Кимбрис Владимир Владимирович, группа 20-1

Лабораторная работа № 3

**Цель работы**

Решить задачу предсказания цены ноутбука.

**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

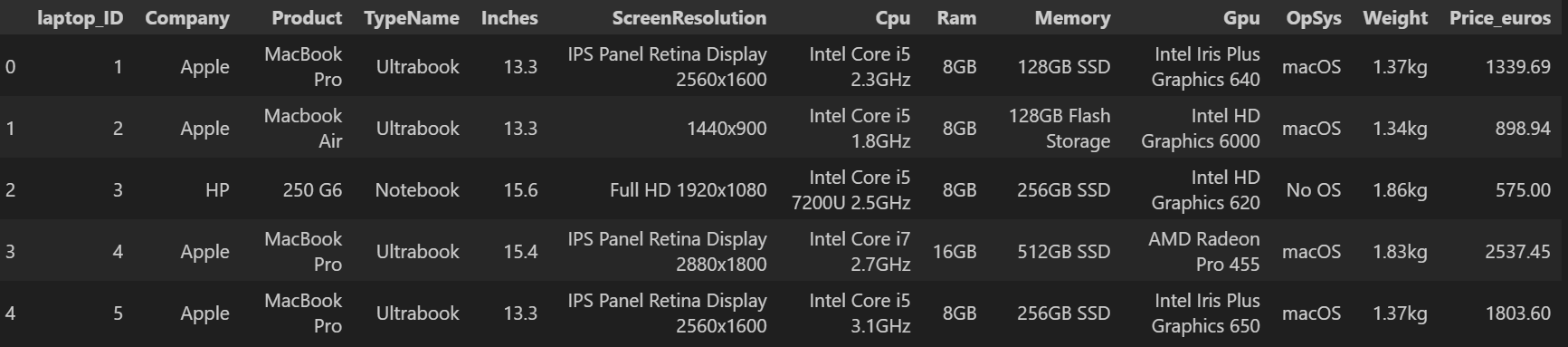
import pandas as pd

import seaborn as sns

# Просмотр датасета

df = pd.read\_csv('laptop\_price.csv', encoding = 'latin-1')

df.head()



df.drop('laptop\_ID', axis=1, inplace=True)

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

df.info()

RangeIndex: 1303 entries, 0 to 1302

Data columns (total 11 columns):

# Column Non-Null Count Dtype

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

0 Company 1303 non-null object

1 TypeName 1303 non-null object

2 Inches 1303 non-null float64

3 ScreenResolution 1303 non-null object

4 Cpu 1303 non-null object

5 Ram 1303 non-null object

6 Memory 1303 non-null object

7 Gpu 1303 non-null object

8 OpSys 1303 non-null object

9 Weight 1303 non-null object

10 Price\_euros 1303 non-null float64

# Ищем отсутствующие значения

df.isna().sum()

Company 0

TypeName 0

Inches 0

ScreenResolution 0

Cpu 0

Ram 0

Memory 0

Gpu 0

OpSys 0

Weight 0

Price\_euros 0

dtype: int64

# Ищем дубликаты

df.duplicated().sum()

29

# Удаляем дубликаты

df.drop\_duplicates(keep='first', inplace=True)

# Смотрим какие данные находятся в столбце ScreenResolution

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

ScreenResolution

Full HD 1920x1080 505

1366x768 262

IPS Panel Full HD 1920x1080 226

IPS Panel Full HD / Touchscreen 1920x1080 51

Full HD / Touchscreen 1920x1080 47

1600x900 23

Touchscreen 1366x768 16

Quad HD+ / Touchscreen 3200x1800 15

IPS Panel 4K Ultra HD 3840x2160 12

IPS Panel 4K Ultra HD / Touchscreen 3840x2160 11

4K Ultra HD / Touchscreen 3840x2160 10

4K Ultra HD 3840x2160 7

Touchscreen 2560x1440 7

IPS Panel 1366x768 7

IPS Panel Retina Display 2560x1600 6

IPS Panel Retina Display 2304x1440 6

Touchscreen 2256x1504 6

IPS Panel Touchscreen 2560x1440 5

IPS Panel Quad HD+ / Touchscreen 3200x1800 4

IPS Panel Touchscreen 1920x1200 4

1440x900 4

IPS Panel Retina Display 2880x1800 4

IPS Panel 2560x1440 4

2560x1440 3

Quad HD+ 3200x1800 3

1920x1080 3

Touchscreen 2400x1600 3

IPS Panel Quad HD+ 2560x1440 3

IPS Panel Touchscreen 1366x768 3

IPS Panel Touchscreen / 4K Ultra HD 3840x2160 2

IPS Panel Full HD 2160x1440 2

IPS Panel Quad HD+ 3200x1800 2

IPS Panel Retina Display 2736x1824 1

IPS Panel Full HD 1920x1200 1

IPS Panel Full HD 2560x1440 1

IPS Panel Full HD 1366x768 1

Touchscreen / Full HD 1920x1080 1

Touchscreen / Quad HD+ 3200x1800 1

Touchscreen / 4K Ultra HD 3840x2160 1

IPS Panel Touchscreen 2400x1600 1

Name: count, dtype: int64

# Разбиваем столбец

df['resolution'] = df['ScreenResolution'].str.extract(r'(\d+x\d+)')

df['screentype'] = df['ScreenResolution'].replace(r'(\d+x\d+|Full HD|Quad HD|Quad HD|\+|/|4K Ultra HD)','',regex=True)

df['screentype'] = df['screentype'].replace(r'(Touchscreen)','',regex=True)

df['touchscreen'] = df['screentype'].str.extract(r'(Touchscreen)')

df['touchscreen'] = df['touchscreen'].replace('Touchscreen',1)

df['touchscreen'] = df['touchscreen'].replace(np.nan,0)

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

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

screentype

547

295

IPS Panel 261

75

IPS Panel 68

IPS Panel Retina Display 17

IPS Panel 11

Name: count, dtype: int64

# Т.к. данные в данном столбце содержат либо IPS, либо пустое значение, можно удалить данный столбец

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

# Смотрим какие данные находятся в столбце CPU

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

Cpu

Intel Core i5 7200U 2.5GHz 190

Intel Core i7 7700HQ 2.8GHz 146

Intel Core i7 7500U 2.7GHz 132

Intel Core i7 8550U 1.8GHz 73

Intel Core i5 8250U 1.6GHz 72

...

Intel Core M M3-6Y30 0.9GHz 1

AMD A9-Series 9420 2.9GHz 1

Intel Core i5 2.9GHz 1

AMD A6-Series 7310 2GHz 1

AMD A9-Series 9410 2.9GHz 1

Name: count, Length: 118, dtype: int64

df['cpu\_freq(GHz)'] = df['Cpu'].str.extract(r'(\d+(?:\.\d+)?GHz)')

df['cpu\_freq(GHz)'] = df['cpu\_freq(GHz)'].str.replace('GHz', '').astype(float)

df['Cpu']= df['Cpu'].str.replace(r'(\d+(?:\.\d+)?GHz)', '', regex=True)

df['cpu\_brand'] = df['Cpu'].str.extract(r'^(\w+)')

df['cpu\_brand'].value\_counts()

def fetch\_processor(x):

    cpu\_name=" ".join(x.split()[0:3])

    if cpu\_name=='Intel Core i7' or cpu\_name == 'Intel Core i5' or cpu\_name == 'Intel Core i3':

        return cpu\_name

    elif cpu\_name.split()[0]=='Intel':

        return 'Other Intel Processor'

    else:

        return 'AMD Processor'

df['cpu\_brand']=df['Cpu'].apply(lambda x:fetch\_processor(x))

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

# Убираем единицы измерения

df['Ram'] = df['Ram'].str.replace('GB', '')

df.rename(columns={'Ram': 'Ram(GB)'}, inplace=True)

df['Ram(GB)'] = df['Ram(GB)'].astype(int)

ssd=df['Memory'].apply(lambda x:x.split('SSD')[0].split('Flash')[0] if 'SSD' in str(x) or 'Flash' in str(x) else x)

ssd=ssd.apply(lambda x:x.replace('GB','') if 'GB' in str(x) else x.replace('TB','') if 'TB' in str(x) else x)

ssd=ssd.apply(pd.to\_numeric,errors='coerce')

ssd=ssd.apply(lambda x:x if x>10 else x\*1024)

df['SSD']=ssd.fillna(0)

hdd=df['Memory'].apply(lambda x:x.split('HDD')[0].split('Hybrid')[0] if ('HDD' in str(x) and '+' not in str(x)) or ('Hybrid' in str(x) and '+' not in str(x))else

                   x.split('+')[-1].split('HDD')[0].split('Hybrid')[0] if '+' in str(x) else x)

hdd=hdd.apply(lambda x:x.replace('GB','') if 'GB' in str(x) else x.replace('TB','') if 'TB' in str(x) else x)

hdd=hdd.apply(pd.to\_numeric,errors='coerce')

hdd=hdd.apply(lambda x:x if x>10 else x\*1024)

df['HDD']=hdd.fillna(0)

df=df.drop(columns='Memory')

df['Weight'] = df['Weight'].str.replace('kg','').astype(float)

df["gpu\_model"] = df['Gpu'].str.extract(r'^(\w+)')

df['gpu\_brand'] = df['Gpu'].str.extract(r'^(\w+)')

df['gpu\_brand'].value\_counts()

df["gpu\_model"] = df['Gpu'].apply(lambda x: x.split()[-1])

df["gpu\_model"] = df["gpu\_model"].str.replace(r'[^0-9]', '', regex=True)

df["gpu\_model"] = df["gpu\_model"].replace('', '0').astype(int)

df.drop("Gpu", axis=1, inplace=True)

df.head()



# Ищем зависимость цены от бренда

import matplotlib.pyplot as plt

import seaborn as sns

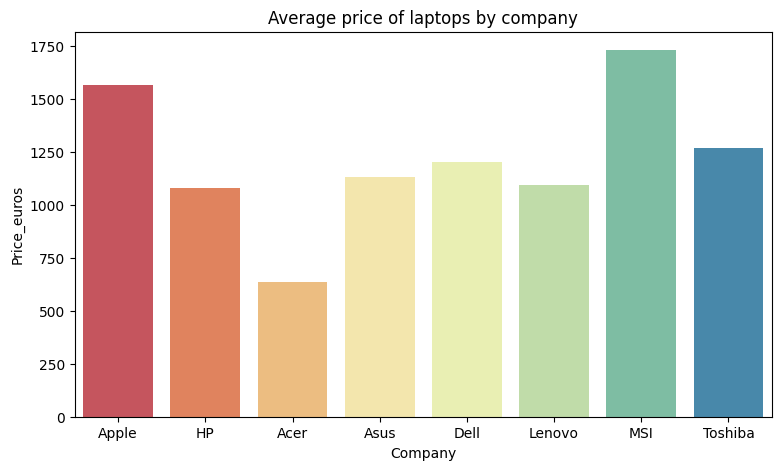
company\_list = df['Company'].value\_counts().index[:8].tolist()

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

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

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

plt.show()



# Ищем зависимость цены от модели процессора

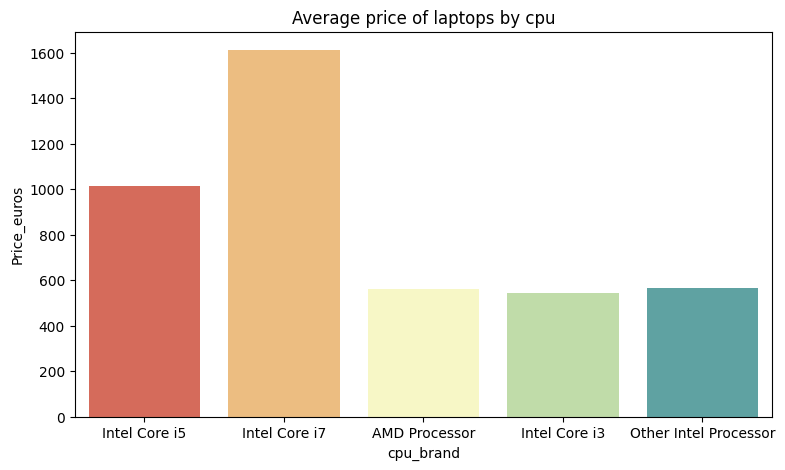
cpu\_list = df['cpu\_brand'].value\_counts().index[:8].tolist()

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

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

plt.title('Average price of laptops by cpu')

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='Inches')

plt.subplot(3, 3, 2)

sns.boxplot(data=df, x='Ram(GB)')

plt.subplot(3, 3, 3)

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

plt.subplot(3, 3, 4)

sns.boxplot(data=df, x='Price\_euros')

plt.subplot(3, 3, 5)

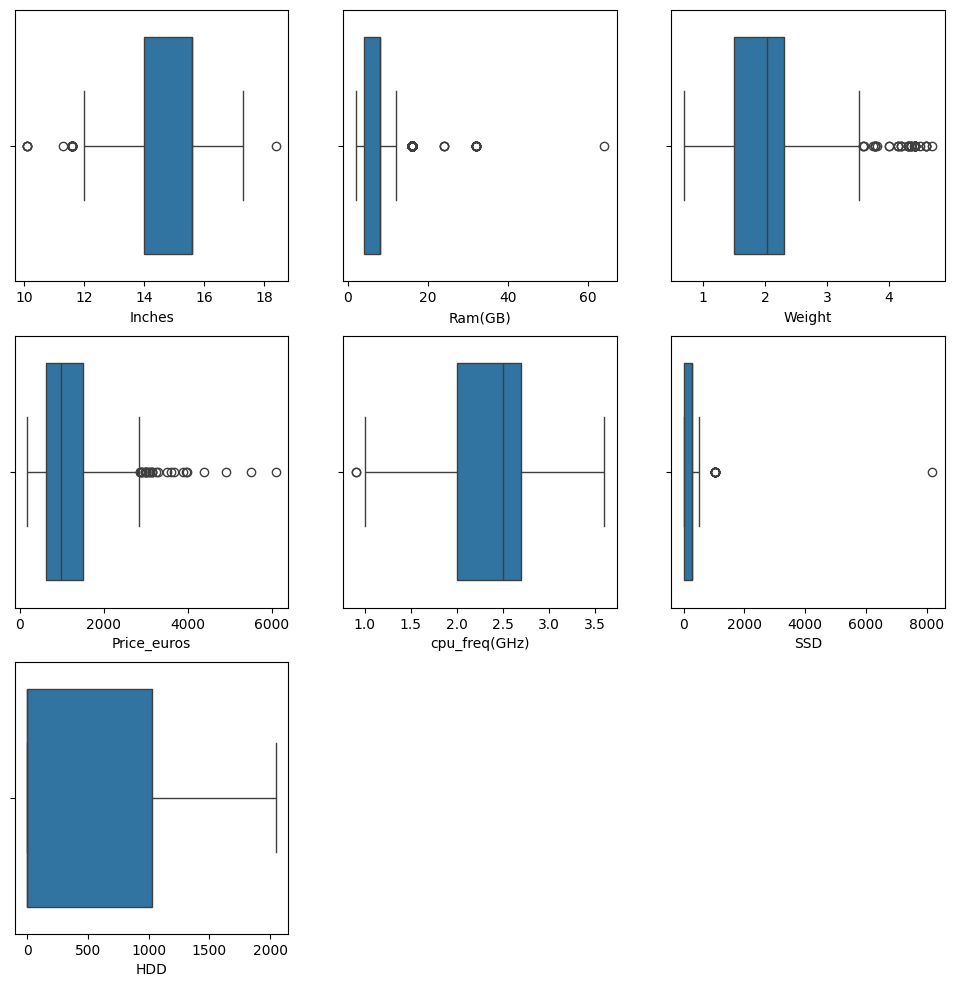
sns.boxplot(data=df, x='cpu\_freq(GHz)')

plt.subplot(3, 3, 6)

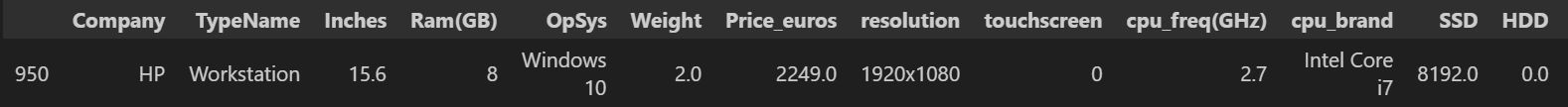
sns.boxplot(data=df, x='SSD')

plt.subplot(3, 3, 7)

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



df[df['SSD'] > 8000]



# Избавляемся от выбросов

def count\_whisker(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

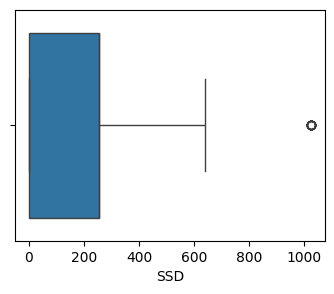
    upper\_whisker = q3 + 1.5 \* iqr # Определение верхнего усика

    return lower\_whisker, upper\_whisker

df['SSD'].replace([8192.0], count\_whisker(df, 'SSD')[1], inplace=True)

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

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



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

OpSys

Windows 10 1047

No OS 66

Linux 58

Windows 7 45

Chrome OS 26

macOS 13

Mac OS X 8

Windows 10 S 8

Android 2

Name: count, dtype: int64

def get\_os(a):

    if a=='Windows 10'or a=='Windows 10 S'or a=='Windows 7':

        return 'Windows'

    elif a=='macOS'or a=='Mac OS X':

        return 'Mac'

    else:

        return 'Others/No OS/Linux'

df['OpSys']=df['OpSys'].apply(get\_os)

# находим разрешение экрана по x и по y

df['x\_res']=df['resolution'].apply(lambda x:x.split()[-1].split('x')[0])

df['y\_res']=df['resolution'].apply(lambda x:x.split()[-1].split('x')[1])

# конвертируем в числовые значение

df['x\_res']=df['x\_res'].astype('int')

df['y\_res']=df['y\_res'].astype('int')

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

df['gpu\_model'].value\_counts()

gpu\_model

620 355

520 201

1050 71

0 64

940 54

...

7 1

5130 1

555 1

455 1

360 1

Name: count, Length: 62, dtype: int64

# Данных очень много, а смысловой нагрузки в них мало

df.drop('gpu\_model', axis=1, inplace=True)

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

TypeName

Notebook 706

Gaming 205

Ultrabook 194

2 in 1 Convertible 116

Workstation 29

Netbook 23

Name: count, dtype: int64

from sklearn.preprocessing import LabelEncoder

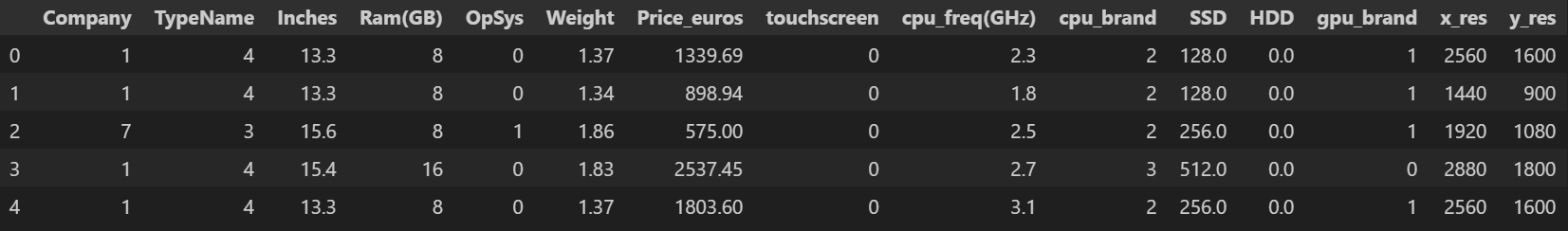
cols=['Company','TypeName','OpSys','cpu\_brand','gpu\_brand']

la= LabelEncoder()

for i in cols:

    df[i] = la.fit\_transform(df[i])

df.head()



# Строим матрицу корреляции

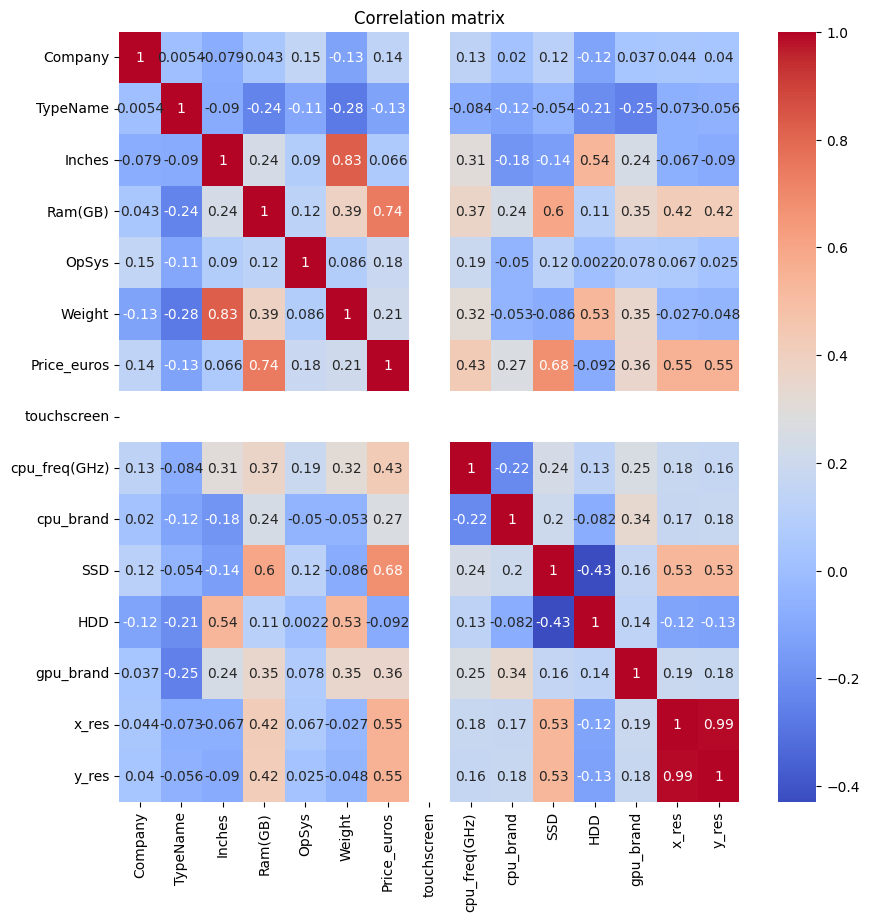
correlation\_matrix = df.corr()

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

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

plt.title("Correlation matrix")

plt.show()



# Создаём столбец с разрешением экрана (общее кол-во пикселей)

df["ScreenResolution"]=(df["x\_res"]\*df["y\_res"]).astype(float)

df.drop('x\_res', axis=1, inplace=True)

df.drop('y\_res', axis=1, inplace=True)

# Вычисление важности признаков

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

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

y = df['Price\_euros']

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)

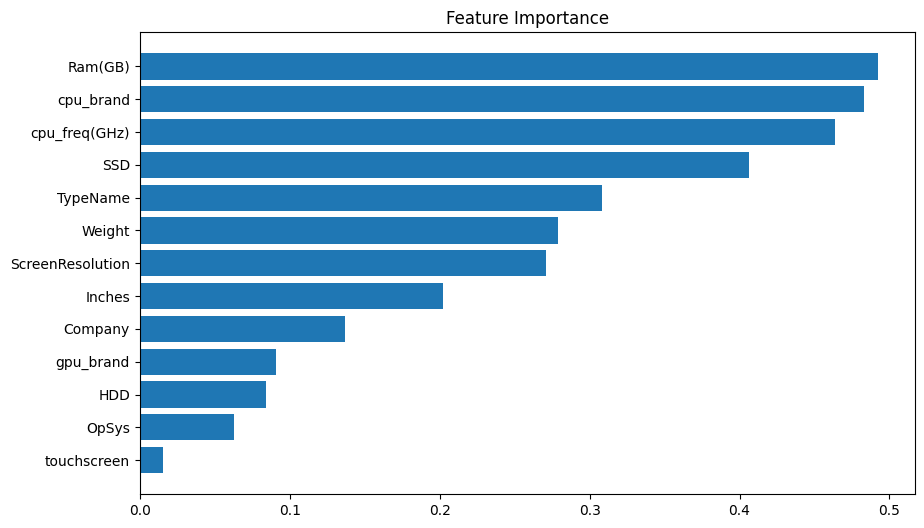
# Визуализация важности признаков

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

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

plt.title('Feature Importance')

plt.show()



ы

# удаляем столбцы с низкой важностью

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

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

y = df['Price\_euros']

# 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[:, ['Inches', 'Ram(GB)', 'Weight', 'cpu\_freq(GHz)', 'SSD', 'HDD', 'ScreenResolution']]

x\_test\_scaled = x\_test.loc[:, ['Inches', 'Ram(GB)', 'Weight', 'cpu\_freq(GHz)', 'SSD', 'HDD', 'ScreenResolution']]

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(['Inches', 'Ram(GB)', 'Weight', 'cpu\_freq(GHz)', 'SSD', 'HDD', 'ScreenResolution'], axis=1).to\_numpy()), axis=1)

x\_test = np.concatenate((x\_test\_scaled, x\_test.drop(['Inches', 'Ram(GB)', 'Weight', 'cpu\_freq(GHz)', 'SSD', 'HDD', 'ScreenResolution'], 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)),

                                 nn.ReLU(),

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

                                 nn.ReLU(),

                                 nn.Linear(round(input / 2), 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 = 1000

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)

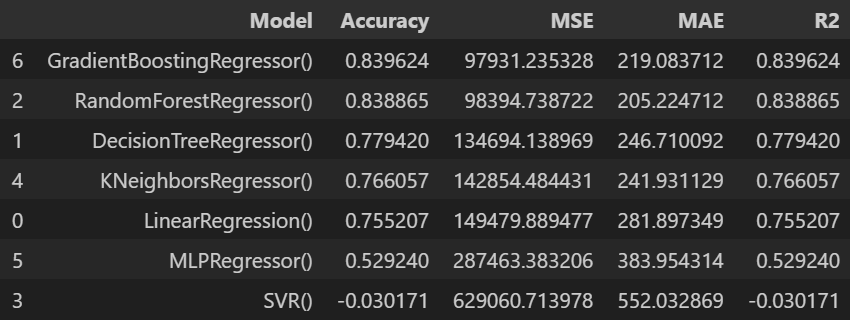
models.append(gradient\_boosting)

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 = 639002.875

mae = 566.251708984375

**Выводы**

В результате выполнения работы был проведён первичный разведочный анализ данных, в следствие которого были удалены некоторые столбцы изначального датасета. Некоторые данные были конвертированы, чтобы с ними могла работать математическая модель.

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 83%.