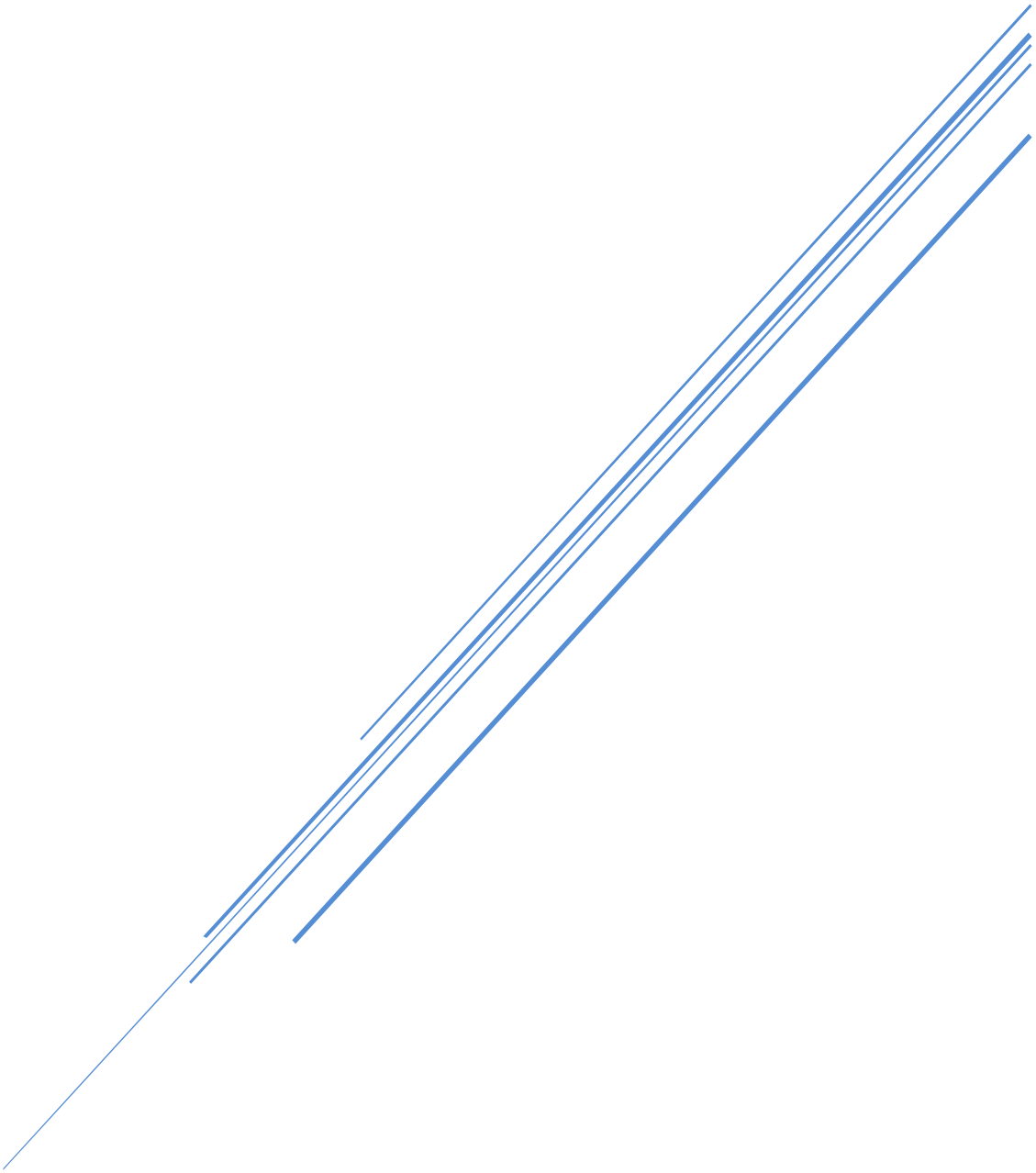
통계 기법을 활용한
데이터 분석 및 시각화 보고서



2022. 12. 02

B4 양정연, 신정훈, 김용현, 조성운

목차

개요

1) **USD/KRW 환율 데이터에 있는 일별 환율 데이터(2021 년 11 월 15 일 ~ 2022 년 11 월 14 일 대상)를 기반으로 회귀분석 또는 시계열분석을 이용하여 2022 년 12 월 우리나라 미국달러대비 원화 환율을 예측하고 시각화 하시오.**

환율 관련 여러 데이터 중 예측에 필요한 데이터(날짜, 종가)를 추출 후 시계열 데이터로 변경한다. ARIMA 모형으로 모델을 생성한 후 2022 년 원화 환율을 예측하고 시각화한다..

2) **위스콘신 유방암 데이터셋을 대상으로 분류기법 2 개를 적용하여 기법별 결과를 비교하고 시각화하시오.**

종속 변수를 Benign(양성)이면 0, Malignancy(악성)이면 1 로 전처리하는 과정이 필요하다. 전처리 후 데이터 셋을 생성하여 xgboost 기법과 랜덤포레스트 기법으로 분석을 진행하였다.

3) **mlbench 패키지 내 BostonHousing 데이터셋을 대상으로 예측기법 2 개를 적용하여 기법별 결과를 비교하고 시각화하시오.**

예측 기법으로 의사결정나무, 선형회귀를 선택하였으며 데이터 전처리 과정에서 의사결정나무는 가지치기 과정이, 선형회귀는 변수 간의 다중공선성 여부를 확인하는 과정이 필요하다.

4) **아래의 조건을 고려하여 군집분석을 실행하시오. (1) 데이터: ggplot2 패키지 내 diamonds 데이터 (2) philentropy::distance() 함수 내 다양한 거리 계산 방법 중 Euclidian 거리를 제외한 3 개를 이용하여 거리 계산 및 사용된 거리에 대한 설명 (3) 탐색적 목적의 계층적 군집분석 실행 (4) 군집수 결정 및 결정 사유 설명 (5) k-means clustering 실행 (6) 시각화 (7) 거리 계산 방법에 따른 결과 차이 비교**

diamonds 데이터의 경우 등간척도와 명목척도가 혼재되어 있으며 데이터 양이 많아 (53940 행) 샘플링을 포함한 데이터 전처리 과정이 필요하였다. 데이터는 1000 개를 임의 추출하였으며, seed 값으로 1234 를 부여하였다.

군집 분석의 경우 범위가 넓은 변수에 영향을 많이 받는 경향이 있어 scale() 함수를 통해 데이터 표준화 과정을 거쳤다. 이상치에도 영향을 많이 받으나 해당 데이터는 실존하는 다이아몬드의 데이터이므로 결측치를 제외한 이상치는 없는 것으로 간주하였다. 결측치 또한 실제로 존재하지 않았다.

dist()함수로 계산되는 manhattan, maximum, canberra 거리의 경우 명목척도를 제외한 데이터셋으로 계산하였다. 또한, distance() 함수로 계산된 결과값을 hclust()에 직접 입력 시 오류가 발생하여 as.dist() 함수를 통해 dist() 결과값으로 변환 후 사용하였다.

계산된 데이터를 기반으로 NbClust() 함수를 통해 최적 군집수를 확인하였고, 2 개의 군집으로 분류하는 것이 가장 적절할 것이라는 결론을 내렸다. 비계층적 군집 분석에 앞서 변수간 상관관계를 확인하였고, 가장 높은 상관계수를 가지는 price 와 carat 데이터로 군집 분석을 실시하였다.

5) (1) 제공된 데이터를 이용하여 토픽 분석을 실시하여 단어구름으로 시각화 하고 단어 출현 빈도수를 기반하여 어떤 단어들이 주요 단어인지 설명하시오.

전처리 기준과 불용어 처리 기준 판단 후 데이터 전처리를 수행했다. 화자가 강조하고자 하는 단어('너희들', '당신들' 등)는 유의어 처리를 하여 빈도수를 조정했다. 또한 지나치게 빈도수가 많거나('우리'라는 단어의 빈도수는 빈도수가 두 번째로 많은 단어 '우크라이나'의 2 배 이상) 적은 단어(빈도수 2 번 미만)는 판단에 따라 제외처리했다. 단어 전처리를 반복 수행 후 시각화한 단어 구름을 분석했다.

(2) 제공된 데이터를 이용하여 연관어 분석을 실시하여 연관어를 시각화 하고 시각화 결과에 대해 설명하시오.

줄 단위로 데이터를 읽어 전처리 후 트랜잭션을 생성했다. 트랜잭션을 이용하여 연관규칙을 발견했다. 이때, supp 와 conf 값은 연관규칙 생성 결과를 보며 조정했다. 연관규칙 결과를 Edgelist 로 시각화하여 결과에 대해 설명했다.

6) R 의 ggplot2 패키지 내 함수와 python 의 matplotlib 패키지 내 함수를 사용하여 막대 차트(가로, 세로), 누적막대 차트, 점 차트, 원형 차트, 상자 그래프, 히스토그램, 산점도, 중첩자료 시각화, 변수간의 비교 시각화, 밀도그래프를 수업자료 pdf 내 데이터를 이용하여 각각 시각화하고 비교하시오.

iris 데이터를 이용하여 문제에서 요구하는 시각화를 단계별로 진행했다. 기본적으로 Species 컬럼을 기준으로 한 변수별 시각화를 진행했으나, 막대그래프나 누적그래프 등 경우에 따라 각 변수의 평균값을 이용했다. R 에서의 원형 그래프 시각화에서는 변수별 범주화 후 원형 그래프를 생성하여 데이터에 대한 다양한 접근을 시도했다. 2 가지 변수가 각각 x, y 값으로 필요한 경우에는 Sepal.length 와 Sepal.Width, Petal.Length 와 Petal.Width 를 묶어 사용했다.

변수간 비교 항목에서는 더 효과적인 시각화를 위해 R 에서는 ggplot2 를 확장시켜주는 GGally 패키지, python 의 경우에는 matplotlib 을 기반으로 한 seaborn 패키지를 활용했다.

2. 환율 데이터 시계열 분석

2.1 데이터 정의

사용 데이터 : USD_KRW.csv

변수	변수정의
Data	2021/11/15~2022/11/14 데이터프레임
data_last	종가 데이터프레임
data_ts	시간에 따른 종가 기준 데이터프레임
Model	시간에 따른 종가 기준 ARIMA 모델링
Fore	2022/12/31 까지 예측 데이터

2.2 문제 풀이

2.2.1 변수 정의

데이터 분석에 앞서 목적에 부합하는 데이터를 선별하기 위해 원본 데이터를 살펴보았다. 시계열 분석을 이용해 2022 년 데이터를 예측하기 위해서 날짜와 종가 컬럼을 선별하였으며, 선별된 컬럼과 해당 컬럼의 정의는 아래 표와 같다.

컬럼명	의미
DATE	년-월-일 날짜
CLOSE	DATE 에 해당하는 날의 종가

원본에서 DATE 컬럼은 2021 년 11 월 15 일부터 오름차순 정렬되어 있어, 날짜 기준 내림차순 정렬 후 종가를 기준으로 하여 시계열 분석을 실시할 것이다.

2.2.2 시계열 모델 생성

1) library 호출

```
library(dplyr)
library(ggplot2)
library(forecast)
```

2) 데이터 불러오기

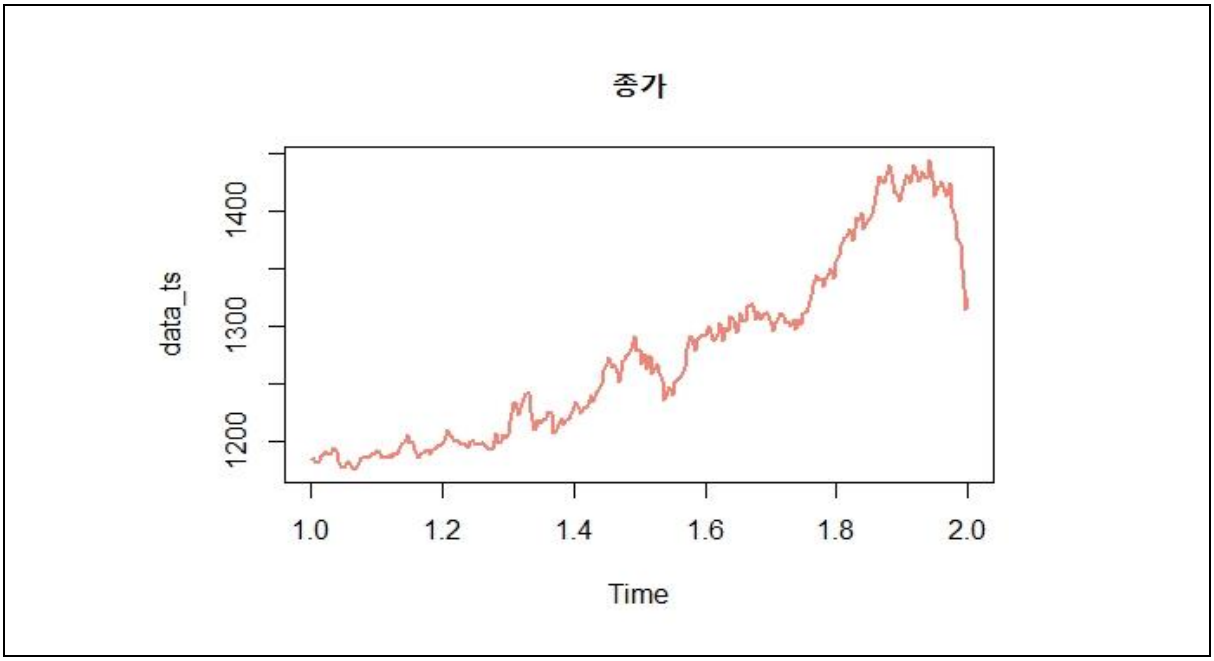
```
data <- read.csv("USD_KRW.csv")
str(data)
data<-data%>%arrange(data,as.Date(unlist(data[1])))
```

2021 년 11 월 15 일 ~ 2022 년 11 월 14 일 환율 데이터 날짜 데이터를 csv 파일로 다운받아 불러왔고, 날짜를 내림차순으로 재정렬 했다.

3) 시계열 데이터로 데이터 변경

```
data_last <- unlist(data[2])
data_last <- sub(pattern=",",replacement="",x=data_last)
data_last <- as.numeric(data_last)
data_ts <- ts(data=data_last,start = c(1), frequency = 260)
plot(data_ts, col= "salmon", lwd= 2, main = "종가")
```

계산을 위해 종가 데이터에 포함되어 있는 쉼표(,)를 제거하고 숫자형으로 변경한 후, ts 함수를 이용하여 종가를 시계열 데이터로 변환했다. 원본 데이터를 고려할 때, 종가는 주말을 제외하고 데이터가 추가되므로 총 데이터의 수는 260 개이다. 따라서 ts 함수의 frequency 를 260 으로 설정했다. 아래는 시계열 데이터 변환의 결과이다.



환율이 꾸준히 상승하다 오른쪽 끝 부분에서 급격히 하락하므로 추세를 가지는 데이터라는 것을 확인할 수 있다.

4) 모델 식별과 추정

```
arima <- auto.arima(data_ts)
arima
```

시계열 데이터를 이용하여 ARIMA 모델을 식별했다. 아래는 그 결과이다.

```
> arima
Series: data_ts
ARIMA(0,1,0)

sigma^2 = 60.03:  log likelihood = -901.24
AIC=1804.47   AICc=1804.49   BIC=1808.03
```

ARIMA(0,1,0)으로 랜덤워크 시계열로 확인 되었다.

5) 시계열 모형 생성

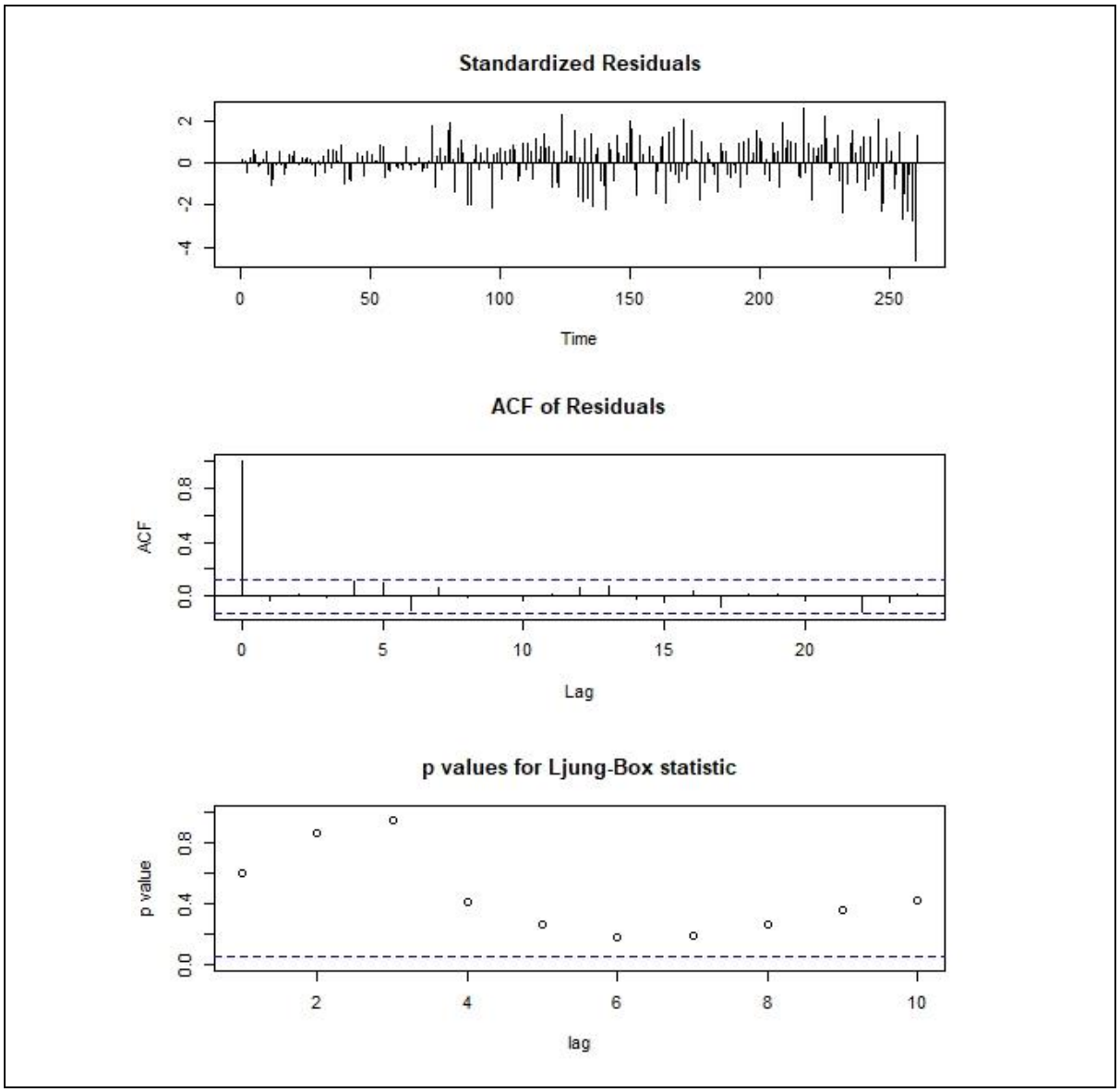
```
model <- arima(data_last, order = c(0, 1, 0))
model
```

4 번 과정의 결과에 따라 ARIMA 시계열 모델을 생성했다.

2.2.3 모델진단

1) tsdiag 함수를 이용한 잔차분석

```
tsdiag(model)
```



위 그림은 tdiag 함수로 잔차분석을 한 결과이며, 이를 통해 세 가지 기준으로 시계열 모델 진단을 할 수 있다. 첫 번째로, Standardized Residuals 에서는 모델이 선형관계이며 등분산성이 고루 분포되어 있다는 것을 확인할 수 있다. 두 번째로, ACF 잔차들 사이에

상관성이 확인되지 않는다. 마지막으로, P-value 값이 0.05 이상으로 분포 되어 있어 현재 ARIMA 모형은 매우 양호한 시계열 모형이라는 진단을 내릴 수 있다.

2) BOX-Ljung 에 의한 잔차항 모형 진단

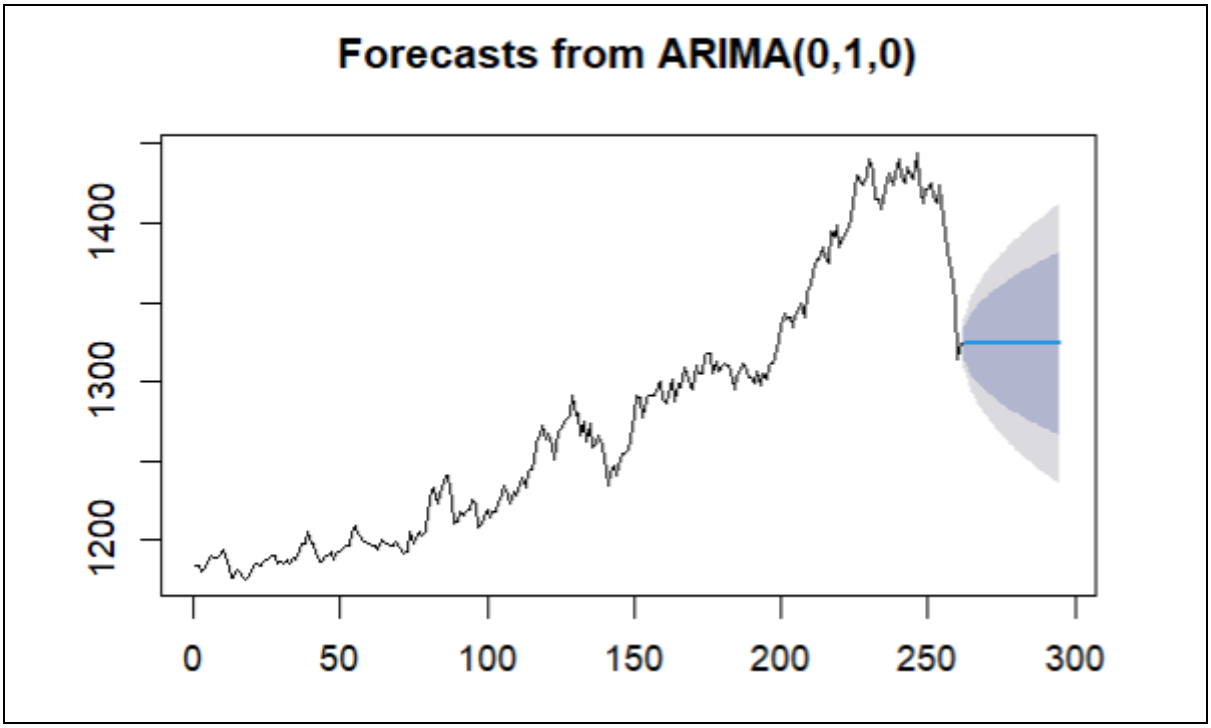
```
Box.test(model$residuals, lag = 1, type = "Ljung")
```

```
Box-Ljung test

data:  model$residuals
X-squared = 0.27245, df = 1, p-value = 0.6017
```

Box-Ljung 모형 진단 결과 p-value(0.6017)가 0.05 이상이므로 모델이 통계적으로 적절하다.

2.2.4 미래 예측

```
par(mfrow = c(1, 2))
fore <- forecast(model, h = 34)
plot(fore)
fore
```



Point	Forecast	Lo 80	Hi 80	Lo 95	Hi 95
262	1323.87	1313.941	1333.799	1308.685	1339.055
263	1323.87	1309.829	1337.911	1302.396	1345.344
264	1323.87	1306.673	1341.067	1297.570	1350.170
265	1323.87	1304.013	1343.727	1293.501	1354.239
266	1323.87	1301.669	1346.071	1289.916	1357.824
267	1323.87	1299.550	1348.190	1286.676	1361.064
268	1323.87	1297.601	1350.139	1283.696	1364.044
269	1323.87	1295.788	1351.952	1280.922	1366.818
270	1323.87	1294.084	1353.656	1278.316	1369.424
271	1323.87	1292.473	1355.267	1275.852	1371.888
272	1323.87	1290.940	1356.800	1273.509	1374.231
273	1323.87	1289.476	1358.264	1271.269	1376.471
274	1323.87	1288.072	1359.668	1269.121	1378.619
275	1323.87	1286.720	1361.020	1267.055	1380.685
276	1323.87	1285.417	1362.323	1265.061	1382.679
277	1323.87	1284.155	1363.585	1263.132	1384.608
278	1323.87	1282.933	1364.807	1261.263	1386.477
279	1323.87	1281.746	1365.994	1259.447	1388.293
280	1323.87	1280.592	1367.148	1257.682	1390.058
281	1323.87	1279.468	1368.272	1255.963	1391.777
282	1323.87	1278.371	1369.369	1254.286	1393.454
283	1323.87	1277.301	1370.439	1252.648	1395.092
284	1323.87	1276.254	1371.486	1251.048	1396.692
285	1323.87	1275.230	1372.510	1249.481	1398.259
286	1323.87	1274.227	1373.513	1247.947	1399.793
287	1323.87	1273.244	1374.496	1246.444	1401.296
288	1323.87	1272.279	1375.461	1244.969	1402.771
289	1323.87	1271.333	1376.407	1243.521	1404.219
290	1323.87	1270.403	1377.337	1242.099	1405.641
291	1323.87	1269.489	1378.251	1240.701	1407.039
292	1323.87	1268.590	1379.150	1239.326	1408.414
293	1323.87	1267.705	1380.035	1237.973	1409.767

294	1323.87	1266.834	1380.906	1236.642	1411.098
295	1323.87	1265.977	1381.763	1235.330	1412.410

앞의 표는 2022 년 11 월 15 일부터 12 월 30 일까지 주말을 제외한 34 일간의 예측 결과표이다. 예측 결과에 따르면, 12 월 30 일 의 종가는 80% 확률로 1265.977~ 1381.763 안에 들어올 것이고, 95% 확률로 1235.330 ~ 1412.410 안에 들어올 것이다.

2.3 결론

시계열 데이터 변환 후 plot 을 통해 2021 년 11 월부터 지속적으로 환율이 상승하는 추세가 있었으나 2022 년 11 월에 가까워지며 급격하게 환율이 떨어지는 것을 확인할 수 있었다.

예측 결과표를 보았을 때, 95%확률 기준 262(11 월 15 일)의 범위는 1308.685 ~ 1339.055 이고, 12 월 30 일의 환율 범위는 1237.973 ~ 1409.767 로 예측되어 시간이 흐를수록 예측 변동폭이 커짐을 확인할 수 있는데, 이는 기존 데이터인 2021 년 11 월부터의 변동폭이 크기 때문인 것으로 해석된다.

3. 위스콘신 유방암 데이터 분류분석

3.1 데이터 정의

사용 데이터: wisc_bc_data.csv

컬럼명	의미
id	환자 식별 번호
diagnosis	양성 여부 (M = 악성, B = 양성)
각 세포에 대한 정보	
radius	반경 (중심에서 외벽까지 거리들의 평균값)
texture	질감 (Gray-Scale 값들의 표준편차)
perimeter	둘레
area	면적
smoothness	매끄러움 (반경길이의 국소적 변화)
compactness	조그만 정도 (둘레^2/면적-1)
concavity	오목함 (윤곽의 오목한 부분의 정도)
points	오목한 점의 수
symmetry	대칭
dimension	프랙탈 차원("해안선 근사" - 1)
_mean	3~12 번까지는 평균값을 의미
_se	13~22 번까지는 표준오차를 의미
_worst	23~32 번까지는 각세포별 구분들에서 제일큰 3 개의 값의 평균값

3.2 xgboost

3.2.1 R 을 이용한 xgboost

1) library 호출

```
library(xgboost)
```

2) 데이터 불러오기 및 전처리

```
raw_wis <- read.csv("wisc_bc_data.csv", header = T)

wis_label <- ifelse(raw_wis$diagnosis == "B", 0,
                    ifelse(raw_wis$diagnosis == "M", 1, 2))

table(wis_label)
raw_wis$label <- wis_label
```

종속 변수인 diagnosis 를 Benign(양성)이면 0, Malignancy(악성)이면 1 로 변환하여 wis_label 에 담았다. 이후 wis_label 를 raw_wis$label 에 추가하였다.

3) xgboost 를 위한 데이터 전처리

```
set.seed(1234)
idx <- sample(nrow(raw_wis), 0.7 * nrow(raw_wis))
train <- raw_wis[idx, -1]
test <- raw_wis[-idx, -1]

train_mat <- as.matrix(train[-c(1, 32)])
dim(train_mat)

train_lab <- train$label
length(train_lab)

dtrain <- xgb.DMatrix(data = train_mat, label = train_lab)
```

데이터셋을 생성할 때 필요없는 id 를 빼고 생성하였다. Matrix 객체로 변환할 때 종속변수를 제거하고 matrix 를 생성하였다. 종속변수는 train_lab 에 따로 담았다. 생성한 matrix 로 xgb.DMatrix 를 실행하여 dtrain 에 담았다.

4) model 생성

```
xgb_model <- xgboost(data = dtrain, max_depth = 2, eta = 1,
                     nthread = 2, nrounds = 2, num_class = 2,
                     objective = "multi:softmax",
                     verbose = 0)
xgb_model
```

결과 :

iter	train_logloss
1	0.212576
2	0.113487

트리의 최대 깊이인 max.depth 2, 과적합을 방지하기 위해 업데이트에 사용되는 eta 단계 크기 축소는 1 로 XGBoost 를 실행하는 데 사용되는 병렬 스레드 수인 nthread 는 2, 최종 모델의 결정 트리 수 nrounds 는 2, "multi:softmax"는 다중 클래스 분류를 수행하도록 XGBoost 를 설정한다.

5) 분류모델 평가

```
test_mat <- as.matrix(test[-c(1, 32)])
dim(test_mat)
test_lab <- test$label
length(test_lab)

pred_wis <- predict(xgb_model, test_mat)
pred_wis
table(pred_wis, test_lab)
sum(pred_wis == test_lab)/ NROW(pred_wis)
```

결과 :

	test_lab	
pred_wis	0	1
0	110	11
1	22	48

분류모델 평가 결과, 92.39766%의 Accuracy 를 보여주고 있다.

6) model 의 중요 변수(feature)와 영향력 보기

```
table(pred_wis, test_lab)
importance_matrix <- xgb.importance(colnames(train_mat),
                                    model = xgb_model)
importance_matrix
```

결과:

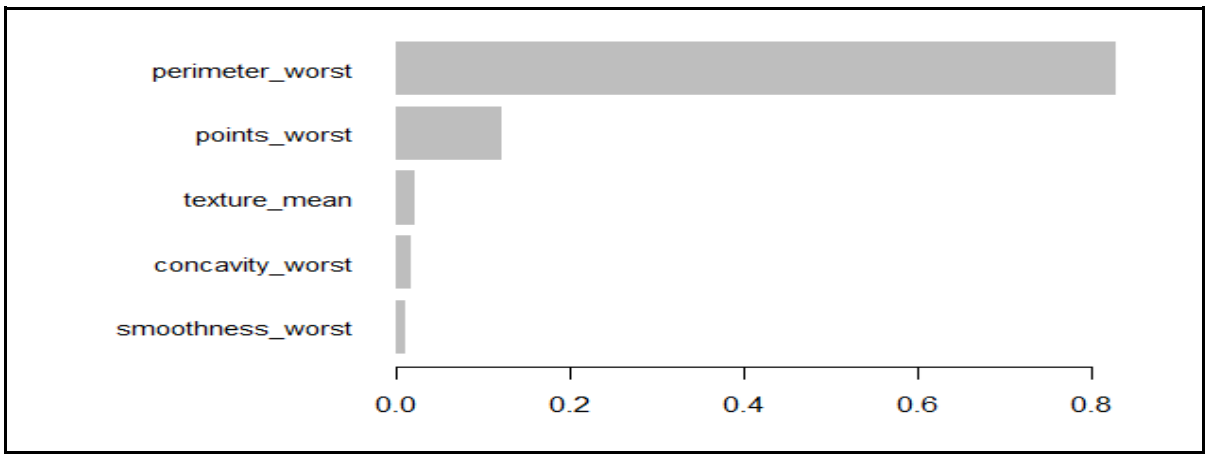
Feature	Gain	Cover	Frequency
1: perimeter_worst	0.82974383	0.50000000	0.3333333
2: points_worst	0.12080276	0.22835643	0.1666667
3: texture_mean	0.02076423	0.07057131	0.1666667
4: concavity_worst	0.01757083	0.10950922	0.1666667
5: smoothness_worst	0.01111835	0.09156305	0.1666667

perimeter_worst, points_worst, texture_mean, concavity_worst, smoothness_worst 가 중요 변수로 확인되었다.

7) 중요 변수 시각화

```
xgb.plot.importance(importance_matrix)
```

결과:



중요변수 시각화 결과, perimeter_worst 가 가장 큰 영향을 미치며 points_worst, texture_mean, concavity_worst, smoothness_worst 순으로 중요 변수라 할 수 있다.

3.2.2. python 을 이용한 xgboost

1) Import packages

```
import pandas as pd
import numpy as np
from  sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

import xgboost as xgb
from xgboost import plot_importance
from xgboost import plot_importance
```

2) 데이터 전처리

```
df= pd.read_csv('wisc_bc_data.csv')
df.head()

df = df.drop('id', axis=1)
df.head()
df['diagnosis'] = df['diagnosis'].apply(lambda x : 1 if x== "M" else 0)

x_train, x_test, y_train, y_test = train_test_split(df.iloc[:,1:],
df['diagnosis'],test_size=0.3,random_state=11)
```

데이터를 불러와 필요없는 'id'컬럼을 삭제한 후 Benign(양성)이면 0, Malignancy(악성)이면 1 로 변환하였다. 이후 독립변수는 x_train, x_test 에 종속 변수는 y_train, y_test 에 각각 데이터 셋을 생성하였다.

3) model 생성

```
dtrain = xgb.DMatrix(data=x_train, label=y_train)
dtest = xgb.DMatrix(data=x_test, label=y_test)
params = {'max_depth' : 2,'eta' : 1,  'objective' : 'multi:softmax',
'num_class' :2, 'eval_metric' : 'merror', 'early_stoppings' : 100}

xgb_model = xgb.train(params = params, dtrain = dtrain,
            num_boost_round = 400, early_stopping_rounds = 100,
            evals=[(dtrain,'train'),(dtest,'dtest')])
```

트리의 최대 깊이인 max.depth 2, 과적합을 방지하기 위해 업데이트에 사용되는 eta 단계 크기 축소는 1 로 train parameter 인 objective 는 softmax 로 지정했다. softmax 는 다중분류로 eval_metric 에서 다중분류의 error rate 를 나태내는 merror 로 설정했다. 과적합을 방지하기 위해 early_stoppings 는 100 으로 설정했다.

결과 :

Will train until dtest-merror hasn't improved in 100 rounds		
1	train-merror:0.030151	dtest-merror:0.064327
2	train-merror:0.025126	dtest-merror:0.05848
3	train-merror:0.015075	dtest-merror:0.05848
4	train-merror:0.01005	dtest-merror:0.046784
5	train-merror:0.007538	dtest-merror:0.05848
6	train-merror:0.002513	dtest-merror:0.046784
7	train-merror:0.002513	dtest-merror:0.046784
8	train-merror:0	dtest-merror:0.046784
9	train-merror:0	dtest-merror:0.052632

10	train-merror:0	dtest-merror:0.040936
11	train-merror:0	dtest-merror:0.035088
12	train-merror:0	dtest-merror:0.035088
13	train-merror:0	dtest-merror:0.035088
14	train-merror:0	dtest-merror:0.040936
15	train-merror:0	dtest-merror:0.040936

Stopping. Best iteration:	
11	train-merror:0 dtest-merror:0.035088

11 번째가 최적의 모델로 선택되었다.

4) 분류모델 평가

```
y_pred_probs = xgb_model.predict(dtest)

print(confusion_matrix(y_test, y_preds))
print(classification_report(y_test, y_preds))
```

결과:

Confusion_matrix	
100	3
5	63

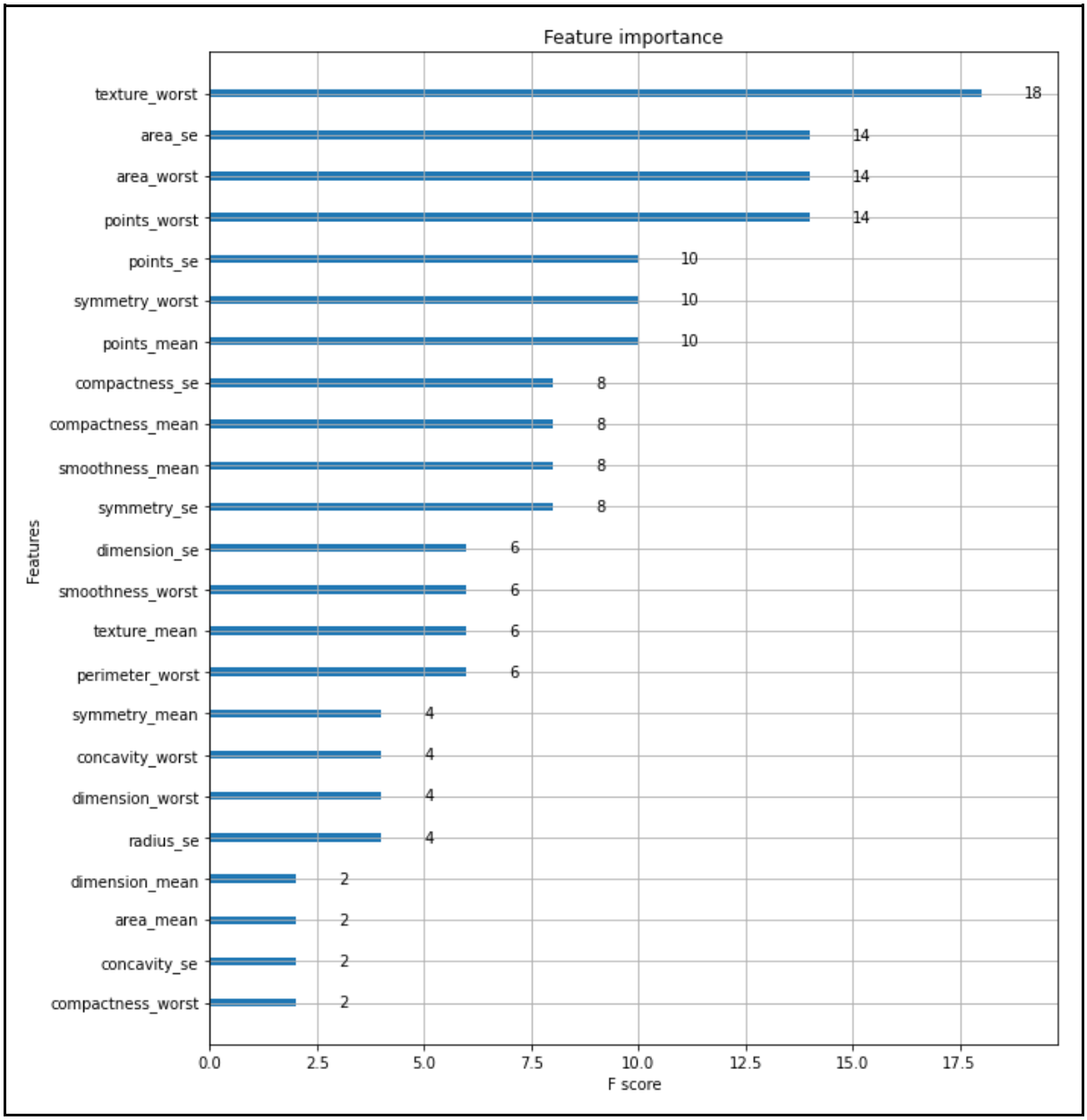
Classification_report				
	precision	recall	f1-score	support
0	0.95	0.97	0.96	103
1	0.95	0.93	0.94	68
accuracy			**0.95**	171
macro avg	0.95	0.95	0.95	171
weighted avg	0.95	0.95	0.95	171

95% 정도의 정확도를 보여주고 있다.

5) 중요 변수 시각화

```
fig, ax = plt.subplots(figsize=(10, 12))
plot_importance(xgb_model, ax=ax)
```

결과:



texture_worst가 가장 중요한 변수이고 area_se, area_worst, points_worst가 그다음 중요변수 point_se, symmetry_worst, points_mean 이 그 뒤를 잇고 있다.

3.3 랜덤포레스트

3.3.1 R 을 이용한 랜덤포레스트

1) Library 호출

```
library(randomForest)
```

2) 랜덤포레스트를 위한 데이터 전처리

```
set.seed(1234)
idx <- sample(nrow(raw_wis), 0.7 * nrow(raw_wis))
train <- raw_wis[idx, -1]
test <- raw_wis[-idx, -1]

train_mat <- as.matrix(train[-c(1, 32)])
train_lab <- train$label
```

데이터셋을 생성할 때 필요없는 id 를 빼고 생성하였다. Matrix 객체로 변환할 때 종속변수를 제거하고 matrix 를 생성하였다. 종속변수는 train_lab 에 따로 담았다.

3) 랜덤포레스트 실행

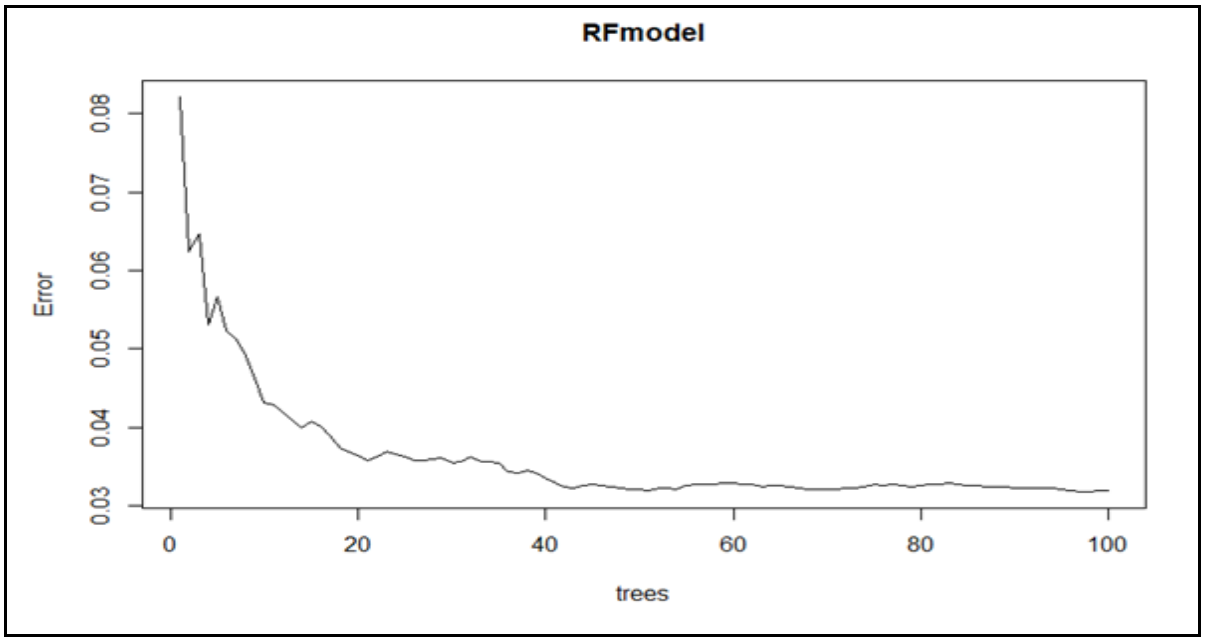
```
RFmodel <- randomForest(train_lab~., data=train_mat, ntree=100,
proximity=T)
RFmodel
```

ntree 는 사용한 트리 수로 100 개로 지정하여 랜덤포레스트를 진행했다.

4) 시각화

```
test_mat <- as.matrix(test[-c(1, 32)])
dim(test_mat)
test_lab <- test$label

plot(RFmodel)
```

결과:



트리가 증가하면서 Error 가 감소하는 것을 확인할 수 있다.

5) 랜덤포레스트의 중요 변수 보기

```
importance = as.data.frame(importance(RFmodel))
importance= arrange(importance,desc(IncNodePurity))
importance$IncNodePurity
```

결과 :

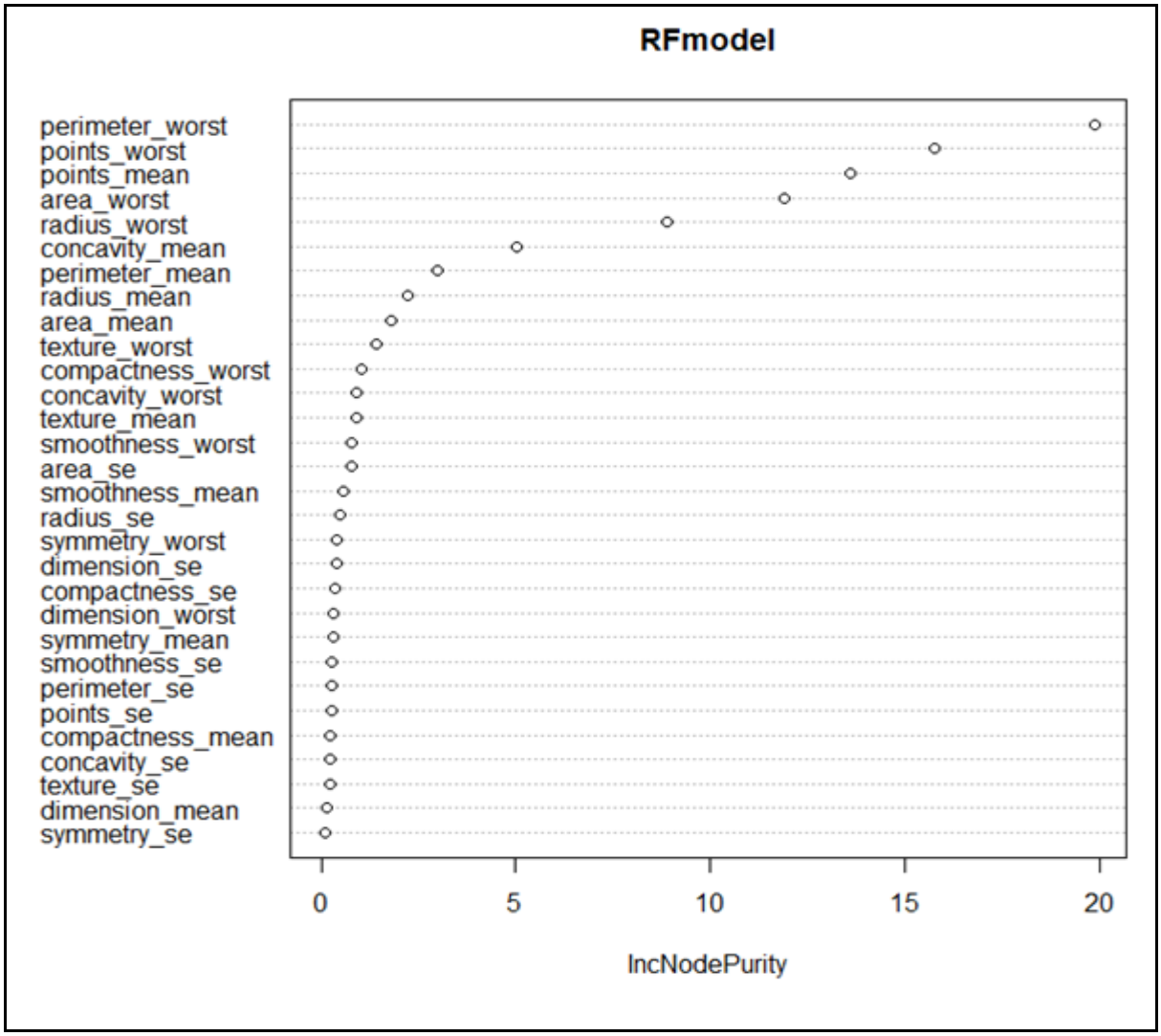
컬럼명	IncNodePurity
perimeter_worst	19.8958356
points_worst	15.7742412
points_mean	13.5944334
area_worst	11.9207533
radius_worst	8.9063491
concavity_mean	5.0389156

perimeter_mean	2.9995902
radius_mean	2.2291575

6) 랜덤포레스트의 중요 변수 시각화

```
varImpPlot(RFmodel)
```

결과 :



perimeter_worst 가 가장 큰영향을 미치고 points_worst, points_mean, area_worst, radius_worst, concavity_mean 등의 순으로 중요변수가 시각화 되었다.

3.3.2 python 을 이용한 랜덤포레스트

1) Import packages

```
import pandas as pd
import numpy as np
from  sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, classification_report,
roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2) 데이터 전처리

```
df = df.drop('id', axis=1)
df.head()
df['diagnosis'] = df['diagnosis'].apply(lambda x : 1 if x== "M" else 0)

x_train,  x_test,  y_train,  y_test  =  train_test_split(df.iloc[:,1:],
df['diagnosis'],test_size=0.3,random_state=11)
```

3) 랜덤포레스트 실행

```
model = RandomForestClassifier(n_estimators=100)
model.fit(x_train, y_train)
```

트리의 갯수를 나타내는 n_estimators 을 100 으로 설정하여 모델을 생성하였다.

4) 분류모델 평가

```
ypred1 = model.predict(x_test)
print("훈련 세트 정확도: {:.3f}".format(model.score(x_train,y_train)))
print("테스트 세트 정확도: {:.3f}".format(model.score(x_test,y_test)))
```

결과 :

훈련 세트 정확도	1.000
테스트 세트 정확도	0.977

97.7%정도의 정확도를 보여주는 것을 확인할 수 있다.

5) 랜덤포레스트의 중요 변수 보기

```
feat_labels = df.iloc[:,1:].columns
importances = model.feature_importances_
indices = np.argsort(importances)
for f in range(df.iloc[:,1:].shape[1]) :
        print('%2d) %-*s %f' %
        (f+1,30,feat_labels[indices[f]],importances[indices[f]]))
```

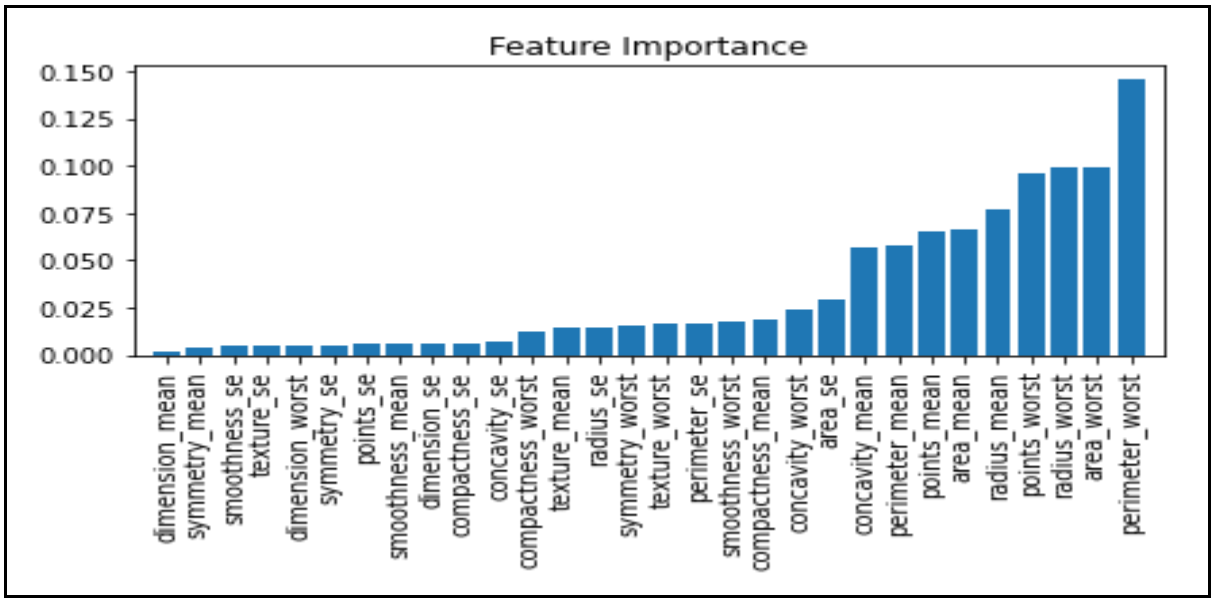
결과 :

perimeter_worst	0.145364
area_worst	0.099239
radius_worst	0.098935
points_worst	0.096283
radius_mean	0.076717
area_mean	0.066593
points_mean	0.065500
perimeter_mean	0.057750
concavity_mean	0.056889
area_se	0.029248

6) 랜덤포레스트의 중요 변수 시각화

```
plt.title('Feature Importance')
plt.bar(range(df.iloc[:,1:].shape[1]),importances[indices],align='cente
r')
plt.xticks(range(df.iloc[:,1:].shape[1]), feat_labels[indices],
rotation=90)
plt.xlim([-1,x_train.shape[1]])
plt.tight_layout()
plt.show()
```

결과:



perimeter_worst 가 가장 중요한 변수이고, area_worst, radius_worst, points_worst, radius_mean, area_mean 순으로 나타난다

3.4 모델 비교

xgboost 는 11 번째 모델이 최적의 모델로 선택 95%정도의 정확도를 보여주고 있다. 랜덤포레스트에서는 97.7%의 정확도로 xgboost 보다 높은 정확도를 보여줬다. xgboost 는 texture_worst 가 가장 중요한 변수였고, 랜덤 포레스트는 perimeter_worst 가 가장 중요한 변수였다. area_worst 와 point_worst 는 xgboost 와 랜덤포레스트 모두에서 높은 중요 변수로 선정되었다.

4. BostonHousing 데이터 예측분석

4.1 데이터 정의

사용 데이터: R 의 melbench 패키지, python 의 sklearn 패키지 내 Bostonhousing dataset

변수명	의미
CRIM	자치시(town) 별 1 인당 범죄율
ZN	25,000 평방피트를 초과하는 거주지역의 비율
INDUS	비소매상업지역이 점유하고 있는 토지의 비율
CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
NOX	10ppm 당 농축 일산화질소
RM	주택 1 가구당 평균 방의 개수
AGE	1940 년 이전에 건축된 소유주택의 비율
DIS	5 개의 보스턴 직업센터까지의 접근성 지수
RAD	방사형 도로까지의 접근성 지수
TAX	10,000 달러 당 재산세율
PTRATIO	자치시(town)별 학생/교사 비율
B	1000(Bk-0.63)^2, 여기서 Bk 는 자치시별 흑인의 비율을 말함.
LSTAT	모집단의 하위계층의 비율(%)
MEDV	본인 소유의 주택가격(중앙값) (단위: $1,000)

MEDV 변수를 종속 변수로 사용하여 예측 모델링 및 분석을 실시한다.

4.2 의사결정나무

4.2.1 R 을 이용한 의사결정나무

1) library 호출

```
library(rpart)
library(mlbench)
library(rpart.plot)
library(car)
library(ggplot2)
```

2) 데이터 불러오기 및 전처리

```
data("BostonHousing")
tree.df <- BostonHousing

set.seed(1234)
idx <- sample(1:nrow(tree.df), nrow(tree.df)*0.7)
train.tree <- tree.df[idx, ]
test.tree <- tree.df[-idx, ]
```

훈련 데이터와 실습 데이터를 7:3 의 비율로 구분하였으며 시드값으로 1234 를 부여하였다.

3) 의사결정나무 모형 생성

```
pre_tree <- rpart(medv ~., data = train.tree, control = rpart.control())
printcp(pre_tree)
```

rpart() 함수를 활용하여 훈련 데이터를 pre_tree 변수에 담았다. control 옵션을 활용하여 노드를 적절하게 조정하였다.

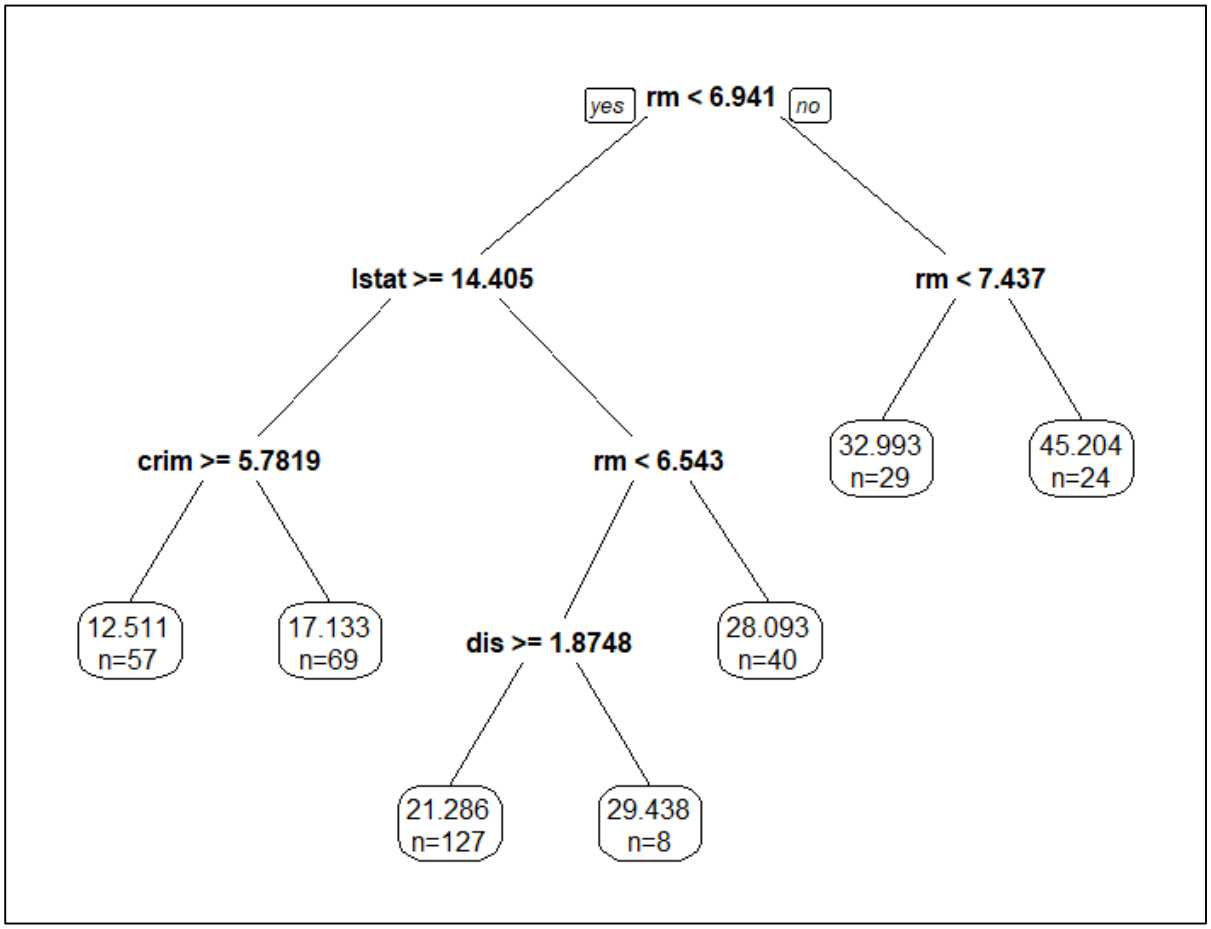
4) 가지치기 및 모형 조정

```
best_cp  <-  pre_tree$cptable[which.min(pre_tree$cptable[,  "xerror"]),
"CP"]
pruned_tree <- prune(pre_tree, cp = best_cp)
```

최초 모형을 확인하여 xerror(교차 타당성 오류)가 최소가 되는 cp(복잡도 모수)값을 구하고, 그에 따른 가지 생성 횟수를 기반으로 기존의 모형을 조정하였다.

5) 의사결정나무 시각화

```
prp(pruned_tree, faclen=0, extra = 1, digits=5)
```



MEDV 변수에 가장 영향을 많이 주는 변수로는 상관도가 높은 순으로 RM, LSTAT, CRIM, DIS 로 확인되었다.

6) 모형 설명력 검증

```
a <- predict(pruned_tree, newdata=test.tree)
test.tree$medv

cor(a, test.tree$medv)
```

실습 데이터를 통해 모형의 설명력을 검증하였다. 상관성은 0.86 가량으로 확인되었다.

4.2.2 python 을 이용한 의사결정나무

1) Import packages

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
import graphviz
```

2) 데이터 전처리

```
boston_raw = load_boston()
def sklearn_to_df(sklearn_dataset):
    df = pd.DataFrame(sklearn_dataset.data,
                                columns=sklearn_dataset.feature_names)
    df['target'] = pd.Series(sklearn_dataset.target)
    return df
df_boston = sklearn_to_df(boston_raw)
df_boston = df_boston.rename({"target": "MEDV"}, axis='columns')
y_target = df_boston['MEDV']
X_data = df_boston.drop(['MEDV'], axis = 1, inplace=False)
x_train, x_test, y_train, y_test = train_test_split(X_data, y_target,
test_size=0.3, random_state = 295)
```

훈련 데이터와 실습 데이터를 7:3 의 비율로 분류하였다.

3) 모형 생성

```
tree_model = DecisionTreeRegressor(max_depth=3 )
tree = tree_model.fit(x_train, y_train)
```

훈련 데이터로 모형을 구축하였으며, 과적합 방지를 위해 최대 깊이는 3 로 설정하였다.

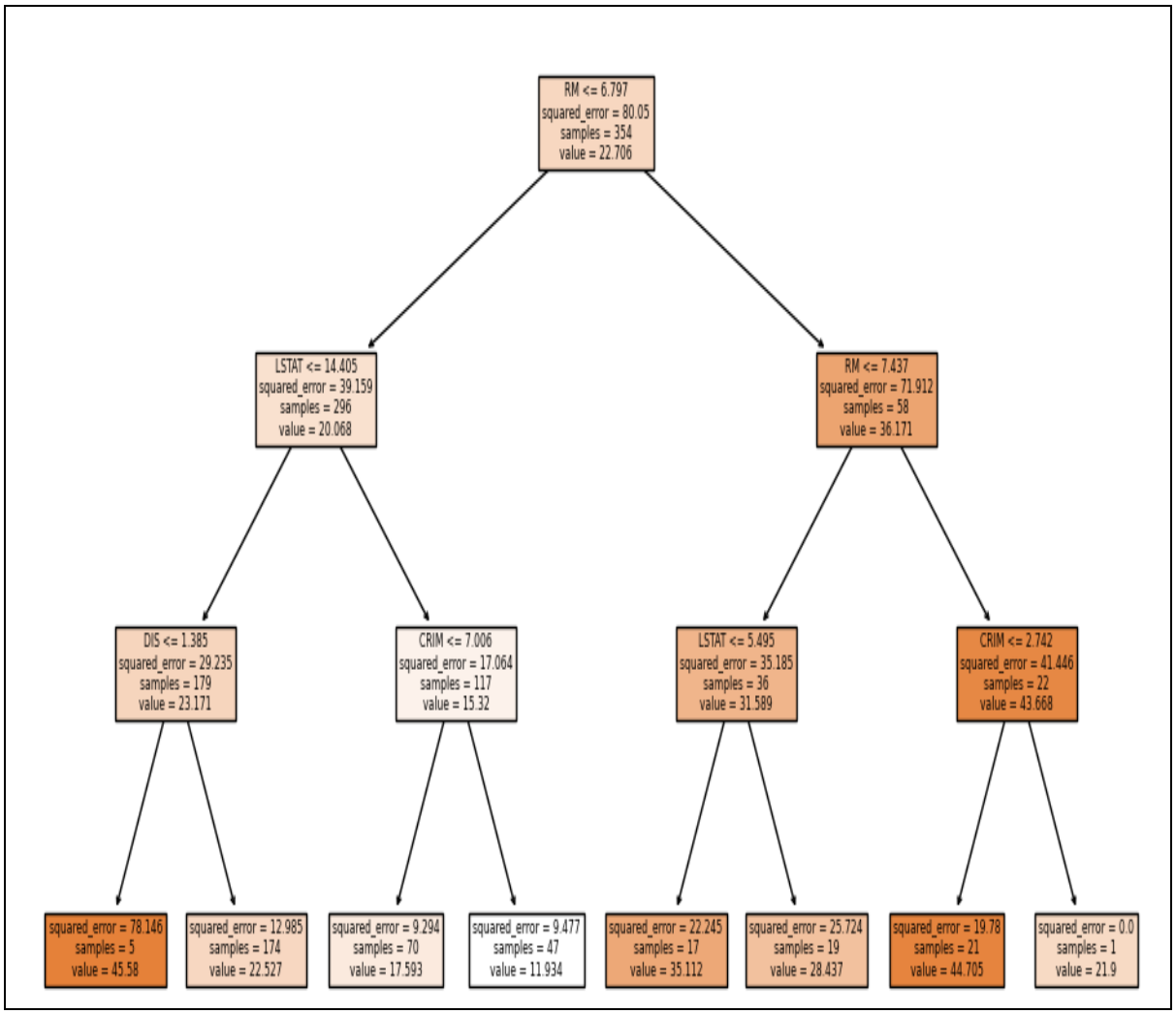
4) 예측 및 모형 설명력 검증

```
pred = tree.predict(x_test)
print('정확도 : ', tree.score(x_train, y_train))
```

실습 데이터로 모형 설명력을 검증하였으며 설명력은 0.82 가량으로 확인되었다.

5) 의사결정나무 시각화

```
fig = plt.figure(figsize=(12,5))
explt_vars = boston_raw.feature_names

decisiontree = plot_tree(tree_model,feature_names =
explt_vars,filled=True)
```



시각화 결과 MEDV 변수에 가장 영향을 많이 주는 변수로는 RM, LSTAT, DIS, CRIM, NOX 등으로 확인되었다.

4.3 선형회귀분석

4.3.1 R 을 이용한 선형회귀분석

1) library 호출

```
library(mlbench)
library(car)
library(ggplot2)
```

2) 선형회귀 모형 생성 및 다중공선성 확인

```
model <- lm(formula = medv ~ ., data = BostonHousing)
vif(model) > 10
```

lm() 함수를 활용하여 선형회귀모형을 만들고 vif() 함수를 활용하여 다중공선성 여부를 확인하였다. 모든 변수에 대하여 다중공선성은 없는 것으로 확인되었다.

3) 변수 선택

```
model <- lm(formula = medv ~ ., data = train)
summary(model)

step <- step(model, direction = 'backward')
formula(step)

model2 <- lm(formula = step, data = train)
summary(model2)
```

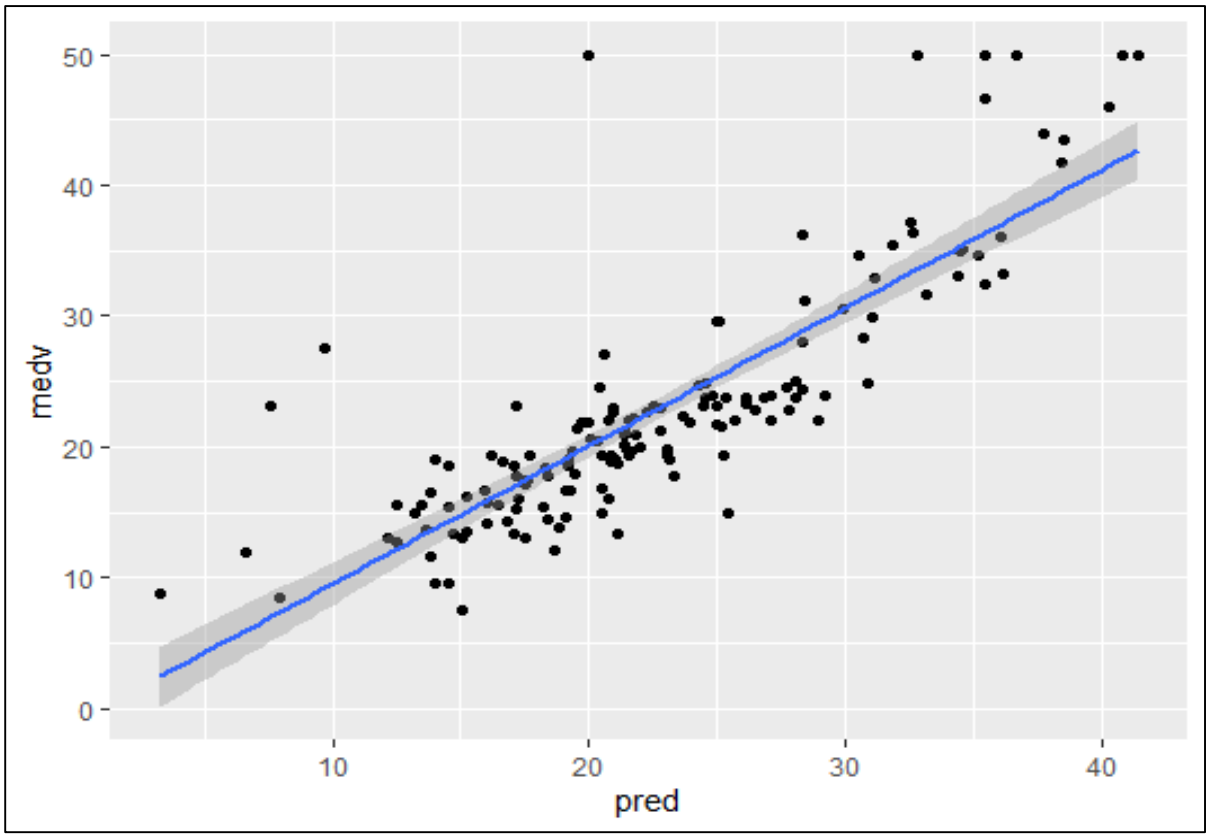
# Adjusted R-squared: 0.7419					
Coefficients:					
	Estimate	Std. Error	t value Pr(>|t|)	Pr(>|t|)	
(Intercept)	26.964605	6.317852	4.268	2.56e-05	***
crim	-0.111504	0.035264	-3.162	0.001707	**
zn	0.047037	0.015544	3.026	0.002665	**
chas1	2.712302	0.963911	2.814	0.005178	**
nox	-18.928319	4.004869	-4.726	3.35e-06	***
rm	4.800240	0.528360	9.085	<2e-16	***
dis	-1.363232	0.215102	-6.338	7.36e-10	***
Rad	0.261290	0.071417	3.659	0.000294	***
Tax	-0.011877	0.003781	-3.141	0.000294	**
Ptratio	-0.851209	0.152907	-5.567	5.25e-08	***
B	0.009343	0.003185	2.934	0.003573	**
Lstat	-0.361711	0.059422	-6.087	3.09e-09	***

훈련 데이터로 모형 생성 시 수정 결정 계수는 0.7413 으로 확인되었으며, 후진 제거법을 통한 변수 선택으로 INDUS, AGE 변수는 분석에서 제외하였다. 변수 제외 후 조정된 모형의 수정 결정 계수는 0.7419 로 유의미한 차이를 보여주지는 않았다. 가장 높은 상관성을 가지는 변수는 높은 순으로 NOX, RM, CHAS1, DIS 등이 있음을 알 수 있었다.

4) 모형 설명력 검증 및 시각화

```
pred <- predict(model2, test)
cor(pred, test$medv)
# [1] 0.8349967

bind <- cbind(pred, test$medv)
head(bind)
colnames(bind) <- c('pred', 'medv')
df <- as.data.frame(bind)

ggplot(df, aes(pred, medv)) +
  geom_point() +
  stat_smooth(method = lm)
```

예측값과 실제 데이터 간의 상관도는 0.83 가량으로 확인되었으며, 예측값과 실제 데이터를 축으로 산점도 및 회귀선을 시각화 하였다.



4.3.2 python 을 이용한 선형회귀분석

1) Import packages

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split


from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import random

import matplotlib.pyplot as plt
```

2) 데이터 불러오기 및 전처리

```
data = load_boston()
```

3) 변수 선택

```
x = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.DataFrame(data.target, columns=['MEDV'])
df = pd.concat([x, y], axis=1)
df.head(10)

# linear regressive analysis
x = df.drop(['MEDV'], axis=1)
y = df[['MEDV']]

var = x.columns.tolist()
var

selected = var
sl_remove = 0.05

sv_per_step = []
adjusted_r_squared = []
steps = []
step = 0

while len(selected) > 0:
    X = sm.add_constant(df[selected])
    p_vals = sm.OLS(y, X).fit().pvalues[1:]
    max_pval = p_vals.max()

    if max_pval >= sl_remove:
        remove_variable = p_vals.idxmax()
        selected.remove(remove_variable)
        step += 1
        steps.append(step)
        adj_r_squared = sm.OLS(y,
sm.add_constant(df[selected])).fit().rsquared_adj
        adjusted_r_squared.append(adj_r_squared)
        sv_per_step.append(selected.copy())
    else:
        break

selected
```

후진 제거법을 통한 변수 선택으로 'INDUS', 'AGE' 변수는 분석에서 제외하고 'CRIM', 'ZN', 'CHAS', 'NOX', 'RM', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'로 회귀분석을 진행했다.

4) 모형 생성 및 변수별 상관계수 추출

```
x = df.drop(["INDUS","AGE","MEDV"], axis=1)
xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size=0.7,
test_size=0.3,random_state=295)
mlr = LinearRegression()
mlr.fit(xtrain, ytrain)
pred = mlr.predict(xtest)
print(mlr.coef_)
```

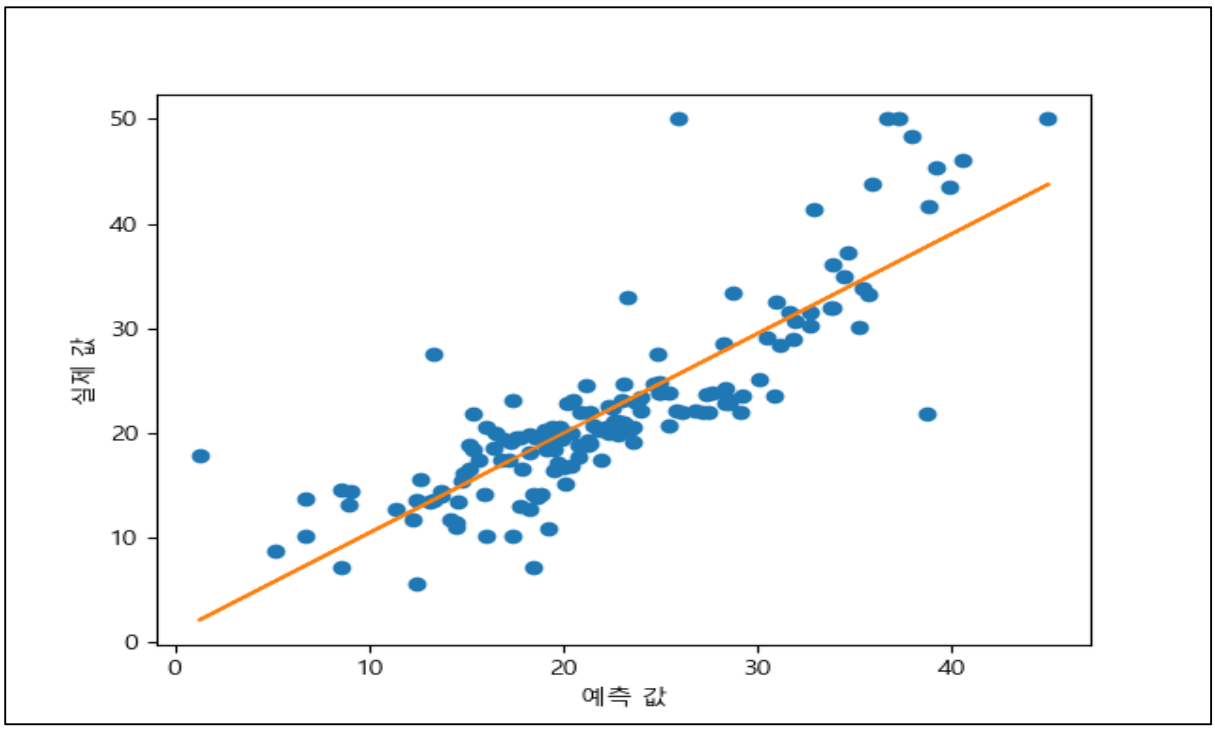
상관관계 계수	
crim	-1.62308538e-01
zn	3.72267658e-02
chas1	3.11801825e+00
nox	-1.63477388e+01
rm	3.48532239e+00
dis	-1.39776781e+00
Rad	3.30832663e-01
Tax	-1.10820251e-02
Ptratio	-8.84128015e-01
B	7.74157735e-03
Lstat	5.82155589e-01

가장 높은 상관성을 가지는 변수는 높은 순으로 RM, NOX, TAX, DIS 등이 있음을 알 수 있었다.

5) 결정 계수 확인 및 시각화

```
mlr.fit(pred.reshape(-1, 1), ytest)
print(mlr.score(pred, ytest))

plt.rc('font', family='Malgun Gothic')
plt.plot(pred, ytest, 'o')
plt.xlabel('예측 값')
plt.ylabel('실제 값')
plt.plot(pred, mlr.predict(pred.reshape(-1, 1)))
```

예측값과 실제 데이터 간의 상관도는 0.82 가량으로 확인되었으며, 예측값과 실제 데이터를 축으로 산점도 및 회귀선을 시각화 하였다.



4.4 분석 결과 비교 및 결론

1) MEDV 변수에 대한 상관관계가 높은 변수

	의사결정나무	선형회귀
R	RM, LSTAT, CRIM, DIS	NOX, RM, CHAS1, DIS
Python	RM, LSTAT, DIS, CRIM, NOX	RM, NOX, TAX, DIS

모든 기법 및 분석 프로그램 상에서 MEDV 변수에 영향을 미치는 변수로 RM 와 DIS 가 있음을 확인할 수 있었다. NOX 또한 R 에서 실시한 의사결정나무 분석을 제외한 모든 곳에서 유의한 변수로 보고 있다. 의사결정나무에서는 LSTAT 과 CRIM 이 주요한 변수로 모두 선택된 반면 선형회귀에서는 그다지 유의하지 않은 변수로 보고 있다.

2) 모형 설명력

	의사결정나무	선형회귀
R	0.86	0.83
Python	0.77	0.74

실제 데이터와 예측 값 사이의 상관계수는 의사결정나무가 선형회귀보다 0.03 가량 높은 것으로 확인되었다. 그러나 동일한 두 가지의 분석법을 각 프로그램에서 실행한 것임에도 R 로 해당 분석을 수행하였을 때 Python 보다 0.09 가량 높은 상관계수를 보여주고 있음을 알 수 있다.

5. Diamonds 데이터 군집분석

5.1 분석 목적

다이아몬드의 가격과 나머지 변수들의 연관성 분석 및 군집 분석을 통하여 관련이 깊은 변수를 찾아내고, 군집의 존재 여부와 군집을 구분하는 기준을 분석하기 위하여 해당 분석을 진행하고자 한다.

5.2 데이터 정의

사용 데이터: ggplot2 패키지 내 diamonds 데이터

컬럼명	의미
price	가격 ($326 ~ $18,823)
carat	무게 (0.2 ~ 5.01)
cut	cut 컷팅의 가치 (Fair, Good, Very Good, Premium, Ideal)
colour	다이아몬드 색상 (J(가장 나쁜)에서 D(가장 좋은)까지)
clarity	깨끗함 (I1 (가장 나쁜), SI1, SI2, VS1, VS2, VVS1, VVS2, IF(가장 좋은))
x	길이 (0 ~ 10.74mm)
y	너비 (0 ~ 58.9mm)
z	깊이 (0~31.8mm)
depth	깊이 비율 = z / mean(x, y)
table	가장 넓은 부분의 너비 대비 다이아몬드 꼭대기의 너비 (43 ~ 95) (비율(%)로 추정됨)

depth 계산식에 x, y, z 가 포함되므로 연관성 분석에서 제외하였으며, 명목척도인 cut, colour, clarity 또한 분석에서 제외하여 price, carat, depth, table 변수에 대해 집중적으로 분석하였다.

5.3 문제 풀이

5.3.1 계층적 군집 분석 (hierarchical clustering)

1) library 호출

```
library(ggplot2)
library(philentropy)
library(cluster)
library(NbClust)
library(corrgram)
```

2) 데이터 샘플링

```
data(diamonds)
str(diamonds)
head(diamonds)
set.seed(1234)
idx <- sample(1:nrow(diamonds), 1000)
head(idx)
t <- diamonds[idx, ]
d <- t[, -c(2:4, 8:10)]
dia <- scale(d)
head(dia)
length(dia$carat)
```

diamonds 데이터 중 임의의 데이터 1000 개를 추출하여 사용하였으며, 명목 척도와 분석에 사용하지 않는 변수를 제거한 후 데이터 표준화를 진행하였다.

3) 거리 계산

```
getDistMethods()
man <- dist(dia, method = 'manhattan')
max <- dist(dia, method = 'maximum')
can <- dist(dia, method = 'canberra')
```

이름	공식 및 설명
maximum	최대 거리. 두 점의 차의 절대값의 최대값. chebyshev 거리와 같다. $$D_{\text{Chebyshev}}(x,y) := \max_i(\lvert x_i - y_i \rvert).$$
manhattan	맨해튼 거리. 두 점의 차의 절대값의 총 합. $$d_1(\mathbf{p},\mathbf{q}) = \lVert \mathbf{p}-\mathbf{q} \rVert_1 = \sum_{i=1}^{n} \lvert p_i - q_i \rvert$$
canberra	캔버라 거리. 두 점의 차의 절대값을 두 점의 합으로 나눈 값의 총 합. $$d(x,y) = \sum_{i=1}^{p} \frac{\lvert x_i - y_i \rvert}{(x_i - y_i)}$$

dist() 함수로 바로 거리 계산이 가능한 manhattan, maximum, canberra 거리 변수에 입력한 후 거리를 계산하였다.

4) 계층적 군집 분석과 덴드로이드 시각화 (일부)

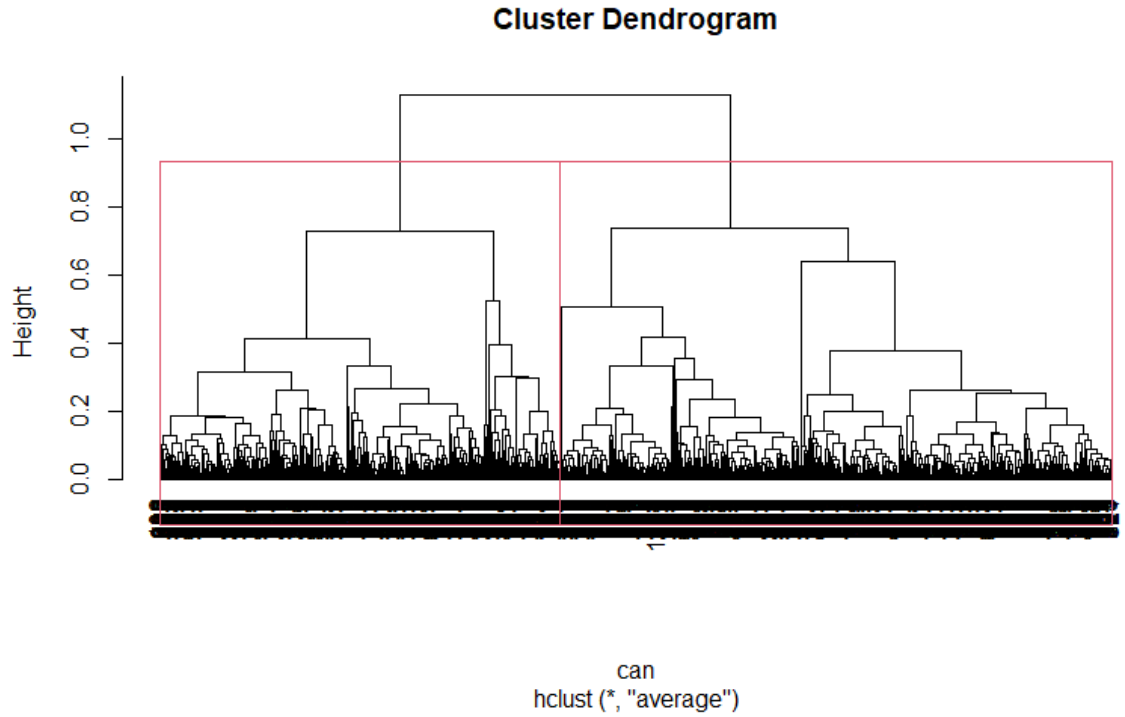
```
man_hc1 <- hclust(man, method = 'average')

NbClust(dia, distance = 'manhattan', method = 'average')
```

모든 연결법에 대해 NbClust()를 진행하였으며, 최적 군집수는 아래와 같이 확인되었다.

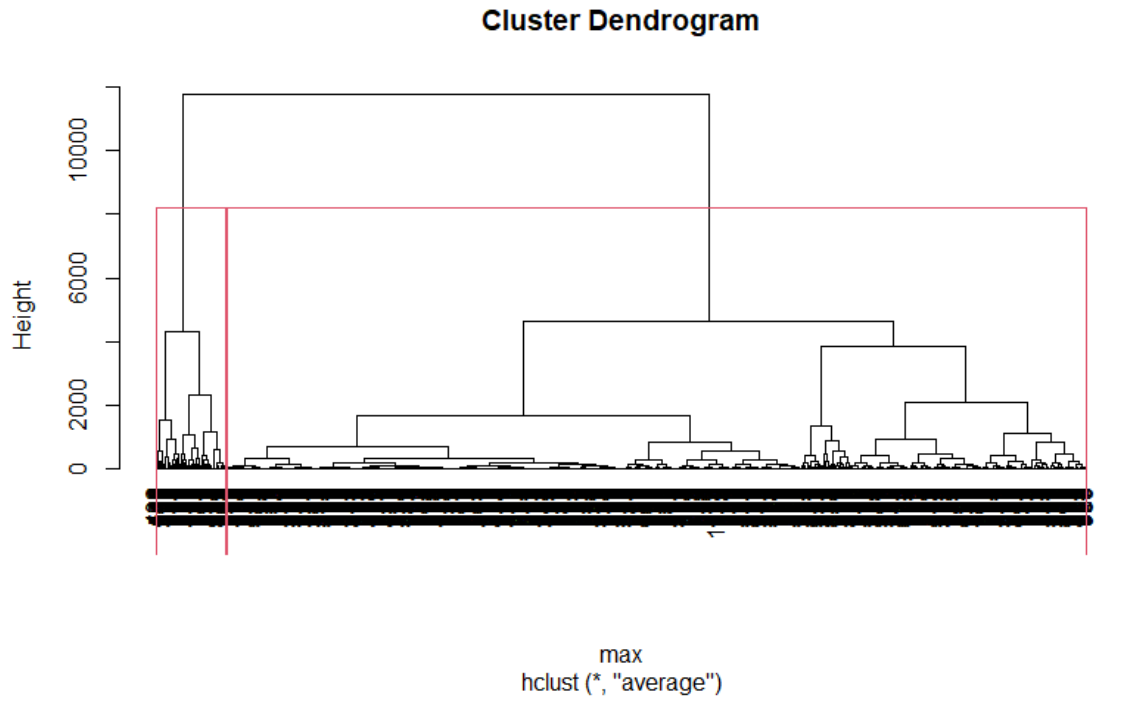
	average	single	complete	centroid	ward.D	mcquitty	median
manhattan	2	2	2	2	2	2	2
maximum	2	2	2, 10	2	3	2	2
canberra	2	2	3	2	2, 3	2	2

NbClust() 결과 2 개의 군집으로 분류하는 것이 적절함을 알 수 있었다. 추가로 덴드로이드를 시각화하여 적절한 군집 수를 확인해 보았으며, 그 중 manhattan 과 gower distance 의 시각화 결과는 아래와 같다.

canberra distance: average



maximum distance: average



덴드로그램 상에서 군집을 나누는 경우 군집 간 높이 차이가 큰 군집의 개수를 선택하여야 군집 간 이질성이 큰 군집이 되므로 이를 활용하고자 하였으나, 데이터 수가 많아 유의미한 군집을 찾기 어려운 경우가 많았다. 이와 같은 경우를 제외한 분기점이 눈에 띄는 데이터의 경우 대부분 2 개의 군집으로 나누는 것이 적절할 것으로 확인되었다.

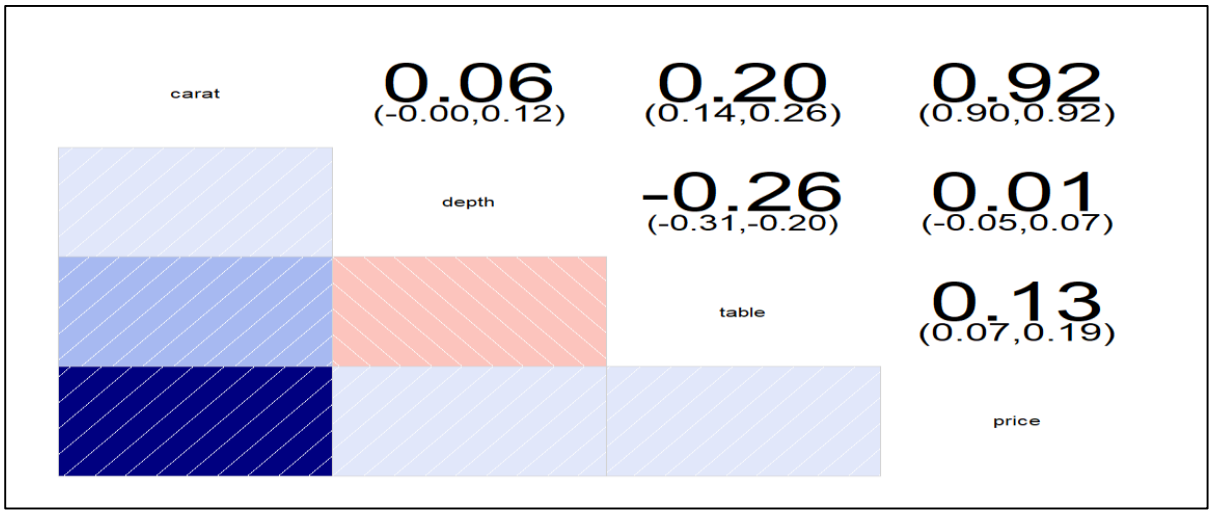
NbClust() 결과 데이터와 덴드로그램 분석 결과를 바탕으로 2 개의 군집으로 군집분석을 진행하기로 결정했다.

5.3.2 비계층적 군집 분석 (k-means clustering)

1) 상관계수 확인

```
corrgram(dia, upper.panel = panel.conf)
corrgram(dia1, upper.panel = panel.conf)
```

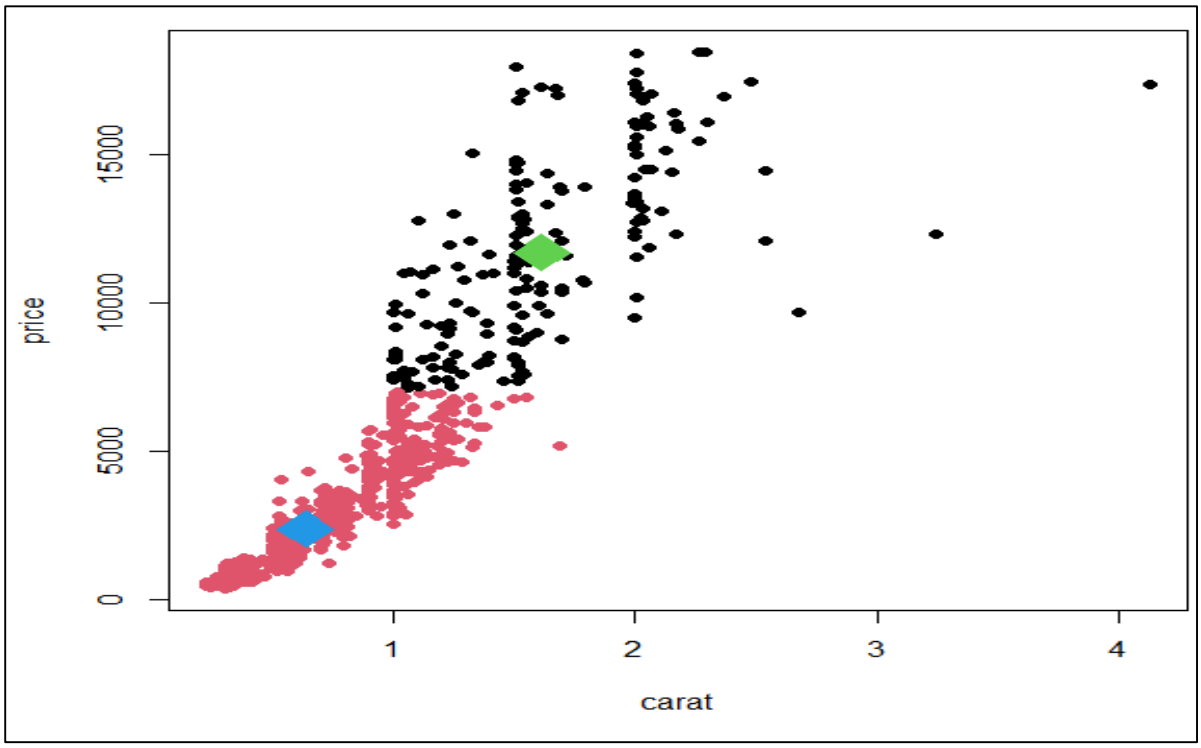
계층적 군집 분석 결과를 통해 2 개의 군집으로 데이터를 분류하는 것이 적절할 것으로 확인하여 kmeans() 함수를 통해 비계층적 군집 분석을 실행하였다. 분석 대상으로는 price 와 상관계수가 0.92 로 가장 높은 carat 데이터로 지정하였다. 상관계수표는 다음과 같다.



▲명목척도를 제외한 데이터셋의 상관계수표 (carat, depth, table, price)

2) k-means 군집분석 및 시각화

```
kmeans <- kmeans(d, 2)

X11()
plot(d[c('carat', 'price')], col = kmeans$cluster,
     pch = 21, bg = kmeans$cluster)
points(kmeans$centers[, c('carat', 'price')],
       col = 3:4, pch = 23, cex = 3, bg = 3:4)
```



k-means 군집 분석 시각화 결과 price 값 7000 근처에서 확연한 구분선이 보이지만 carat 값으로는 명확한 구분선이 보이지 않으며, 군집 간의 거리도 상당히 가까운 것으로 보아 데이터가 유의미한 군집을 형성하고 있지 않다고 결론지을 수 있다.

5.4 결론

diamonds 데이터를 대상으로 다양한 거리 계산법과 연결법을 적용하여 분석해 보았으나 대부분의 경우에서 계층적 군집분석 결과는 해당 데이터가 2 개의 군집으로 이루어져 있는 것으로 유사하게 나타나 유의미한 차이를 보이지 않았다. 이 데이터를 바탕으로 비계층적 군집 분석을 실시하였으나 군집 간의 경계가 모호한 것으로 보아 군집 간 이질성이 떨어지는 것으로 판단되어 해당 데이터가 군집성이 있다고 보기 어려웠다.

6. Zelensky 연설문 텍스트 분석

6.1 분석 개요

6.1.1 분석 목적

본 보고서는 Zelensky 의 연설문에 기반한 빈도 분석에 따른 단어구름 생성과 연관분석을 기술한다. 그에 따라 현재 우크라이나가 러시아에 대해 취하는 태도를 살펴보고, 나아가 어떠한 부분에서 훌륭한 지도자로서의 연설문으로 평가받는지 분석해 보고자 한다.

6.1.2 원본 데이터 정의

사용데이터: zelensky.txt

데이터 분석에 앞서 원본 텍스트 파일을 살펴보았다. 원본 파일은 우크라이나 원어 연설문을 영어로 번역한 후 다시 한국어로 번역한 버전이며, 일부 문단은 원어와 영어가 함께 기술되어 있다. 원문의 의미 훼손을 최소화하는 방향에 따라 아래와 같이 분석 데이터를 선별했다.

원본 출처 링크와 우크라이나어, 영어를 제외한 텍스트 전문

우크라이나어와 영어를 기반으로 한국어 번역을 수행한 텍스트이므로 제거하더라도 의미 분석에는 큰 영향이 없을 것으로 판단했다. 다만, '당신들'이나 '너희들'과 같이 불용어의 판단이 모호한 경우에 영어 번역본을 참고한다.

6.2 문제 풀이 - 단어 구름

6.2.1 변수 정의

변수명	정의
raw_zel	Zelensky 연설문 원본
user_dic	단어 사전에 추가할 단어 (우크라이나, 러시아)
exNouns	명사를 추출하기 위한 사용자 정의 함수
zel_nouns	raw_zel 변수로부터 추출한 명사
zelcorpus	zel_nouns 를 말뭉치로 만든 것
zelcorpus2	전처리한 텍스트 데이터
myStopwords	불용어 처리할 단어 목록
zelcorpus3	전처리 결과(zelcorpus2)중에서 선별한 단어 문서 행렬
zelcor_df2	데이터프레임으로 변경한 단어 문서 행렬
wordResult2	내림차순으로 정렬한 단어 빈도수
word.df	빈도수를 단어 구름으로 생성하기 위한 dataframe
word.df2	word.df 의 결과에서 빈도수 2 이상인 단어만을 선별

6.2.2 텍스트 전처리

1) library 호출 및 데이터 불러오기

```
library(KoNLP)
library(tm)
library(multilinguer)
library(wordcloud)
library(wordcloud2)
library(stringr)
library(tidytext)

setwd('c:/Rwork')
raw_zel <- readLines("zelensky.txt", encoding = "UTF-8")
head(raw_zel)
```

2) 1 차 전처리 및 명사 추출

```
user_dic <- data.frame(term = c("우크라이나", "러시아"), tag = "ncn")
buildDictionary(ext_dic = "sejong", user_dic = user_dic)

raw_zel <- str_replace_all(raw_zel, "사람들", "사람")
raw_zel <- str_replace_all(raw_zel, "영웅들", "영웅")
raw_zel <- str_replace_all(raw_zel, "당신들이", "당신")

exNouns <- function(x){paste(extractNoun(as.character(x)),
                                        collapse = " ")}

zel_nouns <- sapply(raw_zel, exNouns)
zel_nouns[1]
```

'우크라이나'와 '러시아'를 단어 사전에 추가하지 않고 단어 분석을 진행할 경우, 해당 단어 두 가지가 원문에서 5 회 이상 등장함에도 불구하고 빈도수가 실제보다 적게 나온다. 따라서 우크라이나와 러시아를 단어 사전에 추가하는 과정을 거쳤다. 또한, '사람들', '영웅들'과 같은 유의어의 경우에도 앞선 두 단어와 마찬가지로 실제보다 적은 빈도수가 도출되므로, '사람', '영웅'과 같은 단어로 대체했다. 원문에서 "당신"은 바이든이나 유럽과 같은 청자를 직접 지시하고 있기에 빈도수를 높여주었다.

1 차 전처리를 끝내고 extractNoun 을 활용한 사용자 정의 함수를 통해 명사를 추출했다.

3) 말뭉치 생성 및 2 차 전처리

```
zelcorpus <- Corpus(VectorSource(zel_nouns))

zelcorpus2 <- tm_map(zelcorpus, removePunctuation)
zelcorpus2 <- tm_map(zelcorpus, removeNumbers)
zelcorpus2 <- tm_map(zelcorpus, tolower)
myStopwords = c(stopwords('english'),
                "때문", "이것", "그것", "들이", "해서",
                "무엇", "저들", "이번", "우린", "우리")

zelcorpus2 <- tm_map(zelcorpus, removeWords, myStopwords)

inspect(zelcorpus2)

zelcorpus3 <- TermDocumentMatrix(zelcorpus2,
                                 control = list(wordLengths = c(4, 16)))
```

명사 추출 결과로 말뭉치를 생성 후, 문장부호 제거, 수치 제거, 소문자 변경, 불용어 제거 순으로 전처리를 진행했다. 불용어의 경우, 영어를 포함하여 "때문", "이것", "그것", "들이", "해서", "무엇", "저들", "이번", "우린", "우리"를 선별했다.

"우리"와 "우린"의 경우에는 빈도수가 50 회 이상으로 두 번째로 빈도수가 많은 "우크라이나"(22 회) 보다 압도적으로 많아 단어 구름에서 상대적으로 빈도수가 적은 단어들이 지나치게 축소된다는 결점이 있었고, "우크라이나"와 "우리", "우린"은 맥락 상 동일하므로 제거하여도 "우크라이나"라는 단어로 충분히 강조된다고 판단했다. 또한 "너희들"과 같이 모호한 단어에 대해서는, 영어 번역본에서 러시아를 "you"로 강조한 점[1]이 유의하다고 판단해 불용어 처리하지 않았다.

"우크라이나"와 같이 비교적 길이가 긴 단어를 추출해야 하므로, wordLengths 를 4 부터 16 으로 설정했다. 아래는 위 과정의 결과 (zelcorpus3) 이다.

```
> zelcorpus3
<<TermDocumentMatrix (terms: 218, documents: 70)>>
Non-/sparse entries: 340/14920
Sparsity           : 98%
```

[1] Without gas or without you? - The answer is without you. 영어 번역본 일부 발췌

```
Maximal term length: 6
Weighting          : term frequency (tf)
```

6.2.3 단어 구름 생성

```
zelcor_df2 <- as.data.frame(as.matrix(zelcorpus3), stringsAsFactors =
F)
dim(zelcor_df2)

wordResult2 <- sort(rowSums(zelcor_df2), decreasing=TRUE)
wordResult2[1:50]

zelname <- names(wordResult2)
word.df <- data.frame(word = zelname, freq=wordResult2)

word.df2 <- subset(word.df, subset = freq >= 2)
head(word.df2)
str(word.df2)

wordcloud2(data = word.df2,
           size = 1, color = 'random-light', gridSize = 1,
           backgroundColor="black"
           , maxRotation = -pi/5, minRotation = -pi/2, shape = "circle")
```

단어 구름을 생성하기 위해 데이터 프레임을 만들고, 빈도수를 내림차순으로 도출했다.

아래는 내림차순 50 개 단어 중 일부 목록이다.

우크라이나	승리	너희들	사람	여기	당신
22	10	10	9	8	6
국가	영광	영웅	항복	대통령	자유
6	6	6	5	5	5
러시아	전쟁	독립	죽음	세계	위대
5	5	4	4	3	3
여러분	증명	질문	무기	지도자	처벌
3	3	3	3	3	3
불구	조상	약속	국민	이룩	정상
3	3	2	2	2	2
하나	최소한	대상	자식	수호자	수백만
2	2	2	2	2	2

전처리에서 의도했던 대로, "우크라이나"가 가장 큰 빈도수인 22 회로 도출됐음을 확인할 수 있다. 또한 "사람들"이 유의어 처리되어 "사람"으로 9 회 도출되었고, "당신" 역시 6 회 도출되었다. 그 외에 불용어 처리한 것들이 제거되어 빈도에 따른 의미를 분석하기에 적절하다. 빈도수가 1 회인 것은 크게 유의하지 않다 판단해 제거한 후 단어 구름을 생성한 결과는 아래와 같다.

▲ Wordcloud2 시각화 결과

6.2.4 결과 분석

시각화 결과, 빈도분석과 동일하게 "우크라이나"가 최빈 단어로 도출되었음을 볼 수 있다. "승리", "전쟁", "항복", "국가"와 같은 단어를 통해 전쟁이라는 특수한 상황에 놓여 있는 우크라이나를 확인할 수 있었다. 또한 영어 번역본으로는 "you", 그리고 한국어 번역본으로는 "너희들"이라는 다소 강한 표현을 함으로써, 강한 태도를 표출하고 굳건한 모습을 드러냈다.

그리고 "여러분"과 같이 청자를 지속적으로 지칭함으로써 주의를 집중시키고 관심을 끌고 있다. 우크라이나의 지도자로서 강한 태도를 보임으로써 국민들에게는 안심을 주고, 국제사회의 지원과 도움을 효과적으로 요청할 수 있었을 것이며, 적인 러시아에게는 굴복하지 않는 모습으로 보였을 것이라고 해석된다.

6.3. 문제 풀이 - 연관 분석

6.3.1 변수 정의

변수명	정의
raw_zel2	Zelensky 연설문 원본
lword	raw_zel2 에서 줄 단위로 추출한 단어
filter1, filter2	조건에 맞는 단어를 필터링하기 위한 사용자 정의 함수
wordtran	필터링한 단어로 생성한 트랜잭션
tranrules	트랜잭션으로 생성한 연관규칙
rules	연관어 시각화/ 행렬구조로 변경을 위한 변수
rulemat	행렬구조의 연관어 matrix
ruleg	edgelist 를 생성하기 위한 싱글 객체를 포함

6.3.2 텍스트 전처리

1) library 호출 및 데이터 불러오기

```
library(arules)
library(backports)
library(igraph)
library(KoNLP)
library(tm)
library(multilinguer)
library(stringr)
library(tidytext)

raw_zel2 <- readLines("zelensky.txt", encoding = "UTF-8")
```

2) 줄 단위 단어 추출, 추출 단어 확인

```
lword <- Map(extractNoun, raw_zel2)
length(lword)
lword <- unique(lword)
length(lword)

lword <- sapply(lword, unique)
length(lword)
lword
```

readLines 로 줄 단위로 읽어온 데이터를 extractNoun 로 줄 단위로 단어를 추출하고, unique 함수로 중복 단어를 제거해주었다. 아래는 그 결과 중 일부를 정리한 표이다.

unique 처리 전	unique 처리 후
[1] "우리", "오크와", "엘프", "사이[9]", "완충지대", **"우리"**, "정상", "국가"	[1] "우리", "오크와", "엘프", "사이[9]", "완충지대", "정상", "국가", "풍족"
[9] "풍족", "한", **"국가"**, **"우리"**, "사정" **"우리"**, "소유", "명분"	[9] "한", "사정", "소유", "명분", "것" "저", "끝", "위대"
[17] "것", "저", **"우리"**, "끝", **"우리"** "위대", **"한"**, **"국가"**	[17] "이룩", "기회", "우크라이나", "국민" "들", "분열", "반대", "하나"

중복 단어 "우리", "국가", "한"이 unique 처리 후에 한 개씩만 남아 있음을 확인할 수 있다.

3) 필터링 함수를 통한 전처리

```
filter1 <- function(x){
  nchar(x) <= 10 && nchar(x) >=2 && is.hangul(x)
}
filter2 <- function(x){Filter(filter1, x)}

lword <- sapply(lword, filter2)
lword
```

6.3.3 연관 분석

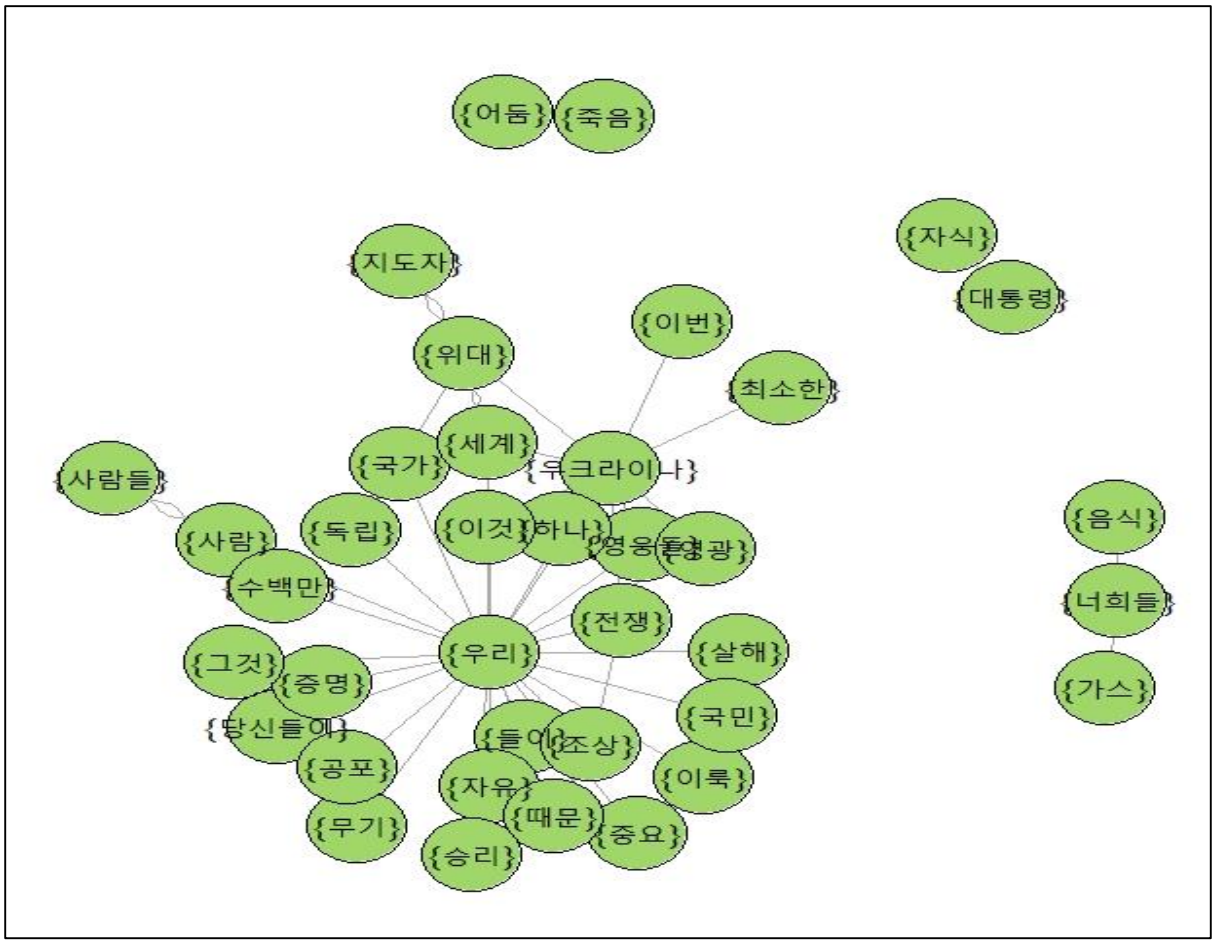
1) 트랜잭션 생성 및 연관규칙 발견

```
wordtran <- as(lword, "transactions")
wordtran

tranrules <- apriori(wordtran, parameter = list(supp=0.04, conf = 0.6))

#detach(package:tm, unload =TRUE)
inspect(head(sort(tranrules, by = "lift")))

rules <- labels(tranrules)

rules <- sapply(rules, strsplit, " ", USE.NAMES = F)
rules

rulemat <- do.call("rbind", rules)
class(rulemat)
rulemat
```

생성된 트랜잭션을 기반으로 연관규칙을 matrix 로 변환했다. supp 값을 0.04, conf 값을 0.6 으로 조정하여 총 118 개의 연관규칙을 도출했는데, 본 보고서에서는 원활한 분석을 위해 118 개 중 60 개의 single 객체만을 활용하여 시각화 및 분석을 진행했다.

2) edgelist 시각화

```
ruleg <- graph.edgelist(rulemat[c(1:60),-2], directed = F)
plot.igraph(ruleg, vertex.label = V(ruleg)$name,
            edge.lty = "solid", edge.width = 1.5,
            vertex.label.cex =1.2, vertex.label.color = 'black',
            vertex.size = 20, vertex.color = "#a1d76a",
            vertex.frame.color = "black")
```



▲ 연관분석 시각화 결과

6.3.4 결과 분석

연관분석 시각화 결과, "우리" 와 "우크라이나" 두 단어를 중심으로 이루어진 단어 간 연관성을 확인할 수 있었다. 중심 단어와 다른 단어들 간의 거리가 모두 유사한 것으로 보아, 어떤 단어를 크게 강조했기 보다는 "우리"를 중심으로 다양한 단어들을 폭넓게 사용했다는 결론을 얻을 수 있다.

7. ggplot2 와 matplotlib 을 이용한 시각화

7.1 데이터 정의

데이터: iris

시각화에 앞서 컬럼의 의미를 확인 및 분석하고 아래 표와 같이 컬럼의 정의를 내렸다.

컬럼명	의미
Sepal_Length	꽃받침의 길이(cm)
Sepal_Width	꽃받침의 너비(cm)
Petal_Length	꽃잎의 길이(cm)
Petal_Width	꽃잎의 너비(cm)
Species	붓꽃의 종(setosa, versicolor, virginica)

변수별 Species 를 중점으로 분석하되, 두 변수를 x, y 축으로 설정해야 하는 경우에는 꽃받침에 대한 설명인 Sepal_Length, Sepal_Width, 꽃잎에 대한 설명인 Petal_Length, Petal_Width 를 각각 묶어 분석한다.

7.2 데이터 시각화

1) library, packages 호출

R	python
library(ggplot2) library(reshape2) library(GGally)	import matplotlib.pylab as plt import seaborn as sns from pandas import Series, DataFrame

2) 데이터 불러오기

R	data("iris") visual_data <- iris data <- aggregate(iris[,1:4], by = list(iris$Species), FUN = mean) mt.visual <- melt(data, id.vars = c('Group.1'))
python	iris = sns.load_dataset('iris') iris_mean = iris.groupby('species').mean() iris_mean

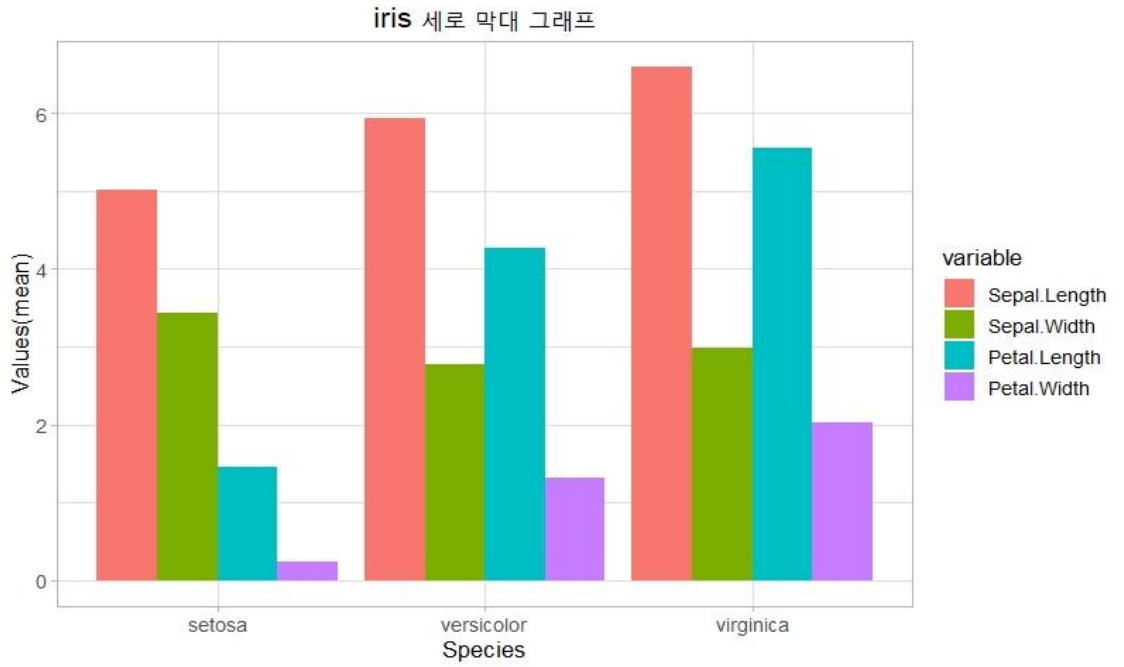
원 데이터를 담고 있는 변수 하나와, 막대그래프나 누적그래프를 생성할 때 사용할 변수 별 평균을 담고 있는 변수를 포함해 총 두 개의 변수를 생성했다.

7.2.3 iris 데이터 시각화

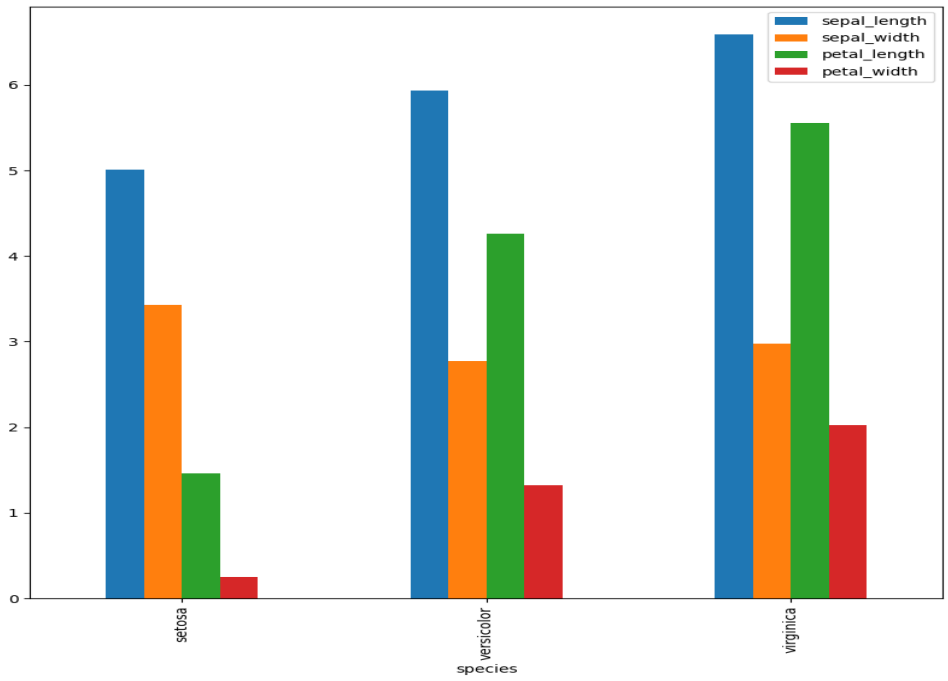
1) 세로 막대 그래프

R	chart2 <- ggplot(mt.visual, aes(x = Group.1, y = value, group = variable, fill = variable)) + geom_bar(stat = "identity", position = "dodge") + xlab('Species') + ylab('Values(mean)') + theme_light()+ theme(text = element_text(size = 13), axis.text.x = element_text(angle = 0, hjust =.5), plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) + ggtitle("iris 세로 막대 그래프")
python	iris_mean.plot(kind='bar')

ggplot2 을 이용한 세로 막대그래프



matplotlib 을 이용한 세로 막대그래프



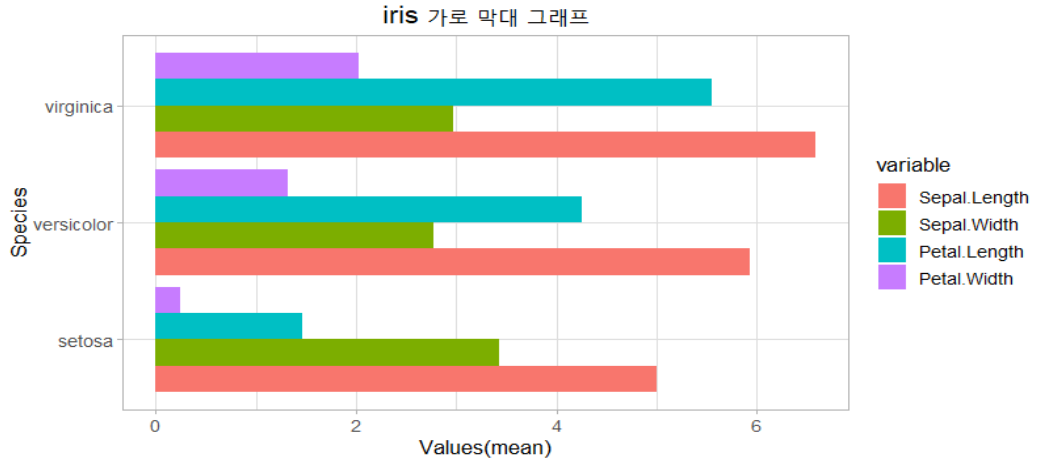
R 은 geom_bar 를 통해 막대차트를 생성하고 Position='dodge'로 막대를 각 범주별로 분리했다. 두 그래프 모두 species 기준 변수의 평균을 세로 막대차트로 시각화 했으며, 그래프의 모양이 동일함을 확인할 수 있다.

2) 가로 막대 그래프

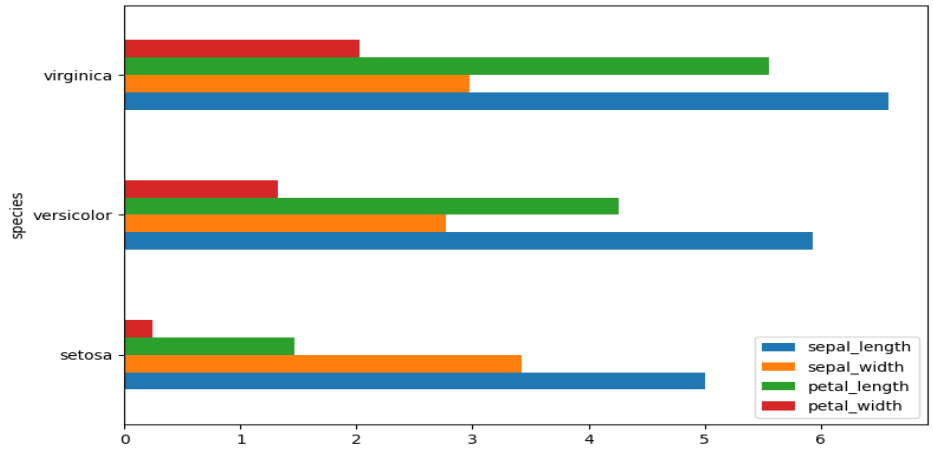
R	chart1 <- ggplot(mt.visual, aes(x = Group.1, y = value, group = variable, fill = variable)) + geom_bar(stat = "identity", position = "dodge") + xlab('Species') + ylab('Values(mean)') + theme_light() + theme(text = element_text(size = 13), axis.text.x = element_text(angle = 0, hjust =.5), plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5)) + coord_flip() + ggtitle("iris 가로 막대 그래프")
python	iris_mean.plot(kind='barh')

R 에서는 coord_flip 함수를 이용하여 x 축과 y 축을 변경하여 가로 막대를 생성했고, python 에서는 'barh'옵션을 줬다. 세로 그래프와 마찬가지로 그래프의 모양이 동일하다.

ggplot2 을 이용한 가로 막대그래프



matplotlib 을 이용한 가로 막대그래프



3) 누적막대 차트

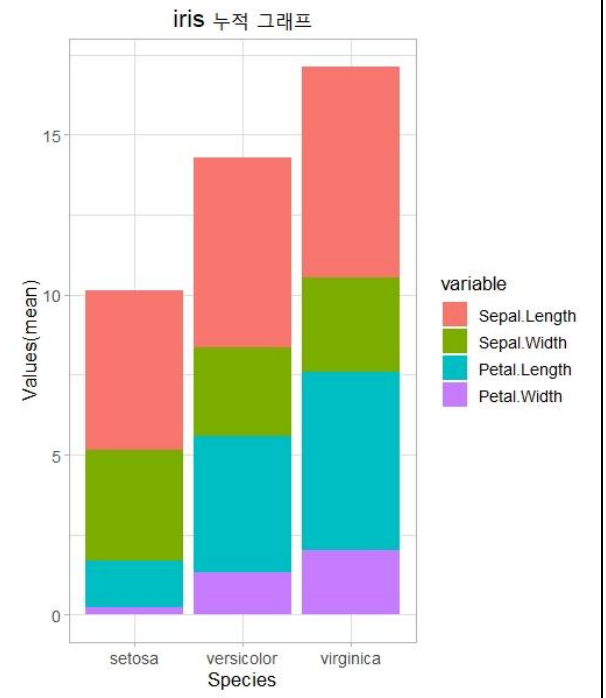
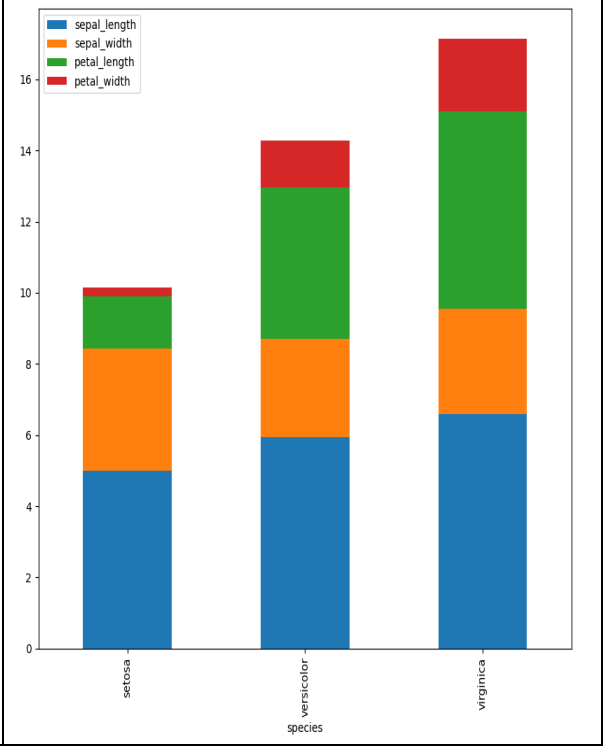
R

```
chart3 <- ggplot(mt.visual, aes(x = Group.1, y = value,
                                group = variable, fill =
variable)) +

  geom_bar(stat = "identity", position = "stack") +
  xlab('Species') + ylab('Values(mean)') + theme_light() +
  theme(text = element_text(size = 13),
        axis.text.x = element_text(angle = 0, hjust =.5),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)) +
  ggtitle("iris 누적 그래프")
```

python

```
iris_mean.plot(kind='bar',stacked = True)
```

R 과 position 을 'stack'으로 변경했고, python 은 stacked = True 라는 옵션을 추가했다.

ggplot2 를 이용한 누적 막대 그래프	matplotlib 을 이용한 누적 막대 그래프
	

누적 막대 그래프 역시 다른 막대 그래프들처럼 변수의 평균을 기반으로 생성했다. 두 막대 그래프를 살펴보면, ggplot2 은 비율이 적은 것이 아래쪽에 배치되어 있으나 matplotlib 은 위에 배치되어 있다는 것을 확인할 수 있다.

4) 점 차트

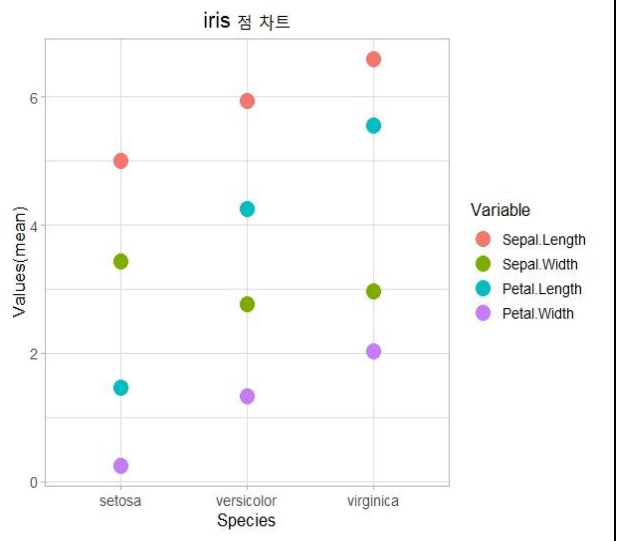
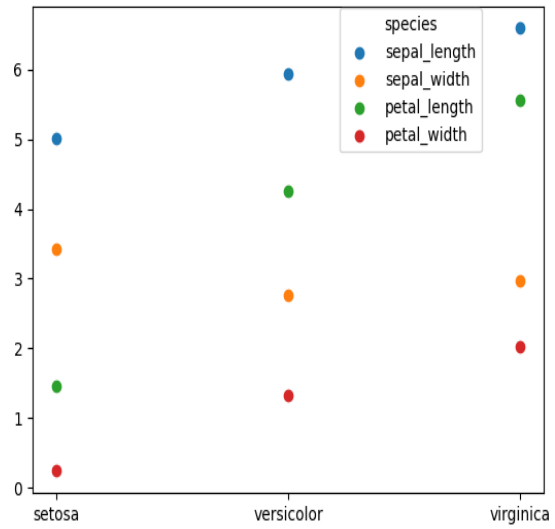
R

```r
chart4 <- ggplot(mt.visual, aes(x = Group.1, y = value,
                                group = variable, color = variable))+
  geom_point(shape = 19, size = 5) +
  xlab('Species') + ylab('Values(mean)') + theme_light()+
  theme(text = element_text(size = 13),
        axis.text.x = element_text(angle = 0, hjust =.5),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))+
  labs(color = "Variable") +
  ggtitle("iris 점 차트")
```

python

```python
plt.scatter(iris_mean.index, iris_mean['sepal_length'])
plt.scatter(iris_mean.index, iris_mean['sepal_width'])
plt.scatter(iris_mean.index, iris_mean['petal_length'])
plt.scatter(iris_mean.index, iris_mean['petal_width'])
plt.legend(loc = (0.6, 0.7), labels = ['sepal_length', 'sepal_width', 'petal_length','petal_width'],title = 'species')
```

R 은 geom_point() 요소를 맵핑해 점 차트를 나타냈고, python 은 scatter 함수를 사용했다.

ggplot2 를 이용한 점 차트	matplotlib 을 이용한 점 차트
	

막대그래프(세로, 가로, 누적)와 점 차트는 iris 데이터의 변수 평균을 활용하여 시각화를 했다. R 과 python 의 결과는 거의 일치하였으나 시각화를 하는 과정에서 차이점을 발견할 수

있었다. ggplot2 는 geom_point()등 요소 맵핑을 통해 형태를 변경하고, matplotlib 은 패키지 내 다양한 함수를 이용하여 시각화를 할 수 있다.

또한 변수 평균을 통한 시각화 결과, Sepal_length, Petal_length, Petal_width 세 개 변수에서 virginica 가 가장 큰 값을 갖는 것을 발견했다. Setal_width 는 setosa 가 3.4cm 정도로 가장 큰 값을 갖는다.

5) 원형 차트

R

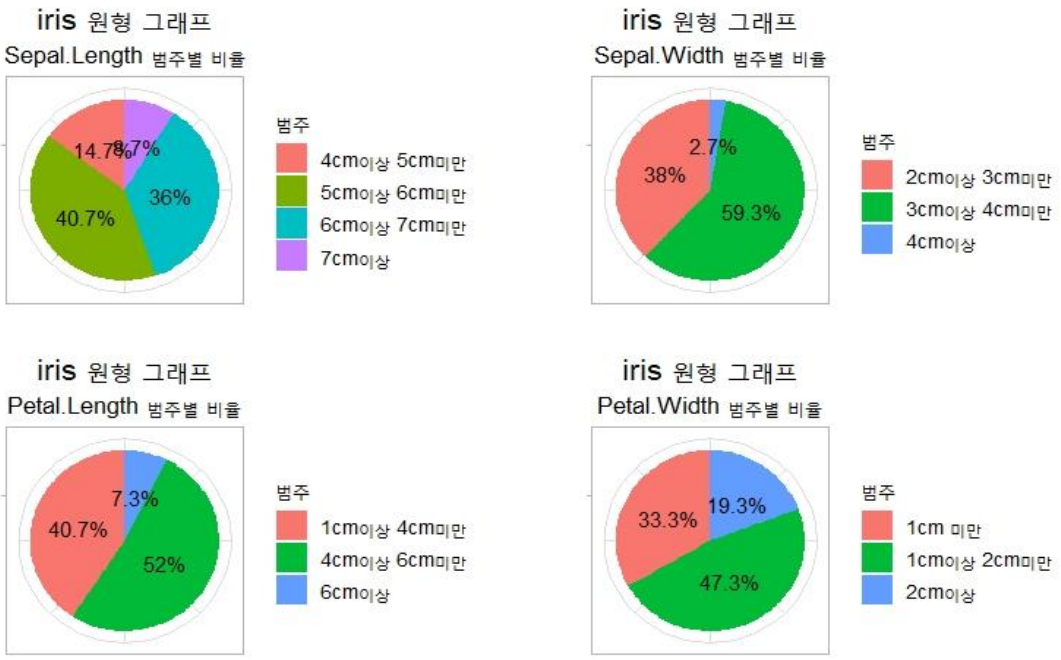
```r
visual_data2 <- visual_data
visual_data2

min(visual_data2$Sepal.Length)
max(visual_data2$Sepal.Length)
table(visual_data2$Sepal.Length)
visual_data2$Sepal.Length2[visual_data2$Sepal.Length >= 4 &
visual_data2$Sepal.Length <5] <- 1
visual_data2$Sepal.Length2[visual_data2$Sepal.Length >= 5 &
visual_data2$Sepal.Length <6] <- 2
visual_data2$Sepal.Length2[visual_data2$Sepal.Length >= 6 &
visual_data2$Sepal.Length <7] <- 3
visual_data2$Sepal.Length2[visual_data2$Sepal.Length >= 7 ] <-
4
pie_data1 <- as.data.frame(table(visual_data2$Sepal.Length2))
pie_data1

pie_data1$RF <- round(pie_data1$Freq/150, 3)
pie_data1

chart5 <- ggplot(pie_data1, aes(x="", y =RF, fill = Var1))+
  geom_bar(stat = "identity") + coord_polar("y") +
  theme_light() +

  geom_text(aes(label = paste0(round(RF*100,1),"%")),
            position = position_stack(vjust = 0.5)) +

  theme(text = element_text(size = 13),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5),
        axis.text.x = element_blank())+
  ylab("") + xlab("") + labs(subtitle = "Sepal.Length 범주별
비율", title = "iris 원형 그래프") +
  scale_fill_discrete(name = "범주", labels = c("4cm 이상
5cm 미만", "5cm 이상 6cm 미만", "6cm 이상 7cm 미만", "7cm 이상"))
```

python

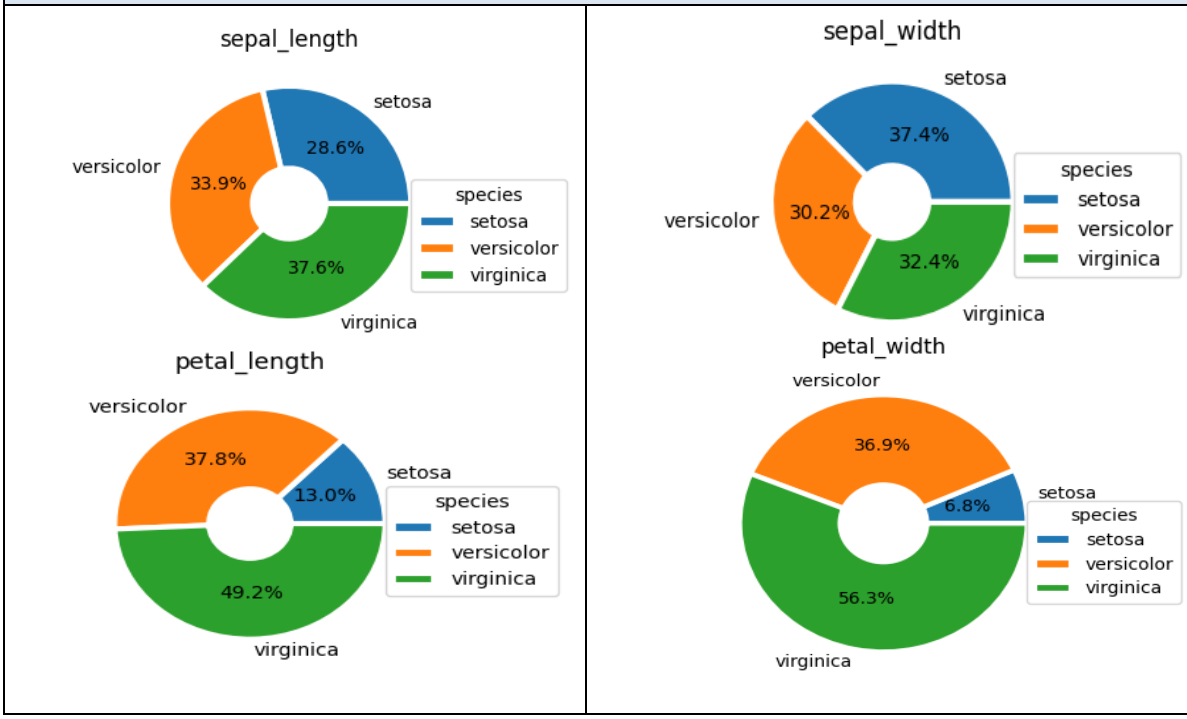
```python
plt.pie(iris_mean['sepal_length'],
  labels = iris_mean.index, autopct = '%.1f%%',
  wedgeprops = {'width':0.7, 'edgecolor':'w', 'linewidth' : 3})
plt.title('sepal_length')
plt.legend(loc = (0.9, 0.20), title = 'species')
```

ggplot2 에서 pie chart 를 그리려면, bar plot 을 만들어서 원형으로 변경해야 한다. 따라서 각 변수를 범주화하여 막대 차트를 생성하고 coord_polar 로 원형 차트를 생성하였다.

ggplot2 를 이용한 원형 차트



matplotlib 을 이용한 원형 차트



원형 차트의 경우에는 R 과 python 에서 다른 방식으로 데이터를 분석했다. R 으로는 iris 데이터에서 Species 를 제외한 4 개 변수를 범주화하여 원형 차트를 생성했다. python 으로는 꽃받침의 길이, 꽃받침의 너비, 꽃잎의 길이, 꽃잎의 너비를 비율로 비교했는데, 꽃받침의 사이즈는 28.6~37.6%로 10%이내의 차이를 보였고 꽃잎의 차이는 붓꽃별로 눈에 띄는 차이가 있음을 확인할 수 있었다.

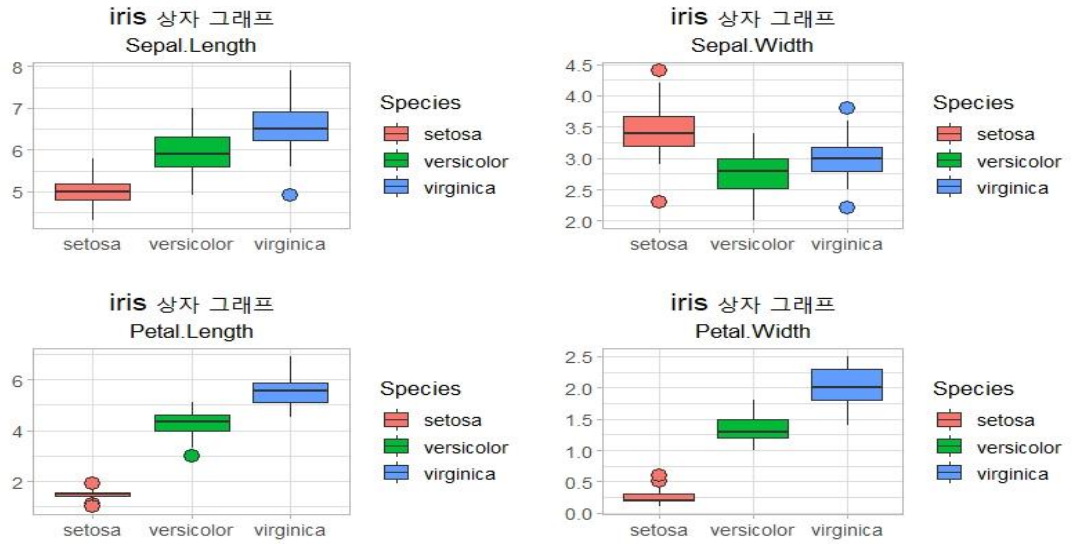
6) 상자 그래프

R	`chart9 <- ggplot(data = visual_data, aes(x = Species, y = Sepal.Length,`
	`fill = Species)) +`
	`theme_light()+`
	`theme(text = element_text(size = 13),`
	`axis.text.x = element_text(angle = 0, hjust =.5),`
	`plot.title = element_text(hjust = 0.5),`
	`plot.subtitle = element_text(hjust = 0.5))+`
	`geom_boxplot(aes(fill = Species), outlier.shape = 21, outlier.size= 4) +`
	`ggtitle("iris 상자 그래프") +`
	`ylab(" ") + xlab(" ") + labs(subtitle = "Sepal.Length")`

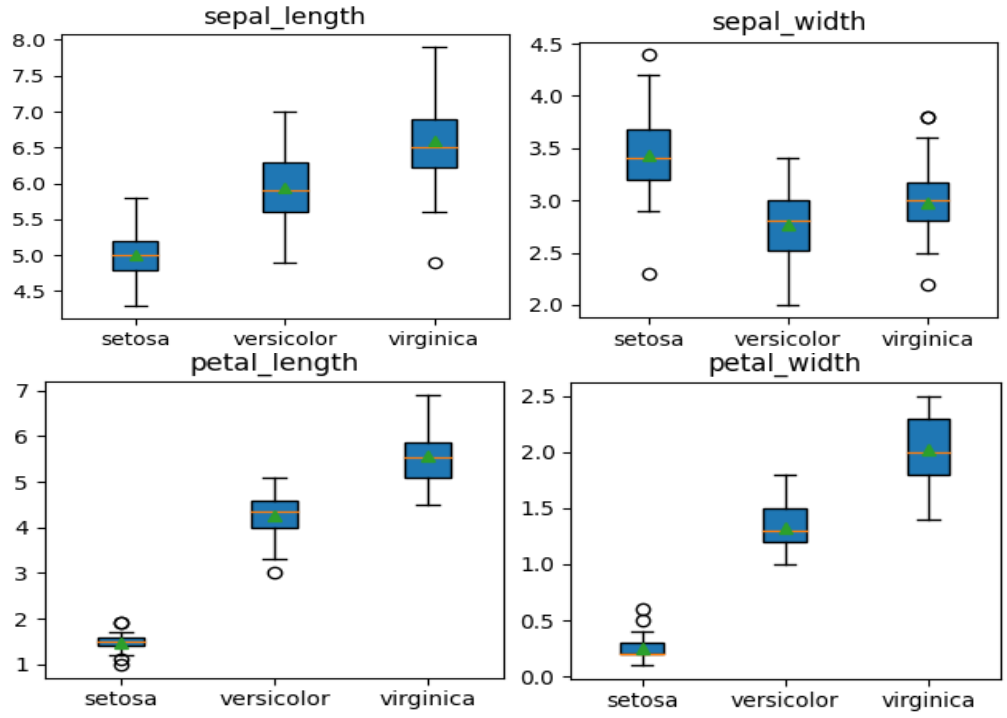
python	`data = [iris[iris['species']=="setosa"]['sepal_length'],`
	`iris[iris['species']=="versicolor"]['sepal_length'],`
	`iris[iris['species']=="virginica"]['sepal_length']]`
	`plt.boxplot(data, labels=['setosa', 'versicolor', 'virginica'],`
	`showmeans=True, patch_artist=True)`
	`plt.title('sepal_length')`

R 에서는 geom_boxplot 을 맵핑하여 상자 그래프를 생성했고, R 과 python 모두 각 변수를 종별로 분석했다.

ggplot2 를 이용한 상자 그래프



matplotlib 을 이용한 상자 그래프



위 그래프를 통해 iris 데이터 4 개 변수의 이상값, 중앙값을 확인할 수 있다.

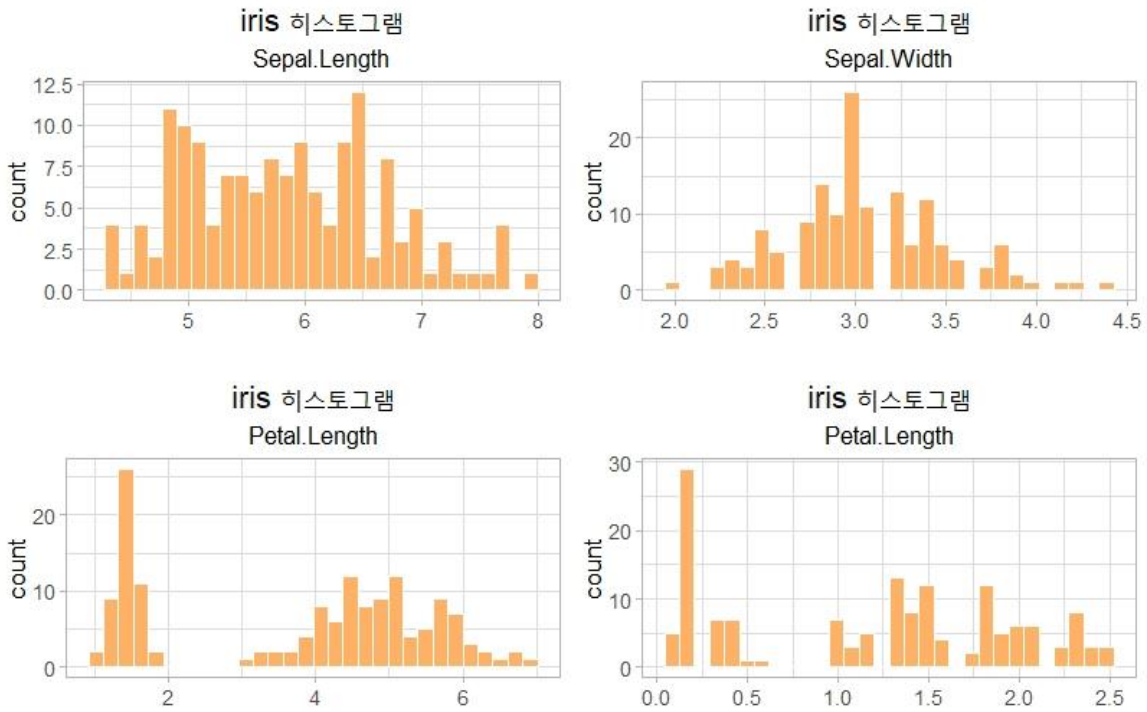
7) 히스토그램

R	`chart13 <- ggplot(visual_data, aes(x = Sepal.Length)) +`
	`  theme_light()+`
	`  geom_histogram(fill = "#FFB266", color = "white") +`
	`  theme(text = element_text(size = 13),`
	`  axis.text.x = element_text(angle = 0, hjust =.5),`
	`  plot.title = element_text(hjust = 0.5),`
	`  plot.subtitle = element_text(hjust = 0.5))+`
	`  ggtitle("iris 히스토그램") +`
	`  xlab(" ") + labs(subtitle = "Sepal.Length")`

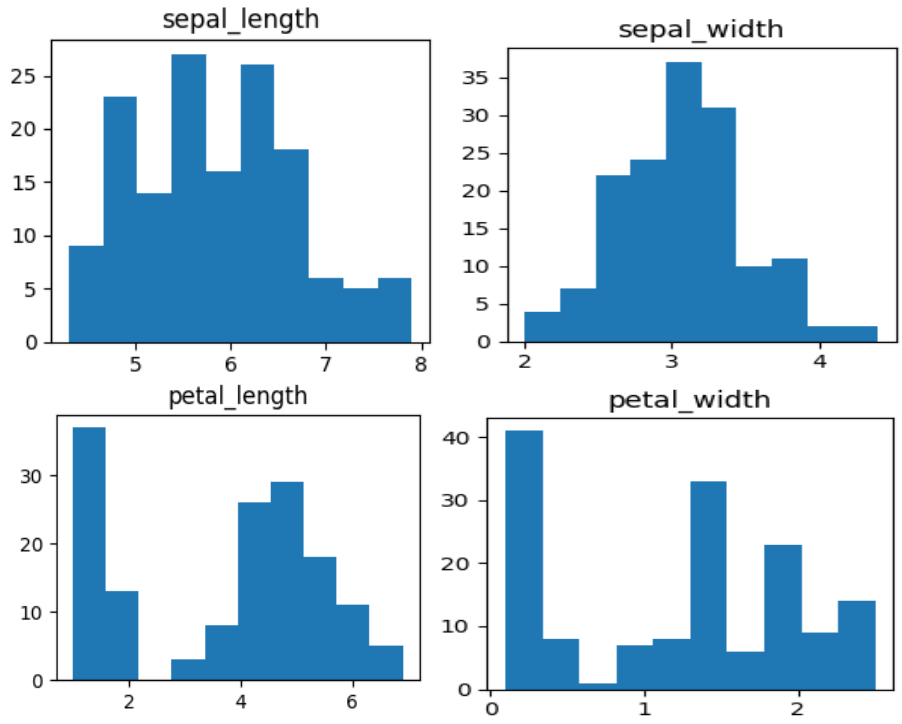
python	`plt.hist(iris['sepal_length'],bin=10)` `plt.title('sepal_length')`

R 의 geom_histogra, python 의 plt.hist 를 통해 변수별 히스토그램 차트를 생성하였다.

ggplot2 를 이용한 히스토그램



matplotlib 을 이용한 히스토그램



히스토그램을 통해 각 변수의 분포를 한 눈에 볼 수 있다.

8) 산점도

R

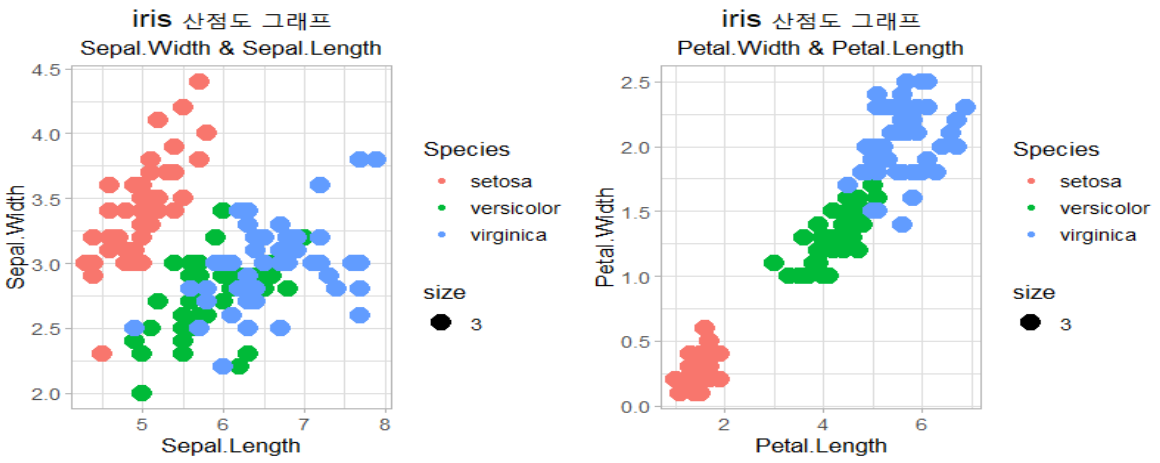
```r
chart17 <- ggplot(visual_data, aes(Sepal.Length, Sepal.Width, col = Species, size = 3))+
  theme_light()+
  theme(text = element_text(size = 13),
        axis.text.x = element_text(angle = 0, hjust =.5),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))+
  labs(title = "iris 산점도 그래프", subtitle = "Sepal.Width & Sepal.Length") +
  geom_point()
```

python

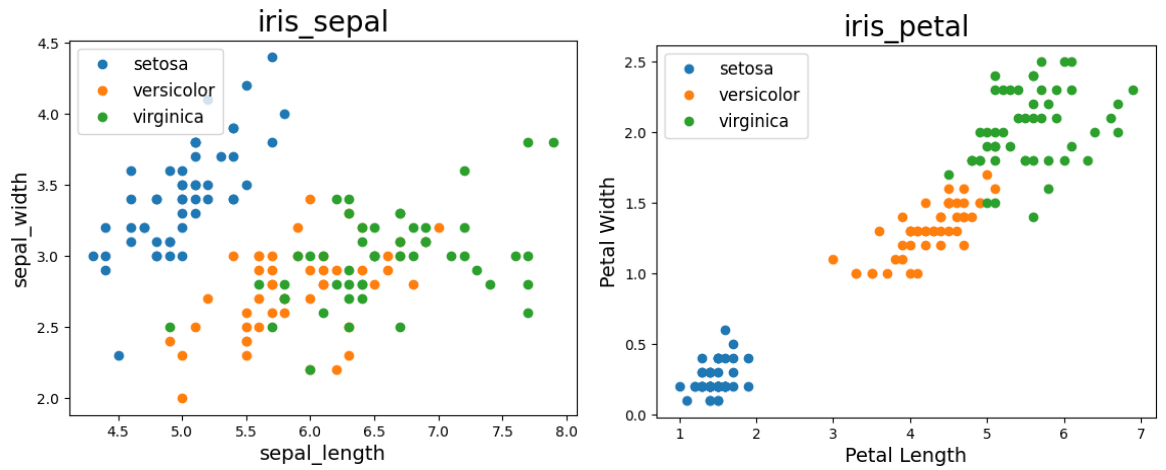
```python
groups = iris.groupby('species')
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.sepal_length,
            group.sepal_width,
            marker='o',
            linestyle='',
            label=name)
ax.legend(fontsize=12, loc='upper left')
plt.title('iris_sepal', fontsize=20)
plt.xlabel('sepal_length', fontsize=14)
plt.ylabel('sepal_width', fontsize=14)
```

Sepal.Width 를 y 축으로 Sepal.Length 를 x 축으로 하여 Species 별로 색을 다르게 하여 ggplot2 의 geom_point 로 시각화 했고, python 에서 역시 동일 변수를 그룹화하여 산점도 그래프를 생성했다.

ggplot2 를 이용한 산점도



matplotlib 을 이용한 산점도



9) 중첩자료 시각화

R

```
irisdata <- data.frame(table(visual_data$Sepal.Length, visual_data$Sepal.Width))
names(irisdata) <- c("Sepal.Length", "Sepal.Width", "freq")

chart19 <- ggplot(data = irisdata, aes(x = Sepal.Length, y = Sepal.Width)) +
  scale_size(range = c(1,20), guide = 'none') +
  geom_point(colour = "grey90", aes(size =freq)) +
  geom_point(aes(colour=freq, size=freq))+
  scale_colour_gradient(low="5", high = "blue")+
  geom_smooth(formula = y~x, method = lm, se = FALSE) +
  theme_light()+

  theme(text = element_text(size = 13),
        axis.text.x = element_text(angle = 0, hjust =.5),
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))+
  labs(title = "iris 중첩자료", subtitle = "Sepal.Width & Sepal.Length")
```

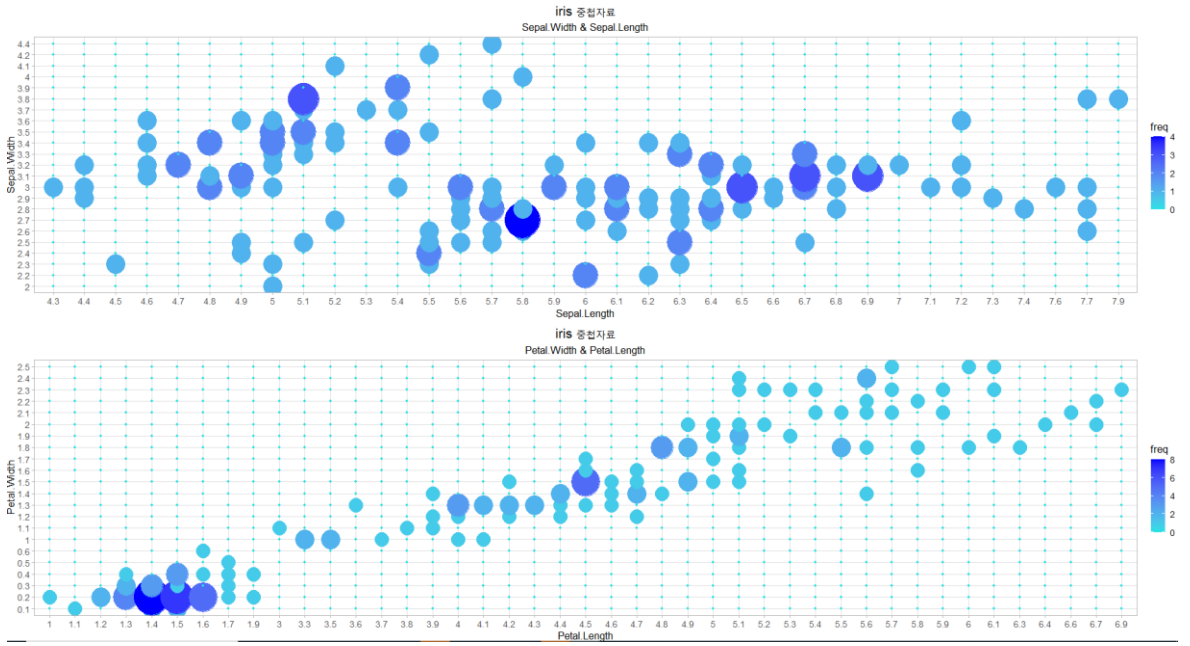
python

```
iris_sepla = iris.groupby(['sepal_length','sepal_width'])
sepal = iris_sepla.size()
sepal = sepal.reset_index(name="count")
plt.scatter(sepal["sepal_length"], sepal["sepal_width"],
s=  sepal["count"]*100,  c=sepal["count"],  cmap='Wistia',
alpha=0.8)
plt.title('iris_sepal', fontsize=20)
```

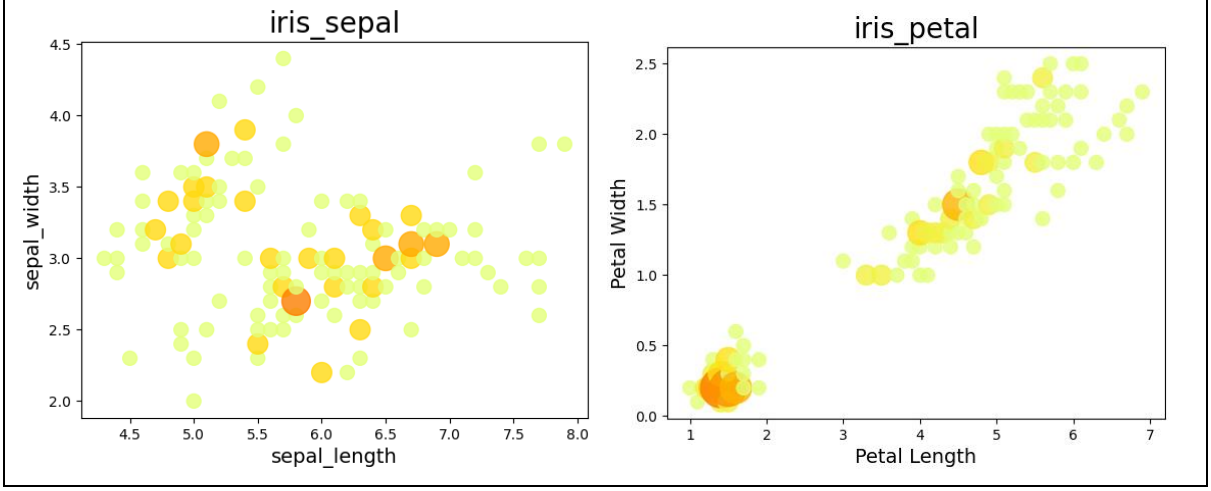
```
plt.xlabel('sepal_length', fontsize=14)
plt.ylabel('sepal_width', fontsize=14)
```

ggplot2 로 Sepal.Width, Sepal.Length 의 중첩 빈도수를 irisdata 에 담아준다. x 축을 Sepal.Length 으로 y 축을 Sepal.Width 로 하여 산점도를 작성했다. 이때 geom_point 의 크기를 빈도수가 높을수록 사이즈가 크도록 하였고, scale_colour_gradient 를 통해 빈도수가 높을수록 색이 변하도록 그라디언트 효과를 주었다. (결과표 위 그래프가 Sepal.Width&Sepal.Length, 아래가 Petal.Width&Petal.Length 이다.) matplotlib 에서는 scatter 함수의 옵션을 조절하여 중첩자료를 시각화 했다.

ggplot2 를 이용한 중첩 자료 시각화



matplotlib 을 이용한 중첩 자료 시각화



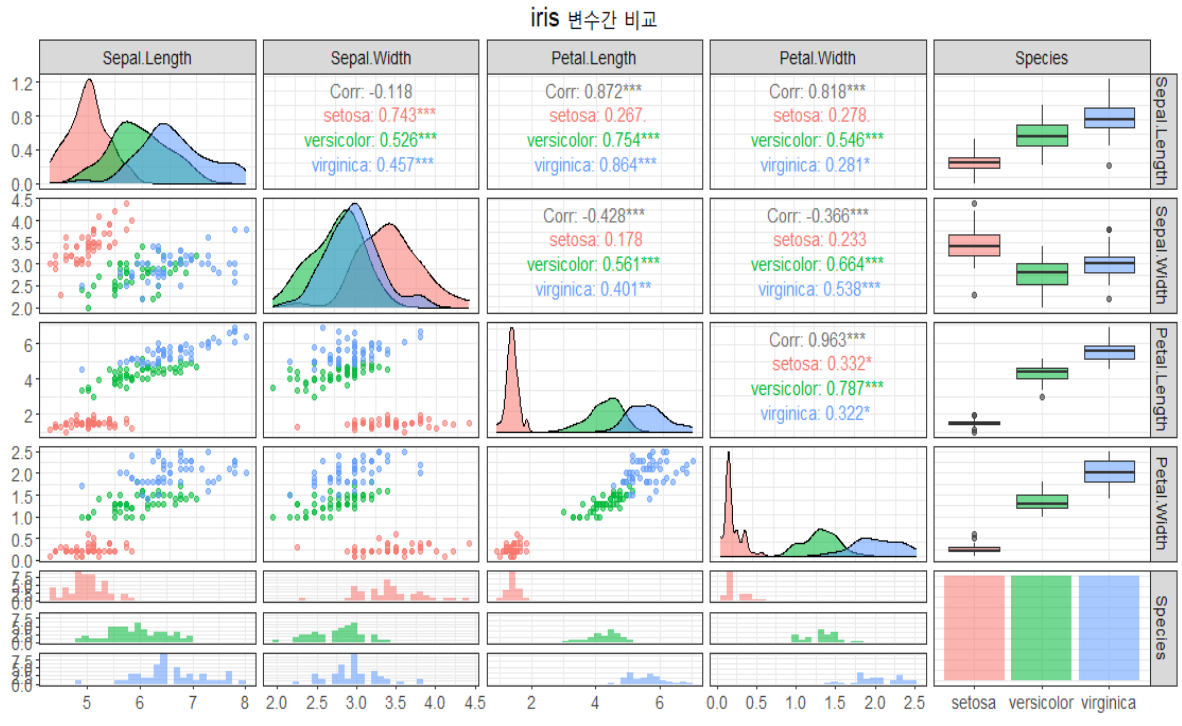
10) 변수간의 비교 시각화, 밀도 그래프

R	`chart21 <- ggpairs(visual_data,`
	`aes(color = Species,`
	`alpha = 0.5)) + theme_bw()+`
	`theme(text = element_text(size = 13),`
	`axis.text.x = element_text(angle = 0, hjust =.5),`
	`plot.title = element_text(hjust = 0.5)) +`
	`labs(title = "iris 변수간 비교")`

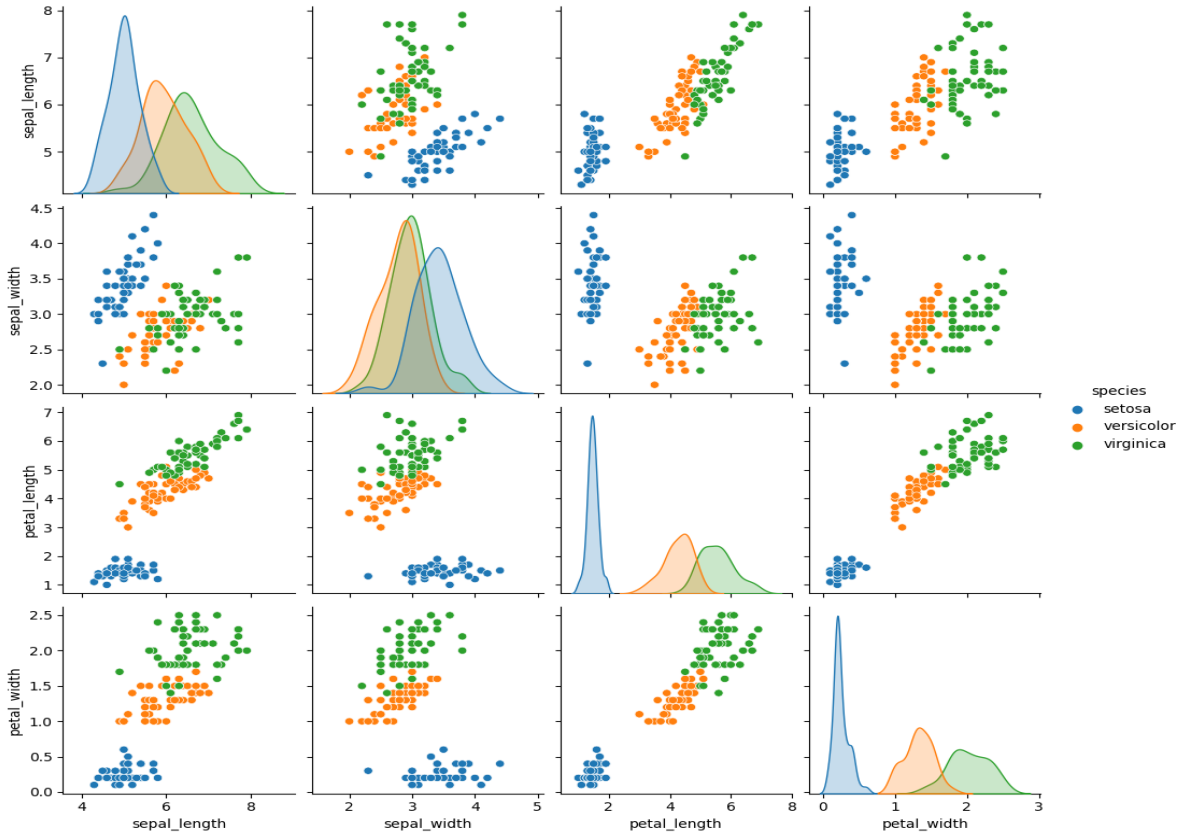
python	`sns.pairplot(iris, hue="species")`

R 에서는 Pairs 를 사용하기 위해 ggplot2 를 extend 해주는 패키지인 GGally 를 설치하였다. ggpairs 함수를 통해 밀도함수, 산점도, 상관계수를 출력하고 유의성을 검정하였다. 또한 python 에서도 matplotlib 을 기반으로 하는 seaborn 패키지를 사용하여 보다 효율적인 시각화 방법을 탐색했다.

ggpairs 를 이용한 변수간 비교 시각화 및 밀도 그래프



Seaborn 을 이용한 변수 간 시각화 및 밀도 그래프



Species 기준으로 변수 간의 비교 및 밀도 그래프 시각화를 실시했다. 이를 통해 꽃잎의 길이와 너비가 가장 관계가 높은 것으로 보이며 꽃받침의 길이와 꽃잎의 너비가 관계성이 적은 것으로 확인된다.

8. 부록

1) 데이터 참조

- 시계열 분석: USD KRW.csv
- 분류분석: wisc_bc_data.csv
- 예측분석: R 의 melbench 패키지, python 의 sklearn 패키지 내 Bostonhousing dataset
- 군집분석: ggplot2 패키지 내 diamonds 데이터
- 텍스트 분석: zelensky.txt
- 시각화: iris 데이터

2) 첨부파일

B4_pythoncode.py

B4_rcode.r

→ code 및 주석