Кулиев Фёдор Русланович, группа 20-1

Лабораторная работа № 3

**Цель работы**

Решить задачу предсказания размера головы опоссума.

**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

# Подключение датасета

data = pd.read\_csv('possum.csv')

data.head()



data.drop('case', axis=1, inplace=True)

# Ищем отсутствующие данные

data.isna().sum()

site 0

Pop 0

sex 0

age 2

hdlngth 0

skullw 0

totlngth 0

taill 0

footlgth 1

earconch 0

eye 0

chest 0

belly 0

dtype: int64

missing\_target\_rows = data.loc[data['age'].isna(), :].index

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

data['footlgth'] = data['footlgth'].fillna(data['footlgth'].mean())

data['Pop'].unique()

array(['Vic', 'other'], dtype=object)

data['sex'].unique()

array(['m', 'f'], dtype=object)

# Заменяем строковые данные на числовые

data['Pop'] = data['Pop'].replace({'other': 0, 'Vic': 1})

data['sex'] = data['sex'].replace({'m': 0, 'f': 1})

# Ищем выбросы в данных

# Строим матрицу корреляции

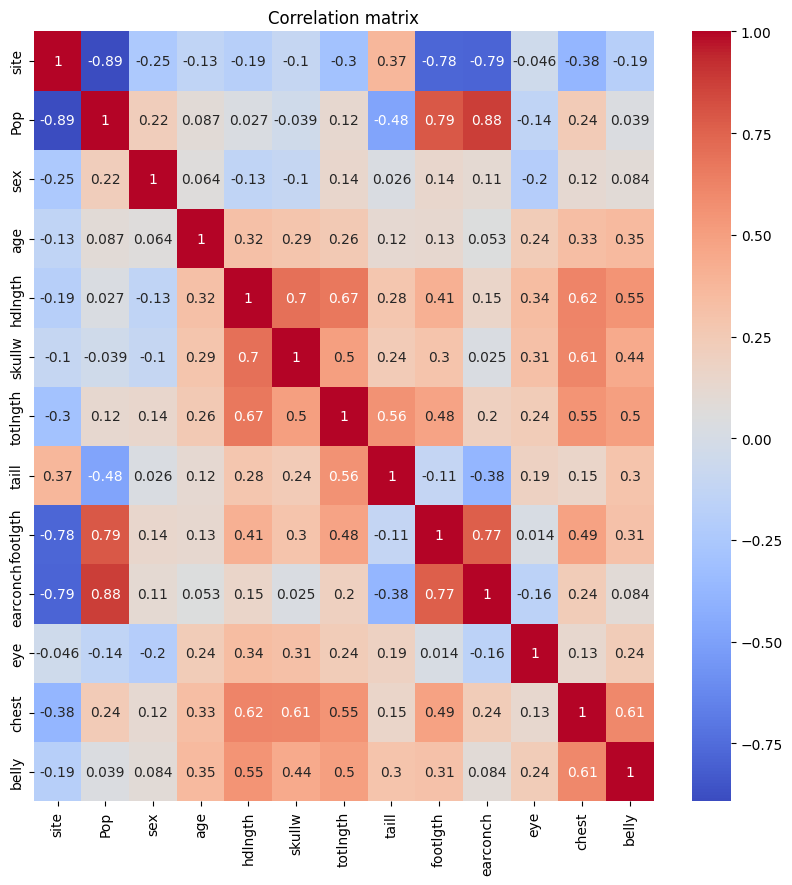
correlation\_matrix = data.corr()

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

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

plt.title("Correlation matrix")

plt.show()



data.drop('site', axis=1, inplace=True)

data.drop('Pop', axis=1, inplace=True)

# Вычисление важности признаков

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

x = data.drop('hdlngth', axis=1)

y = data['hdlngth']

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

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

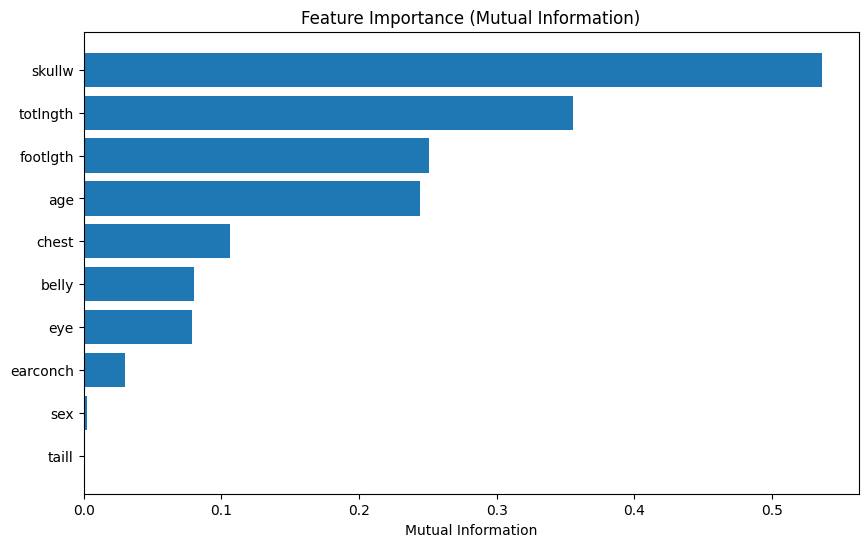
plt.figure(figsize=(10, 6))

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

plt.xlabel('Mutual Information')

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

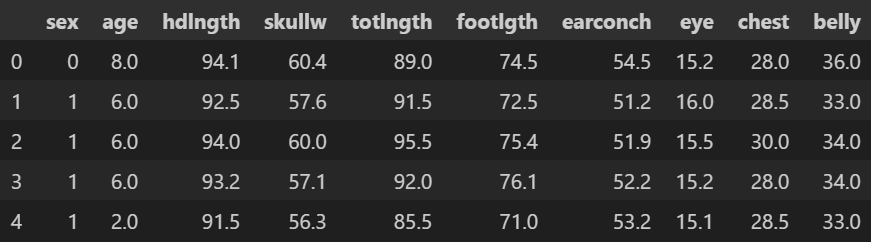
plt.show()



# удаляем столбцы с низкой важностью

data.drop('taill', axis=1, inplace=True)

data.head()



# Ищем выбросы в данных

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

plt.subplot(3, 2, 1)

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

plt.subplot(3, 2, 2)

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

plt.subplot(3, 2, 3)

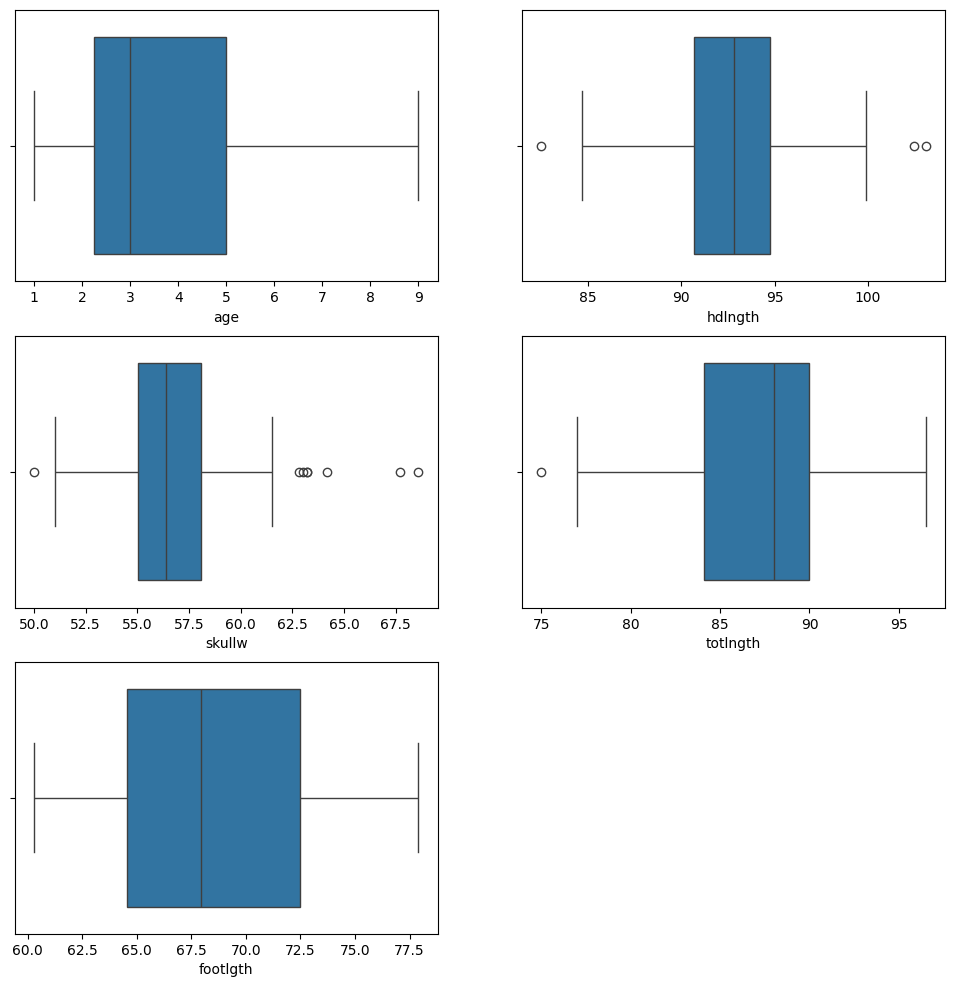
sns.boxplot(data=data, x='skullw')

plt.subplot(3, 2, 4)

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

plt.subplot(3, 2, 5)

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



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

plt.subplot(2, 2, 1)

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

plt.subplot(2, 2, 2)

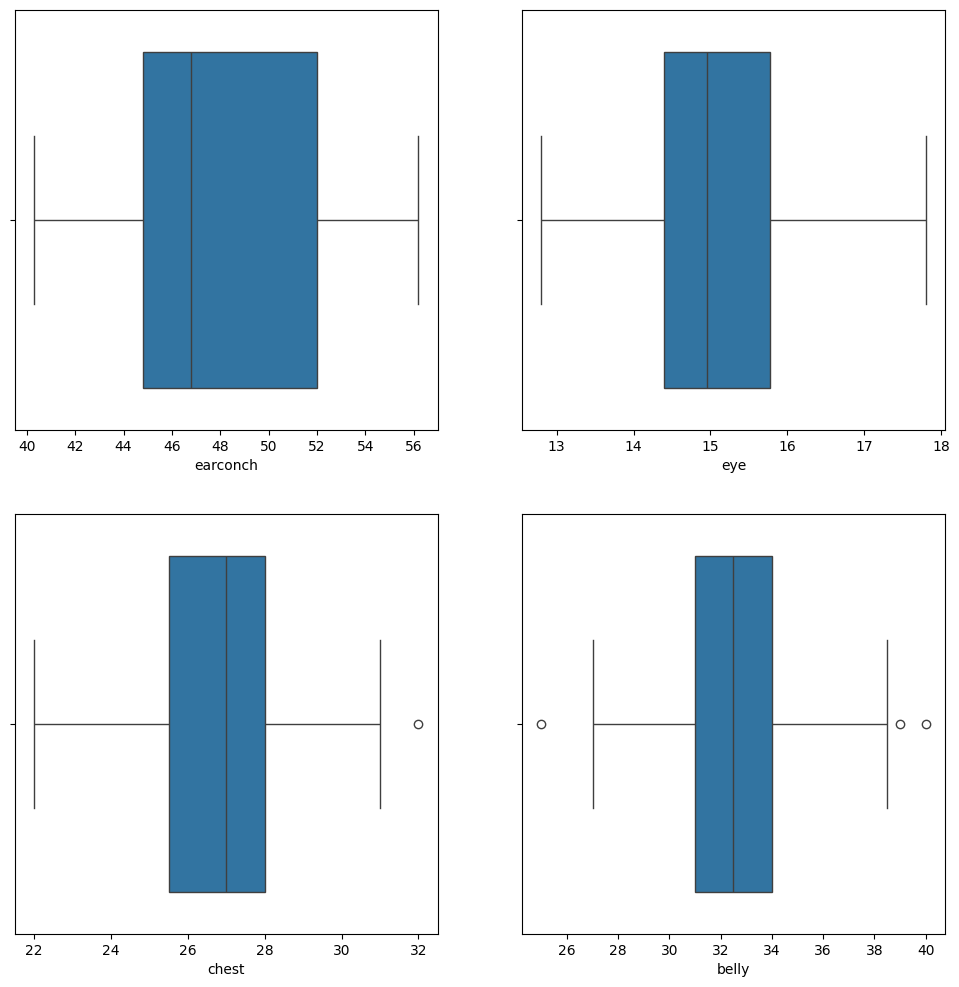
sns.boxplot(data=data, x='eye')

plt.subplot(2, 2, 3)

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

plt.subplot(2, 2, 4)

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



# 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))

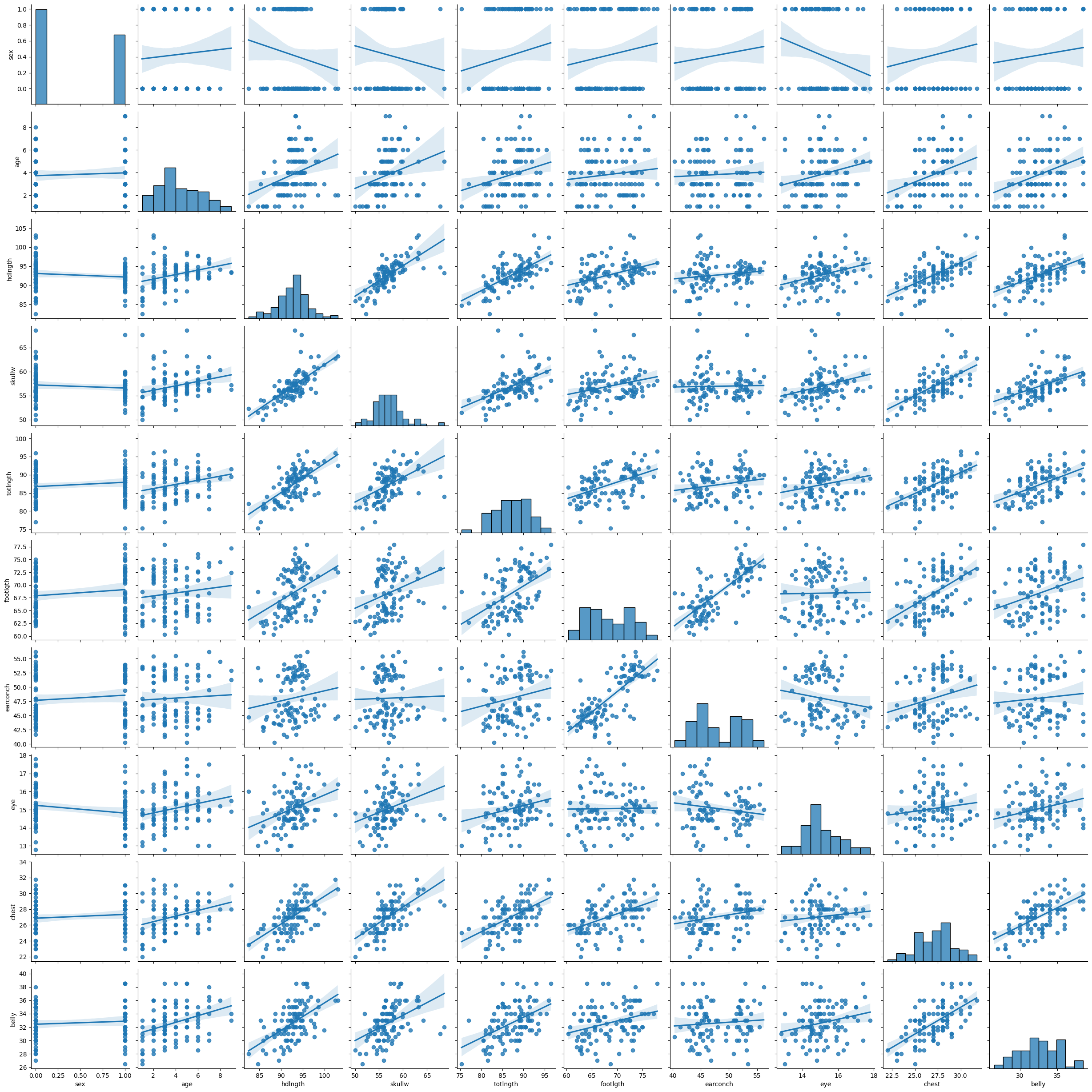
handle\_outliers(data, 'totlngth')

handle\_outliers(data, 'chest')

handle\_outliers(data, 'belly')

# Строим графики зависимостей величин друг от друга

sns.pairplot(data, kind="reg")



# Создание обучающей и тестовой выборок, и их нормализация

x = data.drop('hdlngth', axis=1)

y = data['hdlngth']

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)

xx\_train\_scaled = x\_train.loc[:, ['age', 'skullw', 'totlngth', 'footlgth', 'earconch', 'eye', 'chest', 'belly']]

x\_test\_scaled = x\_test.loc[:, ['age', 'skullw', 'totlngth', 'footlgth', 'earconch', 'eye', 'chest', 'belly']]

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', 'skullw', 'totlngth', 'footlgth', 'earconch', 'eye', 'chest', 'belly'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['age', 'skullw', 'totlngth', 'footlgth', 'earconch', 'eye', 'chest', 'belly'], 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, 32),

                                 nn.ReLU(),

                                 nn.Linear(32, 16),

                                 nn.ReLU(),

                                 nn.Linear(16, 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-2)

loss\_func = nn.MSELoss()

# Обучение и валидация нейросети

num\_iter = 3000

losses = []

losses\_val = []

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

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

for i in train\_loop:

    model\_nn.train()

    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()

    model\_nn.eval()

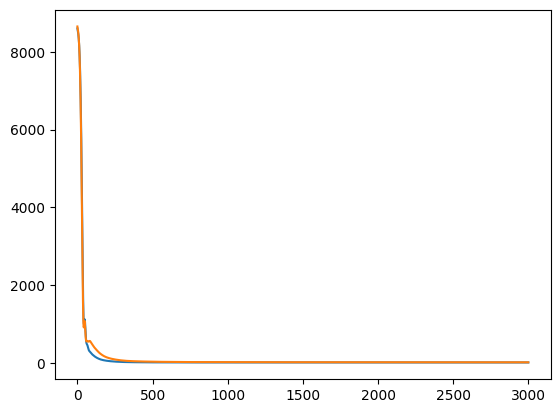
    outputs = model\_nn(x\_test)

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

    losses\_val.append(loss.item())

plt.plot(losses)

plt.plot(losses\_val)



**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 = 20.452560424804688

mae = 3.09315824508667

r2 = -0.47952639903127814

**Выводы**

В результате выполнения работы был проведён первичный разведочный анализ данных, а также обучены и протестированы различные модели машинного обучения. Наилучший результат показала модель KNeighborsRegressor из библиотеки sclearn, Её точность на тестовых данных составляет 72%.