Тулиновка Элина Владимировна, 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 pandas as pd

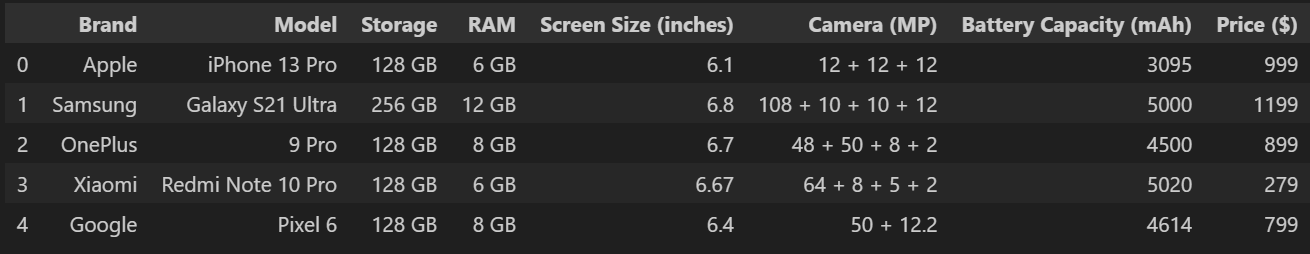
import numpy as np

import re

# Подключение датасета

df = pd.read\_csv('Mobile phone price.csv')

df.head()



df.isna().sum() #Detect missing values

Brand 0

Model 0

Storage 0

RAM 0

Screen Size (inches) 0

Camera (MP) 0

Battery Capacity (mAh) 0

Price ($) 0

dtype: int64

#В названиях столбцов есть лишние пробелы, поэтому удаляем их

df.columns = df.columns.str.strip()

df.columns = df.columns.str.replace(" ","")

df.rename(columns={'Price($)': 'Price',

                     'ScreenSize(inches)': 'ScreenSize',

                     'Camera(MP)': 'Camera',

                     'BatteryCapacity(mAh)': 'BetteryCapacity',

                     'Price($)': 'Price'}, inplace=True)

print(df.columns)

Index(['Brand', 'Model', 'Storage', 'RAM', 'ScreenSize', 'Camera',

'BetteryCapacity', 'Price'],

dtype='object')

df['Brand'].value\_counts()

Brand

Samsung 79

Xiaomi 67

Oppo 56

Realme 43

Vivo 35

Apple 30

Nokia 28

Motorola 23

OnePlus 15

Huawei 12

Google 7

Asus 4

LG 3

Blackberry 3

Sony 1

CAT 1

Name: count, dtype: int64

df['Model'].value\_counts()

Model

Poco X3 Pro 6

Y33s 5

iPhone XR 5

C21Y 5

Redmi 10 Prime 5

..

Mi 11 Lite NE 1

Moto G Stylus 2021 1

A54s 5G 1

Galaxy M52 1

Reno3 1

Name: count, Length: 239, dtype: int64

# Наименований моделей телефонов очень много, поэтому удаляем данный столбец

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

#Переводим RAM, Storage, Price, ScreenSize в числовой тип данных

df["Storage"] = [int(re.findall(r'\d+', value)[0]) for value in df["Storage"].values]

df["RAM"] = [int(re.findall(r'\d+', value)[0]) for value in df["RAM"].values]

df["Price"] = [int(re.findall(r'\d+,\*\d\*', value)[0].replace(',', '')) for value in df["Price"].values]

df["ScreenSize"] = [float(re.findall(r'\d+\.\*\d\*', value)[0]) for value in df["ScreenSize"].values]

df['Camera'].value\_counts()

Camera

13MP + 2MP + 2MP 29

48MP + 8MP + 2MP + 2MP 16

48+8+2+2 13

13MP + 2MP 11

64MP + 8MP + 2MP 11

..

13MP + 5MP + 2MP + 2MP 1

8MP 1

48MP + 8MP + 5MP + 2MP 1

13MP 1

48+13+8+2 1

Name: count, Length: 143, dtype: int64

# На разных моделях телефонах разное количество камер, но максимальное - 4

df['Camera'] = df['Camera'].str.replace('MP', '')

def convert(unconverted):

    converted = re.findall(r'\d+\.\*\d\*', str(unconverted))

    return float(converted[0]) if converted else 0

cameras = df['Camera'].str.split('+')

cameras = [[convert(el) for el in row] for row in cameras]

for i, row in enumerate(cameras):

    for \_ in range(4 - len(row)):

        cameras[i].append(0)

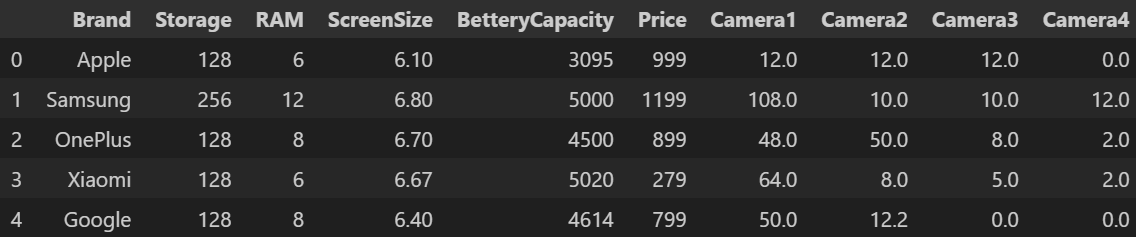
cameras = np.asarray(cameras)

camera\_df = pd.DataFrame({'Camera1': cameras[:, 0], 'Camera2': cameras[:, 1], 'Camera3': cameras[:, 2], 'Camera4': cameras[:, 3]})

df = pd.concat([df.reset\_index(drop=True), camera\_df.reset\_index(drop=True)], axis=1)

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

df.head()



# Ищем зависимости в данных

import matplotlib.pyplot as plt

import seaborn as sns

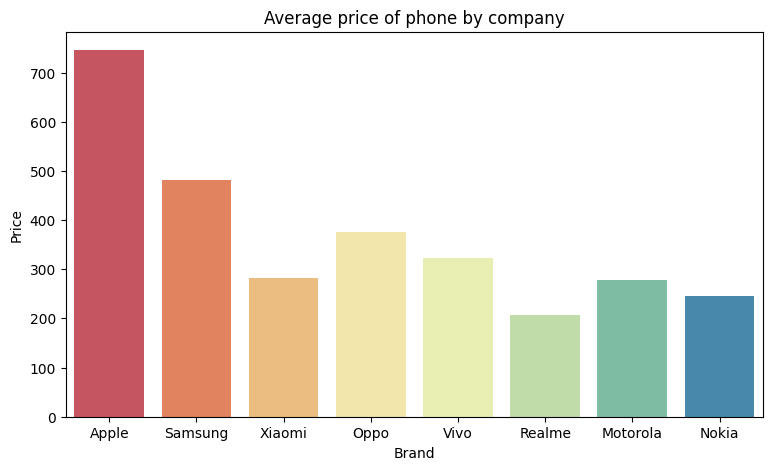
company\_list = df['Brand'].value\_counts().index[:8].tolist()

plt.figure(figsize=(9,5))

ax=sns.barplot(x='Brand', y='Price', data=df[df['Brand'].isin(company\_list)], palette='Spectral', errorbar=('ci', False),)

plt.title('Average price of phone by company')

plt.show()



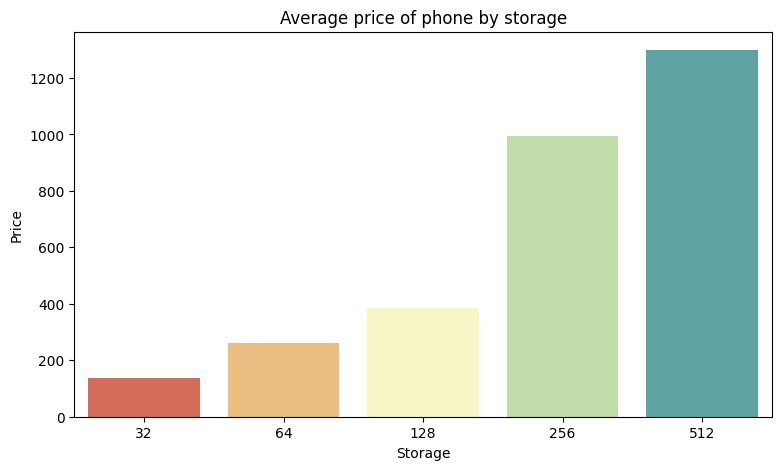
cpu\_list = df['Storage'].value\_counts().index[:8].tolist()

plt.figure(figsize=(9,5))

ax=sns.barplot(x='Storage', y='Price', data=df[df['Storage'].isin(cpu\_list)], palette='Spectral', errorbar=('ci', False),)

plt.title('Average price of phone by storage')

plt.show()



# Ищем выбросы в данных

import matplotlib.pyplot as plt

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

plt.subplot(3, 3, 1)

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

plt.subplot(3, 3, 2)

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

plt.subplot(3, 3, 3)

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

plt.subplot(3, 3, 4)

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

plt.subplot(3, 3, 5)

sns.boxplot(data=df, x='Camera1')

plt.subplot(3, 3, 6)

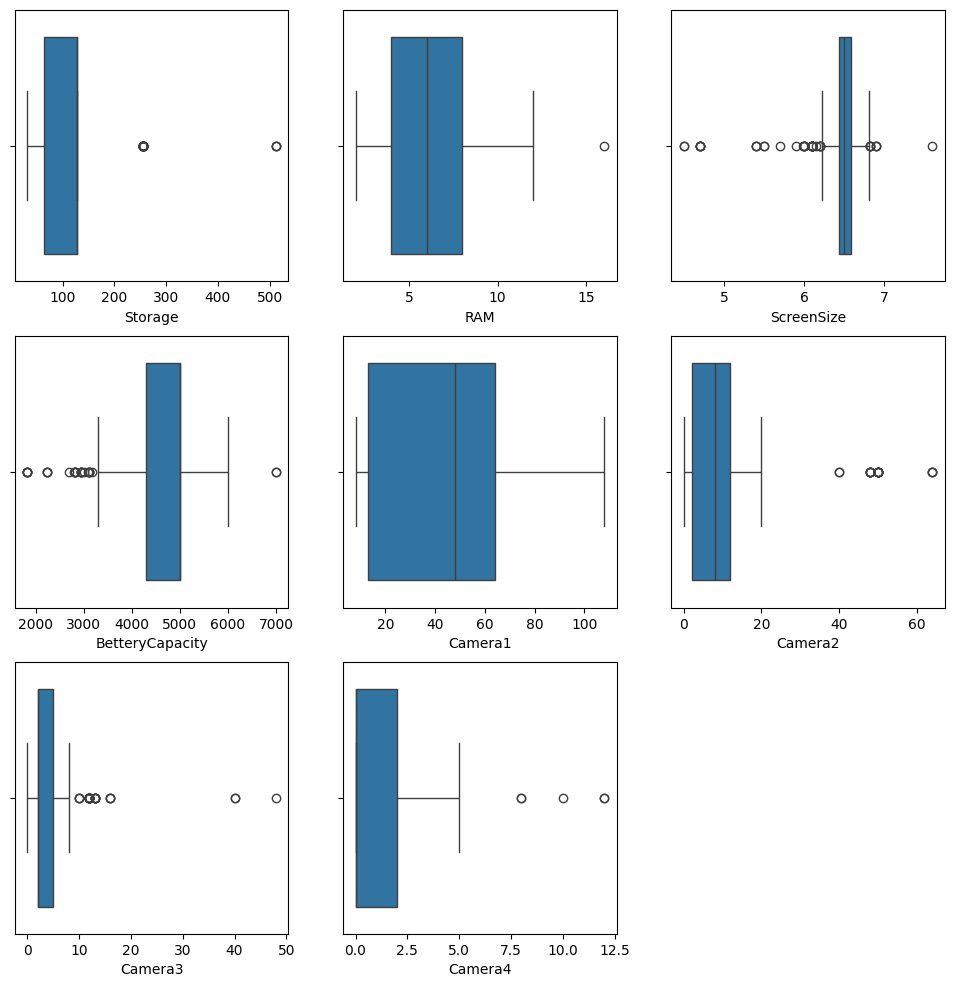
sns.boxplot(data=df, x='Camera2')

plt.subplot(3, 3, 7)

sns.boxplot(data=df, x='Camera3')

plt.subplot(3, 3, 8)

sns.boxplot(data=df, x='Camera4')



# Строим матрицу корреляции

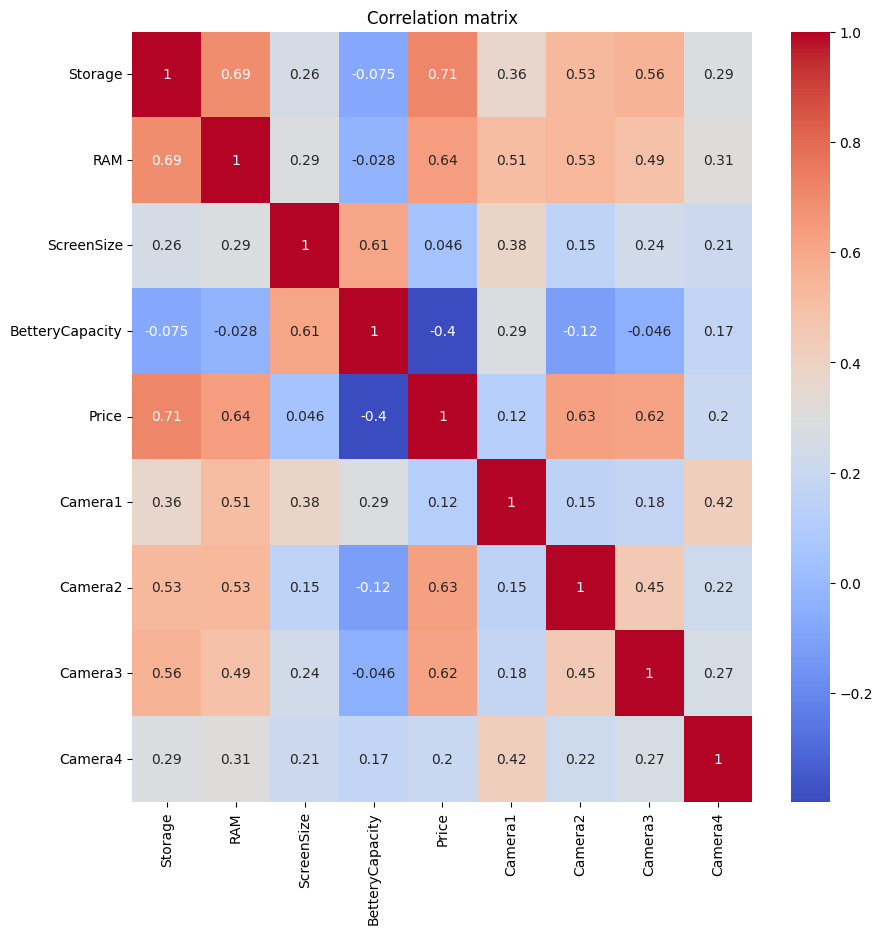
correlation\_matrix = df.drop('Brand', axis=1).corr()

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 = df.drop(['Brand', 'Price'], axis=1)

y = df['Price']

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)

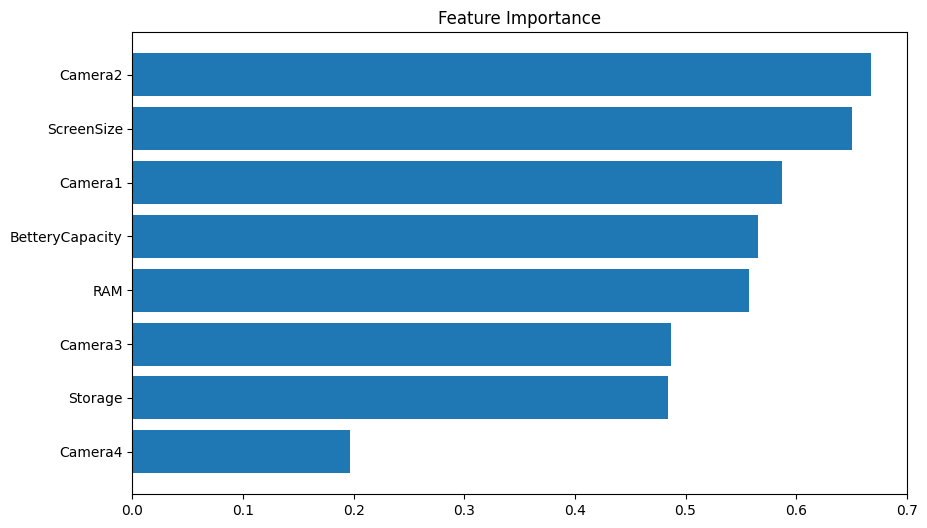
# Визуализация важности признаков

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

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

plt.title('Feature Importance')

plt.show()

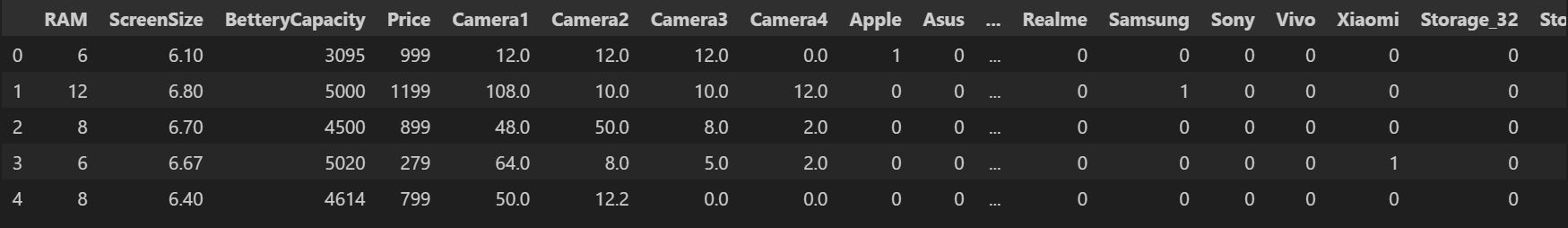


# Создаём one-hot векторы

df = pd.concat([df.drop('Brand', axis=1), pd.get\_dummies(df['Brand'], dtype=int)], axis=1)

df = pd.get\_dummies(df, columns=['Storage',], dtype=int)

df.head()



# Создание обучающей и тестовой выборок, и их нормализация

x = df.drop('Price', axis=1)

y = df['Price']

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[:, ['RAM', 'ScreenSize', 'BetteryCapacity', 'Camera1', 'Camera2', 'Camera3', 'Camera4']]

x\_test\_scaled = x\_test.loc[:, ['RAM', 'ScreenSize', 'BetteryCapacity', 'Camera1', 'Camera2', 'Camera3', 'Camera4']]

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(['RAM', 'ScreenSize', 'BetteryCapacity', 'Camera1', 'Camera2', 'Camera3', 'Camera4'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['RAM', 'ScreenSize', 'BetteryCapacity', 'Camera1', 'Camera2', 'Camera3', 'Camera4'], 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, 64),

                                 nn.ReLU(),

                                 nn.Linear(64, 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-3)

loss\_func = nn.MSELoss()

# Обучение нейросети

num\_iter = 400

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()

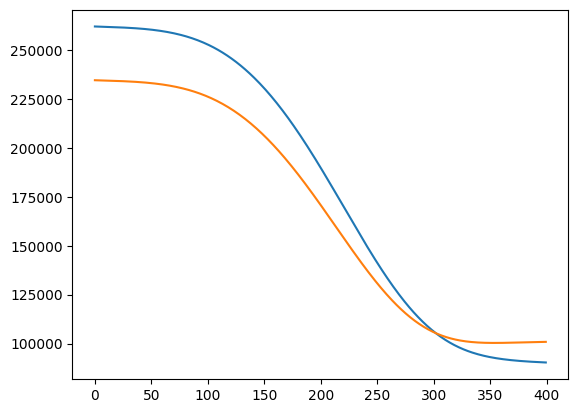
    val\_out = model\_nn(x\_test)

    val\_loss = loss\_func(val\_out, y\_test)

    losses\_val.append(val\_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)

    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)

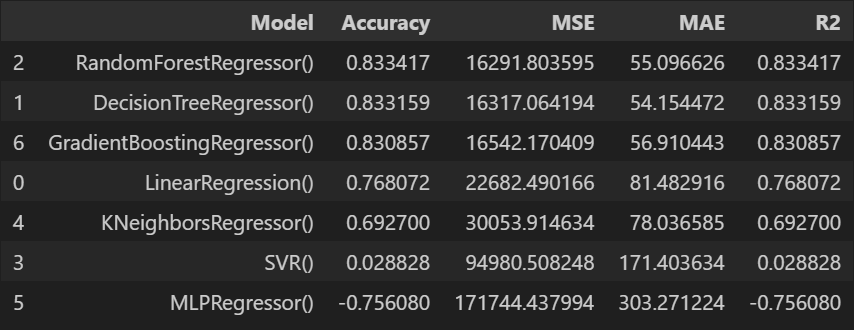
models.append(gradient\_boosting)

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 = 112166.7890625

mae = 239.7161102294922

r2 = -0.14690118241680183

**Выводы**

В результате выполнения лабораторной работы был выполнен предварительный разведочный анализ данных, в ходе которого категориальные данные были заменены на one-hot векторы, а остальные признаки нормализированы при помощи библиотеки sklearn.

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