

## Kaggle Contest - 1조

2023.05.30

발표자 : 김예원

## 스터디원 소개 및 만남 인증



응용통계학과 19 정서현

전자전기공학부 19 성현우

AI학과 21 김예원

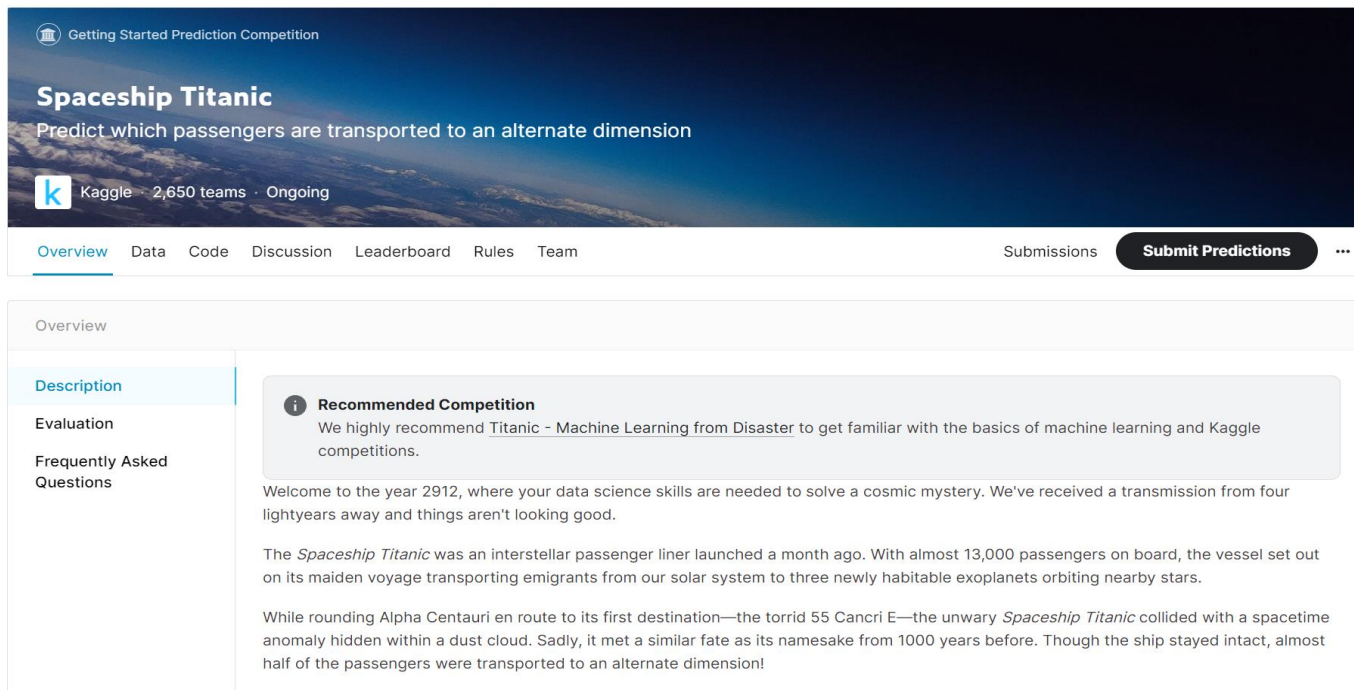
AI학과 21 김건호

에너지시스템공학부 16 김지호

## 목차

1. Kaggle Contest 소개
2. Data 소개
3. Data 전처리
4. Feature Selection
5. 모델 소개

# Kaggle Contest - Spaceship Titanic



The screenshot shows the Kaggle competition page for 'Spaceship Titanic'. The header includes the title 'Spaceship Titanic' and the subtitle 'Predict which passengers are transported to an alternate dimension'. It also shows the Kaggle logo, '2,650 teams', and 'Ongoing' status. The navigation bar includes links for Overview, Data, Code, Discussion, Leaderboard, Rules, Team, Submissions, and a 'Submit Predictions' button. The 'Overview' section is active, showing a 'Description' tab. The description includes a 'Recommended Competition' section that suggests 'Titanic - Machine Learning from Disaster' for beginners. The main text describes the competition scenario: in the year 2912, a spaceship named 'Titanic' is launched with 13,000 passengers. It collides with a spacetime anomaly, resulting in half of the passengers being transported to an alternate dimension.

Getting Started Prediction Competition

## Spaceship Titanic

Predict which passengers are transported to an alternate dimension

Kaggle · 2,650 teams · Ongoing

Overview Data Code Discussion Leaderboard Rules Team Submissions **Submit Predictions** ...

Overview

Description

Evaluation

Frequently Asked Questions

**Recommended Competition**

We highly recommend [Titanic - Machine Learning from Disaster](#) to get familiar with the basics of machine learning and Kaggle competitions.

Welcome to the year 2912, where your data science skills are needed to solve a cosmic mystery. We've received a transmission from four lightyears away and things aren't looking good.

The *Spaceship Titanic* was an interstellar passenger liner launched a month ago. With almost 13,000 passengers on board, the vessel set out on its maiden voyage transporting emigrants from our solar system to three newly habitable exoplanets orbiting nearby stars.

While rounding Alpha Centauri en route to its first destination—the torrid 55 Cancri E—the unwary *Spaceship Titanic* collided with a spacetime anomaly hidden within a dust cloud. Sadly, it met a similar fate as its namesake from 1000 years before. Though the ship stayed intact, almost half of the passengers were transported to an alternate dimension!

타이타닉호가 시공간 이상과 충돌하는 동안  
승객이 다른 차원으로 이송되었는지를 예측

**평가기준 : Accuracy**

## Data

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False

RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
0.0	0.0	0.0	0.0	0.0	Maham Ofracculy	False
109.0	9.0	25.0	549.0	44.0	Juanna Vines	True
43.0	3576.0	0.0	6715.0	49.0	Altark Susent	False
0.0	1283.0	371.0	3329.0	193.0	Solam Susent	False
303.0	70.0	151.0	565.0	2.0	Willy Santantines	True

- **PassengerId** : 승객 ID
  - **HomePlanet** : 출발 행성(거주지)
  - **CryoSleep** : 취침 방식 여부
  - **Cabin** : 객실 종류 및 번호  
(port : 좌현, starboard : 우현)
  - **Destination** : 목적지
  - **Age** : 승객의 나이
  - **VIP** : 승객의 VIP 서비스 유무
  - **RoomService, FoodCourt, ShoppingMall, Spa, VRDeck** : 승객이 해당 서비스에 대해 지불한 금액
  - **Name** : 이름
  - **Transported** : 도착 여부
- 예측해야하는 target

## Data 전처리

```
train.isnull().sum()
```

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

'PassengerId'와 'Transported'(target)를 제외한  
사실상 모든 피처에 결측치가 존재했음.

결측치가 있는 모든 행을 drop하면  
전체 데이터의 24%가 제거됨.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 8693 entries, 0 to 8692
```



```
train.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 6606 entries, 0 to 8692  
Data columns (total 14 columns):
```

약 2주간 결측치를 제거하지 않고  
피처 간 상관관계를 파악해 결측치를 처리하고자 했음.

## Data 전처리

1. 결측치 처리에서의 **Key Point** : PassengerId 앞 번호를 이용해 **group**칼럼을 생성  
-> **group**과 **피쳐간 상관성**을 파악하는 것이었음.
2. Name 피쳐는 미리 삭제함.

PassengerId
0001_01
0002_01
0003_01
0003_02
0004_01

group
1
2
3
3
4

# Data 전처리 - HomePlanet

HomePlanet

CryoSleep

Cabin

Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name

Transported

	Number_missing	Percentage_missing
HomePlanet	201	2.31
CryoSleep	217	2.50

## 201개의 NaN 값 (2.31%)

1. PassengerID는 0000\_00 꼴인데 앞의 4자리가 같으면 일행이다. 즉, 같은 행성 출신이다.
2. Name 피쳐로부터 서로 같은 성씨(Last name)인 경우에는 같은 행성 출신이다.

-> 이 두 가지 사실을 이용하여 결측치 처리



# Data 전처리 - HomePlanet

HomePlanet

CryoSleep

Cabin

Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name

Transported

HomePlanet 컬럼이 결측치인 승객 중, 그룹 내에서 다른 HomePlanet이 적어도 하나 존재하는 그룹의 경우 해당그룹의 존재하는 승객의 고향을 따오는 과정

```
In [13]: # Missing values before
HP_bef=data['HomePlanet'].isna().sum() # HomePlanet이 결측치인 갯수

# Passengers with missing HomePlanet and in a group with known HomePlanet
GHP_index=data[data['HomePlanet'].isna()][(data[data['HomePlanet'].isna()]['Group']).isin(GHP_gb.index)].index #HomePlanet 컬럼이 결측치인 승객의 인덱스

# Fill corresponding missing values
data.loc[GHP_index, 'HomePlanet']=data.iloc[GHP_index,:]['Group'].map(lambda x: GHP_gb.idxmax(axis=1)[x])

# Print number of missing values left
print('#HomePlanet missing values before:',HP_bef)
print('#HomePlanet missing values after:',data['HomePlanet'].isna().sum())

#HomePlanet missing values before: 201
#HomePlanet missing values after: 111
```

결측치가 111개로 줄어듬!

남은 HomePlanet 컬럼이 결측치인 승객 중, 그룹 내에서 Surname이 같을 경우에 해당그룹에 존재하는 승객의 고향을 따오는 과정

```
In [14]: # Joint distribution of Surname and HomePlanet
SHP_gb=data.groupby(['Surname','HomePlanet'])['HomePlanet'].size().unstack().fillna(0)

# Countplot of unique values
plt.figure(figsize=(10,4))
sns.countplot((SHP_gb>0).sum(axis=1))
plt.title('Number of unique planets per surname')
```

Out[14]: Text(0.5, 1.0, 'Number of unique planets per surname')

# Data 전처리 – CryoSleep

HomePlanet **CryoSleep** Cabin Destination Age VIP RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported

## CryoSleep : 217개의 NaN 값 (2.50%)

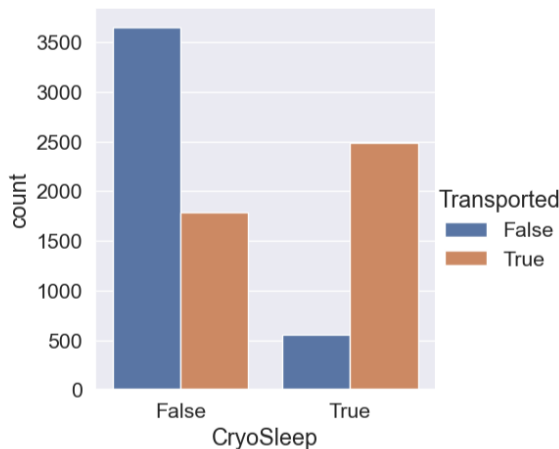
-> 'CryoSleep' True/False로 구분된 논리형(Boolean) 변수

```
In [42]: df['CryoSleep'].describe()
#FALSE 가 5315개 TRUE는 2962개
```

```
Out [42]: count      8476
unique         2
top            False
freq          5439
Name: CryoSleep, dtype: object
```

```
In [43]: sns.catplot(data=df, x='CryoSleep', hue='Transported', kind='count')
```

```
Out [43]: <seaborn.axisgrid.FacetGrid at 0x1f598b92490>
```



- 결측값을 제외한 나머지 값 중 True는 2962개(약 35.8%), False는 5315개(약 64.2%)를 차지함.
- False 값이 약 1.8배정도 더 많이 차지함.
- 만약 결측값을 모두 False로 **보관**할 경우 차지하는 비율은 True(34.9%), False(65.1%)로 1% 미만 범위에서 변함.
- 따라서 1% 미만의 변화는 유의미하지 않다는 가정하에 'CryoSleep'에서 결측된 값은 False로 **보관**함.

# Data 전처리 – Cabin

HomePlanet CryoSleep

Cabin

Destination Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name

Transported

Deck / number / side  
호실 유형 / 호실 번호 / 호실 위치

Cabin

~~B/0/P~~

F/0/S

A/0/S

A/0/S

F/1/S

Cabin\_deck

Cabin\_number

Cabin\_side

B

0.0

P

F

0.0

S

A

0.0

S

A

0.0

S

F

1.0

S

# Data 전처리 – Cabin

HomePlanet CryoSleep

Cabin

Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name

Transported

```
In [16]: g_cs = pd.DataFrame(train_gs.groupby(['group'])['Cabin_side'].value_counts())
g_cs.columns = ['count']
g_cs = g_cs.reset_index()
g_cs
```

```
Out[16]:
```

	group	Cabin_side	count
0	3	S	2
1	6	S	2
2	8	P	3
3	17	P	2
4	20	S	6
...	...	...	...
1407	9252	P	2
1408	9267	S	2
1409	9272	P	2
1410	9275	P	3
1411	9280	S	2

1412 rows × 3 columns

```
In [17]: sum(g_cs.duplicated('group'))
```

```
Out[17]: 0
```

```
sum(g_cd.duplicated('group'))
```

442

```
sum(g_cn.duplicated('group'))
```

441

442/441개가 같은 group이어도  
같은 Cabin\_deck/number 값을 가지지 않았음.

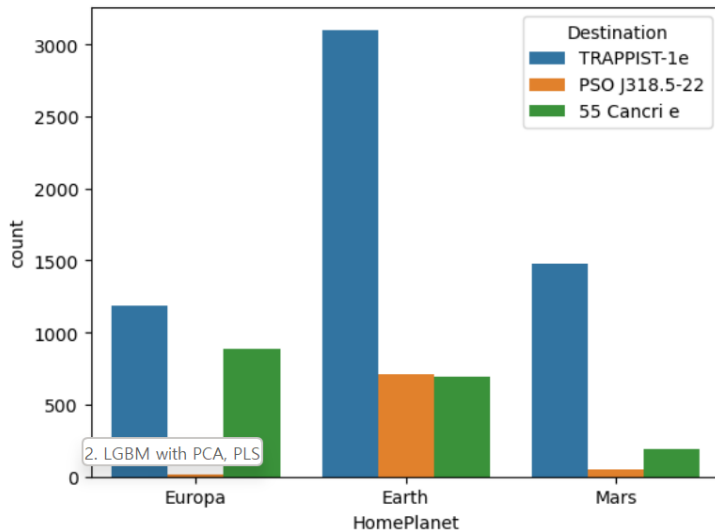
하지만 최빈값 처리보다 오류가 훨씬 적으므로 이 값을 이용.

-> 같은 group이면 모두 같은 Cabin\_side 값을 가짐.

-> Cabin\_side 결측치 199개를 모두 옳은 값으로 채울 수 있다.

# Data 전처리 – Destination

HomePlanet CryoSleep Cabin **Destination** Age VIP RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported



각각의 HomePlanet에서 온 승객들이 Destination 피쳐의 최빈값에 도착한 것을 알 수 있음.

-> **Destination**의 최빈값으로 결측치 채우기.

# Data 전처리 - Age

HomePlanet CryoSleep

Cabin Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name Transported

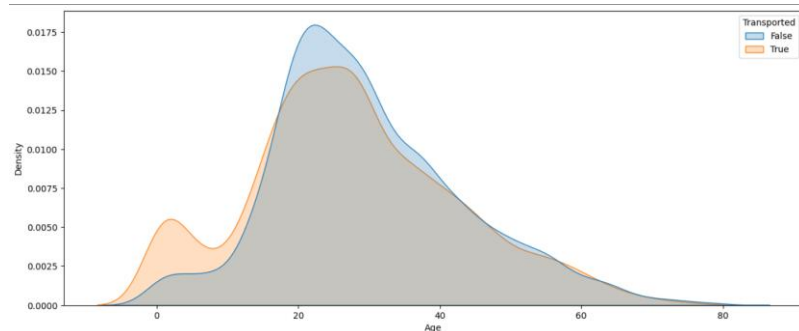
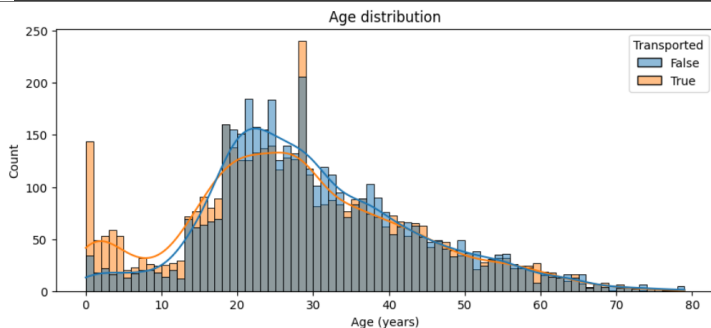
Age : 결측치를 해당 열의 평균값으로 대체

수치형 변수의 경우 평균값으로 대체

```
[ ] df['Age']=df['Age'].fillna(df['Age'].mean())
```

```
[ ] data['Age']=data['Age'].fillna(data['Age'].mean())
```

Age 결측치는 평균값으로 대체



- 20세 미만의 어린이나 아기들이 Transported한 경우가 더 많다.
- 20세에서 40세 사이에서는 Transported 되지 않은 사람이 조금 더 많다.
- 40대 이상에서는 Transported 한 사람과 하지 않은 사람이 비슷하다.

# Data 전처리 - VIP

HomePlanet CryoSleep Cabin Destination Age **VIP** RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported

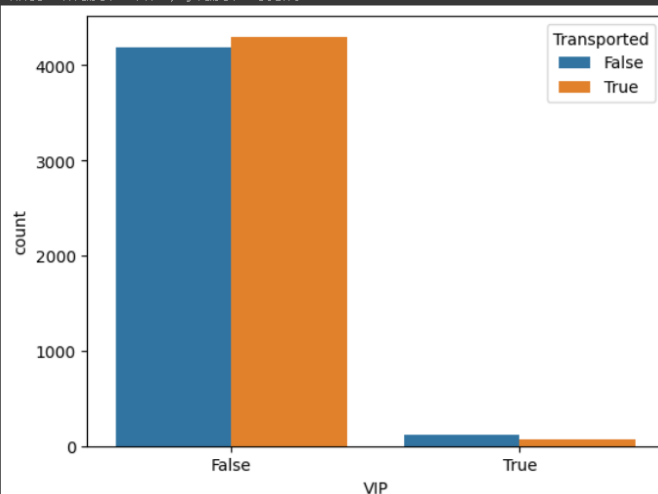
## < 데이터프레임 VIP 결측치를 채우기 >

범주형 변수의 경우 최빈값으로 대체

```
[ ] df['VIP']=df['VIP'].fillna(df['VIP'].mode()[0])  
[ ] data['VIP']=data['VIP'].fillna(data['VIP'].mode()[0])
```

```
sns.countplot(data=df, x='VIP', hue='Transported')
```

<Axes: xlabel='VIP', ylabel='count'>



## Age 결측치는 최빈값으로 대체

VIP의 유무는 결과에 차이를 보여주지 않는다.

- VIP를 신청한 사람 중 제대로 도착을 한 인원의 비중이 약 38퍼센트, 신청하지 않은 사람 중 도착한 인원의 비중이 약 50퍼센트인 것을 확인할 수 있다.
- 그래프로 나타냈을 때 약 1.3배의 차이가 있지만 신청을 하지 않은 사람의 비중이 압도적으로 크므로 크게 영향이 있다고 보기는 어렵다.

# Data 전처리 - VIP

HomePlanet CryoSleep

Cabin

Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

Spa

VRDeck

Name

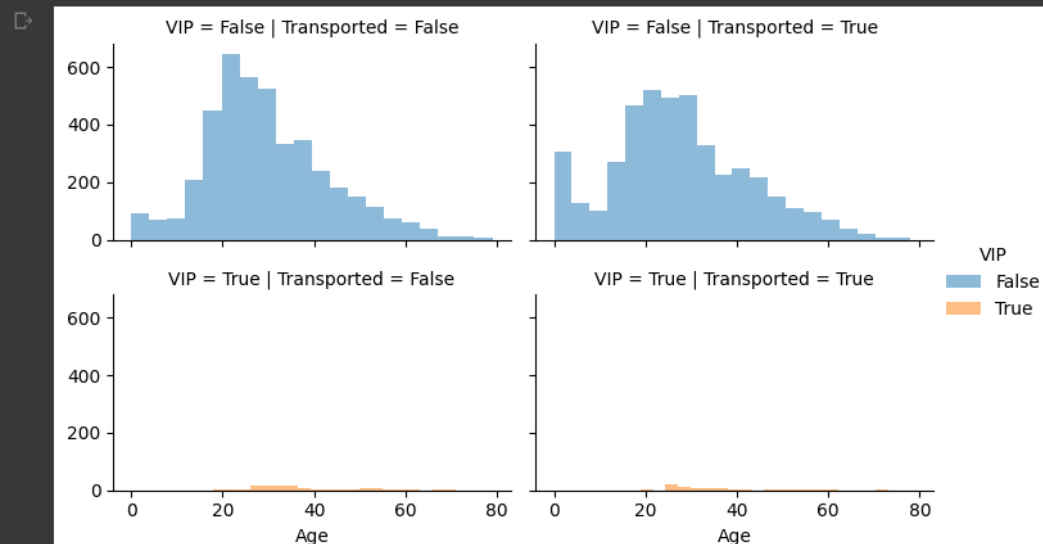
Transported

훈련 자료에서 VIP 여부(VIP)와 이동성공(Transported)에 따른 연령(Age) 분포

```
# 열을 생존 여부, 행(row)과 색깔(hue)을 객실 등급으로 나눔, width = height + aspect
grid = sns.FacetGrid(df, col='Transported', row='VIP', hue='VIP', height=2.2, aspect=1.6)

grid.map(plt.hist, 'Age', alpha=.5, bins=20) # 투명도(alpha): 0.5

# 범례 추가
grid.add_legend();
```





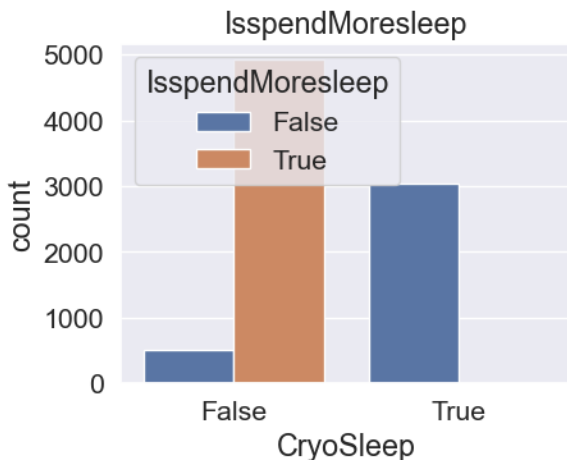
# Data 전처리 – RoomService/FoodCourt/ShoppingMall/Spa/VRDeck

HomePlanet CryoSleep Cabin Destination Age VIP RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported

RoomService/FoodCourt/ShoppingMall/Spa/VRDeck 모두 승객의 소비에 관련된 것들이므로 모두 묶어 **SPEND 피쳐**를 생성

## 1. Cryo Sleep 과의 관계

데이터셋에 isspendmoresleep 피쳐생성 -> Spend항목 중 하나라도 존재하는 사람에 1 값 저장  
-> Cryo Sleep과의 관계파악 -> Spend 결측치 대체



```
1: # Missing values before
Nsp=df[spend].isna().sum().sum()

# CryoSleep has no expenditure
for col in spend:
    df.loc[(df[col].isna()) & (df['CryoSleep']==True), col]=0

# Print number of missing values left
print('#Expenditure missing values before:',Nsp)
print('#Expenditure missing values after:',df[spend].isna().sum().sum())
```

#Expenditure missing values before: 943

#Expenditure missing values after: 582

# Data 전처리 – RoomService/FoodCourt/ShoppingMall/Spa/VRDeck

HomePlanet CryoSleep

Cabin

Destination

Age

VIP

RoomService

FoodCourt

ShoppingMall

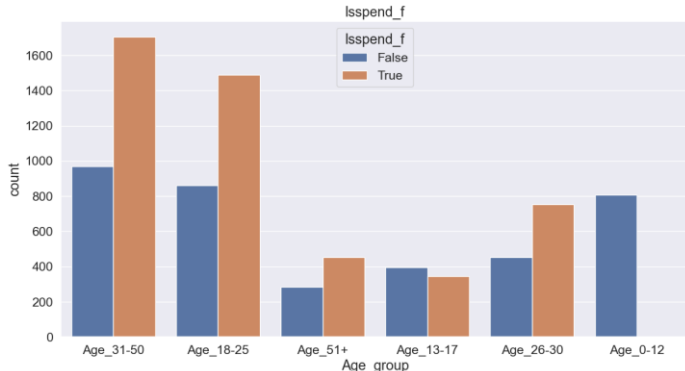
Spa

VRDeck

Name Transported

## 2. Age 와의 관계

Age정보 범주형으로 그룹화 - > Age 12세 이하 spend 항목 결측치 전원 0 처리



```
# Missing values before
Nsp=df[spend].isna().sum().sum()

# 0~12 has no expenditure
for col in spend:
    df.loc[(df[col].isna()) & (df['Age_group']=='Age_0-12'), col]=0

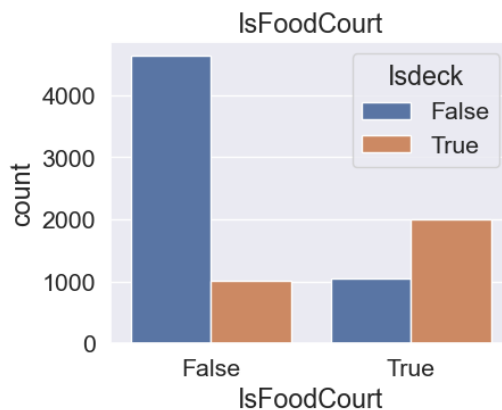
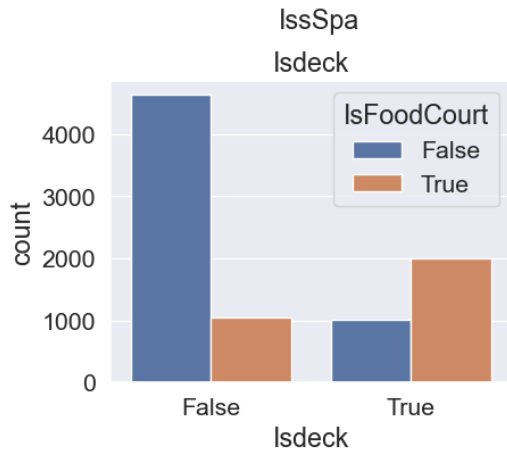
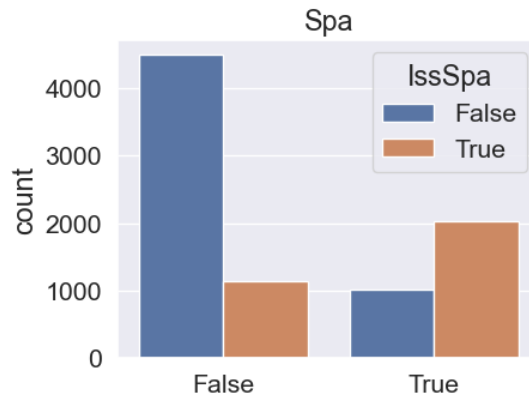
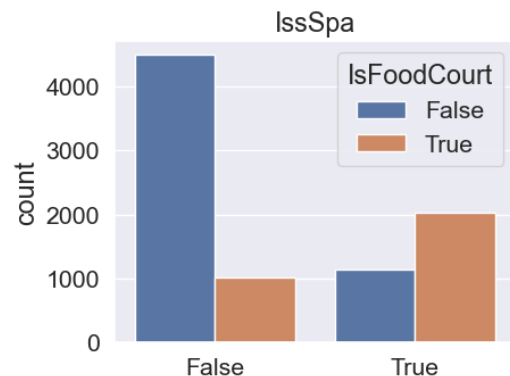
# Print number of missing values left
print('#Expenditure missing values before:',Nsp)
print('#Expenditure missing values after:',df[spend].isna().sum().sum())
```

#Expenditure missing values before: 582

#Expenditure missing values after: 537

# Data 전처리 – RoomService/FoodCourt/ShoppingMall/Spa/VRDeck

HomePlanet CryoSleep Cabin Destination Age VIP RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported



동일한 SPEND  
항목끼리의 존재여부로

Spend 내에서 관계 파악

-> 결측치 채우기.

# Data 전처리 – RoomService/FoodCourt/ShoppingMall/Spa/VRDeck

HomePlanet CryoSleep Cabin Destination Age VIP RoomService FoodCourt ShoppingMall Spa VRDeck Name Transported

Transported와의 음의 상관관계를 이용해 결측치를 채움.

In [28]: *#럭셔리/소박 간의 범주차이에 따른 구분은 크게 없고 전반적으로 수송성공 -> spend false 가 많았으므로 이를 이용해 결측치 대체하자.*

```
# Missing values before
Nsp=df[spend].isna().sum().sum()

# 수송성공 -> spend false
for col in spend:
    df.loc[(df[col].isna()) & (df['Transported']==True), col]=0

# Print number of missing values left
print('#Expenditure missing values before:',Nsp)
print('#Expenditure missing values after:',df[spend].isna().sum().sum())

#Expenditure missing values before: 418
#Expenditure missing values after: 302
```

직접적인 상관관계가 약한 Homplanet 에 따른 spend 항목의 평균값을 이용-> 남은 결측치를 채움.

In [30]:

```
# Missing values before
ABC=df[spend].isna().sum().sum()

for col in spend:
    df.loc[(df[col].isna()) & (df['HomePlanet']=='True'), col]=0

# 평균으로의 대체
for cc in spend:
    rows=df.loc[df[cc].isna(),cc].index
    df.loc[df[cc].isna(),cc]=df.groupby(['HomePlanet'])[cc].transform(lambda x: x.fillna(x.mean()))[rows]

print('#Expenditure missing values before:',ABC)
print('#Expenditure missing values after:',df[spend].isna().sum().sum())

#Expenditure missing values before: 302
#Expenditure missing values after: 0
```

## Data 전처리 - 각각 전처리한 모든 칼럼 최종 병합

```
df1 = pd.read_csv("C:/Users/MSI/Desktop/CJAI/Basic/우주선 타이타닉_베이직/결측치 처리_05.18/Home, Cryo_성현우")
df2 = pd.read_csv("C:/Users/MSI/Desktop/CJAI/Basic/우주선 타이타닉_베이직/결측치 처리_05.18/Cabin, Dest_김예원")
df3 = pd.read_csv("C:/Users/MSI/Desktop/CJAI/Basic/우주선 타이타닉_베이직/결측치 처리_05.18/Age, VIP_정서현")
df4 = pd.read_csv("C:/Users/MSI/Desktop/CJAI/Basic/우주선 타이타닉_베이직/결측치 처리_05.18/Spend_김지호")
```

```
df = pd.concat([df1, df2, df3, df4], axis=1)
```

### 데이터 원 핫 인코딩

```
one_hot_df = pd.get_dummies(Age_train, columns=["HomePlanet", "CryoSleep", "Destination", "Cabin_deck", "Cabin_side", 'VIP'], drop_first=True)
one_hot_df["Transported"].replace({True:1, False:0}, inplace=True)
```

## Feature Selection

**PassengerId**와 **Cabin\_number**(방 번호)는 Transported와 무관할 것으로 예상 -> **피쳐 선택 시 제외**

실제로 Cabin\_number 피쳐를 선택하지 않았을 경우 예측성능이 약 0.21 향상됨.

PassengerId

0001\_01

0002\_01

0003\_01

0003\_02

0004\_01

Cabin\_number

0

0

0

0

1

# 모델 소개 - SVC / LGBM / XGB / RF / KNN

5개의 모델을 정하고 각자  
1개의 모델을 맡아  
하이퍼파라미터 튜닝을 진행.

<div>✓ knn3_submission.csv Complete · 3m ago</div> <div>0.78349</div>	<div>✓ submission2.csv Complete · 15s ago</div> <div>0.78021</div>	<div>✓ submission.csv Complete · 2d ago</div> <div>0.80313</div>
<div>✓ knn2_submission.csv Complete · 9m ago</div> <div>0.78021</div>	<div>✓ submission2.csv Complete · 2d ago</div> <div>0.70984</div>	<div>✓ submission.csv Complete · 2d ago</div> <div>0.80056</div>
<div>✓ knn_submission.csv Complete · 14m ago</div> <div>0.5768</div>	<div>✓ submission1.csv Complete · 4d ago</div> <div>0.69394</div>	<div>✓ submission.csv Complete · 2d ago</div> <div>0.78933</div>
<div>✓ X_cabin_number_gbc_submission.csv Complete · 3d ago</div> <div>0.79892</div>		<div>✓ submission.csv Complete · 2d ago</div> <div>0.80313</div>
<div>✓ submission.csv Complete · 2d ago · Delete Cabin number Using Xgboost</div> <div>0.79424</div>	<div>submission1.csv Complete · 3d ago</div> <div>0.69394</div>	<div>✓ submission.csv Complete · 3d ago</div> <div>0.79939</div>
<div>✓ submission.csv Complete · 2d ago · Index change and Using pycaret knn</div> <div>0.78887</div>	<div>submission2.csv Complete · 3d ago</div> <div>0.69394</div>	
<div>✓ submission.csv Complete · 2d ago · Using pycaret knn</div> <div>0</div>	<div>submission1.csv Complete · 4d ago</div> <div>0.6</div>	<div>✓ submission_Spaceship_Titanic_Seong.csv Complete · 4d ago</div> <div>0.68178</div>
<div>✓ submission.csv Complete · 3d ago · Using Xgboost</div> <div>0.79074</div>		<div>✓ submission_Spaceship_Titanic_Seong.csv Complete · 4d ago</div> <div>0</div>
<div>✓ submission.csv Complete · 4d ago · Using Xgboost</div> <div>0.79027</div>		<div>✓ submission_Spaceship_Titanic_Seong.csv Complete · 4d ago</div> <div>0.53776</div>
		<div>✓ submission_Spaceship_Titanic_Seong.csv Complete · 6d ago</div> <div>0.56511</div>

## 모델 소개 - SVC

```
In [8]: svc = SVC()
        clf = GridSearchCV(svc, param_grid, n_jobs=-1, verbose=2)
        clf.fit(X_train, y_train)
```

Fitting 5 folds for each of 18 candidates, totalling 90 fits

```
Out [8]: GridSearchCV(estimator=SVC(), n_jobs=-1,
                      param_grid={'C': [1.0, 10.0, 100.0], 'degree': [1, 2, 3],
                                   'gamma': ['scale'], 'kernel': ['rbf', 'sigmoid'],
                                   'max_iter': [-1], 'tol': [1e-05]},
                      verbose=2)
```

```
In [13]: clf.best_params_
```

```
Out [13]: {'C': 1.0,
           'degree': 1,
           'gamma': 'scale',
           'kernel': 'rbf',
           'max_iter': -1,
           'tol': 1e-05}
```



submission.csv

Complete · 6h ago

0.80313



# 모델 소개 - LightGBM

```
from hyperopt import fmin, tpe, Trials
```

```
trials = Trials()
```

```
# fmin() 함수를 호출. max_evals 지정된 횟수만큼 반복 후 목적함수의 최소값을 가지는 최적 입력값 추출.  
best = fmin(fn=objective_func, space=lgbm_search_space, algo=tpe.suggest,  
           max_evals=100, # 최대 반복 횟수를 지정합니다.  
           trials=trials, rststate=np.random.default_rng(seed=30))
```

```
print('최적 파라미터 :', best)
```

```
[254] training's binary_logloss: 0.274758 valid_1's binary_logloss: 0.387014  
[255] training's binary_logloss: 0.274411 valid_1's binary_logloss: 0.386987  
[256] training's binary_logloss: 0.274156 valid_1's binary_logloss: 0.387044  
[257] training's binary_logloss: 0.273796 valid_1's binary_logloss: 0.387065  
[258] training's binary_logloss: 0.273478 valid_1's binary_logloss: 0.387127  
[259] training's binary_logloss: 0.273121 valid_1's binary_logloss: 0.387187  
[260] training's binary_logloss: 0.272743 valid_1's binary_logloss: 0.387224  
[261] training's binary_logloss: 0.272344 valid_1's binary_logloss: 0.387243  
[262] training's binary_logloss: 0.271957 valid_1's binary_logloss: 0.387069  
[263] training's binary_logloss: 0.271571 valid_1's binary_logloss: 0.387153  
[264] training's binary_logloss: 0.271221 valid_1's binary_logloss: 0.3873  
[265] training's binary_logloss: 0.270876 valid_1's binary_logloss: 0.387286  
[266] training's binary_logloss: 0.270549 valid_1's binary_logloss: 0.387372  
[267] training's binary_logloss: 0.270111 valid_1's binary_logloss: 0.387183  
[268] training's binary_logloss: 0.269855 valid_1's binary_logloss: 0.387192  
[269] training's binary_logloss: 0.269459 valid_1's binary_logloss: 0.387181  
100% 100/100 [02:46<00:00, 1.67s/trial, best loss: -0.8110440034]  
최적 파라미터 : {'learning_rate': 0.04599390207392148, 'max_depth': 148.0, 'min_child_samples': 68.0, 'num_leaves':  
ple': 0.7816996264425102}
```

```
lgbm_clf = LGBMClassifier(n_estimators=800, num_leaves=int(best['num_leaves']),  
                          max_depth=int(best['max_depth']),  
                          min_child_samples=int(best['min_child_samples']),  
                          subsample=round(best['subsample'], 5),  
                          learning_rate=round(best['learning_rate'], 5))
```

```
# evaluation metric을 accuracy로, early stopping은 500으로 설정하고 학습 수행  
lgbm_clf.fit(X_tr, y_tr, early_stopping_rounds=500,  
            eval_metric="accuracy", eval_set=[(X_tr, y_tr), (X_val, y_val)])
```

```
# 검증 데이터 예측 및 정확도 계산
```

```
y_pred = lgbm_clf.predict(X_test)  
accuracy = accuracy_score(y_test, y_pred)  
print("정확도: {:.4f}".format(accuracy))
```

```
[609] training's binary_logloss: 0.117452 valid_1's binary_logloss: 0.445988  
[610] training's binary_logloss: 0.117284 valid_1's binary_logloss: 0.446178  
[611] training's binary_logloss: 0.117089 valid_1's binary_logloss: 0.446254  
[612] training's binary_logloss: 0.116936 valid_1's binary_logloss: 0.446302  
[613] training's binary_logloss: 0.116902 valid_1's binary_logloss: 0.446532  
[614] training's binary_logloss: 0.116625 valid_1's binary_logloss: 0.446675  
[615] training's binary_logloss: 0.116481 valid_1's binary_logloss: 0.446696  
[616] training's binary_logloss: 0.116344 valid_1's binary_logloss: 0.446822  
[617] training's binary_logloss: 0.11621 valid_1's binary_logloss: 0.447181  
[618] training's binary_logloss: 0.116063 valid_1's binary_logloss: 0.447221  
[619] training's binary_logloss: 0.115901 valid_1's binary_logloss: 0.447216  
[620] training's binary_logloss: 0.11576 valid_1's binary_logloss: 0.447373  
[621] training's binary_logloss: 0.115636 valid_1's binary_logloss: 0.447472  
[622] training's binary_logloss: 0.115496 valid_1's binary_logloss: 0.447641  
[623] training's binary_logloss: 0.11532 valid_1's binary_logloss: 0.447885  
[624] training's binary_logloss: 0.115165 valid_1's binary_logloss: 0.447997  
[625] training's binary_logloss: 0.114953 valid_1's binary_logloss: 0.448016  
[626] training's binary_logloss: 0.114788 valid_1's binary_logloss: 0.448115  
[627] training's binary_logloss: 0.114788 valid_1's binary_logloss: 0.448115  
정확도: 0.8114
```



submission\_Spaceship\_Titanic\_Seong.csv

Complete · 19h ago

0.68178

## 모델 소개 - Xgboost

```
[ ] #Trial 992 finished with value: 0.8112
    params_XGB_best= {'lambda': 3.0610042624477543,
                      'alpha': 4.581902571574289,
                      'colsample_bytree': 0.9241969052729379,
                      'subsample': 0.9527591724824661,
                      'learning_rate': 0.06672065863100594,
                      'n_estimators': 730, #initial value is 651
                      'max_depth': 5,
                      'min_child_weight': 1,
                      'num_parallel_tree': 1}
```

```
[ ] print(get_score(xgb.XGBClassifier(**params_XGB_best),X,y).mean())
```

0.8110037607924149



submission.csv

Complete · now · Delete Cabin number Using Xgboost

0.79424

# 모델 소개 - RandomForest

```
In [27]: params = {
    'n_estimators': [100, 110],
    'max_depth' : [8, 10, 12, 14],
    'min_samples_leaf' : [4, 6, 8],
    'min_samples_split' : [3, 4, 5]
}
# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(random_state=0, n_jobs=-1)
grid_cv = GridSearchCV(rf_clf, param_grid=params, cv=4, n_jobs=-1)
grid_cv.fit(X_train, y_train)

print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

최적 하이퍼 파라미터:

{'max\_depth': 14, 'min\_samples\_leaf': 8, 'min\_samples\_split': 3, 'n\_estimators': 110}

최고 예측 정확도: 0.8067



submission2.csv

Complete · 15s ago

0.78021

# 모델 소개 - KNN

```
from sklearn.model_selection import GridSearchCV
```

```
params = {  
    'n_neighbors' : list(range(1,10)),  
    'leaf_size' : list(range(10,40)),  
    'weights' : ["uniform", "distance"],  
    'metric' : ['euclidean', 'manhattan', 'minkowski']  
}
```

```
knn = KNeighborsClassifier()  
grid_clf = GridSearchCV(knn, param_grid=params, scoring='accuracy', cv=3 )  
grid_clf.fit(X_train, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,  
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,  
                                           metric='minkowski',  
                                           metric_params=None, n_jobs=None,  
                                           n_neighbors=5, p=2,  
                                           weights='uniform'),  
             n_jobs=None,  
             param_grid={'leaf_size': [10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  
                                       20, 21, 22, 23, 24, 25, 26, 27, 28, 29,  
                                       30, 31, 32, 33, 34, 35, 36, 37, 38, 39],  
                         'metric': ['euclidean', 'manhattan', 'minkowski'],  
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9],  
                         'weights': ['uniform', 'distance']}},  
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,  
             scoring='accuracy', verbose=0)
```

```
grid_clf.best_params_
```

```
{'leaf_size': 10,  
 'metric': 'euclidean',  
 'n_neighbors': 9,  
 'weights': 'uniform'}
```



knn3\_submission\_csv.csv

Complete · now

0.78349

감사합니다.

THOHOI