Spaceship Titanic

In [60]:

```python
import pandas as pd
import warnings
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import time
```

In [61]:

```python
X_train=pd.read_csv('./spaceship-titanic/train.csv')
X_test=pd.read_csv('./spaceship-titanic/test.csv')
# submission=pd.read_csv('./spaceship-titanic/sample_submission.csv')
```

- 데이터확인
- 데이터전처리
- ML모델링
- 최적화 및 성능평가

In [62]:

```python
X_train.shape, X_test.shape
```

Out[62]:

```
((8693, 14), (4277, 13))
```

In [63]:

```python
X_train.head(5)
```

Out[63]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Nam
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	Mahar Ofraccul
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	Juann Vine
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	Altar Suser
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	Solar Suser
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	Will Santantine

In [64]:

```python
X_test.head(5)
```

Out[64]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name
0	0013_01	Earth	True	G/3/S	TRAPPIST-1e	27.0	False	0.0	0.0	0.0	0.0	0.0	Nelly Carsoning
1	0018_01	Earth	False	F/4/S	TRAPPIST-1e	19.0	False	0.0	9.0	0.0	2823.0	0.0	Lerome Peckers
2	0019_01	Europa	True	C/0/S	55 Cancri e	31.0	False	0.0	0.0	0.0	0.0	0.0	Sabih Unhearfus
3	0021_01	Europa	False	C/1/S	TRAPPIST-1e	38.0	False	0.0	6652.0	0.0	181.0	585.0	Meratz Caltilter
4	0023_01	Earth	False	F/5/S	TRAPPIST-1e	20.0	False	10.0	0.0	635.0	0.0	0.0	Brence Harperez

- train.csv - Personal records for about two-thirds (~8700) of the passengers, to be used as training data.
- PassengerId - A unique Id for each passenger. Each Id takes the form gggg_pp where gggg indicates a group the passenger is travelling with and pp is their number within the group. People in a group are often family members, but not always.
- HomePlanet - The planet the passenger departed from, typically their planet of permanent residence.
- CryoSleep - Indicates whether the passenger elected to be put into suspended animation for the duration of the voyage. Passengers in cryosleep are confined to their cabins.
- Cabin - The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for Port or S for Starboard.
- Destination - The planet the passenger will be debarking to.
- Age - The age of the passenger.
- VIP - Whether the passenger has paid for special VIP service during the voyage.
- RoomService, FoodCourt, ShoppingMall, Spa, VRDeck - Amount the passenger has billed at each of the Spaceship Titanic's many luxury amenities.
- Name - The first and last names of the passenger.

- Transported - Whether the passenger was transported to another dimension. This is the target, the column you are trying to predict.

- PassengerId - Id for each passenger in the test set.

- Transported - The target. For each passenger, predict either True or False.

- PassengerId : gggg-pp 형식 gggg는 여행객 그룹 pp는 그 그룹 안 숫자, 종종 가족 아닐경우도 존재
- HomePlanet : 고향
- CryoSleep : 동면 시키기 그들의 캐빈 안에 갇힘
- Cabin : deck/num/side/ side_P side_S 가 존재
- Destination 목적지
- Age 나이
- VIP vip 여부
- RoomService,FoodCourt,ShoppingMall,Spa,VRDeck 각 구역 사용금액 총합
- Name 이름
- Transported 전송

In [65]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   8693 non-null   object 
 1   HomePlanet    8492 non-null   object 
 2   CryoSleep     8476 non-null   object 
 3   Cabin         8494 non-null   object 
 4   Destination   8511 non-null   object 
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object 
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object 
 13  Transported   8693 non-null   bool   
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

수치형 특성 탐색

In [66]:
```
X_train.describe()
```

Out[66]:

	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
count	8514.000000	8512.000000	8510.000000	8485.000000	8510.000000	8505.000000
mean	28.827930	224.687617	458.077203	173.729169	311.138778	304.854791
std	14.489021	666.717663	1611.489240	604.696458	1136.705535	1145.717189
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	27.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	38.000000	47.000000	76.000000	27.000000	59.000000	46.000000
max	79.000000	14327.000000	29813.000000	23492.000000	22408.000000	24133.000000

범주형 특성 탐색

In [67]:
```
X_train.describe(include=('object','bool'))
```

Out[67]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	VIP	Name	Transported
count	8693	8492	8476	8494	8511	8490	8493	8693
unique	8693	3	2	6560	3	2	8473	2
top	0001_01	Earth	False	G/734/S	TRAPPIST-1e	False	Gollux Reedall	True
freq	1	4602	5439	8	5915	8291	2	4378

In [68]:
```
X_train[X_train.Transported==True]['VIP'].value_counts()
```

Out[68]:
```
False    4198
True       76
Name: VIP, dtype: int64
```

In [69]:
```
X_train.columns
```

Out[69]:
```
Index(['PassengerId', 'HomePlanet', 'CryoSleep', 'Cabin', 'Destination', 'Age',
       'VIP', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck',
       'Name', 'Transported'],
      dtype='object')
```

In [70]:
```
for i,k in enumerate(X_train.dtypes):
    if k=='object':
        print(X_train.columns[i])
```

```
PassengerId
HomePlanet
CryoSleep
Cabin
Destination
VIP
Name
```

In [71]:
```
icd_lb_idx=[]
icd_lb_list=[]
for k,i in enumerate(X_train.columns) :
    #print(i)
    #print(X_train[i].nunique())
    if X_train[i].nunique()<10:
        icd_lb_idx.append(k)
        icd_lb_list.append(i)
    #print('-----------')

# 1,2,4,6, -1 은 원핫 인코딩 or 라벨링 13은 타겟이므로 제거해야함
print(icd_lb_idx)
print(icd_lb_list)
```

```
[1, 2, 4, 6, 13]
['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'Transported']
```

In [72]:
```
X_train.isna().sum()
```

Out[72]:
```
PassengerId       0
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
Transported       0
dtype: int64
```

In [73]:
```
X_train[X_train.CryoSleep.isna()]
```

Out[73]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	N
92	0099_02	Earth	NaN	G/12/P	TRAPPIST-1e	2.0	False	0.0	0.0	0.0	0.0	0.0	Tl Conn
98	0105_01	Earth	NaN	F/21/P	TRAPPIST-1e	27.0	False	0.0	0.0	570.0	2.0	131.0	Cleacl
104	0110_02	Europa	NaN	B/5/P	TRAPPIST-1e	40.0	False	0.0	331.0	0.0	0.0	1687.0	Al Boc
111	0115_01	Mars	NaN	F/24/P	TRAPPIST-1e	26.0	False	0.0	0.0	0.0	0.0	NaN	Rohs
152	0173_01	Earth	NaN	E/11/S	TRAPPIST-1e	58.0	False	0.0	985.0	0.0	5.0	0.0	Gr
...	...	...	...	...	...	...	...	...	...	...	...	...	
8620	9197_01	Europa	NaN	C/308/P	55 Cancri e	44.0	False	0.0	0.0	0.0	0.0	0.0	E F
8651	9227_05	Earth	NaN	G/1498/P	TRAPPIST-1e	8.0	False	0.0	0.0	0.0	0.0	0.0	Hingle
8664	9246_01	Earth	NaN	G/1490/S	TRAPPIST-1e	32.0	False	0.0	0.0	0.0	0.0	0.0	
8675	9259_01	Earth	NaN	F/1893/P	TRAPPIST-1e	44.0	False	1030.0	1015.0	0.0	11.0	NaN	A Gille
8687	9275_03	Europa	NaN	A/97/P	TRAPPIST-1e	30.0	False	0.0	3208.0	0.0	2.0	330.0	Atl Co

217 rows × 14 columns

In [74]:
```
X_train.duplicated().sum()
```

Out[74]:

```
0
```

In [75]:

```
X_test.isna().sum()
```

Out[75]:

```
PassengerId       0
HomePlanet       87
CryoSleep        93
Cabin           100
Destination      92
Age              91
VIP              93
RoomService      82
FoodCourt       106
ShoppingMall     98
Spa             101
VRDeck           80
Name             94
dtype: int64
```

In [76]:

```
X_test.duplicated().sum()
```

Out[76]:

```
0
```

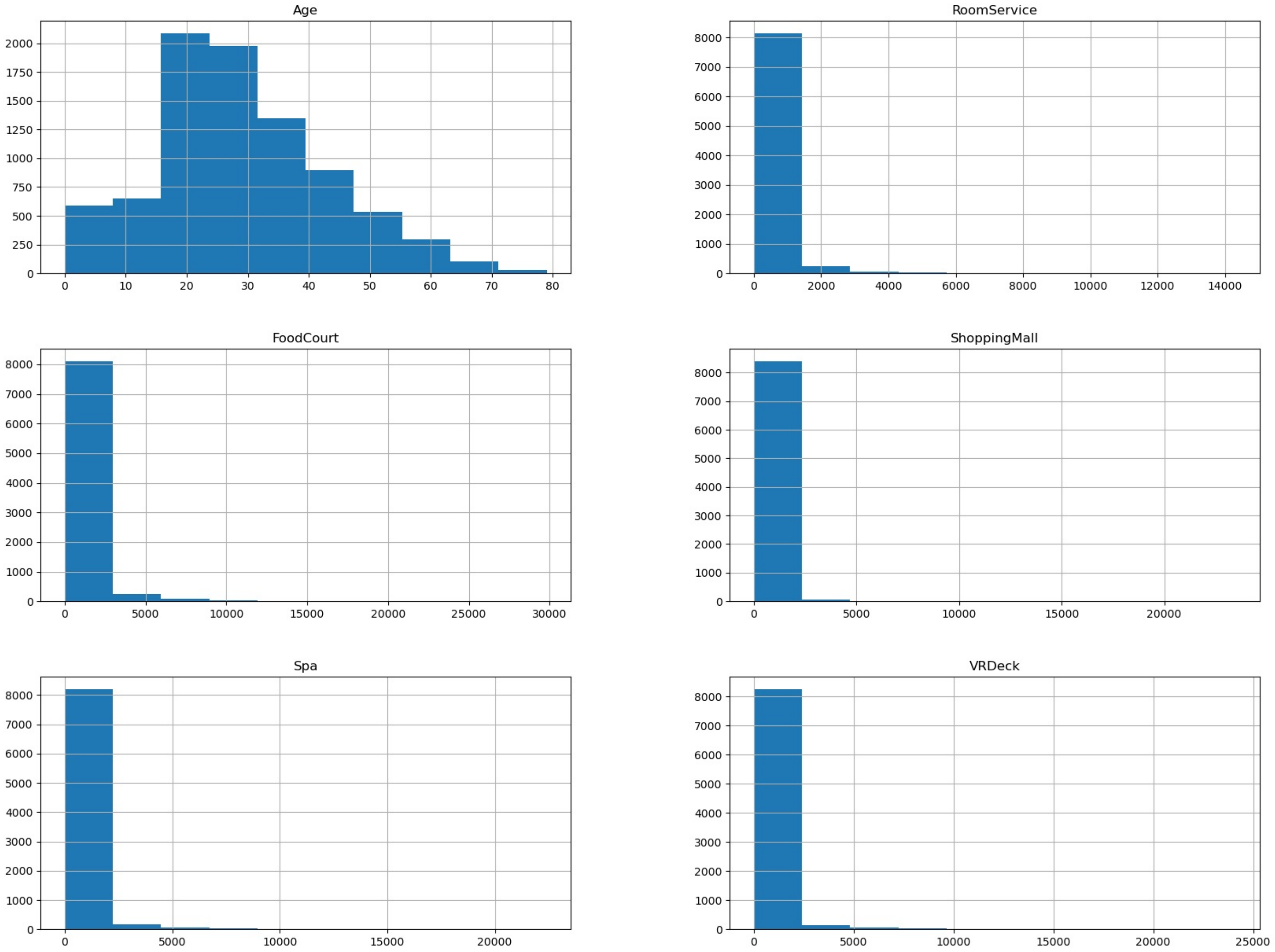
수치형 데이터 히스토그램

In [77]:

```
X_train.hist(bins=10,figsize=(20,15))
```

Out[77]:

```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'RoomService'}>],
       [<AxesSubplot:title={'center':'FoodCourt'}>,
        <AxesSubplot:title={'center':'ShoppingMall'}>],
       [<AxesSubplot:title={'center':'Spa'}>,
        <AxesSubplot:title={'center':'VRDeck'}>]], dtype=object)
```



In [ ]:

범주형 데이터 막대

In [78]:

```
icd_lb_list
```

Out[78]:

```
['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'Transported']
```

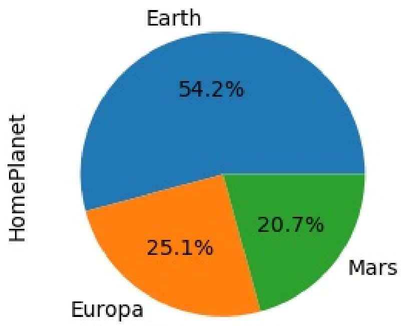
In [79]:

```
X_train['HomePlanet'].value_counts()/X_train[i].count()*100
```

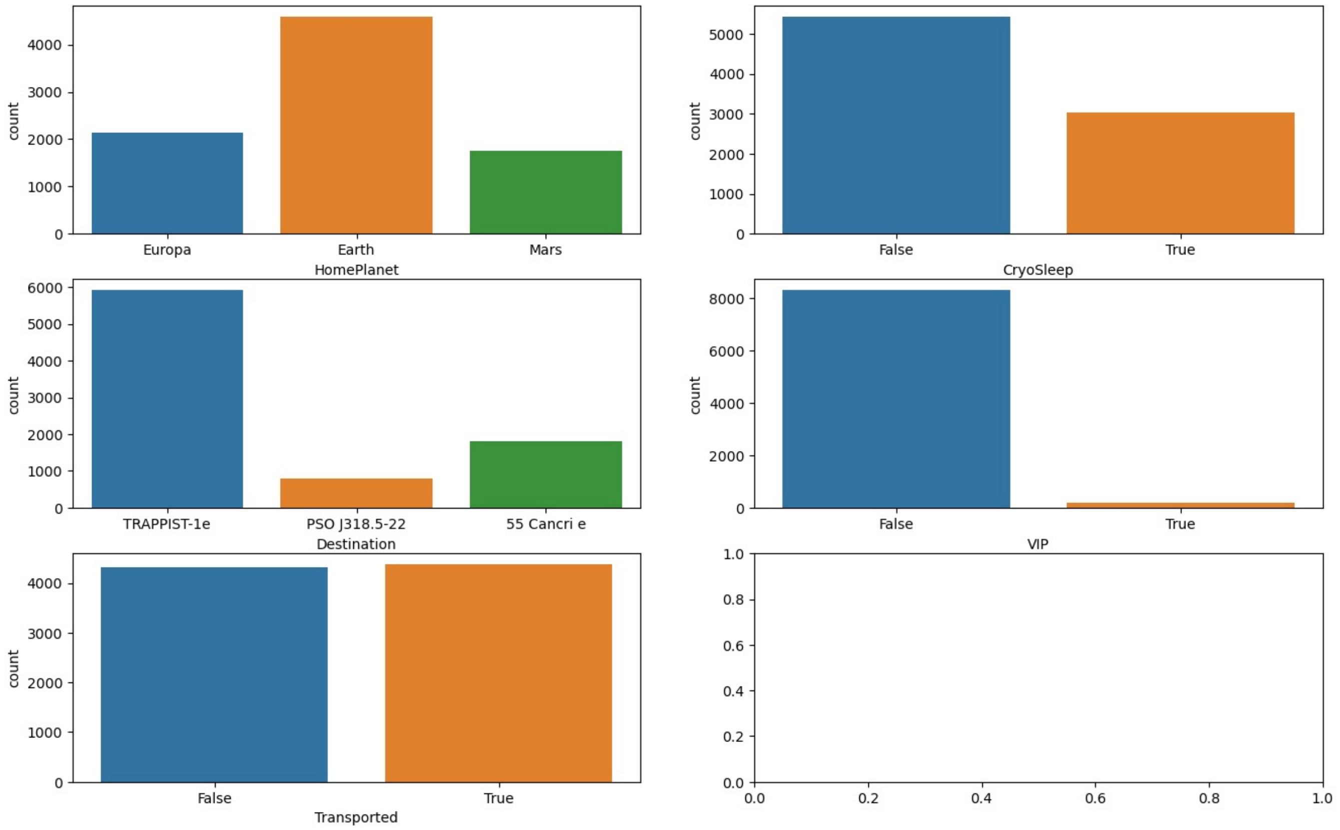
Out[79]:
```
Earth     52.939146
Europa    24.513977
Mars      20.234672
Name: HomePlanet, dtype: float64
```

In [80]:
```
X_train['HomePlanet'].value_counts().plot.pie(autopct="%1.1f%%",figsize=(3,3))
```

Out[80]:
```
<AxesSubplot:ylabel='HomePlanet'>
```



In [81]:
```
fig, axes = plt.subplots(3, 2, figsize=(16, 10))
for i, ax in zip(icd_lb_list, axes.flat):
    sns.countplot(data=X_train,x=i, ax=ax)
plt.show()
```



In [82]:
```
t1=X_train[~X_train['Cabin'].isnull()]
t1.Cabin.isna().sum()
t1
```

Out[82]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	C
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	San
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	No
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	M
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	H
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	H

8494 rows × 14 columns

캐빈값을F/B으로 나누었음

In [83]:

```
t1['front']=t1.Cabin.map(lambda x: x[0])
t1['back']=t1.Cabin.map(lambda x: x[-1])
t1['total']=t1.Cabin.map(lambda x: x[0]+x[-1])
# 빈도수는 F , G , E , B,C,D, A 순
```

In [84]:

```
t1
```

Out[84]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	C
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	San
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	No
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	M
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	H
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	H

8494 rows × 17 columns

In [85]:

```
t1.groupby('total').mean()['Transported'].sort_values(ascending=False)
```

Out[85]:

```
total
BS    0.784038
CS    0.763547
BP    0.674221
GS    0.583788
CP    0.580645
AS    0.546763
FS    0.470501
DS    0.465217
GP    0.448276
AP    0.435897
FP    0.410987
DP    0.403226
ES    0.371365
EP    0.342657
TP    0.250000
TS    0.000000
Name: Transported, dtype: float64
```

In [86]:

```
t1.groupby('front').mean()['Transported'].sort_values(ascending=False)
#생존률은 B,C,G,A,F,D,E,T
```

Out[86]:

```
front
B    0.734275
C    0.680054
G    0.516217
A    0.496094
F    0.439871
D    0.433054
E    0.357306
T    0.200000
Name: Transported, dtype: float64
```

In [87]:

```
t1[t1.front=='B'].groupby('back').mean()['Transported']
```

Out[87]:

```
back
P    0.674221
S    0.784038
Name: Transported, dtype: float64
```

In [88]:

```
t1.groupby('back').mean()['Transported']
```

Out[88]:

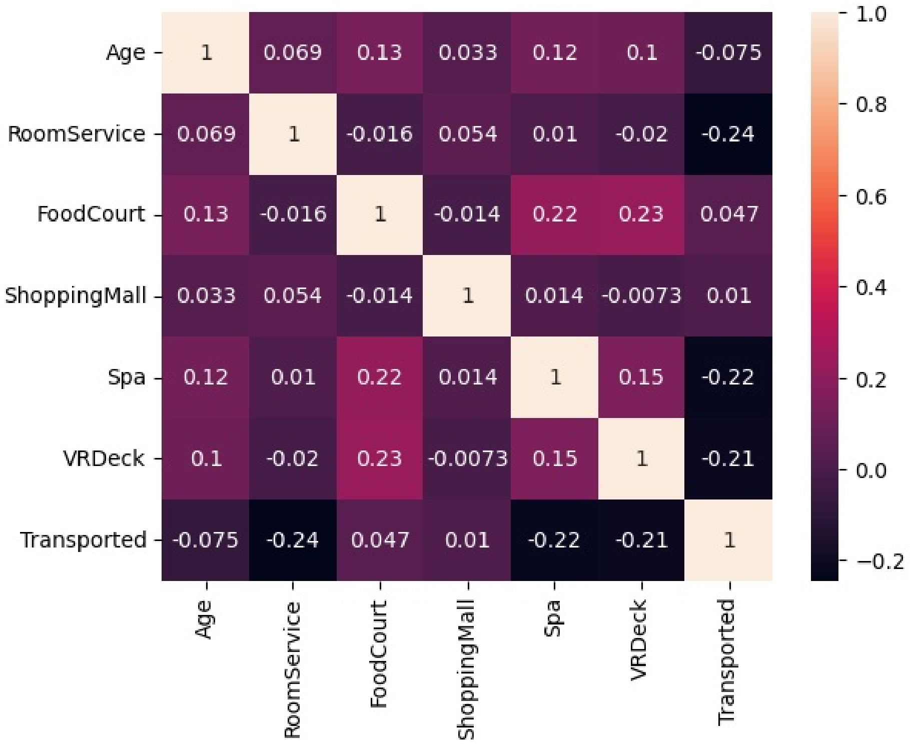
```
back
P    0.451260
S    0.555037
Name: Transported, dtype: float64
```

In [89]:

```
sns.heatmap(X_train.corr(),annot=True)
```

Out[89]:

```
<AxesSubplot:>
```



데이터 전처리

결측치 채우기

In [90]:

```
# X,y분리
```

In [91]:

```
y_train=X_train.iloc[:,-1].copy()
#X_train.drop(['Transported'],axis=1,inplace=True)
```

In [92]:

```
X_train["Group"] = X_train['PassengerId'].apply(lambda x: x.split("_")[0])
X_test["Group"] = X_test['PassengerId'].apply(lambda x: x.split("_")[0])
X_train["GroupSize"] = X_train['PassengerId'].apply(lambda x: int(x.split("_")[1]))
X_test["GroupSize"] = X_test['PassengerId'].apply(lambda x: int(x.split("_")[1]))
```

In [93]:

```
X_train
```

Out[93]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	C
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	San
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	N
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	M
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	H
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	H

8693 rows × 16 columns

In [94]:

```
X_train[~X_train['Cabin'].isnull()].Cabin.map(lambda x: x[0]).value_counts()
X_train[~X_train['Cabin'].isnull()].Cabin.map(lambda x: x[-1]).value_counts()
```

Out[94]:

```
S    4288
P    4206
Name: Cabin, dtype: int64
```

- Cabin - `The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for Port or S for Starboard.

In [95]:

```
X_train['Cabin'].fillna('XXX',inplace=True)
X_test['Cabin'].fillna('XXX',inplace=True)
X_train["Front"] = X_train['Cabin'].apply(lambda x: x[0])
X_train["Back"] = X_train['Cabin'].apply(lambda x: x[-1])
X_test["Front"] = X_test['Cabin'].apply(lambda x: x[0])
X_test["Back"] = X_test['Cabin'].apply(lambda x: x[-1])
X_train
```

Out[95]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	C
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	Sar
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	N
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	M
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	H
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	H

8693 rows × 18 columns

In [96]:

```
X_test.Cabin.isna().sum()  # XXX는 노이즈값
```

Out[96]:

```
0
```

In [97]:

```
a=[]
for i,k in enumerate(X_train.dtypes):
    if k=='object':
        a.append(X_train.columns[i])
a
```

Out[97]:

```
['PassengerId',
 'HomePlanet',
 'CryoSleep',
 'Cabin',
 'Destination',
 'VIP',
 'Name',
 'Group',
 'Front',
 'Back']
```

object 값 나머지는 프리퀀시로 결측값 채우기

In [98]:

```
for i in a:
    X_train[i].fillna(X_train[i].value_counts().idxmax(),inplace=True)
X_train.isna().sum()
```

Out[98]:

```
PassengerId       0
HomePlanet        0
CryoSleep         0
Cabin             0
Destination       0
Age             179
VIP               0
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name              0
Transported       0
Group             0
GroupSize         0
Front             0
Back              0
dtype: int64
```

In [99]:

```
for i in a:
    X_test[i].fillna(X_test[i].value_counts().idxmax(),inplace=True)
X_test.isna().sum()
```

Out[99]:
```
PassengerId       0
HomePlanet        0
CryoSleep         0
Cabin             0
Destination       0
Age              91
VIP               0
RoomService      82
FoodCourt       106
ShoppingMall     98
Spa             101
VRDeck           80
Name              0
Group             0
GroupSize         0
Front             0
Back              0
dtype: int64
```

In [100…
```
a=[]
for i,k in enumerate(X_train.dtypes):
    if k=='float':
        a.append(X_train.columns[i])
a
```

Out[100]:
```
['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
```

In [101…
```
for i in a :
    print(i,X_train[i].median())
```

```
Age 27.0
RoomService 0.0
FoodCourt 0.0
ShoppingMall 0.0
Spa 0.0
VRDeck 0.0
```

In [102…
```
for i in a:
    X_train[i].fillna(X_train[i].mean(),inplace=True)
```

In [103…
```
X_train.isna().sum()
```

Out[103]:
```
PassengerId     0
HomePlanet      0
CryoSleep       0
Cabin           0
Destination     0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
Name            0
Transported     0
Group           0
GroupSize       0
Front           0
Back            0
dtype: int64
```

In [104…
```
for i in a:
    X_test[i].fillna(X_test[i].mean(),inplace=True)
```

In [105…
```
X_test.isna().sum()
```

Out[105]:
```
PassengerId     0
HomePlanet      0
CryoSleep       0
Cabin           0
Destination     0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
Name            0
Group           0
GroupSize       0
Front           0
Back            0
dtype: int64
```

In [106…
```
X_train['Front'].replace('X',X_train['Front'].value_counts().idxmax(),inplace=True)
X_train['Back'].replace('X',X_train['Back'].value_counts().idxmax(),inplace=True)
```

```
X_test['Front'].replace('X',X_test['Front'].value_counts().idxmax(),inplace=True)
X_test['Back'].replace('X',X_test['Back'].value_counts().idxmax(),inplace=True)
```

In [107…

```
bins=[0,9,19,29,39,49,59,69,79]
labels=['0s','10s','20s','30s','40s','50s','60s','70s']
X_train['AgeCat']=pd.cut(X_train.Age,bins=bins,labels=labels)
X_test['AgeCat']=pd.cut(X_test.Age,bins=bins,labels=labels)
```

In [108…

```
X_train['TotalSpent']=X_train.RoomService+X_train.FoodCourt+X_train.ShoppingMall+X_train.Spa+X_train.VRDeck
X_test['TotalSpent']=X_test.RoomService+X_test.FoodCourt+X_test.ShoppingMall+X_test.Spa+X_test.VRDeck

bins=[-1,727,1461,5000,10000,20000,35987]
labels=['a','b','c','d','e','f']
X_train['ToCat']=pd.cut(X_train.TotalSpent,bins=bins,labels=labels)
X_test['ToCat']=pd.cut(X_test.TotalSpent,bins=bins,labels=labels)
```

In [109…

```
X_train
```

Out[109]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	...	VRDeck	N
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	...	0.0	Ma Ofrac
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	...	44.0	Jua V
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	...	49.0	A Su
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	...	193.0	Sc Su
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	...	2.0	V Santant
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	0.0	...	74.0	Gra Noxnu
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	0.0	...	0.0	K Monda
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	...	0.0	Fa Cor
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	0.0	...	3235.0	Ce Honti
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	...	12.0	Pro Honti

8693 rows × 21 columns

In [110…

```
X_train.corr()
```

Out[110]:

	CryoSleep	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	GroupSize	Total
CryoSleep	1.000000	-0.070736	-0.078281	-0.243986	-0.205682	-0.206366	-0.198392	-0.193107	0.460132	0.063291	-0.37
Age	-0.070736	1.000000	0.091574	0.067612	0.127937	0.032655	0.120992	0.099210	-0.074249	-0.131425	0.18
VIP	-0.078281	0.091574	1.000000	0.056595	0.126006	0.018483	0.060573	0.123092	-0.037261	0.000703	0.16
RoomService	-0.243986	0.067612	0.056595	1.000000	-0.015521	0.052962	0.009925	-0.019207	-0.242048	-0.022360	0.23
FoodCourt	-0.205682	0.127937	0.126006	-0.015521	1.000000	-0.013934	0.220587	0.224275	0.046074	0.023136	0.74
ShoppingMall	-0.206366	0.032655	0.018483	0.052962	-0.013934	1.000000	0.013678	-0.007189	0.010019	-0.038388	0.22
Spa	-0.198392	0.120992	0.060573	0.009925	0.220587	0.013678	1.000000	0.147957	-0.218791	0.016637	0.59
VRDeck	-0.193107	0.099210	0.123092	-0.019207	0.224275	-0.007189	0.147957	1.000000	-0.204825	0.009948	0.58
Transported	0.460132	-0.074249	-0.037261	-0.242048	0.046074	0.010019	-0.218791	-0.204825	1.000000	0.066390	-0.19
GroupSize	0.063291	-0.131425	0.000703	-0.022360	0.023136	-0.038388	0.016637	0.009948	0.066390	1.000000	0.01
TotalSpent	-0.376500	0.184509	0.163187	0.234303	0.742208	0.220498	0.592439	0.585835	-0.199445	0.010424	1.00

데이터 탐색

In [111…

```
X_train.columns
```

Out[111]:

```
Index(['PassengerId', 'HomePlanet', 'CryoSleep', 'Cabin', 'Destination', 'Age',
       'VIP', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck',
       'Name', 'Transported', 'Group', 'GroupSize', 'Front', 'Back', 'AgeCat',
       'TotalSpent', 'ToCat'],
      dtype='object')
```

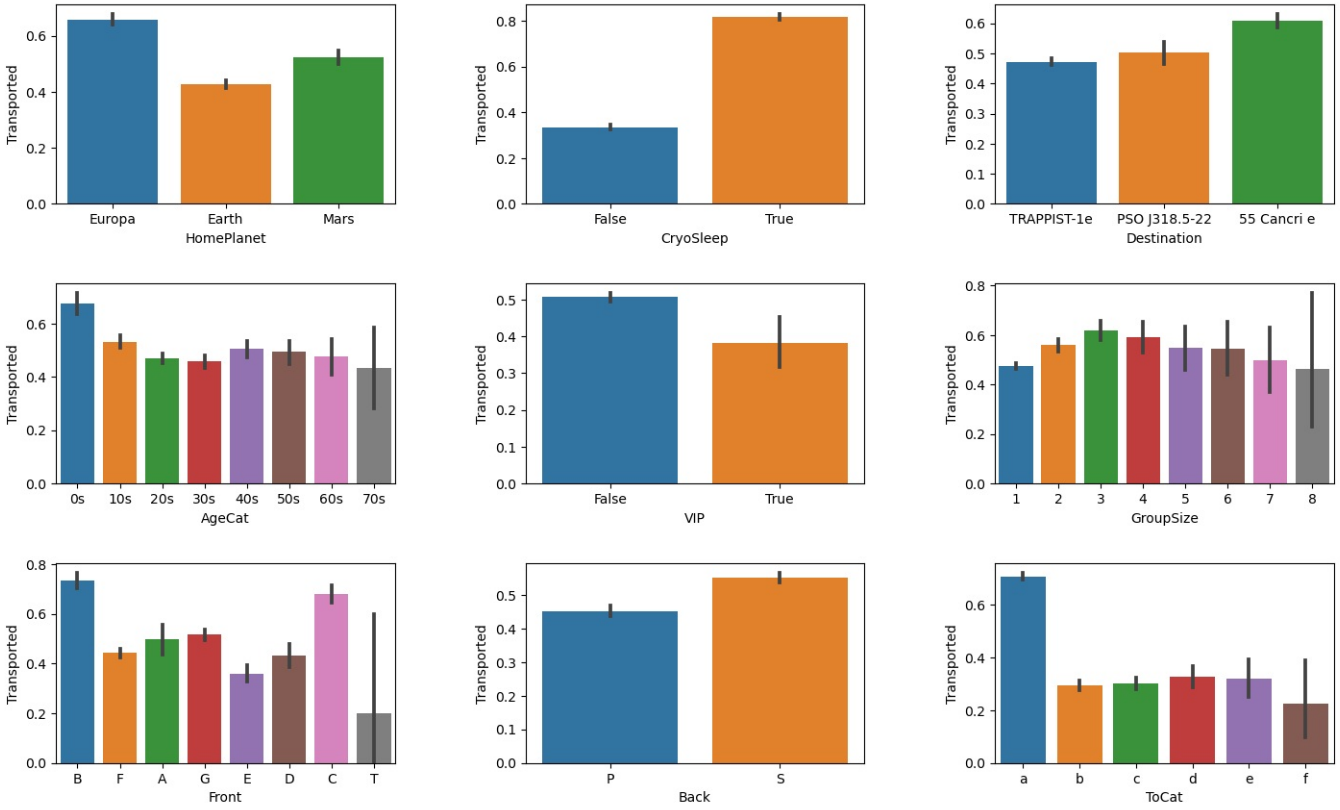
생존률에 따른 그래프

In [112…

```
figure, axes = plt.subplots(nrows=3, ncols=3)
plt.tight_layout()
figure.set_size_inches(15, 9)
sns.barplot(data=X_train,x='HomePlanet',y='Transported',ax=axes[0][0])
sns.barplot(data=X_train,x='CryoSleep',y='Transported',ax=axes[0][1])
sns.barplot(data=X_train,x='Destination',y='Transported',ax=axes[0][2])
sns.barplot(data=X_train,x='AgeCat',y='Transported',ax=axes[1][0])
sns.barplot(data=X_train,x='VIP',y='Transported',ax=axes[1][1])
sns.barplot(data=X_train,x='GroupSize',y='Transported',ax=axes[1][2])
sns.barplot(data=X_train,x='Front',y='Transported',ax=axes[2][0])
sns.barplot(data=X_train,x='Back',y='Transported',ax=axes[2][1])
sns.barplot(data=X_train,x='ToCat',y='Transported',ax=axes[2][2])
```

Out[112]:

```
<AxesSubplot:xlabel='ToCat', ylabel='Transported'>
```

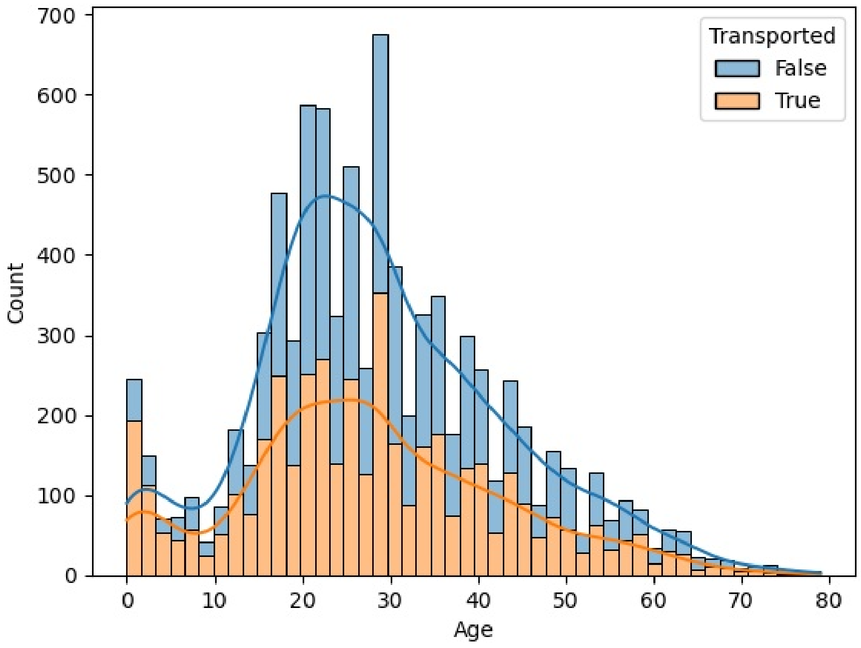


In [113…

```
sns.histplot(data=X_train, x='Age', hue='Transported', multiple='stack', kde=True)
```

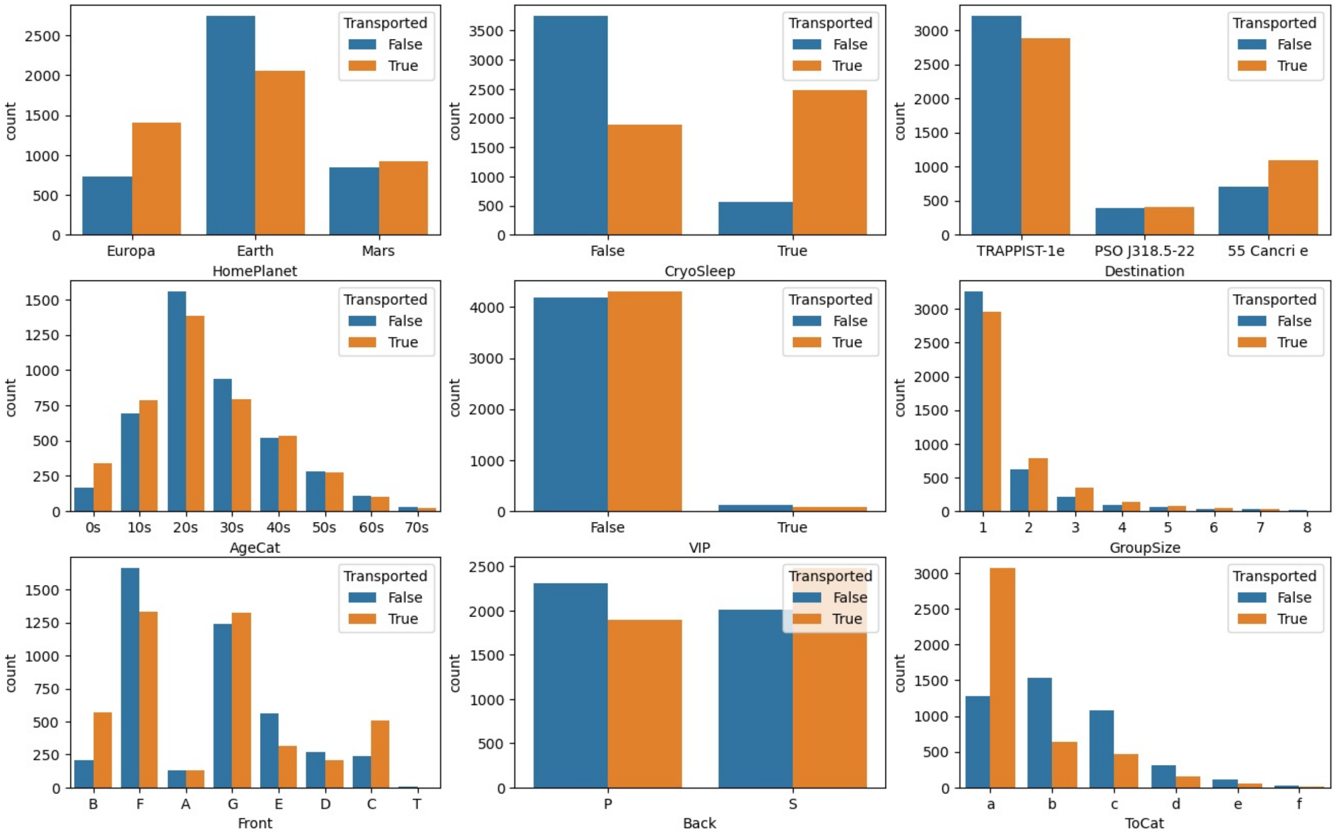
Out[113]:

```
<AxesSubplot:xlabel='Age', ylabel='Count'>
```



In [114…

```
icd_lb_list=['HomePlanet','CryoSleep','Destination','AgeCat','VIP','GroupSize','Front','Back','ToCat']
fig, axes = plt.subplots(3, 3, figsize=(16, 10))
for i, ax in zip(icd_lb_list, axes.flat):
    sns.countplot(data=X_train,x=i, ax=ax,hue='Transported')
plt.show()
```



In [115…

```
X_train.groupby(['ToCat','VIP']).mean()['Transported']
```

Out[115]:

```
ToCat  VIP  
a      False    0.705419
       True     0.896552
b      False    0.295518
       True     0.161290
c      False    0.300824
       True     0.316456
d      False    0.326923
       True     0.342105
e      False    0.325926
       True     0.277778
f      False    0.185185
       True     0.500000
Name: Transported, dtype: float64
```

In [116…

```
X_train.groupby(['Front','VIP']).mean()['Transported']
```

Out[116]:

```
Front  VIP  
A      False    0.520362
       True     0.342857
B      False    0.746269
       True     0.523810
C      False    0.682720
       True     0.634146
D      False    0.449664
       True     0.193548
E      False    0.358885
       True     0.266667
F      False    0.447262
       True     0.171429
G      False    0.516217
T      False    0.200000
Name: Transported, dtype: float64
```

In [117…

```
X_train[X_train['VIP']==True]['Front'].value_counts()
```

Out[117]:

```
B    42
C    41
A    35
F    35
D    31
E    15
Name: Front, dtype: int64
```

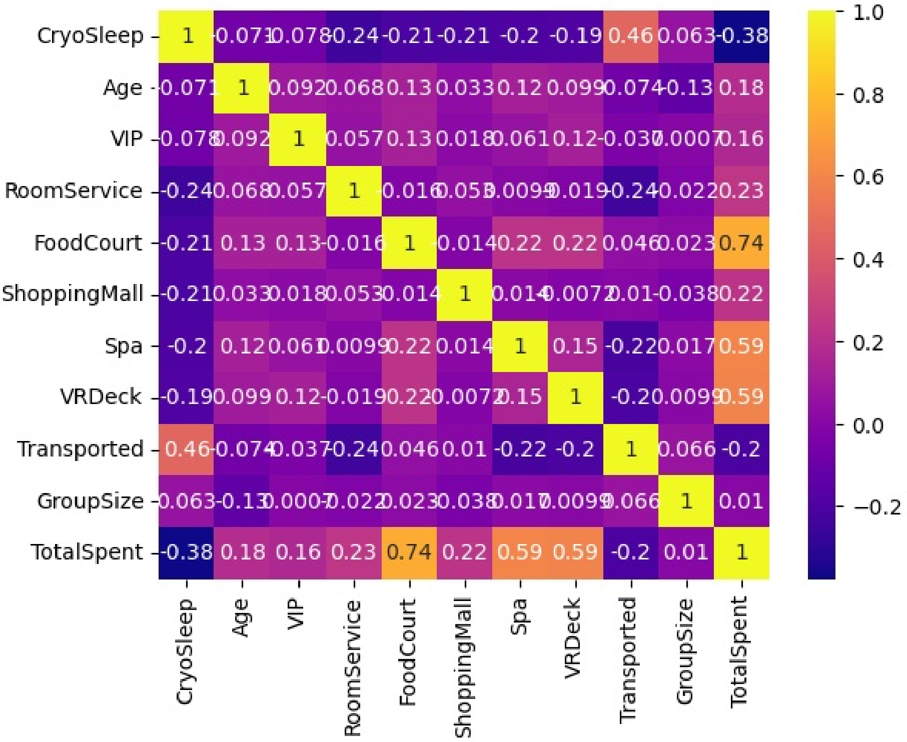
In [118…

```
X_train[X_train['VIP']==False]['Front'].value_counts()
```

```
Out[118]: F    2958
          G    2559
          E     861
          B     737
          C     706
          D     447
          A     221
          T       5
          Name: Front, dtype: int64
```

```
In [119…  sns.heatmap(X_train.corr(),cmap='plasma',annot=True)
```

```
Out[119]: <AxesSubplot:>
```



나만의 변환기 만들기

```
In [120…  #결측기 채울시
          from sklearn.impute import SimpleImputer
          from sklearn.preprocessing import OrdinalEncoder
          from sklearn.preprocessing import LabelEncoder
          from sklearn.preprocessing import OneHotEncoder
          from sklearn.preprocessing import MinMaxScaler
          from sklearn.preprocessing import StandardScaler
          from sklearn.preprocessing import FunctionTransformer
          from sklearn.pipeline import Pipeline
          from sklearn.base import BaseEstimator, TransformerMixin
          from sklearn.compose import ColumnTransformer
```

```
In [121…  X=pd.read_csv('./spaceship-titanic/train.csv')
```

```
In [122…  def prepro(X):
              a=cabin_split(X)
              b=pass_split(X)
              c=pd.concat([X,a,b],axis=1)
              d=fillna_s(c,split_numeric(c))
              d.GroupSize=d.GroupSize.astype('float')
              d['total']=d.RoomService+d.FoodCourt+d.ShoppingMall+d.Spa+d.VRDeck
              droplist=['PassengerId','Cabin','Name','Group']
              d.drop(droplist,axis=1,inplace=True)
              return d
```

```
In [123…  def split_numeric(X):
              numeric_list=[]
              not_numeric_list=[]
              nu_notnu=[]
              for i,v in enumerate(X.dtypes):
                  if v=='int32' or v== 'float':
                      numeric_list.append(X.columns[i])
                  else:
                      not_numeric_list.append(X.columns[i])
              nu_notnu.append(numeric_list)
              nu_notnu.append(not_numeric_list)
              return nu_notnu
```

```
# SimpleImputer를 결측값을 대체(옵션3) 할 수 있음
from sklearn.impute import SimpleImputer

def fillna_s(X,nu_notnu):
    imputer = SimpleImputer(missing_values=np.nan,strategy='median') # 변환기 객체 생성
    npa1=imputer.fit_transform(X[nu_notnu[0]]) # 변환할 준비 : 중앙값을 구함
    time.sleep(1)
    imputer = SimpleImputer(missing_values=np.nan,strategy='most_frequent') # 변환기 객체 생성
    npa2=imputer.fit_transform(X[nu_notnu[1]]) #변환기 객체
    c=pd.concat([pd.DataFrame(npa1,columns=nu_notnu[0]),pd.DataFrame(npa2,columns=nu_notnu[1])],axis=1)
    c=c[X.columns]
    return c

#함수로 캐빈 , 패신저 아이디 쪼개기
def cabin_split(X):
    X.Cabin=X.Cabin.fillna('XXX')
    df1=X.Cabin.apply(lambda x: x[-1])
    df2=X.Cabin.apply(lambda x: x[0])
    df3=pd.concat([df2,df1],axis=1)
    df3.columns=['Front','Back']
    df3.replace('X',np.nan,inplace=True)
    return df3

def pass_split(X):
    df1=X['PassengerId'].apply(lambda x: x.split("_")[0])
    df2=X['PassengerId'].apply(lambda x: int(x.split("_")[1]))
    df3=pd.concat([df1,df2],axis=1)
    df3.columns=['Group','GroupSize']
    return df3

def orencode(X):
    c_list=split_numeric(X)[1]
    ordinal_encoder = OrdinalEncoder()
    X_or=ordinal_encoder.fit_transform(X[c_list])
    columns=ordinal_encoder.get_feature_names_out().copy()
    df=pd.DataFrame(X_or,columns=columns)
    k=X.copy()
    k[df.columns]=df[df.columns]
    return k

def onencode(X) :
    a=pd.DataFrame()
    c=split_numeric(X)[1]
    a=X.drop(c,axis=1)
    o_list=[]
    for i in c :
        onehot_encoder = OneHotEncoder(sparse=False)
        t=onehot_encoder.fit_transform(X[[i]])
        columns=onehot_encoder.get_feature_names_out().copy()
        b=pd.DataFrame(t,columns=columns)
        a=pd.concat([a,b],axis=1)
    #o_list.append(a.columns)
    return a#a[o_list[0]]

def std(X):
    a=split_numeric(X)[0]
    std_scaler = StandardScaler()
    X_or=std_scaler.fit_transform(X[a])
    df= pd.DataFrame(X_or,columns=a)
    k=X.copy()
    k[df.columns]=df[df.columns]
    return k

def minmax(X):
    a=split_numeric(X)[0]
    min_max_scaler = MinMaxScaler(feature_range=(0, 1)) # feature_range=(0, 1)가 기본값, 변경 가능
    X_or = min_max_scaler.fit_transform(X[a])
    df= pd.DataFrame(X_or,columns=a)
    k=X.copy()
    k[df.columns]=df[df.columns]
    return k
```

In [124…

```
cabin_split(X)
```

Out[124]:

	Front	Back
0	B	P
1	F	S
2	A	S
3	A	S
4	F	S
...	...	...
8688	A	P
8689	G	S
8690	G	S
8691	E	S
8692	E	S

8693 rows × 2 columns

In [125…

```
X_train, y_train= X,X.Transported
a=cabin_split(X)
b=pass_split(X)
c=pd.concat([X,a,b],axis=1)

d=fillna_s(c,split_numeric(c))
d.GroupSize=d.GroupSize.astype('float')

d['total']=d.RoomService+d.FoodCourt+d.ShoppingMall+d.Spa+d.VRDeck
droplist=['PassengerId','Cabin','Name','Transported','Group']
d.drop(droplist,axis=1,inplace=True)
```

In [126…

```
onencode(d)
orencode(d)
std(d)
minmax(d)
```

Out[126]:

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Front	Back	Grou
0	Europa	False	TRAPPIST-1e	0.493671	False	0.000000	0.000000	0.000000	0.000000	0.000000	B	P	0.0C
1	Earth	False	TRAPPIST-1e	0.303797	False	0.007608	0.000302	0.001064	0.024500	0.001823	F	S	0.0C
2	Europa	False	TRAPPIST-1e	0.734177	True	0.003001	0.119948	0.000000	0.299670	0.002030	A	S	0.0C
3	Europa	False	TRAPPIST-1e	0.417722	False	0.000000	0.043035	0.015793	0.148563	0.007997	A	S	0.14
4	Earth	False	TRAPPIST-1e	0.202532	False	0.021149	0.002348	0.006428	0.025214	0.000083	F	S	0.0C
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	Europa	False	55 Cancri e	0.518987	True	0.000000	0.228726	0.000000	0.073322	0.003066	A	P	0.0C
8689	Earth	True	PSO J318.5-22	0.227848	False	0.000000	0.000000	0.000000	0.000000	0.000000	G	S	0.0C
8690	Earth	False	TRAPPIST-1e	0.329114	False	0.000000	0.000000	0.079687	0.000045	0.000000	G	S	0.0C
8691	Europa	False	55 Cancri e	0.405063	False	0.000000	0.035186	0.000000	0.015753	0.134049	E	S	0.0C
8692	Europa	False	TRAPPIST-1e	0.556962	False	0.008795	0.157247	0.000000	0.000000	0.000497	E	S	0.14

8693 rows × 14 columns

파이프라인 생성

In [127…

```
#열 추가 파이프라인
```

In [128…

```
class AddColumn(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):
        X['total']=X.RoomService+X.FoodCourt+X.ShoppingMall+X.Spa+X.VRDeck
        return pd.concat([X,pass_split(X),cabin_split(X)],axis=1)
```

In [129…

```
class Impute_na(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):
        X=fillna_s(X,split_numeric(X)).copy()
        X.GroupSize=X.GroupSize.astype('float')
        return X
```

In [130…

```
class Drop_col(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):
        droplist=['PassengerId','Cabin','Name','Group']
        X.drop(droplist,axis=1,inplace=True)
        return X
```

In [131…

```
# 파이프라인 스텝 1 prepro 함수와 같음
```

In [132…

```
pipe1=Pipeline([('add_col',AddColumn()),
                ('impute',Impute_na()),
                ('drop_col',Drop_col())

                        ])
```

In [133…

```
pipe1.fit_transform(X)
```

Out[133]:

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	total	Gro
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	False	0.0	
1	Earth	False	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	True	736.0	
2	Europa	False	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	False	10383.0	
3	Europa	False	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	False	5176.0	
4	Earth	False	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	True	1091.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
8688	Europa	False	55 Cancri e	41.0	True	0.0	6819.0	0.0	1643.0	74.0	False	8536.0	
8689	Earth	True	PSO J318.5-22	18.0	False	0.0	0.0	0.0	0.0	0.0	False	0.0	
8690	Earth	False	TRAPPIST-1e	26.0	False	0.0	0.0	1872.0	1.0	0.0	True	1873.0	
8691	Europa	False	55 Cancri e	32.0	False	0.0	1049.0	0.0	353.0	3235.0	False	4637.0	
8692	Europa	False	TRAPPIST-1e	44.0	False	126.0	4688.0	0.0	0.0	12.0	True	4826.0	

8693 rows × 15 columns

In [134…

```
class SScale(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):

        return std(X)
```

In [135…

```
class Onehotcoder(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):

        return onencode(X)
```

In [136…

```
class Ordinal(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):

        return orencode(X)
```

In [137…

```
class MMScale(BaseEstimator, TransformerMixin):
    def __init__(self):
        # todo
        pass

    def fit(self, X, y=None):
        # todo
        return self

    def transform(self, X):

        return minmax(X)
```

In [138…

```
from sklearn.pipeline import make_pipeline
```

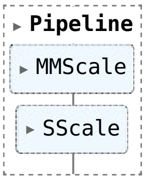
In [ ]:

```

```

In [139…

```
pipe2=Pipeline([('scale',MMScale()),('std',SScale()),


                        ])
pipe2
```

Out[139]:



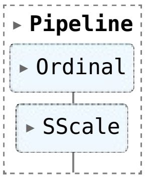
In [140…

```
pipe3= make_pipeline(Ordinal(),
                    SScale())
```

In [141…

```
pipe3
```

Out[141]:



In [142…

```
pipe4= make_pipeline(Onehotcoder(),
                    SScale())
```

In [143…

```
pipe5= make_pipeline(SScale(),Onehotcoder()
                    )
```

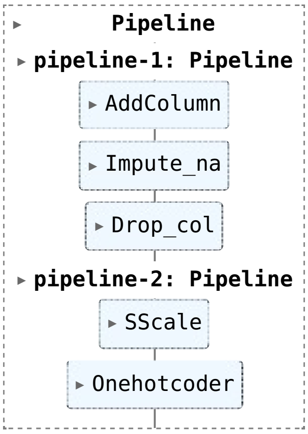
In [144…

```
data_pipe1=make_pipeline(pipe1,pipe3) # 오디 스케일
data_pipe2=make_pipeline(pipe1,pipe4) # 원핫 후 스케일
data_pipe3=make_pipeline(pipe1,pipe5) # 스케일 후 원핫
```

In [145…

```
data_pipe3
```

Out[145]:

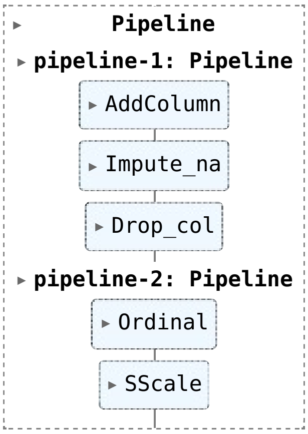


```
▸ Pipeline
▸ pipeline-1: Pipeline
  ▸ AddColumn
  ▸ Impute_na
  ▸ Drop_col
▸ pipeline-2: Pipeline
  ▸ SScale
  ▸ Onehotcoder
```

In [146…

```
data_pipe1
```

Out[146]:

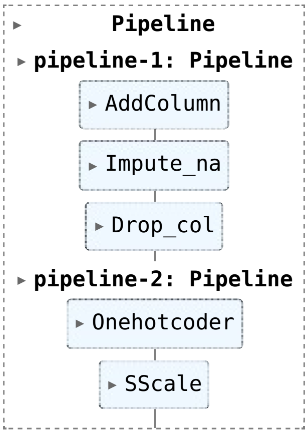


```
▸ Pipeline
▸ pipeline-1: Pipeline
  ▸ AddColumn
  ▸ Impute_na
  ▸ Drop_col
▸ pipeline-2: Pipeline
  ▸ Ordinal
  ▸ SScale
```

In [147…

```
data_pipe2
```

Out[147]:



```
▸ Pipeline
▸ pipeline-1: Pipeline
  ▸ AddColumn
  ▸ Impute_na
  ▸ Drop_col
▸ pipeline-2: Pipeline
  ▸ Onehotcoder
  ▸ SScale
```

6. 모델 선택과 훈련

분류 모델

- 로지스틱
- 나이브 베이즈
- 결정트리
- 서포트 벡터머신SVM
- KNN 알고리즘
- 랜덤 포레스트
- 결정트리
- 확률적 경사하강법

In [148…

```
from sklearn.linear_model import  LogisticRegression,SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split,GridSearchCV, RandomizedSearchCV, StratifiedKFold,KFold, c
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score, f1_score
```

```python
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingClassifier
import random
import catboost as ctb
import graphviz
from sklearn.tree import export_graphviz
```

In [149…

```python
X_o=pd.read_csv('./spaceship-titanic/train.csv')
test=pd.read_csv('./spaceship-titanic/test.csv')
```

In [150…

```python
#자료준비
y_train= X_o.Transported.copy()
X=X_o.drop('Transported',axis=1).copy()
# X_train=prepro(X)
# X_test=prepro(test)
```

In [151…

```python
data_pipe1.fit_transform(X).shape,data_pipe2.fit_transform(X).shape,data_pipe3.fit_transform(X).shape
```

Out[151]:

```
((8693, 14), (8693, 28), (8693, 28))
```

In [152…

```python
# #원핫 후 정규화
# std(onencode(X_train))
# #정규화 후 원핫
# onencode(std(X_train))
```

In [153…

```python
# #ordinal 후 정규화
# std(orencode(X_train))
# #정규화 후 원핫
# orencode(std(X_train))
```

In [154…

```python
lg=LogisticRegression()
kn=KNeighborsClassifier()
svc=SVC(kernel = 'linear')
rf=RandomForestClassifier()
dt=DecisionTreeClassifier()
sgd=SGDClassifier()
GB=GradientBoostingClassifier()
```

In [155…

```python
# X_train=onencode(std(X_train)).copy()
# X_test=onencode(std(X_test)).copy()
X_train=data_pipe2.fit_transform(X)
X_test=data_pipe2.fit_transform(test)
```

In [156…

```python
X_train.shape, y_train.shape,X_test.shape
```

Out[156]:

```
((8693, 28), (8693,), (4277, 28))
```

In [157…

```python
# y_train.astype('int')
# y_train=y_train.astype('int').copy()
# X_train
```

K Fold 검증

In [158…

```python
random.seed=42
models={'lg': 'LogisticRegression()','kn':'KNeighborsClassifier()','svc':'SVC()','rf':'RandomForestClassifier()
        ,'dt':'DecisionTreeClassifier()','sgd':'SGDClassifier()','gaussian':'GaussianNB()','GB':'GradientBoosti

for i,v in models.items():
    accuracy_list = []

    i = v
    t=eval(i)
    kfold = KFold(n_splits=5, shuffle=True)
    for train_index, valid_index in kfold.split(X_train):  #  <--------변수 위치
        # 훈련(학습)

        X_t, y_t = X_train.loc[train_index], y_train.loc[train_index] # 4/5
        X_v, y_v = X_train.loc[valid_index], y_train.loc[valid_index] # 1/5
        t.fit(X_t, y_t)


        # 예측과 평가(정확도)
        pred = t.predict(X_v)
        accuracy = accuracy_score(y_v, pred)
        accuracy_list.append(accuracy)
    print(i)
    print("평균 정확도 : ", np.mean(accuracy_list))
```

```
LogisticRegression()
평균 정확도 :  0.7881058714616485
KNeighborsClassifier()
평균 정확도 :  0.7621086282276693
SVC()
평균 정확도 :  0.7981141364658737
RandomForestClassifier()
평균 정확도 :  0.7940890330871477
DecisionTreeClassifier()
평균 정확도 :  0.7354198112614487
SGDClassifier()
평균 정확도 :  0.7813191052620085
GaussianNB()
평균 정확도 :  0.7112722349458143
GradientBoostingClassifier()
평균 정확도 :  0.7991481553291411
```

StratifiedKFold

In [159…

```
random.seed=42
models={'lg': 'LogisticRegression()','kn':'KNeighborsClassifier()','svc':'SVC()','rf':'RandomForestClassifier()
        ,'dt':'DecisionTreeClassifier()','sgd':'SGDClassifier()','gaussian':'GaussianNB()','GB':'GradientBoosti
for i,v in models.items():
    accuracy_list = []

    i = v
    t=eval(i)

    skfold = StratifiedKFold(n_splits=5, shuffle=True)
    for train_index, valid_index in skfold.split(X_train, y_train):
        # 훈련(학습)
        X_t, y_t = X_train.loc[train_index], y_train.loc[train_index] # 4/5
        X_v, y_v = X_train.loc[valid_index], y_train.loc[valid_index] # 1/5
        t.fit(X_train, y_train)


        # 예측과 평가(정확도)
        pred = t.predict(X_v)
        accuracy = accuracy_score(y_v, pred)
        accuracy_list.append(accuracy)


    print(i)
    print("평균 정확도 : ", np.mean(accuracy_list))
```

```
LogisticRegression()
평균 정확도 :  0.7912110381811432
KNeighborsClassifier()
평균 정확도 :  0.8361897337927502
SVC()
평균 정확도 :  0.8124914057852383
RandomForestClassifier()
평균 정확도 :  0.9654897362411502
DecisionTreeClassifier()
평균 정확도 :  0.9646841464778444
SGDClassifier()
평균 정확도 :  0.7794785702138247
GaussianNB()
평균 정확도 :  0.7327761348499295
GradientBoostingClassifier()
평균 정확도 :  0.8161759830491315
```

cross_val_score

In [160…

```
random.seed=42
models={'lg': 'LogisticRegression()','kn':'KNeighborsClassifier()','svc':'SVC()','rf':'RandomForestClassifier()
        ,'dt':'DecisionTreeClassifier()','sgd':'SGDClassifier()','gaussian':'GaussianNB()','GB':'GradientBoosti
for i,v in models.items():
    accuracy_list = []

    i = v
    t=eval(i)
    scores = cross_val_score(t, X_train, y_train, scoring="accuracy", cv=5) # scoring은 검증 데이터를 어떤 성능 지표

    print(i)
    print("평균 정확도 : ", np.mean(scores))
```

```
LogisticRegression()
평균 정확도 :  0.7868408427525044
KNeighborsClassifier()
평균 정확도 :  0.7588864676933624
SVC()
평균 정확도 :  0.7904082276826687
RandomForestClassifier()
평균 정확도 :  0.7832750459736724
DecisionTreeClassifier()
평균 정확도 :  0.7315084592218984
SGDClassifier()
평균 정확도 :  0.7669395198886175
GaussianNB()
평균 정확도 :  0.7102341133582717
GradientBoostingClassifier()
평균 정확도 :  0.7968503650432011
```

하드보팅

In [161…

```
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(probability=True, random_state=42)
voting_clf = VotingClassifier(
                estimators=[('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)],
                voting='hard'
)

X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
  clf.fit(X_tt, y_tt)
  y_pred = clf.predict(X_td)
  print(clf.__class__.__name__, accuracy_score(y_td, y_pred))
```

```
LogisticRegression 0.791981597108117
RandomForestClassifier 0.7824515280972725
SVC 0.7918172855734472
VotingClassifier 0.799375616168255
```

소프트보팅

In [162…

```
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(probability=True, random_state=42)
voting_clf = VotingClassifier(
                estimators=[('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)],
                voting='soft'
)

X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
  clf.fit(X_tt, y_tt)
  y_pred = clf.predict(X_td)
  print(clf.__class__.__name__, accuracy_score(y_td, y_pred))
```

```
LogisticRegression 0.7972395662175484
RandomForestClassifier 0.7809727242852448
SVC 0.7837660203746303
VotingClassifier 0.8010187315149524
```

train 나눠서 하기 train_size=0.3

In [163…

```
#train 나눠서 하기  train_size=0.3

models={'lg': 'LogisticRegression()','kn':'KNeighborsClassifier()','svc':'SVC()','rf':'RandomForestClassifier()
        ,'dt':'DecisionTreeClassifier()','sgd':'SGDClassifier()','GB':'GradientBoostingClassifier()','gaussian'

estimators=[]
for i,v in models.items():

    istr=i
    i = v
    t=eval(i)
    estimators.append((istr,t))



X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)

for a,b in estimators:
  b.fit(X_tt, y_tt)
  y_pred = b.predict(X_td)
  print(b, accuracy_score(y_td, y_pred))
```

```
LogisticRegression() 0.7875451856720341
KNeighborsClassifier() 0.7548471902727572
SVC() 0.7762076897798226
RandomForestClassifier() 0.77160696680907
DecisionTreeClassifier() 0.7177127834373973
SGDClassifier() 0.7842589549786395
GradientBoostingClassifier() 0.7855734472559974
GaussianNB() 0.6143608281301347
```

In [164…

```
## 소프트보팅


log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(probability=True, random_state=42)
voting_clf = VotingClassifier(
                estimators=[('lr', log_clf), ('rf', rnd_clf), ('svm', svm_clf)],
                voting='soft'
)

X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
  clf.fit(X_tt, y_tt)
  y_pred = clf.predict(X_td)
  print(clf.__class__.__name__, accuracy_score(y_td, y_pred))
```

```
LogisticRegression 0.7942819585934933
RandomForestClassifier 0.7826158396319421
SVC 0.7891883010187315
VotingClassifier 0.8020046007229708
```

In [165…

```
#voting_clf 에 너무 많은 걸 넣으면 돌아가지 않음
```

배깅클래스파이어

In [166…

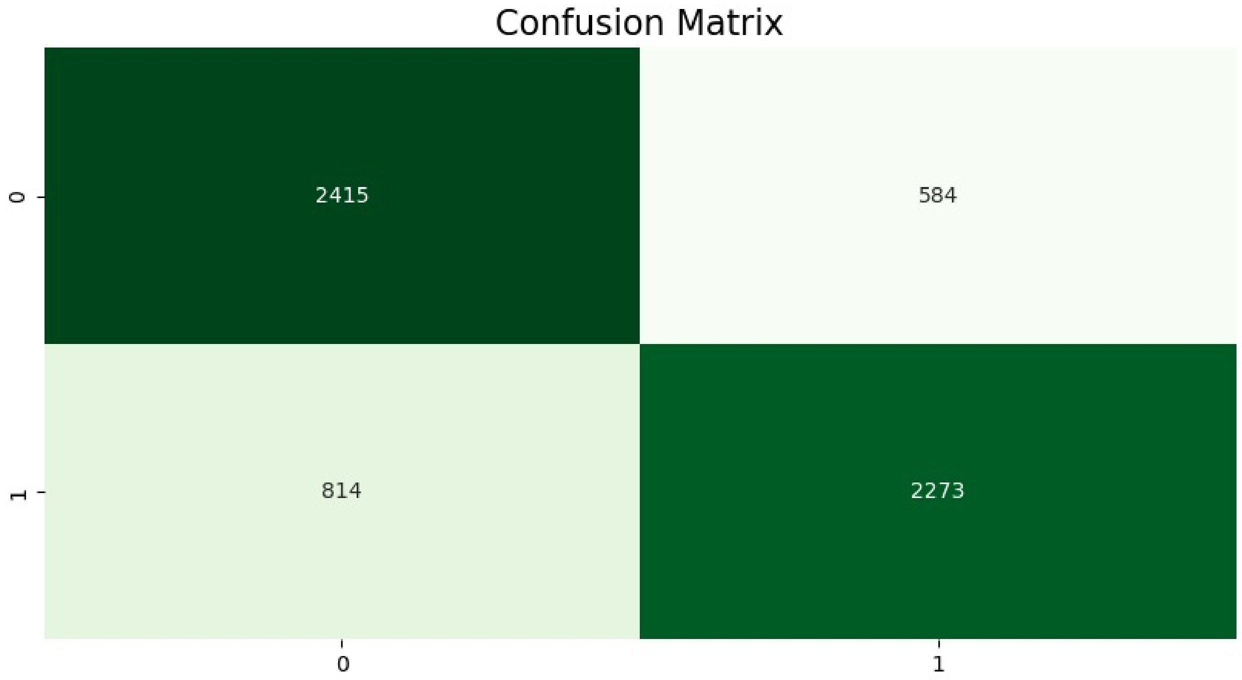
```
X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)


bag_clf = BaggingClassifier(
            DecisionTreeClassifier(max_leaf_nodes=16,random_state=42), n_estimators=500,
            max_samples=100, bootstrap=True, oob_score=True,random_state=42, n_jobs=-1

          )
bag_clf.fit(X_train, y_train)

print("훈련 세트 정확도 : {:.3f}".format(bag_clf.score(X_tt, y_tt)))
print("테스트 세트 정확도 : {:.3f}".format(bag_clf.score(X_td, y_td)))
print("OOB 샘플의 정확도 : {:.3f}".format(bag_clf.oob_score_))
```

```
훈련 세트 정확도 : 0.796
테스트 세트 정확도 : 0.803
OOB 샘플의 정확도 : 0.796
```

- Out-of-Bag 샘플은 부트스트랩 샘플링 과정에서 추출되지 않은 데이터

In [167…

```
#랜덤 포레스트로 정리한 컨퓨전 매트릭스
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42, n_jobs=-1)
X_tt,X_td,y_tt,y_td=train_test_split(X_train,y_train,train_size=0.3,shuffle=True)

rnd_clf.fit(X_tt, y_tt)


y_pred = rnd_clf.predict(X_td)
accuracy_score(y_td, y_pred)
plt.figure(figsize=(10,5))
sns.heatmap(confusion_matrix(y_td,y_pred), annot=True, fmt='d', cmap='Greens',cbar=False).set_title('Confusion
plt.show()

print("훈련 세트 정확도 : {:.3f}".format(rnd_clf.score(X_tt, y_tt)))
print("테스트 세트 정확도 : {:.3f}".format(rnd_clf.score(X_td, y_td)))
```



```
훈련 세트 정확도 : 0.784
테스트 세트 정확도 : 0.770
```

In [168…

```
y_train
```

Out[168]:

```
0       False
1        True
2       False
3       False
4        True
        ...
8688    False
8689    False
8690     True
8691    False
8692     True
Name: Transported, Length: 8693, dtype: bool
```

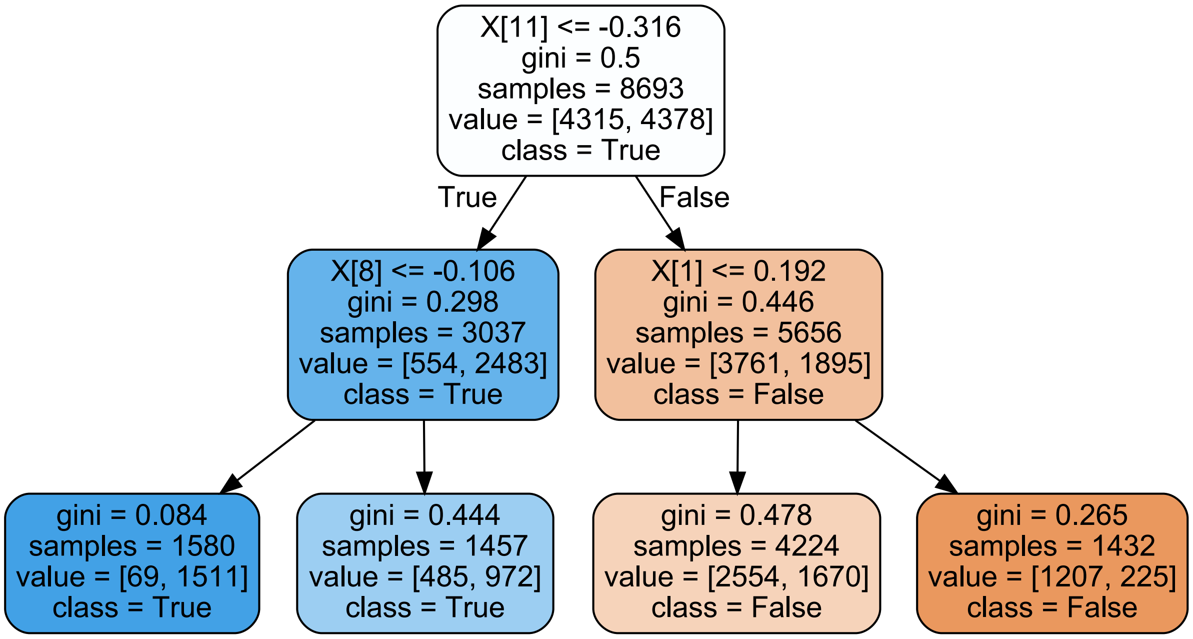
In [169…

```
# 결정트리 한번 뽑아보기
```

In [170…

```
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_train, y_train)
export_graphviz(tree_clf,
                out_file = 'aa.dot',
                #feature_names = iris.feature_names[2:],
                class_names = y_train.astype(str),
                rounded = True,
                filled = True
               )
with open ('aa.dot') as f:
    dot_graph = f.read()

graphviz.Source(dot_graph)
```

Out[170]:



Grid SearchCV

In [171…

```
# GradientBoostingClassifier

# SGDClassifier

# RandomForestClassifier

# LogisticRegression
# 순서로 정리함
```

In [172…

```
parameters = {
#     "learning_rate": [0.01  ,0.1, 0.15, 0.2],
     "max_depth":[3,5,8],
#      'n_estimators' : [50,100,300, 500]

    }

gb_clf = GridSearchCV(GradientBoostingClassifier(random_state=42), parameters, cv=5, n_jobs=-1)

gb_clf.fit(X_train, y_train)
print(gb_clf.score(X_train, y_train))
print(gb_clf.best_params_)
```

```
0.848383757045899
{'max_depth': 5}
```

In [173…

```
parameters = {
    "penalty": ['l1' ,'elasticnet'],
#     "alpha":[0.0001,0.001,0.01,0.1,0.5]

    }

sg_clf = GridSearchCV(SGDClassifier(), parameters, cv=5, n_jobs=-1)

sg_clf.fit(X_train, y_train)
print(sg_clf.score(X_train, y_train))
print(sg_clf.best_params_)
```

```
0.7908662141953295
{'penalty': 'l1'}
```

SGD training을 이용한 SGD Classifier이다.

주요 Parameters:

- loss: 'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_loss', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive' 를 사용할 수 있다.
- penalty: regularization에 사용할 penalty term의 종류. 'l1', 'l2', 'elasticnet'을 사용할 수 있다.
- alpha: regularization term에 곱해줄 가중치
- max_iter: training iteration을 수행할 횟수 (=epoch)

In [174…

```
parameters = {
    'n_estimators':[10,100,500],
#      'max_depth':[6,8,10,12],
#      'min_samples_leaf':[1,2,4,8,12,18],
#      'min_samples_split':[8,16,20]

    }

rf_clf = GridSearchCV(RandomForestClassifier(random_state=42), parameters, cv=5, n_jobs=-1)

rf_clf.fit(X_train, y_train)
print(rf_clf.score(X_train, y_train))
print(rf_clf.best_params_)
```

```
0.9646842286897503
{'n_estimators': 100}
```

In [175…

```
parameters = {
    "penalty": ['l1', 'l2'],
#      'max_iter' :[100,500],
#      'C': [0.1,1,10,100]
    }

lo_rg = GridSearchCV(LogisticRegression(), parameters, cv=5, n_jobs=-1)
# C는 높은 수를 넣을수록 낮은 강도 alpha와 다름
lo_rg.fit(X_train, y_train)
print(lo_rg.score(X_train, y_train))
print(lo_rg.best_params_)
```

```
0.791211319452433
{'penalty': 'l2'}
```

SGD training을 이용한 SGD Classifier이다.

주요 Parameters:

- loss: 'hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_loss', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive' 를 사용할 수 있다.
- penalty: regularization에 사용할 penalty term의 종류. 'l1', 'l2', 'elasticnet'을 사용할 수 있다.
- alpha: regularization term에 곱해줄 가중치
- max_iter: training iteration을 수행할 횟수 (=epoch)

In [176…

```
X_train.shape,X_test.shape
```

Out[176]:

```
((8693, 28), (4277, 28))
```

In [177…

```
#제출 부분
X_train.columns
```

Out[177]:

```
Index(['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck',
       'total', 'GroupSize', 'HomePlanet_Earth', 'HomePlanet_Europa',
       'HomePlanet_Mars', 'CryoSleep_False', 'CryoSleep_True',
       'Destination_55 Cancri e', 'Destination_PSO J318.5-22',
       'Destination_TRAPPIST-1e', 'VIP_False', 'VIP_True', 'Front_A',
       'Front_B', 'Front_C', 'Front_D', 'Front_E', 'Front_F', 'Front_G',
       'Front_T', 'Back_P', 'Back_S'],
      dtype='object')
```

In [178…

```
#X_train=pd.read_csv('./spaceship-titanic/1.csv')

# X_test=pd.read_csv('./spaceship-titanic/2.csv')

# X_train.drop('Transported',axis=1,inplace=True)
```

In [179…

```
#랜덤포레스트 예측
```

In [180…

```
rf_clf=RandomForestClassifier(max_depth=12,n_estimators=100,random_state=42,min_samples_leaf=8,min_samples_spli
rf_clf.fit(X_train, y_train)
y_pred=rf_clf.predict(X_test)
y_train_pred=rf_clf.predict(X_train)


a=precision_score(y_train, y_train_pred)
b=recall_score(y_train, y_train_pred)
c=f1_score(y_train, y_train_pred)
print("정확도 :" ,a, "재현율 :",b,"f1 score :", c)


dddd=pd.read_csv('./spaceship-titanic/test.csv')
sub1 = pd.DataFrame({'Transported':y_pred})
sub2=pd.DataFrame({'PassengerId':dddd.PassengerId})
total_sub=pd.concat([sub2,sub1],axis=1).set_index('PassengerId')

# total_sub.to_csv('submission.csv')
```

```
정확도 : 0.8365165984538426 재현율 : 0.8403380539058931 f1 score : 0.8384229717411121
```

In [181…

```
#그레디언트 부스트
```

In [182…

```
gd_clf=GradientBoostingClassifier(n_estimators=50,max_depth=5,learning_rate=0.1,random_state=42)
gd_clf.fit(X_train, y_train)
y_pred=gd_clf.predict(X_test)
y_train_pred=gd_clf.predict(X_train)


a=precision_score(y_train, y_train_pred)
b=recall_score(y_train, y_train_pred)
c=f1_score(y_train, y_train_pred)
print("정확도 :" ,a, "재현율 :",b,"f1 score :", c)


dddd=pd.read_csv('./spaceship-titanic/test.csv')
sub1 = pd.DataFrame({'Transported':y_pred})
sub2=pd.DataFrame({'PassengerId':dddd.PassengerId})
total_sub=pd.concat([sub2,sub1],axis=1).set_index('PassengerId')

# total_sub.to_csv('submission.csv')
```

```
정확도 : 0.815937635339974 재현율 : 0.860666971219735 f1 score : 0.8377056469542019
```

- gradient boosting regression tree는 여러 개의 decision tree를 묶어 강력한 model을 만드는 ensemble기법입니다.

- random forest와 달리 gradient boosting model은 이전 tree의 오차를 보완하는 방식으로 tree를 만듭니다.
- gradient boosting은 무작위성이 없어 powerful한 pre-pruning이 사용되며
- 1~5 정도 깊이의 tree를 사용하므로 메모리를 적게 사용하고 예측도 빠릅니다.
- gradient boosting은 이런 얕은 트리들을 계속해서 연결해나가는 것입니다.
- gradient boosting은 parmeter설정에 random forest보다 조금 더 민감하지만 잘 조정하면 높은 정확도를 제공합니다.
- gradinet boosting에서 주요 parameter는 이전 트리의 오차를 얼마나 강하게 보정할 것인가를 제어하는
- learning_rate가 있습니다.
- learning_rate를 높이면 보정을 강하게 하기 때문에 복잡한 모델을 만듭니다.
- n_estimator 값을 키우면 ensemble에 트리가 더 많이 추가되어 모델의 복잡도가 커지고 train 세트를 더 정확하게 fitting합니다.

In [183…

```
#캣부스트 패키지 깔아야가능
```

In [184…

```
CBC = ctb.CatBoostClassifier(silent=True,
                             depth=6,
                             iterations=300,
                            )
CBC.fit(X_train, y_train)
y_pred = CBC.predict(X_test)


dddd=pd.read_csv('./spaceship-titanic/test.csv')
sub1 = pd.DataFrame({'Transported':y_pred})
sub2=pd.DataFrame({'PassengerId':dddd.PassengerId})
total_sub=pd.concat([sub2,sub1],axis=1).set_index('PassengerId')

# total_sub.to_csv('submission.csv')
```

In [185…

```
# plt.figure(figsize=(10,5))
# sns.heatmap(confusion_matrix(y_check,ypred), annot=True, fmt='d', cmap='Greens',cbar=False).set_title('Confus
# plt.show()
```

In [186…

```
# print('Confusion Matrix'.center(70,'-'), '\n')
# print(confusion_matrix(y_check,ypred), '\n')
# print('Classification Report'.center(70,'-'), '\n')
# print(classification_report(yvalid,ypred))
# print('Score'.center(70,'-'), '\n')
# print (f'Score of Model NLP is  {round(accuracy_score(y_check, yeprd) * 100,2)}%')
```

In [ ]:

Loading [MathJax]/extensions/Safe.js