Ивонин Даниил Александрович, группа 20-1

Лабораторная работа № 3

**Цель работы**

Решить задачу предсказания эффективности сна (от 0 до 1).

**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

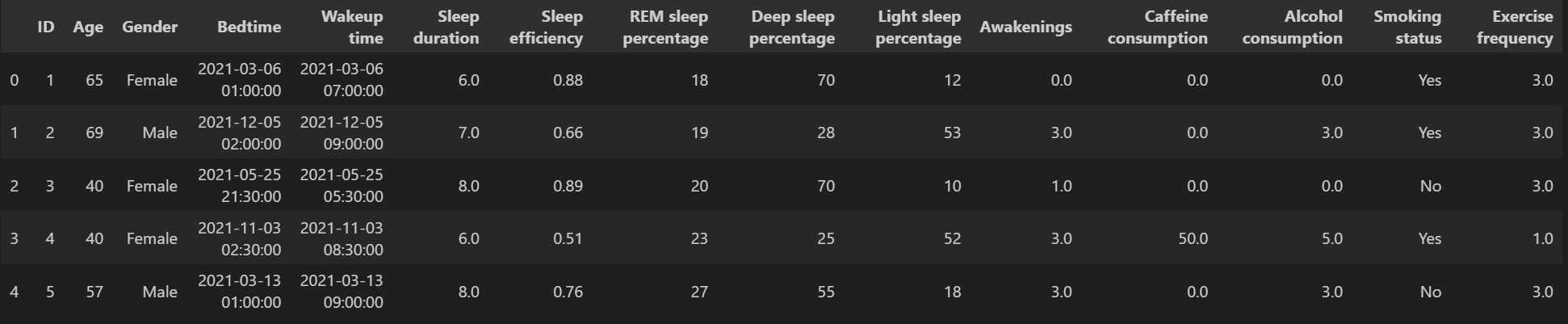
import pandas as pd

import seaborn as sns

# Просмотр датасета

df = pd.read\_csv('Sleep\_Efficiency.csv')

df.head()



df.drop('ID', axis=1, inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 452 entries, 0 to 451

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Age 452 non-null int64

1 Gender 452 non-null object

2 Bedtime 452 non-null object

3 Wakeup time 452 non-null object

4 Sleep duration 452 non-null float64

5 Sleep efficiency 452 non-null float64

6 REM sleep percentage 452 non-null int64

7 Deep sleep percentage 452 non-null int64

8 Light sleep percentage 452 non-null int64

9 Awakenings 432 non-null float64

10 Caffeine consumption 427 non-null float64

11 Alcohol consumption 438 non-null float64

12 Smoking status 452 non-null object

13 Exercise frequency 446 non-null float64

dtypes: float64(6), int64(4), object(4)

memory usage: 49.6+ KB

# Ищем отсутствующие значения

df.isna().sum()

Age 0

Gender 0

Bedtime 0

Wakeup time 0

Sleep duration 0

Sleep efficiency 0

REM sleep percentage 0

Deep sleep percentage 0

Light sleep percentage 0

Awakenings 20

Caffeine consumption 25

Alcohol consumption 14

Smoking status 0

Exercise frequency 6

dtype: int64

# Заполняем отсутствующие значения

df['Awakenings'].fillna(df['Awakenings'].mean(), inplace=True)

df['Caffeine consumption'].fillna(0.0, inplace=True)

df['Alcohol consumption'].fillna(0.0, inplace=True)

df['Exercise frequency'].fillna(df['Exercise frequency'].mean(), inplace=True)

df.isna().sum()

Age 0

Gender 0

Bedtime 0

Wakeup time 0

Sleep duration 0

Sleep efficiency 0

REM sleep percentage 0

Deep sleep percentage 0

Light sleep percentage 0

Awakenings 0

Caffeine consumption 0

Alcohol consumption 0

Smoking status 0

Exercise frequency 0

dtype: int64

# Конвертировать   2021-03-06 01:30:00     ->      1.5

df['Bedtime'] = pd.to\_datetime(df['Bedtime']).dt.hour + pd.to\_datetime(df['Bedtime']).dt.minute / 60

df['Wakeup time'] = pd.to\_datetime(df['Wakeup time']).dt.hour + pd.to\_datetime(df['Wakeup time']).dt.minute / 60

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

df["Smoking status"] = df["Smoking status"].replace({"Yes": 1, "No": 0})

# Ищем выбросы в данных

import matplotlib.pyplot as plt

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

plt.subplot(3, 2, 1)

sns.boxplot(data=df, x='Sleep duration')

plt.subplot(3, 2, 2)

sns.boxplot(data=df, x='Sleep efficiency')

plt.subplot(3, 2, 3)

sns.boxplot(data=df, x='REM sleep percentage')

plt.subplot(3, 2, 4)

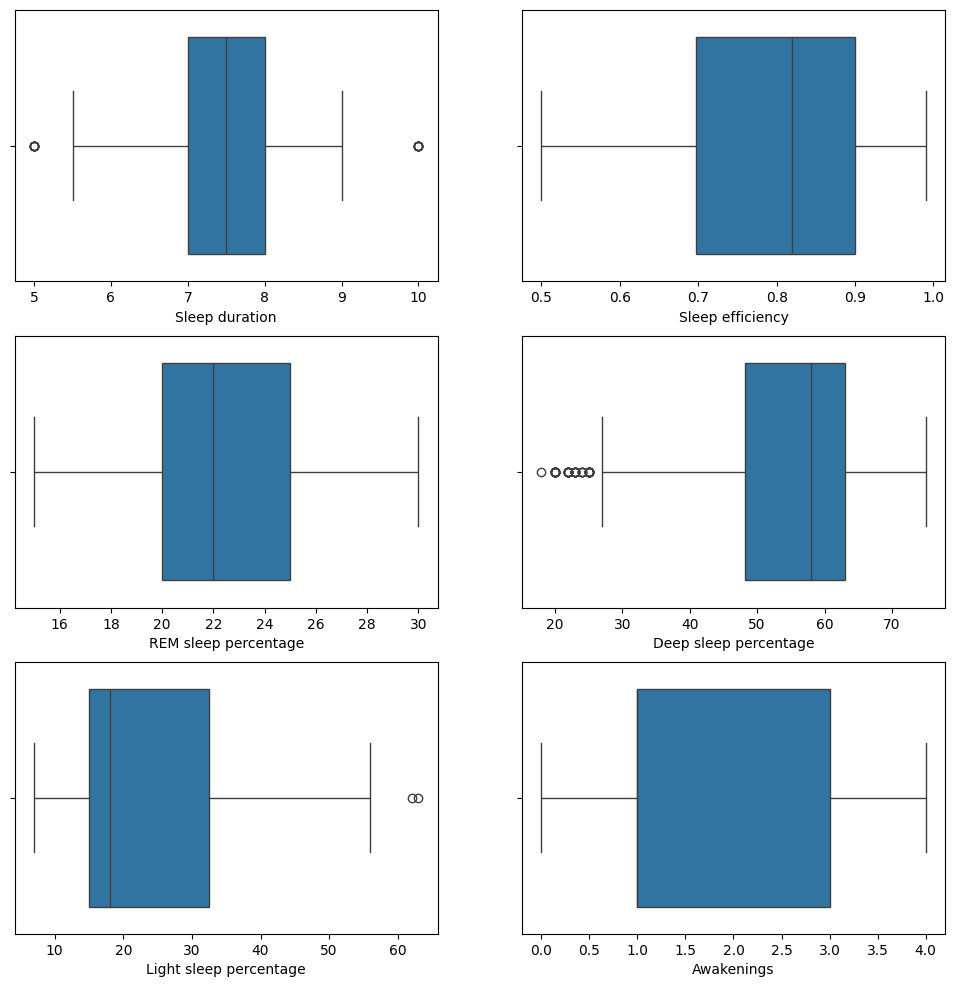
sns.boxplot(data=df, x='Deep sleep percentage')

plt.subplot(3, 2, 5)

sns.boxplot(data=df, x='Light sleep percentage')

plt.subplot(3, 2, 6)

sns.boxplot(data=df, x='Awakenings')



plt.figure(figsize=(12, 3))

plt.subplot(1, 3, 1)

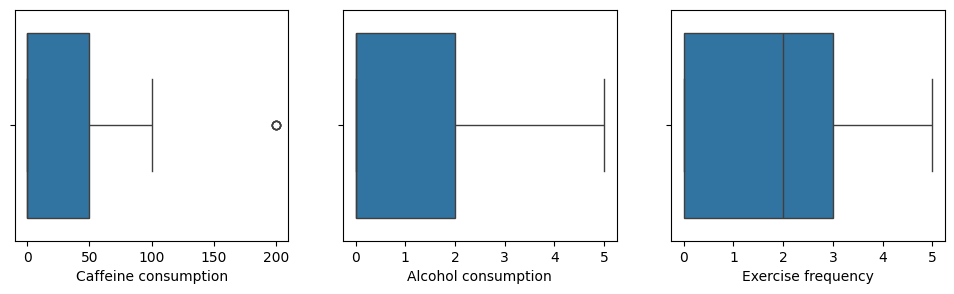
sns.boxplot(data=df, x='Caffeine consumption')

plt.subplot(1, 3, 2)

sns.boxplot(data=df, x='Alcohol consumption')

plt.subplot(1, 3, 3)

sns.boxplot(data=df, x='Exercise frequency')



# Избавляемся от выбросов

def handle\_outliers(data, column):

    q1 = data[column].quantile(0.25) # Определение 25% процентиль

    q3 = data[column].quantile(0.75) # Определение 75% процентиль

    iqr = q3 - q1 # Определение размера ящика

    lower\_whisker = q1 - 1.5 \* iqr # Определение нижнего усика

    upper\_whisker = q3 + 1.5 \* iqr # Определение верхнего усика

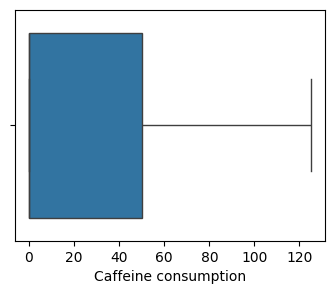
    # Вариант 1: Замена выбросов значениями границ усов (если точки находятся за пределами усов, то тогда заменяем их на граничные значения усов)

    data[column] = data[column].apply(lambda x: lower\_whisker if x < lower\_whisker else (upper\_whisker if x > upper\_whisker else x))

handle\_outliers(df, 'Caffeine consumption')

plt.figure(figsize=(4, 3))

sns.boxplot(data=df, x='Caffeine consumption')



# Строим матрицу корреляции

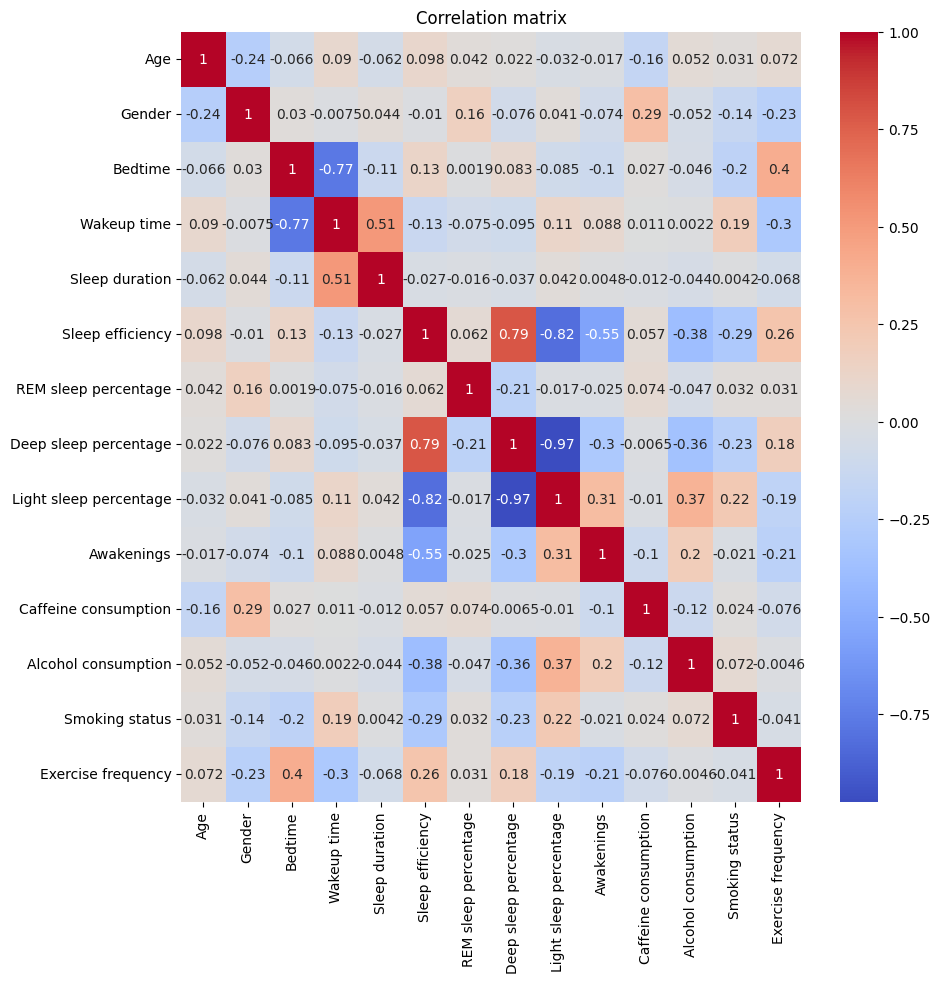
correlation\_matrix = df.corr()

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 = df.drop('Sleep efficiency', axis=1)

y = df['Sleep efficiency']

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

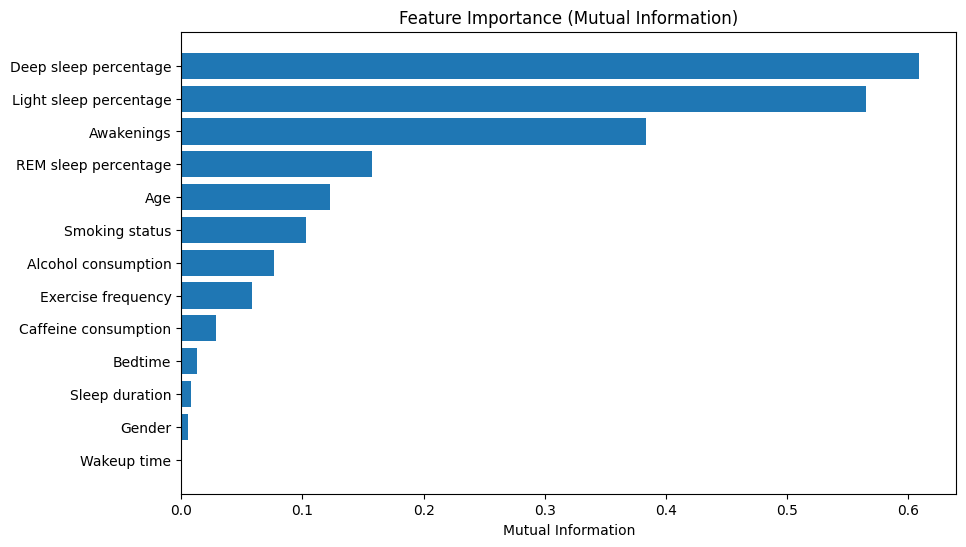
plt.figure(figsize=(10, 6))

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

plt.xlabel('Mutual Information')

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

plt.show()



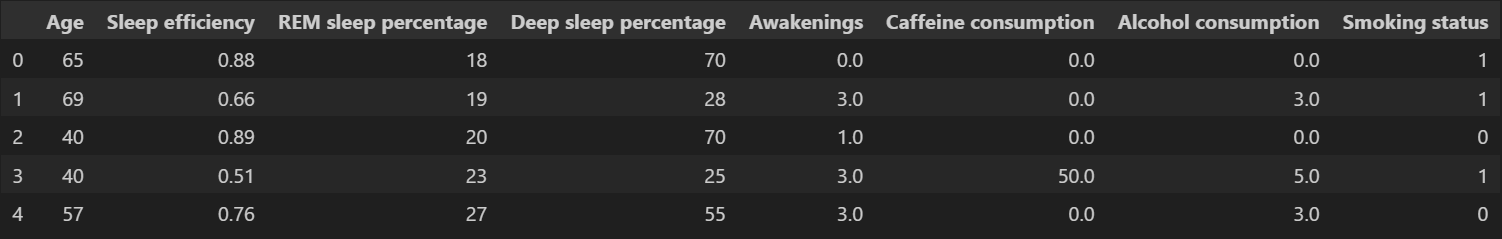
# т.к. deep sleep и light sleep имеют негативную корреляцию, удаляем один из столбцов

df = df.drop('Light sleep percentage',axis=1)

# удаляем столбцы с низкой важностью

df = df.drop(['Gender','Bedtime','Wakeup time','Sleep duration','Exercise frequency'],axis=1)

df.head()



# 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[:, ['Age', 'REM sleep percentage', 'Deep sleep percentage', 'Awakenings', 'Caffeine consumption', 'Alcohol consumption']]

x\_test\_scaled = x\_test.loc[:, ['Age', 'REM sleep percentage', 'Deep sleep percentage', 'Awakenings', 'Caffeine consumption', 'Alcohol consumption']]

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(['Age', 'REM sleep percentage', 'Deep sleep percentage', 'Awakenings', 'Caffeine consumption', 'Alcohol consumption'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['Age', 'REM sleep percentage', 'Deep sleep percentage', 'Awakenings', 'Caffeine consumption', 'Alcohol consumption'], 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, round(input / 2)),

                                 nn.ReLU(),

                                 nn.Linear(round(input / 2), round(input / 4)),

                                 nn.ReLU(),

                                 nn.Linear(round(input / 4), 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)

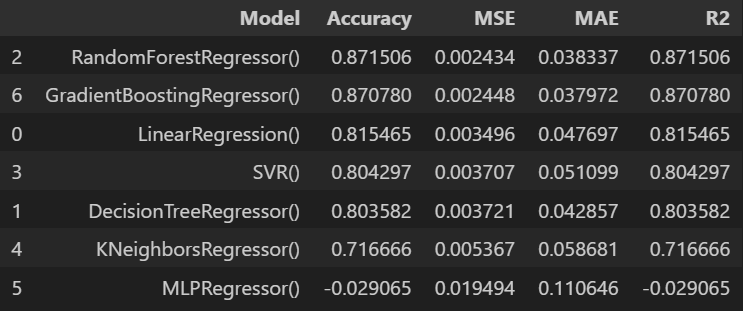
models.append(gradient\_boosting)

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)

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

mse = 0.049836739897727966

mae = 0.18369267880916595

**Выводы**

В результате выполнения работы был проведён первичный разведочный анализ данных, в следствие которого были удалены некоторые столбцы изначального датасета. Некоторые данные были конвертированы, например данные столбца Gender – Male -> 1, Female -> 0.

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

Наилучший результат показала модель RandomForestRegressor. Точность модели 87%.