

1. 군집화 분석 수행

- **Absenteeism at work dataset**
- Clustering task를 위한 numerical attribute type의 데이터를 조건으로 설정하였습니다. Attribute는 10~100개, instance는 100~1000개 사이를 원한다는 데이터 조건도 추가하였더니 옆 사진과 같이 11개의 데이터 셋이 나왔습니다.
- 그 중, 직장 결근(absenteeism at work)과 관련된 데이터 셋이 흥미로운 주제라고 생각하여 선택하였습니다.
- 군집화 분석 수행 과제를 ver1과 ver2로 진행하였습니다. 두 버전의 코드와 실행화면을 설명할 예정입니다.

Browse Through: 11 Data Sets

Default Task - Undo	Attribute Type - Undo	Data Type	Area	# Attributes - Undo	# Instances - Undo	Format Type	Name
Classification (62)	Categorical (0)	Multivariate (10)	Life Sciences (2)	Less than 10 (8)	Less than 100 (4)	Matrix (11)	UCI Absenteeism at work
Regression (23)	Numerical (11)	Univariate (0)	Physical Sciences (1)	10 to 100 (11)	100 to 1000 (11)	Non-Matrix (0)	UCI Dow Jones Index
Clustering (11)	Mixed (0)	Sequential (1)	CS / Engineering (0)	Greater than 100 (6)	Greater than 1000 (24)		UCI HCV data
Other (1)		Time-Series (3)	Social Sciences (0)				UCI Heart failure clinical records
		Text (1)	Business (4)				Libras Movement
		Domain-Theory (0)	Game (0)				UCI Sales Transactions Dataset Weekly
		Other (0)	Other (3)				UCI South German Credit
							UCI South German Credit (UPDATE)
							UCI Tennis Major Tournament Match Statistics
							UCI Travel Reviews
							UCI Water Treatment Plant

군집화 분석 수행 Ver.1

- 필요한 패키지, 라이브러리 설치
- 데이터 불러오기 (csv) ➡ data
- describe(), isnull(), dtypes 실행
- → 결측치 없음

기계학습 과제#1

Absenteeism at work dataset - EDA & Machine Learning Model

```
In [1]: # 설치되지 않은 라이브러리/모듈은, conda prompt에서 pip install ~

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import squarify

# import 클러스터링 위한 라이브러리
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.cluster.hierarchy import linkage
from sklearn.cluster.hierarchy import dendrogram
from sklearn.cluster.hierarchy import cut_tree
```

```
In [4]: # 결측치 확인
data.isnull().sum().sort_values(ascending = False)

Out[4]: ID 0
Disciplinary failure 0
Body mass index 0
Height 0
Weight 0
Pet 0
Social smoker 0
Social drinker 0
Son 0
Education 0
Hit target 0
Reason for absence 0
Work load Average/day 0
Age 0
Service time 0
Distance from Residence to Work 0
Transportation expense 0
Seasons 0
Day of the week 0
Month of absence 0
Absenteeism time in hours 0
dtype: int64
```

1. 데이터 선택

```
In [2]: # 데이터 불러오기
data = pd.read_csv('Absenteeism_at_work.csv', delimiter = ',')
data.head()
```

```
Out[2]:
```

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Disciplinary failure	Education	Son	Social drinker	Social smoker	Pet	Wkg
0	11	26	7	3	1	289	36	13	33	239.554	...	0	1	2	1	0	1
1	36	0	7	3	1	118	13	15	50	239.554	...	1	1	1	1	0	0
2	3	23	7	4	1	179	51	18	38	239.554	...	0	1	0	1	0	0
3	7	7	7	5	1	279	5	14	39	239.554	...	0	1	2	1	1	0
4	11	23	7	5	1	289	36	13	33	239.554	...	0	1	2	1	0	1

5 rows * 21 columns

```
In [3]: data.describe()
```

```
Out[3]:
```

	ID	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Disciplinary failure	Education
count	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000
mean	18.017568	19.216218	6.324324	3.914885	2.544566	221.320730	20.631081	12.554054	36.450000	271.490235	0.054054	1.2918
std	11.021247	8.433406	3.436287	1.421675	1.11831	66.952223	14.836789	4.364873	6.478772	39.058116	0.228277	0.6732
min	1.000000	0.000000	0.000000	2.000000	1.000000	118.000000	5.000000	1.000000	27.000000	255.917000	0.000000	1.0000
25%	9.000000	13.000000	3.000000	3.000000	2.000000	179.000000	16.000000	9.000000	31.000000	244.387000	0.000000	1.0000
50%	18.000000	20.000000	6.000000	4.000000	3.000000	225.000000	20.000000	13.000000	37.000000	264.204000	0.000000	1.0000
75%	28.000000	26.000000	9.000000	5.000000	4.000000	269.000000	30.000000	16.000000	40.000000	294.217000	0.000000	1.0000
max	36.000000	28.000000	12.000000	6.000000	4.000000	388.000000	52.000000	29.000000	58.000000	378.894000	1.000000	4.0000

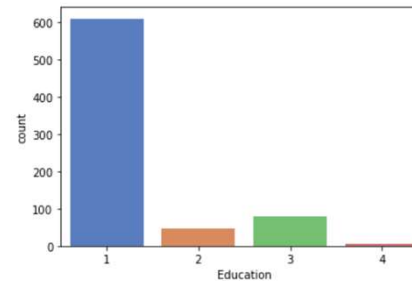
8 rows * 21 columns

```
In [5]: data.dtypes

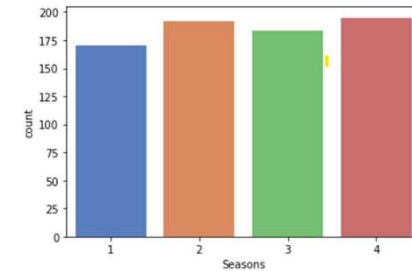
Out[5]: ID int64
Reason for absence int64
Month of absence int64
Day of the week int64
Seasons int64
Transportation expense int64
Distance from Residence to Work int64
Service time int64
Age int64
Work load Average/day float64
Hit target int64
Disciplinary failure int64
Education int64
Son int64
Social drinker int64
Social smoker int64
Pet int64
Wkg int64
dtype: object
```

- data의 Education, Seasons attribute 에 대한 countplot 그리기
- Month of absence vs Seasons countplot 그리기
- pairplot 그려보기
- pairplot : 데이터의 각 컬럼(열)들의 모든 상관관계 출력, 그리드(grid) 형태로 각 집합의 조합에 대해 히스토그램과 분포도 그린다. 3차원 이상의 데이터라면 pairplot 함수를 사용해 분포도 그린다.

Out[6]: <AxesSubplot:xlabel='Education', ylabel='count'>

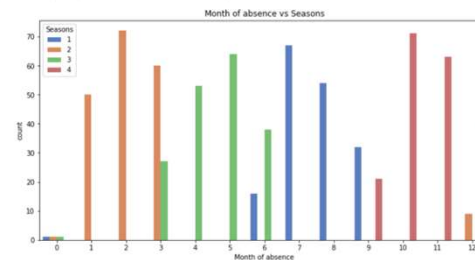


Out[7]: <AxesSubplot:xlabel='Seasons', ylabel='count'>

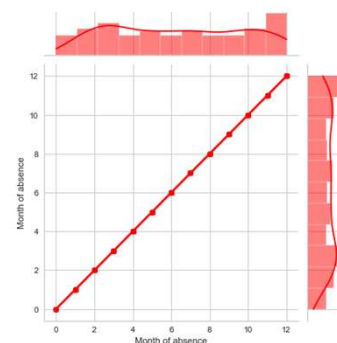
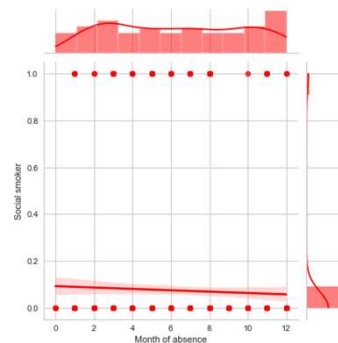
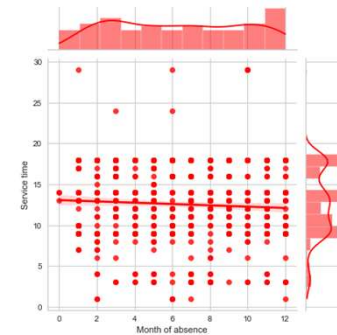


```
In [8]: # x축 : Month of absence
# y축 : Seasons
plt.figure(figsize = (12,8))
sns.countplot(x = 'Month of absence', hue = 'Seasons', data = data,palette = 'muted')
plt.title('Month of absence vs Seasons')
```

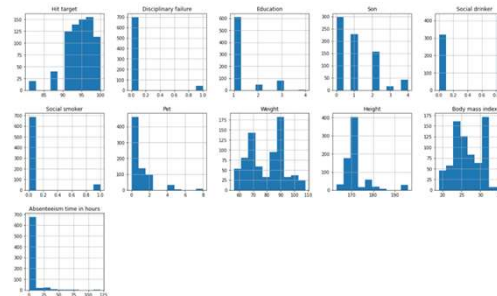
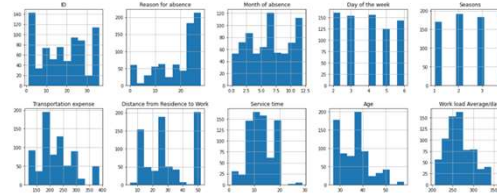
Out[8]: Text(0.5, 1.0, 'Month of absence vs Seasons')



- 변수 간 상관관계 확인 → Joint plot (Month of Absence with Other Variables) 그려보기
- $i \neq \text{'Glucose'}$ and $i \neq \text{'Outcome'}$
- `print(f"Correlation between Month of absence and {i} ==> ", data.corr().loc['Month of absence'][i])`
 ➡ 변수 간 상관성 문장 출력
- 총 21개의 그래프와 상관성 도출, 그 중 일부



- 데이터 분포 관찰 위한 시각화 진행 (hist)
- scatterplot 함수 정의
- → scatterplot 그려보기
- data의 column 확인
- customized_scatterplot(data['Social drinker'], data['Weight'])



```

In [12]: data['Social drinker'].value_counts()
Out[12]:
0    320
Name: Social drinker, dtype: int64

In [13]: from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go

# To plot : array의 list를 만들
vals = data['Social drinker'].value_counts().tolist()
labels = [0,1]

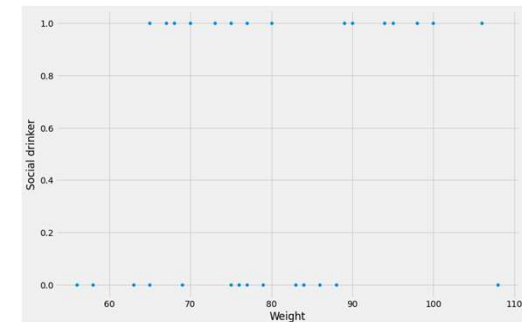
df = [go.Bar(x = labels, y = vals, marker = dict(color='#7030A0'))]

layout = go.Layout(title = "Count by Social drinker")
fig = go.Figure(data = df, layout = layout)
iplot(fig, filename = "basic-bar")

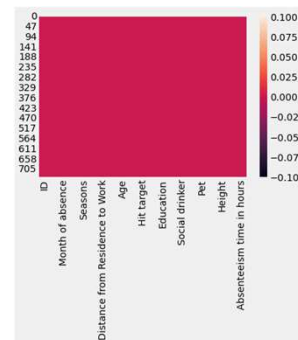
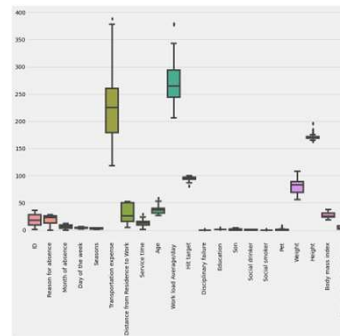
In [14]: # 함수 정의
def customized_scatterplot(y, x):
    # plot setting
    style.use('f10eth/f1weight')
    plt.subplots(figsize = (12,8))
    # 0.0 ~ 0.4까지 0.1씩 증가
    sns.scatterplot(y = y, x = x);

In [15]: data.columns
Out[15]: Index(['ID', 'Reason for absence', 'Month of absence', 'Day of the week',
              'Seasons', 'Transportation expense', 'Distance from Residence to Work',
              'Service time', 'Age', 'Work load Averages', 'Hit target',
              'Disciplinary failure', 'Education', 'Sex', 'Social drinker',
              'Social smoker', 'Pet', 'Weight', 'Height', 'Body mass index',
              'Absence time in hours'],
              dtype='object')

```



- outlier 확인 위해 boxplot 사용
- 누락/null 값 확인 위해 heatmap 사용
- data의 outlier를 drop한 data_o
- → data의 shape (740,21)에 비해 data_o의 shape (576,21) 줄어들었다!



```

in [20]: # Drop outliers -> according to p-score
from scipy import stats
z = np.abs(stats.zscore(data))
print(z)

threshold = 3
data_o = data[z < 3].all(axis=1)

ID      Rescor for absence  Month of absence  Day of the week
0      0.637761      0.004938      0.196763      0.642847
1      1.632719      2.290124      0.196763      0.642847
2      1.362623      0.448970      0.196763      0.659824
3      1.000342      1.449530      0.196763      0.763796
4      0.637761      0.448970      0.196763      0.763796

Seasons  Transportation expense  Distance from Residence to Work  Service time  Age  Work load Average/day  Hit target  Education  Son  Social drinker  Social smoker  Pet  Weight  Height  Absenteeism time in hours
735  0.637761      0.618987      0.196763      0.642847
736  1.545113      0.974965      0.196763      0.642847
737  1.272729      2.290124      1.841698      0.642847
738  0.906547      2.290124      1.841698      0.659824
739  1.541923      2.290124      1.841698      1.467687

Disciplinary failure  Education  Son  Social drinker  Social smoker
0      0.239048      0.439857      0.897232      0.872872      0.269566
1      0.239048      0.439857      0.897232      0.872872      0.269566
2      0.239048      0.439857      0.897232      0.872872      0.269566
3      0.239048      0.439857      0.897232      0.872872      0.269566
4      0.239048      0.439857      0.897232      0.872872      0.269566

Pet  Weight  Height  Body mass index  Absenteeism time in hours
0      0.102650      0.851673      0.019048      0.775802      0.219511
1      0.556240      1.472056      0.975829      1.009438      0.519787
2      0.556240      0.774000      0.260771      1.056438      0.369558
3      0.556240      0.857131      0.682395      0.825100      0.219511
4      0.102650      0.851673      0.019048      0.775802      0.269566

Seasons  Transportation expense  Distance from Residence to Work
0      1.390175      1.011408      0.429556
1      1.390175      1.544376      1.171694
2      1.390175      0.632695      1.441240
3      1.390175      0.907947      1.891250
4      1.390175      1.011408      0.429556

Pet  Weight  Height  Body mass index  Absenteeism time in hours
735  0.102650      0.851673      0.019048      0.775802      0.269566
736  0.102650      0.851673      0.019048      0.775802      0.269566
737  0.556240      1.472056      0.975829      1.009438      0.519787
738  0.556240      0.774000      0.260771      1.056438      0.369558
739  0.102650      0.851673      0.019048      0.775802      0.269566

```

```

[740 rows x 21 columns]
array([[ 1,  3,  8, 27, 31, 31, 33, 33, 38, 38, 44, 50, 51,
        54, 55, 59, 64, 83, 85, 89, 91, 99, 104, 116, 140, 167,
        169, 184, 185, 187, 189, 172, 180, 185, 189, 187, 199, 200, 203,
        203, 203, 205, 206, 210, 213, 213, 214, 215, 215, 218, 226, 231,
        232, 234, 236, 242, 245, 251, 253, 255, 256, 257, 270, 271, 272,
        273, 273, 274, 275, 276, 277, 277, 278, 278, 280, 281, 282,
        283, 284, 285, 286, 286, 287, 288, 293, 294, 299, 300, 303, 311,
        312, 313, 313, 322, 323, 323, 326, 336, 336, 337, 357, 360,
        362, 364, 396, 397, 400, 405, 406, 407, 420, 421, 432, 433, 434,
        446, 468, 468, 469, 471, 479, 481, 488, 490, 507, 510, 513, 521,
        530, 548, 549, 551, 562, 584, 588, 593, 572, 573, 576, 576, 577,
        579, 592, 604, 601, 603, 608, 620, 622, 622, 640, 647, 648, 652,
        661, 670, 675, 681, 682, 682, 683, 689, 690, 691, 692, 692,
        695, 702, 703, 704, 706, 710, 712, 714, 714, 714, 715, 721, 724,
        726, 727, 729, 729, 734, 737], dtype=int64), array([15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
        15, 18, 18, 15, 20, 20, 18, 18, 15, 15, 15, 15, 15, 15, 15, 15,
        18, 20, 18, 15, 11, 15, 18, 18, 18, 18, 15, 15, 11, 15, 11, 15, 11, 18,
        20, 15, 7, 18, 18, 15, 11, 15, 8, 15, 15, 10, 10, 10, 11, 10,
        10, 10, 11, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
        11, 11, 20, 11, 11, 11, 11, 15, 15, 18, 20, 11, 11, 15, 11, 15,
        15, 18, 18, 15, 15, 15, 11, 11, 11, 11, 20, 15, 15, 15, 4, 11, 15,
        15, 18, 20, 15, 15, 15, 18, 7, 7, 8, 11, 11, 11, 11, 15, 16,
        18, 20, 15, 15, 15, 7, 10, 18, 12, 15, 18, 18, 8, 8, 20, 8,
        15, 18, 20, 15, 18, 12, 12, 15, 20, 12, 11, 11, 15, 15, 16, 20, 15,
        18, 15, 15, 15, 15, 16, 11, 15, 16, 11, 18, 18, 15, 8, 8, 20, 20,
        16]), dtype=int64))

data.shape
(740, 21)

data_o.shape
# 740 -> 576 줄어듦!
(576, 21)

```

- 정규화 진행 Normalization
- scaler에 data_o 채우고 normalize 진행, numpy array → DataFrame 변환
- 정규화된 데이터 boxplot 그리기
- heatmap 그리기
- 높은 상관성을 갖는 feature 도출

```
In [23]: # Normalize
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, normalize

# 할 일
names = data_o.columns

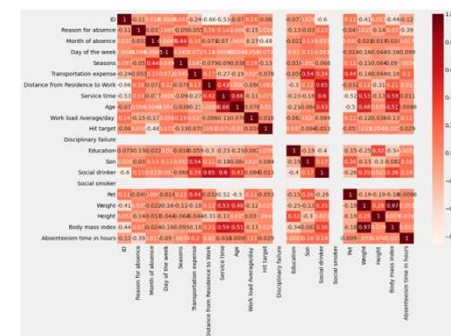
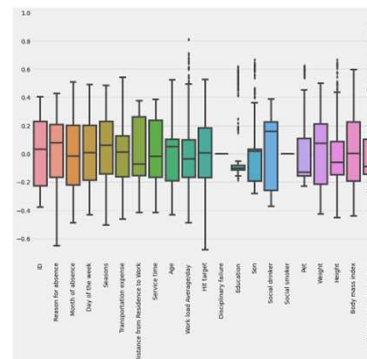
# Create Scaler object
scaler = preprocessing.StandardScaler()

# Fit data on the scaler object
scaled_df = scaler.fit_transform(data_o)

# Normalizing the Data
normalized_df = pd.DataFrame(scaled_df, columns=names)

# Converting the numpy array -> pandas DataFrame
normalized_df = pd.DataFrame(normalized_df, columns=names)

In [24]: plt.figure(figsize = (18,10))
sns.boxplot(data = normalized_df)
plt.xticks(rotation = 90)
```



```
In [26]: # Correlation with output variable
cor_target = abs(cor["Absenteeism time in hours"])

# Selecting highly correlated features
relevant_features = cor_target[cor_target>0.5]
relevant_features

Out[26]: Absenteeism time in hours    1.0
         Name: Absenteeism time in hours, dtype: float64
```

- 1-1. **LassoCV 회귀분석** 진행
- “정규화 선형 회귀의 일종으로, 선형 회귀 계수에 대한 제약 조건을 추가하여 모델이 과도하게 최적화하는 현상 (과적합 overfitting)을 막는 분석 방법”
- feature 선택 후 LassoCV() 진행
- best alpha, score? (LassoCV) 도출
- variable, eliminated variables? 도출
- Lasso 모델에서의 feature importance plot
- X에서 attribute drop → X 갱신

Modeling

LassoCV 라소 회귀분석

```
In [27]: # Feature Selection using LassoCV
# 라소 회귀분석

from sklearn.linear_model import LassoCV

# Feature Selection
X = normalized_df.drop("Reason for absence", axis=1) # Feature Matrix
y = normalized_df["Reason for absence"] # Target Variable

reg = LassoCV()
reg.fit(X, y)
print("Best alpha using built-in LassoCV: %f" % reg.alpha_)
print("Best score using built-in LassoCV: %f" % reg.score(X, y))
coef = pd.Series(reg.coef_, index = X.columns)

Best alpha using built-in LassoCV: 0.000676
Best score using built-in LassoCV: 0.231187

In [28]: print("Lasso picked %d * %d variables and eliminated the other %d * %d variables."
Lasso picked 12 variables and eliminated the other 8 variables.
```

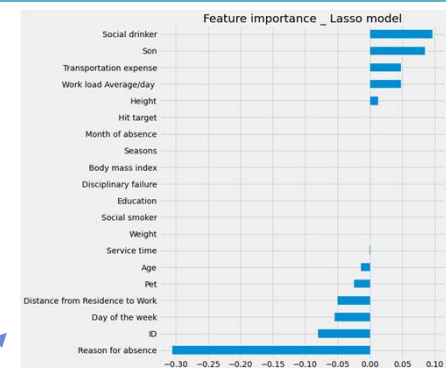
```
In [29]: imp_coef = coef.sort_values()

import matplotlib

matplotlib.rcParams['figure.figsize'] = (8, 10)
imp_coef.plot(kind = "barh")

plt.title("Feature importance _ Lasso model")

Out[29]: Text(0.5, 1.0, 'Feature importance _ Lasso model')
```



```
In [30]: X=X.drop(['ID', 'Reason for absence', 'Month of absence', 'Day of the week',
'Seasons', 'Distance from Residence to Work',
'Service time', 'Age', 'Hit target',
'Disciplinary failure', 'Education',
'Social smoker', 'Pet', 'Weight', 'Body mass index'], axis = 1)
```

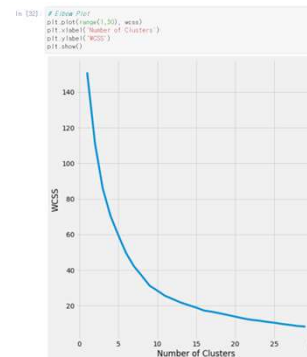

1-2. K-Means Clustering ① 진행

- “비지도 학습의 클러스터링 모델 중 하나, K는 클러스터의 개수, K-means Clustering의 목적은 유사한 데이터 포인트끼리 grouping 하여 패턴을 찾아내는 것”
- Elbow method : works by finding WCSS
- “**WCSS(Within-Cluster Sum of Square)** is the sum of the squared distance between each point and the centroid in a cluster.”
- elbow plot 그리기 + silhouette score elbow kmeans clustering plot
- !pip install -U scikit-learn
- 오류 해결 과정에서 stackoverflow 참고!

```
K-Means Clustering
In [31]: # K-Means Clustering
from sklearn.cluster import KMeans

# WCSS Elbow point 찾기
wcss = []
for i in range(1, 30):
    kmeans = KMeans(i)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

wcss
```

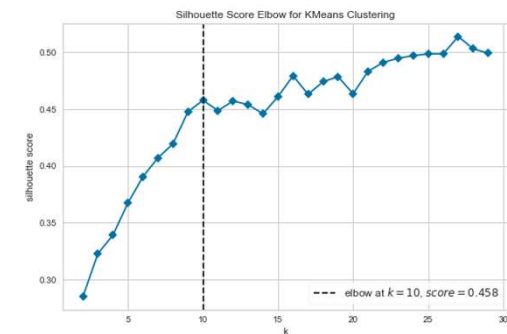


```
In [34]: # Another Technique ~ n_cluster 정의
# KElbowVisualizer method import
from yellowbrick.cluster import KElbowVisualizer

# scikit-learn K-Means model 인스턴스화
model = KMeans(random_state=42)

# KElbowVisualizer 인스턴스화 <- clusters 수, metric, timings
visualizer = KElbowVisualizer(model, k=(2, 30), metric='silhouette', timings=False)

# Fit the data and visualize
visualizer.fit(X)
visualizer.poof()
```



- k_means_new 👁 10개로 군집화
- cluster_new, fit_predict
- 클러스터 모양 3d 시각화
- cluster_new data로 plot 그리기
- lplot : seaborn.lplot, column 간의 선형관계 확인에 용이, 각 데이터셋의 분포 (산점도)와 회귀선을 동시에 나타내도록 시각화 출력 가능

```
In [35]: k_means_new = KMeans(10)
kmeans.fit(X)
cluster_new = X.copy()
cluster_new['cluster_pred'] = k_means_new.fit_predict(X)
cluster_new.head()
```

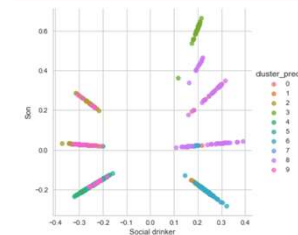
```
In [36]: # Visualize cluster shapes (3d)
```

```
cluster1=cluster_new.loc[cluster_new['cluster_pred'] == 0]
cluster2=cluster_new.loc[cluster_new['cluster_pred'] == 1]
cluster1=cluster_new.loc[cluster_new['cluster_pred'] == 2]
cluster2=cluster_new.loc[cluster_new['cluster_pred'] == 3]
cluster1=cluster_new.loc[cluster_new['cluster_pred'] == 4]
cluster2=cluster_new.loc[cluster_new['cluster_pred'] == 5]
cluster1=cluster_new.loc[cluster_new['cluster_pred'] == 6]
cluster2=cluster_new.loc[cluster_new['cluster_pred'] == 7]
cluster1=cluster_new.loc[cluster_new['cluster_pred'] == 8]
cluster2=cluster_new.loc[cluster_new['cluster_pred'] == 9]
```

```
Out[35]:
```

	Transportation expense	Work load Average/day	Son	Social drinker	Height	cluster_pred
0	0.328957	-0.254351	0.324561	0.294374	0.120332	8
1	-0.177749	-0.214543	-0.217476	0.248302	-0.069948	6
2	0.329408	-0.254699	0.325005	0.294778	0.120497	8
3	-0.165427	-0.199670	-0.202400	0.231089	-0.065099	6
4	0.377743	-0.140836	0.018475	0.162997	0.066629	8

```
In [37]: import seaborn as sns
# plot data with seaborn
facet = sns.lplot(data = cluster_new, x = 'Social drinker', y = 'Son',
hue = 'cluster_pred', fit_reg = False, legend = True, legend_out = True)
C:\Users\jwoo\anaconda3\lib\site-packages\seaborn\regression.py:592: UserWarning:
legend_out is deprecated from the "lplot" function signature. Please update your code to pass
```

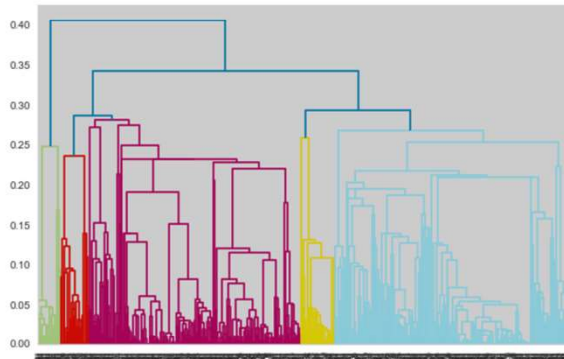


Hierarchical Clustering

```
In [38]: import scipy.cluster.hierarchy as hcluster
from sklearn.cluster import AgglomerativeClustering

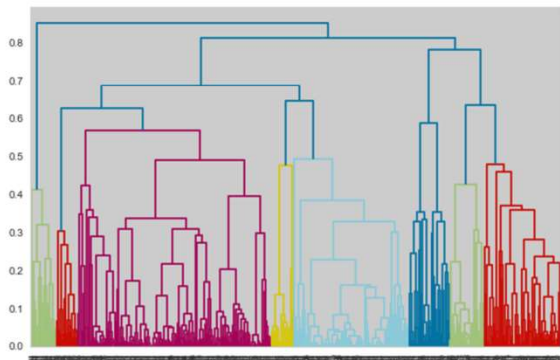
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree

# single linkage
mergings = linkage(X, method = "single", metric = 'euclidean')
dendrogram(mergings) # 덴드로그램
plt.show()
```



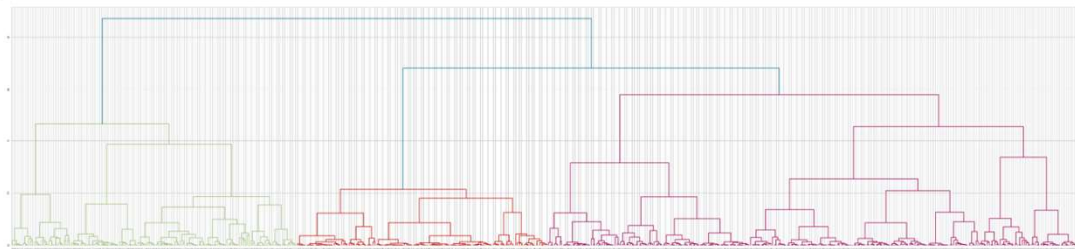
```
In [39]: features = normalized_df.columns
```

```
In [40]: # average linkage
mergings = linkage(X, method = "average", metric = 'euclidean')
dendrogram(mergings)
plt.show()
```



- 1-3. Hierarchical clustering ② 진행
- “계층적 트리 모델을 이용해 개별 개체들을 순차적, 계층적으로 유사한 개체 내지 그룹과 통합하여 군집화를 수행하는 알고리즘”, dendrogram 덕분에 군집 수를 사전에 정하지 않아도 학습 수행 가능
- Dendrogram : 개체들이 결합되는 순서를 나타내는 트리형태의 구조
- Single/average method linkage → dendrogram
- Dendrogram plot

```
In [41]: # dendrogram plot
plt.figure(figsize=(50, 12))
dend = hcluster.dendrogram(hcluster.linkage(X, method = 'ward'))
```



- 1-4. Agglomerative Hierarchical clustering
- “각 데이터가 모두 나뉘어져 있는 상태에서, 작은 단위로부터 클러스터링을 시작하여 모든 데이터를 묶을 때까지 반복하는 Bottom-Up 방식으로 클러스터링 진행”
- hcluster_df → Implot 그리기
- Silhouette analysis → 각 n_cluster에 대한 Silhouette 점수
- n_clusters=10 실루엣 점수(0.77358..)가 가장 높다!

→ “실루엣 점수”는 개체가 다른 클러스터에 비해 자신의 클러스터와 얼마나 유사한 지 측정

- 실루엣 범위는 -1에서 +1까지, 값이 높으면 개체가 자체 클러스터와 잘 일치하고 인접 클러스터와 잘 일치하지 않음을 나타낸다.
- 대부분의 개체에 높은 값이 있으면 클러스터링 구성이 적합하다. 많은 포인트의 값이 낮거나 음수이면 클러스터링 구성에 클러스터가 너무 많거나 적을 수 있다.

```
In [42]: # Agglomerative Hierarchical clustering -> label
hcluster = AgglomerativeClustering(n_clusters = 10, affinity = 'euclidean', linkage = 'ward')
hcluster.fit_predict(X)
hcluster_label = hcluster.labels_
```

```
In [43]: hcluster_df = pd.DataFrame(X)

# adding hcluster labels in hcluster_df
hcluster_df['hcluster'] = hcluster_label

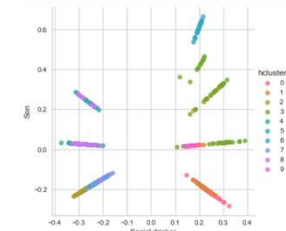
hcluster_df.head()
```

```
Out[43]:
```

	Transportation expense	Work load Average/day	Son	Social drinker	Height	hcluster
0	0.328957	-0.254351	0.324561	0.294374	0.120332	3
1	-0.177749	-0.214543	-0.217476	0.248302	-0.069948	0
2	0.329408	-0.254699	0.325005	0.294778	0.120497	3
3	-0.165427	-0.199670	-0.202400	0.231089	-0.065099	0
4	0.377743	-0.140836	0.018475	0.162997	0.066629	3

```
In [44]: facet = sns.lmplot(data=hcluster_df, x = 'Social drinker', y = 'Son',
hue = 'hcluster', fit_reg = False, legend = True, legend_out = True)

C:\Users\jwo\Anaconda3\lib\site-packages\seaborn\regression.py:592: UserWarning:
legend_out is deprecated from the "lmplot" function signature. Please update your code to s
```



Silhouette analysis

```
In [45]: # Silhouette analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

for num_clusters in range_n_clusters:
    hcluster = AgglomerativeClustering(n_clusters = num_clusters, affinity = 'euclidean', linkage = 'ward')
    hcluster.fit_predict(X)
    hcluster_label = hcluster.labels_

    # silhouette score
    silhouette_avg = silhouette_score(X, hcluster_label)
    print("For n_clusters={0}, the silhouette score is {1}:".format(num_clusters, silhouette_avg))

For n_clusters=2, the silhouette score is 0.543143159888007
For n_clusters=3, the silhouette score is 0.57376785575804
For n_clusters=4, the silhouette score is 0.584695484861825
For n_clusters=5, the silhouette score is 0.60850072754206
For n_clusters=6, the silhouette score is 0.6301201129504711
For n_clusters=7, the silhouette score is 0.666506206956714
For n_clusters=8, the silhouette score is 0.680415408475967
For n_clusters=9, the silhouette score is 0.741825564483151
For n_clusters=10, the silhouette score is 0.773589017485222
For n_clusters=11, the silhouette score is 0.688289563995439
```

- 1-5. Affinity Propagation Clustering ④ 진행
- “모든 데이터가 특정한 기준에 따라 자신을 대표할 대표 데이터를 선택한다. 만약 스스로가 자기 자신을 대표하게 되면 클러스터의 중심이 된다.”
- scaled data에 affinity propagation 알고리즘 적용 with 선택된 hyperparameter
- 각 data point의 cluster label?
- 시각화 위해 3D scatter plot
- plot에 있는 data point 색칠 위해 cluster label 사용

Affinity Propagation Clustering

```
In [3]: import pandas as pd
from sklearn.cluster import AffinityPropagation
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("Absenteeism_at_work.csv", sep=";")

# Separate the target variable [absenteeism time in hours]
target = data["Absenteeism time in hours"]
data.drop("Absenteeism time in hours", axis = 1, inplace = True)

# Scale the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Affinity Propagation 클러스터링
aff_prop = AffinityPropagation(damping = 0.5, preference = -50)
aff_prop.fit(data_scaled)

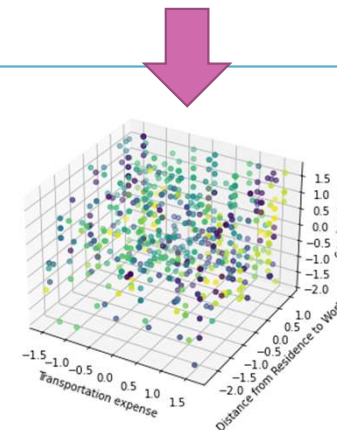
# Get the cluster labels
cluster_labels = aff_prop.labels_

# Print the cluster labels for each data point
print("Cluster labels : \n", cluster_labels)
```

```
Cluster labels :
[18  4  2 17 18 39 51  0 15 30  0  0  0  2  2 24 39  2 40 41 29  2 51 27
18 51 18 15 18  2  2  1 30  1  2 51 18 31  1 27 26 50  2 28 15 50  2 31
27 34 37 37 36  3 37  4 36  6 37  5  6 27  6  3  4 47  6 25  3 36  5 34
 6 36 27 34 36 34 35 27  6 36  6 25 34 11 36 34 36 11 36 25 36 27  5 36
34 34  5  3 46 22 36 27  3 35 36 27 36 35  3 24 26  9  7  7 11  9  9  9
 9  7  7  7  7  7  7  8 38  7  9  7  7  9 38  9 12 10 11  9 27 38
12 12  9  9  8  9  9 29 29  9  9 12  0 15 25 12  0  8 41 27 25 15  0 11
38 25 27  9 17 10  9  9  8  9 10 17  9 41  9 34  9 11 12 11  9 27 10
46 12 12 34 41 27 38 11 13 38 22 14 38 11 11 10  9  9 11 10 38 37 48 23
48 10 46 51 12 35 12 12 12 10 11 12 46 22 46 46 13 10 32  9 15 46 50 50
29 47 15 29 29 16 51 18 38 18 18  0 18 16 18 42 17 17 18 41 29 47 46
41 30 46 30 51 40 19 19 31 20 19 31 20 37 19 31 19 31 31 28 46 20 34 34
19 34 22 34 34  4 20 21  3 33  6 21 20 34 27 20 28 28 46 34 21 46 35 37
37 37 21 21 46  3  5  6 27 34 13 15 28 46 22  3 22  6 12 34 12  6  0 27
23 37  6 36  6  6 22 46 27 34  6  6 21  3 22  6 28 30 27 16  2 18  9  7
11 30 11 38 12 28 46 39 39  7 39 40 38 27 39 43 39 39 24 39 40 43 24 24
16 39 39 24 39 39  7 24 24 27 43 30 25 13 43 51 37 30 46 46 46 48 37  4
39 38  0 24 43 27 44 33 41 29 43 46 46 17 27 43 50 28 41 33 47 43 51 27
25 42 26 29  8  3 30 47  3 43  3 26 27 20 34 43 24 43 28  2 26 46  2
 8 24 29 30 47 26  0 29 29 50  0 50 17  1 24 40  2 46 51 18 28 24 31 17
 2 25 24 50 18 29 31  2 25 29 30 24 50 31  0 31 31 50 15 31 47  5  2 26
 6 33 35 32 21 34 32 34  3 32 34 36 33 33 36 33  6 42 34 33 36 33 26 36
21  5 37 36  6  3 29  6  3 34 21 26 35 34  3 36  6  3  3 36 37 37  5 20
26 33 35  6 34 36 47 26 36 35 25 28 49 21 36 36 36 15  8  8 25 25  8 25
32 25  9 11  8 39 36 39 38 43 40 41 39 26 39 47 45 39 49 43 40 39 38 26
 8 25 39 49  8 39 39 39 15 44 43 40 43 41 26 39 39 44 43 41 42 41 42 43
26 39 26 47 44 51 43 43 39 47 43 41 43 40 43 18 42 43 41 39 47 39 43 25
15 43 39  8 18 33 40  3 51  5 26 47 33 25 46 51 50 30 47 26 44 47 11 26
46 47 30 45 44  3 28 47 24 45 17 45 47 46 46  3  4 47 23 25 15 44 24 25
40 29 44 24 47 35 15 25 15 40 25 47 50  3 13 46 49 47 14 48 46 47 34  8
49 48 50 50 49 28 49 42 50 42 18 50 51 26 33 18 30 38 38 12]
```

```
In [5]: import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Plot the clusters
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(data_scaled[:, 0], data_scaled[:, 1], data_scaled[:, 2], c=cluster_labels)
ax.set_xlabel('Transportation expense')
ax.set_ylabel('Distance from Residence to Work')
ax.set_zlabel('Service time')
plt.show()
```



군집화 분석 수행 Ver.2

기계학습 과제#1 ver2

Absenteeism at work dataset - EDA & Machine Learning Model

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import ShuffleSplit, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE

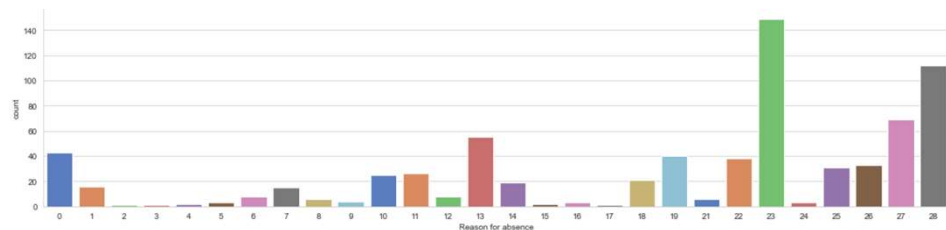
import plotly.express as px
from sklearn.decomposition import PCA
from sklearn.inspection import permutation_importance
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score, homogeneity_score,
completeness_score, v_measure_score, adjusted_rand_score, adjusted_mutual_info_score, fowlkes_mallows_score

from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
from sklearn.cluster import DBSCAN
import matplotlib.cm as cm
```

```
In [7]: # plot distribution of "reason for absence" column
sns.set_style("whitegrid")
sns.catplot(data=data, x='Reason for absence', kind='count', size=4, aspect=4, palette='muted')

C:\Users\jwoo\anaconda3\lib\site-packages\seaborn\categorical.py:3750: UserWarning: The 'size' parameter has been renamed to 'height'
; please update your code.
warnings.warn(msg, UserWarning)
```

Out [7]: <seaborn.axisgrid.FacetGrid at 0x1c1924e25e0>



ANALYSING DATA

```
In [2]: data = pd.read_csv("Absenteeism_at_work.csv", sep=";", index_col="ID")
data.head()
```

Out [2]:

	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Ht target	Disciplinary failure	Education	Sm	Social drinker	Social smoker	Pet	Wdr
ID																	
11	26	7	3	1	289	36	13	33	239.054	97	0	1	2	1	0	1	
26	0	7	3	1	118	13	18	50	239.054	97	1	1	1	1	0	0	
3	23	7	4	1	179	31	18	38	239.054	97	0	1	0	1	0	0	
7	7	7	5	1	279	5	14	39	239.054	97	0	1	2	1	1	0	
11	23	7	5	1	289	36	13	33	239.054	97	0	1	2	1	0	1	

In [3]: data.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 740 entries, 11 to 35
Data columns (total 20 columns):
# Column      Non-Null Count  Dtype
---
0 Reason for absence  740 non-null    int64
1 Month of absence  740 non-null    int64
2 Day of the week   740 non-null    int64
3 Seasons           740 non-null    int64
4 Transportation expense  740 non-null    int64
5 Distance from Residence to Work  740 non-null    int64
6 Service time      740 non-null    int64
7 Age              740 non-null    int64
8 Work load Average/day  740 non-null    float64
9 Ht target         740 non-null    int64
10 Disciplinary failure  740 non-null    int64
11 Education        740 non-null    int64
12 Sm              740 non-null    int64
13 Social drinker   740 non-null    int64
14 Social smoker    740 non-null    int64
15 Pet              740 non-null    int64
16 Weight          740 non-null    int64
17 Wdr             740 non-null    int64
```

```
In [4]: data.describe()
```

Out [4]:

	Reason for absence	Month of absence	Day of the week	Seasons	Transportation expense	Distance from Residence to Work	Service time	Age	Work load Average/day	Ht target	Disciplinary failure	Education
count	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000	740.000000
mean	19.214216	6.524324	3.914868	2.544595	221.329770	29.631981	12.554054	36.455000	271.490235	94.567838	0.564084	1.291892
std	6.435406	3.426287	1.421675	1.111631	66.952223	14.836786	4.384873	6.478772	39.056116	3.779313	0.226277	0.673236
min	0.000000	0.000000	2.000000	1.000000	118.000000	5.000000	1.000000	27.000000	205.917000	81.000000	0.000000	1.000000
25%	13.000000	3.000000	3.000000	2.000000	179.000000	16.000000	9.000000	31.000000	244.387000	93.000000	0.000000	1.000000
50%	23.000000	6.000000	4.000000	3.000000	225.000000	26.000000	13.000000	37.000000	264.249000	95.000000	0.000000	1.000000
75%	25.000000	9.000000	5.000000	4.000000	260.000000	50.000000	18.000000	40.000000	294.217000	97.000000	0.000000	1.000000
max	28.000000	12.000000	6.000000	4.000000	388.000000	62.000000	29.000000	58.000000	378.884000	106.000000	1.000000	4.000000

In [5]: categorical_atts = ["Reason for absence", "Month of absence", "Day of the week", "Seasons", "Disciplinary failure", "Education", "Social drinker", "Social smoker", "Pet", "Sm"]
numerical_atts = data.drop(columns=categorical_atts).columns.tolist()
Update types of categorical columns
For cat in categorical_atts:
data[cat] = data[cat].astype('category')

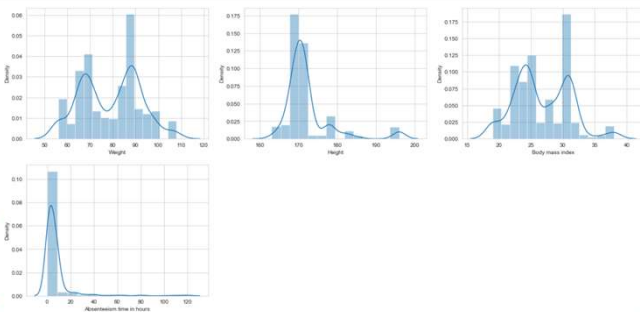
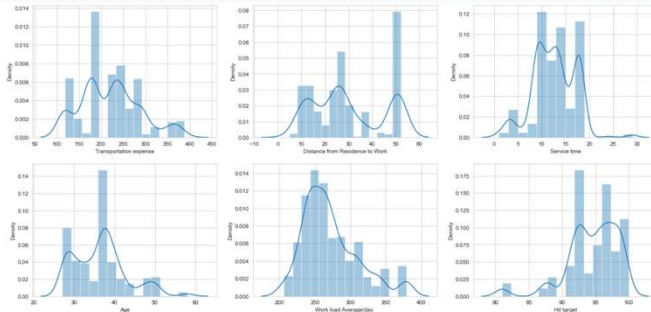
In [6]: # Number of unique values in categorical_atts
data[categorical_atts].nunique()

Out [6]:

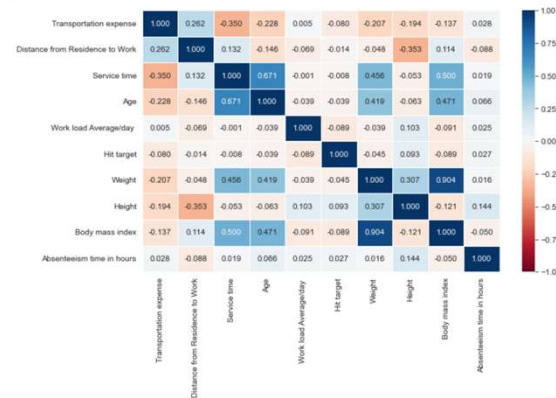
	Reason for absence	Month of absence	Day of the week	Seasons	Disciplinary failure	Education	Social drinker	Social smoker	Pet	Sm
dtype:	int64	int64	int64	int64	int64	int64	int64	int64	int64	int64

- import 라이브러리, 패키지
- data (index_col을 ID로!)
- info, describe, nunique
- catplot : 범주형 변수와 수치형 변수 간 관계 시각화

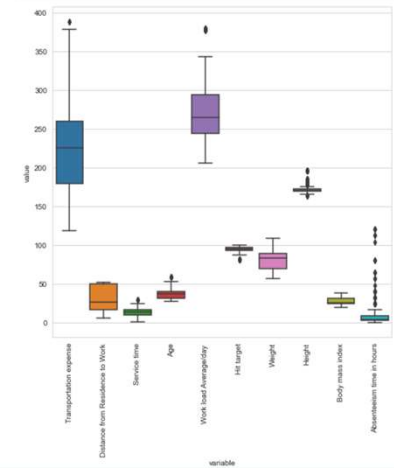
```
In [8]: # plot distribution of numerical attributes
num_df = data[numerical_atts]
plt.figure(figsize=(20,20))
for i in range(1, 11):
    plt.subplot(4, 3, i)
    sns.distplot(num_df[num_df.columns[i-1]], bins=14)
```



```
In [9]: # correlation matrix
num_corr = num_df.corr()
plt.figure(figsize=(10,6))
sns.heatmap(num_corr, annot=True, fmt=".3f", vmin=-1, vmax=1, linewidths=.5, cmap = sns.color_palette("RdBu", 100))
plt.show()
```



```
In [10]: # display boxplots
plt.figure(figsize=(8,8))
sns.boxplot(x="variable", y="value", data=pd.melt(num_df))
plt.xticks(rotation=90)
plt.show()
```



- subplot : 2개 이상의 그래프 한 figure에 출력 가능
- distplot : 분포 표현, 히스토그램 그려준다
- heatmap
- boxplot

- outlier 확인 위해 제1분위수(Q1), 제3분위수(Q3)로 min, max 확인
- outlier 값들을 → mean 값으로 대체
- 정규화 진행
- best parameters?
- → grid.best_estimator_.feature_importances_ 대해 pie chart 그려보기

OUTLIER DETECTION

```
In [11]: # Check for outliers using boxplots and drop them
for num_att in numerical_atts:
    # Getting 75 and 25 percentile of variable "j"
    Q3, Q1 = np.percentile(data[num_att], [75,25])
    MEAN = data[num_att].mean()

    # Calculating Interquartile range
    IQR = Q3 - Q1

    # Calculating upper extream and lower extream
    minimum = Q1 - (IQR*1.5)
    maximum = Q3 + (IQR*1.5)

    # Replacing all the outliers value to Mean
    data_clean=data.drop(data.loc[data[num_att]< minimum,num_att].index)
    data_clean=data.drop(data.loc[data[num_att]> maximum,num_att].index)
```

```
In [12]: # size reduced to 150 from 740
len(data_clean)
```

Out[12]: 150

NORMALIZATION

```
In [13]: # transform data
numeric_transformer = MinMaxScaler()
#categorical_transformer = OneHotEncoder(sparse=False, handle_unknown='ignore')

# exclude target attr,bute
numerical_atts = [x for x in numerical_atts if x not in ['Absenteeism time in hours']]
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_atts),
        ('cat', 'passthrough', categorical_atts)
    ], remainder='passthrough')
data_pp = preprocessor.fit_transform(data_clean)
```

FEATURE SELECTION

1. w Random Forest importances:

- Train a RF model to see feature importances. If's ok, we are already have target attribute in the dataset.

```
In [14]: # X, y splitting (we dont need train test splitting since this model is just for feature importances)
X, y = data_pp[:, :-1].reshape(-1, 1)
y = np.round(MinMaxScaler((0,2)).fit_transform(y, y)).ravel()
X = np.delete(data_pp, -1, 1)

# cross-validation with 10 splits
cross_val = ShuffleSplit(n_splits=10, random_state = 40)

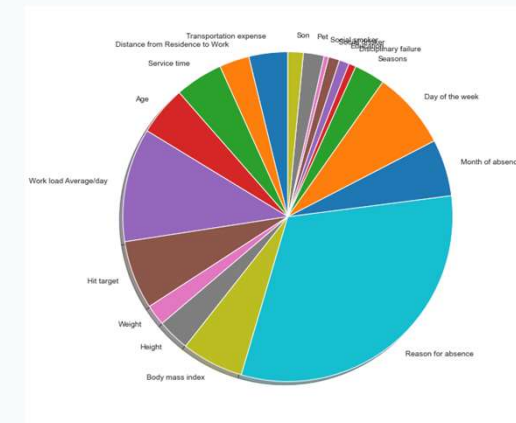
# define model
rf = RandomForestClassifier(random_state = 0, max_features=None, n_jobs=-1)

# parameters
parameters = {
    "n_estimators": [1000],
    "max_depth": [100, 112],
    "criterion": ["gini", "entropy"],
    "criterion": ["mse", "mae"],
    "class_weight": [None, "balanced"],
    "max_features": ["auto", None, "log2"],
}

# grid search for parameters
grid = GridSearchCV(estimator=rf, param_grid=parameters, cv=cross_val, n_jobs=-1) #multithreading: all cores are used
grid.fit(X, y)

# print best score
print("The best parameters are %s with a score of %0.4f" % (grid.best_params_, grid.best_score_))

The best parameters are: {'n_estimators': 1000} with a score of 0.7133
```



```
In [15]: # pie-chart
atts = numerical_atts+categorical_atts
plt.figure(figsize=(10,10))
plt.pie(grid.best_estimator_.feature_importances_, labels=atts, shadow=True, startangle=90)
```


- permutation feature importance 변수 중요도 확인
- '특정 모델에 특화된(Model-specific)'이 아닌, '어느 모델이든 (Model-agnostic) 학습시킨 후(Post-hoc)'에 적용
- "Black-box 모델에 대하여, 특정 feature를 안 썼을 때, 이것이 성능 손실에 얼마만큼의 영향을 주는지를 통해 그 feature의 중요도를 파악하는 방법"
- 📖 재학습 필요 없다! 그러나 무작위로 섞기 때문에 실행마다 결과 달라질 수 있다
- 변수 중요도 각각 확인 후 attribute 9개 선택

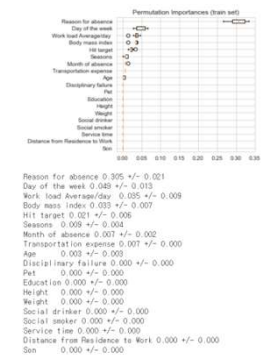
2. w Permutation importance:

• see https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance

```
In [16]: # permutation importance with RF classifier
atts_arr = np.array(atts)
result = permutation_importance(grid, X, y, n_repeats=10,
                                random_state=42, n_jobs=-1)
sorted_idx = result.importances_mean.argsort()

fig, ax = plt.subplots()
ax.boxplot(result.importances[sorted_idx].T,
            vert=False, labels=atts_arr[sorted_idx])
ax.set_title("Permutation Importances (train set)")
fig.tight_layout()
plt.show()

for i in result.importances_mean.argsort()[::-1]:
    print(f"atts_arr[{i}]<3>")
    f"result.importances_mean[{i}]: {3f}"
    f"result.importances_std[{i}]: {3f}"
```



4.2. Permutation feature importance

Permutation feature importance is a model inspection technique that can be used for any fitted estimator when the data is tabular. This is especially useful for non-linear or opaque estimators. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled [1]. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature. This technique benefits from being model agnostic and can be calculated many times with different permutations of the feature.

Warning: Features that are deemed of **low importance for a bad model** (low cross-validation score) could be **very important for a good model**. Therefore it is always important to evaluate the predictive power of a model using a held-out set (or better with cross-validation) prior to computing importances. Permutation importance does not reflect to the intrinsic predictive value of a feature by itself but **how important this feature is for a particular model**.

The `permutation_importance` function calculates the feature importance of estimators for a given dataset. The `n_repeats` parameter sets the number of times a feature is randomly shuffled and returns a sample of feature importances.

• choose 9 attributes based on Permutation importances w RF

```
In [17]: # select important attr by RF
selected_atts = ['Reason for absence', 'Day of the week', 'Work load Average/day', 'Body mass index', 'Hit target',
                'Seasons', 'Month of absence', 'Transportation expense', 'Age']

selected_atts_idx = np.where(np.isin(atts.selected_atts), 0)
X_selected = X[selected_atts_idx]
```

SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/PERMUTATION_IMPORTANCE.HTML#PERMUTATION-IMPORTANCE](https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance)

2-1. K-MEANS CLUSTERING ①

1. K-means:

"inertia_" attribute provides sum of squared distances of samples to their closest cluster center. We can plot it and decide on the parameter "n_clusters" by using elbow method.

see <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

```
In [18]: # run k-means for range of 10 clusters then analyse with elbow method
clusters = []

for i in range(1, 11):
    km = KMeans(n_clusters=i).fit(X_selected)
    clusters.append(km.inertia_)

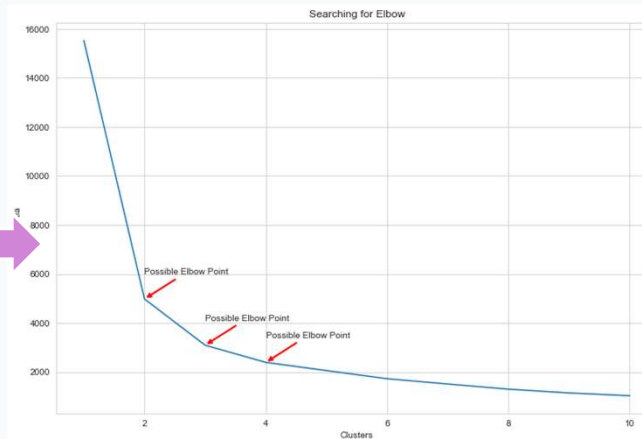
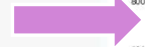
fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x=list(range(1, 11)), y=clusters, ax=ax)
ax.set_title('Searching for Elbow')
ax.set_xlabel('Clusters')
ax.set_ylabel('Inertia')

# Annotate arrow
ax.annotate('Possible Elbow Point', xy=(4, clusters[3]), xytext=(4, clusters[3]+1000), xycoords='data',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='red', lw=2))

ax.annotate('Possible Elbow Point', xy=(3, clusters[2]), xytext=(3, clusters[2]+1000), xycoords='data',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='red', lw=2))

ax.annotate('Possible Elbow Point', xy=(2, clusters[1]), xytext=(2, clusters[1]+1000), xycoords='data',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3', color='red', lw=2))

plt.show()
```



Notes

The k-means problem is solved using either Lloyd's or Elkan's algorithm.

The average complexity is given by $O(knT)$, where n is the number of samples and T is the number of iteration.

The worst case complexity is given by $O(n^4(k+2/p))$ with $n = n_{\text{samples}}$, $p = n_{\text{features}}$. Refer to "How slow is the k-means method?" D. Arthur and S. Vassilvitskii - SoCG2006, for more details.

In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times.

If the algorithm stops before fully converging (because of `tol` or `max_iter`), `labels_` and `cluster_centers_` will not be consistent, i.e. the `cluster_centers_` will not be the means of the points in each cluster. Also, the estimator will reassign `labels_` after the last iteration to make `labels_` consistent with `predict` on the training set.

SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/GENERATED/SKLEARN.CLUSTER.KMEANS.HTML](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

I TRIED POSSIBLE ELBOW POINTS BY CREATING PLOTS. USED **T-SNE** FOR VISUALIZING DATA ON 3D SPACE.
SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/GENERATED/SKLEARN.MANIFOLD.TSNE.HTML](https://scikit-learn.org/stable/modules/generated/sklearn.manifold.tsne.html)

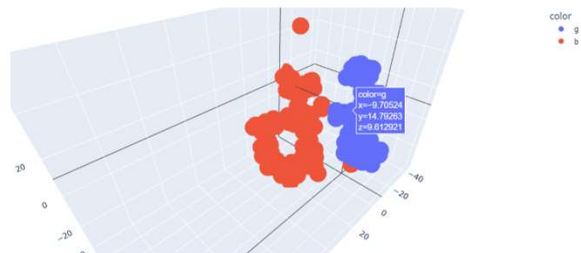
```
In [19]: # 2 clusters
km2 = KMeans(n_clusters=2).fit(X_selected)

# plot using tsne
X_embedded = TSNE(n_components=3, perplexity=10, random_state=24).fit_transform(X_selected)
colors = np.array([x for x in 'bgorykbrnykbgorykbrnyk'])

fig = px.scatter_3d(x=X_embedded[:,0], y=X_embedded[:,1], z=X_embedded[:,2], color=colors[km2.labels_])
fig.show()
```

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:670: FutureWarning: The default value of 'n_init' will change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the warning

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.

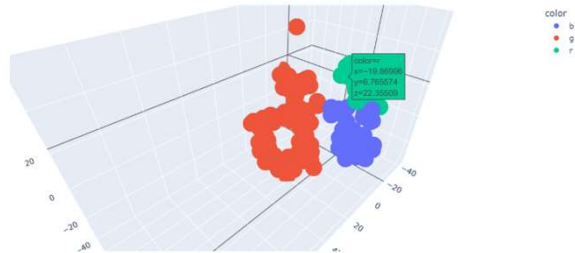


```
In [20]: # 3 clusters
km3 = KMeans(n_clusters=3).fit(X_selected)

# plot using tsne
X_embedded = TSNE(n_components=3, perplexity=10, random_state=24).fit_transform(X_selected)
fig = px.scatter_3d(x=X_embedded[:,0], y=X_embedded[:,1], z=X_embedded[:,2], color=colors[km3.labels_])
fig.show()
```

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:670: FutureWarning: The default value of 'n_init' will change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the warning

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.

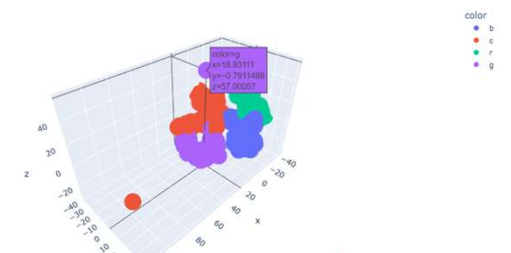


```
In [21]: # 4 clusters
km4 = KMeans(n_clusters=4).fit(X_selected)

# plot using tsne
X_embedded = TSNE(n_components=3, perplexity=10, random_state=24).fit_transform(X_selected)
fig = px.scatter_3d(x=X_embedded[:,0], y=X_embedded[:,1], z=X_embedded[:,2], color=colors[km4.labels_])
fig.show()
```

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:670: FutureWarning: The default value of 'n_init' will change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the warning

C:\Users\jwoofanaconda\lib\site-packages\sklearn\cluster_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.



It seems like 4 clusters work best. Let's try the same process again with 2-2. **PCA + K-means.**

FOR PCA SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/GENERATED/SKLEARN.DECOMPOSITION.PCA.HTML](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.pca.html)

```
In [22]: # PCA for reducing dimensions to 3
pca = PCA(n_components=3)
pca_results = pca.fit_transform(X_selected)

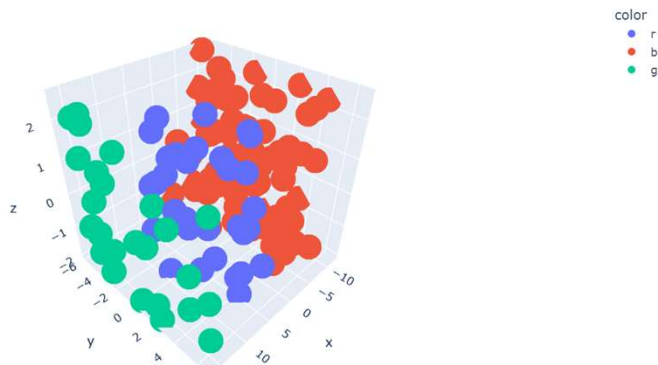
# running k-means on results of pca
km_pca = KMeans(n_clusters=3).fit(pca_results)
fig = px.scatter_3d(x=pca_results[:,0],y=pca_results[:,1],z=pca_results[:,2],color=km_pca.labels_)
fig.show()
```

C:\Users\jwwoo\anaconda3\lib\site-packages\sklearn\cluster_kmeans.py:870: FutureWarning:

The default value of 'n_init' will change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the warning

C:\Users\jwwoo\anaconda3\lib\site-packages\sklearn\cluster_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.



→ **Lost** most of valuable information about data with **PCA(차원축소)**. So, resulting graph seems **incomplete**.

- **T-SNE**
- 매니폴드(manifold) 학습의 하나, 복잡한 데이터의 시각화가 목적
- 높은 차원의 데이터를 2차원 또는 3차원으로 축소시켜 시각화
- 높은 차원 공간에서 비슷한 데이터 구조는 낮은 차원 공간에서 가깝게 대응, 비슷하지 않은 데이터 구조는 멀리 떨어져 대응되는 결과 도출!

→ USING K-MEANS ON ORIGINAL 7-DIMENSIONAL DATA THEN PLOTTING WITH T-SNE GIVES BETTER RESULTS.

SEE

[HTTPS://SCIKIT-](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score)

[LEARN.ORG/STABLE/MODULES/GENERATED/SKLEARN.CLUSTER.AGGLOMERATIVECLUSTERING.HTML#SKLEARN.CLUSTER.AGGLOMERATIVECLUSTERING](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score)

- 2-3. **AGNES (Agglomerative Clustering)**
- silhouette coefficient 사용 (best : 1, worst : -1) → to select cluster number of AGNES.
- 클러스터의 실루엣 값 plot 함수 정의
- **Silhouette Score** 종합
- 실루엣 coefficient plot
- It seems like 3, 6 or 9 are best amongst them
- → Choose to work with $n_clusters = 3$.

2. AGNES:

There's no `merge` (Sum of squared distances of samples to their closest cluster center) attribute of `AgglomerativeClustering` class so we used `silhouette` coefficient (best: 1, worst: -1) to select cluster number of AGNES.
see https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score
for AGNES see <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering>

```
in [20]: # Define a function to plot silhouette values of clusters
def silhouette_plot(X, y, n_clusters, ax=None):
    if ax is None:
        ax = plt.gca()

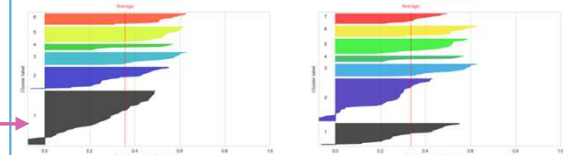
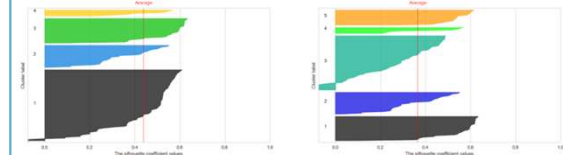
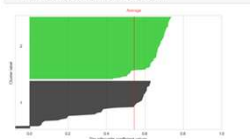
    # Compute the silhouette scores for each sample
    silhouette_avg = silhouette_score(X, y)
    samples_silhouette_values = silhouette_samples(X, y)

    y_lower = padding = 0
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # the i-th cluster, sorted by their silhouette values
        (y_upper, silhouette_values) = samples_silhouette_values[i]
        size_cluster_i = len(silhouette_values)
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral((float(i) / n_clusters) * 255).int()
        ax.fill_between(y_lower, y_upper, color,
                        label='Cluster %i' % (i+1))
        # Label the silhouette plot with their cluster numbers at the middle
        ax.text(0.5, (y_lower + y_upper) * 0.5, str(i + 1))
        # Draw the new x_lower for next plot
        y_lower = y_upper + padding

    ax.set_xlabel('The silhouette coefficient values')
    ax.set_ylabel('Cluster label')
```

```
# The vertical line for average silhouette score of all the values
ax.axvline(silhouette_avg, color='r', lw=2, label='Average')
ax.set_ylim(-0.5, y_upper + 1.05)
ax.set_xlim(0, 1)
ax.set_xlabel('The silhouette coefficient values')
ax.set_ylabel('Cluster label')
```

```
in [20]: # Plot silhouette coefficients
plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1)
ax = AgglomerativeClustering(n_clusters=3, linkage='average')
ax.labels_ = ax.labels_
plt.subplots(2, 1)
```

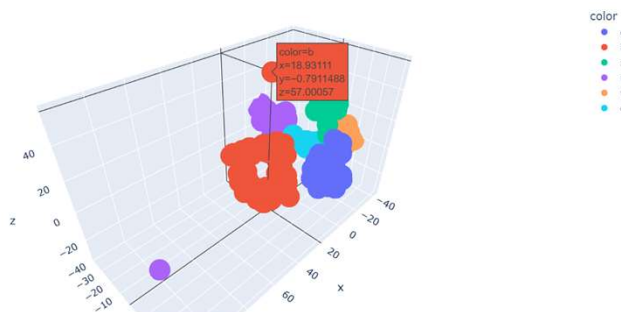


It seems like 3, 6 or 9 are best amongst them (cluster with equal sizes and significant silhouette value). Choose to work with $n_clusters = 3$.

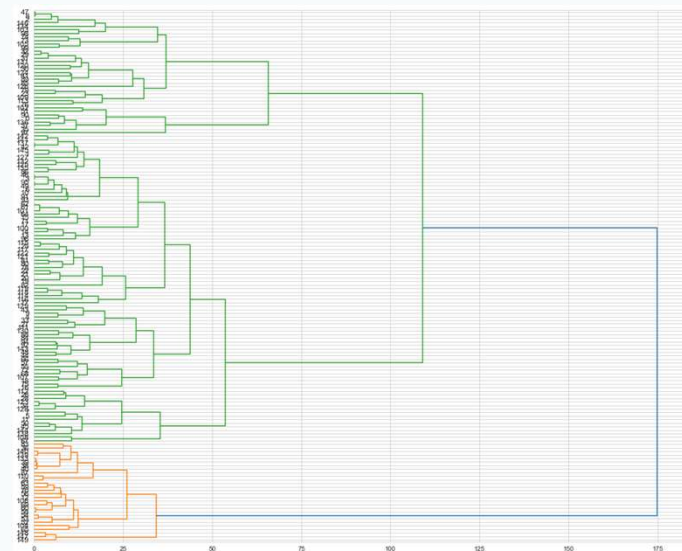
SEE https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score

```
In [25]: # 6 clusters
agnes6 = AgglomerativeClustering(n_clusters=6, linkage='average').fit(X_selected)

# plot using tsne
X_embedded = TSNE(n_components=3, perplexity=10, random_state=24).fit_transform(X_selected)
fig = px.scatter_3d(x=X_embedded[:,0], y=X_embedded[:,1], z=X_embedded[:,2], color=agnes6.labels_)
fig.show()
```



- TSNE 이용하여 3d plot 그리기
- 가능한 clustering 대해 dendrogram 그려보기



Let's plot the complete dendrogram to see the possible clusterings.

```
In [26]: # there's no inertia attribute of AgglomerativeClustering class so we use scipy's distance matrix
dist = distance_matrix(X_selected, X_selected)

# dendrogram
Z = hierarchy.linkage(dist, 'average')
plt.figure(figsize=(18, 15))
dendro = hierarchy.dendrogram(Z, leaf_rotation=0, leaf_font_size=12, orientation='right')
```

3. DBSCAN:

eps: The maximum distance between two samples for one to be considered as in the neighborhood of the other (default: 0.5).

min_samples: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself (default: 5).

see <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>

```
In [27]: # search for best parameters by using silhouette_score
score_list=[]
for eps in np.arange(0.5,20,0.5):
    for min_samples in range(3,20):
        db = DBSCAN(eps=eps, min_samples=min_samples).fit(X_selected)
        labels = db.labels_
        n = len(np.unique(labels))
        if n>1:
            score=silhouette_score(X_selected, labels)
            score_list.append((score,(eps,min_samples)))

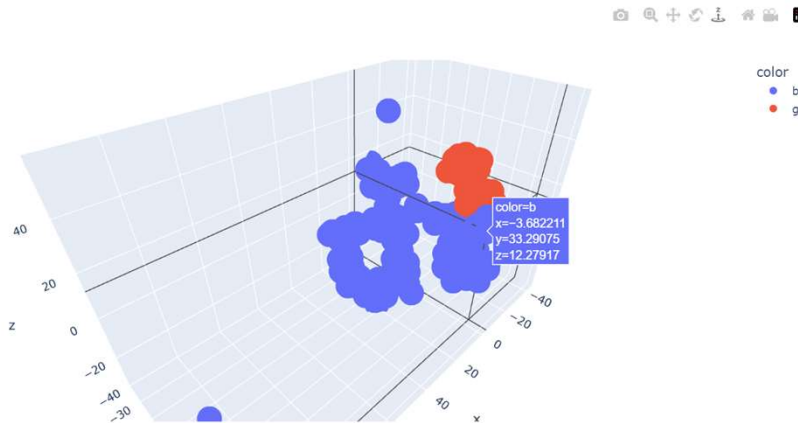
biggest_score = sorted(score_list)[-1]
best_eps, best_min_spamles = biggest_score[1]
best_eps, best_min_spamles
```

Out [27]: (5.0, 11)

```
In [28]: # best model for DBSCAN
db_best = DBSCAN(eps=best_eps, min_samples=best_min_spamles).fit(X_selected)

# plot using tsne
X_embedded = TSNE(n_components=3, perplexity=10, random_state=24).fit_transform( X_selected )

fig = px.scatter_3d(x=X_embedded[:,0],y=X_embedded[:,1],z=X_embedded[:,2],color=db_best.labels_)
fig.show()
```



2-4. DBSCAN clustering ③

(Density-Based-Spatial Clustering of Applications with Noise)

- DBSCAN : “서로 인접한 데이터들은 같은 클러스터 내에 있다는 것을 가정한 알고리즘”
- 가깝게 인접해 있는 데이터들은 같은 클러스터라 가정하고 인접한 데이터를 잘 찾아서 클러스터 할당 (K-Means 약점 극복)
- 특정 데이터를 중심으로 밀도가 높은 곳에 포함된 데이터에는 클러스터를 할당하고, 밀도가 낮으면 그 데이터를 노이즈로 취급!
- **Silhouette score** 이용하여 최선의 parameter 탐색
- best model for DBSCAN?

SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/GENERATED/SKLEARN.CLUSTER.DBSCAN.HTML](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html)

Evaluation METRICS FOR

K-MEANS PCA+K-MEANS AGNES DBSCAN

- Compare best models by using **8 metrics**:
- **1) Estimated number of clusters Estimated number of noise points**
- **2) Homogeneity**: For perfect clustering, each cluster contains only members of a single class.
- **3) Completeness**: For perfect clustering, all members of a given class are assigned to the same cluster.
- **4) V-measure**: Harmonic mean of Homogeneity and Completeness.
- **5) Adjusted Rand Index**: Given the knowledge of the ground truth class, the adjusted Rand index is a function that measures the similarity of the two assignments, ignoring permutations and with chance normalization:
- **6) Adjusted Mutual Information**: Given the knowledge of the ground truth class, the Mutual Information is a function that measures the agreement of the two assignments, ignoring permutations.
- **7) Fowlkes-Mallows score**: Geometric mean of precision and recall.
- **8) Silhouette Coefficient**: If the ground truth labels are not known, the Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample by $(b - a) / \max(a, b)$

SEE [HTTPS://SCIKIT-LEARN.ORG/STABLE/MODULES/CLUSTERING.HTML#CLUSTERING-PERFORMANCE-EVALUATION](https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation)


```

In [29]: # print metric for chosen models
models = [km4, km_pca, agnes6, db_best]
names = ["K-MEANS:", "PCA + K-MEANS:", "AGNES:", "DBSCAN:"]

for i, model in enumerate(models):
    labels = model.labels_
    n = len(np.unique(labels))
    y = np.round(MinMaxScaler((0,n)).fit_transform(y_original)).ravel()

    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise_ = list(labels).count(-1)

    print(names[i])
    print('Estimated number of clusters: %d' % n_clusters_)
    print('Estimated number of noise points: %d' % n_noise_)
    print("Homogeneity: %0.3f" % homogeneity_score(y, labels))
    print("Completeness: %0.3f" % completeness_score(y, labels))
    print("V-measure: %0.3f" % v_measure_score(y, labels))
    print("Adjusted Rand Index: %0.3f" % adjusted_rand_score(y, labels))
    print("Adjusted Mutual Information: %0.3f" % adjusted_mutual_info_score(y, labels))
    print("Fowlkes-Mallows score: %0.3f" % fowlkes_mallows_score(y, labels))
    print("Silhouette Coefficient: %0.3f" % silhouette_score(X, labels))
    print("\n")

```

- ① Only metric doesn't require ground-truth(실측계수) labels is **Silhouette Coefficient**.
- For it → **DBSCAN > PCA+KMEANS > K-MEANS > AGNES**.
② Its results are correlated with **Fowlkes-Mallows** score although they require different inputs with only difference of **swapping AGNES and K-MEANS**.
- ③ For **Homogeneity and Completeness** principles → **AGNES > PCA+KMEANS > K-MEANS > DBSCAN**. ④ The results are similar for **Mutual Information**.
- ⑤ For **Rand Index** → **AGNES > K-MEANS > PCA+KMEANS > DBSCAN**.

군집화 분석 결과 평가

K-MEANS:

Estimated number of clusters: 4
 Estimated number of noise points: 0
 Homogeneity: 0.157
 Completeness: 0.133
 V-measure: 0.144
 Adjusted Rand Index: 0.075
 Adjusted Mutual Information: 0.121
 Fowlkes-Mallows score: 0.344
 Silhouette Coefficient: 0.330

PCA + K-MEANS:

Estimated number of clusters: 3
 Estimated number of noise points: 0
 Homogeneity: 0.147
 Completeness: 0.176
 V-measure: 0.160
 Adjusted Rand Index: 0.077
 Adjusted Mutual Information: 0.143
 Fowlkes-Mallows score: 0.423
 Silhouette Coefficient: 0.437

AGNES:

Estimated number of clusters: 6
 Estimated number of noise points: 0
 Homogeneity: 0.211
 Completeness: 0.188
 V-measure: 0.199
 Adjusted Rand Index: 0.110
 Adjusted Mutual Information: 0.156
 Fowlkes-Mallows score: 0.348
 Silhouette Coefficient: 0.276

DBSCAN:

Estimated number of clusters: 2
 Estimated number of noise points: 0
 Homogeneity: 0.029
 Completeness: 0.045
 V-measure: 0.035
 Adjusted Rand Index: -0.035
 Adjusted Mutual Information: 0.024
 Fowlkes-Mallows score: 0.557
 Silhouette Coefficient: 0.518