데이터 전처리 및 탐색적 분석
차원축소

PIC
실습

박석희 교수님

실습조교 이승문
ch273404@naver.com

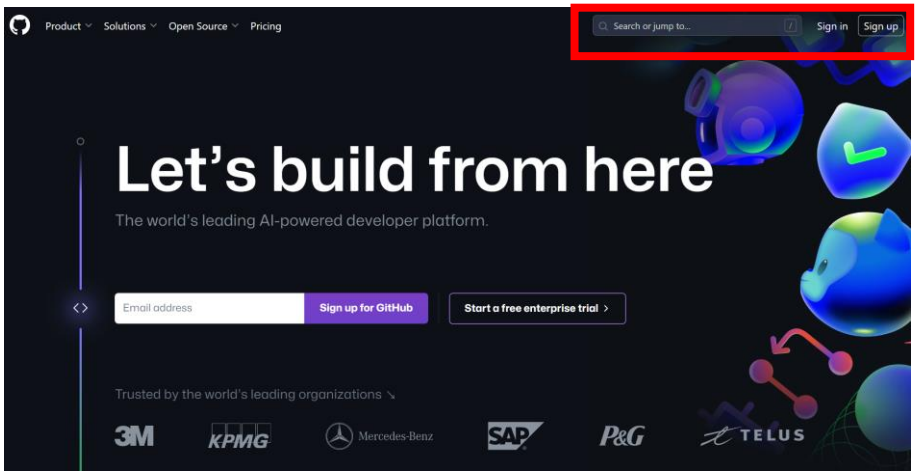
1/9 (목)

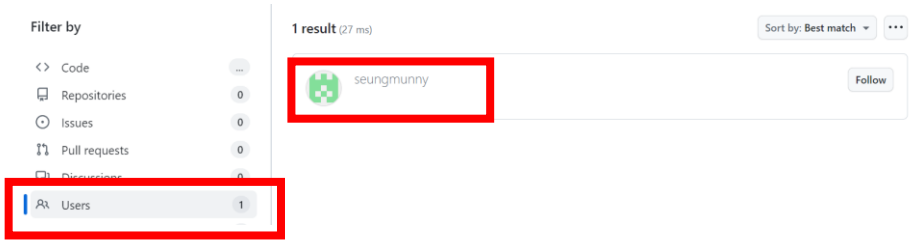
목차

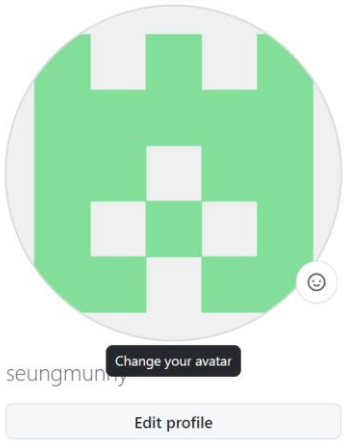
코드 및 데이터 다운로드

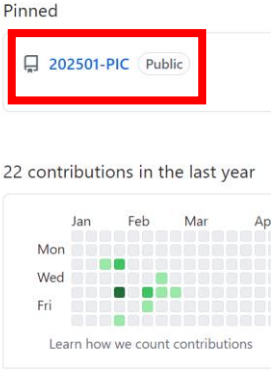
깃허브 접속

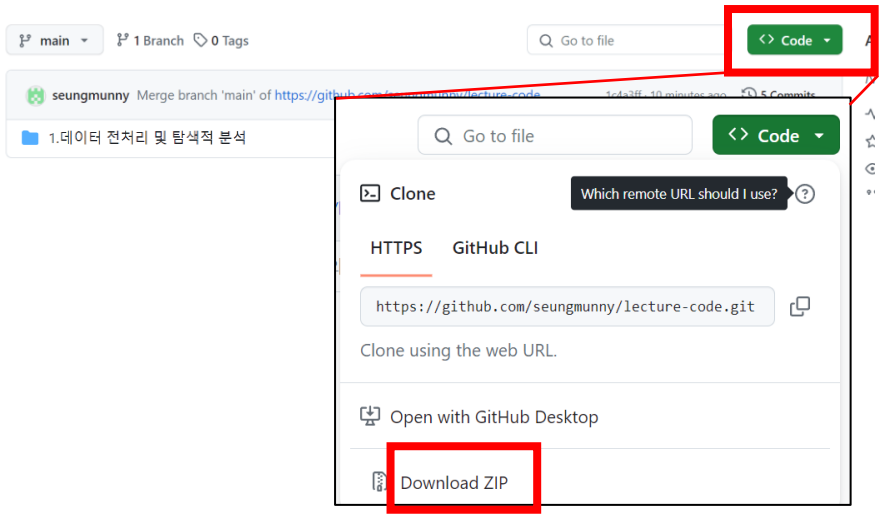
Seungmunny 입력











Miniforge jupyter notebook 실행

- **가상환경 만들기**
  envname에 원하는 이름 입력

```
mamba create –n PIC2501
python=3.9.12
```
원하는 이름 입력하시면 됩니다.

- **Jupyter notebook 설치**

```
pip install jupyter
```

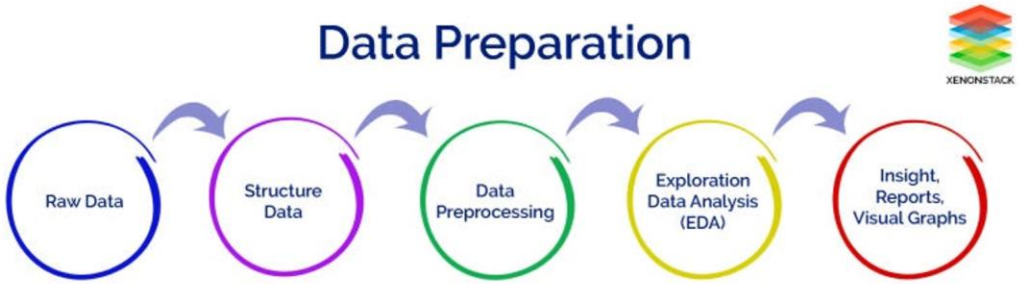
- **가상환경 실행 및 Jupyter notebook 실행**

```
mamba activate PIC2501
Jupyter notebook
```

데이터의 전처리 및 탐색적 분석

❖ 데이터 전처리 및 탐색적 분석

- 데이터 전처리 (Data Preprocessing)
- 탐색적 데이터 분석 (EDA, Exploratory Data Analysis)



https://medium.com/@mallrishabh52/data-preprocessing-and-workflow-of-a-machine-learning-project-bc6ca9e6f3ad

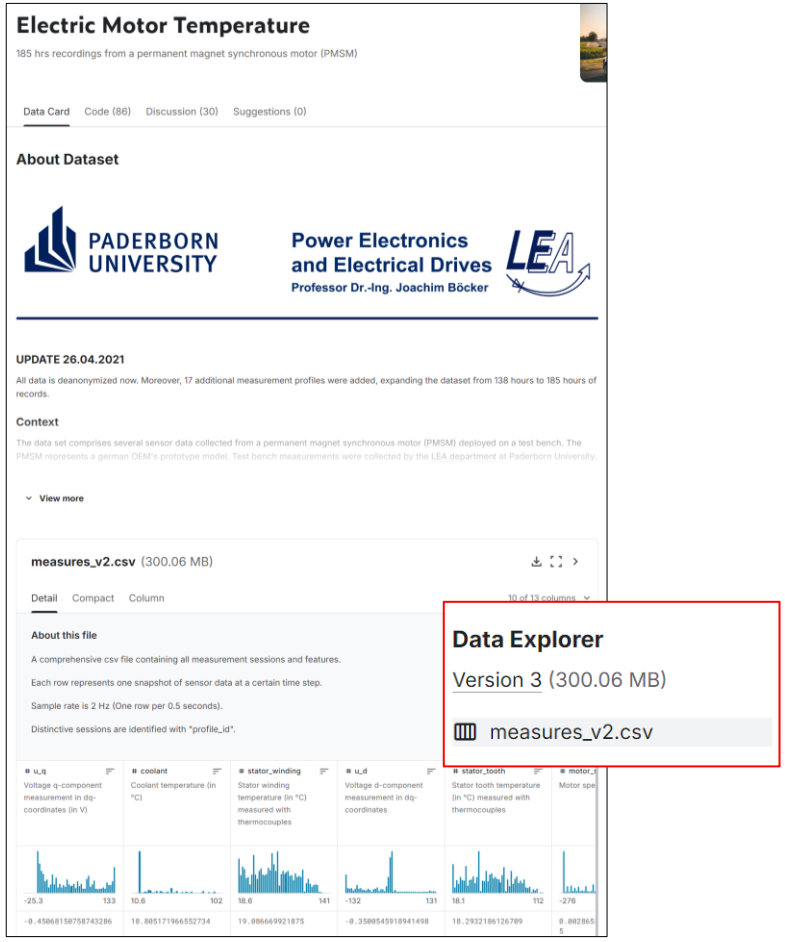
구분	데이터 전처리 (Data Preprocessing)	탐색적 데이터 분석 (EDA)
정의	원시 데이터를 머신러닝 모델에 적합하게 가공 및 변환하는 과정	데이터를 다양한 각도에서 분석하여 패턴, 관계, 이상치를 발견하는 과정
목적	데이터를 정제하고 구조화하여 모델이 잘 학습할 수 있도록 준비	데이터의 분포, 패턴, 관계를 이해하고 모델링에 필요한 인사이트 도출
단계	- 결측값 처리 - 이상치 제거 - 데이터 정규화/표준화 - 인코딩 (범주형 데이터 처리) - 데이터 스케일링	- 데이터 시각화 (히스토그램, 산점도 등) - 통계적 분석 (평균, 분산, 상관계수 등) - 이상치 탐지 - 변수 간 관계 분석
결과물	모델 학습을 위한 정제된 데이터셋	데이터에 대한 인사이트 및 모델링 전략
기법/도구	- Pandas - Scikit-learn (StandardScaler, MinMaxScaler) - 데이터 클리닝 스크립트	- Matplotlib - Seaborn - Plotly - Pandas Profiling
시기	데이터 분석 및 모델링 전에 수행	데이터 전처리 후, 모델링 전에 수행
핵심 질문	"데이터를 어떻게 정제하고 구조화할 것인가?"	"데이터에서 어떤 패턴이나 관계가 존재하는가?"

교수님 자료

데이터 전처리-모터 온도 데이터

❖ 데이터셋 확인 및 불러오기

- Electric motor temperature (Kaggle 데이터)



✓ 전압, 전류 등 구동조건과
냉각수, 영구자석, tooth 등의 온도 측정 결과 데이터

https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature

- 라이브러리 import 및 데이터 읽어오기

```
!pip install pandas
!python -m pip install -U matplotlib
!pip install seaborn
!pip install missingno
!pip install scikit-learn
!pip install umap-learn
```

✓ 실습에 필요한 라이브러리들을 설치하는 코드

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.impute import KNNImputer
from sklearn.manifold import TSNE, Isomap
import umap
```

✓ 라이브러리 import

```
df=pd.read_csv("data/electric_motor_temperature_50000.csv")
#https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature
```

✓ csv 파일의 데이터 읽어오기

데이터 전처리-모터 온도 데이터

실습

❖ 결측치 처리

- 결측치 확인

i_d	i_q	pm	stator_yok	ambient	torque
0.004419	0.000328	24.55421	18.31655	19.85069	0.187101
0.000606	-0.00079	24.53808		19.85067	0.245417
0.00129	0.000386	24.54469	18.32631	19.85066	0.176615
2.56E-05	0.002046	24.55402	18.33083	19.85065	0.238303
-0.06432	0.037184	24.5654	18.32666	19.85064	0.208197

✓ csv 파일 내부 특정 값이 비어있거나 기록되지 않은 부분
→데이터 분석의 정확도 저하 모델링 시 오류 발생

df.head()

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambie
0	-0.450682	18.805172	19.086670	-0.350055	18.293219	0.002866	0.004419	0.000328	24.554214	18.316547	19.8506
1	-0.325737	18.818571	19.092390	-0.305803	18.294807	0.000257	0.000606	-0.000785	24.538078	NaN	19.8506
2	-0.440864	18.828770	19.089380	-0.372503	18.294094	0.002355	0.001290	0.000386	24.544693	18.326307	19.8506
3	-0.327026	18.835567	19.083031	-0.316199	18.292541	0.006105	0.000026	0.002046	24.554018	18.330833	19.8506
4	-0.471150	18.857033	19.082525	-0.332272	18.291428	0.003133	-0.064317	0.037184	24.565397	18.326662	19.8506

✓ 데이터 앞부분 5개를 출력하는 df.head() 입력 시
결측치는 NaN으로 표시됨

- 결측치 개수 확인

```
df.isnull().sum()
```

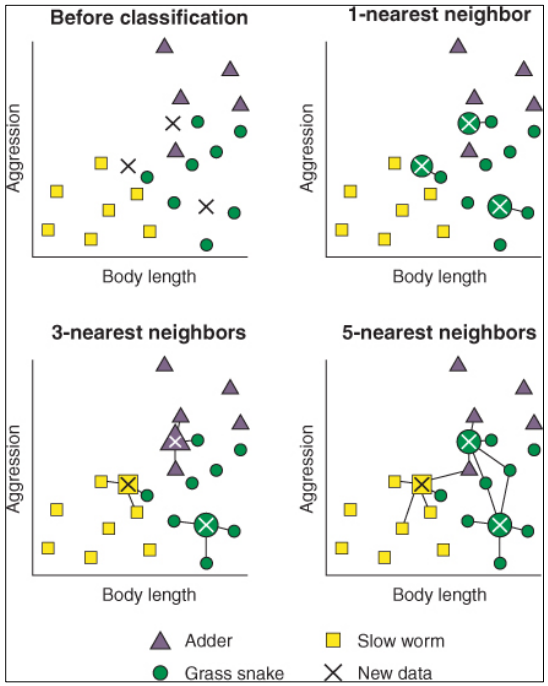
```
u_q               1
coolant           0
stator_winding    0
u_d               2
stator_tooth      0
motor_speed       1
i_d               0
i_q               2
pm                1
stator_yoke       1
ambient           0
torque            2
profile_id        0
dtype: int64
```

데이터 전처리-모터 온도 데이터

실습

❖ 결측치 처리-K nearest neighbor(KNN)

- KNN imputation



✓ 결측치의 가장 가까운 주변 K개의 평균으로 대체

```
# KNN Imputer를 사용한 결측치 대체
imputer = KNNImputer(n_neighbors=5)
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

✓ n_neighbors 변수가 k를 의미
✓ 주변 5개 데이터의 평균으로 결측치를 대체함

```
df_imputed.head()
```

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambient	torque	profile_id
0	-0.450682	18.805172	19.086670	-0.350055	18.293219	0.002866	0.004419	0.000328	24.554214	18.316547	19.850691	0.187101	17.0
1	-0.325737	18.818571	19.092390	-0.305803	18.294807	0.000257	0.000606	-0.000785	24.538078	18.324842	19.850672	0.245417	17.0
2	-0.440864	18.828770	19.089380	-0.372503	18.294094	0.002355	0.001290	0.000386	24.544693	18.326307	19.850657	0.176615	17.0
3	-0.327026	18.835567	19.083031	-0.316199	18.292541	0.006105	0.000026	0.002046	24.554018	18.330833	19.850647	0.238303	17.0
4	-0.471150	18.857033	19.082525	-0.332272	18.291428	0.003133	-0.064317	0.037184	24.565397	18.326662	19.850639	0.208197	17.0

```
df_imputed.isnull().sum()
```

```
u_q              0
coolant          0
stator_winding   0
u_d              0
stator_tooth     0
motor_speed      0
i_d              0
i_q              0
pm               0
stator_yoke      0
ambient          0
torque           0
profile_id       0
dtype: int64
```

✓ 결측치 대체 후 insull 결과가 0으로 변화

데이터 전처리-모터 온도 데이터

실습

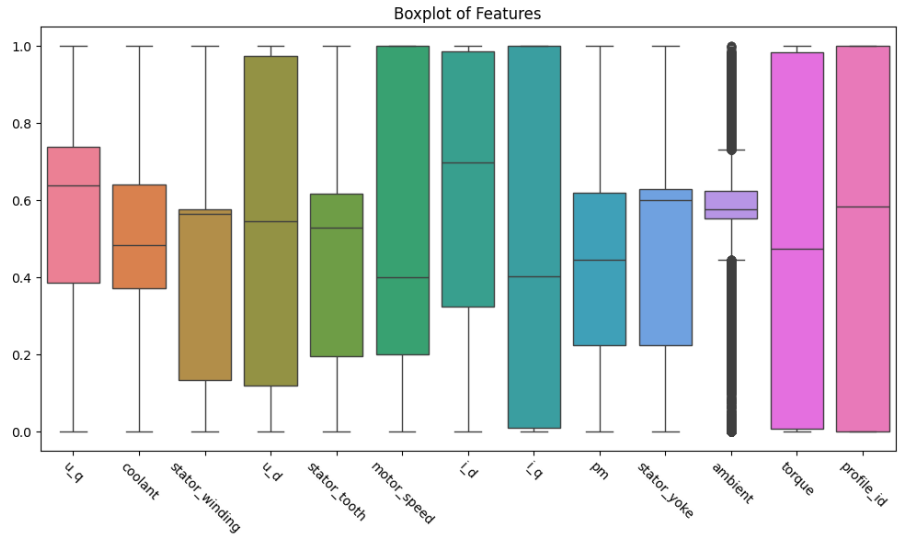
❖ 데이터 스케일링 및 이상치 제거(box plot)

- 데이터 스케일링

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_imputed), columns=df.columns)
```

- Box plot 그리기

```
# Boxplot을 통한 이상치 시각화
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_scaled)
plt.xticks(rotation=-45)
plt.title("Boxplot of Features")
plt.show()
```

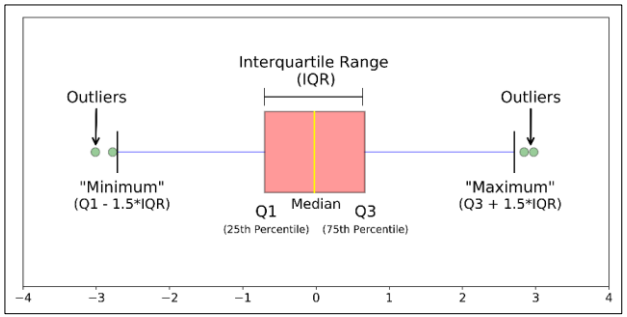


- 이상치 제거

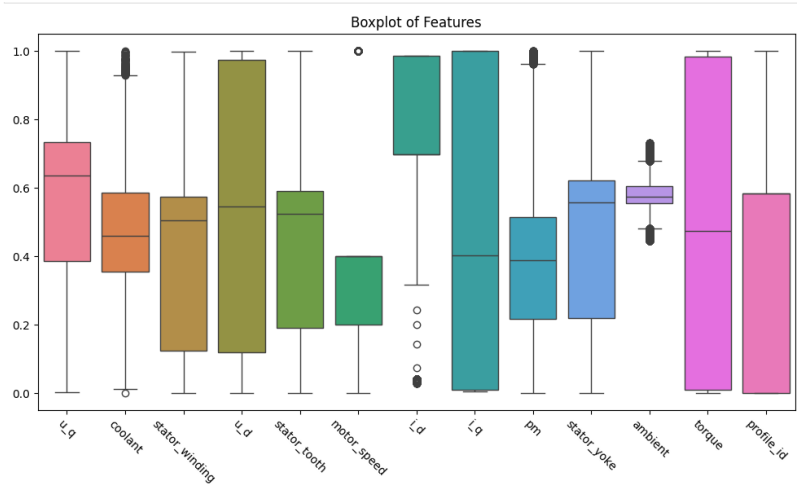
이상치 판단 기준

✓ Tukey 기준 : 하한 Q1−1.5×IQR / 상한 $Q3+1.5\times IQR$ 외의 데이터

✓ Carling 기준 : 하한 Median−2.3×IQR / 상한 Median+2.3×IQR 외의 데이터




```
# IQR 계산
Q1 = df_scaled.quantile(0.25)
Q3 = df_scaled.quantile(0.75)
IQR = Q3 - Q1
df_cleaned = df_scaled[~((df_scaled < (Q1 - 1.5 * IQR)) | (df_scaled > (Q3 + 1.5 * IQR))).any(axis=1)]
```



탐색적 데이터 분석 (EDA)-모터 온도 데이터

실습

❖ 통계적 분석

```
df_imputed.describe()
```

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambient
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	82.080538	18.737684	44.313960	-44.461346	38.016045	2658.007438	-55.146969	59.851039	44.711224	30.415893	23.371411
std	27.498737	0.430306	16.036946	40.896766	11.510279	1653.298110	48.547100	59.769524	12.306459	6.926126	0.877733
min	-0.974433	17.468487	19.049341	-97.081703	18.268650	-0.005365	-143.617004	-0.000785	24.082582	18.076689	19.850620
25%	49.993831	18.403530	26.161150	-85.316334	26.204615	999.995483	-97.258123	1.096891	34.244715	23.584067	23.172017
50%	83.362717	18.685298	49.131987	-43.000139	39.695734	1999.979126	-43.511848	53.330410	44.281654	32.877623	23.313335
75%	96.769711	19.079448	49.805905	-0.621031	43.200775	4999.945801	-2.001594	132.617340	52.280387	33.588692	23.602050
max	131.331100	19.986029	72.430313	2.103876	58.746349	4999.971191	0.004419	132.619171	69.588531	42.770332	25.875355

```
df_scaled.describe()
```

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambient
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	0.627751	0.504141	0.473289	0.530524	0.487859	0.531605	0.615995	0.451303	0.453317	0.499692	0.584389
std	0.207843	0.170923	0.300424	0.412326	0.284361	0.330661	0.338021	0.450683	0.270436	0.280482	0.145688
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.385232	0.371411	0.133227	0.118620	0.196058	0.200001	0.322785	0.008277	0.223314	0.223028	0.551293
50%	0.637442	0.483333	0.563546	0.545256	0.529355	0.399999	0.697007	0.402136	0.443878	0.599382	0.574750
75%	0.738776	0.639895	0.576171	0.972527	0.615947	0.999995	0.986033	0.999986	0.619651	0.628178	0.622671
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

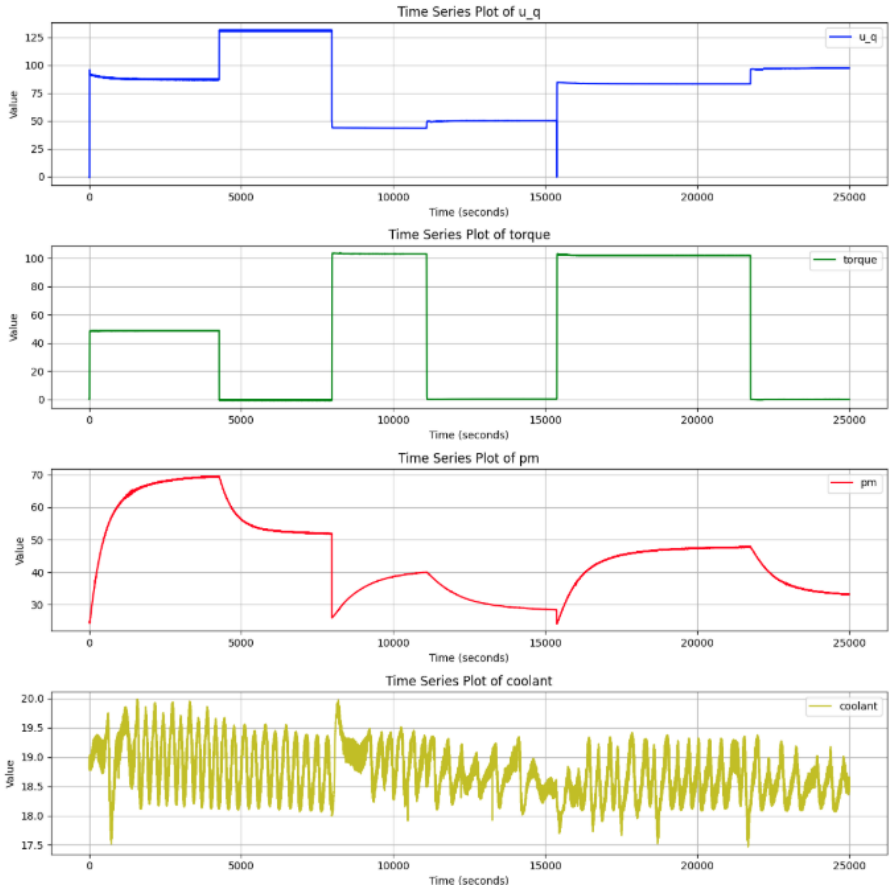
탐색적 데이터 분석 (EDA)-모터 온도 데이터

실습

❖ 데이터 시각화

- 시계열 그래프

```
df_imputed['time'] = df_imputed.index * 0.5
#그래프를 그릴 열 선택
features_to_plot = ['u_q', 'torque','pm', 'coolant'] #전류, 토크, 영구자석 온도, 냉각수 온도


plt.figure(figsize=(12, 12))
colors = ['b', 'g', 'r','y']
for i, feature in enumerate(features_to_plot):
    plt.subplot(len(features_to_plot), 1, i + 1)
    plt.plot(df_imputed['time'], df_imputed[feature], label=feature, color=colors[i])
    plt.title(f"Time Series Plot of {feature}")
    plt.xlabel("Time (seconds)")
    plt.ylabel("Value")
    plt.grid(True)
    plt.legend()

# 레이아웃 조정 및 그래프 표시
plt.tight_layout()
plt.show()
```



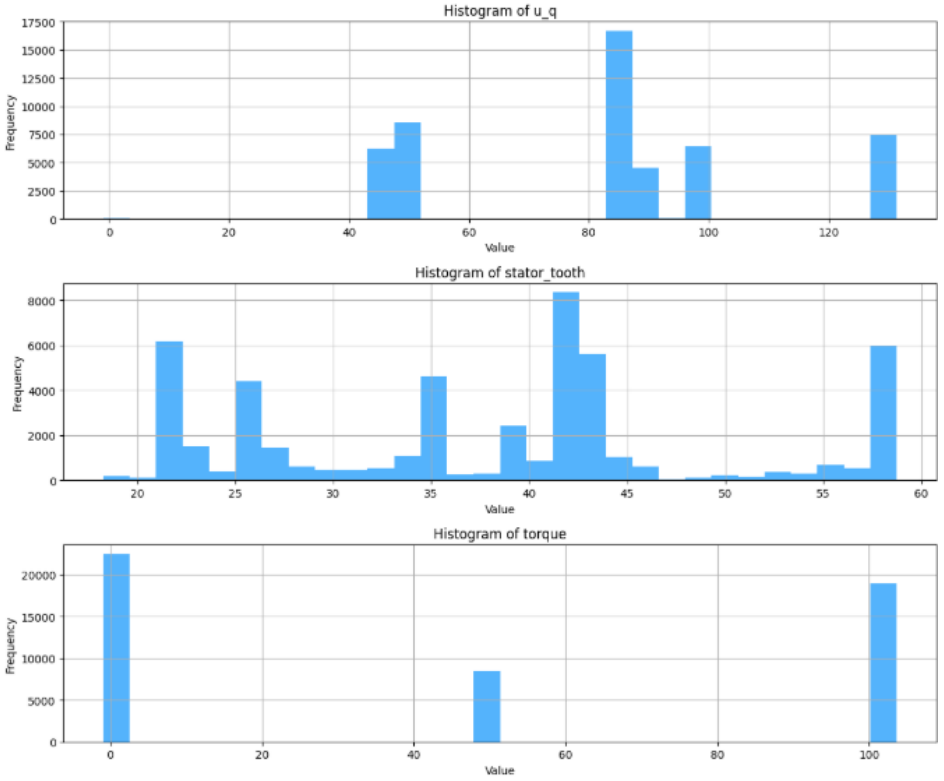
탐색적 데이터 분석 (EDA)-모터 온도 데이터

실습

❖ 데이터 시각화

- 히스토그램

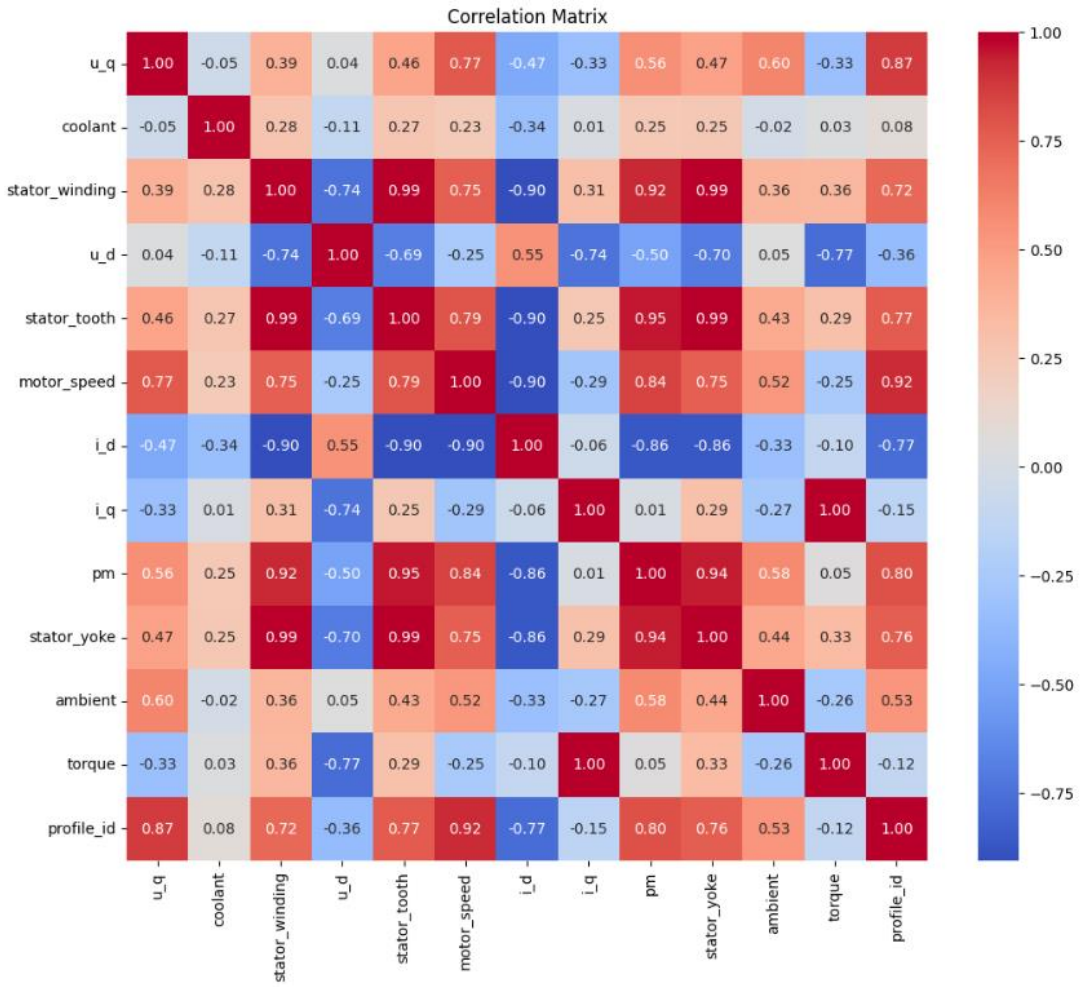
```
# 히스토그램 그릴 feature 선택
features_to_plot = ['u_q', 'stator_tooth', 'torque']

# 개별 히스토그램 그리기
plt.figure(figsize=(12, 10))

for i, feature in enumerate(features_to_plot):
    plt.subplot(len(features_to_plot), 1, i + 1)
    plt.hist(df_imputed[feature], bins=30, color='dodgerblue', alpha=0.7)
    plt.title(f"Histogram of {feature}")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.grid(True)

# 레이아웃 조정 및 그래프 표시
plt.tight_layout()
plt.show()
```



탐색적 데이터 분석 (EDA)-모터 온도 데이터

❖ 변수간 관계 분석

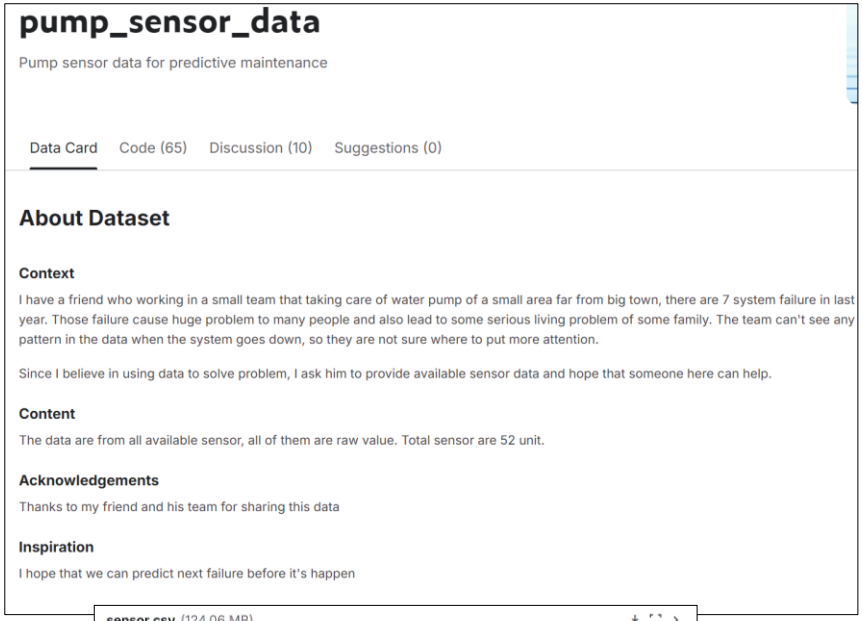
```
# 변수 간 상관관계 분석
df_imputed=df_imputed.drop('time',axis=1)
plt.figure(figsize=(12, 10))
sns.heatmap(df_imputed.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

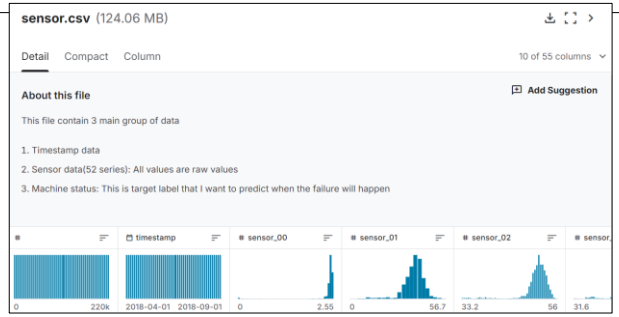


데이터 전처리-펌프 고장 데이터

❖ 데이터셋 확인 및 불러오기

- Pump sensor data (Kaggle 데이터)





✓ 펌프에 부착된 센서 데이터와 펌프 상태 데이터

- 라이브러리 import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
from sklearn.manifold import TSNE
```

✓ 실습에 필요한 라이브러리 import

```
df=pd.read_csv("data/pump_sensor.csv")
```

✓ csv 파일의 데이터 읽어오기

https://www.kaggle.com/datasets/nphantawee/pump-sensor-data/data

데이터 전처리-펌프 고장 데이터

실습

❖ 결측치 처리

- 데이터 확인

df

	Unnamed: 0	timestamp	sensor_04	sensor_06	sensor_07	sensor_08	sensor_09	machine_status
0	0	2018-04-01 0:00	634.375000	13.41146	16.13136	15.56713	15.05353	NORMAL
1	1	2018-04-01 0:01	634.375000	13.41146	16.13136	15.56713	15.05353	NORMAL
2	2	2018-04-01 0:02	638.888900	13.32465	16.03733	15.61777	15.01013	NORMAL
3	3	2018-04-01 0:03	628.125000	13.31742	16.24711	15.69734	15.08247	NORMAL
4	4	2018-04-01 0:04	636.458300	13.35359	16.21094	15.69734	15.08247	NORMAL
...	...	...	...	...	...	...	...	...
220315	220315	2018-08-31 23:55	634.722229	15.11863	16.65220	15.65393	15.16204	NORMAL
220316	220316	2018-08-31 23:56	630.902771	15.15480	16.70284	15.65393	15.11863	NORMAL
220317	220317	2018-08-31 23:57	625.925903	15.08970	16.70284	15.69734	15.11863	NORMAL
220318	220318	2018-08-31 23:58	635.648100	15.11863	16.56539	15.74074	15.11863	NORMAL
220319	220319	2018-08-31 23:59	639.814800	15.11863	16.65220	15.65393	15.01013	NORMAL

✓ 번호, 시간, 센서 데이터, 기계 상태로 구성됨
✓ 기계 상태는 숫자가 아닌 문자 데이터

- 결측치 개수 확인 및 제거

```
df.isnull().sum()
```

```
Unnamed: 0          0
timestamp           0
sensor_04          19
sensor_06        4798
sensor_07        5451
sensor_08        5107
sensor_09        4595
machine_status      0
dtype: int64
```

```
df= df.dropna()
```

```
df.isnull().sum()
```

```
Unnamed: 0         0
timestamp          0
sensor_04          0
sensor_06          0
sensor_07          0
sensor_08          0
sensor_09          0
machine_status     0
dtype: int64
```

데이터 전처리-펌프 고장 데이터

실습

❖ 범주형 데이터 처리

- 인코딩

```
conditions = [(df['machine_status'] =='NORMAL'), (df['machine_status'] =='BROKEN'), (df['machine_status'] =='RECOVERING')]
choices = [1, 0.1, 0.5]
df['Operation'] = np.select(conditions, choices, default=0)
```

```
df=df.drop(['Unnamed: 0', 'timestamp', 'machine_status'],axis=1)
```

```
df.describe()
```

	sensor_04	sensor_06	sensor_07	sensor_08	sensor_09	Operation
count	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000
mean	595.903550	12.850905	15.628197	14.992736	14.650216	0.970861
std	124.816689	2.250475	2.454559	2.232157	2.151662	0.117211
min	2.873264	0.014468	0.000000	0.028935	0.007234	0.100000
25%	625.694500	13.172740	15.776910	15.147570	14.901620	1.000000
50%	631.250000	13.353590	16.167530	15.335650	15.082470	1.000000
75%	636.111100	13.563370	16.210940	15.617770	15.118630	1.000000
max	671.990700	21.390340	23.509840	21.419270	21.274600	1.000000

데이터 전처리-펌프 고장 데이터

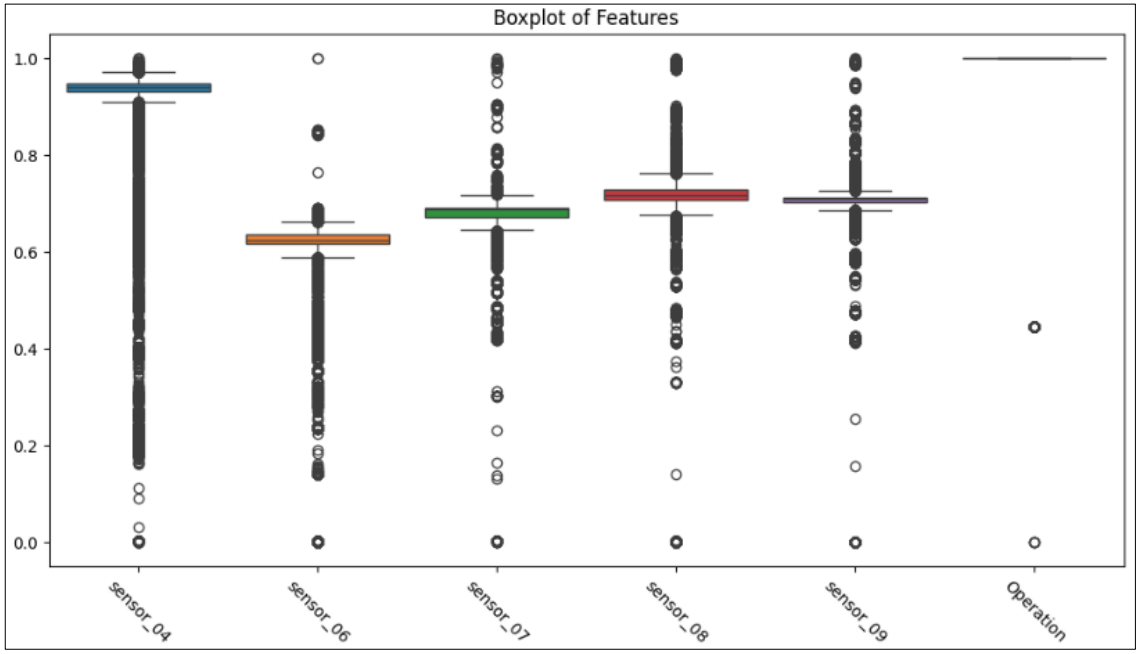
❖ 데이터 스케일링

- 데이터 스케일링

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

- Box plot

```
# Boxplot을 통한 이상치 시각화
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_scaled)
plt.xticks(rotation=-45)
plt.title("Boxplot of Features")
plt.show()
```

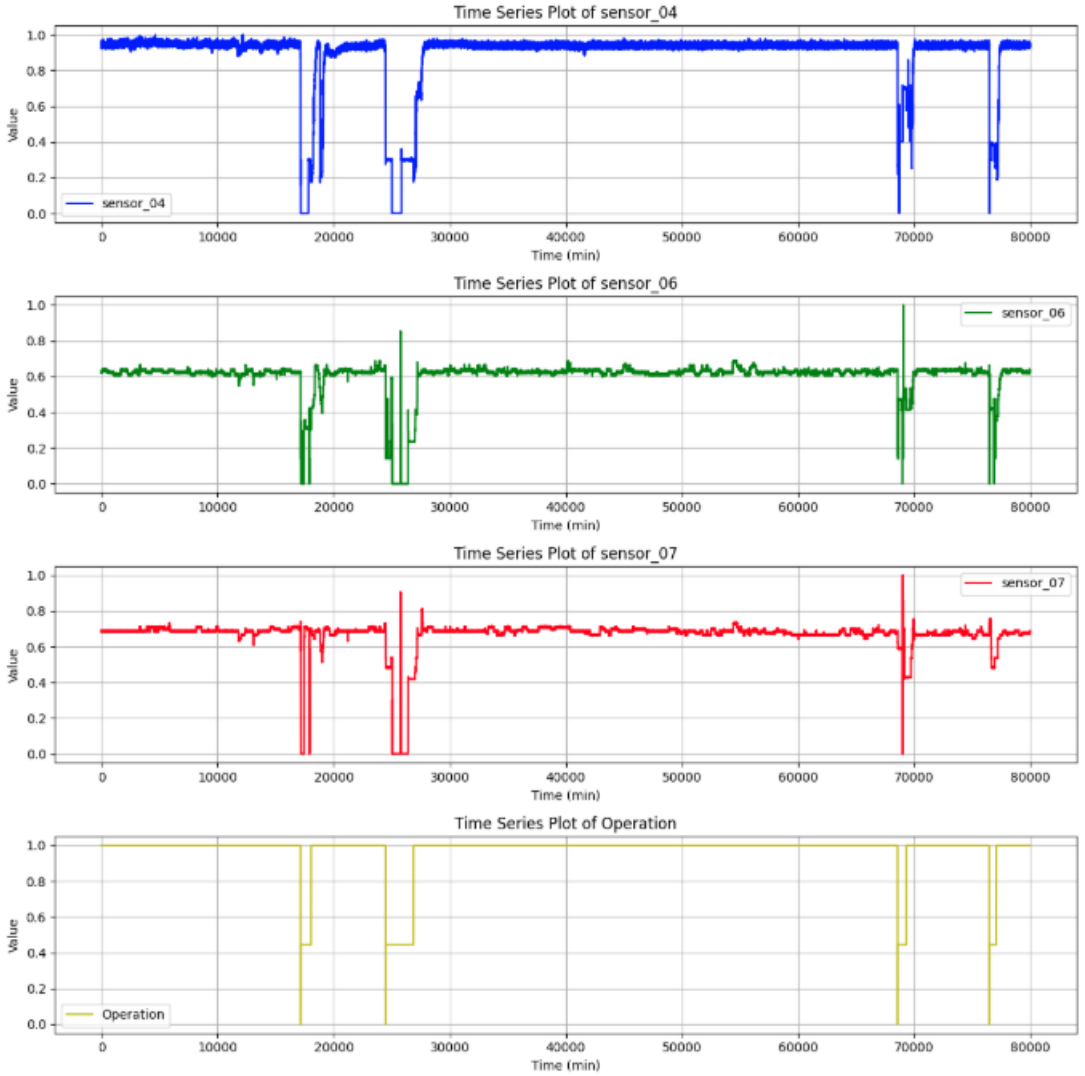


탐색적 데이터 분석 (EDA)- 펌프 고장 데이터

❖ 데이터 시각화

- 시계열 그래프

```
df_scaled['time'] = df_scaled.index
#그래프를 그릴 열 선택
features_to_plot = ['sensor_04', 'sensor_06','sensor_07', 'Operation']

plt.figure(figsize=(12, 12))
colors = ['b', 'g', 'r','y']
for i, feature in enumerate(features_to_plot):
    plt.subplot(len(features_to_plot), 1, i + 1)
    plt.plot(df_scaled[feature], label=feature, color=colors[i])
    plt.title(f"Time Series Plot of {feature}")
    plt.xlabel("Time (min)")
    plt.ylabel("Value")
    plt.grid(True)
    plt.legend()

# 레이아웃 조정 및 그래프 표시
plt.tight_layout()
plt.show()
```



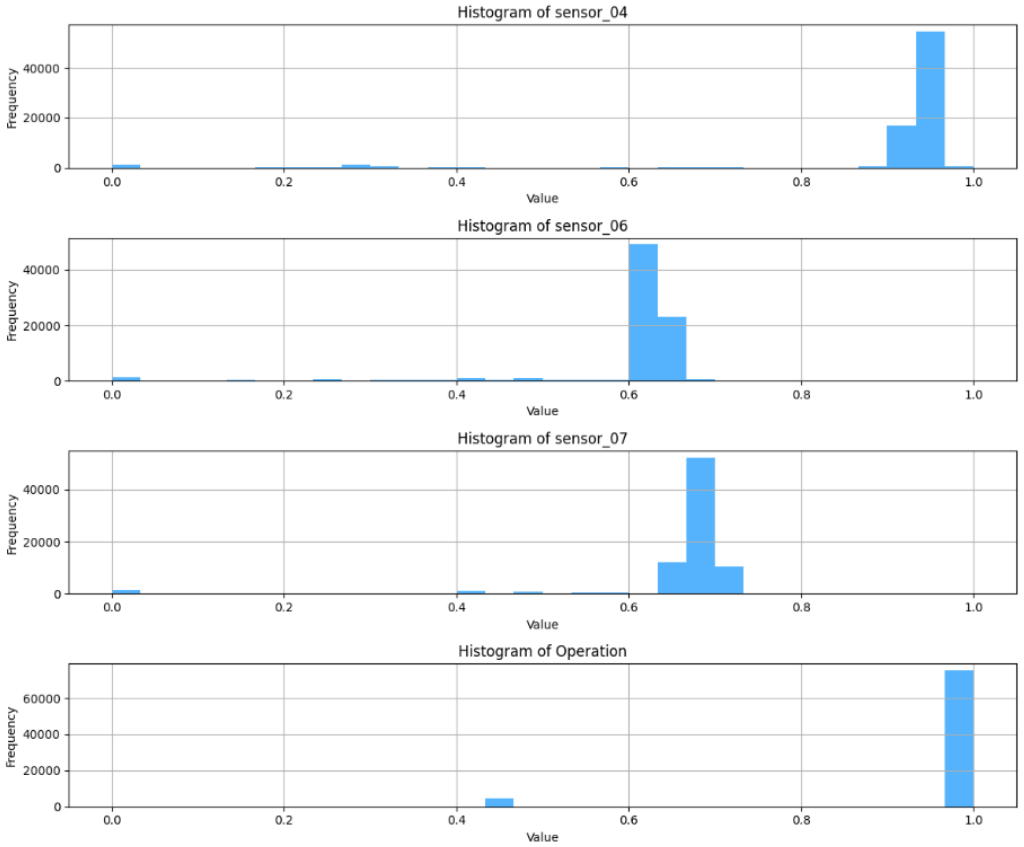
탐색적 데이터 분석 (EDA)- 펌프 고장 데이터

실습

❖ 데이터 시각화

- 히스토그램

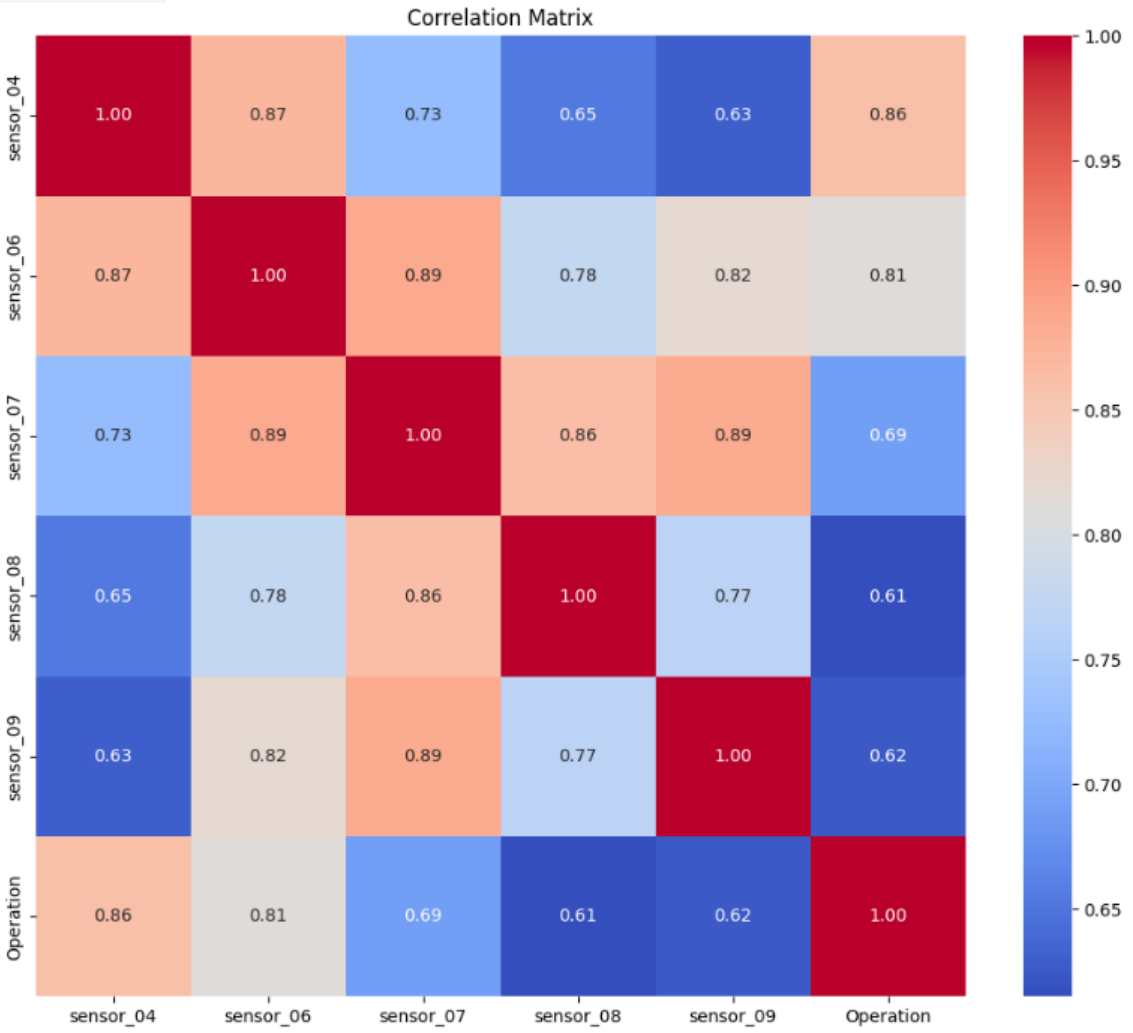
```python
# 히스토그램 그릴 feature 선택
features_to_plot = ['sensor_04', 'sensor_06','sensor_07', 'Operation']

# 개별 히스토그램 그리기
plt.figure(figsize=(12, 10))

for i, feature in enumerate(features_to_plot):
    plt.subplot(len(features_to_plot), 1, i + 1)
    plt.hist(df_scaled[feature], bins=30, color='dodgerblue', alpha=0.7)
    plt.title(f"Histogram of {feature}")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.grid(True)

# 레이아웃 조정 및 그래프 표시
plt.tight_layout()
plt.show()
```



탐색적 데이터 분석 (EDA)- 펌프 고장 데이터

❖ 변수간 관계 분석

```
# 변수 간 상관관계 분석
df_scaled=df_scaled.drop('time',axis=1)
plt.figure(figsize=(12, 10))
sns.heatmap(df_scaled.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



차원 축소 (Dimensionality reduction)

❖ t-SNE (t-Distributed Stochastic Neighbor Embedding)

- T-SNE의 기본 개념
  - ✓ 고차원 공간에서의 데이터 포인트 간의 유사성을 저차원 공간에서도 최대한 유지하도록 데이터를 재배치
  - ✓ 고차원 공간과 저차원 공간에서의 점들의 두 확률 분포 간 차이를 최소화하는 방향으로 최적화
  - ✓ 고차원과 저차원에서의 확률 분포 P와 Q 간의 Kullback-Leibler (KL) 다이버전스를 최소화

Visualizing Data using t-SNE

Laurens van der Maaten — LVDMAATEN@GMAIL.COM
TiCC
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands

Geoffrey Hinton — HINTON@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
6 King's College Road, M5S 3G4 Toronto, ON, Canada

Editor: Yoshua Bengio

Abstract

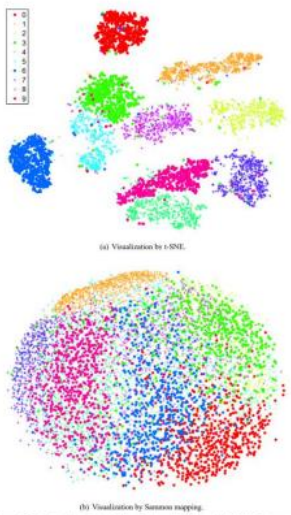
We present a new technique called "t-SNE" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE is better than existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. For visualizing the structure of very large data sets, we show how t-SNE can use random walks on neighborhood graphs to allow the implicit structure of all of the data to influence the way in which a subset of the data is displayed. We illustrate the performance of t-SNE on a wide variety of data sets and compare it with many other non-parametric visualization techniques, including Sammon mapping, Isomap, and Locally Linear Embedding. The visualizations produced by t-SNE are significantly better than those produced by the other techniques on almost all of the data sets.

Keywords: visualization, dimensionality reduction, manifold learning, embedding algorithms, multidimensional scaling

1. Introduction

Visualization of high-dimensional data is an important problem in many different domains, and deals with data of widely varying dimensionality. Cell nuclei that are relevant to breast cancer, for example, are described by approximately 30 variables (Street et al., 1993), whereas the pixel intensity vectors used to represent images or the word-count vectors used to represent documents typically have thousands of dimensions. Over the last few decades, a variety of techniques for the visualization of such high-dimensional data have been proposed, many of which are reviewed by de Oliveira and Levkowitz (2003). Important techniques include iconographic displays such as Chernoff faces (Chernoff, 1973), pixel-based techniques (Keim, 2000), and techniques that represent the dimensions in the data as vertices in a graph (Battista et al., 1994). Most of these techniques simply provide tools to display more than two data dimensions, and leave the interpretation of the

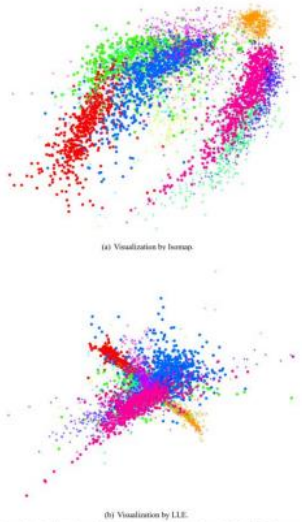
©2008 Laurens van der Maaten and Geoffrey Hinton.



(a) Visualization by t-SNE.

(b) Visualization by Sammon mapping.

Figure 2: Visualizations of 6,000 handwritten digits from the MNIST data set.



(a) Visualization by Isomap.

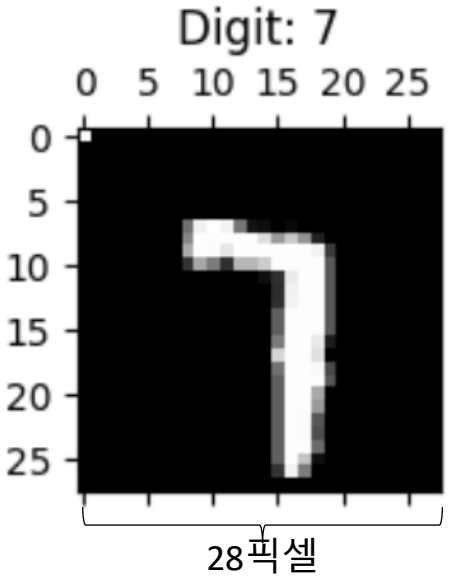
(b) Visualization by LLE.

Figure 3: Visualizations of 6,000 handwritten digits from the MNIST data set.

t-SNE 예제 – mnist data

❖ 데이터셋-mnist data

- 0-9 숫자의 손글씨 데이터



✓ 28x28 크기, 70000개 데이터

https://medium.com/@binaya.puri/mnist-dataset-for-machine-learning-8987e0b20bb3

t-SNE 예제 – mnist data

❖ 데이터 전처리

- 라이브러리 import

```
1 from __future__ import print_function
2 import time
3
4 import numpy as np
5 import pandas as pd
6
7 from sklearn.datasets import fetch_openml
8 from sklearn.decomposition import PCA
9 from sklearn.manifold import TSNE
10
11 %matplotlib inline
12 import matplotlib.pyplot as plt
13 from mpl_toolkits.mplot3d import Axes3D
14
15 import seaborn as sns
```

- Mnist data 불러오기

```
1 mnist = fetch_openml('mnist_784', version=1)
2 X = mnist.data / 255.0
3 y = mnist.target
4
5 print(X.shape, y.shape)
```

```
(70000, 784) (70000,)
```

t-SNE 예제 – mnist data

❖ 데이터 전처리

❖ 데이터 정리 및 라벨 부여

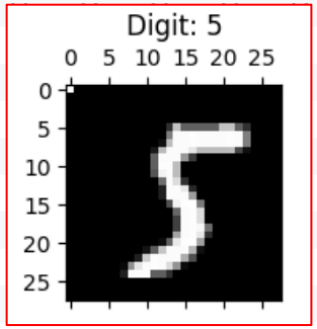
```
1 feat_cols = [ 'pixel'+str(i) for i in range(X.shape[1]) ]
2
3 df = pd.DataFrame(X,columns=feat_cols)
4 df['y'] = y
5 df['label'] = df['y'].apply(lambda i: str(i))
6
7 X, y = None, None
8
9 print('Size of the dataframe: {}'.format(df.shape))
10
```

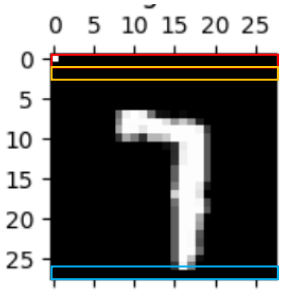
```
Size of the dataframe: (70000, 786)
```

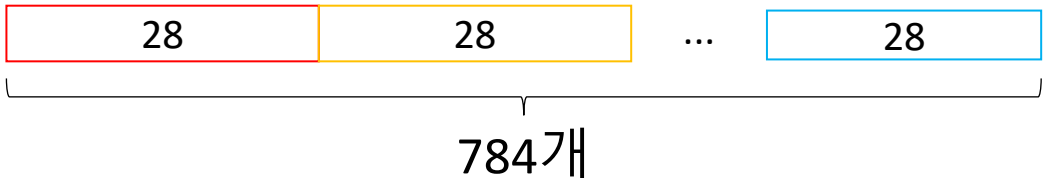
```
1 df
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	..
0	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	NaN	0.0	0.0	0.0	0.0						
3	NaN	0.0	0.0	0.0	0.0						
4	NaN	0.0	0.0	0.0	0.0						
...	...	...	...	...	...						
69995	NaN	0.0	0.0	0.0	0.0						
69996	NaN	0.0	0.0	0.0	0.0						
69997	NaN	0.0	0.0	0.0	0.0						
69998	NaN	0.0	0.0	0.0	0.0						
69999	NaN	0.0	0.0	0.0	0.0						

70000 rows × 786 columns







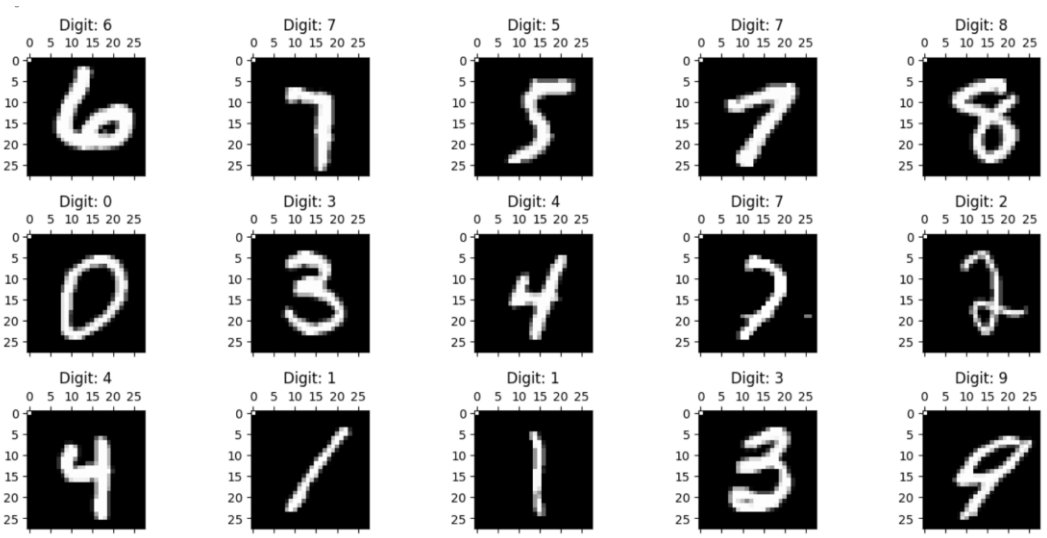
- ✓ 픽셀 수 784개 + 라벨 y(숫자형 데이터) + 라벨 y(문자형 데이터)
- ✓ 하나의 행에 하나의 숫자 데이터를 저장함
- ✓ 이미지 데이터를 데이터 프레임으로 나타내는 과정

t-SNE 예제 – mnist data

❖ 데이터 전처리

- Mnist data 시각화

```
# For reproducability of the results
np.random.seed(10)
rndperm = np.random.permutation(df.shape[0])

plt.gray()
fig = plt.figure( figsize=(16,7) )
fig.subplots_adjust(hspace=0.5)
for i in range(0,15):
    ax = fig.add_subplot(3,5,i+1, title="Digit: {}".format(str(df.loc[rndperm[i],'label'])) )
    ax.matshow(df.loc[rndperm[i],feat_cols].values.reshape((28,28)).astype(float))
plt.show()
```



t-SNE 예제 – mnist data

실습

❖ 데이터 전처리

❖ 데이터 정리 및 데이터프레임 생성

```
1 N = 10000
2
3 # NaN 값이 있을 경우 이를 0으로 채움
4 df[feat_cols] = df[feat_cols].fillna(0)
5
6 df_subset = df.loc[rndperm[:N],:].copy()
7
8 data_subset = df_subset[feat_cols].values
```

- ✓ 시간 절약을 위해 10000개의 데이터만 사용
- ✓ NaN 값은 데이터 처리 시 오류 발생 원인이므로 0으로 대체

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	y	label
21955	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6	6
24094	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7	7
47968	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5	5
51361	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7	7
19662	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
45107	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6	6
31123	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4	4
37823	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5	5
4096	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2	2
47420	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8	8

10000 rows × 786 columns

t-SNE 예제 – mnist data

실습

❖ t-SNE 정의 및 실행

- 파라미터 설정 및 실행

```
1 time_start = time.time()
2 tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
3 tsne_results = tsne.fit_transform(data_subset)
4
5 print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))
```

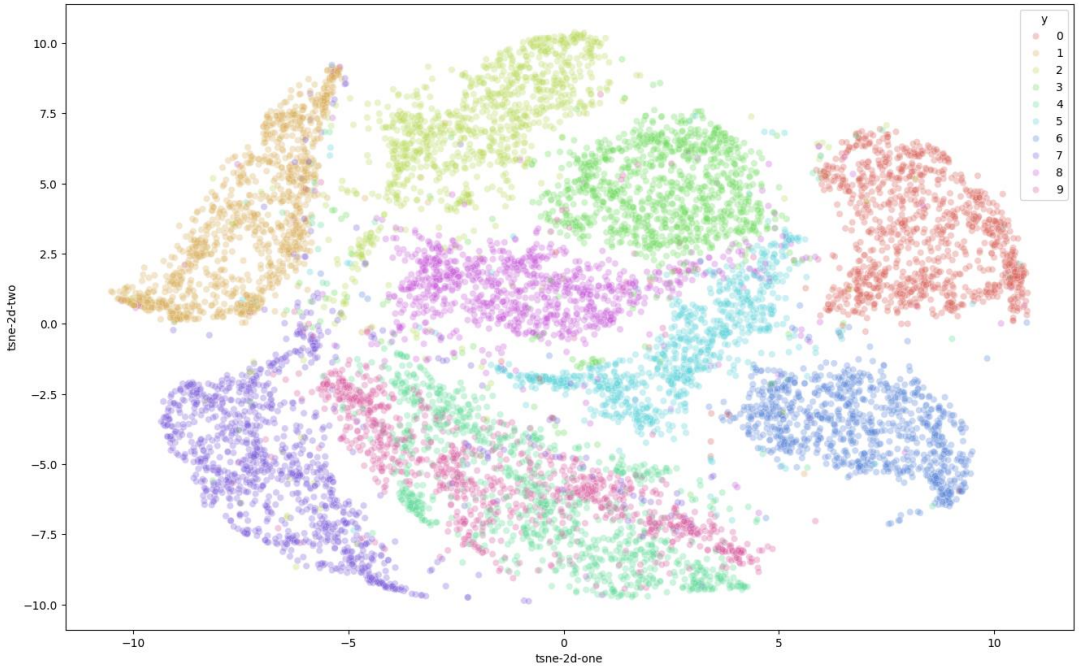
```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_t_sne.py:1162: FutureWar
  warnings.warn(
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 10000 samples in 0.037s...
[t-SNE] Computed neighbors for 10000 samples in 6.447s...
[t-SNE] Computed conditional probabilities for sample 1000 / 10000
[t-SNE] Computed conditional probabilities for sample 2000 / 10000
[t-SNE] Computed conditional probabilities for sample 3000 / 10000
[t-SNE] Computed conditional probabilities for sample 4000 / 10000
[t-SNE] Computed conditional probabilities for sample 5000 / 10000
[t-SNE] Computed conditional probabilities for sample 6000 / 10000
[t-SNE] Computed conditional probabilities for sample 7000 / 10000
[t-SNE] Computed conditional probabilities for sample 8000 / 10000
[t-SNE] Computed conditional probabilities for sample 9000 / 10000
[t-SNE] Computed conditional probabilities for sample 10000 / 10000
[t-SNE] Mean sigma: 2.127103
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.727661
[t-SNE] KL divergence after 300 iterations: 2.781246
t-SNE done! Time elapsed: 52.95404553413391 seconds
```

t-SNE 예제 – mnist data

❖ 결과 확인

- 데이터 시각화

```
df_subset['tsne-2d-one'] = tsne_results[:,0]
df_subset['tsne-2d-two'] = tsne_results[:,1]

plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3
)
```



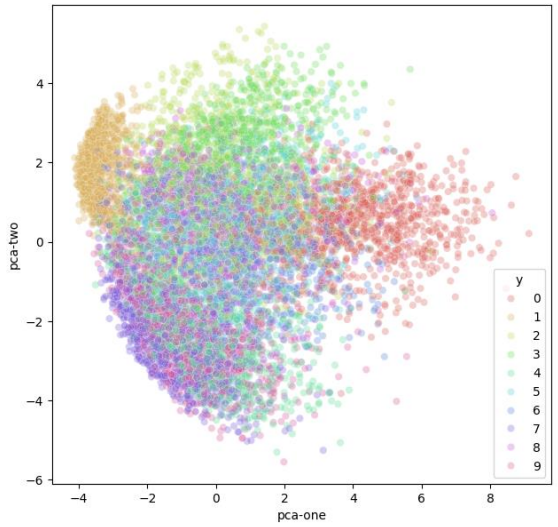
시각화 모델 비교

❖ PCA와 t-SNE 비교

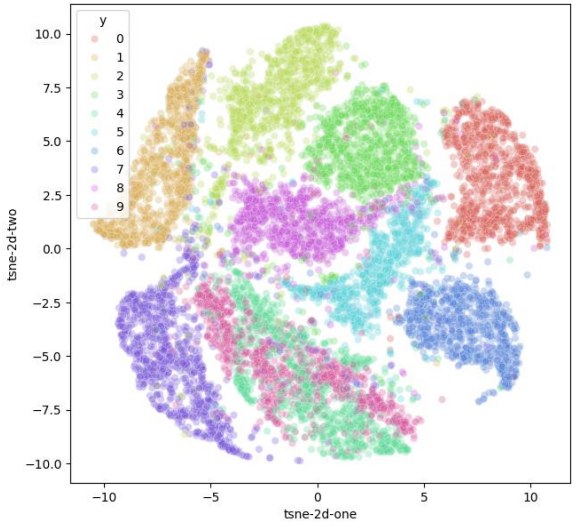
```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_subset)

df_subset['pca-one'] = pca_result[:,0]
df_subset['pca-two'] = pca_result[:,1]


print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))

```

```
Explained variation per principal component: [0.09862132 0.07236824]
```

```
plt.figure(figsize=(16,7))

ax1 = plt.subplot(1, 2, 1)
sns.scatterplot(
    x="pca-one", y="pca-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax1
)

ax2 = plt.subplot(1, 2, 2)
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="y",
    palette=sns.color_palette("hls", 10),
    data=df_subset,
    legend="full",
    alpha=0.3,
    ax=ax2
)
```



PCA 차원축소 결과



t-SNE 차원축소 결과

t-SNE

❖ 학습 파라미터

```
def __init__(n_components=2, *, perplexity=30.0, early_exaggeration=12.0, learning_rate='auto', max_iter=None,
n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', metric_params=None, init='pca', verbose=0, random_state=None,
method='barnes_hut', angle=0.5, n_jobs=None, n_iter='deprecated')
```

탭에서 열기 소스 보기

T-distributed Stochastic Neighbor Embedding.
t-SNE [1] is a tool to visualize high-dimensional data. It converts
similarities between data points to joint probabilities and tries
to minimize the Kullback-Leibler divergence between the joint
probabilities of the low-dimensional embedding and the
high-dimensional data. t-SNE has a cost function that is not convex,
i.e. with different initializations we can get different results.
It is highly recommended to use another dimensionality reduction
method (e.g. PCA for dense data or TruncatedSVD for sparse data)

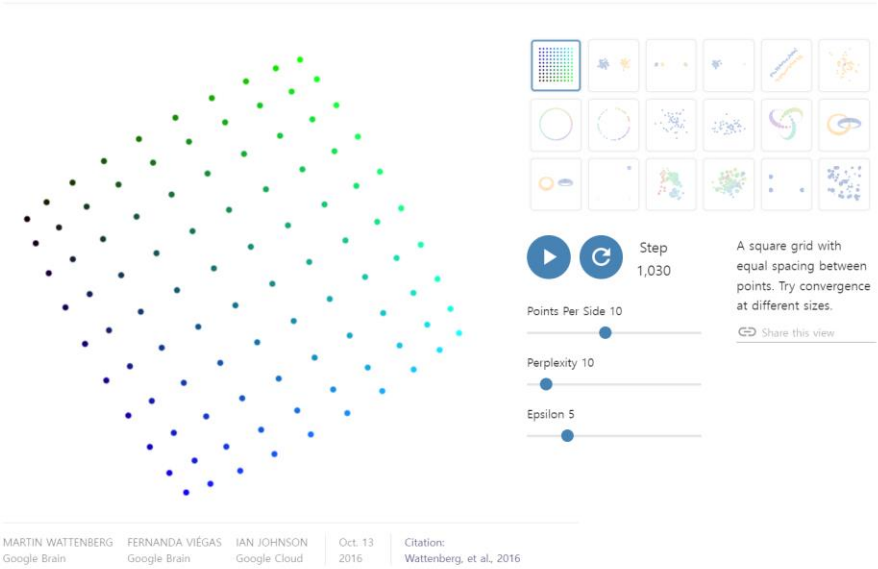
```
TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300,learning_rate=10)
```

✓ 난감도(perplexity)
군집 크기와 관련, 작을 작은 군집

perplexity : float, default=30.0
The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results. The perplexity must be less than the number of samples.

✓ 학습률(learning_rate)
학습속도, 수렴, local optima와 관련

learning_rate : float or "auto", default="auto"
The learning rate for t-SNE is usually in the range [10.0, 1000.0]. If the learning rate is too high, the data may look like a 'ball' with any point approximately equidistant from its nearest neighbours. If the learning rate is too low, most points may look compressed in a dense cloud with few outliers. If the cost function gets stuck in a bad local minimum increasing the learning rate may help. Note that many other t-SNE implementations (bhtsne, FIt-SNE, openTSNE, etc.) use a definition of learning_rate that is 4 times smaller than ours. So our learning_rate=200 corresponds to learning_rate=800 in those other implementations. The 'auto' option sets the learning_rate to `max(N / early_exaggeration / 4, 50)` where N is the sample size, following [4] and [5].



https://distill.pub/2016/misread-tsne/

감사합니다