作业三 Video Game Sales 电子游戏销售分析

范啸猛

3220200870

电子游戏市场分析

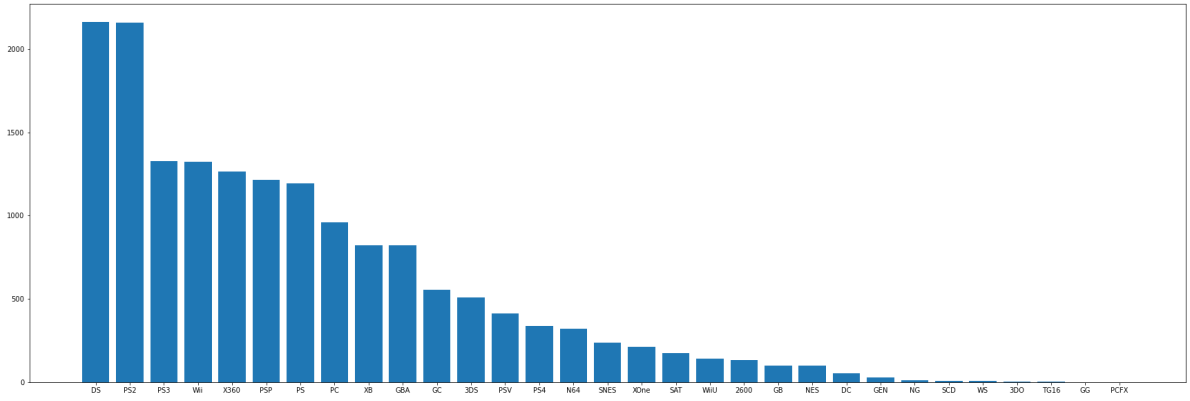
Video Game Sales 数据集中包含属性有游戏名称，平台，发行年份，发行商，销售数据。下面，我们将依次对游戏平台，发行年份，发行商与游戏数目，游戏销售数据之间的关系进行分析

对平台的相关数据进行

- 首先，我们将对各个平台的游戏数量进行可视化分析，代码如下：

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
data_name = r'C:\Users\Xiaomeng Fan\Documents\WeChat
Files\wxid_0xm3leswco1922\FileStorage\File\2021-06\vgsales.csv'
file = pd.read_csv(data_name)
data = pd.DataFrame(file)

plt.figure(figsize=(30,10))
plt.bar(data['Platform'].value_counts().index,data['Platform'].value_counts())
```

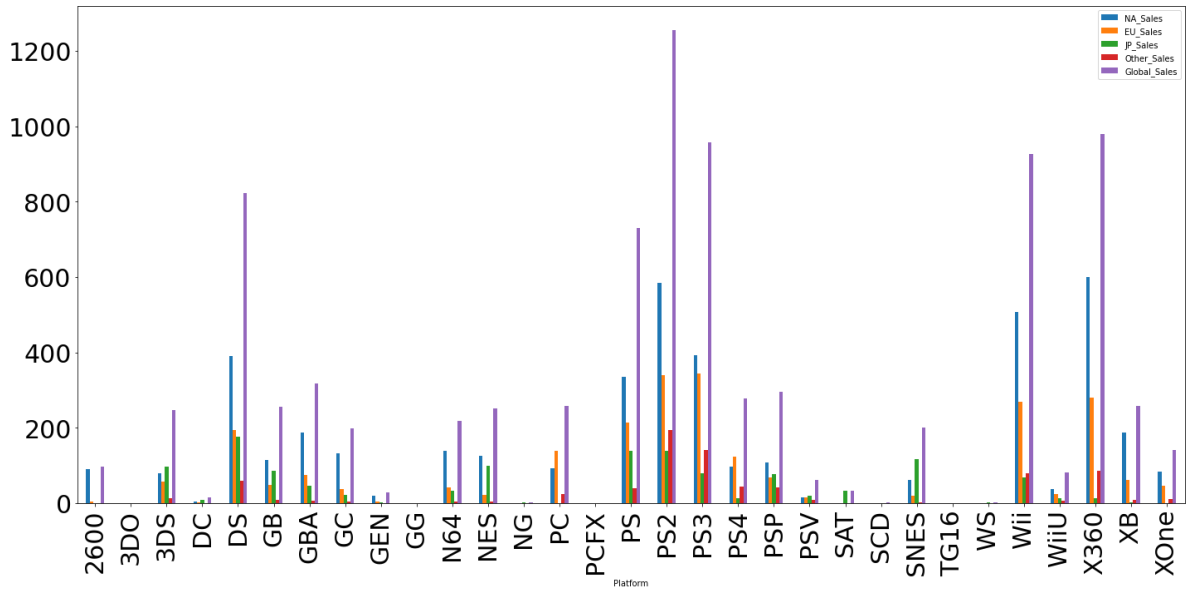
```
<BarContainer object of 31 artists>
```



由此可以看出，排名前五的游戏平台为DS,PS2,PS3,Wii,X360。并且，DS与PS2平台的游戏数远远大于其他公司的游戏数。

下面，我们对全球销量前十的平台进行可视化，代码如下：

```
sales1=data[['Platform', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales',
'Global_Sales']].groupby('Platform').sum()
# sales1.index=sales1.index.astype(int)
sales1.plot.bar(figsize=(20,10),fontsize=30)
plt.tight_layout()
plt.show()
```

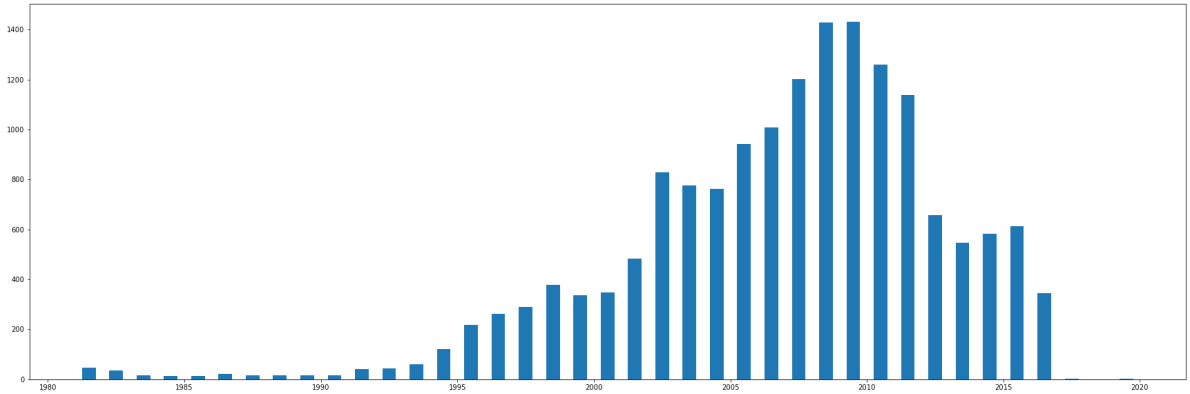


由此可以看出，PS2，PS3，X360是销售数据的前三名，而DS虽然游戏数量多，但是销售数据并不是最多的。在游戏的欢迎程度上，PS系列的平台是最受欢迎的平台。

对年份属性进行分析

- 下面我们将对每年发行的游戏数目进行可视化，代码如下：

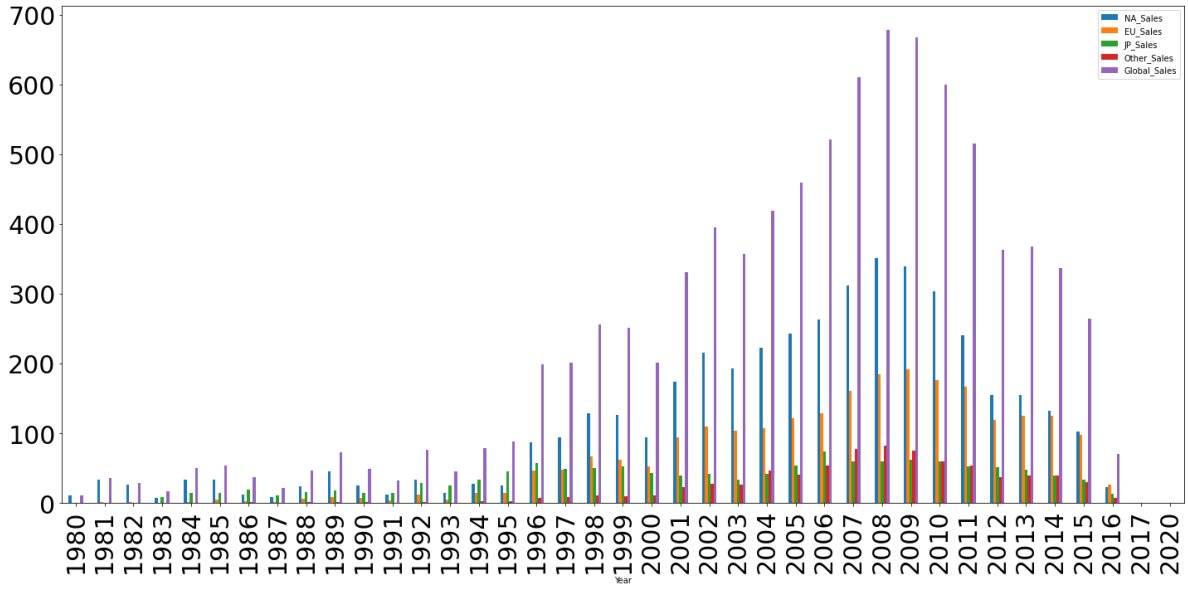
```
plt.figure(figsize=(30,10))
plt.hist(data['Year'],bins=[a for a in range(1981,2021)],rwidth=0.5)
plt.show()
```



可以看出，发行游戏最多的年份是在2007年左右，并且在2016年之后游戏的发行数骤减。

- 下面将绘制年份与销售数据的相关图像，代码如下：

```
sales1=data[['Year', 'NA_Sales', 'EU_Sales', 'JP_Sales',
'Other_Sales','Global_Sales']].groupby('Year').sum()
sales1.index=sales1.index.astype(int)
sales1.plot.bar(figsize=(20,10),fontsize=30)
plt.tight_layout()
plt.show()
```

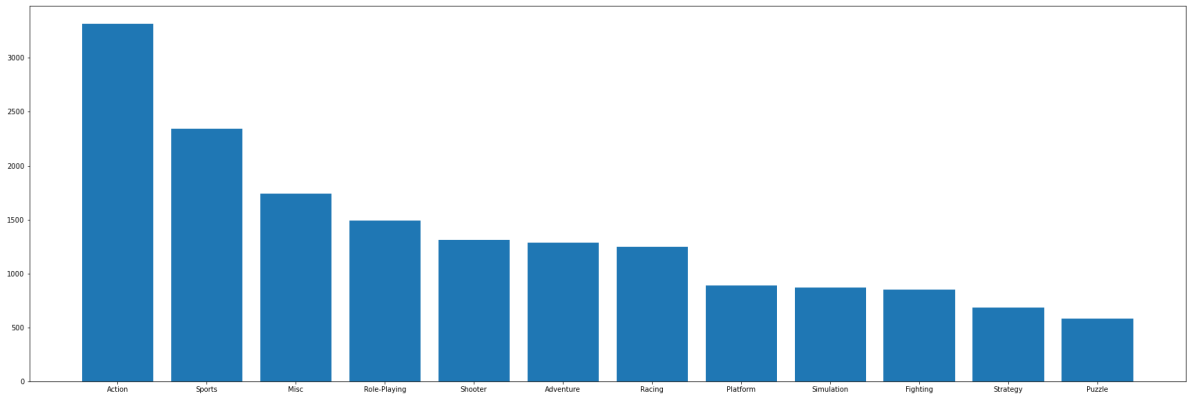


由此可以看出，在2007-2009年，游戏的销售额到达了顶峰，但是在2009年之后，销售数据逐年下降，在2011年之后更是骤降，在2017之后销售数据趋向于0.可以得到，当下video game并不再向之前那样受大家的欢迎。

游戏类型相关数据分析

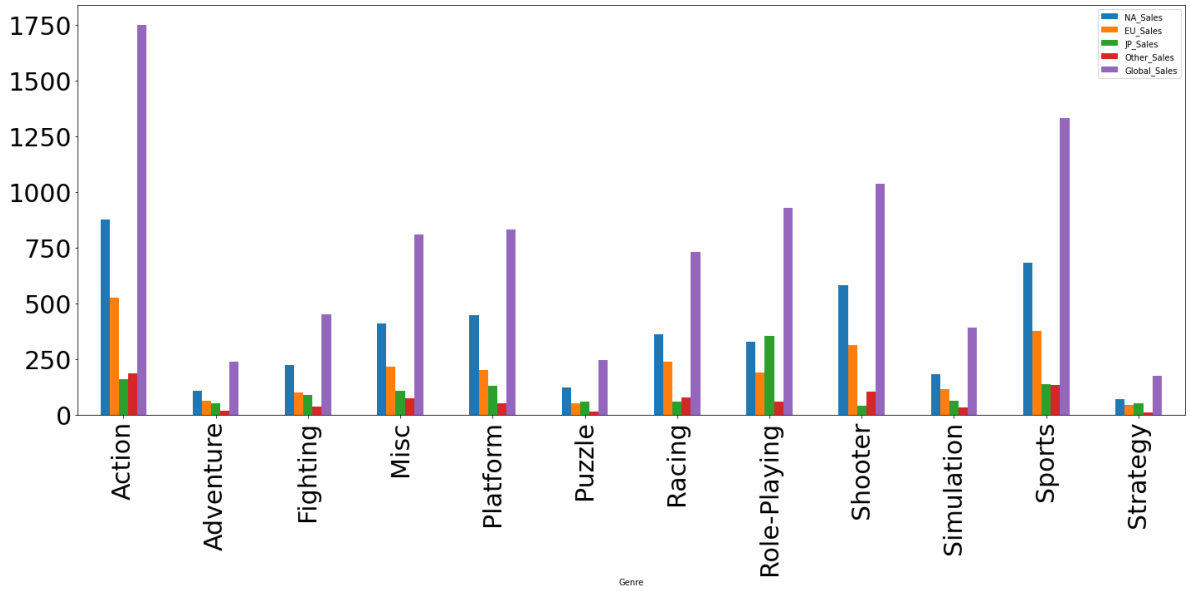
- 我们将对哪一种类型的游戏拥有最多的游戏数目进行分析

```
plt.figure(figsize=(30,10))
plt.bar(data['Genre'].value_counts().index,data['Genre'].value_counts())
plt.show()
```



可以看出，排名前五名的类型为action，sports,Moc和Role-playing。并且action的游戏数目远远多于其他类型的游戏。

```
sales=data[['Genre', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales',
'Global_Sales']]
sales = sales.groupby('Genre').sum()
sales.plot.bar(figsize=(20,10),fontsize=30)
plt.tight_layout()
plt.show()
```

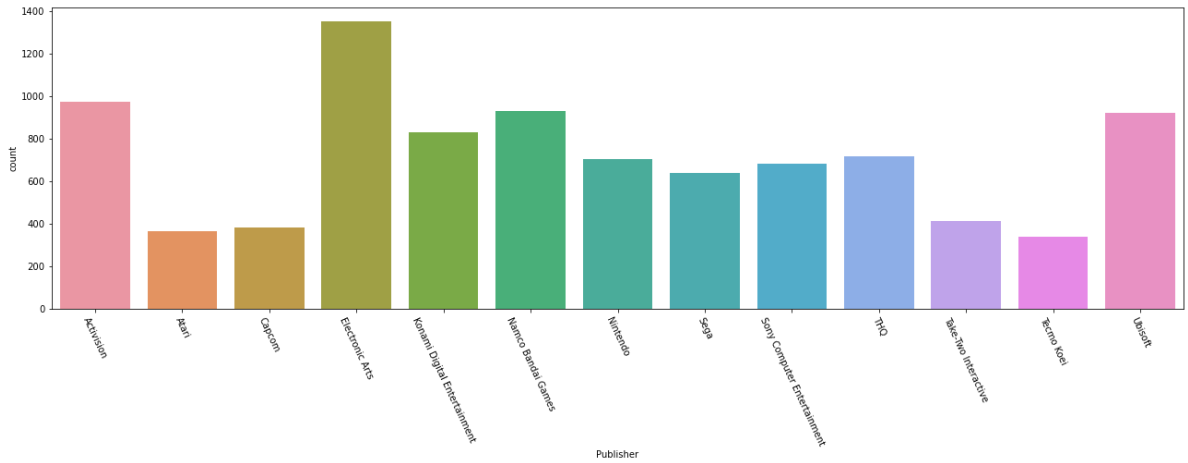


action、sports、shooter的销售额是最高的，这与相对应的游戏数目基本一致。

发行商相关数据分析

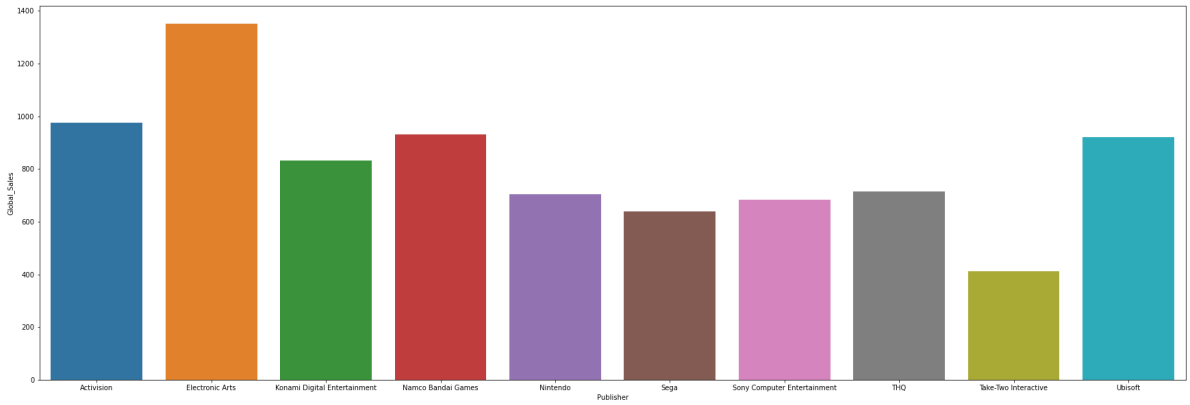
- 下面将对游戏的发行商发行的游戏数目进行可视化分析，代码如下：

```
df_Publisher=pd.DataFrame(data.groupby("Publisher").Rank.count())
df_Publisher = df_Publisher.rename(columns={"Rank": "count"})
df_Publisher = df_Publisher.loc[df_Publisher["count"] > 300]

ingredients = list(df_Publisher.index)
plt.figure(figsize=(22,6))
a = sns.barplot(x=df_Publisher.index, y=df_Publisher["count"])
a.set_xticklabels(labels=df_Publisher.index, rotation=-65)
plt.show()
```



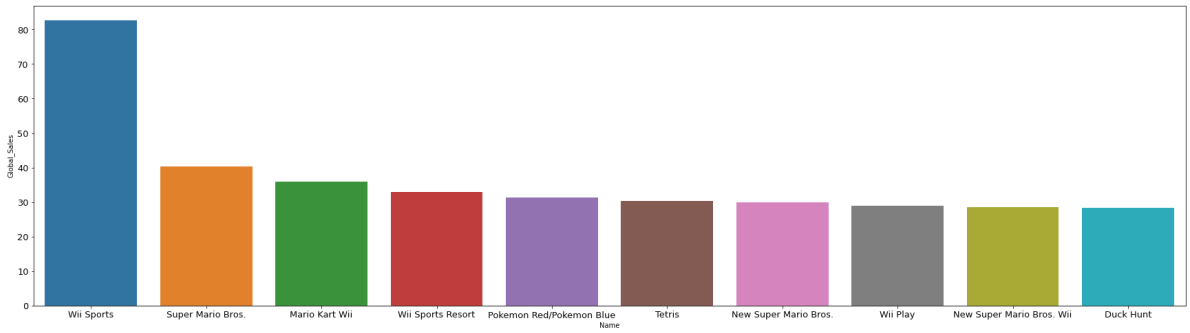

```
dff = data.groupby(['Publisher'], as_index=False)['Global_Sales'].count()
dff=dff[dff['Global_Sales']>400]
plt.figure(figsize=(30,10))

sns.barplot(x='Publisher',data=dff,y='Global_Sales')
```

```
<AxesSubplot:xlabel='Publisher', ylabel='Global_Sales'>
```



- 下面将对全球销量最高前十的游戏进行可视化，代码如下：

```
df=data[['Name','Global_Sales']].sort_values('Global_Sales',ascending=False)
[:10]
plt.figure(figsize=(30,8))

a= sns.barplot(data=df,x='Name',y='Global_Sales')
# a.set_xticklabels(labels=df.index, rotation=-65)
plt.tick_params(labelsize=13)
plt.show()
```



由此，可以看出全球销量最高前十的游戏是Wii sports, Super Mario Bros, Mario Kart Wii等。最受欢迎的游戏是Wii sports。

通过上述分析，我们可以得到以下结论：

- 最受大家欢迎的游戏平台是PS系列的游戏平台
- 最受大家欢迎的游戏类型是action类型
- 最受大家欢迎的发行商是Electronic Arts
- 最受大家欢迎的游戏是Wii Sports
- Video game的热度再2007-2010达到了顶峰，随后热度开始骤降。

预测销售数据

数据处理

- 首先检测数据是否存在空值，代码如下：

```
dataset = pd.read_csv(data_name)
dataset.info()
```

```
<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
```

```
dataset.isnull().values.any()
```

```
True
```

检查那一列存在缺失值，代码如下

```
print(dataset['Rank'].isnull().values.any())
print(dataset['Name'].isnull().values.any())
print(dataset['Platform'].isnull().values.any())
print(dataset['Year'].isnull().values.any())
print(dataset['Genre'].isnull().values.any())
print(dataset['Publisher'].isnull().values.any())
print(dataset['NA_Sales'].isnull().values.any())
print(dataset['EU_Sales'].isnull().values.any())
print(dataset['JP_Sales'].isnull().values.any())
print(dataset['Other_Sales'].isnull().values.any())
print(dataset['Global_Sales'].isnull().values.any())
```

```
False
False
False
True
False
True
False
False
False
False
False
```

检查缺失值的个数，代码如下，

```
print(dataset['Year'].isnull().sum())
print(dataset['Publisher'].isnull().sum())
```

```
271
58
```

移除数据集中的缺失值，代码如下，

```
dataset = dataset.dropna(axis=0, subset=['Year','Publisher'])
dataset.isnull().values.any()
```

```
False
```

将数据集中的游戏平台与游戏发行商换成数值属性，代码如下，

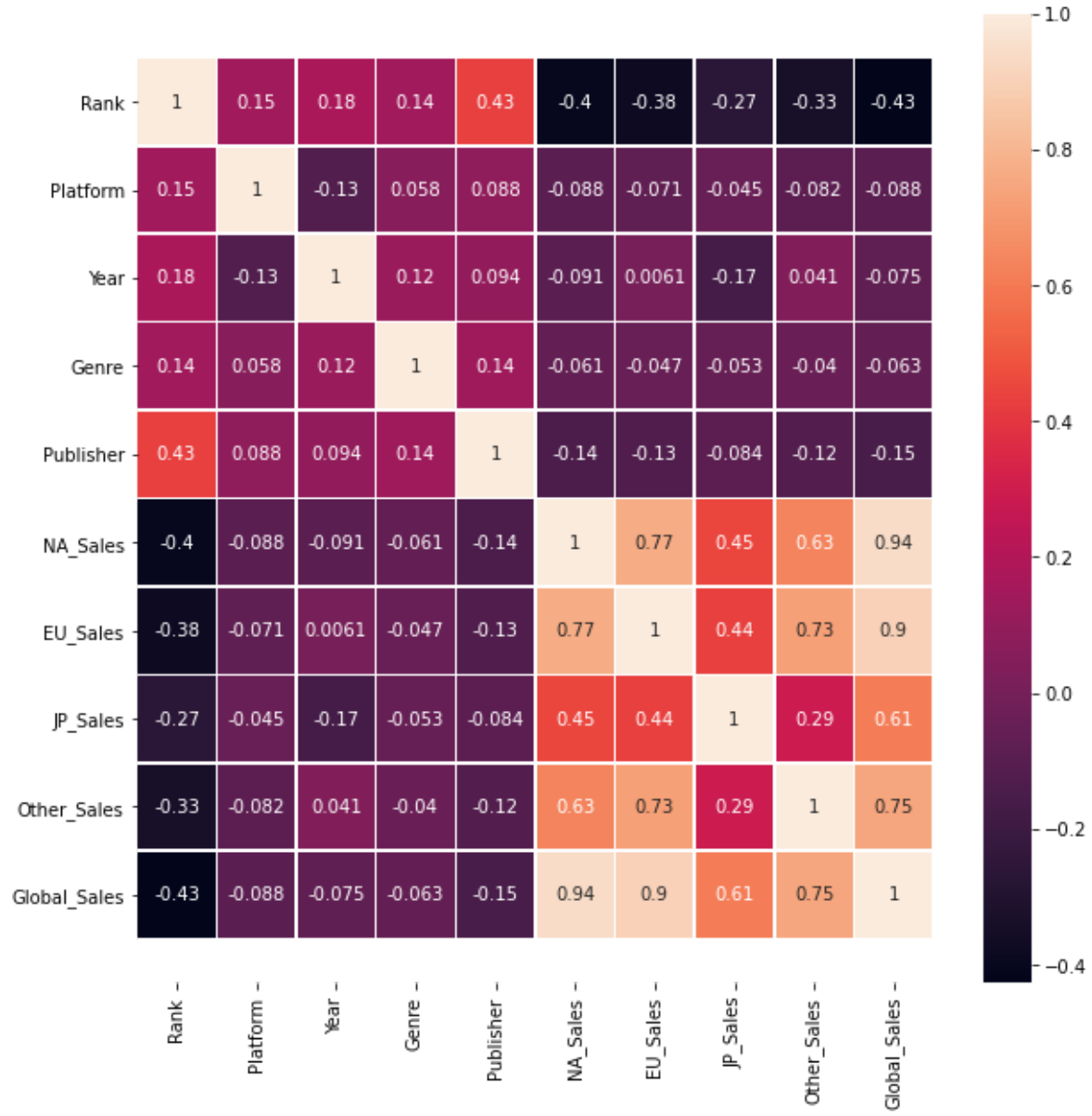
```
x = dataset.iloc[:,1:-1].values
y = dataset.iloc[:,-1].values
print(x[1])
print(y[1])

dataset[dataset.columns[2]]=dataset[dataset.columns[2]].factorize()[0]
dataset[dataset.columns[4]]=dataset[dataset.columns[4]].factorize()[0]
dataset[dataset.columns[5]]=dataset[dataset.columns[5]].factorize()[0]
```

```
['Super Mario Bros.' 'NES' 1985.0 'Platform' 'Nintendo' 29.08 3.58 6.81
 0.77]
40.24
```

```
x = dataset.iloc[:,1:-1].values
y = dataset.iloc[:,-1].values
print(x[1])
print(y[1])
```

```
['Super Mario Bros.' 1 1985.0 1 0 29.08 3.58 6.81 0.77]
40.24
```

```
corrmat = dataset.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))
#Plotting heat map
g=sns.heatmap(dataset[top_corr_features].corr(),annot=True,linewidths=.5)
b, t = plt.ylim() # Finding the values for bottom and top
b += 0.5
t -= 0.5
plt.ylim(b, t)
plt.show()
```



由此可以看出，NA_sales,EU_sales,JP_sales,other_sales与全球销量呈现非常明显的相关关系。为使模型更加精确，在这里我们提出引入其他的属性来拟合更强的模型。

```
x = dataset.iloc[:,[2,3,4,5,6,7,8,9]].values
print(x[0])
```

```
[   0.   2006.      0.      0.     41.49   29.02    3.77    8.46]
```

将数据集划分成训练集与测试集，代码如下：

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
```

利用线性回归的方法来进行预测

```
from sklearn.linear_model import LinearRegression
regressor_MultiLinear = LinearRegression()
regressor_MultiLinear.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred = regressor_MultiLinear.predict(x_test)
```

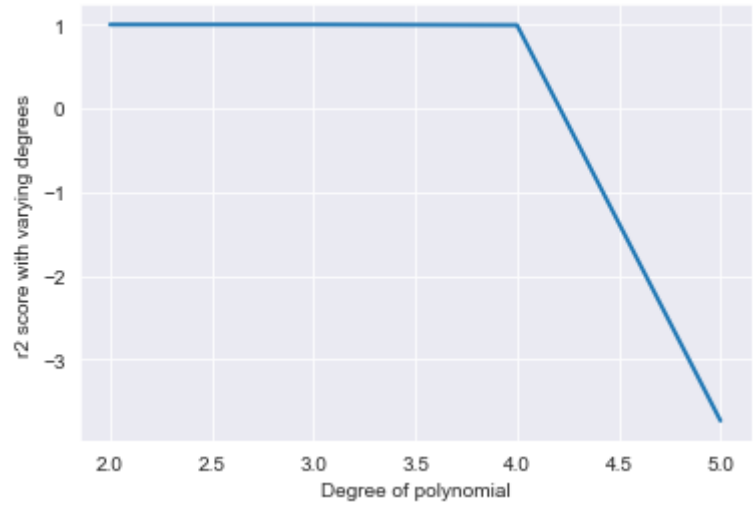
```
from sklearn.metrics import r2_score
r2_MultiLinear = r2_score(y_test,y_pred)
print(r2_MultiLinear)
```

```
0.9999863025192613
```

使用多项式回归进行预测

首先寻找最优的多项式次数，代码如下：

```
from sklearn.preprocessing import PolynomialFeatures
sns.set_style('darkgrid')
scores_list = []
pRange = range(2,6)
for i in pRange :
    poly_reg = PolynomialFeatures(degree=i)
    x_poly = poly_reg.fit_transform(x_train)
    poly_regressor = LinearRegression()
    poly_regressor.fit(x_poly,y_train)
    y_pred = poly_regressor.predict(poly_reg.fit_transform(x_test))
    scores_list.append(r2_score(y_test,y_pred))
plt.plot(pRange,scores_list,linewidth=2)
plt.xlabel('Degree of polynomial')
plt.ylabel('r2 score with varying degrees')
plt.show()
```



利用最优的多项式次数2进行拟合，代码如下：

```
poly_reg = PolynomialFeatures(degree=2)
x_poly = poly_reg.fit_transform(x_train)
poly_regressor = LinearRegression()
poly_regressor.fit(x_poly,y_train)
y_pred = poly_regressor.predict(poly_reg.fit_transform(x_test))
r2_poly = r2_score(y_test,y_pred)
print(r2_poly)
```

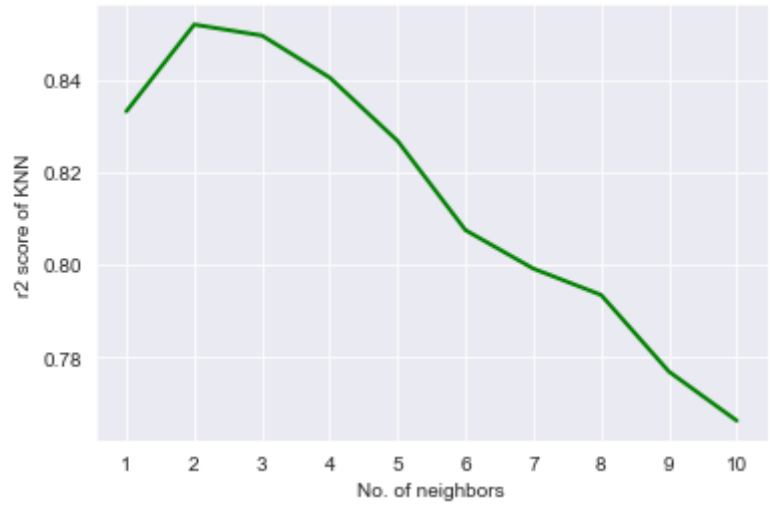
```
0.999986386077783
```

最终得到的准确率是0.999986386077783。

K-近邻回归

首先分析得到最优的邻居数，代码如下：

```
from sklearn.neighbors import KNeighborsRegressor
knnRange = range(1,11,1)
scores_list = []
for i in knnRange:
    regressor_knn = KNeighborsRegressor(n_neighbors=i)
    regressor_knn.fit(x_train,y_train)
    y_pred = regressor_knn.predict(x_test)
    scores_list.append(r2_score(y_test,y_pred))
plt.plot(knnRange,scores_list,linewidth=2,color='green')
plt.xticks(knnRange)
plt.xlabel('No. of neighbors')
plt.ylabel('r2 score of KNN')
plt.show()
```



由此可以看出，最优的邻居数为2，利用该邻居数进行预测分析，代码如下：

```
regressor_knn = KNeighborsRegressor(n_neighbors=7)
regressor_knn.fit(x_train,y_train)
y_pred = regressor_knn.predict(x_test)
r2_knn = r2_score(y_test,y_pred)
print(r2_knn)
```

```
0.7991214215403432
```

最终得到的准确率是0.7991214215403432

决策树回归

首先在训练集上训练决策树模型，代码如下，

```
from sklearn.tree import DecisionTreeRegressor
regressor_Tree = DecisionTreeRegressor(random_state=0)
regressor_Tree.fit(x_train,y_train)
```

```
DecisionTreeRegressor(random_state=0)
```

预测测试集上的结果，代码如下，

```
y_pred = regressor_Tree.predict(x_test)
```

计算分数，代码如下：

```
r2_tree = r2_score(y_test,y_pred)
print(r2_tree)
```

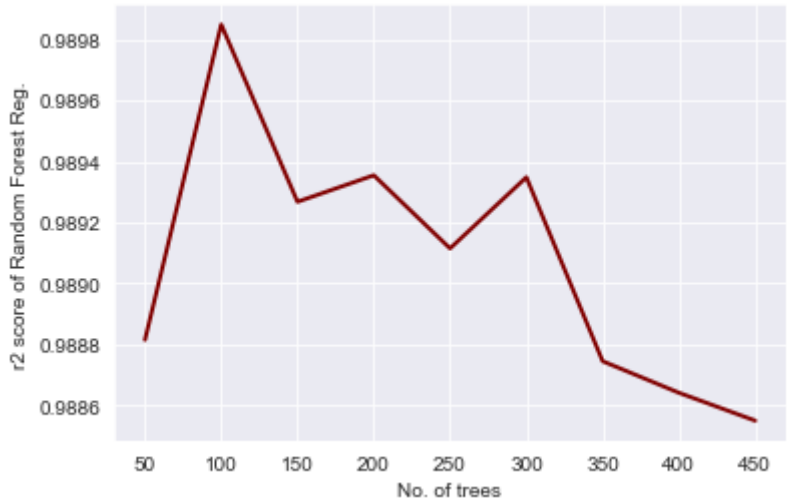
```
0.9288927448327867
```

得到的分数为0.9288927448327867

随机森林回归

首先寻找最优的树木的数量，代码如下：

```
from sklearn.ensemble import RandomForestRegressor
forestRange=range(50,500,50)
scores_list=[]
for i in forestRange:
    regressor_Forest = RandomForestRegressor(n_estimators=i,random_state=0)
    regressor_Forest.fit(x_train,y_train)
    y_pred = regressor_Forest.predict(x_test)
    scores_list.append(r2_score(y_test,y_pred))
plt.plot(forestRange,scores_list,linewidth=2,color='maroon')
plt.xticks(forestRange)
plt.xlabel('No. of trees')
plt.ylabel('r2 score of Random Forest Reg.')
plt.show()
```



由此得出，100是最优的树木数量

下面在训练集上训练回归模型

```
regressor_Forest = RandomForestRegressor(n_estimators=100,random_state=0)
regressor_Forest.fit(x_train,y_train)
y_pred = regressor_Forest.predict(x_test)
r2_forest = r2_score(y_test,y_pred)
print(r2_forest)
```

```
0.9898484041369164
```

使用SVM进行预测

首先对数据进行归一化，代码如下：

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(np.reshape(y_train,(len(y_train),1)))
y_test = sc_y.transform(np.reshape(y_test,(len(y_test),1)))
```

在训练集上训练模型，代码如下

```
from sklearn.svm import SVR
regressor_SVR = SVR(kernel='linear')
regressor_SVR.fit(x_train,y_train)
```

```
C:\anaconda\lib\site-packages\sklearn\utils\validation.py:72:
DataConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
SVR(kernel='linear')
```

在测试集上预测分数，代码如下：

```
r2_linearSVR = r2_score(y_test,y_pred)
print(r2_linearSVR)
```

```
0.277058090568169
```

可以看出，得到的结果很差，因此我们使用非线性的svm进行拟合，代码如下：

```
from sklearn.svm import SVR
regressor_NonLinearSVR = SVR(kernel='rbf')
regressor_NonLinearSVR.fit(x_train,y_train)
y_pred = regressor_NonLinearSVR.predict(x_test)

r2_NonlinearSVR = r2_score(y_test,y_pred)
print(r2_NonlinearSVR)
```

```
C:\anaconda\lib\site-packages\sklearn\utils\validation.py:72:
DataConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
0.6979055477850219
```

最终的结果是0.6979055477850219

由此可以看出，最终得到最好的模型分别是线性回归，多项式回归与随机森林的方法。在统计销售数据时，其他销售数据统计起来很繁琐，因此我们提出剔除其他销售数据，利用剩下属性的数据来预测全球销售数据。

剔除其他销售数据后预测销售数据

线性回归方法

```
x = dataset.iloc[:,[2,3,4,5,6,7,8]].values
y = dataset.iloc[:,-1].values
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
```

```
regressor_MultiLinear = LinearRegression()
regressor_MultiLinear.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred = regressor_MultiLinear.predict(x_test)
```

```
r2_MultiLinear = r2_score(y_test,y_pred)
print(r2_MultiLinear)
```

```
0.9969335354997467
```

多项式回归

```
poly_reg = PolynomialFeatures(degree=2)
x_poly = poly_reg.fit_transform(x_train)
poly_regressor = LinearRegression()
poly_regressor.fit(x_poly,y_train)
y_pred = poly_regressor.predict(poly_reg.fit_transform(x_test))
r2_poly = r2_score(y_test,y_pred)
print(r2_poly)
```

```
0.9947849824411171
```

决策树回归

```
regressor_Forest = RandomForestRegressor(n_estimators=100,random_state=0)
regressor_Forest.fit(x_train,y_train)
y_pred = regressor_Forest.predict(x_test)
r2_forest = r2_score(y_test,y_pred)
print(r2_forest)
```

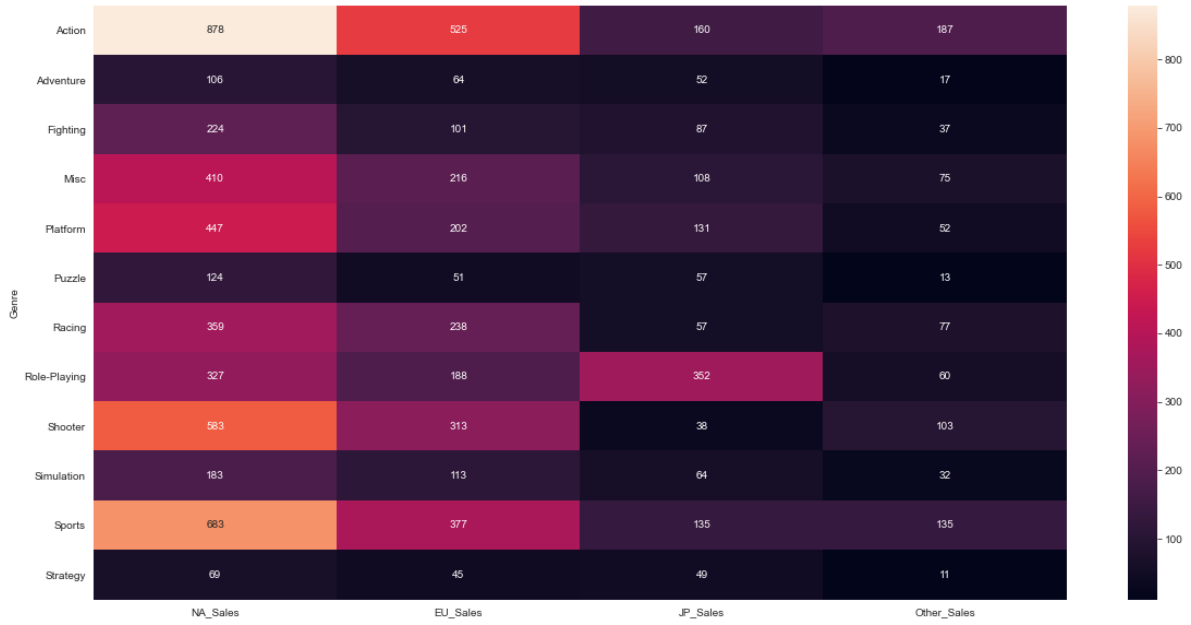
```
0.988983428091243
```

我们可以看到，在剔除其他销售数据之后，我们的模型依旧可以取得较好的结果。

可视化应用

```
sales=data[['Genre', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
sales = sales.groupby('Genre').sum()
plt.figure(figsize=(20,10))
sns.heatmap(sales,annot=True,fmt= '.0f')
```

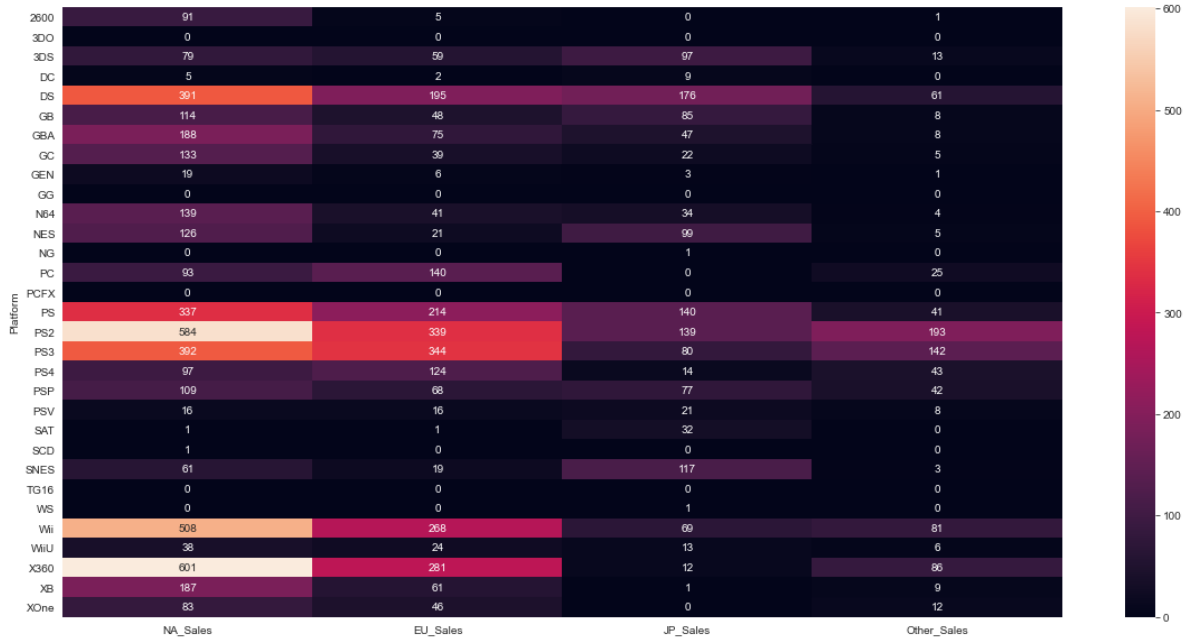
```
<AxesSubplot:ylabel='Genre'>
```



可以看出，在美国销售额度最好的类型是action与shooting，并且action远远高于其他类型的游戏。而在欧洲地区action的销售额同样远远超过了其他类型的游戏。而在日本地区，Role-playing类型的游戏远远超出其他类型的游戏。

```
sales=data[['Platform', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
sales = sales.groupby('Platform').sum()
plt.figure(figsize=(20,10))
sns.heatmap(sales,annot=True,fmt= '.0f')
```

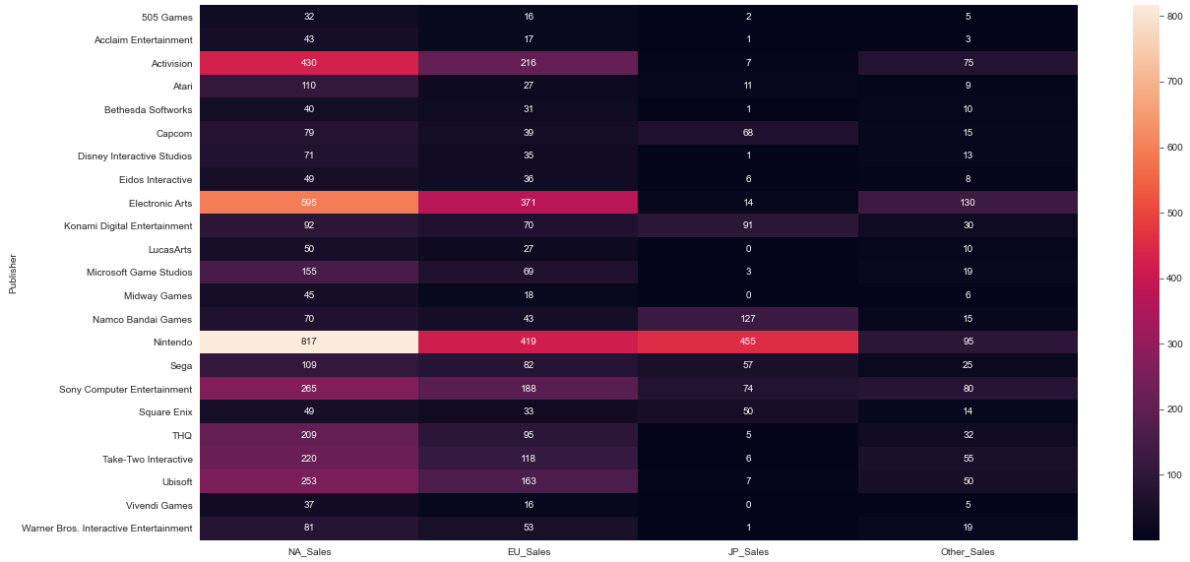
```
<AxesSubplot:ylabel='Platform'>
```



可以看出，在美国地区，销售额最高的平台是X360、Wii与PS系列的平台。而在欧洲地区，销售额度最高的平台是PS系列的平台。在日本销售额度最高的平台是DS。

```
sales=data[['Publisher', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
sales = sales.groupby('Publisher').sum()
sales=sales[sales['NA_Sales']>20]
plt.figure(figsize=(20,10))
sns.heatmap(sales,annot=True,fmt= '.0f')
```

```
<AxesSubplot:ylabel='Publisher'>
```



我们可以看到，在三个地区，任天堂的销售数据都是最高的。而Electronic Arts与Activision在美国和欧洲的销售额也很高，但是在日本的地区的销售数据并不好。而Namco Bandai Games在日本的销售额很高，但是在美国与欧洲的数据并不好。

通过上述三点分析，我们可以看出每一个区域都有不同的游戏类型偏向，不同的游戏平台偏向与不同的游戏发行商偏向。因此，可以针对不同的游戏市场，进行游戏类型，发行商的适应。

代码地址

https://github.com/XiaomengFanmcislab/DataMining_3