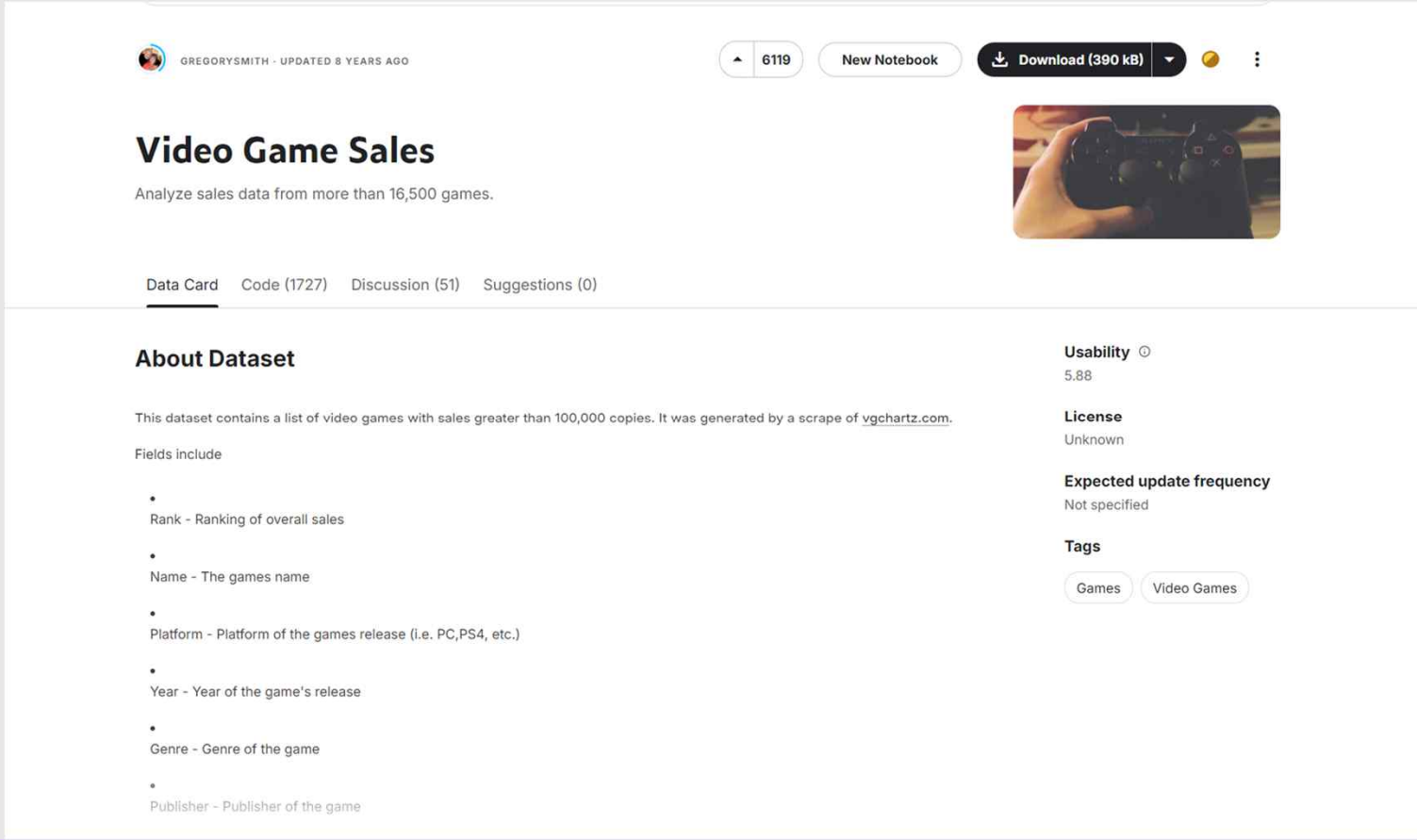
게임 플랫폼 분류

6조 김경환, 김도연, 김영주

INDEX

데이터 출처

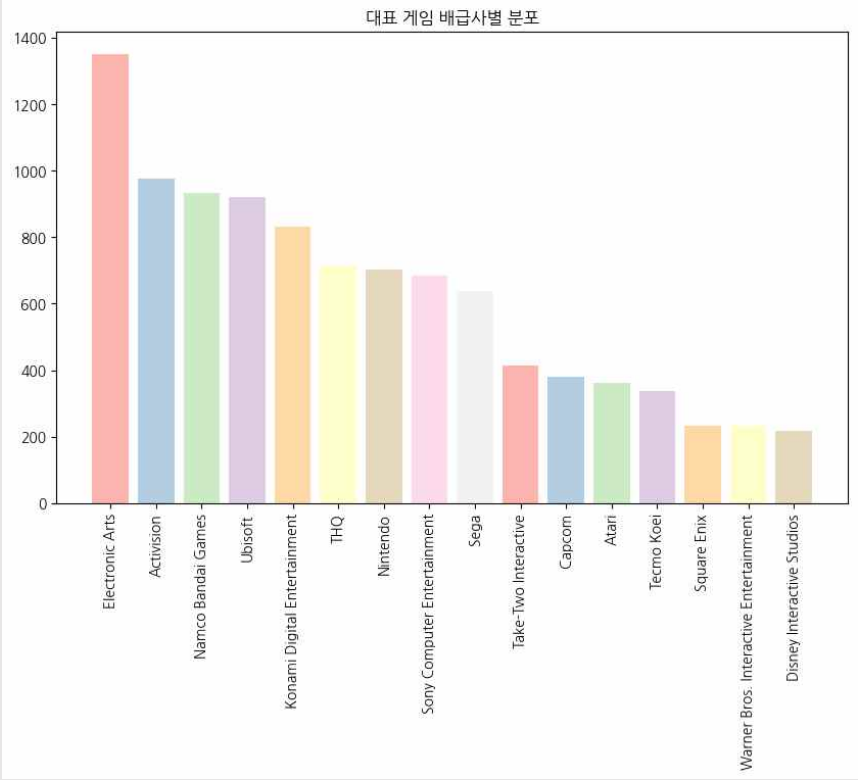


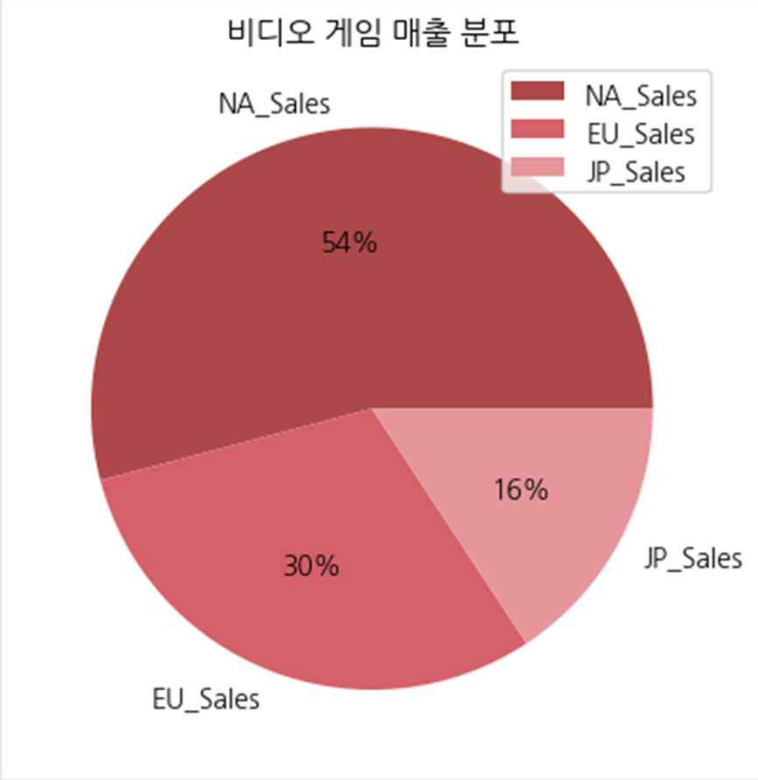
https://www.kaggle.com/datasets/gregorut/videogamesales

주제 선정 이유

공통 관심사 => 게임

주제 선정 이유





이 외에도 100개 이상의 배급사 존재

전처리 과정

Publisher ‘unknown’ 제거
공급한 게임이 15개 이상의 publisher만 사용

```
publisher_counts = df['Publisher'].value_counts()
df = df[~((df['Publisher'] == 'unknown') | (df['Publisher'].isin(publisher_counts[publisher_counts <= 15].index)))]
df
```

✓ 0.0s

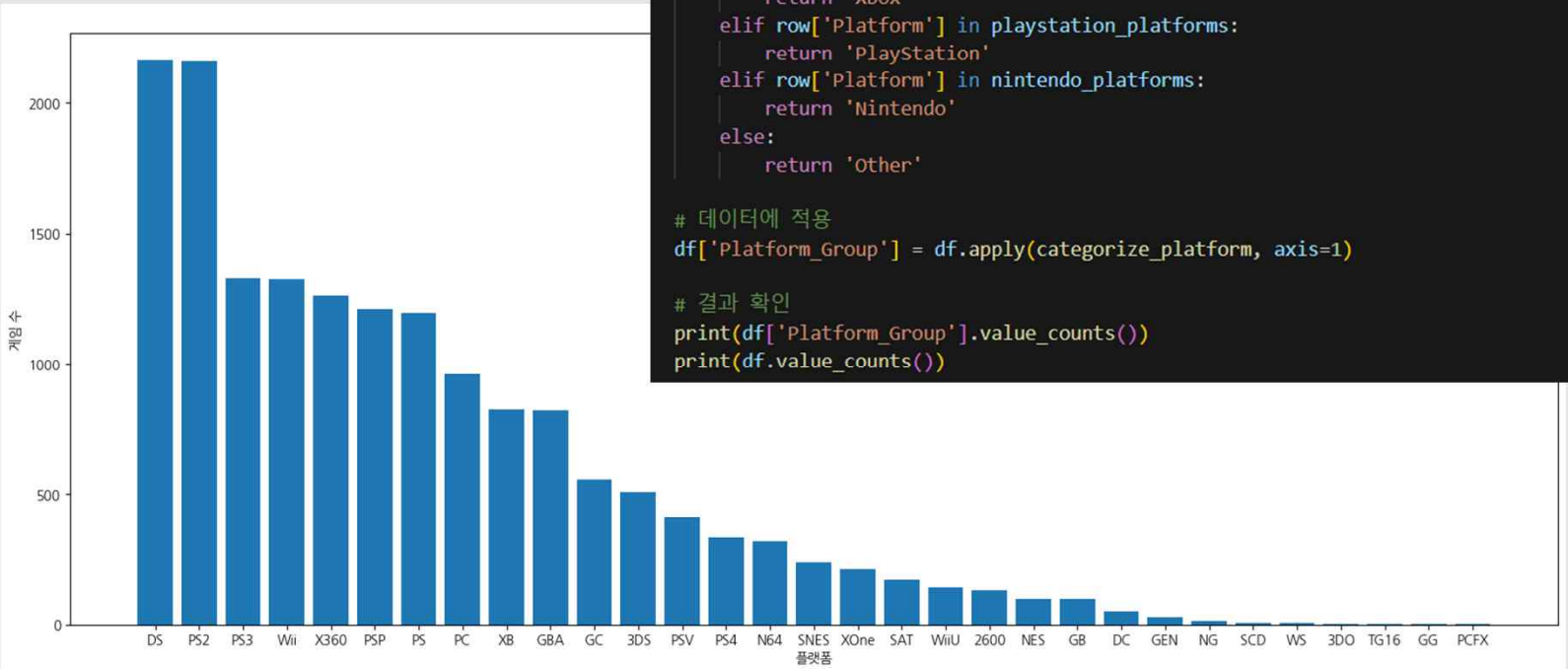
	Rank	Name	Platform	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
0	1	Wii Sports	Wii	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37
...	...	...	...	...	...	...	...	...	...	...
16591	16594	Myst IV: Revelation	PC	Adventure	Ubisoft	0.01	0.00	0.00	0.00	0.01
16592	16595	Plushees	DS	Simulation	Destineer	0.01	0.00	0.00	0.00	0.01
16593	16596	Woody Woodpecker in Crazy Castle 5	GBA	Platform	Kemco	0.01	0.00	0.00	0.00	0.01
16594	16597	Men in Black II: Alien Escape	GC	Shooter	Infogrames	0.01	0.00	0.00	0.00	0.01
16595	16598	SCORE International Baja 1000: The Official Game	PS2	Racing	Activision	0.00	0.00	0.00	0.00	0.01

14875 rows × 10 columns

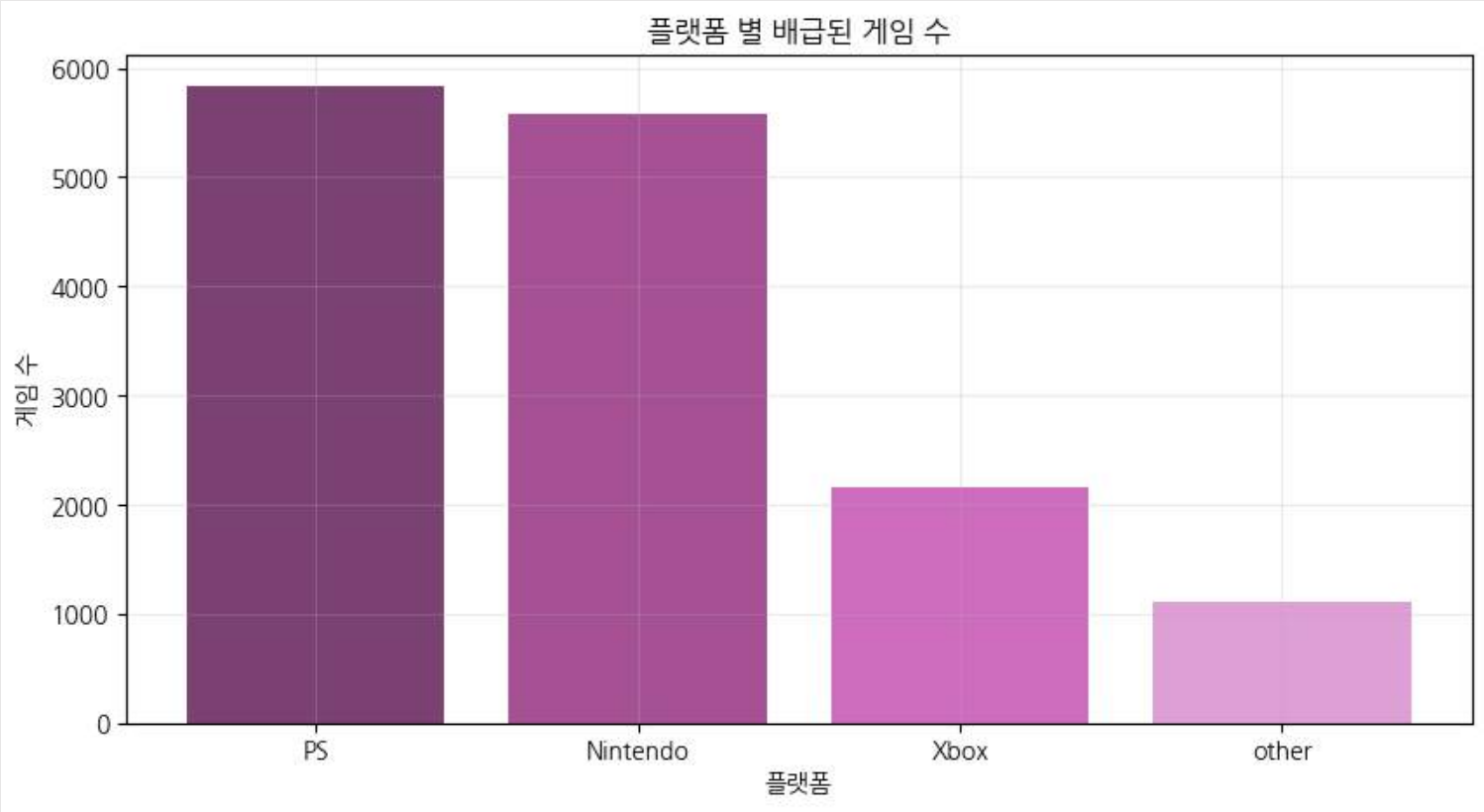
전처리 과정

platform 4개로 분류

```python
# 플랫폼 그룹 정의
xbox_platforms = ['X360', 'XOne', 'XB']  # Xbox 계열
playstation_platforms = ['PS2', 'PS3', 'PS4', 'PS', 'PSP', 'PSV']  # PlayStation 계열
nintendo_platforms = ['Wii', 'WiiU', 'GB', 'GBA', 'DS', '3DS', 'SNES', 'NES', 'N64', 'GC']  # Nintendo 계열

# 새로운 'Platform_Group' 컬럼 추가
def categorize_platform(row):
    if row['Platform'] in xbox_platforms:
        return 'Xbox'
    elif row['Platform'] in playstation_platforms:
        return 'PlayStation'
    elif row['Platform'] in nintendo_platforms:
        return 'Nintendo'
    else:
        return 'Other'

# 데이터에 적용
df['Platform_Group'] = df.apply(categorize_platform, axis=1)

# 결과 확인
print(df['Platform_Group'].value_counts())
print(df.value_counts())
```



전처리 과정


플랫폼 별 배급된 게임 수
6000
5000
4000
3000
2000
1000
0
게임 수
PS
Nintendo
Xbox
other
플랫폼


랜덤 포레스트 모델 활용

랜덤 포레스트

```
[30]
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
x=df[['NA_Sales','EU_Sales','JP_Sales','Other_Sales','Global_Sales','publisher']]
y=df['Platform_Group']
✓ 0.0s
```

```
[31]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=45)
✓ 0.0s
```

```
[32]
from sklearn.preprocessing import LabelEncoder

# Label Encoding으로 문자열 데이터를 숫자로 변환
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)

md_rf = RandomForestClassifier(random_state= 10)
md_rf.fit(X_train, y_train_encoded)
pred_rf = md_rf.predict(X_test)
✓ 1.4s
```

```
[33]
print(f'train score {md_rf.score(X_train,y_train_encoded)}')
print(f'test score {md_rf.score(X_test,y_test_encoded)}')
✓ 0.2s
```

```
...
train score 0.8236974789915966
test score 0.6968067226890756
```

랜덤 포레스트 – 그리드 서치

```python
from sklearn.model_selection import GridSearchCV

# 하이퍼파라미터 그리드 설정
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Grid Search 수행
grid_search = GridSearchCV(estimator=md_rf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train_encoded)

# 최적의 하이퍼파라미터와 점수 출력
print(grid_search.best_params_)
print(grid_search.best_score_)
```

[34] ✓ 43.5s

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
{'max_depth': 30, 'min_samples_split': 10, 'n_estimators': 100}
0.6881507052446425
```

```python
grid_search.best_estimator_.score(X_test,y_test_encoded)
```

[35] ✓ 0.0s

```
0.6968067226890756
```

```python
grid_search.best_estimator_.score(X_train,y_train_encoded)
```

[36] ✓ 0.1s

```
0.7855462184873949
```

로지스틱 회귀 모델 활용

로지스틱 회귀 모델

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Rank          16598 non-null  int64
 1   Name          16598 non-null  object
 2   Platform      16598 non-null  object
 3   Year          16327 non-null  float64
 4   Genre         16598 non-null  object
 5   Publisher     16540 non-null  object
 6   NA_Sales      16598 non-null  float64
 7   EU_Sales      16598 non-null  float64
 8   JP_Sales      16598 non-null  float64
 9   Other_Sales   16598 non-null  float64
 10  Global_Sales  16598 non-null  float64
dtypes: float64(6), int64(1), object(4)
memory usage: 1.4+ MB
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16337 entries, 0 to 16597
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Platform      16337 non-null  object
 1   Genre         16337 non-null  object
 2   Publisher     16337 non-null  object
 3   NA_Sales      16337 non-null  float64
 4   EU_Sales      16337 non-null  float64
 5   JP_Sales      16337 non-null  float64
 6   Global_Sales  16337 non-null  float64
dtypes: float64(4), object(3)
memory usage: 1021.1+ KB
```

로지스틱 회귀 모델

feature/target 분리

```
target_sr=game_df['Platform']
feature_df=game_df.drop('Platform',axis=1)
```

✓ 0.0s

인코딩

- feature: 원 핫 인코딩
- target: 라벨 인코딩

```
from sklearn.preprocessing import LabelEncoder
import numpy as np
```

✓ 0.0s

```
label=LabelEncoder()
target_label=label.fit_transform(target_sr)
target_label
```

✓ 0.0s

```
array([0, 0, 0, ..., 0, 0, 2])
```

```
feature_ohe=pd.get_dummies(feature_df)
```

target: Platform -> 라벨 인코딩

feature: 그 외 -> 원-핫 인코딩

로지스틱 회귀 모델

스케일링

```
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler

ss=StandardScaler()
ss.fit(x_train)

mm=MinMaxScaler()
mm.fit(x_train)

rs=RobustScaler()
rs.fit(x_train)

ss_scaled_train=ss.transform(x_train)
ss_scaled_test=ss.transform(x_test)

mm_scaled_train=mm.transform(x_train)
mm_scaled_test=mm.transform(x_test)

rs_scaled_train=rs.transform(x_train)
rs_scaled_test=rs.transform(x_test)
```

```
ss_train_score=lr.score(ss_scaled_train,y_train)
mm_train_score=lr.score(mm_scaled_train,y_train)
rs_train_score=lr.score(rs_scaled_train,y_train)

print('StandardScaler Score:',{ss_train_score})
print('MinMaxScaler Score:',{mm_train_score})
print('RobustScaler Score:',{rs_train_score})
```

✓ 0.0s

```
StandardScaler Score: {0.5065178495356565}
MinMaxScaler Score: {0.49424895629206783}
RobustScaler Score: {0.6320184033398654}
```

로지스틱 회귀 모델

```
rs_train_score=lr.score(rs_scaled_train,y_train)
rs_test_score=lr.score(rs_scaled_test,y_test)

print(f'train_score: {rs_train_score}, test_score: {rs_test_score}')
```

✓ 0.0s

```
train_score: 0.6320184033398654, test_score: 0.6194207836456559
```

하이퍼 파라미터 튜닝 진행

- penalty: l2, l1
- C: 295, 296, 297, 298, 299
- Solver: lbfgs, liblinear, newton-cg, sag, saga
- Multi class: ovr, auto, multinomial

로지스틱 회귀 모델

```
params={'penalty':['l2','l1'],
        'C':[295,296,297,298,299],
        'solver':['lbfgs','liblinear','newton-cg','sag','saga'],
        'multi_class':['ovr','auto','multinomial']}

lr_clf=LogisticRegression()

grid_clf=GridSearchCV(lr_clf,param_grid=params,scoring='accuracy',cv=3)
grid_clf.fit(rs_scaled_train,y_train)
print('최적 하이퍼 파라미터: {0}, 최적 평균 정확도: {1:.3f}'.format(grid_clf.best_params_,
                                                    grid_clf.best_score_))
```

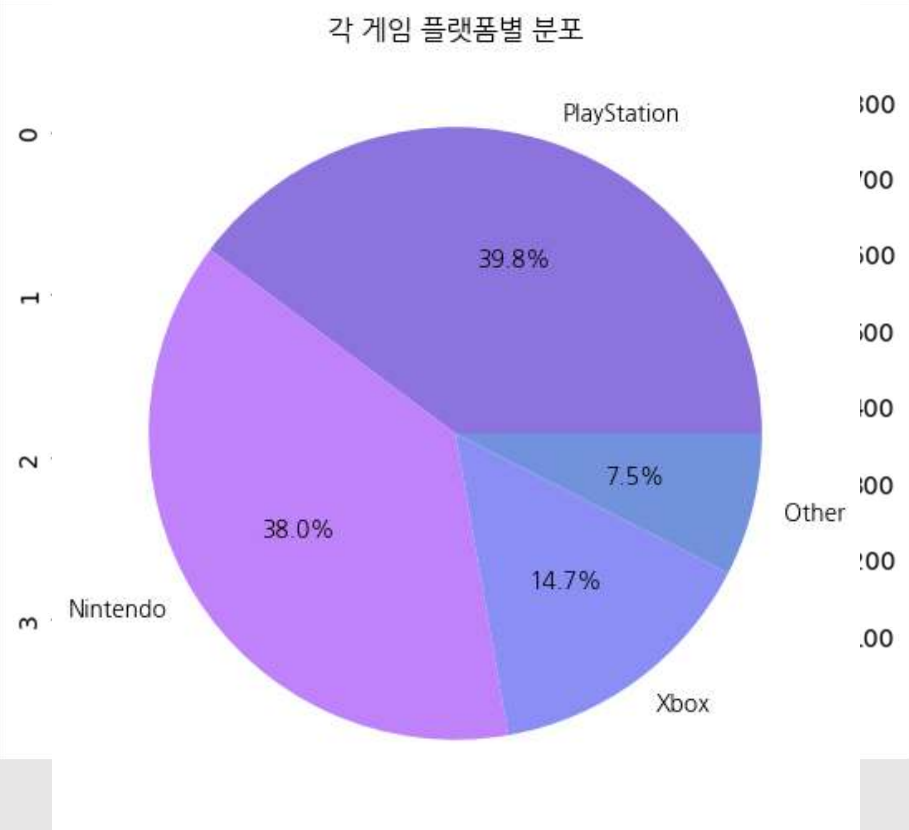
```
lr2=LogisticRegression(C=295,penalty='l2',solver='newton-cg',multi_class='auto')
lr2.fit(rs_scaled_train,y_train)

train_score=lr2.score(rs_scaled_train,y_train)
test_score=lr2.score(rs_scaled_test,y_test)

print(f'train_score: {train_score}, test_score: {test_score}')
```

✓ 7.6s

```
train_score: 0.638919655789384, test_score: 0.6265758091993185
```

Train Score	Test Score
0.63	0.61
0.63	0.62

로지스틱 회귀 모델



	precision	recall	f1-score	support
0	0.63	0.72	0.67	1116
1	0.48	0.21	0.30	221
2	0.65	0.74	0.69	1167
3	0.45	0.23	0.31	431
accuracy			0.62	2935
macro avg	0.55	0.48	0.49	2935
weighted avg	0.60	0.62	0.60	2935

Decision Tree 모델 활용

Decision Tree

타겟 변경 + 데이터 전처리 + 인코딩 방식 시도

타겟 : 플랫폼 (4개로 축약)
인코딩 : 범주형 모두 원-핫 인코딩 처리

```
train_score = dt_model.score(X_train, y_train)
test_score = dt_model.score(X_test, y_test)
print(f'train_score : test_score = {train_score:.4f} : {test_score:.4f}')
```

✓ 0.0s Python

```
train_score : test_score = 0.9495 : 0.6265
```

Decision Tree

ALL(Platform, Genre, Publisher) 원핫

```
ohEncoder = OneHotEncoder()
np.array(targetSR).reshape(-1,1)
featureDF = pd.get_dummies(featureDF)
targetSR = pd.get_dummies(targetSR)
print(f'featureDF - shape : {featureDF.shape}, ndim : {featureDF.ndim}')
print(f'targetSR - shape : {targetSR.shape}, ndim : {targetSR.ndim}')   # get_dummies 사용 시 머신러닝을 진행할 수
type(targetSR)
```

[84] ✓ 0.0s

```
featureDF - shape : (14672, 126), ndim : 2
targetSR - shape : (14672, 4), ndim : 2

pandas.core.frame.DataFrame
```

```
X_train, X_test, y_train, y_test = train_test_split(featureDF, targetSR, random_state=10, stratify=targetSR)

dt_model = DecisionTreeClassifier(random_state=10)
dt_model.fit(X_train, y_train)
```

[83] ✓ 0.3s

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=10)
```

원-핫 인코딩 진행

Decision Tree

```
[dt_model.max_features] 126
[dt_model.feature_importances_] [1.70036704e-01 1.39825945e-01 7.43208040e-02 1.41807010e-01
 1.92447554e-02 1.24753650e-02 7.15628570e-03 1.43508244e-02
 1.01231910e-02 7.30260529e-03 9.49983434e-03 9.25935769e-03
 1.50807287e-02 1.22717532e-02 1.29118890e-02 8.24176383e-03
 2.27876203e-04 5.68847653e-03 1.29024680e-03 2.15476883e-04
 2.60428563e-03 1.30309814e-02 1.08356742e-03 4.53228567e-04
 1.16236524e-03 1.13958899e-03 8.34189256e-03 1.30207677e-03
 1.16962835e-03 4.24375523e-04 2.10224001e-03 2.91109476e-03
 1.04782017e-03 5.49866740e-04 7.12792535e-03 1.07184906e-03
 6.68114562e-04 3.96444795e-03 7.13126819e-04 1.30762050e-03
 4.33599866e-03 1.16814665e-03 4.28245533e-03 4.52086730e-04
 5.63366107e-03 9.38235160e-05 3.17508691e-03 1.61878881e-02
 1.11337314e-03 6.82859927e-04 7.41662584e-04 2.07994075e-03
 5.97351204e-04 2.48750645e-04 6.57446688e-04 1.91121977e-04
 4.80530835e-04 4.07894944e-04 2.34867118e-03 4.07395446e-03
 1.90253241e-04 4.16907160e-04 1.53795046e-03 1.21623076e-04
 8.59543056e-04 1.34654120e-04 5.26985796e-04 1.36222275e-03
 3.12020109e-04 3.82243954e-05 1.15409148e-02 4.74937620e-04
 1.59820209e-03 3.21765053e-03 1.34698505e-03 3.18678218e-04
 8.98075965e-04 0.00000000e+00 1.62841438e-02 0.00000000e+00
 2.80188225e-03 4.26228215e-04 3.90764478e-04 8.98391741e-03
 2.99637637e-04 4.68247407e-02 3.42592641e-03 1.68640247e-03
 6.65435090e-04 7.96652136e-04 1.05268838e-03 6.33760674e-04
...
 8.29180227e-03 3.55858768e-03 2.16669164e-03 6.80550665e-04
 1.34081327e-04 0.00000000e+00 1.28733054e-02 3.17027433e-04
 1.21098677e-03 2.45349601e-03 6.05139583e-03 6.30332827e-04
 5.42960162e-05 5.15017392e-04]
```

```
# print(f'[dt_model.classes_] {dt_model.classes_}')
print(f'[dt_model.n_classes_] {dt_model.n_classes_}')
print(f'[dt_model.max_features] {dt_model.max_features_}')
print(f'[dt_model.feature_names_in_] {dt_model.feature_names_in_}')
del.feature_importances_] {dt_model.feature_importances_}')
```

Python

```
s_] [2 2 2 2]
ures] 126
names_in_] ['NA_Sales' 'EU_Sales' 'JP_Sales' 'Global_Sales' 'Genre_Action'
 'Genre_Fighting' 'Genre_Misc' 'Genre_Platform'
enre_Racing' 'Genre_Role-Playing' 'Genre_Shooter'
' 'Genre_Sports' 'Genre_Strategy' 'Publisher_3DO'
mes' 'Publisher_5pb' 'Publisher_ASCII Entertainment'
m Entertainment' 'Publisher_Activision'
sion Value' 'Publisher_Alchemist' 'Publisher_Aqua Plus'
stem Works' 'Publisher_Atari' 'Publisher_Atlus'
est' 'Publisher_BAM! Entertainment'
sto' 'Publisher_Bethesda Softworks'
 Bean Games' 'Publisher_Broccoli' 'Publisher_Capcom'
ft' 'Publisher_City Interactive' 'Publisher_Codemasters'
e Heart' 'Publisher_Crave Entertainment'
isher' 'Publisher_DTP Entertainment'
ilver' 'Publisher_Destineer'
 Interactive Studios'
atcher Interactive' 'Publisher_Eidos Interactive'
onic Arts' 'Publisher_Empire Interactive'
orporation' 'Publisher_Falcom Corporation'
 Home Interactive' 'Publisher_FuRyu' 'Publisher_GSP'
eractive' 'Publisher_Game Factory'
 Star' 'Publisher_Hasbro Interactive'
 Soft' 'Publisher_Idea Factory'
```

```
8.29180227e-03 3.55858768e-03 2.16669164e-03 6.80550665e-04
```

피쳐 중요도

Decision Tree

```
[gscv.best_params_] {'max_depth': 12, 'min_samples_leaf': 2}
[gscv.best_score_] 0.5657017058361902
[gscv.best_estimator_] DecisionTreeClassifier(max_depth=12, min_samples_leaf=2)
[best_model.max_depth] : 12
[best_model.min_samples_leaf] : 2
```

```
X_train, X_test, y_train, y_test = train_test_split(featureDF, targetSR, random_state=10, stratify=targetSR)

dt_model = DecisionTreeClassifier(random_state=10, max_depth = 20)
dt_model.fit(X_train, y_train)
print(dt_model.score(X_test,y_test))
```

✓ 0.2s

```
0.6123227917121047
```

하이퍼 파라미터

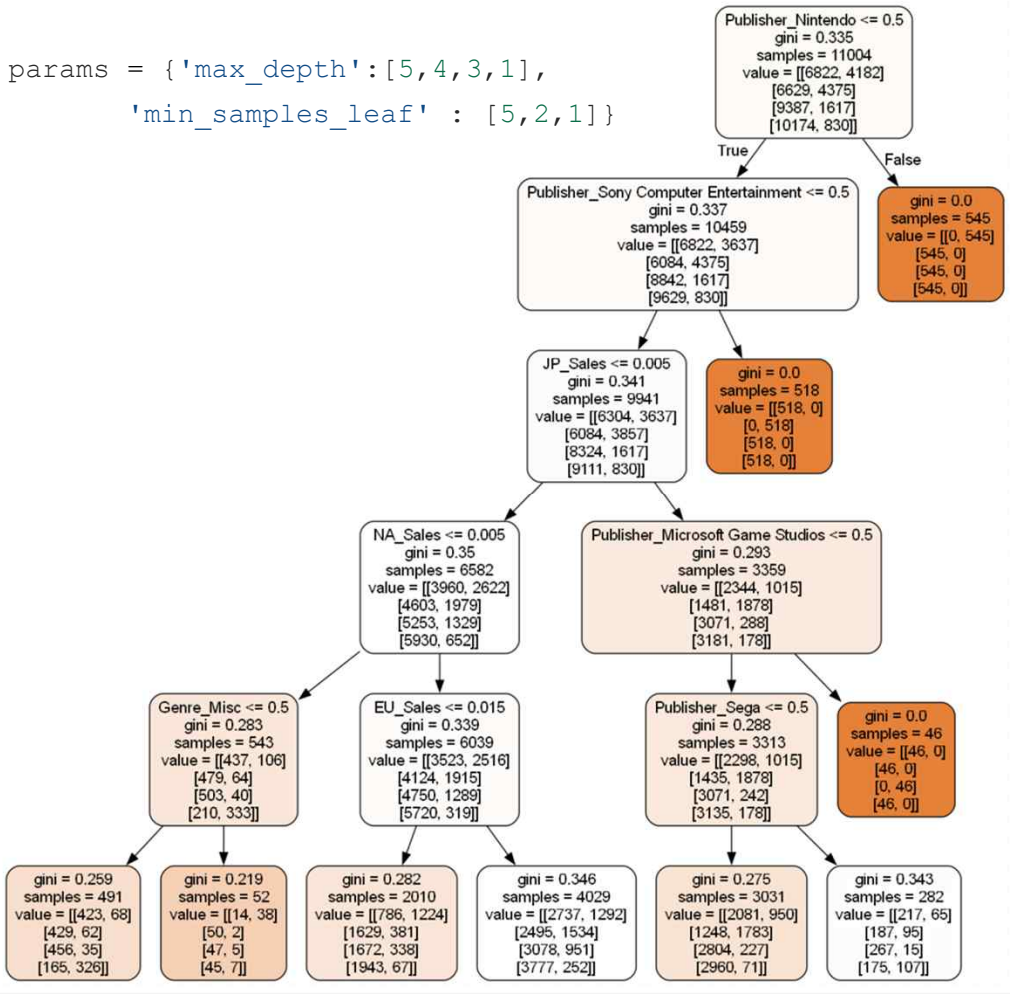
Decision Tree

```
params = {'max_depth':[5,4,3,1],
          'min_samples_leaf' : [5,2,1]}

gscv=GridSearchCV(DecisionTreeClassifier(),
                  param_grid=params,
                  refit=True,
                  return_train_score=True)
gscv.fit(X_train, y_train)

print(f'[gscv.best_params_] {gscv.best_params_}')
print(f'[gscv.best_score_] {gscv.best_score_}')
print(f'[gscv.best_estimator_] {gscv.best_estimator_}')
# print(f'[gscv.cv_results_] {gscv.cv_results_}')

cv_resultDF = pd.DataFrame(gscv.cv_results_)

best_model = gscv.best_estimator_
print(f'[best_model.max_depth] : {best_model.max_depth}')
print(f'[best_model.min_samples_leaf] : {best_model.min_samples_leaf}')
```

✓ 2.9s Python

```
[gscv.best_params_] {'max_depth': 5, 'min_samples_leaf': 5}
[gscv.best_score_] 0.40912374540498125
[gscv.best_estimator_] DecisionTreeClassifier(max_depth=5, min_samples_leaf=
[best_model.max_depth] : 5
[best_model.min_samples_leaf] : 5
```

```
params = {'max_depth':[5,4,3,1],
          'min_samples_leaf' : [5,2,1]}
```



GridSearchCV

Decision Tree

```
params = {'max_depth':[12,9,6,3],
          'min_samples_leaf' : [5,3,2]}

gscv=GridSearchCV(DecisionTreeClassifier(),
                  param_grid=params,
                  refit=True,
                  return_train_score=True)
gscv.fit(X_train, y_train)

print(f'[gscv.best_params_] {gscv.best_params_}')
print(f'[gscv.best_score_] {gscv.best_score_}')
print(f'[gscv.best_estimator_] {gscv.best_estimator_}')
# print(f'[gscv.cv_results_] {gscv.cv_results_}')

cv_resultDF = pd.DataFrame(gscv.cv_results_)

best_model = gscv.best_estimator_
print(f'[best_model.max_depth] : {best_model.max_depth}')
print(f'[best_model.min_samples_leaf] : {best_model.min_samples_leaf}')
```

✓ 4.4s

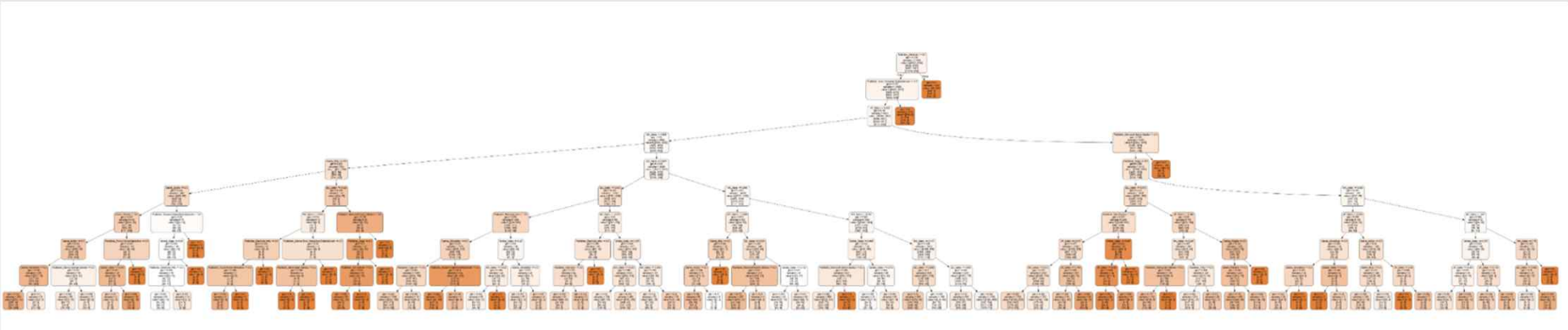
```
[gscv.best_params_] {'max_depth': 12, 'min_samples_leaf': 2}
[gscv.best_score_] 0.5657017058361902
[gscv.best_estimator_] DecisionTreeClassifier(max_depth=12, min_samples_leaf=2)
[best_model.max_depth] : 12
[best_model.min_samples_leaf] : 2
```

GridSearchCV

Decision Tree

params = {'max_depth':[9,4,3,1],

'min_samples_leaf' : [5,3,2]}



Decision Tree

	precision	recall	f1-score	support
0	0.65	0.64	0.65	1394
1	0.71	0.71	0.71	1459
2	0.41	0.37	0.38	539
3	0.63	0.58	0.61	276
micro avg	0.64	0.62	0.63	3668
macro avg	0.60	0.57	0.59	3668
weighted avg	0.64	0.62	0.63	3668
samples avg	0.62	0.62	0.62	3668


예측

예측

```python
def pred_ML(X_new):
        y_pred = dt_model.predict(X_new)
        proba = dt_model.predict_proba(X_new)
        if y_pred[0][0] == True:
                y_pred = 'Nintendo'
        elif y_pred[0][1] == True:
                y_pred = 'PS'
        elif y_pred[0][2] == True:
                y_pred = 'Xbox'
        else:
                y_pred = 'other'
        return [y_pred, proba]

def y_True_return(idx):
    y_temp = y_test[y_test.index == idx]
    for c in y_temp.columns.to_list():
        if y_temp[c].any() == True:
            re = c
    return re

import random as rd
for i in range(5):
    x = rd.randint(0,3368)
    x = list(X_test.index)[x]
    print(f'정답 : {y_True_return(x)}')
    tempP = pred_ML(X_test[X_test.index == x])
    print(f'Predict : {tempP[0]}  ||   Proba : {tempP[1]}')
    print()
```

✓ 0.0s Python

예측

```
정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Xbox
Predict : Xbox  ||   Proba : [array([[1., 0.]]), array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Xbox
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : other
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Xbox
Predict : Xbox  ||   Proba : [array([[1., 0.]]), array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]])]

정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]
```

X_test의 데이터를 활용해 Predict. Proba 진행

예측

PS와 Nintendo의 예측은 뛰어나지만,

Xbox와 other부분이 미흡

```
정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]
```

```
정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Xbox
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : other
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : PS
Predict : PS  ||   Proba : [array([[1., 0.]]), array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]])]

정답 : Nintendo
Predict : Nintendo  ||   Proba : [array([[0., 1.]]), array([[1., 0.]]), array([[1., 0.]]), array([[1., 0.]])]
```