AI활용

21860019 서주완

Linear-Regression

보험료 예측 모델

Linear-Regression

데이터셋 분석
Age – 나이
Sex – 성별
Bmi – 체지방지수
Children – 자녀수
Smoker – 흡연여부
Region – 거주지역
Expenses - 보험료

Linear-Regression

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler


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

```
df = pd.read_csv('/content/drive/MyDrive/ai/insurance.csv')
```

Linear-Regression

```
df.head(5)
df.isnull().sum().sort_values(ascending=False)
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
expenses    0
dtype: int64
```

df

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86
...	...	...	...	...	...	...	...
1333	50	male	31.0	3	no	northwest	10600.55
1334	18	female	31.9	0	no	northeast	2205.98
1335	18	female	36.9	0	no	southeast	1629.83
1336	21	female	25.8	0	no	southwest	2007.95
1337	61	female	29.1	0	yes	northwest	29141.36

1338 rows × 7 columns

Linear-Regression

```
le = LabelEncoder()
df['sex'] = le.fit_transform(df['sex'])

# 전처리된 데이터 확인
print(df['sex'])
```

```
0       0
1       1
2       1
3       1
4       1
       ..
1333    1
1334    0
1335    0
1336    0
1337    0
Name: sex, Length: 1338, dtype: int64
```

```
df['smoker'] = le.fit_transform(df['smoker'])

# 전처리된 데이터 확인
print(df['smoker'])
```

```
0       1
1       0
2       0
3       0
4       0
       ..
1333    0
1334    0
1335    0
1336    0
1337    1
Name: smoker, Length: 1338, dtype: int64
```

```
df['region'] = le.fit_transform(df['region'])

# 전처리된 데이터 확인
print(df['region'])
```

```
0       3
1       2
2       2
3       1
4       1
       ..
1333    1
1334    0
1335    2
1336    3
1337    1
Name: region, Length: 1338, dtype: int64
```

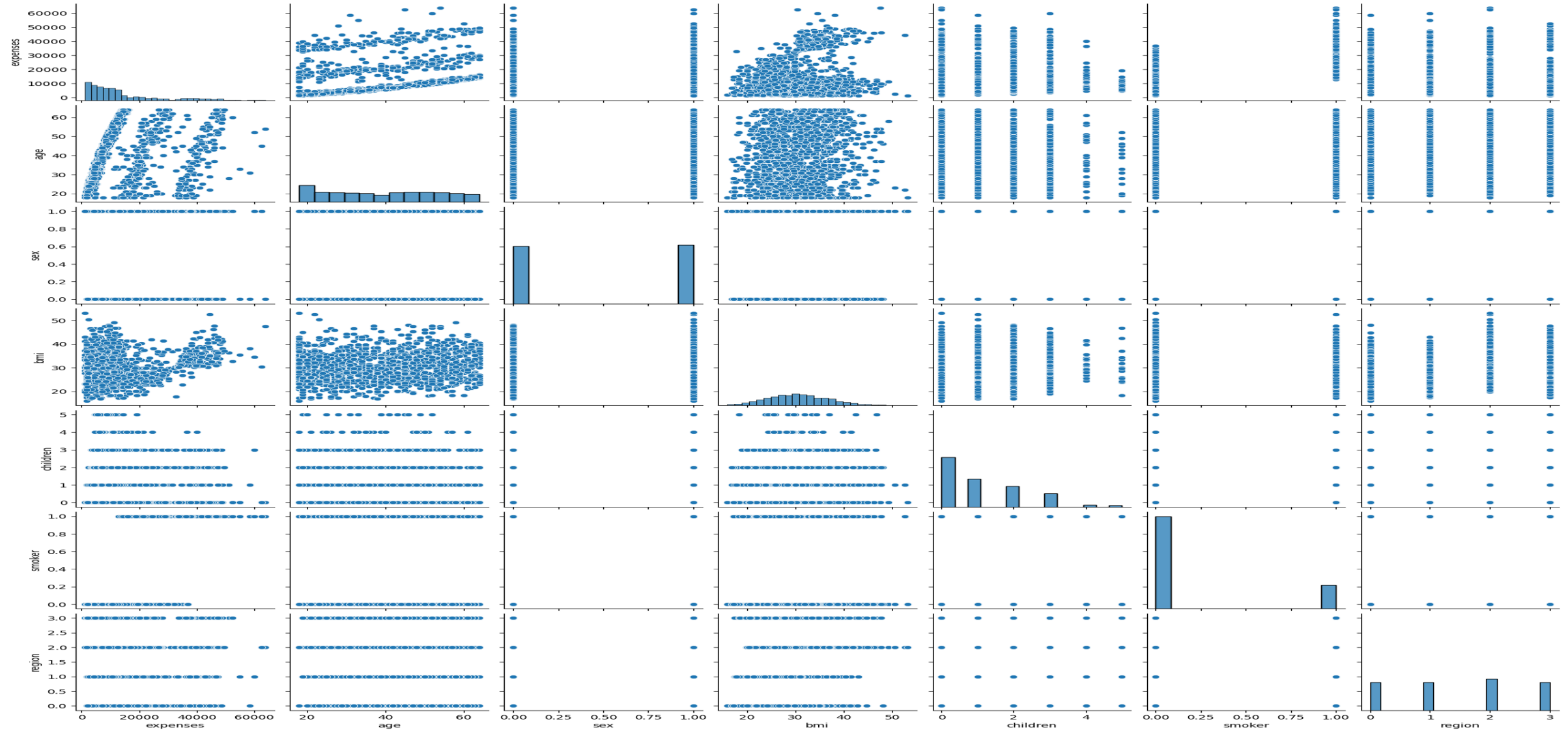
Linear-Regression

```
df_corr = df.corr()
df_corr_sort = df_corr.sort_values("expenses", ascending=False)
df_corr_sort['expenses'].head(10)
```

```
expenses    1.000000
smoker      0.787251
age         0.299008
bmi         0.198576
children    0.067998
sex         0.057292
region     -0.006208
Name: expenses, dtype: float64
```

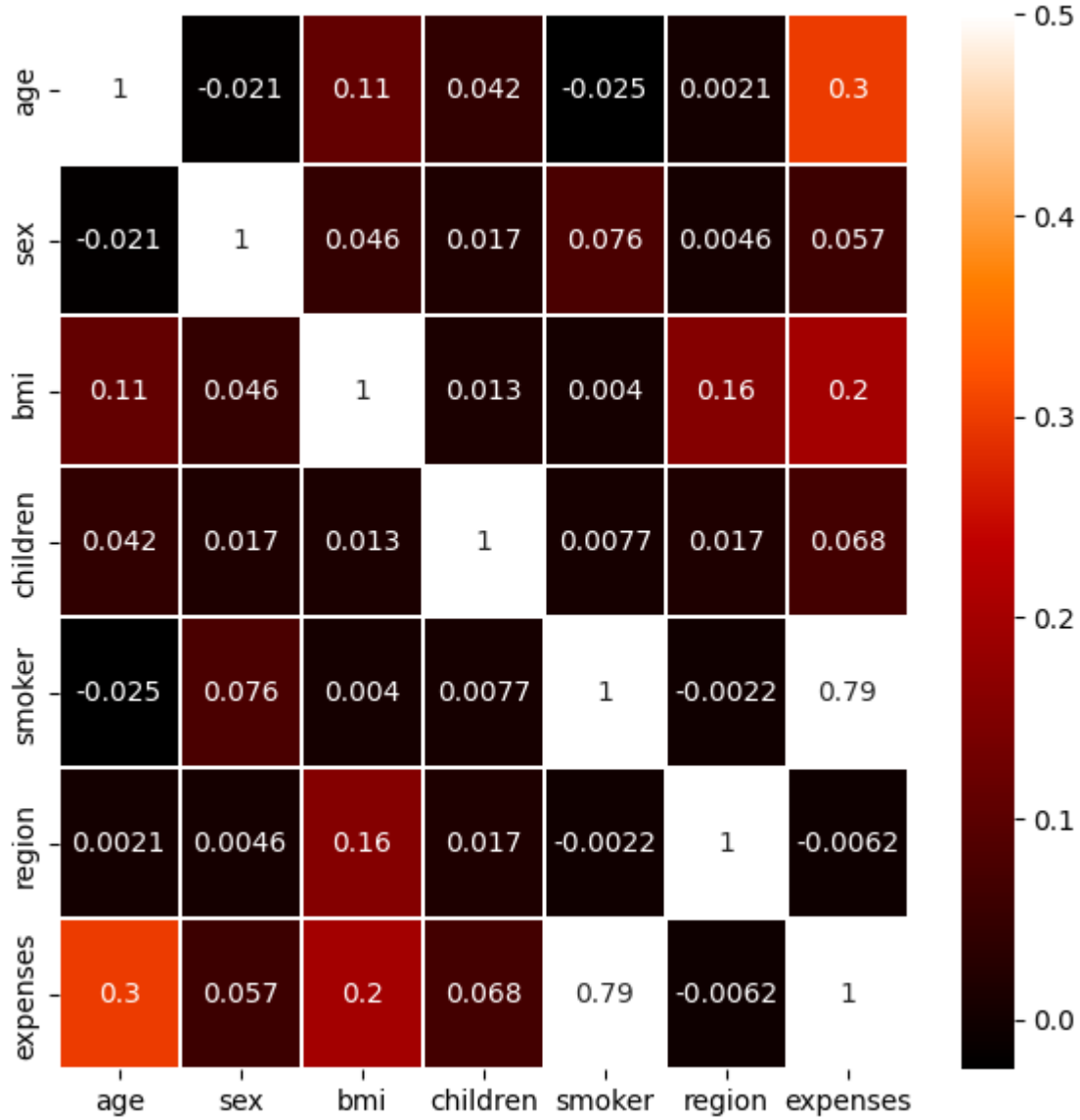
Linear-Regression

```
cols = ['expenses','age', 'sex', 'bmi', 'children', 'smoker','region']
sns.pairplot(df[cols])
plt.show()
```



Linear-Regression

```
colormap = plt.cm.gist_heat
plt.figure(figsize=(7,7))
sns.heatmap(df.corr(), linewidths=0.1, vmax=0.5, cmap=colormap, linecolor='white', annot=True)
plt.show()
```



Linear-Regression

```
cols_train = ['age', 'sex', 'bmi', 'children', 'smoker','region']
X_train_pre = df[cols_train]
y= df['expenses'].values

X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_size=0.2, shuffle=True)
```

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

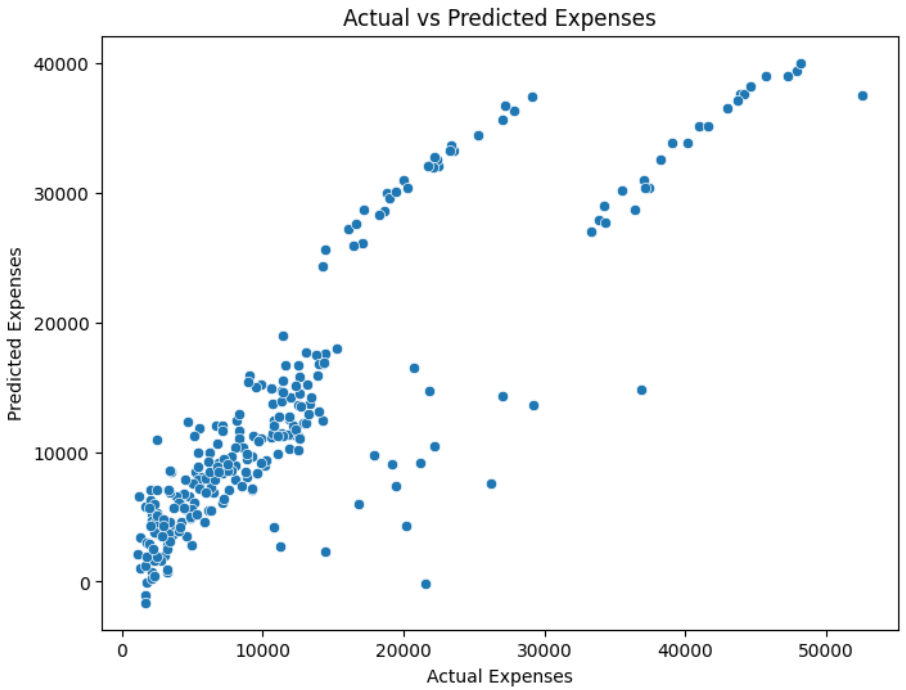

```
y_pred = model.predict(X_test)
```


```
# 성능 평가
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
# 결과 출력
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
print('R-squared Score:', r2)
```

```
Mean Absolute Error: 3974.4165959075804
Mean Squared Error: 31658845.955824777
Root Mean Squared Error: 5626.619407408393
R-squared Score: 0.7461054175658259
```

Linear-Regression




```
# 예측 결과 시각화
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel('Actual Expenses')
plt.ylabel('Predicted Expenses')
plt.title('Actual vs Predicted Expenses')
plt.show()
```

Linear-Regression

```
model = Sequential()

model.add(Dense(81, input_dim=6, activation='linear'))
model.add(Dense(1, activation='linear'))
model.summary()

modelpath = "./data/model/all{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath = modelpath, verbose=0, save_best_only = True)
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=2000, batch_size=500, validation_split=0.2, callbacks=[checkpointer,early_stopping])
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 81)                567

 dense_7 (Dense)             (None, 1)                 82

=================================================================
Total params: 649
Trainable params: 649
Non-trainable params: 0
```

Linear-Regression

```
Epoch 1985/2000
2/2 [==============================] - 0s 79ms/step - loss: 131905912.0000 - accuracy: 0.0000e+00 - val_loss: 126473120.0000 - val_accuracy: 0.0000e+00
Epoch 1986/2000
2/2 [==============================] - 0s 82ms/step - loss: 131902800.0000 - accuracy: 0.0000e+00 - val_loss: 126470504.0000 - val_accuracy: 0.0000e+00
Epoch 1987/2000
2/2 [==============================] - 0s 83ms/step - loss: 131899912.0000 - accuracy: 0.0000e+00 - val_loss: 126467752.0000 - val_accuracy: 0.0000e+00
Epoch 1988/2000
2/2 [==============================] - 0s 61ms/step - loss: 131897000.0000 - accuracy: 0.0000e+00 - val_loss: 126465456.0000 - val_accuracy: 0.0000e+00
Epoch 1989/2000
2/2 [==============================] - 0s 59ms/step - loss: 131893456.0000 - accuracy: 0.0000e+00 - val_loss: 126462728.0000 - val_accuracy: 0.0000e+00
Epoch 1990/2000
2/2 [==============================] - 0s 77ms/step - loss: 131890536.0000 - accuracy: 0.0000e+00 - val_loss: 126460344.0000 - val_accuracy: 0.0000e+00
Epoch 1991/2000
2/2 [==============================] - 0s 84ms/step - loss: 131887184.0000 - accuracy: 0.0000e+00 - val_loss: 126457696.0000 - val_accuracy: 0.0000e+00
Epoch 1992/2000
2/2 [==============================] - 0s 77ms/step - loss: 131883984.0000 - accuracy: 0.0000e+00 - val_loss: 126454960.0000 - val_accuracy: 0.0000e+00
Epoch 1993/2000
2/2 [==============================] - 0s 84ms/step - loss: 131881016.0000 - accuracy: 0.0000e+00 - val_loss: 126452304.0000 - val_accuracy: 0.0000e+00
Epoch 1994/2000
2/2 [==============================] - 0s 79ms/step - loss: 131879904.0000 - accuracy: 0.0000e+00 - val_loss: 126449160.0000 - val_accuracy: 0.0000e+00
Epoch 1995/2000
2/2 [==============================] - 0s 80ms/step - loss: 131875104.0000 - accuracy: 0.0000e+00 - val_loss: 126447336.0000 - val_accuracy: 0.0000e+00
Epoch 1996/2000
2/2 [==============================] - 0s 62ms/step - loss: 131871504.0000 - accuracy: 0.0000e+00 - val_loss: 126445128.0000 - val_accuracy: 0.0000e+00
Epoch 1997/2000
2/2 [==============================] - 0s 76ms/step - loss: 131868536.0000 - accuracy: 0.0000e+00 - val_loss: 126442920.0000 - val_accuracy: 0.0000e+00
Epoch 1998/2000
2/2 [==============================] - 0s 75ms/step - loss: 131865344.0000 - accuracy: 0.0000e+00 - val_loss: 126440480.0000 - val_accuracy: 0.0000e+00
Epoch 1999/2000
2/2 [==============================] - 0s 79ms/step - loss: 131862176.0000 - accuracy: 0.0000e+00 - val_loss: 126437760.0000 - val_accuracy: 0.0000e+00
Epoch 2000/2000
2/2 [==============================] - 0s 63ms/step - loss: 131859848.0000 - accuracy: 0.0000e+00 - val_loss: 126435528.0000 - val_accuracy: 0.0000e+00
```

Linear-Regression

Linear-Regression은 loss값이 높게 나오는 것이 일반적인 현상이다.

손실 값이 높다고 모델이 잘못되었거나 문제가 있는 것은 아니다.

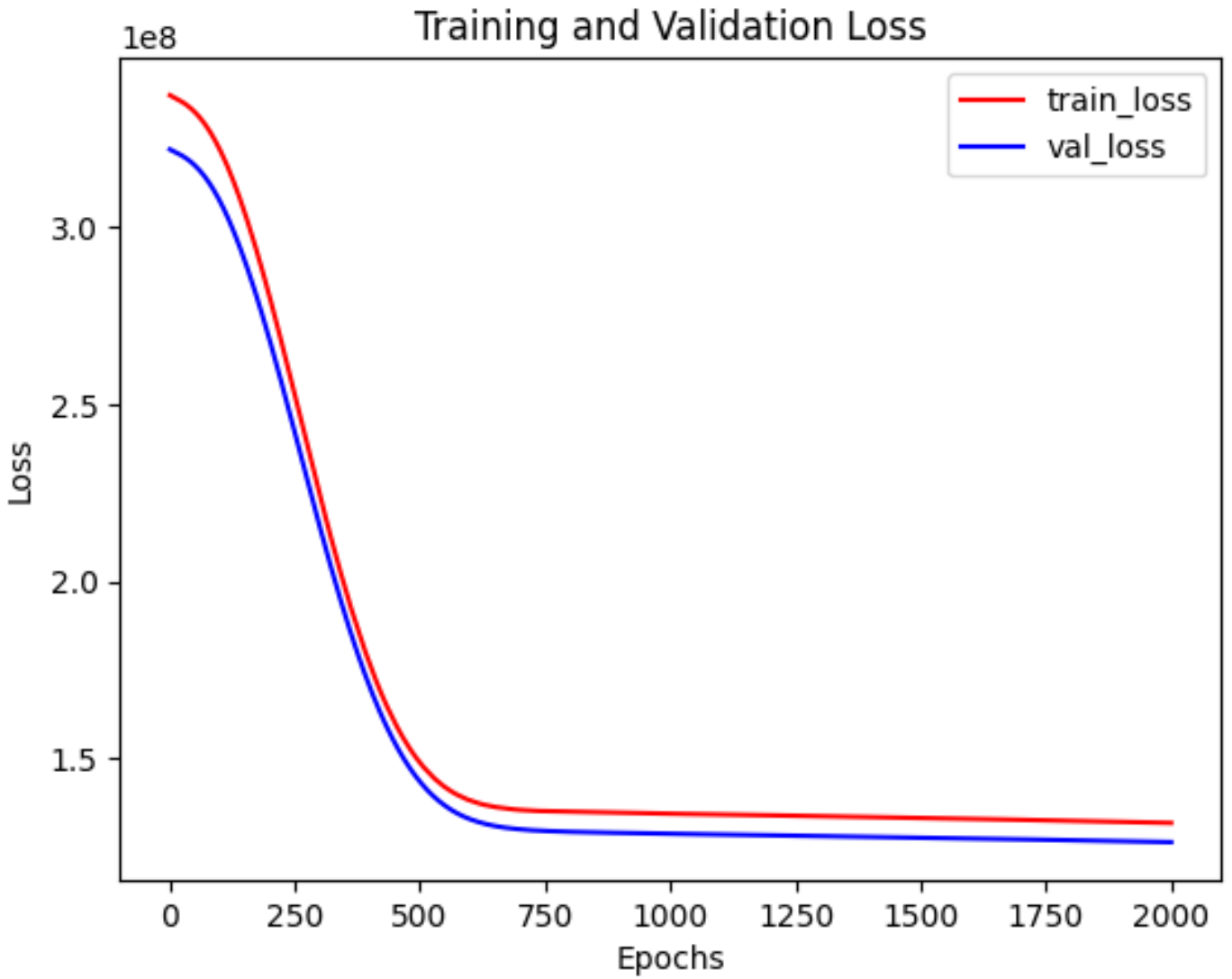
평균 제곱 오차를 사용하여 손실을 계산하며, 손실 값은 데이터의 크기와 범위에 따라 달라질 수 있다. 손실 값 자체보다는 R-squared score를 주로 사용하여 모델의 성능을 평가한다.

Linear-Regression

```
# Loss 값 추출
loss = history.history['loss']
val_loss = history.history['val_loss']

# 그래프 그리기
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'r', label='train_loss')
plt.plot(epochs, val_loss, 'b', label='val_loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Binary-CrossEntropy

수질 판별 후 식수 가능한지?

Binary-CrossEntropy

데이터셋 분석
pH – 물의 산성 or 알칼리성 정도
Hardness – 물의 칼슘과 마그네슘 함량
Solids – 물에 용해된 총 불순물의 양
Chloramines – 물 처리에 사용되는 염소와 암모니아의 혼합물(소독효과)
Sulfate – 물에 용해된 황산염의 양
Conductivity – 물의 전기 전도도
Organic_carbon – 물에 용해된 유기 탄소의 양
Trihalomethanes – 물 처리에 의해 생성된 삼할로메탄류 화합물의 양
Turbidity – 물의 탁도
Potability – 물의 음용 가능 여부

Binary-CrossEntropy

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

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

```
df = pd.read_csv('/content/drive/MyDrive/ai/water_potability.csv')
```

```
df.head(5)
df.isnull().sum().sort_values(ascending=False)
```

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

Binary-CrossEntropy

df

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	8.316766	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	100.341674	4.628771	0
1	9.092223	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
2	5.584087	188.313324	28748.68774	7.544869	326.678363	280.467916	8.399735	54.917862	2.559708	0
3	10.223862	248.071735	28749.71654	7.513408	393.663395	283.651634	13.789695	84.603556	2.672989	0
4	8.635849	203.361523	13672.09176	4.563009	303.309771	474.607645	12.363817	62.798309	4.401425	0
...	...	...	...	...	...	...	...	...	...	...
2006	8.989900	215.047358	15921.41202	6.297312	312.931021	390.410231	9.899115	55.069304	4.613843	1
2007	6.702547	207.321086	17246.92035	7.708117	304.510230	329.266002	16.217303	28.878601	3.442983	1
2008	11.491011	94.812545	37188.82602	9.263166	258.930600	439.893618	16.172755	41.558501	4.369264	1
2009	6.069616	186.659040	26138.78019	7.747547	345.700257	415.886955	12.067620	60.419921	3.669712	1
2010	4.668102	193.681736	47580.99160	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1

2011 rows × 10 columns

Binary-CrossEntropy

```
df["Potability"].value_counts()
```

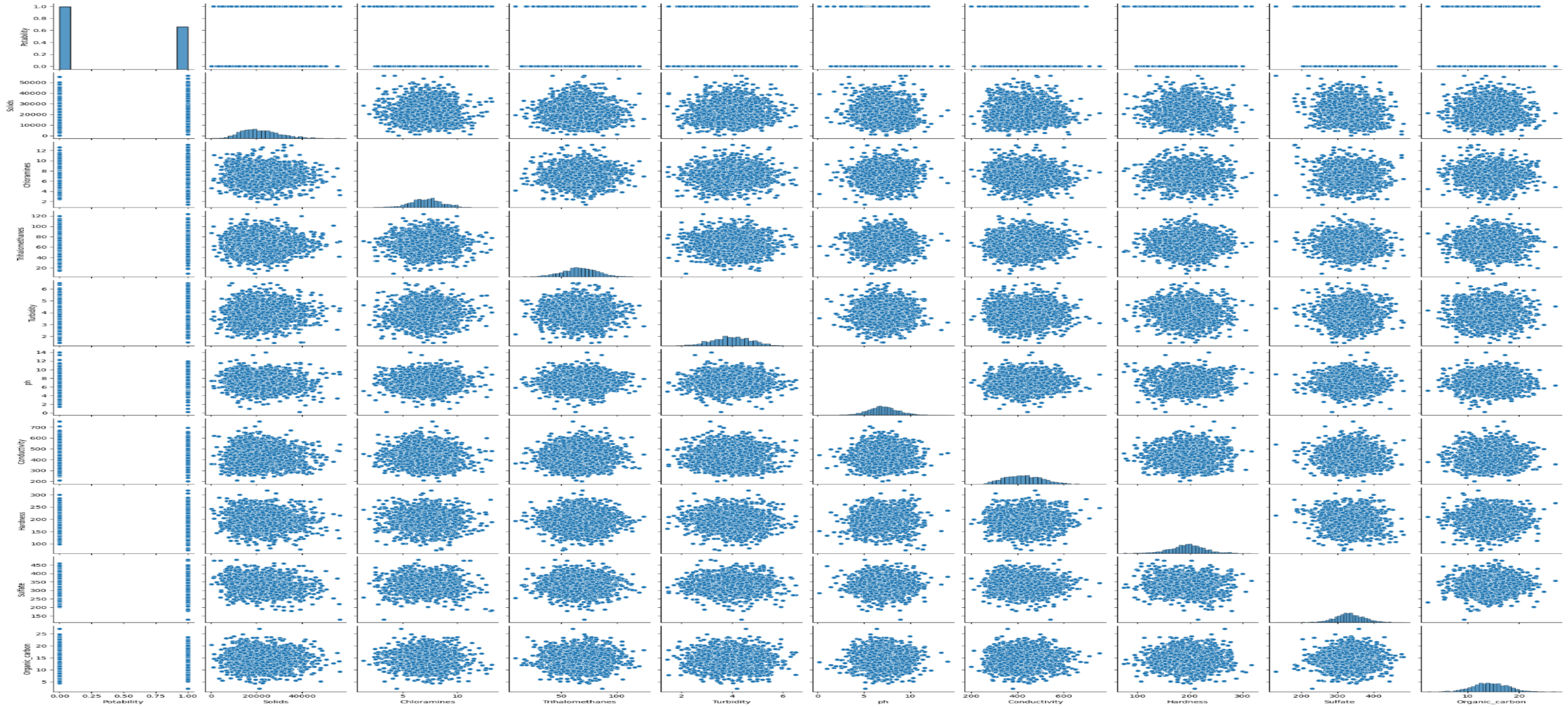
```
0    1200
1     811
Name: Potability, dtype: int64
```

```
df_corr = df.corr()
df_corr_sort = df_corr.sort_values("Potability", ascending=False)
df_corr_sort['Potability'].head(10)
```

```
Potability          1.000000
Solids              0.040674
Turbidity           0.022682
Chloramines         0.020784
ph                  0.014530
Trihalomethanes     0.009244
Hardness           -0.001505
Sulfate            -0.015303
Conductivity       -0.015496
Organic_carbon     -0.015567
Name: Potability, dtype: float64
```

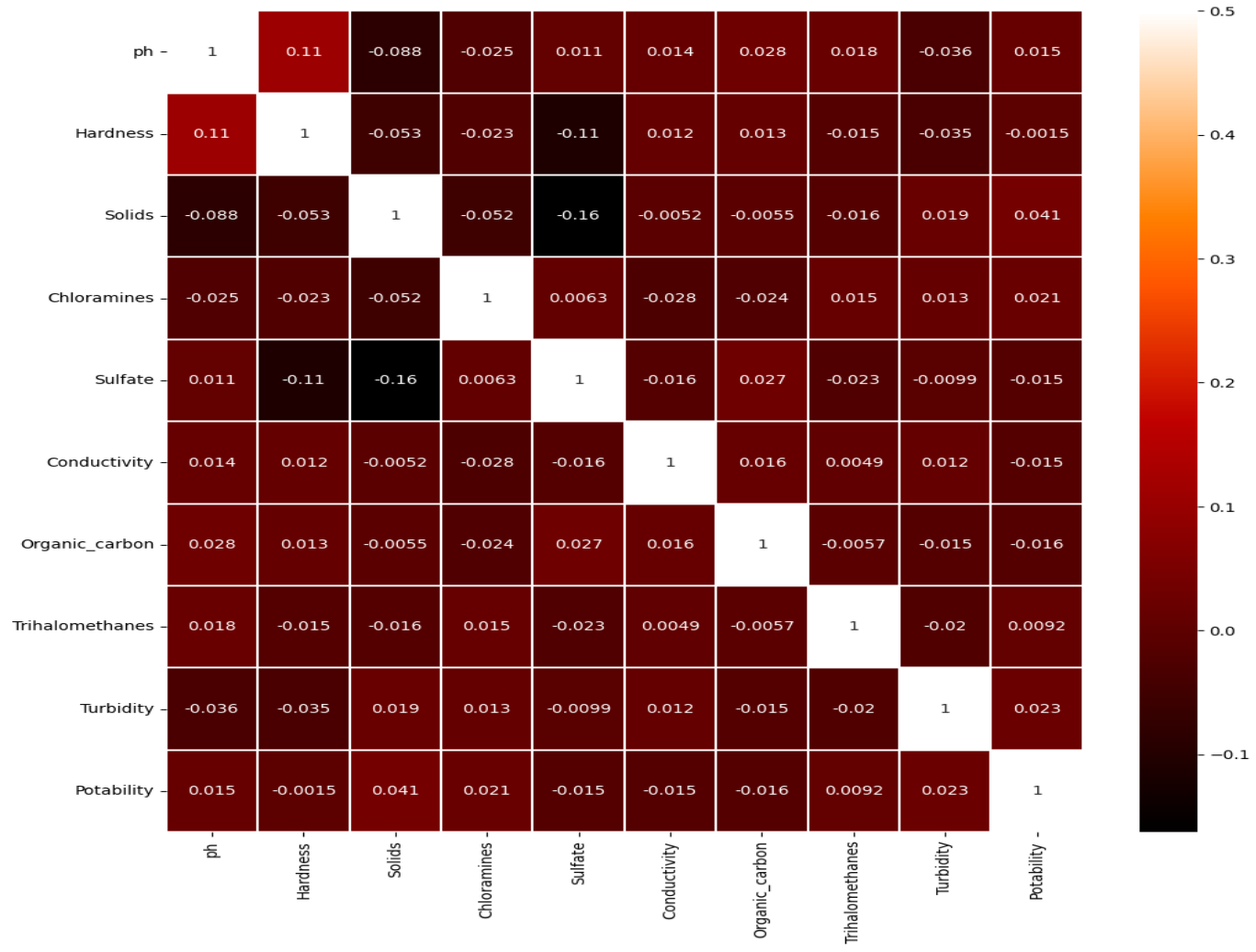
Binary-CrossEntropy

```
cols = ['Potability', 'Solids', 'Chloramines', 'Trihalomethanes', 'Turbidity' , 'ph', 'Conductivity', 'Hardness', 'Sulfate', 'Organic_carbon' ]
sns.pairplot(df[cols])
plt.show()
```



Binary-CrossEntropy

```
colormap = plt.cm.gist_heat
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(), linewidths=0.1, vmax=0.5, cmap=colormap, linecolor='white', annot=True)
plt.show()
```



Binary-CrossEntropy

```
print(df.shape[1])
```

```
10
```

```
cols_train = ['Solids', 'Chloramines', 'Trihalomethanes', 'Turbidity' , 'ph', 'Conductivity', 'Hardness', 'Sulfate', 'Organic_carbon' ]
X_train_pre = df[cols_train]
y= df['Potability'].values

X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_size=0.2, shuffle=True)
```

```
model = Sequential()

model.add(Dense(100, input_dim=9, activation='relu'))
model.add(Dense(90, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

modelpath = "./data/model/all{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath = modelpath, verbose=0, save_best_only = True)
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y, epochs=2000, batch_size=50, validation_split=0.2, callbacks=[checkpointer,early_stopping])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 100)               1000

 dense_21 (Dense)            (None, 90)                9090

 dense_22 (Dense)            (None, 100)               9100

 dense_23 (Dense)            (None, 50)                5050

 dense_24 (Dense)            (None, 100)               5100

 dense_25 (Dense)            (None, 50)                5050

 dense_26 (Dense)            (None, 25)                1275

 dense_27 (Dense)            (None, 25)                650

 dense_28 (Dense)            (None, 5)                 130

 dense_29 (Dense)            (None, 1)                 6

=================================================================
Total params: 36,451
Trainable params: 36,451
Non-trainable params: 0
```

Binary-CrossEntropy

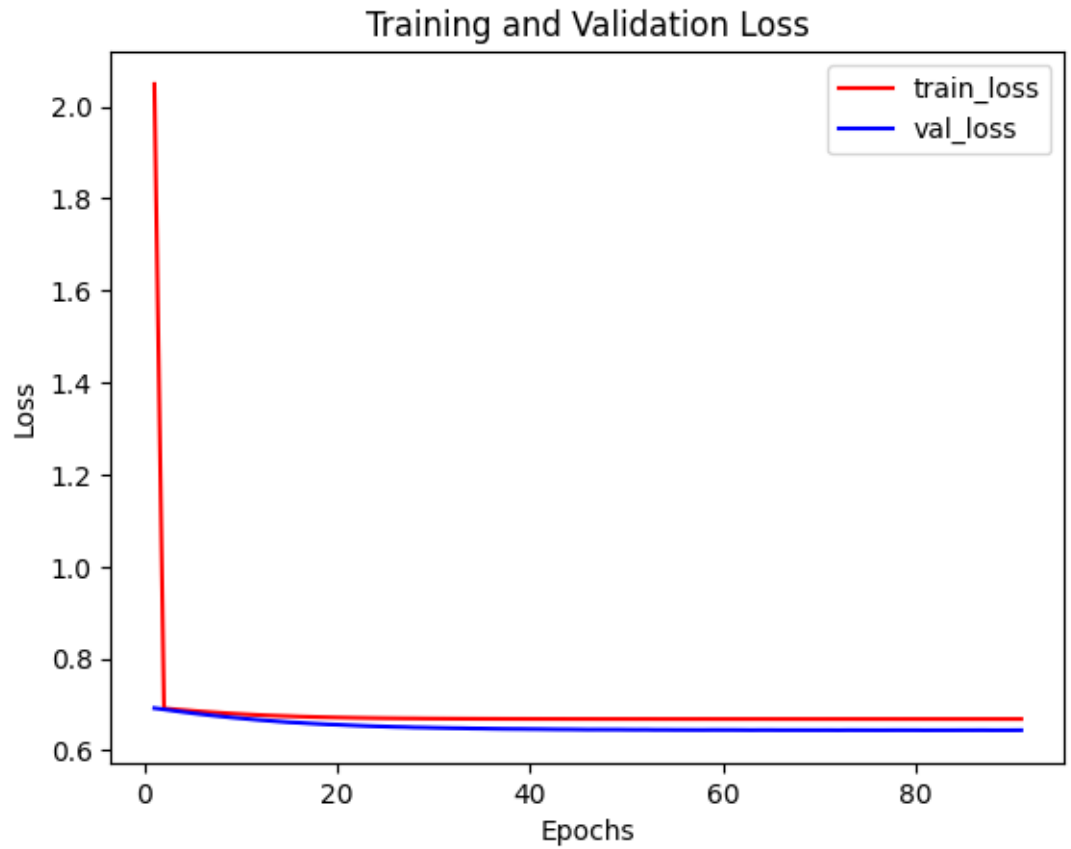
```
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 77/2000
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 78/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 79/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 80/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 81/2000
26/26 [==============================] - 0s 6ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 82/2000
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 83/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 84/2000
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 85/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 86/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 87/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
Epoch 88/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 89/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 90/2000
26/26 [==============================] - 0s 5ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6442 - val_accuracy: 0.6646
Epoch 91/2000
26/26 [==============================] - 0s 4ms/step - loss: 0.6686 - accuracy: 0.6104 - val_loss: 0.6443 - val_accuracy: 0.6646
```

Binary-CrossEntropy

```
# Loss 값 추출
loss = history.history['loss']
val_loss = history.history['val_loss']

# 그래프 그리기
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'r', label='train_loss')
plt.plot(epochs, val_loss, 'b', label='val_loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Multi-classfication

와인 품질 예측

Multi-classfication

데이터 셋 분석

1 - fixed acidity
2 – volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 – alcohol
12 - quality (score between 0 and 10)

1.고정산도
2.휘발성산도
3.구연산
4.잔류 당분
5.염화물
6.유리황산
7.총황산
8.밀도
9.pH
10.황산염
11.알코올
12.품질 (0에서 10까지의 점수)

Multi-classfication

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df = pd.read_csv('/content/drive/MyDrive/ai/WineQT.csv')
```

+ 코드 + 텍스트

```
df.head(5)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	Id	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	1	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	2	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	3	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	4	5

Multi-classfication

```
df.isnull().sum().sort_values(ascending=False)
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
Id                      0
quality                 0
dtype: int64
```

```
df["quality"].value_counts()
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	Id	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	0	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	1	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	2	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	3	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	4	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1138	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	1592	6
1139	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	1593	6
1140	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	1594	5
1141	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	1595	6
1142	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	1597	5

1143 rows × 13 columns

Multi-classfication

```
df_corr = df.corr()
df_corr_sort = df_corr.sort_values("quality", ascending=False)
df_corr_sort['quality'].head(12)
```

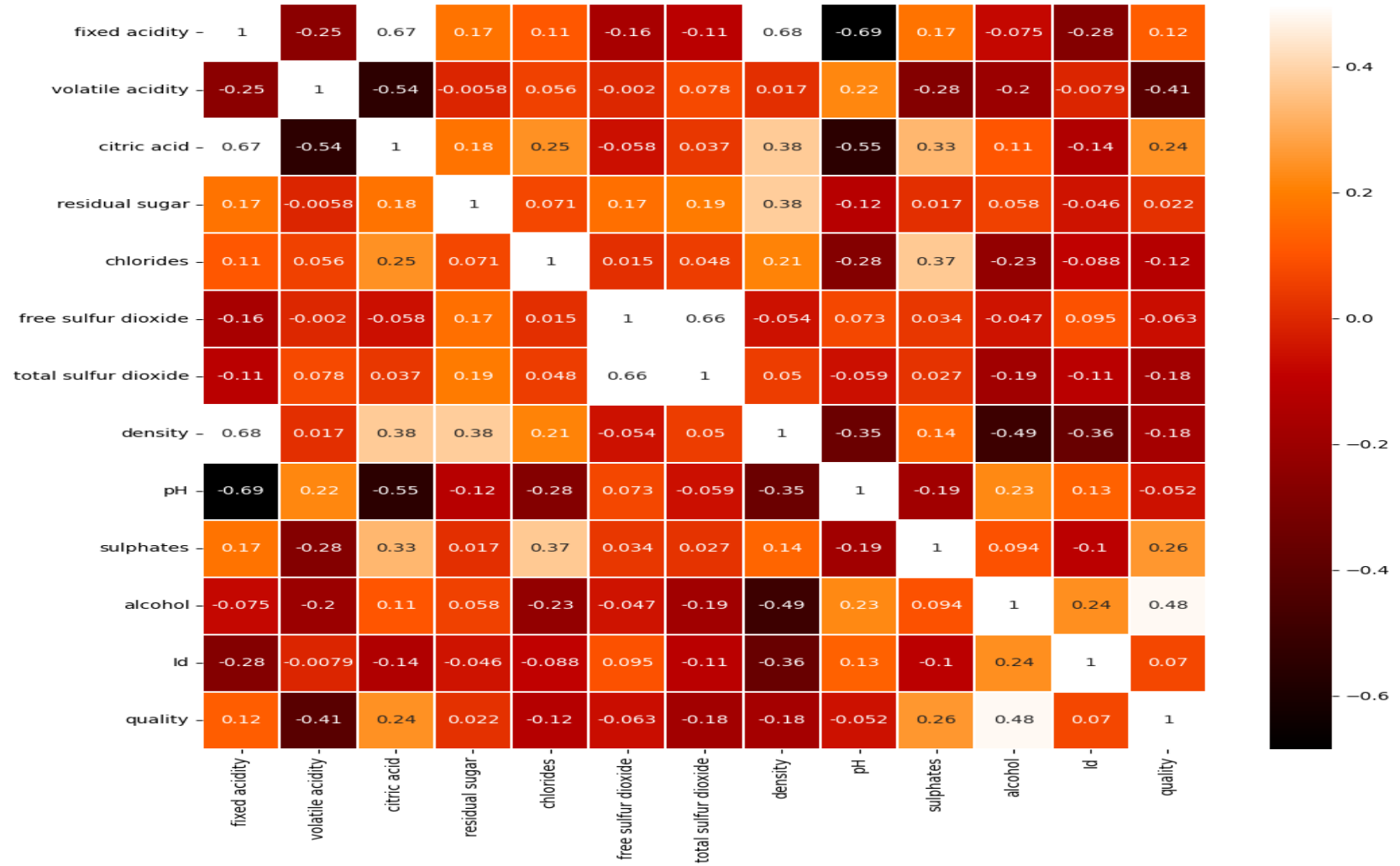
```
quality                1.000000
alcohol                0.484866
sulphates              0.257710
citric acid            0.240821
fixed acidity          0.121970
Id                     0.069708
residual sugar         0.022002
pH                    -0.052453
free sulfur dioxide   -0.063260
chlorides             -0.124085
density               -0.175208
total sulfur dioxide  -0.183339
Name: quality, dtype: float64
```

Multi-classfication

```
cols = ['quality', 'alcohol', 'sulphates', 'citric acid', 'fixed acidity' , 'residual sugar', 'pH', 'free sulfur dioxide', 'chlorides', 'density' ]
sns.pairplot(df[cols])
plt.show()
```

Multi-classfication

```
colormap = plt.cm.gist_heat
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(), linewidths=0.1, vmax=0.5, cmap=colormap, linecolor='white', annot=True)
plt.show()
```



Multi-classfication

```
print(df.shape[1])
```

```
13
```

```
cols_train = ["fixed acidity", "volatile acidity", "citric acid", "residual sugar", "chlorides", "free sulfur dioxide",
              "total sulfur dioxide", "density", "pH", "sulphates", "alcohol", "Id"]
X_train_pre = df[cols_train]

y= df['quality'].values
num_classes = 9

X_train, X_test, y_train, y_test = train_test_split(X_train_pre, y, test_size=0.2, shuffle=True)
y_train_encoded = to_categorical(y_train, num_classes)
```

```
model = Sequential()
num_classes = 9

model.add(Dense(72, input_dim=12, activation='relu'))
model.add(Dense(144, activation='relu'))
model.add(Dense(144, activation='relu'))
model.add(Dense(144, activation='relu'))
model.add(Dense(72, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

modelpath = "./data/model/all{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath = modelpath, verbose=0, save_best_only = True)
early_stopping = EarlyStopping(monitor='val_loss', patience=50)

model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train_encoded, epochs=2000, batch_size=300, validation_split=0.2, callbacks=[checkpointer, early_stopping])
```

Multi-classfication

```
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_188 (Dense)           (None, 72)                936

 dense_189 (Dense)           (None, 144)               10512

 dense_190 (Dense)           (None, 144)               20880

 dense_191 (Dense)           (None, 144)               20880

 dense_192 (Dense)           (None, 72)                10440

 dense_193 (Dense)           (None, 9)                 657

=================================================================
Total params: 64,305
Trainable params: 64,305
Non-trainable params: 0
```

```
Epoch 81/2000
3/3 [==============================] - 0s 16ms/step - loss: 0.2085 - accuracy: 0.5732 - val_loss: 0.2839 - val_accuracy: 0.4262
Epoch 82/2000
3/3 [==============================] - 0s 25ms/step - loss: 0.2119 - accuracy: 0.5732 - val_loss: 0.2472 - val_accuracy: 0.4317
Epoch 83/2000
3/3 [==============================] - 0s 14ms/step - loss: 0.2098 - accuracy: 0.5321 - val_loss: 0.2905 - val_accuracy: 0.3934
Epoch 84/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.2214 - accuracy: 0.5103 - val_loss: 0.2619 - val_accuracy: 0.4044
Epoch 85/2000
3/3 [==============================] - 0s 19ms/step - loss: 0.2053 - accuracy: 0.5486 - val_loss: 0.2367 - val_accuracy: 0.4426
Epoch 86/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.2027 - accuracy: 0.5267 - val_loss: 0.2513 - val_accuracy: 0.4262
Epoch 87/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.2023 - accuracy: 0.5540 - val_loss: 0.2457 - val_accuracy: 0.4262
Epoch 88/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.1988 - accuracy: 0.5691 - val_loss: 0.2428 - val_accuracy: 0.4153
Epoch 89/2000
3/3 [==============================] - 0s 21ms/step - loss: 0.1998 - accuracy: 0.5595 - val_loss: 0.2436 - val_accuracy: 0.4262
Epoch 90/2000
3/3 [==============================] - 0s 21ms/step - loss: 0.1973 - accuracy: 0.5650 - val_loss: 0.2380 - val_accuracy: 0.4481
Epoch 91/2000
3/3 [==============================] - 0s 19ms/step - loss: 0.2087 - accuracy: 0.5198 - val_loss: 0.2365 - val_accuracy: 0.4262
Epoch 92/2000
3/3 [==============================] - 0s 19ms/step - loss: 0.2056 - accuracy: 0.5075 - val_loss: 0.2440 - val_accuracy: 0.4153
Epoch 93/2000
3/3 [==============================] - 0s 20ms/step - loss: 0.2030 - accuracy: 0.5376 - val_loss: 0.2776 - val_accuracy: 0.3880
Epoch 94/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.2077 - accuracy: 0.5540 - val_loss: 0.2347 - val_accuracy: 0.4372
Epoch 95/2000
3/3 [==============================] - 0s 18ms/step - loss: 0.2033 - accuracy: 0.5636 - val_loss: 0.2464 - val_accuracy: 0.4317
Epoch 96/2000
3/3 [==============================] - 0s 18ms/step - loss: 0.2011 - accuracy: 0.5554 - val_loss: 0.2385 - val_accuracy: 0.4153
Epoch 97/2000
3/3 [==============================] - 0s 17ms/step - loss: 0.2003 - accuracy: 0.5431 - val_loss: 0.2454 - val_accuracy: 0.4098
Epoch 98/2000
3/3 [==============================] - 0s 18ms/step - loss: 0.2021 - accuracy: 0.5445 - val_loss: 0.2983 - val_accuracy: 0.3880
Epoch 99/2000
3/3 [==============================] - 0s 18ms/step - loss: 0.2151 - accuracy: 0.5349 - val_loss: 0.2463 - val_accuracy: 0.4262
Epoch 100/2000
3/3 [==============================] - 0s 18ms/step - loss: 0.2113 - accuracy: 0.5417 - val_loss: 0.2609 - val_accuracy: 0.4372
```

Multi-classfication

```
# Loss 값 추출
loss = history.history['loss']
val_loss = history.history['val_loss']

# 그래프 그리기
epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'r', label='train_loss')
plt.plot(epochs, val_loss, 'b', label='val_loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```