Description of the project

For our project, we decided to take data about laptops from the technodom.kz website. Now let's explain why we took this:

1. A large amount of information to work with. Information about the laptop, of course, will contain its characteristics and there are quite a lot of components of the them. Due to the fact that there are many types of data, we have room for experiments, a large selection of what and how to analyze.

2. The second point comes directly from the first. The information taken from the site helps us to fully meet all the requirements of the final project.

3. Interest. We are programmers and laptops have become part of our daily lives. Because of this, it will be interesting to dive into the analysis of the components of a laptop, what affects their price, what are the key components that distinguish different classes of laptops, for example, the difference between gaming laptops and work laptops, and so on.

During the implementation of this project, we will use all the tools that took place during the semester. We will:
1. Collect data from a website using web scraping.
2. Clean up the data we have collected.
3. Visualize the data to visually see the big picture.
4. Use machine learning algorithms to make a classification of laptop types.

Team members: Aliyev Damir, Bagdauletov Nurdaulet

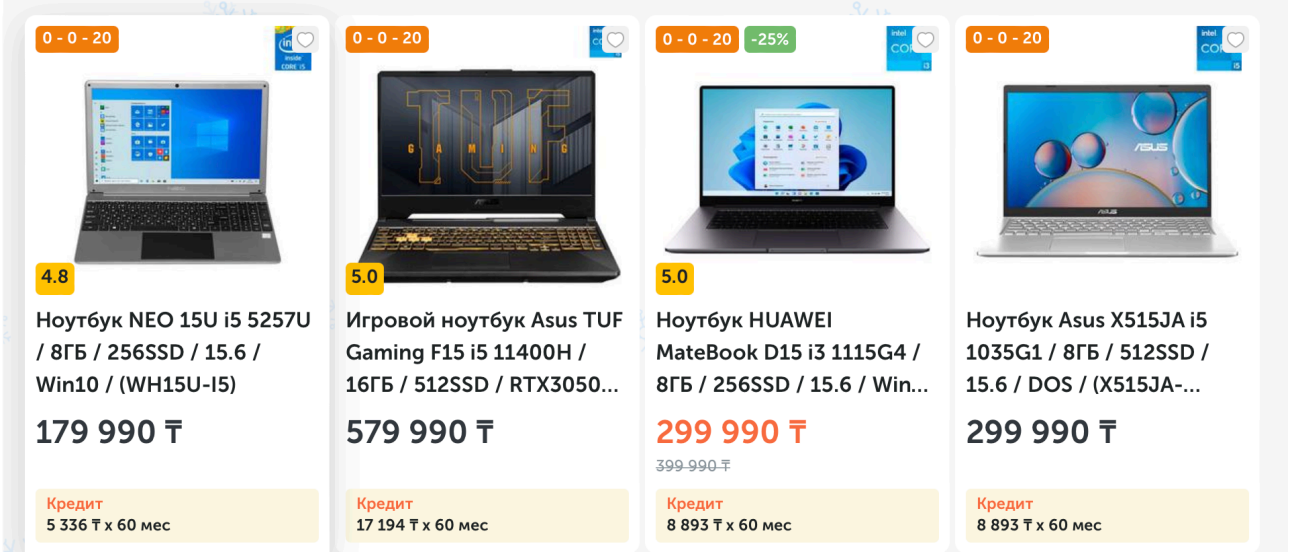
Detailed process

1. Data collection

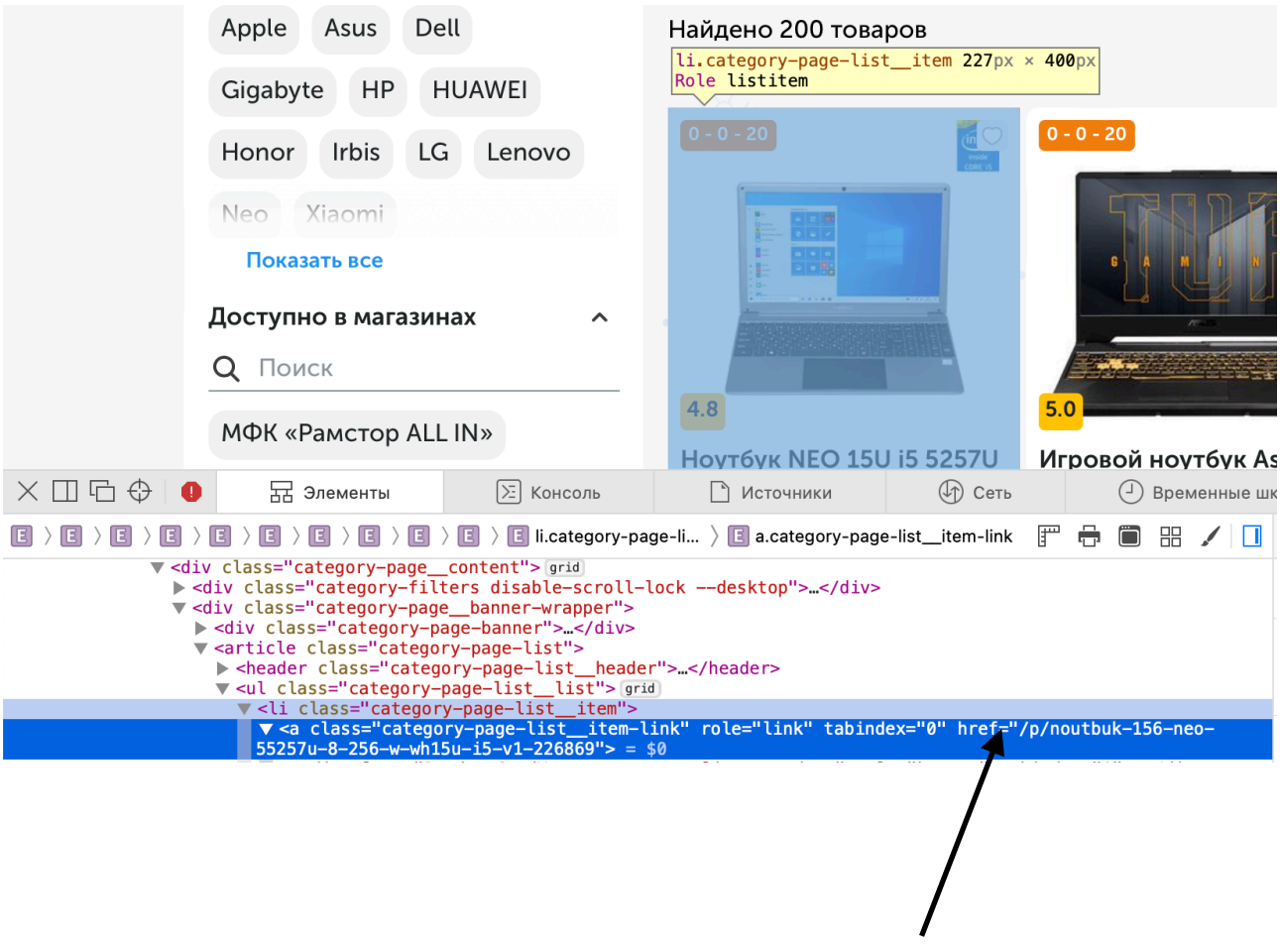
Tools: Pandas, urllib, BeautifulSoup, Selenium

The page from which the initial data is collected: https://www.technodom.kz/catalog/noutbuki-i-komp-jutery/noutbuki-i-aksessuary/noutbuki

This page contains a list of laptops. Why initial data? Because, on this page, there is only basic information such as the name of the laptop, a short list of features and price.

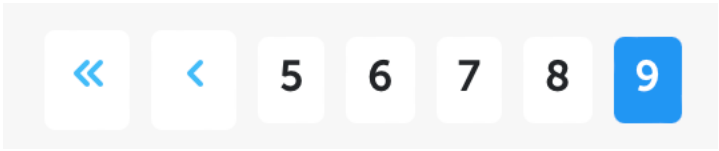


But we need a lot more information. To get them, we have to go into each of these laptops and collect data that is already stored there. We can do this with the help of links that are stored in each cell of the list of laptops.



In order to get detailed information about each laptop, we just need these links.

Also below we see that we have 9 such pages, with a collections of laptops.



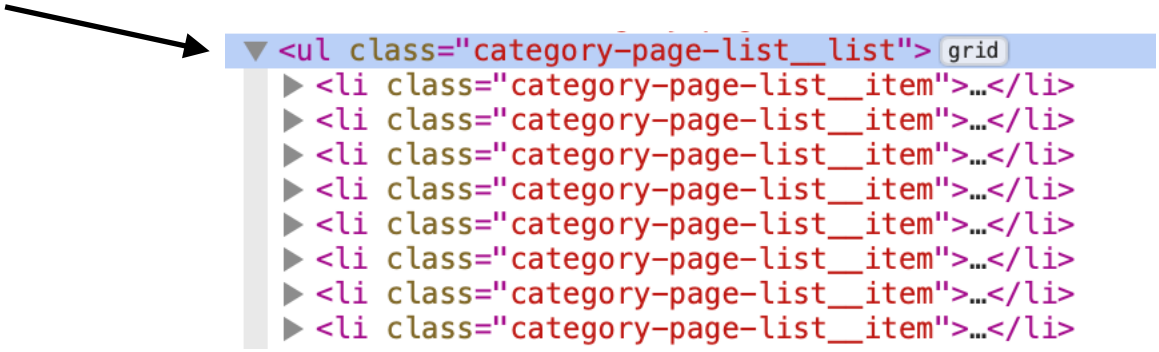
To collect all this, we need two for loops. One to iterate through the list, the other to iterate through 9 pages:

```
In [15]:
hrefs = []
for i in range(1, 10):
    url = ""
    if i == 1:
        url = f"https://www.technodom.kz/catalog/noutbuki-i-komp-jutery/noutbuki-i-aksessuary/noutbuki"
    else:
        url = f"https://www.technodom.kz/catalog/noutbuki-i-komp-jutery/noutbuki-i-aksessuary/noutbuki?page={i}
    response = Request(url, headers={'User-Agent': 'Mozilla/5.0'})
    html = urlopen(response)
    bs = BeautifulSoup(html, 'html.parser')
    ul = bs.find_all('ul', class_="category-page-list__list")
    li = ul[0].find_all('li')
    i = 0
    for item in li:
        if item.find('a') == None: continue
        hrefs.append(item.find('a').get('href'))
```

Iterating through 9 pages is not difficult. Because only one digit changes there.
For example:
Page 8: https://www.technodom.kz/catalog/noutbuki-i-komp-jutery/noutbuki-i-aksessuary/noutbuki?page=8

Page 9: https://www.technodom.kz/catalog/noutbuki-i-komp-jutery/noutbuki-i-aksessuary/noutbuki?page=9

Using Request tool, we open the desired link and store the response in the variable and read the site content using urlopen().
Now we can parse the content of the website using BeautifulSoup.
From now on, we have access to the html structure of the site. We need to find right tags:



The entire list is contained in the ul tag. ul contain li tags, which contain the links we need.

With for loop we iterate through these tags and store all tags in the ‘hrefs’ list:

```
for item in li:
    if item.find('a') == None: continue
    hrefs.append(item.find('a').get('href'))
```

Now we have 104 links:

```
17 print(len(hrefs))
```

```
104
```

The next job is to go to all these links and scrape up the detailed characteristics of the laptop.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException
import time
```

To do this, we will need the Selenium library. Let's explain why we need it.
It happens that not all html structure will be available after opening the page. In our case, a detailed characteristic will not be immediately displayed on the page. Because of this, scraping will not give us the necessary tags and information.

In order for the information we need to be displayed, we need to click on the «Все технические характеристики» button.

And it is in order to perform and automate such actions as scrolling and clicking, we will use Selenium.

And so, we begin the process of scraping detailed information. We have already imported the libraries we need. Now we need to create lists in which we will store our information.

```
videocards = []
memory_sizes = []
memory_types = []
rams = []
laptop_series = []
classes = []
resolutions = []
matrix_types = []
weights = []

cpu_performances = []
manufacturers = []
cpu_cores = []
cpu_frequency = []
cpu_series = []

prices = []
```

Unfortunately, it's not enough for us to just import some tools. For example WebDriver will have to be downloaded and installed separately. WebDriver is needed to control the browser.

```
options = webdriver.ChromeOptions()

driver = webdriver.Chrome(executable_path="chromedriver",
                          options=options)
```

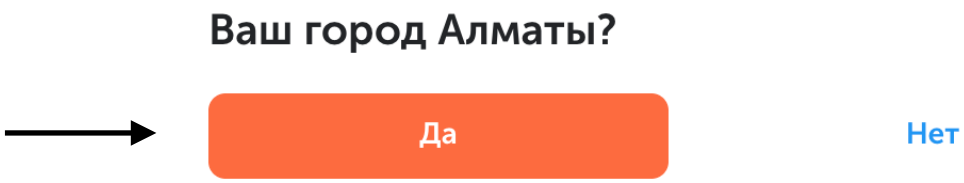
The whole process was placed in this function. We did this for sure, so that in the future it would be convenient to parse information piece by piece.

The parameters of this function (start, end) are the indexes of the links in the hrefs list. Thanks to this, we can parse different segments of links (it will be convenient to find errors and test).

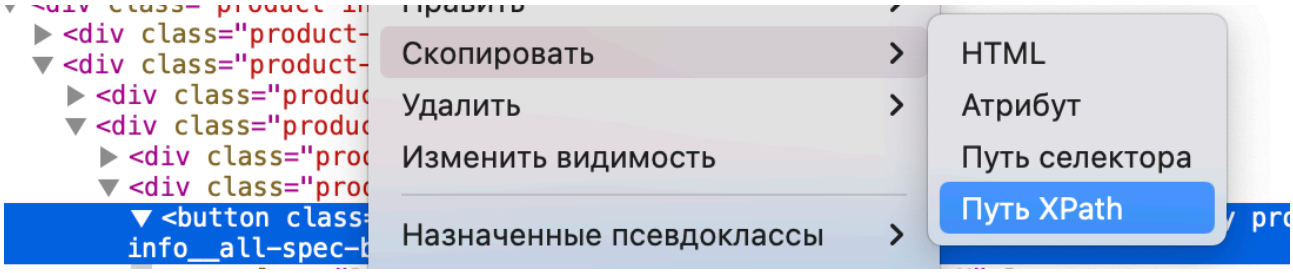
The next steps will be numbered. Items 1-8 explain the code below:

```
30 def fetchInformation(start, end):
31     for i in range(start, end):
32         # Send a get request to the url
33         laptop_url = f'https://www.technodom.kz/{hrefs[i]}'
34 |
35
36         if (i == 5 or i == 8) and start == 0:
37             continue
38
39         driver.get(laptop_url)
40         if i == 0:
41             time.sleep(10)
42
43             driver.find_element('xpath','/html/body/div[5]/div/div/div/button[1]').click()
44
45         driver.find_element(by=By.TAG_NAME, value='body').send_keys(Keys.PAGE_DOWN)
46
47         time.sleep(3)
```

1. Starting a loop to traverse the given links.
2. We open the variable and save the desired link there.
3. Using driver.get(laptop_url) we go to the page.
4. We use time.sleep() to let the page fully load, otherwise we simply won't find some tags. This time we give this value 10 seconds to wait for the element described in item 5 to appear.
5. We find the element using driver.find_element() and pass the XPath of the desired element. The desired element in this case is the "Да" button, which is located in such an element. This element pops up when we visit the site for the first time with browser.



XPath can be found in this way:



6. After finding the desired element, we call the click() function to click on this button. After that, the previous element "Ваш город" disappears and nothing else prevents us from continuing further.
7. Using the send_keys(Keys.PAGE_DOWN) function, we scroll down a little.
8. time.sleep() to wait for the page to load.
9. We skipped two iterations (if i == 5, i == 8 at line 34), because there were problems at this iterations during scraping.

Items 10-11 explain the code below.

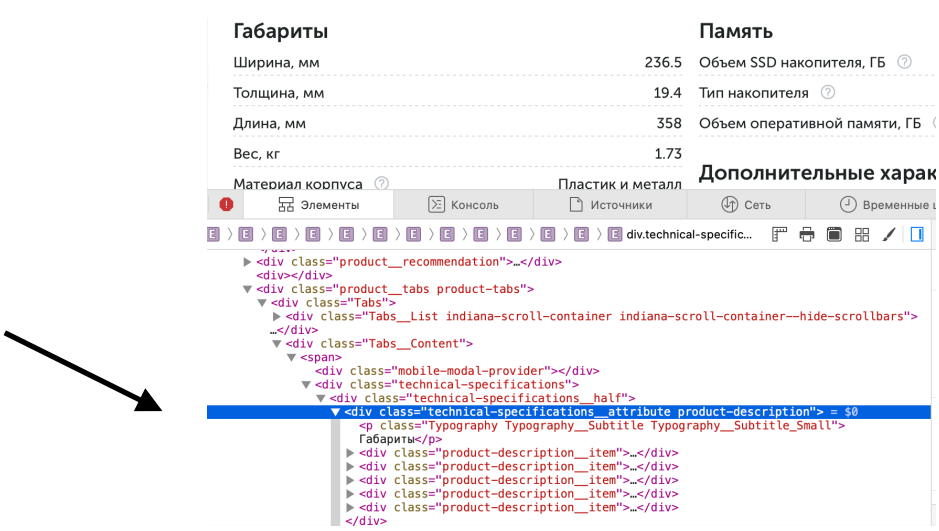
```
try:
    driver.find_element('xpath','/html/body/div[1]/section/main/section/article/div/div[1]/div[1]/div[2]

except NoSuchElementException:
    driver.find_element('xpath', '/html/body/div[1]/section/main/section/article/div/div[1]/div[1]/div[2

except:
    continue
```

10. Now we need to click on the «Все характеристики" button. To do this, we again find the desired element using find_element() and call the click() function (it did not fit in the photo).
11. We take it in a try except block. Since there are some cases where the XPath of the desired button is slightly different, even if it does not fit, then just go to the next iteration.

Items 12-15 explain the code below.

```
    bs = BeautifulSoup(driver.page_source, 'html.parser')

    desired_divs = bs.find_all('div', class_='technical-specifications__attribute product-description')

    price_div = bs.find('div', class_='product-info__prices product-prices')
    price_p = price_div.find('p')
    prices.append(price_p.text.replace('\xa0', '')[:-1])
```

12. All the necessary html structure is ready. We pass this into the BeautifulSoup object.
13. We find the necessary elements using the web inspector.
14. At this stage, we can immediately get the price information and add it to the prices list.
15. We have all the characteristics in the div, which is shown in the photo. We find them and save them to the desired_divs list.



Items 16-17 explain the code below.

```
65          for div in desired_divs:
66              if div.find('p').text == 'Основные характеристики':
67                  p_texts = div.find_all('p', class_='Typography product-description__right-text Typography_
68                  if len(p_texts) == 1:
69                       laptop_series.append("")
70                       classes.append(p_texts[0].text)
71                  else:
72                      laptop_series.append(p_texts[0].text)
73                      classes.append(p_texts[1].text)
74
75              if div.find('p').text == 'Память':
76                  p_texts = div.find_all('p', class_='Typography product-description__right-text Typography_
77                  memory_sizes.append(p_texts[0].text)
78                  memory_types.append(p_texts[1].text)
79                  rams.append(p_texts[2].text)
80
```

16.With a loop, we iterate through the found divs and check which item of the characteristics this element matches in order to add information to the corresponding list.
17. Добавляем найденную информацию в соответсвующие листы.

Items 16-17 explain the code below.

```
65          for div in desired_divs:
66              if div.find('p').text == 'Основные характеристики':
67                  p_texts = div.find_all('p', class_='Typography product-description__right-text Typography_
68                  if len(p_texts) == 1:
69                       laptop_series.append("")
70                       classes.append(p_texts[0].text)
71                  else:
72                      laptop_series.append(p_texts[0].text)
73                      classes.append(p_texts[1].text)
74
75              if div.find('p').text == 'Память':
76                  p_texts = div.find_all('p', class_='Typography product-description__right-text Typography_
77                  memory_sizes.append(p_texts[0].text)
78                  memory_types.append(p_texts[1].text)
79                  rams.append(p_texts[2].text)
80          |
81
82              if div.find('p').text == 'Видео':
83                   p_texts = div.find_all('p', class_='Typography product-description__right-text Typography
84                   videocards.append(p_texts[0].text)
85
86              if div.find('p').text == 'Дисплей':
87                   p_texts = div.find_all('p', class_='Typography product-description__right-text Typography
88                   resolutions.append(p_texts[0].text)
89                   matrix_types.append(p_texts[-1].text)
90
91              if div.find('p').text == 'Габариты':
92                   p_texts = div.find_all('p', class_='Typography product-description__right-text Typography
93                   weights.append(p_texts[-2].text)
94
95              if div.find('p').text == 'Процессор':
96                   p_texts = div.find_all('p', class_='Typography product-description__right-text Typography
97                   manufacturers.append(p_texts[0].text)
98
99                   if 'поколение' in p_texts[-1].text or 'Ryzen' in p_texts[-1].text:
100                      cpu_frequency.append(p_texts[-2].text)
101                      cpu_series.append(p_texts[-3].text)
102                      cpu_cores.append(p_texts[2].text)
103                  elif p_texts[0].text == 'Apple':
104                      cpu_frequency.append("")
105                      cpu_cores.append(p_texts[1].text)
106                      if 'M' in p_texts[-1].text:
107                          cpu_series.append(p_texts[-1].text)
108                      else:
109                          cpu_series.append(p_texts[-2].text)
110                  else:
111                      cpu_frequency.append(p_texts[-1].text)
112                      cpu_cores.append(p_texts[2].text)
113                      cpu_series.append(p_texts[-2].text)
```

18. We also continue to check and grab for the necessary data and add information to the required lists.
19. In some parts of the code, you also see nested if statements. For example, on 68 line, on lines from 99 to 113. These additional checks are made due to the fact that the structure of characteristics is not the same for all laptops. This may cause us to take inappropriate data. For example, some laptop has some characteristic recorded, another does not, etc. In order to try as much as possible to take exactly what you need, we have added these checks.

Item 20 explains the code below.

```
118 fetchInformation(0, len(hrefs))
```

20. We call our function and pass the parameter values.

Item 21 explains the code below.

```
15 laptops_dict = {'serie': laptop_series, 'cpu_manufacturer': manufacturers, 'core': cpu_cores,
16                 'cpu_frequency': cpu_frequency, 'cpu_series': cpu_series,'memory_size': memory_sizes,
17                 'memory_type': memory_types, 'RAM': rams, 'resolution': resolutions, 'matrix_type': matrix
18                 'videocard': videocards, 'weight': weights, 'laptop_class': classes, 'prices': prices}
```

21. We create a dictionary and transfer our lists there. They will form our columns in the dataframe.

Item 22-23 explains the code below.

```
1 df = pd.DataFrame(laptops_dict)
2 df.to_csv('laptops_info.csv')
```

22. We create a DataFrame where we put our dictionary.
23. This dataframe is converted to a csv file laptops_info.csv.

Our data has been collected! Let's move on to the next part.

2. Data cleaning

Tools: Pandas, numpy

```
import pandas as pd
```

As usual, we import the libraries that we need. For starters, a pandas will be enough for us.

We create our dataframe with csv file that we have after data collection.

```
df = pd.read_csv('laptops_info.csv', index_col=0)
df
```

	serie	cpu_manufacturer	core	cpu_frequency	cpu_series	memory_size	memory_type	RAM	resolution	matrix_type	videocard	weight	laptop_c
0	NaN	Intel	2	2.7	Intel Core i5	256.0	SSD	8.0	1920x1080 Full HD	IPS	Встроенная	1.80	Для ра и у
102	Gigabyte G7 ME	Intel	12	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	144 Hz	NGF RTX 3050 Ti	2.50	Для продаж

103 rows × 14 columns

Before starting, we need to examine our data to know what we have to clean.

```
df.isnull().sum()
```

```
serie               2
cpu_manufacturer    1
core                1
cpu_frequency      16
cpu_series          1
memory_size         1
memory_type         1
RAM                 1
resolution          1
matrix_type         1
videocard           1
weight              1
laptop_class        1
prices              0
dtype: int64
```

First, we'll start dealing with null values.

64					
65	Asus Vivobook S 16X	AMD	6	3.3	AMD Ryzen 5

In our dataframe, we can notice these empty row, in which most of the cells are empty. Such empty cells were obtained due to the fact that sometimes the information on the page was not fully loaded.

We will drop rows, that have null values in 12 columns. But I don't put to subset laptop_class and price because all rows has this value.

```
df = df.dropna(thresh=12).reset_index(drop=True)
df.shape
```

```
(102, 14)
```

This warning appears when we change the original dataframe. But this is what I want, so I will disable this warning.

```
pd.options.mode.chained_assignment = None  # default='warn'
```

Next, we can notice that the "CPU frequency" column is empty for macbooks. As it turned out, this characteristic is not written for them on the site.

39	Apple MacBook Air Series, Apple MacBook Air 2022	Apple	8		M2
40	Apple MacBook Pro Series, Apple MacBook Pro 2022	Apple	8		M2

Because of this, we just decided to look for this information on the Internet and put these values.

Processor frequency for M1 - 3.2
M2 processor frequency - 3.5

```
for i in range(0, len(df['cpu_series'])):

    if 'M1' in df['cpu_series'][i]:
        df['cpu_frequency'][i] = 3.2
    elif 'M2' in df['cpu_series'][i]:
        df['cpu_frequency'][i] = 3.5
```

Now we have less number of null values.

```
20 df.isnull().sum()
```

```
serie                1
cpu_manufacturer     0
core                 0
cpu_frequency        0
cpu_series           0
memory_size          0
memory_type          0
RAM                  0
resolution           0
matrix_type          0
videocard            0
weight               0
laptop_class         0
prices               0
dtype: int64
```

We have on row with one null value, let's drop it.

```
2 df = df.dropna().reset_index(drop=True)
3 df.isnull().sum()
```

```
serie                0
cpu_manufacturer     0
core                 0
cpu_frequency        0
cpu_series           0
memory_size          0
memory_type          0
RAM                  0
resolution           0
matrix_type          0
videocard            0
weight               0
laptop_class         0
prices               0
dtype: int64
```

We have cleared our data from null values!

While observing data, we found some floating point numbers in ‘cpu_series' and ‘matrix type’ columns. It can be because of different order and structure of characteristics in Technodom's website. Let’s remove this rows where ‘cpu_series' and ‘matrix type’ contains float numbers.

```
for i in range(0, len(df['cpu_series'])):
    try:
        df['cpu_series'][i] = float(df['cpu_series'][i])
        df = df.drop(i, axis=0).reset_index(drop=True)
    except:
        continue

#same with matrix_types
for i in range(0, len(df['matrix_type'])):
    try:
        df['matrix_type'][i] = float(df['matrix_type'][i])
        df = df.drop(i, axis=0).reset_index(drop=True)
    except:
        continue
```

We put it in a try except block. That is, if the element turns into a float without an error, we delete this row, in other case, we just go to next iteration.

Our data is almost clean. But some of our numeric values are actually strings. We need to convert them.

```
df.dtypes
```

```
serie                object
cpu_manufacturer     object
core                 object
cpu_frequency       float64
cpu_series           object
memory_size         float64
memory_type          object
RAM                 float64
resolution           object
matrix_type          object
videocard            object
weight              float64
laptop_class         object
prices                int64
dtype: object
```

Core actually should be float. The same principeThe same principle that we used when working with ‘cpu_series’ and ‘matrix_type’.

```
for i in range(0, len(df['core'])):
    try:
        df['core'][i] = float(df['core'][i])
    except:
        df = df.drop(i, axis=0).reset_index(drop=True)
```

Our columns as ‘videocard’ and ‘laptop’ class has data in russian. Let’s translate it.

```
for i in range(0, len(df['videocard'])):
    if df['videocard'][i] == 'Встроенная':
        df['videocard'][i] = 'Integrated'
```

We have classes like this -> 'Для работы и учебы', 'Для игр и продакшена', ‘Премиум', 'Повседневный серфинг', 'MacBook', 'Играй на Ultra'
But in the future, we want to make a classification of laptops by dividing them into gaming and non-gaming laptops.

The division will be like this:
'Для работы и учебы', 'Премиум', 'Повседневный серфинг', 'Macbook' - non-gaming
Others - gaming
I have added ‘Non-gaming’ to the ‘if’ statement, because if we re-run the cell, all cases will go to ‘else’ part and all laptops will become gaming.

```
for i in range(0, len(df['laptop_class'])):
    if df['laptop_class'][i] == 'Для работы и учебы' or df['laptop_class'][i] == 'Премиум' or df['laptop_c
        df['laptop_class'][i] = 'Non-gaming'
    else:
        df['laptop_class'][i] = 'Gaming'
```

Almost done.

While observing data, we found '144Hz' instead of matrix type in the dataframe.
It seems that there wasn't matrix type characteristic for this laptop on website and we get another characteristics value.
But we don't have a lot of data, so let's keep this laptop and change matrix type for this laptop with hands.

Also, we for one laptop there wasn't company name. This will mess up our visualization part a bit. Because the program will consider it for another company. So we'll just add it manually.

```
df['matrix_type'][len(df['matrix_type']) - 1] = 'IPS'


#And o laptop doesn't have company name in series of laptop
for i in range(0, len(df['serie'])):
    if 'MateBook' in df['serie'][i]:
        df['serie'][i] = 'Huawei ' +  df['serie'][i]
```

Also we can see that the weight of some laptops are very high, maybe it is a wrong data.

1.6
1.8
2.1
2.2
399

We need to find outliers and replace it. We will replace it with mode value.

For finding outliers will use this function that returns. to use lower and upper bounds.

```
def find_outliers(my_list):
    new_list = sorted(my_list)

    q1, q3 = np.percentile(new_list,[25,75])
#     print(q1, q3)
    iqr = q3 - q1
    lower_bound = q1 - (1.5 * iqr)
    upper_bound = q3 + (1.5 * iqr)

    return (lower_bound, upper_bound)
```

It remains only to loop through the list and replace the values that are below the lower bound or greater than the upper bound with the mode.

```
lower_bound, upper_bound = find_outliers(df['weight'])
print('Lower bound', lower_bound)
print('Upper bound', upper_bound)

mode = df['weight'].mode()
print('Mode', mode)

for i in range(0, len(df['weight'])):
    if df['weight'][i] < lower_bound or df['weight'][i] > upper_bound:
        df['weight'][i] = mode[1]

```

```
Lower bound 0.1999999999999995
Upper bound 3.4000000000000004
Mode 0    1.6
1    1.8
```

Our dataset is clear now!
We will convert it to another csv file ‘cleaned_data.csv’.

```
df.to_csv('cleaned_data.csv')
```

3. Classification

Tools: Pandas, matplotlib, seaborn, sklearn.

For classification, we will use the KNN algorithm.

Start from observing data.

```
df = pd.read_csv('cleaned_data.csv', index_col=0)
df
```

	serie	cpu_manufacturer	core	cpu_frequency	cpu_series	memory_size	memory_type	RAM	resolution	matrix_type	videocard	weight	laptop_clas
0	Asus TUF Gaming F15	Intel	6.0	2.6	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX3050	2.30	Gamir
1	HUAWEI MateBook D15	Intel	2.0	3.0	Intel Core i3	256.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.62	Non-gamir
2	Asus X515JA	Intel	4.0	1.0	Intel Core i5	512.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.80	Non-gamir
3	Asus X515JA	Intel	2.0	1.2	Intel Core i3	256.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.80	Non-gamir
4	Asus TUF Gaming F15	Intel	4.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF GTX 1650	2.30	Gamir
...	...	...	...	...	...	...	...	...	...	...	...	...	
92	Gigabyte U4 UD	Intel	4.0	2.5	Intel Core i7	512.0	SSD	16.0	1920x1080 Full HD	IPS	Integrated	1.00	Non-gamir
93	Gigabyte AERO 16 KE5	Intel	14.0	2.3	Intel Core i7	1000.0	SSD	16.0	3840x2400 WQUXGA	IPS	NGF RTX 3060	1.80	Gamir
94	Gigabyte G7 KE	Intel	6.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3060P	2.40	Gamir
95	Gigabyte G7 KE	Intel	12.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3060P	2.40	Gamir
96	Gigabyte G7 ME	Intel	12.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3050 Ti	2.50	Gamir

97 rows × 14 columns

We have a lot of categorical variables like in cpu_manufacturer, cpu_series, memory type, resolution, matrix_type, videocard
But our machine will not understand these values. Because of this we have to convert them to numeric values. For this, we will use LabelEncoding.
It would also be better for us if we work with a copy of the original.

We create the object of the LabelEncoder and use the fit_transform() function for changing categorical values to numeric.

```
from sklearn.preprocessing import LabelEncoder

df_copy = df.copy()
le = LabelEncoder()

# df['name'] = le.fit_transform(df['name'].values)
df_copy['videocard'] = le.fit_transform(df['videocard'].values)
df_copy['cpu_manufacturer'] = le.fit_transform(df['cpu_manufacturer'].values)
df_copy['cpu_series'] = le.fit_transform(df['cpu_series'].values)
df_copy['memory_type'] = le.fit_transform(df['memory_type'].values)
df_copy['resolution'] = le.fit_transform(df['resolution'].values)
df_copy['matrix_type'] = le.fit_transform(df['matrix_type'].values)
```

We will need to take two features. We have a lot of different features to work with, and because of this, it would be better if we create a function that will take different features as parameters and make classification with the help of them.

Our function will take a boolean value that determines whether or not to draw the plots. You will understand later why we need this. And the last parameter is the dataframe, since we will be passing different copies.

```
def make_classification(features_1, features_2, show_plot, df_):
```

We will create a dataframe with 3 columns. Two of them will be features and the 3rd one will be the answers.

```
    features_df = df_[[features_1, features_2, 'laptop_class']]
```

If our show_plot parameter is True, we will draw the plots.

```
    if show_plot == True:
        features_df.plot(kind='scatter', x=features_1, y=features_2, s=30)
        plt.show()

        sns.FacetGrid(features_df, hue="laptop_class", height=5).map(plt.scatter, features_1, features_2).
        plt.show()
```

After this, we need to take our X and y values. X values will contain our features and y values will contain answers.

```
    #For X values we don't need a species, because it is an answer
    X = features_df.iloc[:, 0:2]


    #For y we need only answers
    y = features_df.iloc[:, 2]
```

We split our data for testing and training. For testing we will take 30% of our data and other 70% will be taken for training. For this we have train_test_split() function from sklearn library.

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

For standardizing data we will use StandardScaler().

```
    stand_Sc = StandardScaler()
    X_train = stand_Sc.fit_transform(X_train)
    X_test = stand_Sc.transform(X_test)
```

Of course, for the algorithm, we need to determine the number of neighbors. The square root of the amount of data to test will be the number of neighbors.
If the result is even, we add 1 to make it odd. This will help us avoid ambiguities if the data is 2 by 2, 4 by 4 in, and so on.

```
    root = round(math.sqrt(len(y_test)))
    if root % 2 == 0:
        root += 1
```

KNeighborsClassifier implements classification based on voting by nearest k-neighbors.

```
    classifier = KNeighborsClassifier(n_neighbors=root, p=2, metric='euclidean')
    classifier.fit(X_train, y_train)
```

Make prediction and store it in y_pred.

```
    y_pred = classifier.predict(X_test)
```

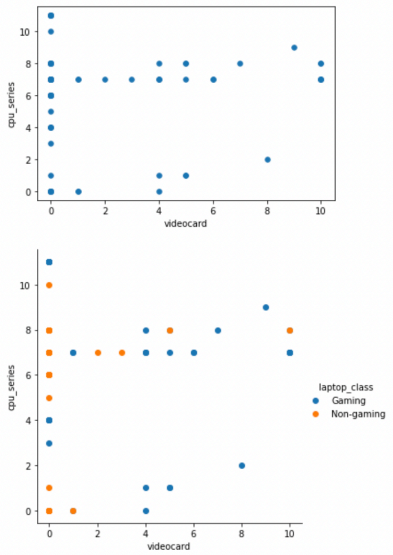
Our prediction is ready. Let's evaluate it.
accuracy_score() will compare actual values and our prediction for us. It takes items for comparing and boolean value normalize. If it False, it returns the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples.
We will need to just print it and return the result.

```
    percentage = accuracy_score(y_test, y_pred)
    if show_plot == True:
        print('Number of test cases:', len(y_test))
        print('Number of correct answers:', accuracy_score(y_test, y_pred, normalize=False))

        print('Percentage:', percentage)

    return percentage
```

It remains only to display the result on the screen and return it.

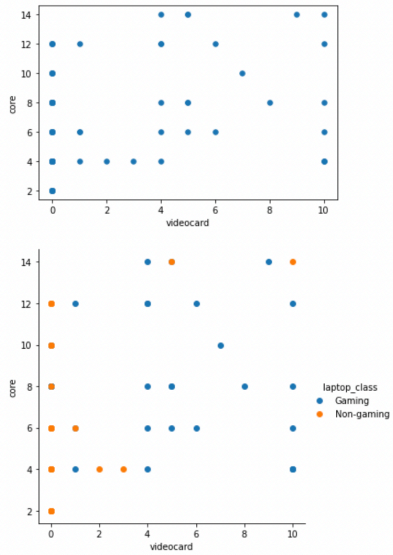
We took different features as an example. We also did visualization to see their distribution.
This was done in order to understand what are the key characteristics that distinguish a gaming laptop from a non-gaming laptop.

In [11]:
```
1 make_classification('videocard', 'cpu_series', True, df_copy)
```



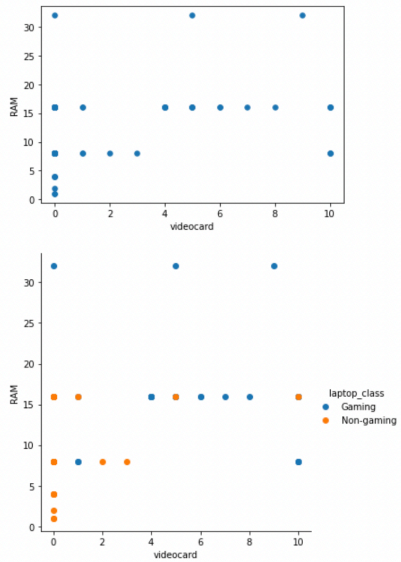

```
Number of test cases: 30
Number of correct answers: 27
Fraction: 0.9
```

In [12]:
```
1 make_classification('videocard', 'core', True, df_copy)
```



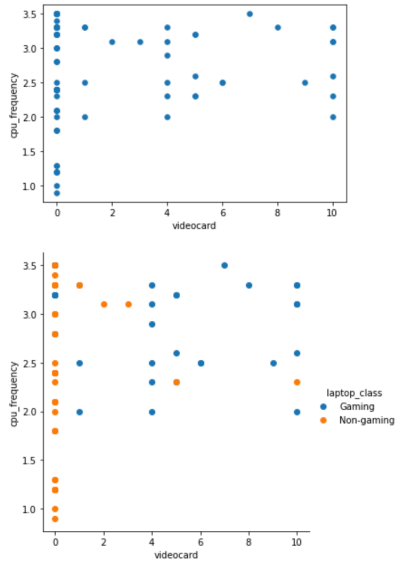

```
Number of test cases: 30
Number of correct answers: 27
Fraction: 0.9
```

In [9]:
```
1 make_classification('videocard', 'RAM', True, df_copy)
```




```
Number of test cases: 30
Number of correct answers: 23
Fraction: 0.7666666666666667
```

In [10]:
```
1 make_classification('videocard', 'cpu_frequency', True, df_copy)
```




```
Number of test cases: 30
Number of correct answers: 29
Fraction: 0.9666666666666667
```

It is difficult to judge by single results. So we decided to do the classification 100 times, save the data in lists and find the average value, which will show a more objective picture.

Creating lists, that will hold the results:

```
videocard_ram_class = []
videocard_frequency_class = []
videocard_series_class = []
videocard_core_class = []

series_core_class = []
ram_core_class = []
frequency_core_class = []
frequency_ram_class = []
frequency_series_class = []
```

We create the for loop and call our make_classification() function 100 times and store the results in the appropriate lists.

```
for i in range(100):
    videocard_ram_class.append(make_classification('videocard', 'RAM', False, df_copy))
    videocard_frequency_class.append(make_classification('videocard', 'cpu_frequency', False, df_copy))
    videocard_series_class.append(make_classification('videocard', 'cpu_series', False, df_copy))
    videocard_core_class.append(make_classification('videocard', 'core', False, df_copy))
    series_core_class.append(make_classification('cpu_series', 'core', False, df_copy))
    ram_core_class.append(make_classification('RAM', 'core', False, df_copy))
    frequency_core_class.append(make_classification('cpu_frequency', 'core', False, df_copy))
    frequency_ram_class.append(make_classification('cpu_frequency', 'RAM', False, df_copy))
    frequency_series_class.append(make_classification('cpu_frequency', 'cpu_series', False, df_copy))
```

We have another function that will take a list as parameter and calculate the average.

```
def find_average(lst):
    sum = 0

    for num in lst:
        sum += num

    return sum / len(lst)
```

Displaying the result on the screen:

```
print('Features: Videocard and RAM', find_average(videocard_ram_class))
print('Features: Videocard and CPU frequency', find_average(videocard_frequency_class))
print('Features: Videocard and CPU series', find_average(videocard_series_class))
print('Features: Videocard and Cores', find_average(videocard_core_class))

print('Features: CPU series and Cores', find_average(series_core_class))
print('Features: RAM series and Cores', find_average(ram_core_class))
print('Features: CPU frequency series and Cores', find_average(frequency_core_class))
print('Features: CPU frequency series and RAM', find_average(frequency_ram_class))
print('Features: CPU frequency series and CPU series', find_average(frequency_series_class))
```

This is our result.

```
Features: Videocard and RAM 0.7503333333333332
Features: Videocard and CPU frequency 0.8826666666666668
Features: Videocard and CPU series 0.8903333333333336
Features: Videocard and Cores 0.863333333333333
Features: CPU series and Cores 0.7403333333333335
Features: RAM series and Cores 0.7116666666666669
Features: CPU frequency series and Cores 0.805333333333333
Features: CPU frequency series and RAM 0.7310000000000001
Features: CPU frequency series and CPU series 0.7260000000000002
```

Conclusion of our experiment:

The combination of video card + other element gives higher accuracy than other combinations.

Another experiment:
We wanted to experiment more, to change one moment in the column of video cards.
What happens if we take NGF RTX3050, NGF GTX 1650 and so on and change them just to «discrete»?

Making a copy of original dataframe and divide video card values to integrated and discrete:

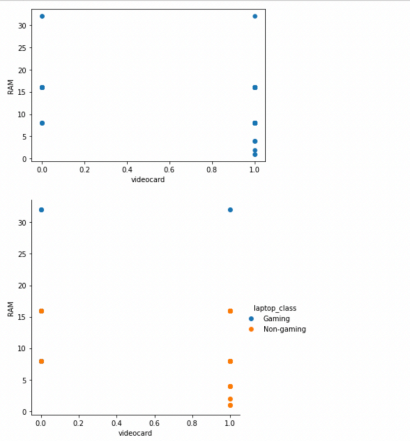
```
2  df_copy_2 = df.copy()
3
4  for i in range(0, len(df_copy_2['videocard'])):
5      if df_copy_2['videocard'][i] != 'Integrated':
6          df_copy_2['videocard'][i] = 'Discrete'
```

Again, we need to make a label encoding so that the machine understands these values.

```
3  df_copy_2['videocard'] = le.fit_transform(df_copy_2['videocard'].values)
4  df_copy_2['cpu_series'] = le.fit_transform(df_copy_2['cpu_series'].values)
```

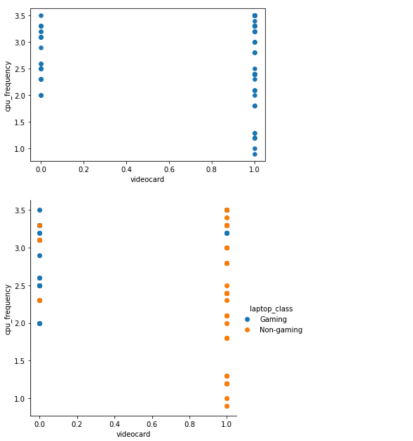
Make some of them with plots.

```
In [25]: 1 make_classification('videocard', 'RAM', True, df_copy_2)
```




```
Number of test cases: 30
Number of correct answers: 25
Fraction: 0.8333333333333334
```

```
In [26]: 1 make_classification('videocard', 'cpu_frequency', True, df_copy_2)
```




```
Number of test cases: 30
Number of correct answers: 27
Fraction: 0.9
```

Then we repeat process that we did previously.

```python
videocard_ram_class = []
videocard_frequency_class = []
videocard_series_class = []
videocard_core_class = []

def find_average(lst):
    sum = 0

    for num in lst:
        sum += num

    return sum / len(lst)


for i in range(100):
    videocard_ram_class.append(make_classification('videocard', 'RAM', False, df_copy_2))
    videocard_frequency_class.append(make_classification('videocard', 'cpu_frequency', False, df_copy_2))
    videocard_series_class.append(make_classification('videocard', 'cpu_series', False, df_copy_2))
    videocard_core_class.append(make_classification('videocard', 'core', False, df_copy_2))

print('Features: Videocard and RAM', find_average(videocard_ram_class))
print('Features: Videocard and CPU frequency', find_average(videocard_frequency_class))
print('Features: Videocard and CPU series', find_average(videocard_series_class))
print('Features: Videocard and Cores', find_average(videocard_core_class))
```

Let's see the results this time:
Result with after changing to the «discrete».

```
Features: Videocard and RAM 0.7503333333333331
Features: Videocard and CPU frequency 0.8869999999999991
Features: Videocard and CPU series 0.9020000000000001
Features: Videocard and Cores 0.8893333333333332
```

Previous result:

```
Features: Videocard and RAM 0.7503333333333332
Features: Videocard and CPU frequency 0.8826666666666668
Features: Videocard and CPU series 0.8903333333333336
Features: Videocard and Cores 0.863333333333333
```

We can see that results are slightly better in some cases. The biggest difference was shown by the video card + cores.

4. Data vizualization

Tools: Pandas, Numpy, Matplotlib, Seaborn, Plotly

This is our data frame after cleaning. It has all the technical characteristics of laptops, such as display, memory, processor

Unnamed: 0	serie	cpu_manufacturer	core	cpu_frequency	cpu_series	memory_size	memory_type	RAM	resolution	matrix_type	videocard	weight	laptop_class	prices
0	Asus TUF Gaming F15	Intel	6.0	2.6	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX3050	2.30	Gaming	579990
1	HUAWEI MateBook D15	Intel	2.0	3.0	Intel Core i3	256.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.62	Non-gaming	299990
2	Asus X515JA	Intel	4.0	1.0	Intel Core i5	512.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.80	Non-gaming	299990
3	Asus X515JA	Intel	2.0	1.2	Intel Core i3	256.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.80	Non-gaming	219990
4	Asus TUF Gaming F15	Intel	4.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF GTX 1650	2.30	Gaming	489990
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
91	Gigabyte U4 UD	Intel	4.0	2.5	Intel Core i7	512.0	SSD	16.0	1920x1080 Full HD	IPS	Integrated	1.00	Non-gaming	476390
92	Gigabyte AERO 16 KE5	Intel	14.0	2.3	Intel Core i7	1000.0	SSD	16.0	3840x2400 WQUXGA	IPS	NGF RTX 3060	1.80	Gaming	1317890
93	Gigabyte G7 KE	Intel	6.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3060P	2.40	Gaming	649490
94	Gigabyte G7 KE	Intel	12.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3060P	2.40	Gaming	719490
95	Gigabyte G7 ME	Intel	12.0	2.5	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX 3050 Ti	2.50	Gaming	654490

96 rows × 17 columns

series, processor frequency and so on.

- A new column with the header "Company" has been added for the name of the laptop manufacturer to make it convenient to count the number of laptops in a particular company. We determined this by splitting and taking the first string from the "series" column.

```
In [88]: df['Company'] = df.serie.str.split(" ", n=1).str[0]
         df
```

Out[88]:

serie	cpu_manufacturer	core	cpu_series	memory_size	memory_type	RAM	resolution	matrix_type	videocard	weight	laptop_class	prices	Company
Asus TUF Gaming F15	Intel	6.0	Intel Core i5	512.0	SSD	16.0	1920x1080 Full HD	IPS	NGF RTX3050	2.30	Gaming	579990	Asus
HUAWEI MateBook D15	Intel	2.0	Intel Core i3	256.0	SSD	8.0	1920x1080 Full HD	IPS	Integrated	1.62	Non-gaming	299990	HUAWEI

I. In the first visualization, we decided to count the number of laptops in each company.

First, we took the full name of the company, dropping duplicates from the "Company" column, and then converted this data frame into an list

```
In [151]: company_name_list = list(df['Company'].drop_duplicates())
          company_name_list
Out[151]: ['Asus', 'HUAWEI', 'HP', 'Lenovo', 'Apple', 'LG', 'Gigabyte']
```

To count the number of laptops we've created dictionary

```
laptops_count_dict = {}
```

According to the code given below, we filled out this dictionary

```
laptops_count_dict = {}

for i in company_name_list:
    company = df[df['Company'] == i]
    laptops_count_dict[i] = len(company)

laptops_count_dict
```

```
Asus : 40
Huawei : 5
Hp : 14
Lenovo : 14
Apple : 15
Lg : 2
Gigabyte : 6
```

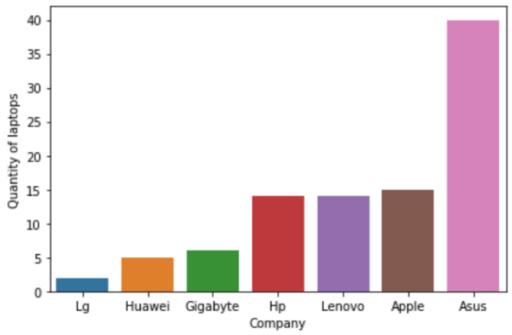
Then we converted it to data frame

```
laptop_df = pd.DataFrame(laptops_count_dict.items(), columns=['Company', 'Quantity'])
laptop_df = laptop_df.sort_values("Quantity")
laptop_df
```

Company	Quantity
Lg	2
Huawei	5
Gigabyte	6
Hp	14
Lenovo	14
Apple	15
Asus	40

And so, we begin the visualization process to determine the number of laptops of each company. To begin with, we import libraries Matplotlib and Seaborn

```
sns.barplot(x=laptop_df['Company'], y = laptop_df['Quantity'])
plt.xlabel('Company')
plt.ylabel('Quantity of laptops')
plt.tight_layout()
plt.show()
```



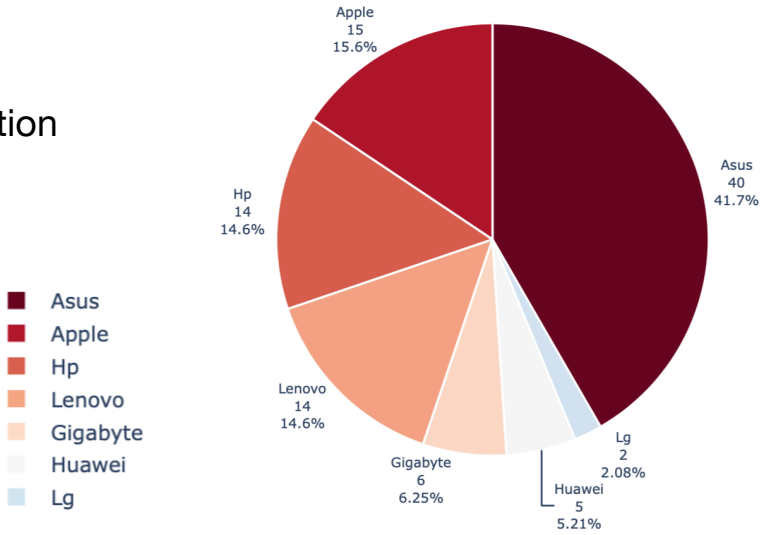
We made another visualization for this situation with pie chart. Because a pie chart instantly gives an idea of the ratios when multiple sectors are used as measurements. For this we imported library plotly

```
import plotly.express as px
```

To create a pie chart in plotly express, we used **px.pie()** method. Also changed the color of the pie chart, we used **color_discrete_sequence** parameter.

```
fig = px.pie(values=laptop_df['Quantity'], names=laptop_df['Company'],
             color_discrete_sequence=px.colors.sequential.RdBu)

fig.update_traces(textposition='outside',
                  textinfo='percent+label+value',
                  marker=dict(line=dict(color='#FFFFFF', width=2)),
                  textfont_size=12)
fig.show()
```

And so we got such a beautiful visualization



II. Next, we have visualization on the most expensive and cheapest laptops

We have created a static grouped bar chart with multiple (double) columns using Python libraries: Pandas, Matplotlib and Seaborn.

Data Preparation

We have created an list with company names and a dictionary for expensive and cheap laptops

```
company_name = list(df['Company'].drop_duplicates())

expensive_laptops = {}
cheap_laptops = []
```

We have determined the maximum and minimum price of each company according to the code given below

```
for i in company_name:
    price = df[df['Company'] == i]['prices'].max()
    expensive_laptops[i] = price
    price = df[df['Company'] == i]['prices'].min()
    cheap_laptops.append(price)
```

→

```
'Asus': 1629990,
'HUAWEI': 599990,
'HP': 774990,
'Lenovo': 629990,
'Apple': 1783990,
'LG': 1069990,
'Gigabyte': 1317890
```

Then we created a data frame with this dictionary and list

```
laptop_prices = pd.DataFrame(expensive_laptops.items(), columns=['Company', 'Expensive price'])
laptop_prices['Cheap price'] = pd.DataFrame(cheap_laptops)
```

The output will look like this:

	Company	Expensive price	Cheap price
0	Asus	1629990	161490
1	HUAWEI	599990	299990
2	HP	774990	314990
3	Lenovo	629990	189990
4	Apple	1783990	699990
5	LG	1069990	969990
6	Gigabyte	1317890	476390

We'll use this piece of data frame with multiple columns to create our Matplotlib bar chart.

Plotting:

Here is a list of variables that were used in our code.

```
first_bar = laptop_prices['Expensive price']
first_bar_label = 'The price of an expensive laptop'
first_bar_color = '#32628d'
second_bar = laptop_prices['Cheap price']
second_bar_label = 'The price of a cheap laptop'
second_bar_color = '#cde01d'
labels = laptop_prices['Company']
width = 0.4  # the width of the bars
plot_title = 'The most expensive and cheapest prices of laptops'
title_size = 18
```

We've created a Matplotlib chart with double bars in 4 steps.

1. Create subplots

   First, sorted data for plotting:

   ```
   laptop_prices.sort_values(by='Expensive price', inplace=True, ascending=True)
   ```

   Next, drew a figure with a subplot:

   ```
   fig, ax = plt.subplots(figsize=(10,6), facecolor=(.94, .94, .94))
   ```

2. Create bars:

   ax.barh() would draw horizontal bar plots. We've used Matplotlib barh, and our chart has a horizontal layout.
   We're also using set_major_formatter() to format ticks with commas (like 1,500 or 2,000).

   ```
   # Plot double bars
   y = np.arange(len(labels))  # Label locations
   ax.barh(y + width/2, first_bar, width, label=first_bar_label, color=first_bar_color)
   ax.barh(y - width/2, second_bar, width, label=second_bar_label, color=second_bar_color)

   # Format ticks
   ax.xaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))
   ```

3. Set title:

   pad=20 sets the title's padding, and .35 sets its left margin.

   ```
   # Set title
   title = plt.title(plot_title, pad=20, fontsize=title_size)
   title.set_position([.40, 1])
   ```

4. Create bar labels:

   Since we're created a Python bar graph with labels, we've defined label values and label position.

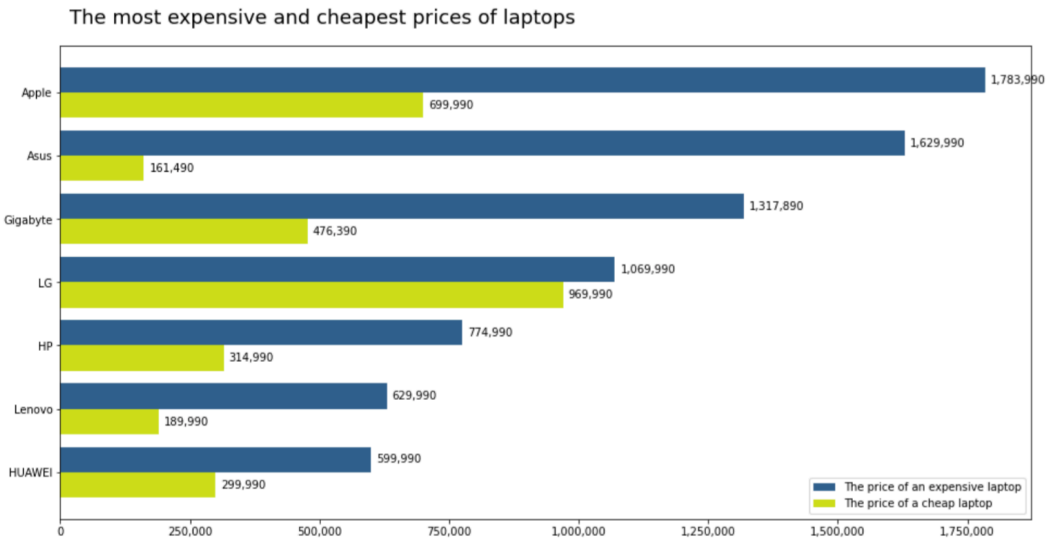
To label bars with values, we've used the following function: →

```
# Create labels
rects = ax.patches
for rect in rects:
    # Get X and Y placement of label from rect.
    x_value = rect.get_width()
    y_value = rect.get_y() + rect.get_height() / 2
    space = 5
    ha = 'left'
    if x_value < 0:
        space *= -1
        ha = 'right'
    label = '{:,.0f}'.format(x_value)
    plt.annotate(
        label,
        (x_value, y_value),
        xytext=(space, 0),
        textcoords='offset points',
        va='center',
        ha=ha)
```

Finally, we set y-axis labels and the legend:

```
# Set y-labels and legend
ax.set_yticklabels(labels)
ax.legend()

# To show each y-label, not just even ones
plt.yticks(np.arange(min(y), max(y)+1, 1.0))
```

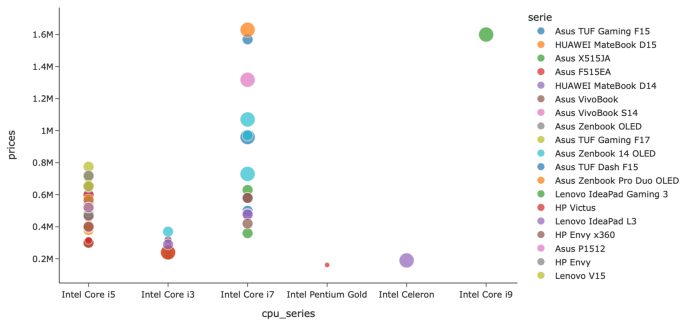
visualization:

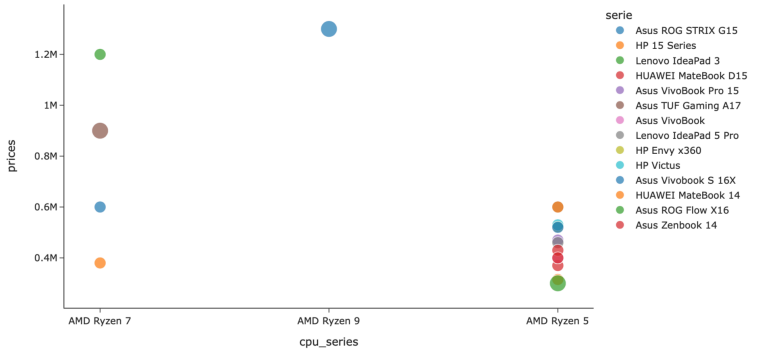


III. Our latest visualization is designed to show the relationship between prices and serial processors of laptops.

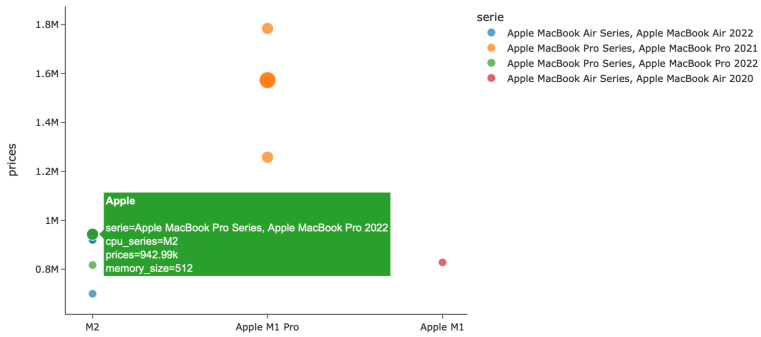
We chose a Scatterplot for visualization because it allows us to compare several data sets with each other, taking into account their features and relationships.

We made a separate visualization for each processor (Intel, AMD, Apple)





We have also added a few parameters. For example, the size of the data points depends on the memory size, and when you hover the cursor over these points, it shows you all the characteristics of the current laptop.



At the beginning, we imported a library for our graph

```
import plotly.express as px
```

Then we identified all the processors from our data frame by dropping duplicates. After this and converted them into an list

```
cpu = df['cpu_manufacturer'].drop_duplicates()
cpu_manufacturer = list(cpu)
```

```
['Intel', 'AMD', 'Apple']
```

We used for loop to get the data frame one by one, and used them for visualization.
On plotli.express, using the scatter() method, we created a graph. It has parameters x, y: graph labels; variables that define positions on the x and y axes. On x we put the heading 'prices', and on y the heading ‘cpu_series’. For the size parameter, we set the header 'memory_size' from the data frame

```
for i in cpu_manufacturer:
    cpu = df[df["cpu_manufacturer"] == i]
    fig = px.scatter(cpu, y = 'prices', x = 'cpu_series', template= "simple_white",
           hover_name="Company", color = "serie", size = "memory_size", size_max = 15)
    fig.show()
```