Shape 확인

```python
import pandas as pd
import numpy as np
path ='/content/drive/MyDrive/airline_dataset/'
train = pd.read_csv(path+"train.csv")
test= pd.read_csv(path+'test.csv')
sample = (path+'sample_submission.csv')


print(f'train set은 {train.shape[1]} 개의 feature를 가진 {train.shape[0]} 개의 데이터 샘플로 이루어져 있습니다.')
print(f'test set은 {test.shape[1]} 개의 feature를 가진 {test.shape[0]} 개의 데이터 샘플로 이루어져 있습니다.')
test.head()
```

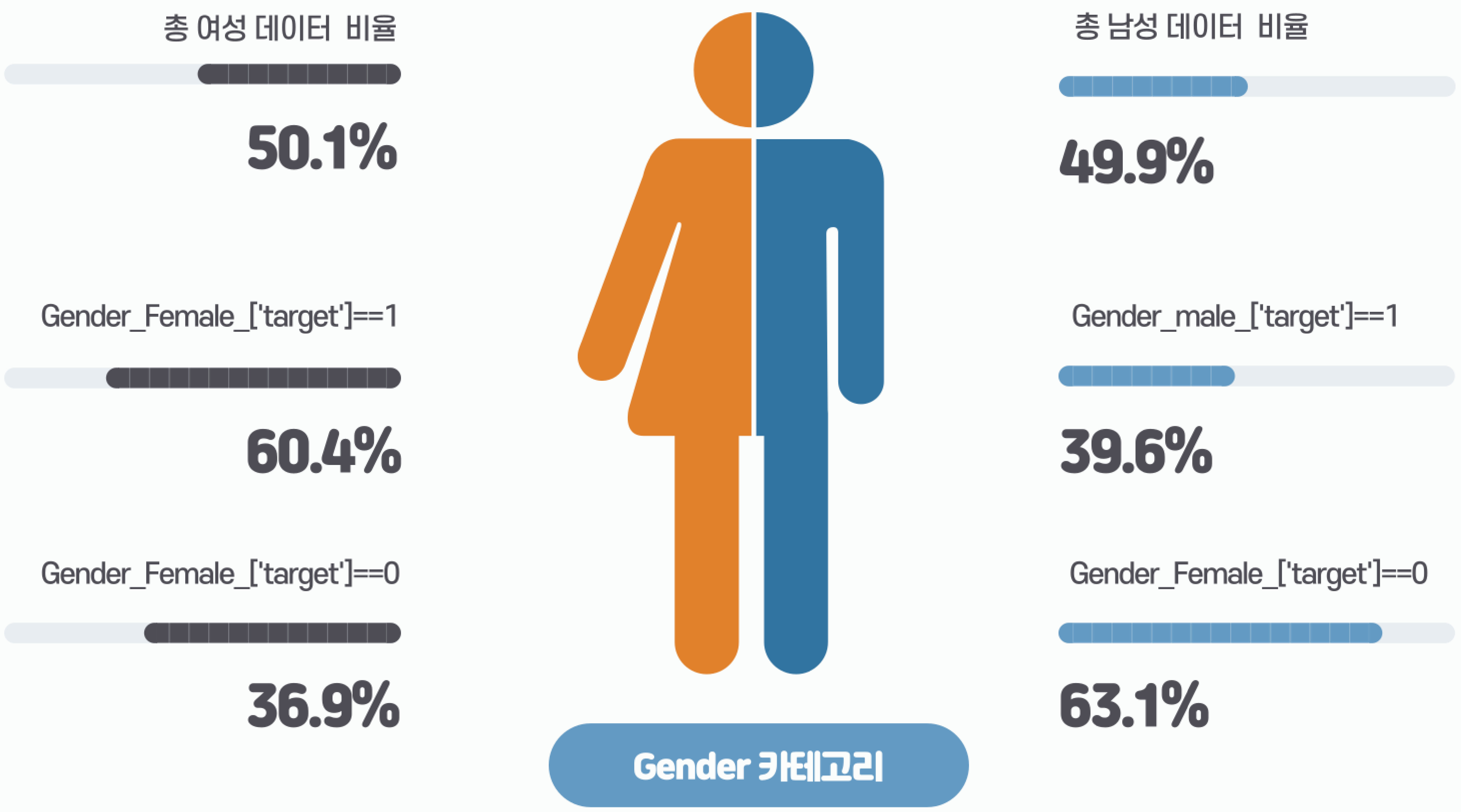
```
train set은 24 개의 feature를 가진 3000 개의 데이터 샘플로 이루어져 있습니다.
test set은 23 개의 feature를 가진 2000 개의 데이터 샘플로 이루어져 있습니다.
```

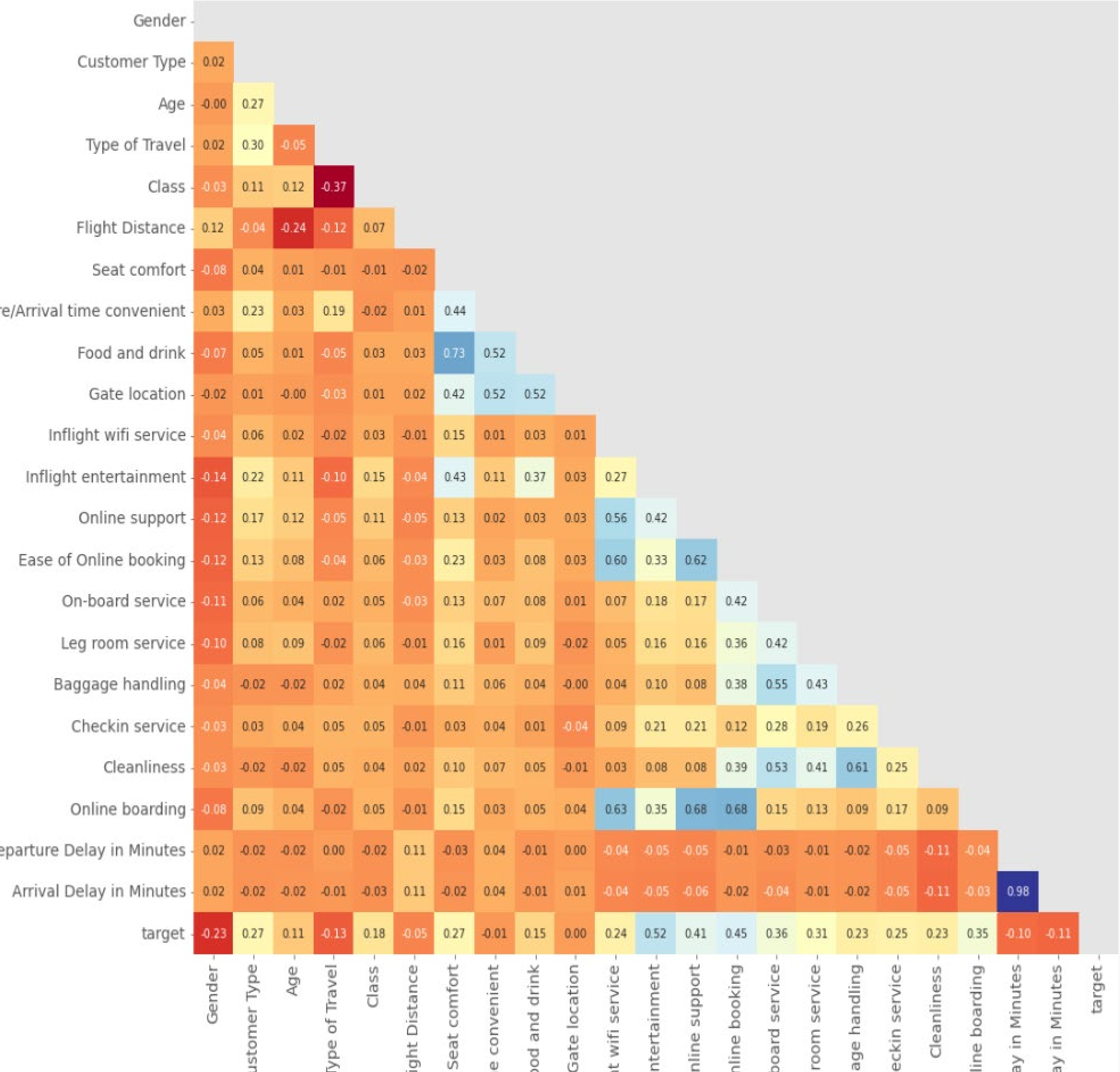
※ train set은 target을 포함하므로 24의 fearture를 가집니다.

결측치 확인

```python
def check_missing_col(dataframe):
    missing_col = []
    for col in dataframe.columns:
        missing_values = sum(dataframe[col].isna())
        is_missing = True if missing_values >= 1 else False
        if is_missing:
            print(f'결측치가 있는 컬럼은: {col} 입니다')
            print(f'해당 컬럼에 총 {missing_values} 개의 결측치가 존재합니다.')
            missing_col.append([col, dataframe[col].dtype])
    if missing_col == []:
        print('결측치가 존재하지 않습니다')
    return missing_col

missing_col = check_missing_col(train)
missing_col = check_missing_col(test)

train.head()

```

```
결측치가 존재하지 않습니다
결측치가 존재하지 않습니다
```


총 여성 데이터 비율
50.1%
Gender_Female_['target']==1
60.4%
Gender_Female_['target']==0
36.9%
총 남성 데이터 비율
49.9%
Gender_male_['target']==1
39.6%
Gender_Female_['target']==0
63.1%
Gender 카테고리



지표 간의 상관관계

Food and Drink
&
Seat comfort

73%

Departure Delay
&
Arrival Delay

98%

데이터 전처리

- 평균 지연 시간 컬럼을 생성

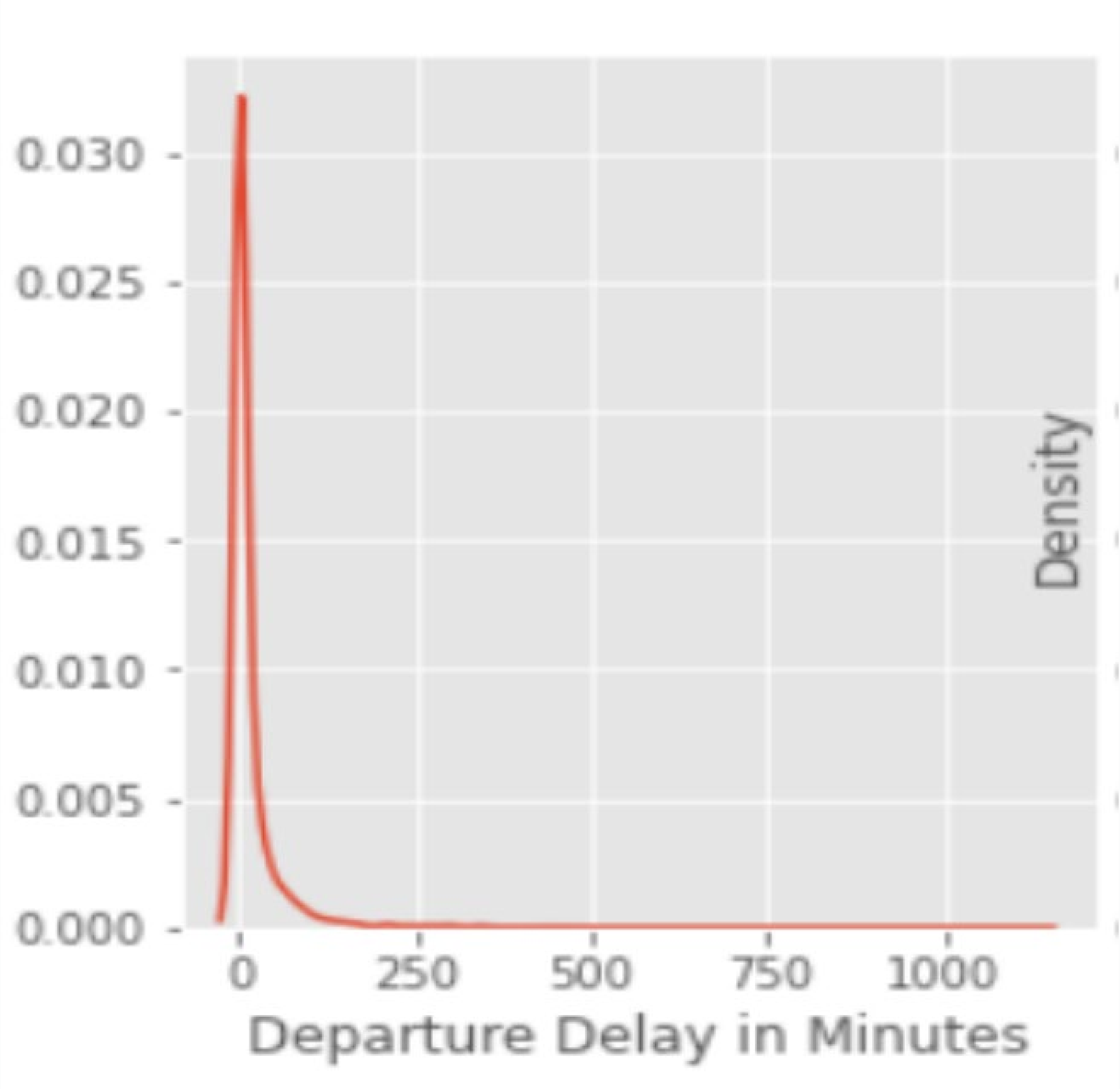
```
train["Mean Delay in Minutes"] = (train["Departure Delay in Minutes"] + train['Arrival Delay in Minutes']) / 2
test["Mean Delay in Minutes"] = (test["Departure Delay in Minutes"] + test['Arrival Delay in Minutes']) / 2

```

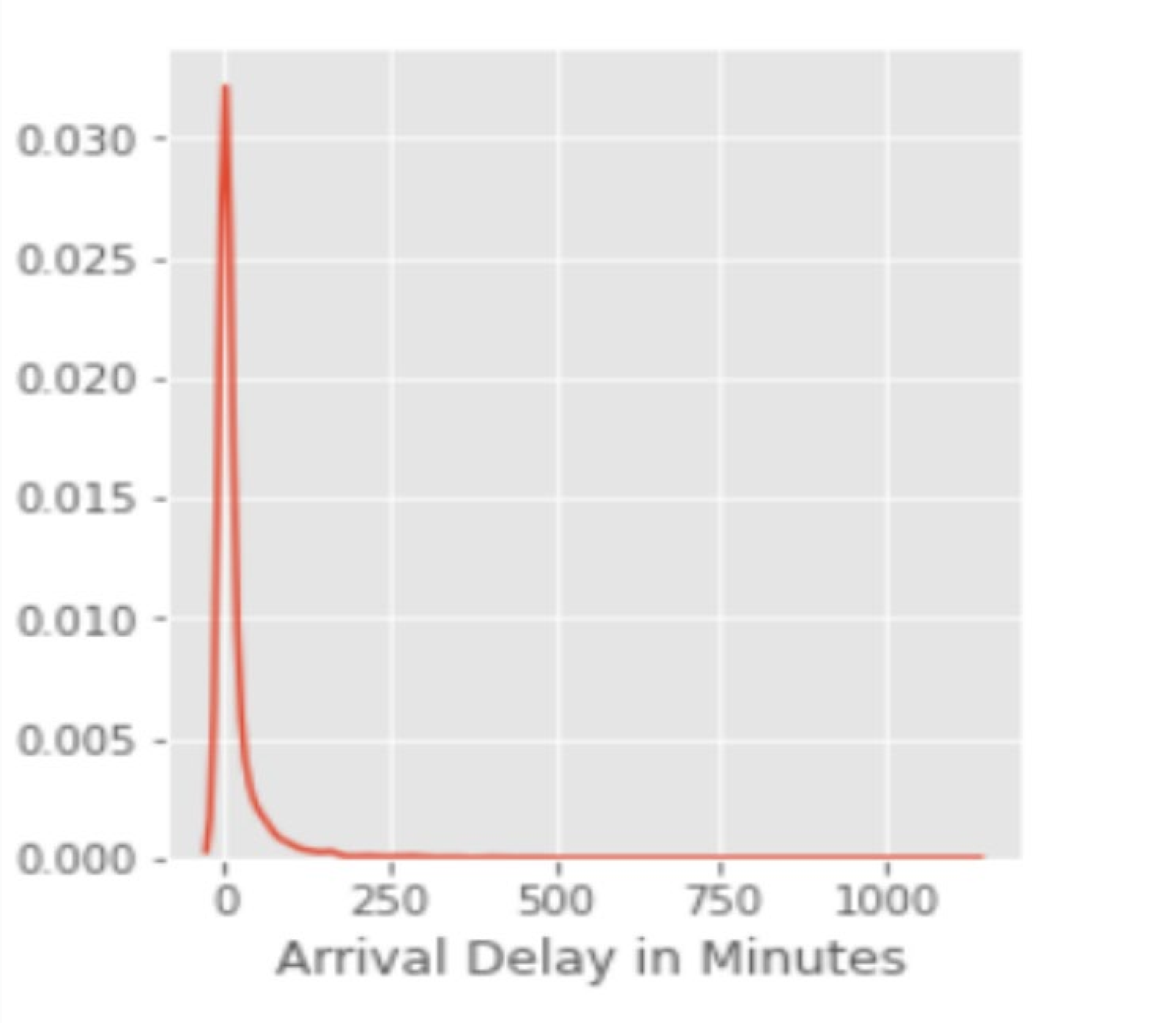
- 불필요한 컬럼은 제거

```
train_x = train.drop(["id", "Departure Delay in Minutes","Arrival Delay in Minutes","Food and drink",'target'],axis=1)
train_y = train.target

test_x = test.drop(["id", "Departure Delay in Minutes","Arrival Delay in Minutes","Food and drink"],axis=1)

```

데이터 전처리_로그(log) 변환

- 큰 값은 엄청 큰 우측 꼬리가 긴 분포형태
  → log 변환을 이용하여 치우쳐진 정도를 줄임





데이터 전처리_로그(log) 변환

- 0은 로그변환이 안되므로 전체 값에 1을 더한 뒤 로그변환 하는 log1p 함수를 사용해야 한다

```
#log 변환 실행
train['Mean Delay in Minutes'] = np.log1p(train['Mean Delay in Minutes'])

# test 데이터에도 변환 실행
test['Mean Delay in Minutes'] = np.log1p(test['Mean Delay in Minutes'])

```

```
Departure Delay in Minutes Scew : 9.190139679910239
Arrival Delay in Minutes Scew : 8.887761727831762
```

```
Departure Delay in Minutes Scew : 0.9302111175258293
Arrival Delay in Minutes Scew : 0.8979015577156512
```

ㄴ모든 skew(왜도) 가 1 이하로 내려간 모습

라벨 인코딩

라벨인코딩을 하기 위한 dictionary map 생성 함수

```
def make_label_map(dataframe):
    label_maps = {}
    for col in dataframe.columns:
        if dataframe[col].dtype=='object':
            label_map = {'unknown':0}
            for i, key in enumerate(dataframe[col].unique()):
                label_map[key] = i+1  ←새로 등장하는 유니크 값들에 대해 1부터 1씩 증가시켜 키값을 부여해줍니다.
            label_maps[col] = label_map
    return label_maps
```

각 범주형 변수에 인코딩 값을 부여하는 함수

```
def label_encoder(dataframe, label_map):
    for col in dataframe.columns:
        if dataframe[col].dtype=='object':
            dataframe[col] = dataframe[col].map(label_map[col])
            dataframe[col] = dataframe[col].fillna(label_map[col]['unknown'])
    return dataframe
```

train 데이터 라벨 인코딩

```
label_map = make_label_map(train)  # train 사용해 label map 생성
train_x = label_encoder(train, label_map)  # train 라벨 인코딩


train_x.head()

```

라벨 인코딩

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort
0	1	Female	disloyal Customer	22	Business travel	Eco	1599	3
1	2	Female	Loyal Customer	37	Business travel	Business	2810	2
2	3	Male	Loyal Customer	46	Business travel	Business	2622	1
3	4	Female	disloyal Customer	24	Business travel	Eco	2348	3
4	5	Female	Loyal Customer	58	Business travel	Business	105	3

- 라벨 인코딩 전

문자열 형식 → 데이터 숫자로 표현

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	De
0	1	1	0.205479	1	1	0.226501	3	
1	1	2	0.410959	1	2	0.403807	2	
2	2	2	0.534247	1	2	0.376281	1	
3	1	1	0.232877	1	1	0.336164	3	
4	1	2	0.698630	1	2	0.007760	3	

- 라벨 인코딩 후

min-max 정규화

- 수치형 데이터의 값을 0~1 사이의 값으로 변환

  -데이터 값의 크기를 줄이고 분산을 줄여 모델이 데이터를 이상하게 해석하는 것을 방지

```python
from sklearn.preprocessing import MinMaxScaler

num_features = ['Age','Flight Distance','Mean Delay in Minutes']

scaler = MinMaxScaler()
train_x[num_features] = scaler.fit_transform(train_x[num_features])
train_x.head()
```

↓

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	...	On-board service	Leg room service	Baggage handling	Checkin service	Cleanliness	Onli boardi
0	1	1	1	0.205479	1	1	0.226501	3	0	3	...	5	4	4	4	5	
1	2	1	2	0.410959	1	2	0.403807	2	4	4	...	5	4	2	1	5	
2	3	2	2	0.534247	1	2	0.376281	1	1	1	...	4	4	4	5	4	
3	4	1	1	0.232877	1	1	0.336164	3	3	3	...	2	4	5	3	4	
4	5	1	2	0.698630	1	2	0.007760	3	3	3	...	4	4	4	4	4	

5 rows × 25 columns

Batch_size , epoch

- batch_size = 한 번 연산할 때 들어가는 데이터의 크기
- epoch = 학습 횟수

```
#Define training hyperprameters.
batch_size = 50
num_epochs = 1000

#Calculate some other hyperparameters based on data.
batch_no = len(train_x) // batch_size  #batches
cols=train_x.shape[1] #Number of columns in input matrix
n_output=1

```

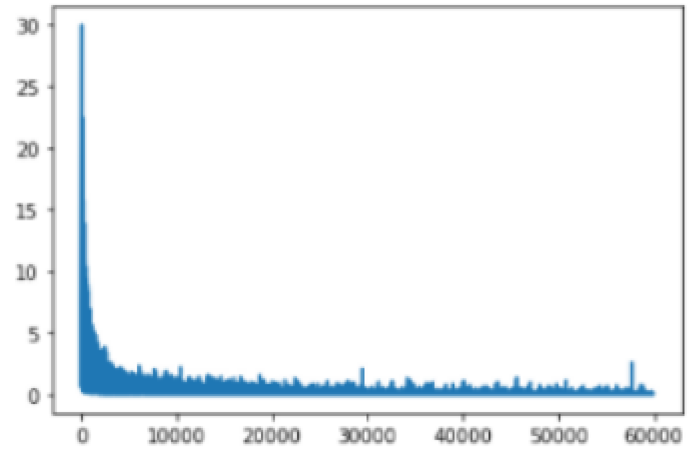
Net 설계

```python
##########
import torch
import torch.nn as nn
#Create the model
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assume that we are on a CUDA machine, then this should print a CUDA device:
print("Executing the model on :",device)
class Net(torch.nn.Module):
    def __init__(self,cols, n_output):
        super().__init__()
        self.linear1=nn.Linear(cols,100)
        self.linear2=nn.Linear(100,50)
        self.linear3=nn.Linear(50,30)
        self.linear4=nn.Linear(30,10)
        self.linear5=nn.Linear(10, n_output)

        #self.dropout = nn.Dropout(p=0.2)
        self.batchnorm1 = nn.BatchNorm1d(100)
        self.batchnorm2 = nn.BatchNorm1d(50)
        self.batchnorm3 = nn.BatchNorm1d(30)
        self.batchnorm4 = nn.BatchNorm1d(10)

        self.relu=nn.ReLU()

        self.sigmoid=nn.Sigmoid()

    def forward(self,X):
        out=self.linear1(X)
        out=self.batchnorm1(out)
        out=self.relu(out)

        out=self.linear2(out)
        out=self.batchnorm2(out)
        out=self.relu(out)

        out=self.linear3(out)
        out=self.batchnorm3(out)
        out=self.relu(out)

        out=self.linear4(out)
        out=self.batchnorm4(out)
        out=self.relu(out)


        out=self.linear5(out)
        #out=self.dropout(out)
        out=self.sigmoid(out)
        return out

model = Net(cols, n_output)
```

학습

```python
import torch.nn.functional as F
from sklearn.utils import shuffle
from torch.autograd import Variable
from matplotlib import pyplot as plt
running_loss = 0.0
losses=[]
for epoch in range(num_epochs):
    #Shuffle just mixes up the dataset between epocs
    X_train, y_train = shuffle(train_x, train_y)
    # Mini batch learning
    for i in range(batch_no):
        start = i * batch_size
        end = start + batch_size
        inputs = Variable(torch.FloatTensor(X_train[start:end]))
        labels = Variable(torch.FloatTensor(y_train[start:end]))
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        #print("outputs",outputs)
        #print("outputs",outputs,outputs.shape,"labels",labels, labels.shape)
        loss = criterion(outputs, torch.unsqueeze(labels,dim=1))
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        losses.append(running_loss)

    print('Epoch {}'.format(epoch+1), "loss: ",running_loss)
    running_loss = 0.0

plt.plot(losses)
plt.savefig('loss_1.jpg')
plt.show()
```

```
Epoch 971 loss:  0.18322661785441596
Epoch 972 loss:  0.10760269820821122
Epoch 973 loss:  0.1769741404123124
Epoch 974 loss:  0.26035080274050415
Epoch 975 loss:  0.24881812533294578
Epoch 976 loss:  0.17376035023607983
Epoch 977 loss:  0.7506302666920419
Epoch 978 loss:  0.5750932950868446
Epoch 979 loss:  0.49313112707568507
Epoch 980 loss:  0.41810559765417565
Epoch 981 loss:  0.7280912714759324
Epoch 982 loss:  0.3019555547089112
Epoch 983 loss:  0.40656463149935007
Epoch 984 loss:  0.24431196725709015
Epoch 985 loss:  0.19043419955960417
Epoch 986 loss:  0.2741489303598428
Epoch 987 loss:  0.19289281975216
Epoch 988 loss:  0.09748574052900949
Epoch 989 loss:  0.06079437142579991
Epoch 990 loss:  0.18782002945954446
Epoch 991 loss:  0.22536235321331333
Epoch 992 loss:  0.10690255970348517
Epoch 993 loss:  0.09410200575894123
Epoch 994 loss:  0.10763223485992057
Epoch 995 loss:  0.15392030361999787
Epoch 996 loss:  0.26773202342064906
Epoch 997 loss:  0.15198937654986366
Epoch 998 loss:  0.07009553032912663
Epoch 999 loss:  0.08804050797061791
Epoch 1000 loss:  0.04363294974427845
```



결과 및 결론

(Conclusion)

92%의 정확도를 보임