Задание 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Загрузить данные
df = pd.read_csv("dataset.csv")
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27
...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00

16719 rows × 16 columns

```
# Изучить данные
df.columns
```

```
Index(['Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher',
'NA_Sales',
       'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales',
'Critic_Score',
       'Critic_Count', 'User_Score', 'User_Count', 'Developer', 'Rating'],
      dtype='object')
```

```
df.head()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sa
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	2
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	1
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	1
	Pokemon			Role-			

```
df.tail()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Name             16717 non-null  object 
 1   Platform         16719 non-null  object 
 2   Year_of_Release  16450 non-null  float64
 3   Genre            16717 non-null  object 
 4   Publisher        16665 non-null  object 
 5   NA_Sales         16719 non-null  float64
 6   EU_Sales         16719 non-null  float64
 7   JP_Sales         16719 non-null  float64
 8   Other_Sales      16719 non-null  float64
 9   Global_Sales     16719 non-null  float64
 10  Critic_Score     8137 non-null   float64
 11  Critic_Count     8137 non-null   float64
 12  User_Score       10015 non-null  object 
 13  User_Count       7590 non-null   float64
 14  Developer        10096 non-null  object 
 15  Rating           9950 non-null   object 
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

```
df.describe()
```

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Gl
count	16450.000000	16719.000000	16719.000000	16719.000000	16719.000000	
mean	2006.487356	0.263330	0.145025	0.077602	0.047332	
std	5.878995	0.813514	0.503283	0.308818	0.186710	
min	1980.000000	0.000000	0.000000	0.000000	0.000000	
25%	2003.000000	0.000000	0.000000	0.000000	0.000000	
50%	2007.000000	0.080000	0.020000	0.000000	0.010000	
75%	2010.000000	0.240000	0.110000	0.040000	0.030000	
max	2020.000000	41.360000	28.960000	10.220000	10.570000	

```
df.nunique()
```

```
Name               11562
Platform              31
Year_of_Release       39
Genre                 12
Publisher            581
NA_Sales             402
EU_Sales             307
JP_Sales             244
Other_Sales          155
Global_Sales         629
Critic_Score          82
Critic_Count         106
User_Score            96
User_Count           888
Developer           1696
Rating                 8
dtype: int64
```

```
df.drop_duplicates()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27
...	...	...	...	...	...	...
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo Koei	0.00
16715	LMA Manager 2007	X360	2006.0	Sports	Codemasters	0.00
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Factory	0.00
16717	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo Koei	0.00

16719 rows × 16 columns

```
#обработка датасета
df=df.drop(['NA_Sales','EU_Sales', 'JP_Sales', 'Other_Sales'],axis=1)
df=df.dropna()
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	Global_Sales
0	Wii Sports	Wii	2006.0	Sports	Nintendo	82.53
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	35.52
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	32.77
6	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	29.80
7	Wii Play	Wii	2006.0	Misc	Nintendo	28.92
...	...	...	...	...	...	...
16667	E.T. The Extra-Terrestrial	GBA	2001.0	Action	NewKidCo	0.01

```
df.loc[:, 'Name']= pd.factorize(df['Name'].str.capitalize())[0]
df.loc[:, 'Platform']= pd.factorize(df['Platform'].str.capitalize())[0]
df.loc[:, 'Genre']= pd.factorize(df['Genre'].str.capitalize())[0]
df.loc[:, 'Publisher']= pd.factorize(df['Publisher'].str.capitalize())[0]
df.loc[:, 'Developer']= pd.factorize(df['Developer'].str.capitalize())[0]
df.loc[:, 'Rating']= pd.factorize(df['Rating'].str.capitalize())[0]
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	Global_Sales	Criti
0	0	0	2006.0	0	0	82.53	
2	1	0	2008.0	1	0	35.52	
3	2	0	2009.0	0	0	32.77	
6	3	1	2006.0	2	0	29.80	
7	4	0	2006.0	3	0	28.92	
...	...	...	...	...	...	...	
16667	4374	13	2001.0	4	261	0.01	
16677	359	13	2002.0	7	29	0.01	
16696	780	9	2014.0	4	10	0.01	
16700	4375	9	2011.0	6	119	0.01	
16706	4376	9	2011.0	11	60	0.01	

6825 rows × 12 columns

```
#проверка на выбросы
mean = df['Year_of_Release'].mean()
std = df['Year_of_Release'].std()

z_scores = (df['Year_of_Release'] - mean) / std

outliers = df[z_scores > 3]
outliers_index = outliers.index
df.drop(outliers_index, inplace=True)
```

```
#проверка на выбросы методом Z-score
mean = df['Global_Sales'].mean()
std = df['Global_Sales'].std()

z_scores = (df['Global_Sales'] - mean) / std

outliers = df[z_scores > 3]

outliers_index = outliers.index
df.drop(outliers_index, inplace=True)



mean = df['Critic_Count'].mean()
std = df['Critic_Count'].std()

z_scores = (df['Critic_Count'] - mean) / std

outliers = df[z_scores > 3]

outliers_index = outliers.index
df.drop(outliers_index, inplace=True)



mean = df['User_Count'].mean()
std = df['User_Count'].std()

z_scores = (df['User_Count'] - mean) / std

outliers = df[z_scores > 3]

outliers_index = outliers.index
df.drop(outliers_index, inplace=True)


mean = df['Critic_Score'].mean()
std = df['Critic_Score'].std()

z_scores = (df['Critic_Score'] - mean) / std

outliers = df[z_scores > 3]

outliers_index = outliers.index
df.drop(outliers_index, inplace=True)
```

```
df['User_Score'] = df['User_Score'].astype(float)

# Проверяем тип данных столбца User_Score
print(df.dtypes['User_Score'])
```

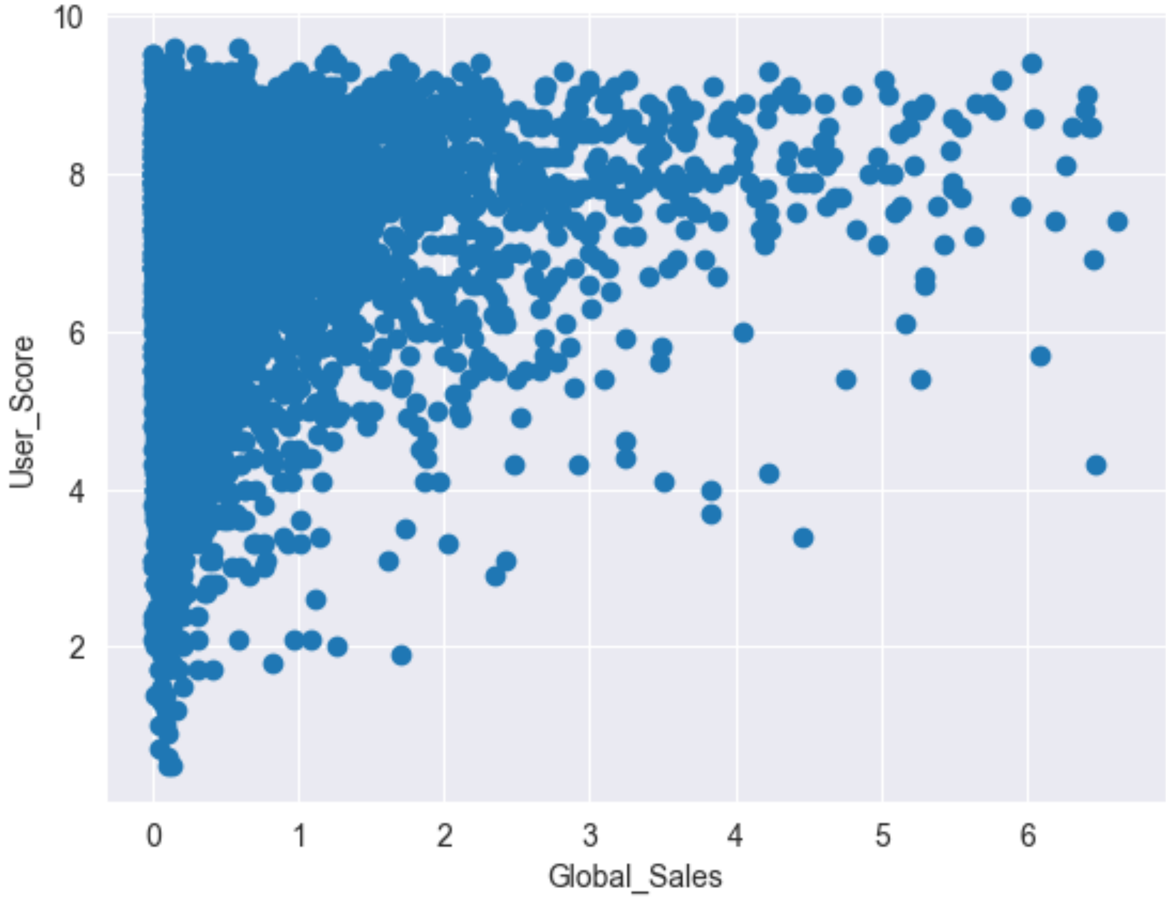
```
float64
```

```
mean = df['User_Score'].mean()
std = df['User_Score'].std()

z_scores = (df['User_Score'] - mean) / std

outliers = df[z_scores > 3]

outliers_index = outliers.index
df.drop(outliers_index, inplace=True)
```

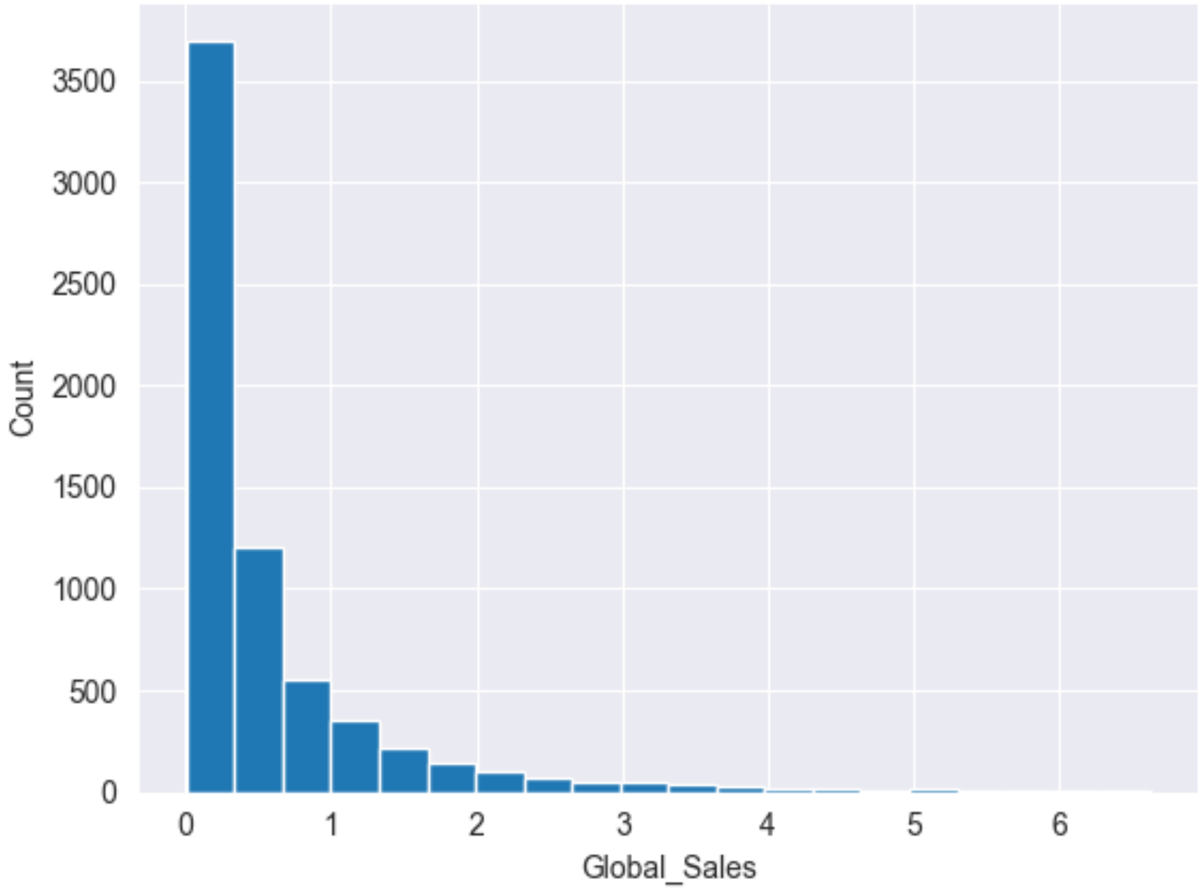
```
df.columns
```

```
Index(['Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher',
       'Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Score',
       'User_Count', 'Developer', 'Rating'],
      dtype='object')
```

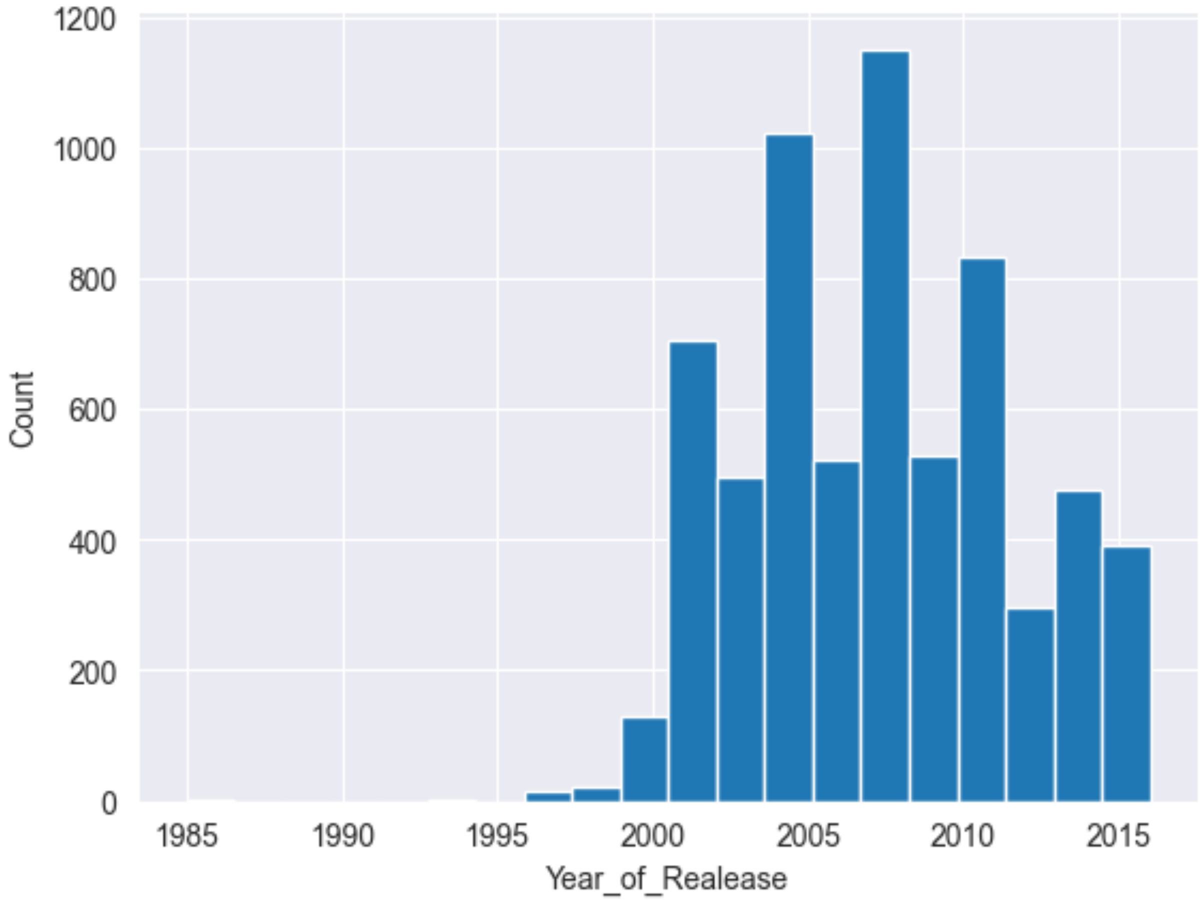
```
plt.scatter(df['Global_Sales'], df['User_Score'])
plt.xlabel('Global_Sales')
plt.ylabel('User_Score')
plt.show()
```



```
plt.hist(df['Global_Sales'], bins=20)
plt.xlabel('Global_Sales')
plt.ylabel('Count')
plt.show()
```

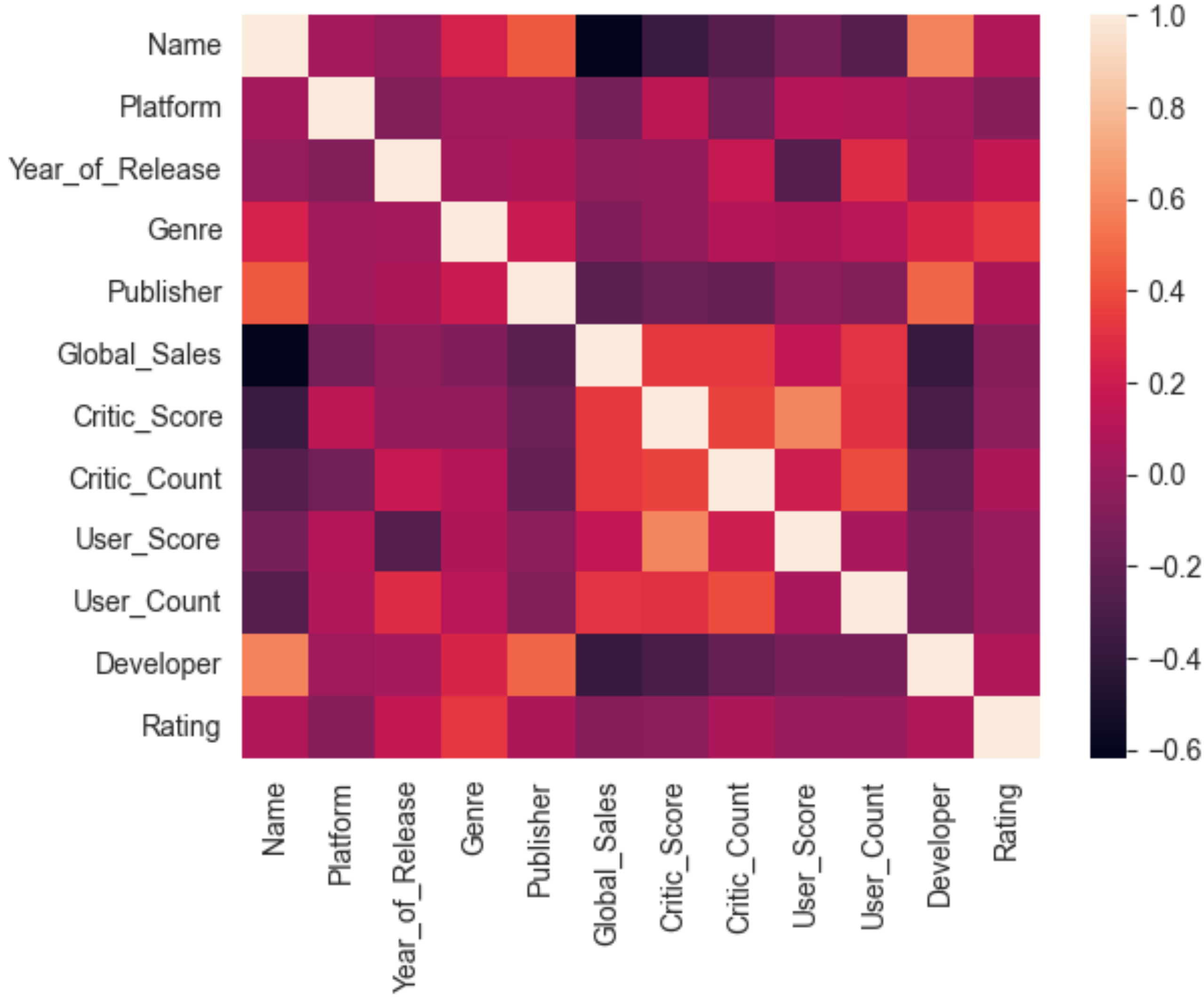


```
plt.hist(df['Year_of_Release'], bins=20)
plt.xlabel('Year_of_Realease')
plt.ylabel('Count')
plt.show()
```



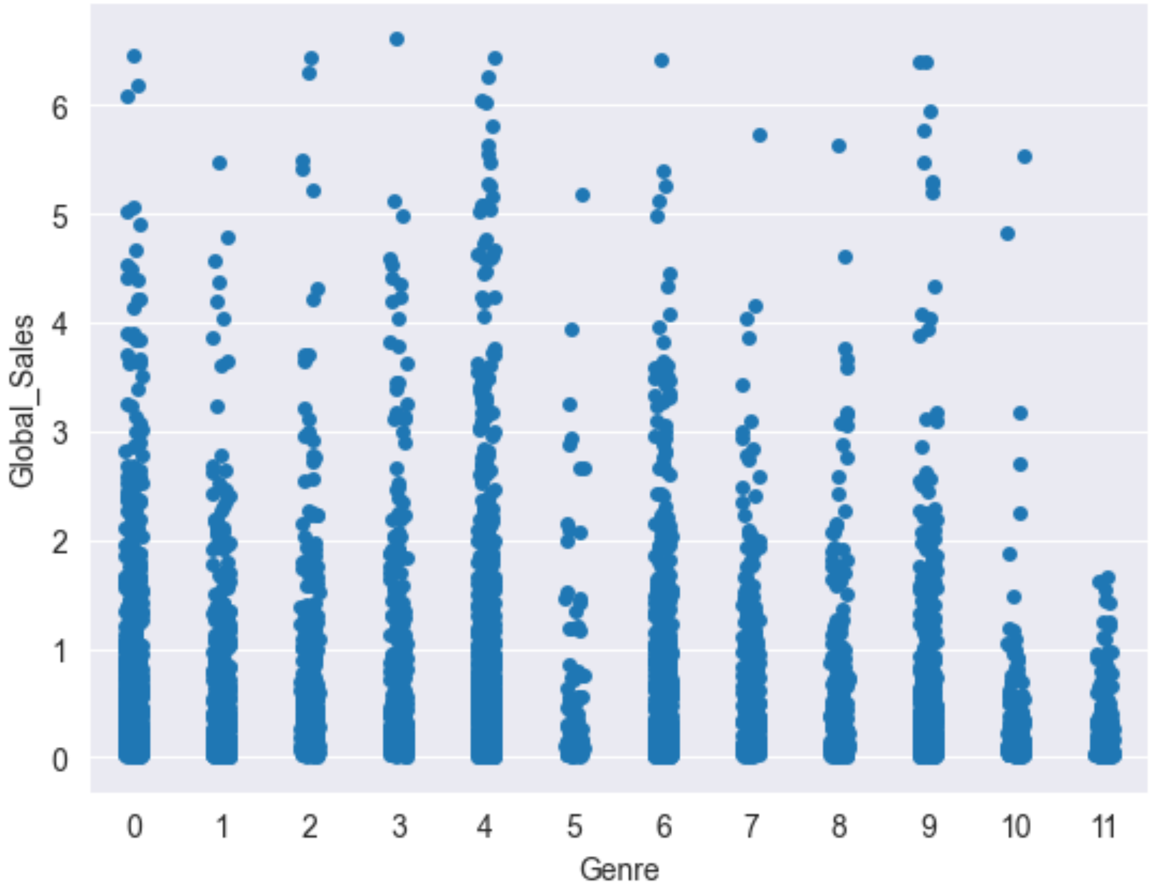
```
import seaborn as sns
corr = df.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

<Axes: >



Как видно из корреляционной матрицы зависимость между переменными Developer и Publisher равна 0.8

```
sns.stripplot(x='Genre', y='Global_Sales', data=df)
plt.show()
```



задание 2

```
y = df['Critic_Score']
X = df.drop(['Critic_Score'], axis=1)
```

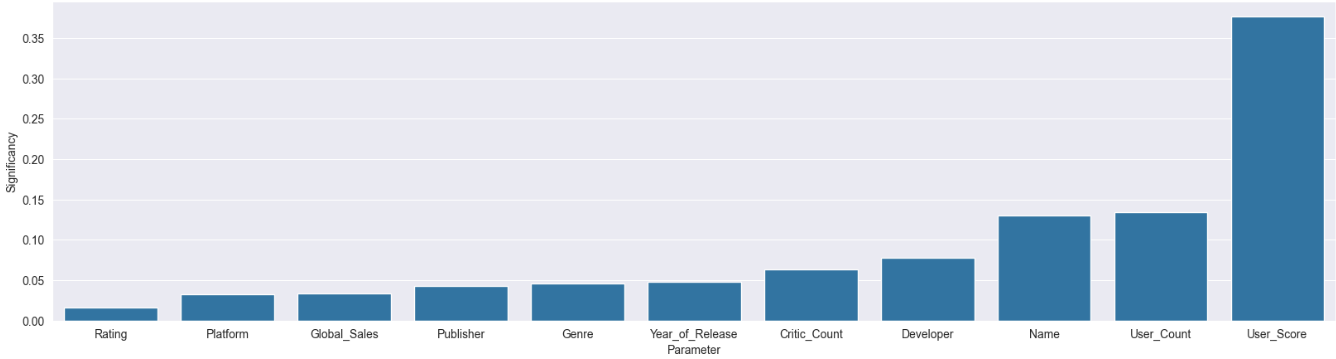
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
train_score = [model.score(X_train, y_train)]
test_score = [model.score(X_test, y_test)]

train_error_MSE = [mean_squared_error(y_train_pred, y_train) ** 0.5]
test_error_MSE = [mean_squared_error(y_pred, y_test) ** 0.5]

train_error_MAE = [mean_absolute_error(y_train_pred, y_train) ]
test_error_MAE = [mean_absolute_error(y_pred, y_test) ]

train_error_R2 = [r2_score(y_train_pred, y_train) ]
test_error_R2 = [r2_score(y_pred, y_test) ]

print(f"train_error_MSE = {train_error_MSE[0]}")
print(f"test_error_MSE = {test_error_MSE[0]}")
print(f"train_error_MAE = {train_error_MAE[0]}")
print(f"test_error_MAE = {test_error_MAE[0]}")
print(f"train_error_R2 = {train_error_R2[0]}")
print(f"test_error_R2 = {test_error_R2[0]}")
```

```
train_error_MSE = 9.493034715376957
test_error_MSE = 9.710143845726934
train_error_MAE = 7.491528284470958
test_error_MAE = 7.5616096815091645
train_error_R2 = 0.09447766669192992
test_error_R2 = -0.015052959722918535
```

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(X_train, y_train)
dfg = pd.DataFrame(model.feature_importances_, index=X.columns, columns=['f']).

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
```

```
train_score += [model.score(X_train, y_train)]
test_score += [model.score(X_test, y_test)]

train_error_MSE += [mean_squared_error(y_train_pred, y_train) ** 0.5]
test_error_MSE += [mean_squared_error(y_pred, y_test) ** 0.5]

train_error_MAE += [mean_absolute_error(y_train_pred, y_train) ]
test_error_MAE += [mean_absolute_error(y_pred, y_test) ]

train_error_R2 += [r2_score(y_train_pred, y_train) ]
test_error_R2 += [r2_score(y_pred, y_test) ]

print(f"train_error_MSE = {train_error_MSE[1]}")
print(f"test_error_MSE = {test_error_MSE[1]}")
print(f"train_error_MAE = {train_error_MAE[1]}")
print(f"test_error_MAE = {test_error_MAE[1]}")
print(f"train_error_R2 = {train_error_R2[1]}")
print(f"test_error_R2 = {test_error_R2[1]}")

plt.figure(figsize=(20,5))
sns.barplot(x=dfg.index,y=dfg['f'])
plt.xlabel('Parameter')
plt.ylabel('Significancy')
plt.show()
```

```
train_error_MSE = 3.15798368457149
test_error_MSE = 8.71219811859323
train_error_MAE = 2.410813578608003
test_error_MAE = 6.6427445034116746
train_error_R2 = 0.9311678612419511
test_error_R2 = 0.32238860519838863
```

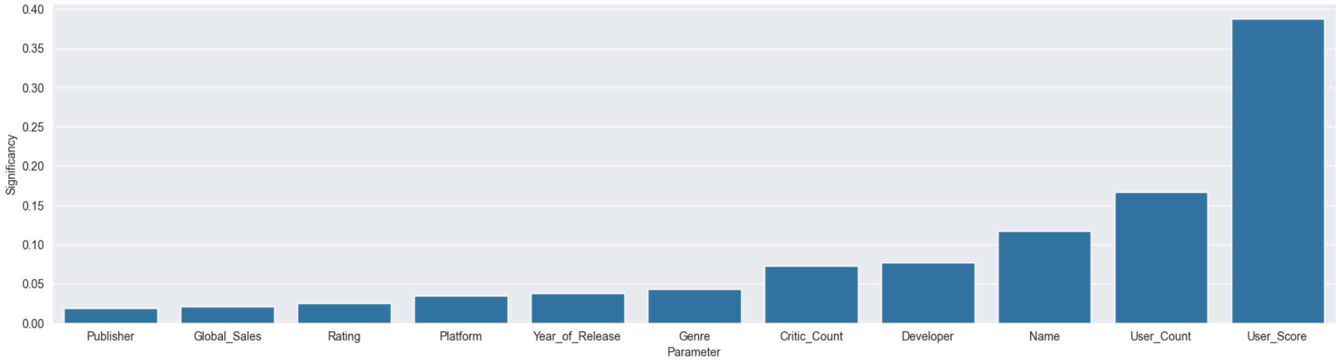


LGBMRegressor

```
import xgboost as xgb

import warnings
warnings.filterwarnings("ignore")

model = xgb.XGBRegressor(max_depth=3, verbose=-1)
model.fit(X_train, y_train)

dfg = pd.DataFrame(model.feature_importances_, index=X.columns, columns=['f']).

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
train_score += [model.score(X_train, y_train)]
test_score += [model.score(X_test, y_test)]
```

```
train_error_MSE += [mean_squared_error(y_train_pred, y_train) ** 0.5]
test_error_MSE += [mean_squared_error(y_pred, y_test) ** 0.5]

train_error_MAE += [mean_absolute_error(y_train_pred, y_train) ]
test_error_MAE += [mean_absolute_error(y_pred, y_test) ]

train_error_R2 += [r2_score(y_train_pred, y_train) ]
test_error_R2 += [r2_score(y_pred, y_test) ]

print(f"train_error_MSE = {train_error_MSE[2]}")
print(f"test_error_MSE = {test_error_MSE[2]}")
print(f"train_error_MAE = {train_error_MAE[2]}")
print(f"test_error_MAE = {test_error_MAE[2]}")
print(f"train_error_R2 = {train_error_R2[2]}")
print(f"test_error_R2 = {test_error_R2[2]}")

plt.figure(figsize=(20,5))
sns.barplot(x=dfg.index,y=dfg['f'])
plt.xlabel('Parameter')
plt.ylabel('Significancy')
plt.show()
```

```
train_error_MSE = 7.0369765134225855
test_error_MSE = 9.453131674128592
train_error_MAE = 5.439208770598584
test_error_MAE = 7.220780771008218
train_error_R2 = 0.607459100421541
test_error_R2 = 0.32677174068670856
```



DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(max_depth=3)
model.fit(X_train, y_train)
dfg = pd.DataFrame(model.feature_importances_, index=X.columns, columns=['f']).

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
train_score += [model.score(X_train, y_train)]
test_score += [model.score(X_test, y_test)]

train_error_MSE += [mean_squared_error(y_train_pred, y_train) ** 0.5]
test_error_MSE += [mean_squared_error(y_pred, y_test) ** 0.5]

train_error_MAE += [mean_absolute_error(y_train_pred, y_train) ]
```

```
test_error_MAE += [mean_absolute_error(y_pred, y_test) ]

train_error_R2 += [r2_score(y_train_pred, y_train) ]
test_error_R2 += [r2_score(y_pred, y_test) ]

print(f"train_error_MSE = {train_error_MSE[3]}")
print(f"test_error_MSE = {test_error_MSE[3]}")
print(f"train_error_MAE = {train_error_MAE[3]}")
print(f"test_error_MAE = {test_error_MAE[3]}")
print(f"train_error_R2 = {train_error_R2[3]}")
print(f"test_error_R2 = {test_error_R2[3]}")



plt.figure(figsize=(20,5))
sns.barplot(x=dfg.index,y=dfg['f'])
plt.xlabel('Parameter')
plt.ylabel('Significancy')
plt.show()
```

```
train_error_MSE = 10.219612153358877
test_error_MSE = 10.54554576530477
train_error_MAE = 8.070902669320507
test_error_MAE = 8.248758733958438
train_error_R2 = -0.22586492399017244
test_error_R2 = -0.27510624791536986
```

Lasso Regressor

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Lasso()

param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}
```

```
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_e
grid_search.fit(X_train, y_train)

best_alpha = grid_search.best_params_['alpha']

model = Lasso(alpha=best_alpha)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)

train_score += [model.score(X_train, y_train)]
test_score += [model.score(X_test, y_test)]

train_error_MSE += [mean_squared_error(y_train_pred, y_train) ** 0.5]
test_error_MSE += [mean_squared_error(y_pred, y_test) ** 0.5]

train_error_MAE += [mean_absolute_error(y_train_pred, y_train) ]
test_error_MAE += [mean_absolute_error(y_pred, y_test) ]

train_error_R2 += [r2_score(y_train_pred, y_train) ]
test_error_R2 += [r2_score(y_pred, y_test) ]

print(f"train_error_MSE = {train_error_MSE[4]}")
print(f"test_error_MSE = {test_error_MSE[4]}")
print(f"train_error_MAE = {train_error_MAE[4]}")
print(f"test_error_MAE = {test_error_MAE[4]}")
print(f"train_error_R2 = {train_error_R2[4]}")
print(f"test_error_R2 = {test_error_R2[4]}")
```

```
train_error_MSE = 9.522178243148321
test_error_MSE = 9.604575764932017
train_error_MAE = 7.481461890884281
test_error_MAE = 7.620447060019611
train_error_R2 = 0.021178601117710305
test_error_R2 = -0.033477441395928675
```

Выбор модели

```
pd.DataFrame({'Model':['linear', 'Random Forest', 'xgboost', 'Decision Tree', '
              'Train score':train_score,
              'Test score':test_score,
              'Train error MSE':train_error_MSE,
              'Test error Mse':test_error_MSE,
              'Train error MAE':train_error_MAE,
              'Test error MAE':test_error_MAE,
              'Train error R2':train_error_R2,
              'Test error R2':test_error_R2})
```

	Model	Train score	Test score	Train error MSE	Test error Mse	Train error MAE	Test error MAE	Train error R2
0	linear	0.524790	0.472015	9.493035	9.710144	7.491528	7.561610	0.094478
1	Random Forest	0.947411	0.574964	3.157984	8.712198	2.410814	6.642745	0.931168
2	LGBM	0.738876	0.499595	7.036977	9.453132	5.439209	7.220781	0.607459
3	Decision	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Выбираем Random Forest.

```
X = df.drop(['Critic_Score'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [2, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

model = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scorir
grid_search.fit(X_train, y_train)

print('Лучшие гиперпараметры:', grid_search.best_params_)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Лучшие гиперпараметры: {'max_depth': None, 'min_samples_leaf': 1, 'min_samp
```

```
model = RandomForestRegressor(**grid_search.best_params_, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)


train_error_MSE = mean_squared_error(y_train_pred, y_train) ** 0.5
test_error_MSE = mean_squared_error(y_pred, y_test) ** 0.5

train_error_MAE = mean_absolute_error(y_train_pred, y_train)
test_error_MAE = mean_absolute_error(y_pred, y_test)
train_error_R2 = r2_score(y_train_pred, y_train)
test_error_R2 = r2_score(y_pred, y_test)

print(f"train_error_MSE = {train_error_MSE}")
print(f"test_error_MSE = {test_error_MSE}")
print(f"train_error_MAE = {train_error_MAE}")
print(f"test_error_MAE = {test_error_MAE}")
print(f"train_error_R2 = {train_error_R2}")
print(f"test_error_R2 = {test_error_R2}")

scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')
print('Среднее качество модели на основе кросс-валидации:', scores.mean())
```

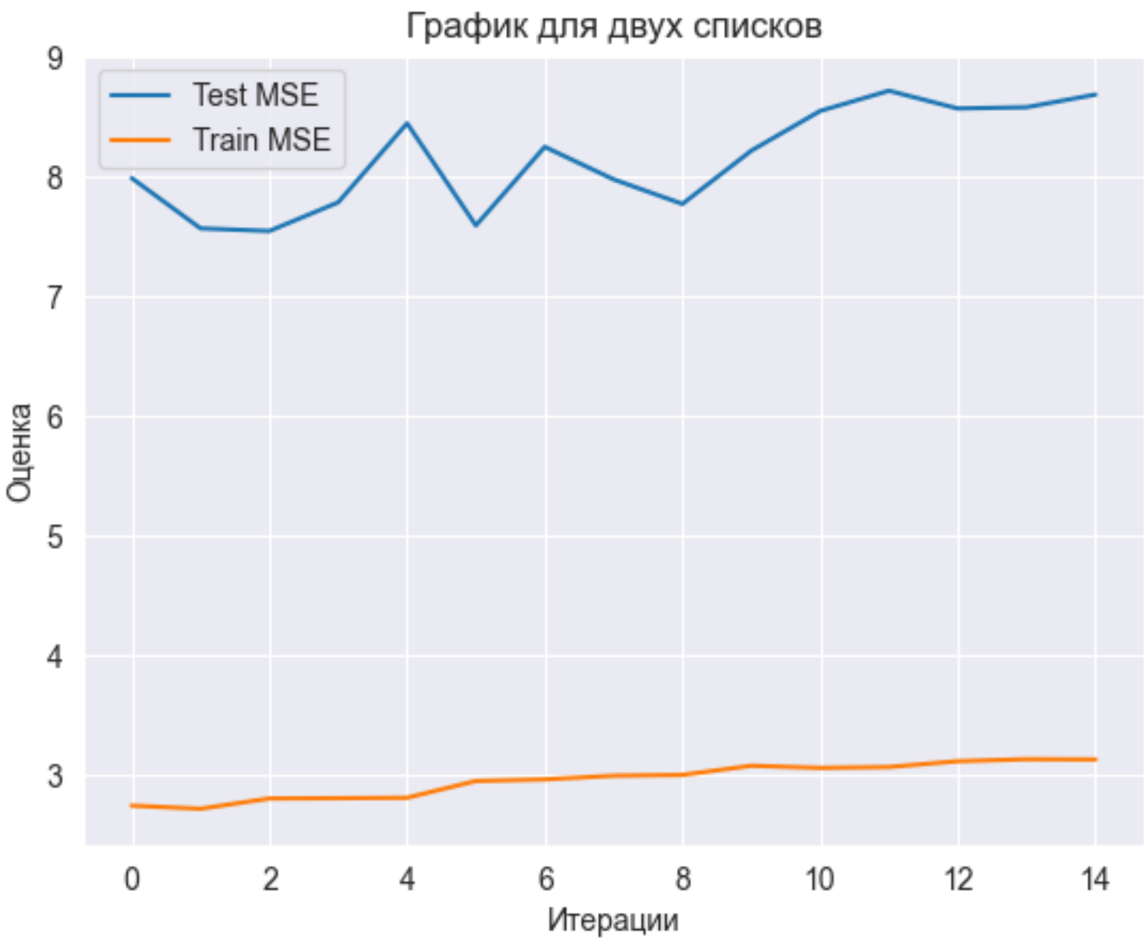
```
train_error_MSE = 3.135274088329641
test_error_MSE = 8.756957313109615
train_error_MAE = 2.412465389721221
test_error_MAE = 6.634150871872631
train_error_R2 = 0.9318079483229733
test_error_R2 = 0.3224183692610061
Среднее качество модели на основе кросс-валидации: 0.61774674695941
```

```
plt.figure(figsize=(20,5))
sns.barplot(x=dfg.index,y=dfg['f'])
plt.xlabel('Parameter')
plt.ylabel('Significancy')
plt.show()
```

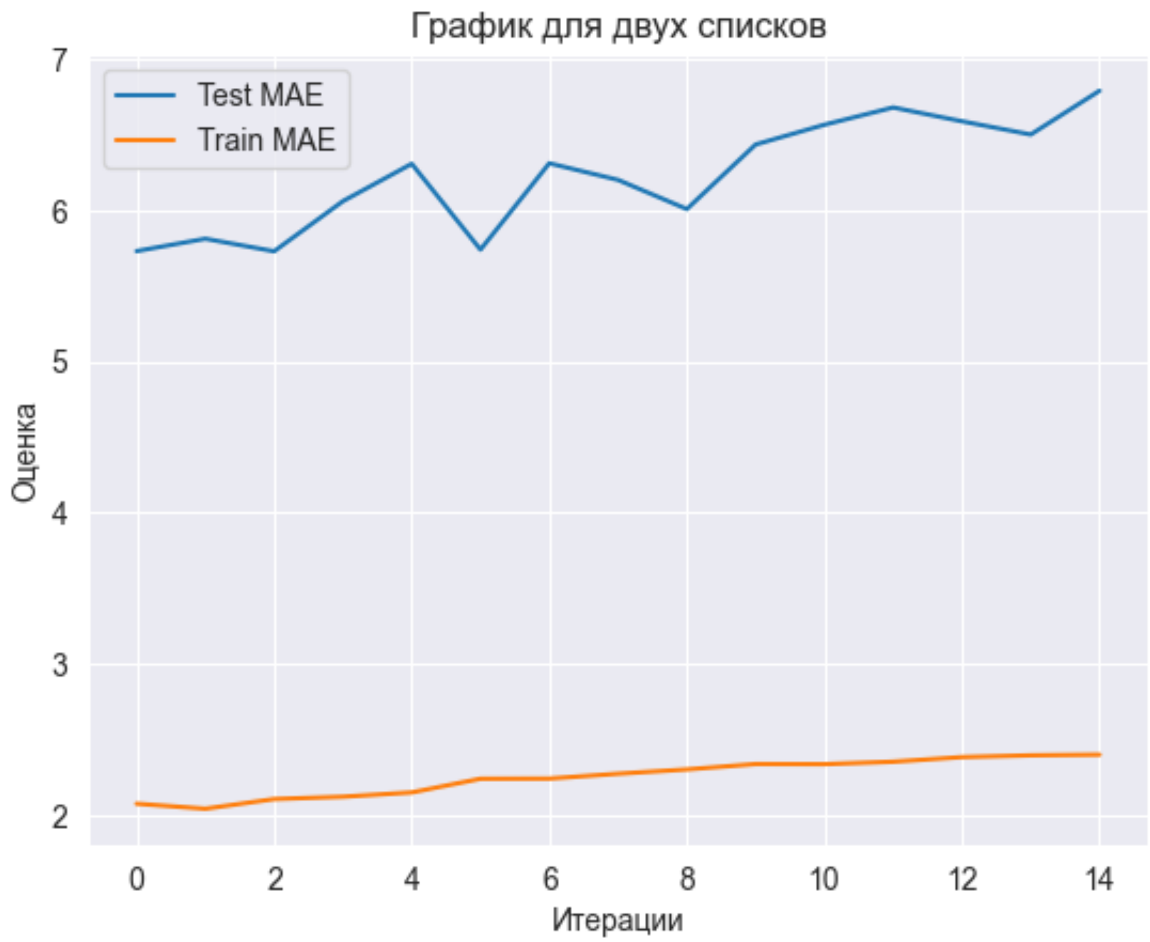
Кривая обучения/потерь. Оценка недообучения/переобучения

```
proc=30
train_error_MSE=[]
test_error_MSE=[]
train_error_MAE=[]
test_error_MAE=[]
train_error_R2=[]
test_error_R2=[]
while proc!=105:
    new_df=df.head(int(len(df)*proc/100))
    y = new_df['Critic_Score']
    X = new_df.drop(['Critic_Score'], axis=1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.fit_transform(X_test)

    params = {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
    model = RandomForestRegressor(**params)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)
    train_score += [model.score(X_train, y_train)]
    test_score += [model.score(X_test, y_test)]


    train_error_MSE += [mean_squared_error(y_train, y_train_pred) ** 0.5]
    test_error_MSE += [mean_squared_error(y_test, y_pred) ** 0.5]

    train_error_MAE += [mean_absolute_error(y_train, y_train_pred) ]
    test_error_MAE += [mean_absolute_error(y_test, y_pred) ]

    train_error_R2 += [r2_score(y_train, y_train_pred) ]
    test_error_R2 += [r2_score(y_test, y_pred) ]


    proc+=5
```

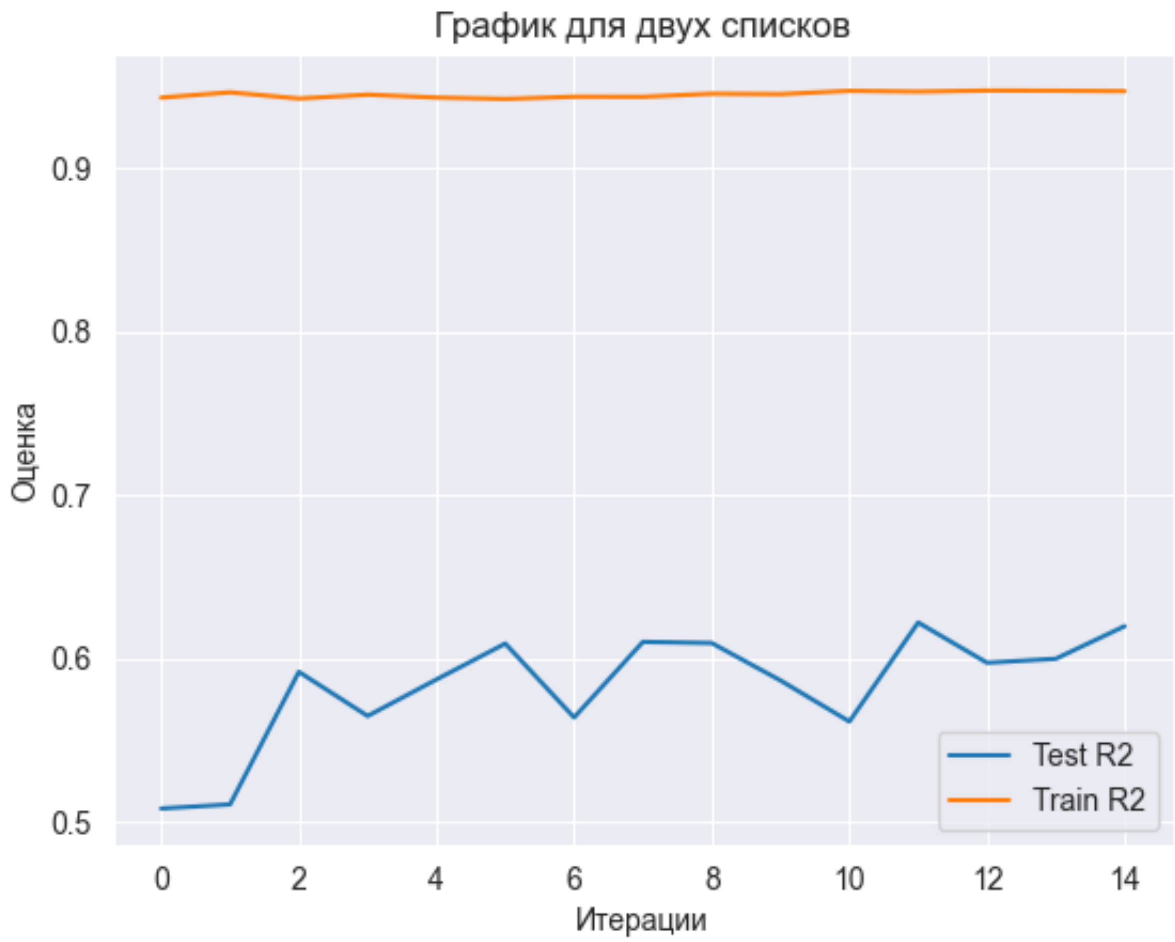
```
plt.plot(test_error_MSE, label='Test MSE')
plt.plot(train_error_MSE, label='Train MSE')

plt.title('График для двух списков')
plt.xlabel('Итерации')
plt.ylabel('Оценка')

plt.legend()

plt.show()
```



```
plt.plot(test_error_MAE, label='Test MAE')
plt.plot(train_error_MAE, label='Train MAE')

plt.title('График для двух списков')
plt.xlabel('Итерации')
plt.ylabel('Оценка')

plt.legend()

plt.show()
```



```
plt.plot(test_error_R2, label='Test R2')
plt.plot(train_error_R2, label='Train R2')

plt.title('График для двух списков')
plt.xlabel('Итерации')
plt.ylabel('Оценка')

plt.legend()

plt.show()
```



произошло переобучение

Задание 3

Задача классификации

```
from sklearn.model_selection import train_test_split
# Считаем средний рейтинг критиков
mean_critic_score = df['Critic_Score'].mean()

y = np.where(df['Critic_Score'] > mean_critic_score, 1, 0)
X = df.drop(['Critic_Score'], axis=1)


# Разделяем данные на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Рассмотрение 5ти моделей и 4х метрик

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, mean_squared_error

# Создаем пустой DataFrame для результатов
results =[]

# Обучаем модели
models = [
    LogisticRegression(max_iter=1000),
    RandomForestClassifier(),
    SVC(),
    GaussianNB(),
    GradientBoostingClassifier()
]

# Тестируем модели
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    # Создаем DataFrame с результатами для модели
    result = {
        'Model': model.__class__.__name__,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='weighted'),
        'Recall': recall_score(y_test, y_pred, average='weighted'),
        'F1-score': f1_score(y_test, y_pred, average='weighted'),
        'RMSE': mean_squared_error(y_test, y_pred) ** 0.5,
        'RMSE_train': mean_squared_error(y_train, y_pred_train) ** 0.5
    }

    results.append(result)

results = pd.DataFrame(results)
```

```
results
```

	Model	Accuracy	Precision	Recall	F1-score	RMSE	RMSE
0	LogisticRegression	0.755876	0.754636	0.755876	0.753297	0.494089	0
1	RandomForestClassifier	0.784685	0.783735	0.784685	0.783228	0.464020	0
2	SVC	0.778620	0.777995	0.778620	0.776282	0.470510	0
3	GaussianNB	0.703563	0.744744	0.703563	0.702557	0.544460	0
4	GradientBoostingClassifier	0.769522	0.768543	0.769522	0.767273	0.480081	0

Кросс-валидация

```
from sklearn.model_selection import GridSearchCV
results_2=[]
# Выбираем лучшую модель
best_model = RandomForestClassifier()

# Задаем сетку параметров
param_grid = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Производим grid search
grid_search = GridSearchCV(best_model, param_grid, cv=5, scoring='f1_weighted')
grid_search.fit(X_train, y_train)

# Выводим лучшие параметры
print('Best parameters:', grid_search.best_params_)

# Обучаем лучшую модель с оптимальными параметрами
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators'
```

```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)

# Тестируем лучшую модель
y_pred = best_model.predict(X_test)
y_pred_train = best_model.predict(X_train)
result_2 = {
        'Model': best_model.__class__.__name__,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='weighted'),
        'Recall': recall_score(y_test, y_pred, average='weighted'),
        'F1-score': f1_score(y_test, y_pred, average='weighted'),
        'RMSE': mean_squared_error(y_test, y_pred) ** 0.5,
        'RMSE_train': mean_squared_error(y_train, y_pred_train) ** 0.5
    }
results_2.append(result_2)
results_2 = pd.DataFrame(results_2)


results_2
```

	Model	Accuracy	Precision	Recall	F1-score	RMSE	RMSE_t
0	RandomForestClassifier	0.777104	0.77603	0.777104	0.775595	0.472119	

ЗАДАНИЕ 4 Задача уменьшения размерности

```
df
```

	Name	Platform	Year_of_Release	Genre	Publisher	Global_Sales	Criti
123	72	1	2005.0	3	0	6.62	
126	74	3	2013.0	0	7	6.47	
127	75	3	2012.0	4	5	6.45	
128	76	0	2010.0	2	0	6.44	
129	77	8	2001.0	6	1	6.43	
...	...	...	...	...	...	...	
16667	4374	13	2001.0	4	261	0.01	
16677	359	13	2002.0	7	29	0.01	
16696	780	9	2014.0	4	10	0.01	
16700	4375	9	2011.0	6	119	0.01	
16706	4376	9	2011.0	11	60	0.01	

6592 rows × 12 columns

```
from sklearn.decomposition import PCA
y = df['Critic_Score']
X = df.drop('Critic_Score', axis=1)

scaler = StandardScaler()
X = scaler.fit_transform(X)

pca = PCA(n_components=4).fit_transform(X)
pca2D_df = pd.DataFrame(data = pca, columns = ['x1', 'x2', 'x3','x4'])
pca2D_df
```

	x1	x2	x3	x4
0	5.139883	-0.459584	0.021632	-1.079790
1	5.314013	0.968068	2.609974	1.232583
2	6.285816	3.313890	0.595479	2.071458
3	6.163718	1.377798	-0.001513	-0.226534
4	6.709868	2.782741	-2.320194	2.003020
...	...	...	...	...
6587	-5.641435	-0.548230	0.697282	2.706379
6588	0.211580	-1.398072	-2.023180	0.625935
6589	1.081308	0.520759	0.571151	1.629179
6590	-4.051213	1.243184	0.381361	1.028244
6591	-3.174160	1.757009	-0.789726	-0.112838

6592 rows × 4 columns

Использование лучшей модели Задания 2

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

params = {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_
model = RandomForestRegressor(**params)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_train_pred = model.predict(X_train)
train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)


train_error_MSE = mean_squared_error(y_train, y_train_pred) ** 0.5
test_error_MSE = mean_squared_error(y_test, y_pred) ** 0.5

train_error_MAE = mean_absolute_error(y_train, y_train_pred)
test_error_MAE = mean_absolute_error(y_test, y_pred)

train_error_R2 = r2_score(y_train, y_train_pred)
test_error_R2 = r2_score(y_test, y_pred)
results_3=[]
result_3 = {
        'Model': best_model.__class__.__name__,
        'train_score': train_score,
        'test_score': test_score,
        'train_error_MSE': mean_squared_error(y_train, y_train_pred) ** 0.5,
        'test_error_MSE': mean_squared_error(y_test, y_pred) ** 0.5,
        'train_error_MAE': mean_absolute_error(y_train, y_train_pred),
        'test_error_MAE': mean_absolute_error(y_test, y_pred),
        'train_error_R2': r2_score(y_train, y_train_pred),
        'test_error_R2 ': r2_score(y_test, y_pred)
    }
results_3.append(result_3)
results_3 = pd.DataFrame(results_3)


results_3
```

	Model	train_score	test_score	train_error_MSE	test_error_
0	RandomForestClassifier	0.946872	0.617988	3.15303	8.490

значения немного улучшились

Использование лучшей модели Задания 3

```
y = np.where(df['Critic_Score'] > mean_critic_score, 1, 0)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

params = {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
model = RandomForestClassifier(**params)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_pred_train = model.predict(X_train)


results_4=pd.DataFrame([{
        'Model': best_model.__class__.__name__,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='weighted'),
        'Recall': recall_score(y_test, y_pred, average='weighted'),
        'F1-score': f1_score(y_test, y_pred, average='weighted'),
        'RMSE': mean_squared_error(y_test, y_pred) ** 0.5,
        'RMSE_train': mean_squared_error(y_train, y_pred_train) ** 0.5
        }])
```

```
results_4
```

	Model	Accuracy	Precision	Recall	F1-score	RMSE	RMSE_tr
0	RandomForestClassifier	0.796816	0.796172	0.796816	0.795887	0.45076	

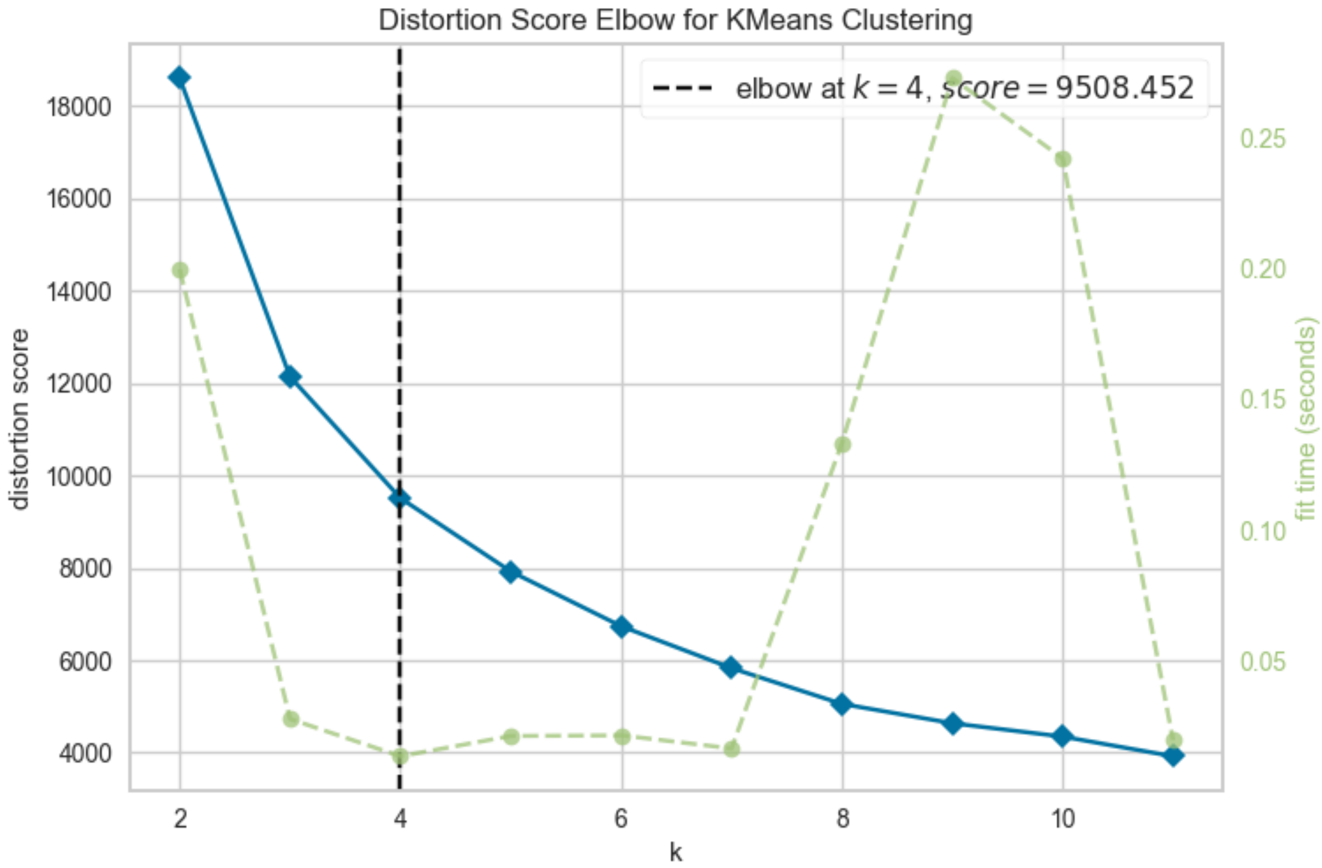
значения незначитильно улучшились

Задание 5 Решите задачу кластеризации, используя минимум 3 модели

```
X = df.drop('Critic_Score', axis=1)
X = StandardScaler().fit_transform(X)
X = PCA(n_components=2).fit_transform(X)
```

метод локтя

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
# Elbow method
elbow_visualizer = KElbowVisualizer(KMeans(), k=(2, 12))
elbow_visualizer.fit(X)
elbow_visualizer.show()
plt.show()
```



Обучение модели

```
from sklearn import cluster
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_b

silhouette_scores = []
calinski_scores = []
davies_scores = []
models = [
    cluster.KMeans(n_clusters=4),
    cluster.Birch(n_clusters=4),
    cluster.AgglomerativeClustering(n_clusters=4)]
labels=[]
for model in models:
    model.fit(X)
    labels.append(model.labels_)
    silhouette_scores.append(silhouette_score(X, model.labels_))
    calinski_scores.append(calinski_harabasz_score(X, model.labels_))
    davies_scores.append(davies_bouldin_score(X, model.labels_))
```

Определите метрики кластеризации

```
print('Silhouette scores:', silhouette_scores)
print('Calinski-Harabasz scores:', calinski_scores)
print('Davies-Bouldin scores:', davies_scores)
```

```
Silhouette scores: [0.34287332142318927, 0.292352375611249, 0.2864180123443
Calinski-Harabasz scores: [4626.568723367803, 2936.7076879784427, 3767.9959
Davies-Bouldin scores: [0.9232016486866692, 0.8781204632292621, 1.000908549
```

```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Clustering Visualization')



axs[0].scatter(X[:, 0], X[:, 1], c=labels[0], cmap='viridis', s=50)
axs[0].set_title('KMeans')

axs[1].scatter(X[:, 0], X[:, 1], c=labels[1], cmap='viridis', s=50)
axs[1].set_title('Birch')

axs[2].scatter(X[:, 0], X[:, 1], c=labels[2], cmap='viridis', s=50)
axs[2].set_title('Agglomerative Clustering')

plt.show()
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

6. Описание потенциального внедрения полученных моделей и результатов в проект "Маркетплейс ассетов"

В рамках работы была проведена обработка и анализ данных, полученных в результате продаж игр. Были выявлены зависимости между продажами игр и их оценками, а также построены модели регрессии и классификации для предсказания оценок товаров на основе их характеристик.

Для внедрения полученных моделей в проект необходимо выполнить следующие шаги:

1. Интеграция моделей в систему маркетплейса: необходимо разработать API-интерфейс для взаимодействия моделей с системой маркетплейса. Это позволит автоматизировать процесс предсказания оценок товаров и обеспечить его высокую точность.
2. Создание системы рекомендаций: на основе моделей регрессии и классификации можно создать систему рекомендаций, которая будет предлагать пользователям товары, наиболее подходящие под их потребности. Это позволит увеличить продажи товаров и улучшить качество обслуживания пользователей.
3. Анализ данных и мониторинг качества моделей: необходимо постоянно анализировать данные, поступающие в систему маркетплейса, и обновлять модели в соответствии с изменениями. Это позволит поддерживать высокое качество предсказаний и рекомендаций, а также своевременно реагировать на изменения рынка.
4. Создание нового умного функционала: на основе моделей машинного обучения можно создать новый умный функционал продукта/сервиса, например, систему прогнозирования спроса на товары.

Таким образом, внедрение полученных моделей и результатов в проект "Маркетплейс ассетов" позволит улучшить качество обслуживания пользователей, увеличить продажи товаров и создать новый умный функционал сервиса. Это обеспечит его дальнейшее развитие.