Хаустов Игорь Сергеевич, группа 20-1

Лабораторная работа № 3

**Цель работы**

Решить задачу предсказания выгорания сотрудников.

**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 библиотек

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler

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

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

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

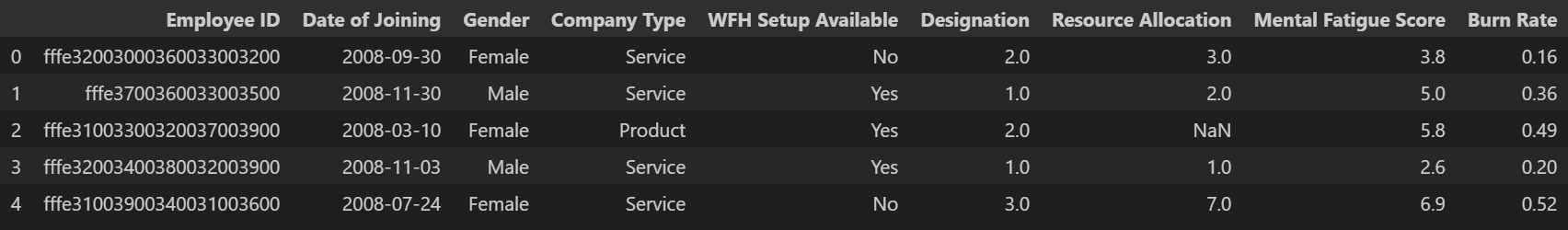
from sklearn.neural\_network import MLPRegressor

from sklearn.ensemble import GradientBoostingRegressor

# Подключение датасета

data = pd.read\_csv('Datasets/Burnout/train.csv')

data.head()



# Заменяем строковые данные на числовые и заполняем отсутствующие данные

def preprocess\_inputs(df):

    df = df.copy()

    # Employee ID non-correlati with their burnout

    df = df.drop('Employee ID', axis=1)

    # Need to drop rows where target value is missing

    missing\_target\_rows = df.loc[df['Burn Rate'].isna(), :].index

    df = df.drop(missing\_target\_rows, axis=0).reset\_index(drop=True)

    # Fill missing non-target values with mean values

    for column in ['Resource Allocation', 'Mental Fatigue Score']:

        df[column] = df[column].fillna(df[column].mean())

    # Date features

    df['Date of Joining'] = pd.to\_datetime(df['Date of Joining'])

    df['Month'] = df['Date of Joining'].apply(lambda x: x.month)

    df['Day'] = df['Date of Joining'].apply(lambda x: x.day)

    # Year is the same for every row (2008) so we can drop it

    df = df.drop('Date of Joining', axis=1)

    # Binary features extract

    df['Gender'] = df['Gender'].replace({'Female': 0, 'Male': 1})

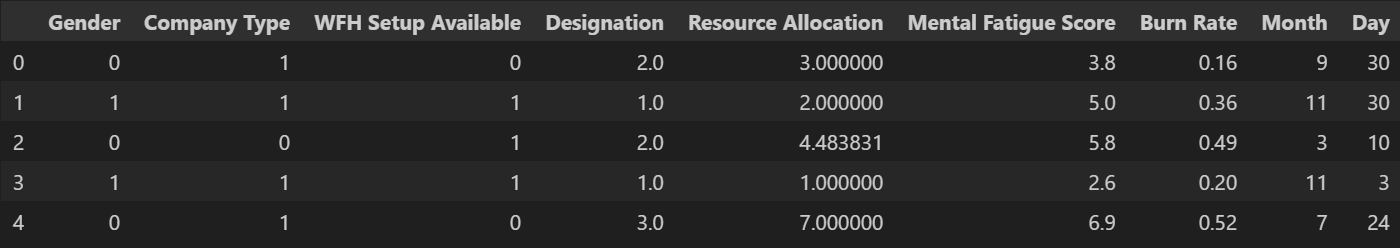
    df['Company Type'] = df['Company Type'].replace({'Product': 0, 'Service': 1})

    df['WFH Setup Available'] = df['WFH Setup Available'].replace({'No': 0, 'Yes': 1})

    return df

pure\_data = preprocess\_inputs(data)

pure\_data.head()



# Ищем выбросы в данных

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

plt.subplot(3, 2, 1)

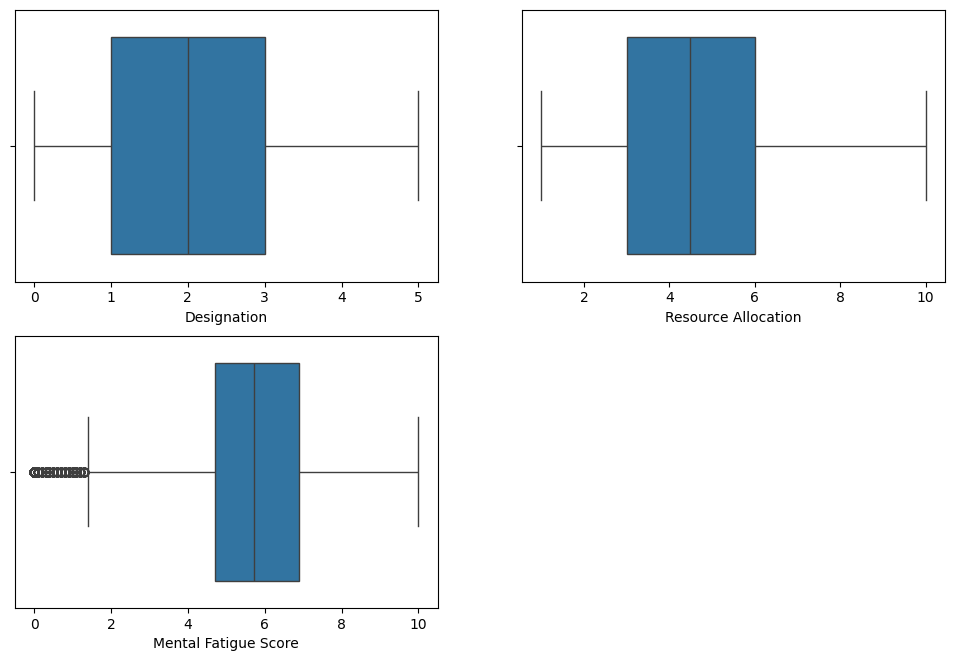
sns.boxplot(data=pure\_data, x='Designation')

plt.subplot(3, 2, 2)

sns.boxplot(data=pure\_data, x='Resource Allocation')

plt.subplot(3, 2, 3)

sns.boxplot(data=pure\_data, x='Mental Fatigue Score')



# Строим матрицу корреляции

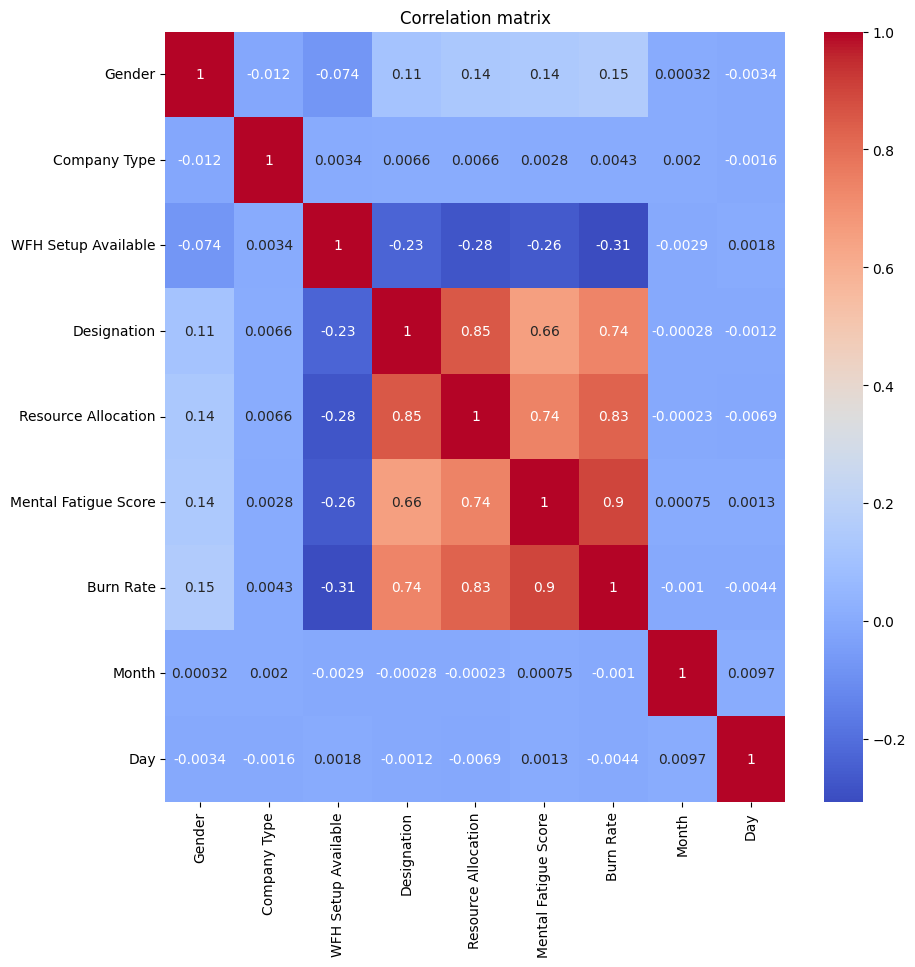
correlation\_matrix = pure\_data.corr()

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

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

plt.title("Correlation matrix")

plt.show()



# Вычисление важности признаков

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

x = pure\_data.drop('Burn Rate', axis=1)

y = pure\_data['Burn Rate']

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

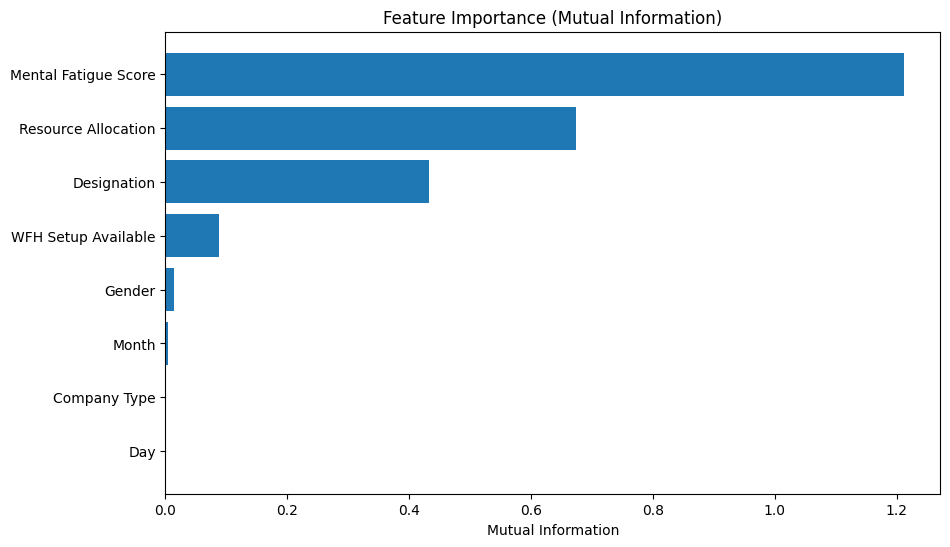
plt.figure(figsize=(10, 6))

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

plt.xlabel('Mutual Information')

plt.title('Feature Importance (Mutual Information)')

plt.show()



# удаляем столбцы с низкой важностью

def drop\_unnecessary\_correlation(data):

    data = data.copy()

    # Drop low importance

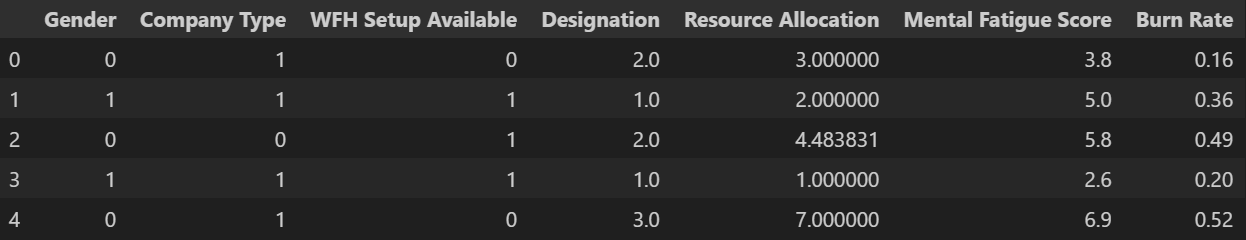
    data = data.drop('Month', axis=1)

    data = data.drop('Day', axis=1)

    return data

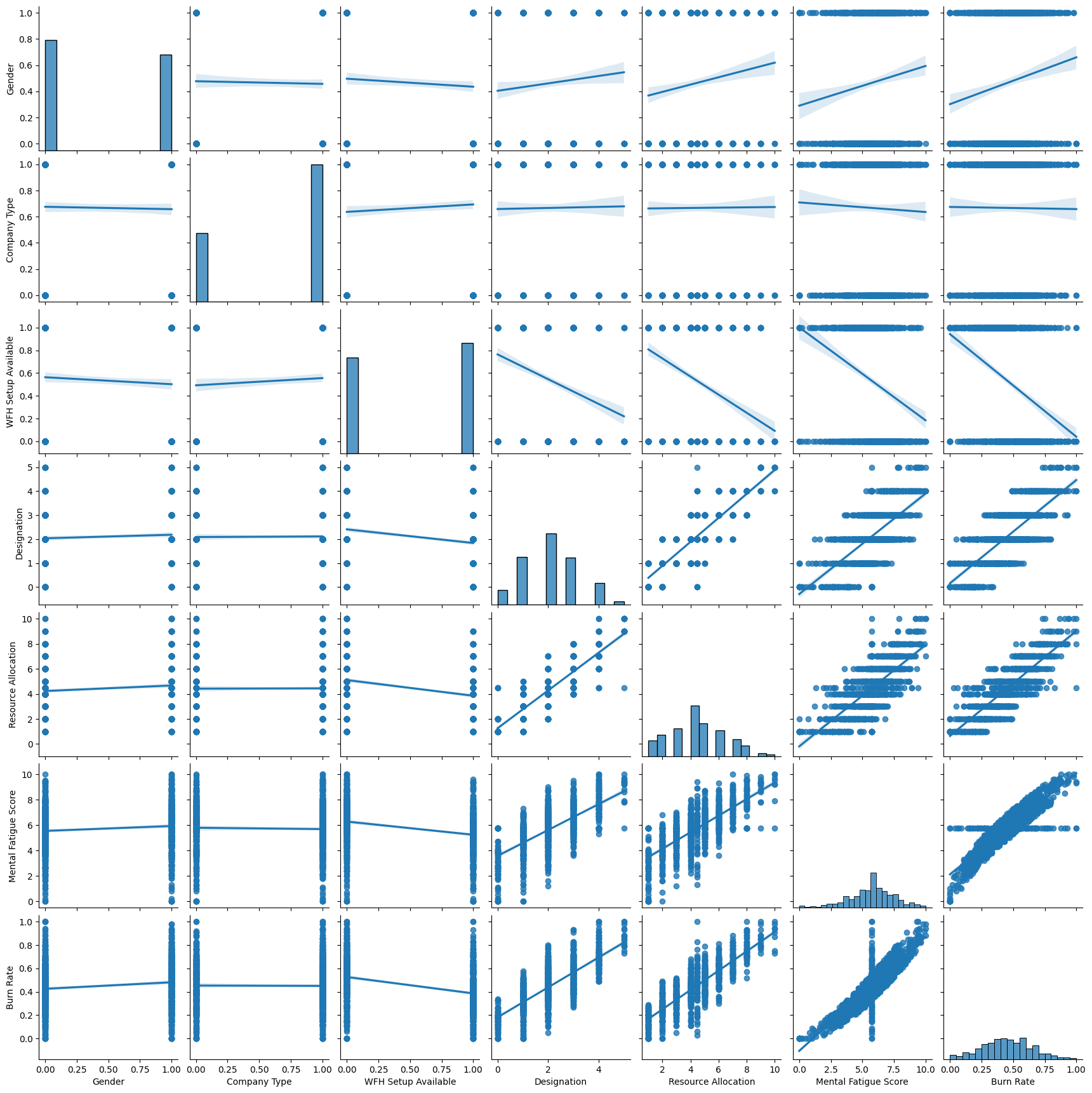
pure\_data = drop\_unnecessary\_correlation(pure\_data)

pure\_data.head()



# Строим графики зависимостей величин друг от друга

sns.pairplot(pure\_data[:1000], kind="reg")



# Создаём one-hot векторы

pure\_data = pd.get\_dummies(pure\_data, columns=['Designation',], dtype=int)

# Создание обучающей и тестовой выборок, и их нормализация

x = pure\_data.drop('Burn Rate', axis=1)

y = pure\_data['Burn Rate']

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[:, ['Resource Allocation', 'Mental Fatigue Score']]

x\_test\_scaled = x\_test.loc[:, ['Resource Allocation', 'Mental Fatigue Score']]

scaler = MinMaxScaler().fit(x\_train\_scaled)

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

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

x\_train = np.concatenate((x\_train\_scaled, x\_train.drop(['Resource Allocation', 'Mental Fatigue Score'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['Resource Allocation', 'Mental Fatigue Score'], 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, 12),

                                 nn.ReLU(),

                                 nn.Linear(12, 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 = 200

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)

    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 test\_mse, test\_mae, test\_r2

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:

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

    result.loc[len(result.index)] = [str(model), 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 = 0.0377657487988472

mae = 0.1560211181640625

r2 = 0.03376091493816158

**Выводы**

Целью работы было провести первичный разведочный анализ данных и подготовить их для использования в математической модели. Для этого были исключены некоторые столбцы из исходного датасета и преобразованы некоторые данные.

Затем были обучены и протестированы разные модели машинного обучения из библиотеки sklearn, такие как DecisionTreeRegressor, RandomForestRegressor, SVR, KNeighborsRegressor, MLPRegressor, GradientBoostingRegressor. Самой эффективной моделью оказалась модель GradientBoostingRegressor, которая достигла точности 83%