Бозоров Руслан Миралиевич, группа 20-1

Лабораторная работа № 3

**Цель работы**

Решить задачу предсказания рейтинга игр из Steam.

**Задание**

1. Решить задачу регрессии.
2. Провести первичный разведочный анализ данных и предобработку данных.
3. Сравните несколько моделей машинного обучения для вашей задачи из библиотеки sklearn
4. Написать нейронную сеть для задачи регрессии

**Код программы / ответ на вопрос и т.п.**

# Подключение необходимых библиотек

import pandas as pd

import seaborn as sns

# Просмотр датасета

data = pd.read\_csv('games-release-ALL.csv')

data.head()



# Ищем отсутствующие значения

data.isna().sum()

Unnamed: 0 0

game 0

link 0

release 0

peak\_players 0

positive\_reviews 0

negative\_reviews 0

total\_reviews 0

rating 0

dtype: int64

# Т.к. значения столбца total\_reviews, очевидно, некорректны, создаём новый столбец с отношением количества отзывов, а остальные столбцы с отзывами удаляем

data['negative\_reviews'] = data['negative\_reviews'].replace(r'\D','',regex=True) .astype(float)

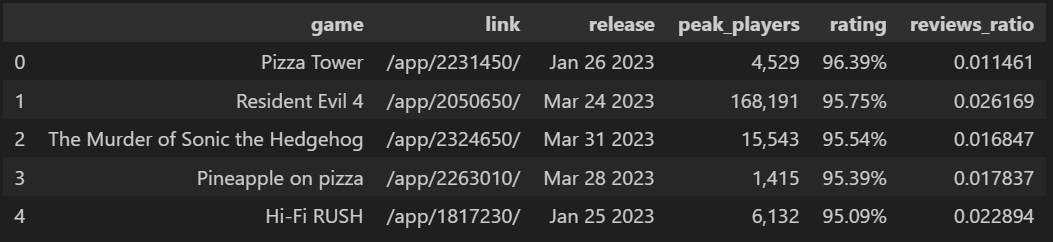
data['positive\_reviews'] = data['positive\_reviews'].replace(r'\D','',regex=True) .astype(float)

data = data[(data['negative\_reviews'] > 0) & (data['positive\_reviews'] > 0)]

data['reviews\_ratio'] = data['negative\_reviews'] / data['positive\_reviews']

data = data.drop(['Unnamed: 0', 'negative\_reviews', 'positive\_reviews', 'total\_reviews'], axis=1)

data.head()



# Перевод данных в numeric

data['peak\_players'] = data['peak\_players'].replace({',': ''}, regex=True)

# Convert the columns to numeric (optional, if needed)

data['peak\_players'] = data['peak\_players'].apply(pd.to\_numeric, errors='ignore')

# Удаляем столбцы с названиями и ссылками

data = data.drop(['game', 'link'], axis=1)

# Конвертируем даты

data['release'] = pd.to\_datetime(data['release'])

data['release\_year'] = data['release'].apply(lambda x: x.year)

data['release\_month'] = data['release'].apply(lambda x: x.month)

data['release\_day'] = data['release'].apply(lambda x: x.day)

data = data.drop('release', axis=1)

# Убираем символ %

data['rating'] = data['rating'].replace({'%': ''}, regex=True)

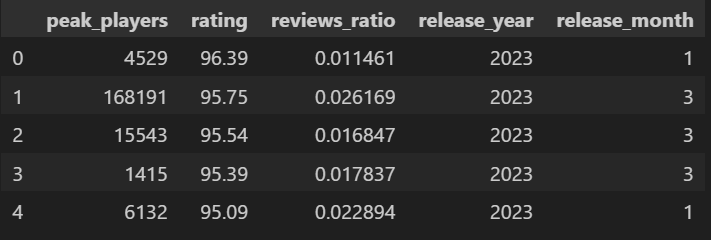
data['rating'] = data['rating'].apply(pd.to\_numeric, errors='ignore')

data = data.apply(pd.to\_numeric, errors='ignore')

# Удаляем столбец release\_day

data.drop('release\_day', axis=1, inplace=True)

data.head()



# Ищем выбросы в данных

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(12, 12))

plt.subplot(3, 2, 1)

sns.boxplot(data=data, x='peak\_players')

plt.subplot(3, 2, 2)

sns.boxplot(data=data, x='rating')

plt.subplot(3, 2, 3)

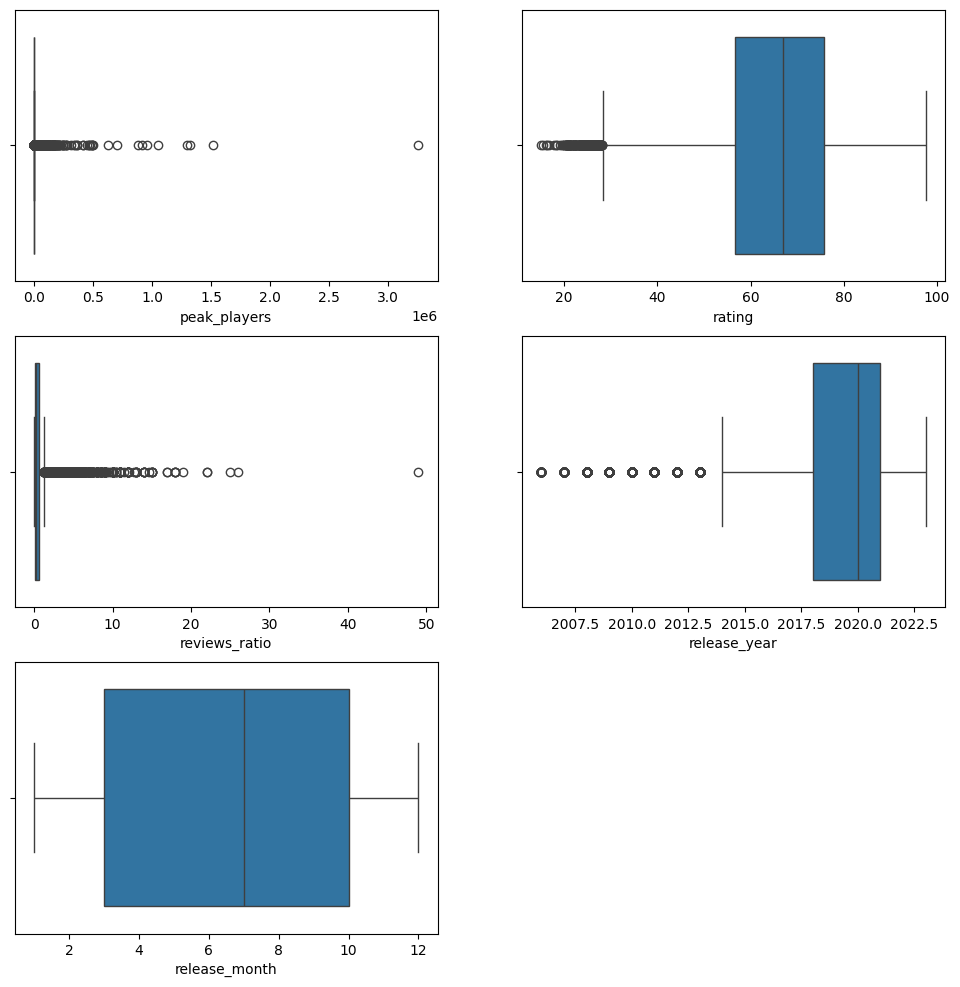
sns.boxplot(data=data, x='reviews\_ratio')

plt.subplot(3, 2, 4)

sns.boxplot(data=data, x='release\_year')

plt.subplot(3, 2, 5)

sns.boxplot(data=data, x='release\_month')



# Строим матрицу корреляции

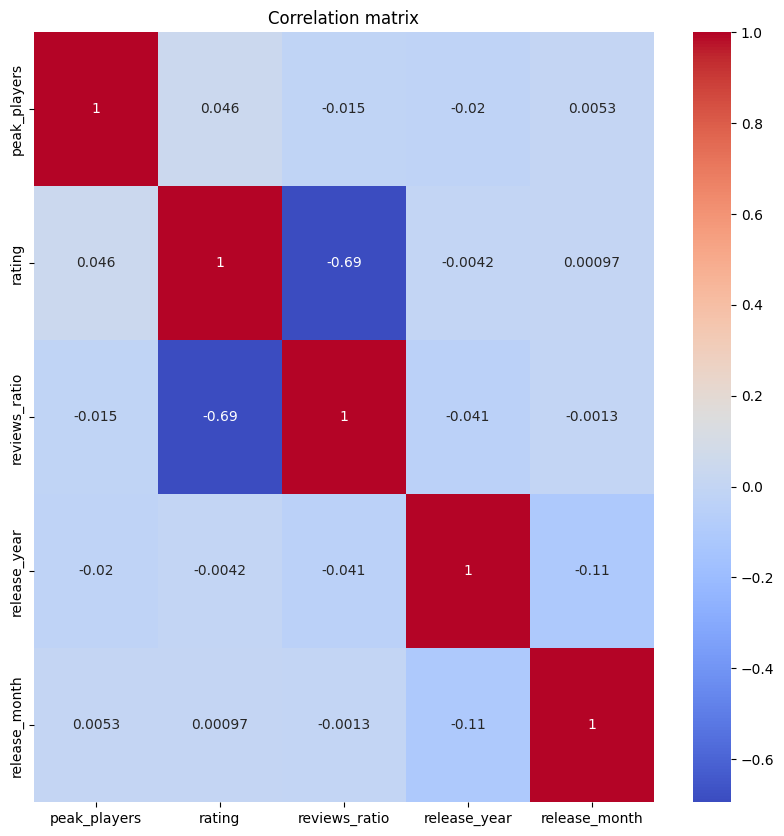
correlation\_matrix = data.corr()

plt.figure(figsize=(10, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm")

plt.title("Correlation matrix")

plt.show()



# Вычисление важности признаков с использованием mutual information на данных X\_train и y\_train

from sklearn.feature\_selection import mutual\_info\_regression

x = data.drop(['rating'], axis=1)

y = data['rating']

mutual\_info = mutual\_info\_regression(x, y, random\_state=42)

# Создание DataFrame для отображения результатов

feature\_importance\_df = pd.DataFrame({'Feature': x.columns, 'Importance': mutual\_info})

# Сортировка по важности

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=True)

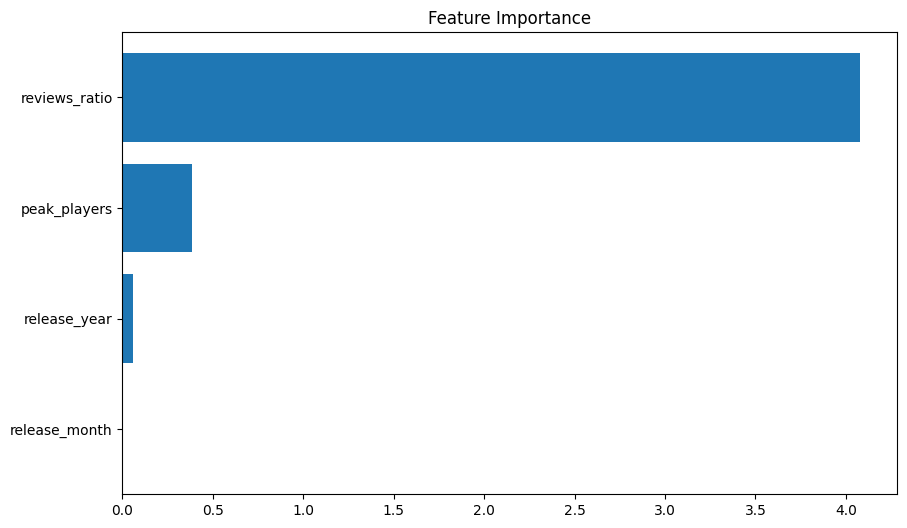
# Визуализация важности признаков

plt.figure(figsize=(10, 6))

plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'])

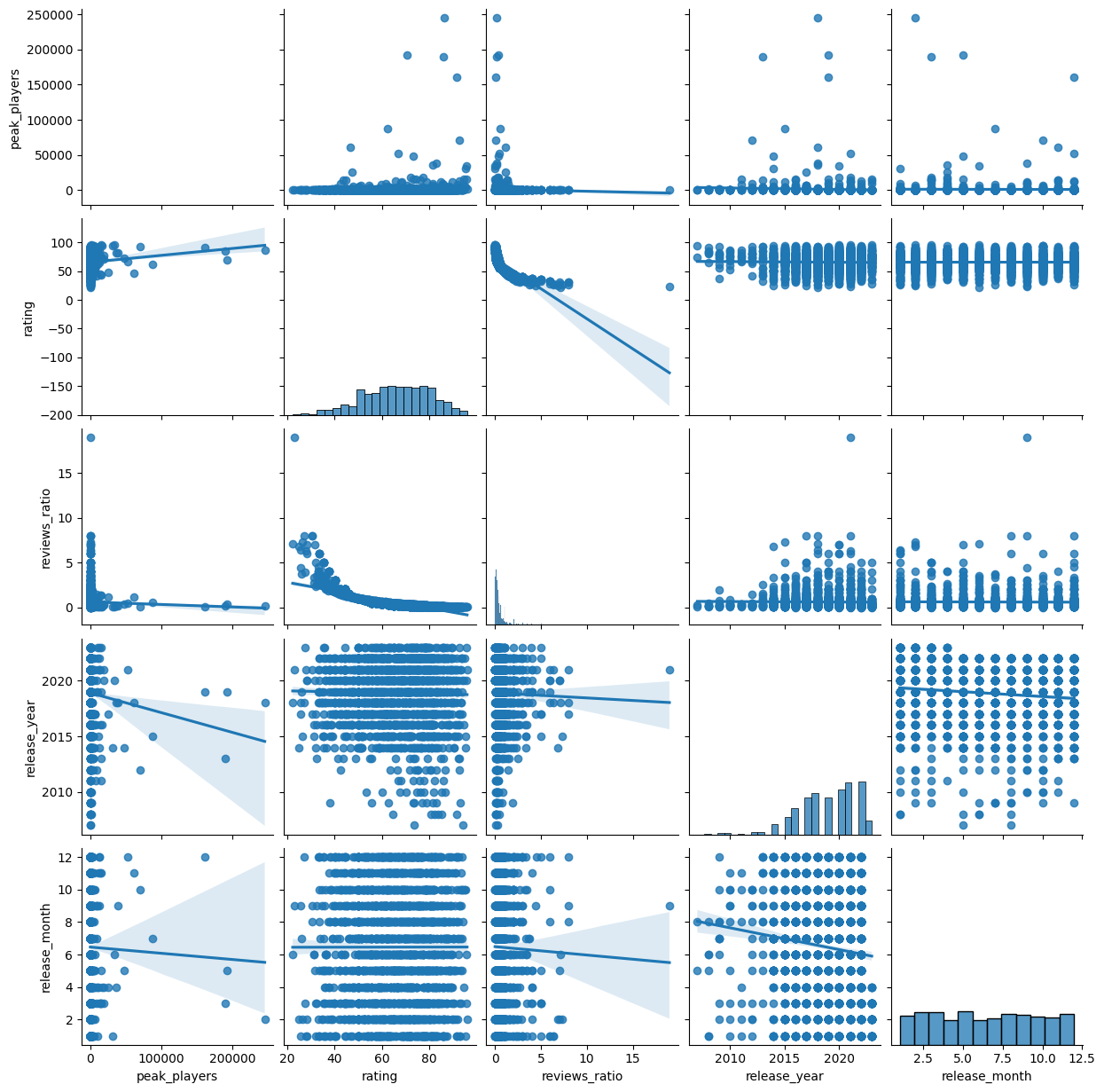
plt.title('Feature Importance')

plt.show()



# Вычисление зависимостей признаков друг от друга

sns.pairplot(data[2000:4000], kind="reg")



# Можно заметить, что с годами рейтинги игр в среднем стали ниже, а игры   
# выпущенные ближе к концу года имеют более высокие рейтинги

# Переводим данные года и месяца выпуска в one-hot векторы

data = pd.get\_dummies(data, columns=['release\_year', 'release\_month'], dtype=int)

# Создание обучающей и тестовой выборок, и их нормализация

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.preprocessing import StandardScaler

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=30)

x\_train\_scaled = x\_train.loc[:, ['peak\_players', 'reviews\_ratio']]

x\_test\_scaled = x\_test.loc[:, ['peak\_players', 'reviews\_ratio']]

scaler\_linear = StandardScaler().fit(x\_train\_scaled)

x\_train\_scaled = scaler\_linear.transform(x\_train\_scaled)

x\_test\_scaled = scaler\_linear.transform(x\_test\_scaled)

x\_train = np.concatenate((x\_train\_scaled, x\_train.drop(['peak\_players', 'reviews\_ratio'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['peak\_players', 'reviews\_ratio'], axis=1).to\_numpy()), axis=1)

# Создание нейросети

import torch

import torch.nn as nn

from torch.optim import Adam

from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

x\_train = torch.tensor(x\_train).float().to(device)

x\_test = torch.tensor(x\_test).float().to(device)

y\_train = torch.tensor(y\_train.to\_numpy()).float().to(device)

y\_test = torch.tensor(y\_test.to\_numpy()).float().to(device)

class NN(nn.Module):

    def \_\_init\_\_(self, input):

        super(NN, self).\_\_init\_\_()

        self.fc1 = nn.Sequential(nn.Linear(input, 256),

                                 nn.ReLU(),

                                 nn.Dropout(0.3),

                                 nn.Linear(256, 128),

                                 nn.ReLU(),

                                 nn.Dropout(0.3),

                                 nn.Linear(128, 1))

    def forward(self, x):

        x = self.fc1(x)

        return x

model\_nn = NN(len(x\_train[0]))

model\_nn.to(device)

optimizer = Adam(params=model\_nn.parameters(), lr=1e-3)

loss\_func = nn.MSELoss()

# Обучение нейросети

num\_iter = 500

losses = []

train\_loop = tqdm(range(num\_iter))

train\_loop.set\_description\_str("Model training")

for i in train\_loop:

    outputs = model\_nn(x\_train)

    loss = loss\_func(outputs, y\_train)

    losses.append(loss.item())

    train\_loop.set\_postfix\_str(f"Current error = {loss.item()}")

    loss.backward()

    optimizer.step()

    optimizer.zero\_grad()

**Результаты выполнения**

# Тест различных моделей

result = pd.DataFrame({'Model':[], 'Accuracy':[], 'MSE':[], 'MAE':[], 'R2':[]})

def test\_model(model, x\_train, y\_train, x\_test, y\_test):

    # Train the model

    model.fit(x\_train, y\_train)

    # Feed the scaled training set and get the predictions

    y\_hat = model.predict(x\_test)

    accuracy = model.score(x\_test, y\_test)

    test\_mse = mean\_squared\_error(y\_test, y\_hat)

    test\_mae = mean\_absolute\_error(y\_test, y\_hat)

    test\_r2 = r2\_score(y\_test,y\_hat)

    return accuracy, test\_mse, test\_mae, test\_r2

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.neural\_network import MLPRegressor

from sklearn.ensemble import GradientBoostingRegressor

models = []

linear = LinearRegression()

decision\_tree = DecisionTreeRegressor()

rendom\_forest = RandomForestRegressor()

svr = SVR()

k\_neighbors = KNeighborsRegressor()

mlp = MLPRegressor()

gradient\_boosting = GradientBoostingRegressor()

models.append(linear)

models.append(decision\_tree)

models.append(rendom\_forest)

models.append(svr)

models.append(k\_neighbors)

models.append(mlp)

models.append(gradient\_boosting)

for model in models:

    accuracy, test\_mse, test\_mae, test\_r2 = test\_model(model, x\_train, y\_train, x\_test, y\_test)

    result.loc[len(result.index)] = [str(model), accuracy, test\_mse, test\_mae, test\_r2]

result.sort\_values(by=['MSE', 'MAE'])



# Тест нейросети

results = model\_nn(x\_test).cpu().detach().numpy()

y\_real = y\_test.cpu().detach().numpy()

mse = mean\_squared\_error(y\_real, results)

mae = mean\_absolute\_error(y\_real, results)

r2 = r2\_score(y\_real, results)

print(f"mse =\t{mse}\nmae =\t{mae}\nr2 =\t{r2}")

mse = 222.562744140625

mae = 12.071578025817871

r2 = -0.10810351106660354

**Выводы**

В результате выполнения работы был проведён первичный разведочный анализ данных, в следствие которого были удалены некоторые столбцы изначального датасета.

Были добавлены собственные столбцы, например reviews\_ratio.

Были обучены и протестированы различные модели машинного обучения из библиотеки sklearn, такие как DecisionTreeRegressor, RandomForestRegressor, SVR, KNeighborsRegressor, MLPRegressor, GradientBoostingRegressor.

Наилучший результат показала модель GradientBoostingRegressor. Точность модели 98%, среднеквадратичная ошибка модели 3,66, что является отличным результатом.

Нейросеть показала плохие результаты, по сравнению с другими моделями машинного обучения.