

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

## КУРСОВА РОБОТА

з дисципліни «Аналіз даних в інформаційних системах»

на тему: «Прогнозування ціни на ноутбук на основі його ключових компонентів»

Студента 2 курсу групи ПІ-22

Спеціальності: 121

«Інженерія програмного забезпечення»

Нижника Дмитра Сергійовича

«ПРИЙНЯВ» з оцінкою

---

доц. Ліхоузова Т.А. / доц. Олійник Ю.О.

---

Підпис

Дата

Київ - 2024 рік

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна Аналіз даних в інформаційно-управляючих системах

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 2 Група ІІ-22

Семестр 4

## ЗАВДАННЯ

на курсову роботу студента

Нижника Дмитра Сергійовича

---

1. Тема роботи Прогнозування ціни на ноутбук на основі його ключових компонентів

---

2. Строк здачі студентом закінченої роботи 04.06.2024

---

3. Вхідні дані до роботи методичні вказівки до курсової роботи, обрані дані з сайтів  
<https://www.kaggle.com/datasets/siddiquifaiznaeem/laptop-sales-price-prediction-dataset-2024>

---

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

1. Вступ

2. Постановка задачі

3. Аналіз предметної області

4. Робота з даними

5. Інтелектуальний аналіз даних

6. Висновки

7. Перелік посилань

8. Додаток А – програмний код обробки даних

9. Додаток Б – програмний код аналізу даних

---

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

---

---

---

---

6. Дата видачі завдання 12.02.2024

---

## КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	12.02.2024	
2.	Визначення зовнішніх джерел даних	23.03.2024	
3.	Пошук та вивчення літератури з питань курсової роботи	06.04.2024	
4.	Розробка моделі сховища даних	14.04.2024	
4.	Розробка ETL процесів	14.04.2024	
5.	Обґрунтування методів інтелектуального аналізу даних	20.05.2024	
6.	Застосування та порівняння ефективності методів інтелектуального аналізу даних	22.05.2024	
7.	Підготовка пояснювальної записки	27.05.2024	
8.	Здача курсової роботи на перевірку	04.06.2024	
9.	Захист курсової роботи	06.06.2024	

Студент

\_\_\_\_\_  
(підпис)

Нижник Дмитро Сергійович

\_\_\_\_\_  
(прізвище, ім'я, по батькові)

Керівник

\_\_\_\_\_  
(підпис)

доц. Ліхоузова Т.А

\_\_\_\_\_  
(прізвище, ім'я, по батькові)

Керівник

\_\_\_\_\_  
(підпис)

доц. Олійник Ю.О.

\_\_\_\_\_  
(прізвище, ім'я, по батькові)

"06" червня 2024 р.

## АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 77 сторінок, 50 рисунків, 2 таблиці, 6 посилань.

Об'єкт дослідження: інтелектуальний аналіз даних.

Предмет дослідження: створення програмного забезпечення, що проводить аналіз даних з подальшим прогнозуванням та графічним відображенням результатів.

Мета роботи: пошук та збір даних, обробка та підготовка цих даних до аналізу, аналіз та реалізація ПЗ, що використовує отримані дані для подальшого аналізу та прогнозування результату.

Дана курсова робота включає в себе: постановку задачі, аналіз предметної області, опис процесу роботи із даними, аналіз обраних методів аналізу, опис створення програмного забезпечення для інтелектуального аналізу даних, їх графічного відображення та прогнозування за допомогою різних моделей.

ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, DATASET, ОЧИСТКА ДАНИХ, РЕГРЕСІЯ, LINEAR REGRESSION, RIDGE REGRESSION, RANDOM FOREST.

## ЗМІСТ

ВСТУП.....	6
1. ПОСТАНОВКА ЗАДАЧІ.....	7
2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
3. РОБОТА З ДАНИМИ .....	10
3.1. Завантаження та дослідження даних .....	10
3.2. Заповнення пропусків у даних .....	13
3.3. Виправлення помилок у даних .....	19
4. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ.....	25
4.1. Обґрунтування вибору факторів.....	25
4.2. Обґрунтування вибору методів аналізу даних .....	38
4.3. Аналіз отриманих результатів для моделі Linear Regression.....	40
4.4. Аналіз отриманих результатів для моделі Ridge Regression .....	42
4.5. Аналіз отриманих результатів для моделі Random Forest .....	45
4.6. Порівняння отриманих результатів моделей .....	48
ВИСНОВКИ.....	50
ПЕРЕЛІК ПОСИЛАНЬ .....	51
ДОДАТОК А - ПРОГРАМНИЙ КОД ОБРОБКИ ДАНИХ.....	52
ДОДАТОК Б - ПРОГРАМНИЙ КОД АНАЛІЗУ ДАНИХ.....	64

## ВСТУП

У сучасному світі ноутбуки стали невід'ємною частиною життя багатьох людей. Вони відіграють важливу роль у професійній сфері, сфері розваг (відеоігри, кіно, книги тощо), навчанні, спілкуванні і особистому життю в цілому. Необхідність у портативній електротехніці пояснюється швидким темпом життя сучасної людини. У країнах з високим темпом життя (Японії, наприклад) споживачі віддають перевагу покупці портативних девайсів: портативні консолі, портативні ПК, ноутбуки та нетбуки, адже техніка, яку можна використовувати лише вдома, вповільнює життя цих людей.

Швидкий темп технологічного прогресу призвів до появи значного розмаїття моделей ноутбуків, що ускладнює вибір відповідного пристрою. Різні моделі відрізняються за своїми характеристиками, ціною та призначенням. У продажу представлено широкий вибір ноутбуків із різними компонентами, починаючи від параметрів екрану і закінчуючи моделями процесора та відеокарти.

Лінійки ноутбуків у різних виробників постійно поповнюються новими моделями з різними конфігураціями компонентів та цінами. У зв'язку з цим, передбачення ціни на ноутбук на основі його ключових характеристик є актуальним завданням, здатним допомогти споживачам зробити обґрунтований вибір, а продавцям – оптимізувати свою цінову політику.

Мета даного дослідження полягає у розробці моделі, здатної передбачати ціну ноутбука на основі його характеристик. Для досягнення цієї мети будуть використані методи машинного навчання, такі як лінійна регресія, гребенева регресія та випадкові ліси. Аналіз буде проведено з використанням бібліотек мови програмування Python 3, включаючи такі інструменти, як Pandas, NumPy, Matplotlib, Seaborn та Scikit-Learn.

Очікується, що результати даного дослідження дозволять виявити ключові фактори, що впливають на вартість ноутбуків, та забезпечити ефективну методику прогнозування ціни на основі характеристик кожного пристрою.

## 1. ПОСТАНОВКА ЗАДАЧІ

Мета цієї курсової роботи полягає у передбаченні ціни на ноутбук на основі його характеристик. Під час виконання роботи необхідно підготувати дані до аналізу, провести повноцінний інтелектуальний аналіз та забезпечити максимально можливу точність передбачення.

Виконання курсової роботи проходить у декілька етапів:

- 1) Аналіз предметної області
- 2) Робота з даними
- 3) Інтелектуальний аналіз даних
- 4) Формування висновків

На першому етапі виконання курсової роботи необхідно повноцінно дослідити та описати предметну область, її особливості та тенденції.

На етапі роботи з даними досліднику необхідно завантажити дані, підготувати їх до аналізу та дослідити. Дані треба очистити від зайвих даних та помилкових записів, пропуски у даних необхідно опрацювати та заповнити, помилки у даних необхідно виправити, помилки у типах даних, за їх наявності, також слід виправити. Ціль первинної обробки даних – підвищення точності та загальної якості аналізу. Дослідження наявних даних допоможе зрозуміти їх суть та вміст, структуру, ідентифікувати можливі проблеми та тенденції. Ціль такого дослідження – прийняття подальших рішень для коректного аналізу.

Етап інтелектуального аналізу включає в себе аналіз та вибір потенційних факторів (предикторів), які впливають на відгук (цільову змінну), виявлення закономірностей, вибір моделі прогнозування, її оцінку та покращення. Вибір моделі та предикторів слід описати та обґрунтувати.

Після виконання всіх етапів аналізу, слід зробити висновки. Вони мають включати в себе зведення основних результатів аналізу, підсумок виявлених закономірностей, оцінку найкращої моделі, оцінку значимості аналізу.

Весь процес виконання курсової роботи описується в рамках пояснювальної записки. Код виконується в спеціальних блокнотах застосунку Google Colab.

## 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сучасні ноутбуки – це не просто електронні пристрої для роботи та розваг, це справжні технологічні шедеври, що поєднують найновіші комп'ютерні компоненти та інноваційні розробки. Вони стали невід'ємною частиною повсякденного життя багатьох людей, особливо школярів та студентів, офісних працівників, програмістів та геймерів.

Ключові елементи ноутбука – це те, що визначає його функціональність, призначення, продуктивність та, звичайно ж, ціну. Відомі на весь світ виробники електротехніки змагаються між собою на ринку ноутбуків та нетбуків, виробляючи десятки різних моделей ноутбуків із різними конфігураціями, ціною та призначенням.

Основні компоненти ноутбуків наступні:

- 1) Центральний процесор
- 2) Оперативна пам'ять
- 3) Графічний процесор
- 4) Накопичувач
- 5) Акумулятор
- 6) Дисплей
- 7) Роз'єми
- 8) Клавіатура і тачпад
- 9) Камера та мікрофон
- 10) Система охолодження

Центральний процесор (CPU) – це серце ноутбука, що багато в чому визначає його обчислювальну потужність, енергоспоживання та час роботи від акумулятора. Процесори різних виробників (Intel, AMD, MediaTek), різних лінійок (Core i5, Ryzen 5) та різних моделей (Core i5 1235U, Ryzen 5 5700U) мають різні характеристики, такі як кількість ядер та потоків, тактова частота та кеш-пам'ять. Високопродуктивні процесори забезпечують плавну роботу та швидкодію. Від вибору процесору залежить вибір чипсету, типу ОЗП та материнської плати загалом.



Оперативна пам'ять (RAM) – це важливий елемент ноутбуку. Ця пам'ять призначена для тимчасового зберігання даних і миттєвого доступу до них. ОЗП набагато швидше за звичайний накопичувач, що підвищує швидкодію системи. Великий обсяг ОЗП дозволяє запускати велику кількість програм одночасно без втрати продуктивності, а такі параметри, як тактова частота та таймінги (memory timings) визначають швидкодію ОЗП.

Графічний процесор (GPU) – це ще один дуже важливий компонент ноутбуку. Він необхідний для обробки графіки та відео. Графічні процесори можуть бути інтегрованими або дискретними, можуть мати власну відеопам'ять або не мати її. В залежності від продуктивності відеоядра, ноутбук матиме різні параметри продуктивності в відеоіграх, редагуванні відео та 3D-модельованні.

Накопичувач відповідає за зберігання даних на ноутбуці. Накопичувачі бувають різних форматів та типів. SSD-накопичувачі забезпечують високу швидкість завантаження та швидкодію системи, тоді як HDD пропонують більший обсяг зберігання за нижчою ціною. В ноутбуках зазвичай ставлять SSD або комбінують SSD та жорсткий диск.

Акумулятор багато в чому визначає час автономної роботи пристрою. Автономність ноутбука залежить від ємності батареї та енергоефективності компонентів. Чим більша ємність акумулятора і чим менше енергоспоживання його компонентів – тим довше девайс здатен працювати без зарядки. Для мобільності та зручності використання тривалий час роботи без підзарядки дуже важливо. Ємність акумулятора рідко є типовою.

Параметри дисплею визначають якість відображення. Основними характеристиками є роздільна здатність, розмір матриці, густина пікселів на квадратний дюйм, тип панелі (TN, IPS, OLED) та частота оновлення.

Інші характеристики ноутбука, такі як кількість роз'ємів, розмір клавіатури або якість мікрофону рідко є типовими або визначними, тому при аналізі можуть не враховуватися.

### 3. РОБОТА З ДАНИМИ

#### 3.1. Завантаження та дослідження даних

Для виконання курсової роботи було обрано 1 джерело даних, а саме dataset “Laptop sales price prediction 2024” [1] на веб-ресурсі Kaggle. Він містить інформацію про характеристики ноутбуків, представлених у магазині цифрової електроніки, рейтинг у 5-бальній шкалі та ціну в індійських рупіях.

Перед початком роботи дані було завантажено у блокнот Google Colab. Перші 5 записів показано на рисунку 3.1.1. На цьому етапі було помічено пропуски в даних.

Unnamed: 0	Name	Brand	Price	Rating	Processor_brand	Processor_name	Processor_variant	Processor_gen	Core_per_processor	Graphics_name	Graphics_brand	Graphics_GB	Graphics_
0	HP Victus 15-fb015/AX Gaming Laptop (AMD Ryzen...	HP	50399	4.30	AMD	AMD Ryzen 5	5600H	5.0	6.0	AMD Radeon RX 6500M	AMD	4.0	
1	Lenovo V15 G4 82YU00W7IN Laptop (AMD Ryzen 3 ...	Lenovo	26690	4.45	AMD	AMD Ryzen 3	7320U	7.0	4.0	AMD Radeon Graphics	AMD	NaN	
2	HP 15-fq5007TU Laptop (12th Gen Core i3/ 8GB/...	HP	37012	4.65	Intel	Intel Core i3	1215U	12.0	6.0	Intel UHD Graphics	Intel	NaN	
3	Samsung Galaxy Book2 Pro 13 Laptop (12th Gen C...	Samsung	69990	4.75	Intel	Intel Core i5	1240P	12.0	12.0	Intel Iris Xe Graphics	Intel	NaN	
4	Tecno Megabook T1 Laptop (11th Gen Core i3/ 8G...	Tecno	23990	4.25	Intel	Intel Core i3	1115G4	11.0	2.0	Intel UHD Graphics	Intel	NaN	

Рисунок 3.1.1 — Перші 5 записів обраного датасету

Далі було проаналізовано структуру даних (рис. 3.1.2), завдяки чому можна описати вміст датасету та кількість пропусків. В даних помічено багато пропущених значень, а саме:

- 1) 24 пропущених значень Processor\_variant
- 2) 125 пропущених значень Processor\_gen
- 3) 12 пропущених значень Core\_per\_processor
- 4) По 447 пропущених значень Total\_processor та Execution\_units
- 5) 38 пропущених значень Threads
- 6) 22 пропущених значення RAM\_type
- 7) По 2 значення Graphics\_name, Graphics\_brand, Graphics\_integreted
- 8) 652 пропущених значень Graphics\_GB

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 29 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Unnamed: 0                            1020 non-null   int64
1   Name                                  1020 non-null   object
2   Brand                                 1020 non-null   object
3   Price                                1020 non-null   int64
4   Rating                               1020 non-null   float64
5   Processor_brand                       1020 non-null   object
6   Processor_name                        1020 non-null   object
7   Processor_variant                     996 non-null    object
8   Processor_gen                         891 non-null    float64
9   Core_per_processor                    1008 non-null   float64
10  Total_processor                       573 non-null    float64
11  Execution_units                       573 non-null    float64
12  Low_Power_Cores                       1020 non-null   float64
13  Energy_Efficient_Units                1020 non-null   int64
14  Threads                               972 non-null    float64
15  RAM_GB                                1020 non-null   int64
16  RAM_type                              998 non-null    object
17  Storage_capacity_GB                   1020 non-null   int64
18  Storage_type                          1020 non-null   object
19  Graphics_name                         1018 non-null   object
20  Graphics_brand                        1018 non-null   object
21  Graphics_GB                           368 non-null    float64
22  Graphics_integrated                   1018 non-null   object
23  Display_size_inches                  1020 non-null   float64
24  Horizontal_pixel                      1020 non-null   int64
25  Vertical_pixel                        1020 non-null   int64
26  ppi                                   1020 non-null   float64
27  Touch_screen                         1020 non-null   bool
28  Operating_system                     1020 non-null   object
dtypes: bool(1), float64(10), int64(7), object(11)
memory usage: 224.2+ KB
```

Рисунок 3.1.2 — Структура обраного датасету

Окрім пропущених даних, в датафреймі помітні неправильні типи даних та помилки у назвах таблиць. При подальшій роботі із даними пропуски треба заповнити, а помилки виправити.

Структуру даних можна описати за допомогою таблиці 3.1.1.

Таблиця 3.1.1 — Структура датафрейму

Назва поля	Опис поля
Unnamed	Містить унікальні ID для кожного запису
Name	Повна назва ноутбука
Brand	Назва виробника
Price	Ціна девайсу в індійських рупіях

Продовження таблиці 3.1.1

Назва поля	Опис поля
Rating	Користувацька оцінка товару
Processor_brand	Містить назву виробника процесору
Processor_name	Містить назву лінійки процесору
Processor_variant	Містить назву моделі процесору
Processor_gen	Покоління процесору
Core_per_processor	Кількість ядер процесора
Total_processor	Кількість процесорів у ноутбука
Execution_units	Число одиниць виконання
Low_Power_Cores	Кількість малопотужних ядер
Energy_Efficient_Units	Кількість енергоефективних блоків
Threads	Кількість потоків процесора
RAM_GB	Об'єм оперативної пам'яті
RAM_type	Тип оперативної пам'яті
Storage_capacity_GB	Об'єм накопичувача
Storage_type	Тип накопичувача
Graphics_name	Назва графічного процесору
Graphics_brand	Виробник графічного процесору
Graphics_GB	Об'єм відеопам'яті
Graphics_integreted	Маркер інтегрованості відеоядра
Display_size_inches	Розмір дисплею
Horizontal_pixel	Кількість горизонтальних пікселів

Продовження таблиці 3.1.1

Назва поля	Опис поля
Vertical_pixel	Кількість вертикальних пікселів
ppi	Кількість пікселів на квадратний дюйм
Touch_screen	Маркер наявності сенсорного екрану
Operating_system	Тип вбудованої операційної системи

Датасет містить 29 колонок та 1020 записів. В колонках містяться дані різних типів: object, float, int, bool. На етапі перевірки помилок правильність типізації треба буде перевірити.

### 3.2. Заповнення пропусків у даних

Пропуски у даних можуть спотворити результати аналізу. Коректне заповнення пропусків у даних має покращити якість моделювання, підвищити точність прогнозів та забезпечити більш стабільну роботу моделей. Видаляти неповні записи не варто, адже іноді видалення рядків із пропусками може призвести до значної втрати даних.

Для заповнення пропущених даних використано вже наявні дані. Перша колонка, яка потребує заповнення, це Processor\_variant. Ця колонка містить дані про конкретні моделі процесора кожного ноутбуку. При аналізі відсутніх даних виявилось, що проблема актуальна лише для процесорів від виробників Apple та MediaTek (рис. 3.2.1). Така закономірність полегшує відновлення даних. Помилку було виправлено наступним чином: для процесорів від виробника Apple дані про модель взято з колонки Processor\_name (колонка містить дані про лінійку процесора), а для виробника MediaTek – з назви ноутбука (колонка “Name”). Результатом таких дій стало коректне заповнення пропусків, що видно на рисунку 3.2.2.

```
filtered_df = df[df['Processor_variant'].isna()]
result_df = filtered_df[['id', 'Name', 'Processor_brand', 'Processor_name']]
result_df
```

	id	Name	Processor_brand	Processor_name
13	13	Apple MacBook Air 2020 MGN3HN Laptop (Apple M...	Apple	Apple M1
17	17	Apple MacBook Air 2024 Laptop (Apple M3/ 8GB/ ...	Apple	Apple M3
23	23	Apple MacBook Air 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2
73	73	Apple MacBook Air 2024 MRYN3HN/A Laptop (Apple...	Apple	Apple M3
83	83	Apple MacBook Pro 16 2023 Laptop (Apple M3 Max...	Apple	Apple M3 Max
100	100	Apple MacBook Air 15 2024 MRYR3HN/A Laptop (Ap...	Apple	Apple M3
116	116	Apple MacBook Pro 14 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro
138	138	Apple MacBook Air 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2 Apple M2 Chip
154	154	Lenovo Ideapad Slim 3 Chrome 14M868 82XJ002RHA...	MediaTek	MediaTek Kompanio
163	163	Apple MacBook Pro 14 2023 Laptop (Apple M3/ 8G...	Apple	Apple M3
202	202	Apple MacBook Air 2024 MRXW3HN/A Laptop (Apple...	Apple	Apple M3
203	203	Apple MacBook Air 15 2024 Laptop (Apple M3/ 16...	Apple	Apple M3
224	224	Lenovo Ideapad Slim 3 Chrome 14M868 82XJ002LHA...	MediaTek	MediaTek Kompanio
261	261	Apple MacBook Air 15 2023 Laptop (Apple M2/ 8G...	Apple	Apple M2
271	271	Apple MacBook Air 15 2023 Laptop (Apple M2/ 8G...	Apple	Apple M2
294	294	Apple MacBook Pro 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2 Apple M2 Chip
347	347	Apple MacBook Pro 16 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro
348	348	Apple MacBook Pro 14 2023 Laptop (Apple M3/ 8G...	Apple	Apple M3
349	349	Apple MacBook Pro 14 2023 Laptop (Apple M3 Max...	Apple	Apple M3 Max
350	350	Apple MacBook Pro 14 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro
372	372	Primebook PBMTWIFI11064 Wi-Fi Laptop (MediaTek...	MediaTek	MediaTek
384	384	Primebook S Wi-Fi Laptop (MediaTek MT8183/ 4GB...	MediaTek	MediaTek
527	527	Primebook 4G Android Laptop (MediaTek Kompanio...	MediaTek	MediaTek
628	628	Asus Chromebook CM14 CM1402CM2A-EK0085 Laptop ...	MediaTek	MediaTek Kompanio

Рисунок 3.2.1 — Аналіз відсутніх даних колонки Processor\_variant

```
filtered_df = df[(df['Processor_brand'] == 'Apple') | (df['Processor_brand'] == 'MediaTek')]
result_df = filtered_df[['id', 'Name', 'Processor_brand', 'Processor_name', 'Processor_variant']]
result_df
```

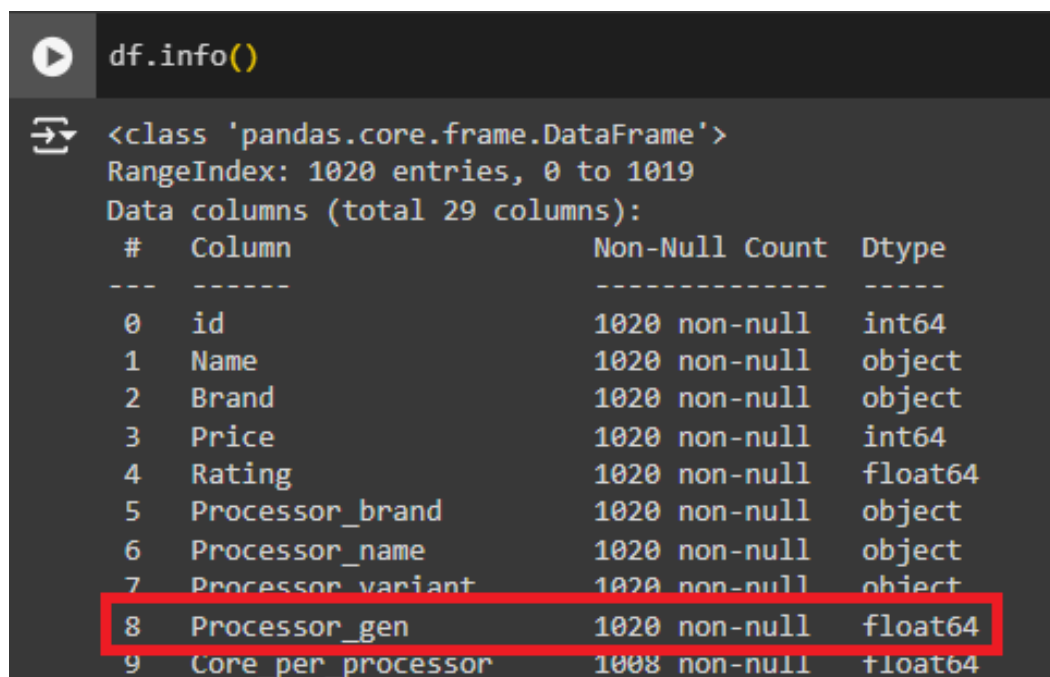
	id	Name	Processor_brand	Processor_name	Processor_variant
13	13	Apple MacBook Air 2020 MGN3HN Laptop (Apple M...	Apple	Apple M1	M1
17	17	Apple MacBook Air 2024 Laptop (Apple M3/ 8GB/ ...	Apple	Apple M3	M3
23	23	Apple MacBook Air 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2	M2
73	73	Apple MacBook Air 2024 MRYN3HN/A Laptop (Apple...	Apple	Apple M3	M3
83	83	Apple MacBook Pro 16 2023 Laptop (Apple M3 Max...	Apple	Apple M3 Max	M3 Max
100	100	Apple MacBook Air 15 2024 MRYR3HN/A Laptop (Ap...	Apple	Apple M3	M3
116	116	Apple MacBook Pro 14 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro	M3 Pro
138	138	Apple MacBook Air 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2 Apple M2 Chip	M2
154	154	Lenovo Ideapad Slim 3 Chrome 14M868 82XJ002RHA...	MediaTek	MediaTek Kompanio	520
163	163	Apple MacBook Pro 14 2023 Laptop (Apple M3/ 8G...	Apple	Apple M3	M3
202	202	Apple MacBook Air 2024 MRXW3HN/A Laptop (Apple...	Apple	Apple M3	M3
203	203	Apple MacBook Air 15 2024 Laptop (Apple M3/ 16...	Apple	Apple M3	M3
224	224	Lenovo Ideapad Slim 3 Chrome 14M868 82XJ002LHA...	MediaTek	MediaTek Kompanio	520
261	261	Apple MacBook Air 15 2023 Laptop (Apple M2/ 8G...	Apple	Apple M2	M2
271	271	Apple MacBook Air 15 2023 Laptop (Apple M2/ 8G...	Apple	Apple M2	M2
294	294	Apple MacBook Pro 2022 Laptop (Apple M2/ 8GB/ ...	Apple	Apple M2 Apple M2 Chip	M2
344	344	Jio JioBook Cloud Laptop (Octa Core/ 4GB/ 64GB...	MediaTek	Mediatek	MT8788
347	347	Apple MacBook Pro 16 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro	M3 Pro
348	348	Apple MacBook Pro 14 2023 Laptop (Apple M3/ 8G...	Apple	Apple M3	M3
349	349	Apple MacBook Pro 14 2023 Laptop (Apple M3 Max...	Apple	Apple M3 Max	M3 Max
350	350	Apple MacBook Pro 14 2023 Laptop (Apple M3 Pro...	Apple	Apple M3 Pro	M3 Pro
372	372	Primebook PBMTWIFI11064 Wi-Fi Laptop (MediaTek...	MediaTek	MediaTek	MT8183
384	384	Primebook S Wi-Fi Laptop (MediaTek MT8183/ 4GB...	MediaTek	MediaTek	MT8183
527	527	Primebook 4G Android Laptop (MediaTek Kompanio...	MediaTek	MediaTek	500
628	628	Asus Chromebook CM14 CM1402CM2A-EK0085 Laptop ...	MediaTek	MediaTek Kompanio	520

Рисунок 3.2.2 — Результат виправлення пропусків у Processor\_variant

Заповнення пропусків у колонці Processor\_gen було дещо складнішим, оскільки нумерація поколінь у кожного виробника процесорів особлива. Пропуски було заповнено за наступними правилами:

- 1) Для процесорів від Apple номер покоління взято з його назви (наприклад, процесор M1 – це процесор першого покоління, M2 – другого і т.д.).
- 2) Для інших процесорів пропуски Processor\_gen заповнено модою за виробником. Для цього фільтруються записи для кожного виробника, знаходиться мода і нею заповнюються пропуски.
- 3) Для виробників, у яких дані про покоління процесорів відсутні, встановлюється 1 покоління. Таке рішення обумовлено тим, що часто у лінійках менш відомих виробників ЦП відсутня нумерація поколінь.

Результат заповнення відображено на рисунку 3.2.3.



```
df.info()
```

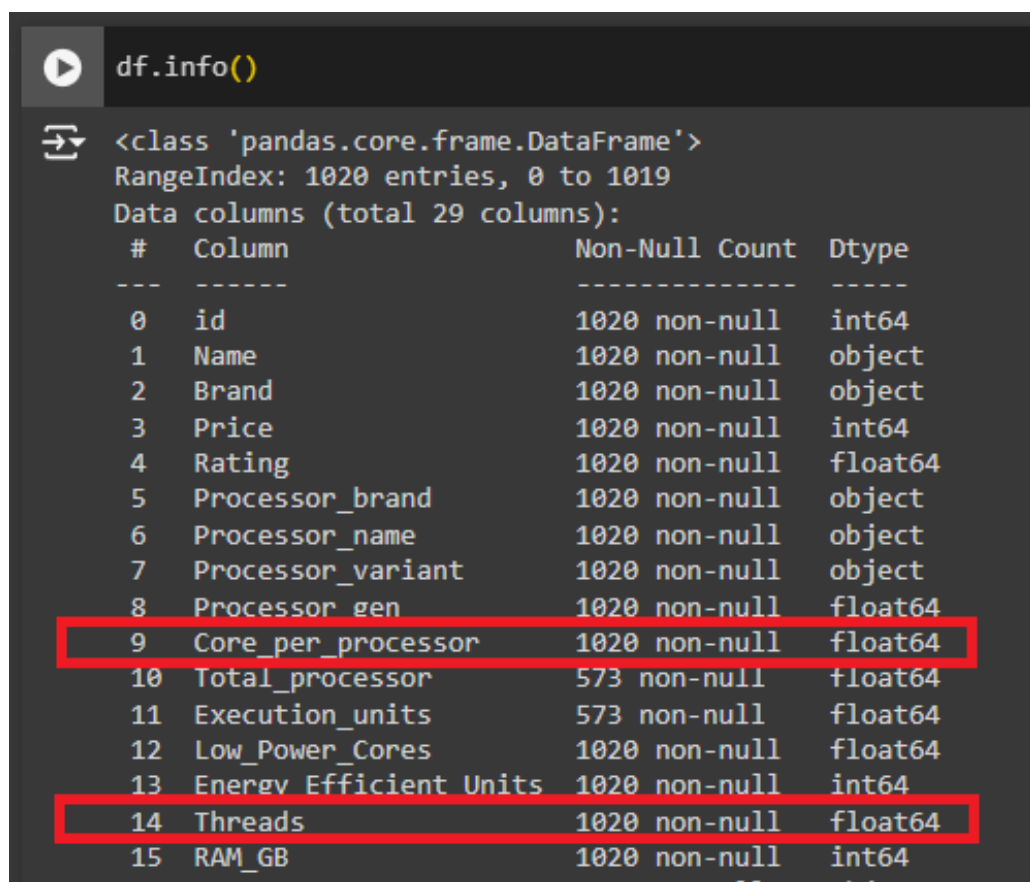
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 29 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   id                    1020 non-null  int64
 1   Name                  1020 non-null  object
 2   Brand                 1020 non-null  object
 3   Price                 1020 non-null  int64
 4   Rating                1020 non-null  float64
 5   Processor_brand       1020 non-null  object
 6   Processor_name        1020 non-null  object
 7   Processor_variant     1020 non-null  object
 8   Processor_gen         1020 non-null  float64
 9   Core_per_processor    1008 non-null  float64
```

Рисунок 3.2.3 — Результат виправлення пропусків у Processor\_gen

Заповнення пропусків у колонках Core\_per\_processor та Threads також вимагають деяких правил заповнення, адже у кожного виробника своя архітектура ЦП. Дані було заповнено за наступними правилами:

- 1) Якщо дані про кількість ядер у даної моделі процесора відома, то її слід знайти і нею заповнити дані
- 2) Якщо кількість ядер не відома, її слід заповнити модою датафрейму
- 3) Кількість потоків процесора слід шукати за допомогою моди. Для цього формується вибірка із записів, у яких зустрічається така сама кількість ядер, як і у запису з відсутніми даними про кількість потоків. В цій вибірці знаходиться моді кількості потоків і це число записується.
- 4) У процесорів Apple кількість потоків дорівнює кількості ядер, оскільки Apple не використовують технологію Multithreading
- 5) Всі інші процесори часто використовують технологію Multithreading, тоді кількість ядер слід подвоїти і таким чином знайти кількість потоків для не заповнених на минулих етапах даних.

Результат такого виправлення пропусків на рисунку 3.2.4.



```
df.info()
```

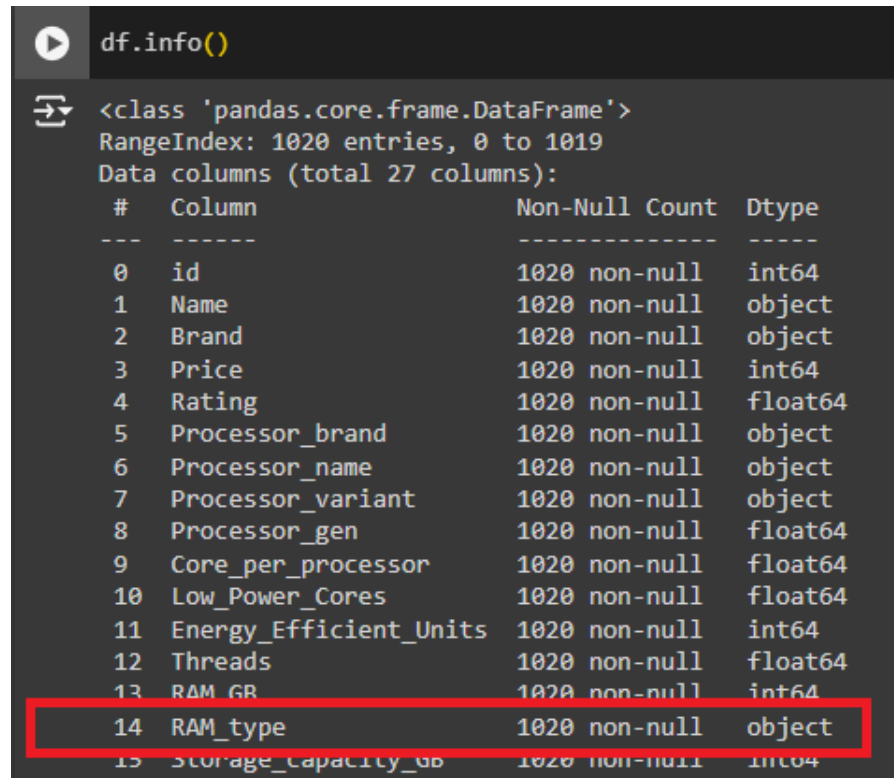
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 29 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                    1020 non-null   int64
1   Name                                 1020 non-null   object
2   Brand                               1020 non-null   object
3   Price                               1020 non-null   int64
4   Rating                             1020 non-null   float64
5   Processor_brand                     1020 non-null   object
6   Processor_name                      1020 non-null   object
7   Processor_variant                   1020 non-null   object
8   Processor_gen                       1020 non-null   float64
9   Core_per_processor                  1020 non-null   float64
10  Total_processor                     573 non-null    float64
11  Execution_units                     573 non-null    float64
12  Low_Power_Cores                     1020 non-null   float64
13  Energy Efficient Units              1020 non-null   int64
14  Threads                             1020 non-null   float64
15  RAM_GB                              1020 non-null   int64
16  RAM_type                             998 non-null    object
```

Рисунок 3.2.4 — Результат виправлення пропусків у Core\_per\_processor та Threads



Колонки `Total_processor` та `Execution_units` слід видалити, адже майже половина даних відсутня і спроба імітувати їх себе не виправдає.

Пропуски у `RAM_type` виправити легше. Для кожного об'єму ОЗП (колонка `RAM_GB`) знайдено моду значення `RAM_type`. Для техніки Apple `RAM_type` слід замінити на власний тип ОЗП виробника – `Unified memory`. Результат такого виправлення на рисунку 3.2.5.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     1020 non-null   int64
1   Name                                  1020 non-null   object
2   Brand                                 1020 non-null   object
3   Price                                 1020 non-null   int64
4   Rating                               1020 non-null   float64
5   Processor_brand                       1020 non-null   object
6   Processor_name                        1020 non-null   object
7   Processor_variant                     1020 non-null   object
8   Processor_gen                         1020 non-null   float64
9   Core_per_processor                    1020 non-null   float64
10  Low_Power_Cores                       1020 non-null   float64
11  Energy_Efficient_Units                 1020 non-null   int64
12  Threads                               1020 non-null   float64
13  RAM_GB                                1020 non-null   int64
14  RAM_type                              1020 non-null   object
15  Storage_capacity_gb                    1020 non-null   int64
```

Рисунок 3.2.5 — Результат виправлення пропусків `RAM_type`

Виправлення пропущених даних для характеристик графічного процесора виявилось дещо складнішим завданням. Заповнення відбулося за наступними правилами:

- 1) Колонка `Graphics_name` заповнюється модою датафрейму
- 2) Колонка `Graphics_brand` заповнюється завдяки відфільтрованому датафрейму, в якому `Graphics_name` дорівнює моді із попереднього пункту. Таким чином встановлюється правильний виробник графічного процесору, який є модою.
- 3) Колонка `Graphics_integrated` заповнюється по такому ж правилу, як і `Graphics_brand`.

- 4) Порожні значення колонки Graphics\_GB заповнені нулями, адже ці процесори не мають власної відеопам'яті і використовують ОЗП.

Результат виправлення колонок, що характеризують графічний процесор, відображені на рисунку 3.2.6. На цьому процес заповнення пропусків завершується.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                    1020 non-null   int64
1   Name                                 1020 non-null   object
2   Brand                               1020 non-null   object
3   Price                               1020 non-null   int64
4   Rating                              1020 non-null   float64
5   Processor_brand                     1020 non-null   object
6   Processor_name                      1020 non-null   object
7   Processor_variant                   1020 non-null   object
8   Processor_gen                       1020 non-null   float64
9   Core_per_processor                  1020 non-null   float64
10  Low_Power_Cores                     1020 non-null   float64
11  Energy_Efficient_Units               1020 non-null   int64
12  Threads                             1020 non-null   float64
13  RAM_GB                              1020 non-null   int64
14  RAM_type                            1020 non-null   object
15  Storage_capacity_GB                 1020 non-null   int64
16  Storage_type                        1020 non-null   object
17  Graphics_name                       1020 non-null   object
18  Graphics_brand                      1020 non-null   object
19  Graphics_GB                         1020 non-null   float64
20  Graphics_integrated                  1020 non-null   object
21  Display_size_inches                 1020 non-null   float64
22  Horizontal_pixel                     1020 non-null   int64
23  Vertical_pixel                      1020 non-null   int64
24  ppi                                 1020 non-null   float64
25  Touch_screen                        1020 non-null   bool
26  Operating_system                    1020 non-null   object
dtypes: bool(1), float64(8), int64(7), object(11)
memory usage: 208.3+ KB
```

Рисунок 3.2.6 — Результат заповнення пропусків колонок Graphics\_name, Graphics\_brand, Graphics\_integrated та Graphics\_GB

### 3.3. Виправлення помилок у даних

Виправлення помилок у даних критично важливе для забезпечення якості та точності аналізу. Виправлення помилок допомагає переконатися, що результати аналізу є правдивими і відображають реальний стан речей. Цей процес може включати як автоматизоване очищення даних, так і ручну перевірку та корекцію, що допомагає покращити загальну чистоту та якість даних.

Першим кроком було проаналізовано типізацію колонок. Визначено, що колонки `Threads`, `Graphics_GB`, `Processor_gen`, `Core_per_processor` та `Graphics_integrated` мають неправильні типи даних. Їх було виправлено, а типи даних до і після зміни зображено на рисунку 3.3.1.

#	Column	Non-Null Count	Dtype
0	id	1020 non-null	int64
1	Name	1020 non-null	object
2	Brand	1020 non-null	object
3	Price	1020 non-null	int64
4	Rating	1020 non-null	float64
5	Processor_brand	1020 non-null	object
6	Processor_name	1020 non-null	object
7	Processor_variant	1020 non-null	object
8	Processor_gen	1020 non-null	float64
9	Core_per_processor	1020 non-null	float64
10	Low_Power_Cores	1020 non-null	float64
11	Energy_Efficient_Units	1020 non-null	int64
12	Threads	1020 non-null	float64
13	RAM_GB	1020 non-null	int64
14	RAM_type	1020 non-null	object
15	Storage_capacity_GB	1020 non-null	int64
16	Storage_type	1020 non-null	object
17	Graphics_name	1020 non-null	object
18	Graphics_brand	1020 non-null	object
19	Graphics_GB	1020 non-null	float64
20	Graphics_integrated	1020 non-null	object
21	Display_size_inches	1020 non-null	float64
22	Horizontal_pixel	1020 non-null	int64
23	Vertical_pixel	1020 non-null	int64
24	ppi	1020 non-null	float64
25	Touch_screen	1020 non-null	bool
26	Operating_system	1020 non-null	object

dtypes: bool(1), float64(8), int64(7), object(11)  
memory usage: 208.3+ KB

#	Column	Non-Null Count	Dtype
0	id	1020 non-null	int64
1	Name	1020 non-null	object
2	Brand	1020 non-null	object
3	Price	1020 non-null	int64
4	Rating	1020 non-null	float64
5	Processor_brand	1020 non-null	object
6	Processor_name	1020 non-null	object
7	Processor_variant	1020 non-null	object
8	Processor_gen	1020 non-null	int64
9	Core_per_processor	1020 non-null	int64
10	Low_Power_Cores	1020 non-null	float64
11	Energy_Efficient_Units	1020 non-null	int64
12	Threads	1020 non-null	int64
13	RAM_GB	1020 non-null	int64
14	RAM_type	1020 non-null	object
15	Storage_capacity_GB	1020 non-null	int64
16	Storage_type	1020 non-null	object
17	Graphics_name	1020 non-null	object
18	Graphics_brand	1020 non-null	object
19	Graphics_GB	1020 non-null	int64
20	Graphics_integrated	1020 non-null	bool
21	Display_size_inches	1020 non-null	float64
22	Horizontal_pixel	1020 non-null	int64
23	Vertical_pixel	1020 non-null	int64
24	ppi	1020 non-null	float64
25	Touch_screen	1020 non-null	bool
26	Operating_system	1020 non-null	object

dtypes: bool(2), float64(4), int64(11), object(10)  
memory usage: 201.3+ KB

Рисунок 3.3.1 — Результат виправлення типів даних колонок

В рамках процесу перевірки на помилки в записах датафрейму було визначено, що колонки `id`, `Name`, `Brand`, `Price`, `Rating`, `Processor_brand`, `Processor_gen`, `Core_per_processor`, `Threads`, `RAM_GB`, `Storage_capacity_GB`, `Graphics_GB`, `Graphics_integrated`, `Display_size_inches`, `ppi`, `Touch_screen` явних помилок не мають.

Числові дані було проаналізовано за допомогою методу Pandas `.describe()`. Результат виклику методу зображено на рисунку 3.3.2. Виявлено наступне:

- 1) середні значення `Low_Power_Cores` та `Energy_Efficient_Units` майже дорівнюють 0. Ці дані явно порожні, їх варто видалити
- 2) Всі характеристики, що мають абсолютне значення, додатні
- 3) Значення 1080 для `Horizontal_pixel` явно не є типовим

	id	Price	Rating	Processor_gen	Core_per_processor	Low_Power_Cores	Energy_Efficient_Units	Threads	RAM_GB	Storage_capacity_GB	Graphics_GB	Display_size_inches	Horizontal_pixel	Vertical_pixel	ppi
count	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000	1020.000000
mean	509.500000	82063.474510	4.373876	10.442157	8.577451	0.086275	0.043137	12.968627	13.992157	627.733333	2.143137	15.163775	2035.512745	1214.019608	157.178265
std	294.592939	66502.150607	0.233295	3.213317	4.386714	0.406531	0.203266	5.705652	7.189564	316.911679	3.272256	1.001537	409.209289	306.863086	33.585713
min	0.000000	8000.000000	3.950000	1.000000	2.000000	0.000000	0.000000	2.000000	2.000000	32.000000	0.000000	11.600000	1080.000000	768.000000	100.450000
25%	254.750000	43990.000000	4.200000	7.000000	6.000000	0.000000	0.000000	8.000000	8.000000	512.000000	0.000000	14.000000	1920.000000	1080.000000	141.210000
50%	509.500000	63689.500000	4.350000	12.000000	8.000000	0.000000	0.000000	12.000000	16.000000	512.000000	0.000000	15.600000	1920.000000	1080.000000	141.210000
75%	764.250000	94990.000000	4.550000	13.000000	10.000000	0.000000	0.000000	16.000000	16.000000	512.000000	4.000000	15.600000	1920.000000	1200.000000	161.730000
max	1019.000000	599990.000000	4.750000	14.000000	24.000000	2.000000	1.000000	32.000000	64.000000	4000.000000	16.000000	18.000000	3840.000000	2560.000000	337.930000

Рисунок 3.3.2 — Результат виклику методу `.describe()`

Здогадку із 3 пункту варто перевірити. Для цього було виведено всі записи, де значення колонки `Horizontal_pixel` дорівнює 1080. Результат на рисунку 3.3.3.

```
[93] filtered_df = df[df['Horizontal_pixel'] == 1080]
      filtered_df[['Vertical_pixel', 'Horizontal_pixel']].head(10)
```

	Vertical_pixel	Horizontal_pixel
3	1920	1080
10	1920	1080
42	1920	1080
412	1920	1080
560	1920	1080
689	1920	1080
813	1920	1080

Рисунок 3.3.3 — Результат пошуку помилок у колонці `Horizontal_pixel`

Можна зробити наступний висновок: помилка у колонках `Horizontal_pixel` та `Vertical_pixel` присутня. Горизонтальна кількість пікселів має бути вищою за вертикальну, адже всі ноутбуки виготовляються горизонтальними, а не вертикальними. У деяких ноутбуках ці параметри переплутані і їх треба виправити, помінявши місцями числа.

У ході подальшого дослідження виявлено, що серед категоріальних характеристик ноутбуків колонки Processor\_name, Processor\_variant, RAM\_type, Storage\_type, Graphics\_brand, Graphics\_name та Operating\_system є помилки або деякі неточності. Розглянемо кожну характеристику окремо.

Колонка Processor\_name має численні помилки, а саме:

- 1) для процесорів лінійки Apple M3 вказана конкретна модель процесора
- 2) У записі Apple M2 Apple M2 Chip має бути просто Apple M2.
- 3) "Intel " та "Intel Pentium " - мають зайві пробіли в кінці
- 4) Для процесору Intel Core I3-1115G4 значення колонки має бути Intel Core i3, а в колонці Processor\_variant має залишитись 1115G4
- 5) HiSilicon Kirin 9006C 9006C треба замінити на просто HiSilicon Kirin, а в колонці Processor\_variant слід залишити 9006C
- 6) Замість Intel Core 3, 5, 7 має бути Intel Core i3, i5, i7
- 7) MediaTek та Mediatek варто замінити на MediaTek Kompanio
- 8) Microsoft SQ1 SQ1 слід замінити на Microsoft, а в колонці Processor\_variant слід залишити SQ1
- 9) Qualcomm X Elite слід замінити на Qualcomm X, а в колонці Processor\_variant слід залишити Elite
- 10) Помилка у записі “eration Intel Core” нетипова

Перші 9 помилок виправлено за допомогою умовних конструкцій мови Python. 10 помилка нетипова. В процесі дослідження запису із такою помилкою виявлено, що ноутбук має процесор Intel Core i5-1235U. Процес цього дослідження зображено на рисунку 3.3.4.

```
[99] filtered_df = df[df['Processor_name'] == 'eration Intel Core']
    filtered_df[['Name', 'Processor_name', 'Processor_variant']]
```

	Name	Processor_name	Processor_variant
140	Samsung Galaxy Book2 15 Laptop (12th Gen Core ...	eration Intel Core	L3...

Всі дані про процесор помилкові. Їх можна відновити за назвою ноутбуку

```
[100] for word in filtered_df['Name']:
    print(word)
```

```
Samsung Galaxy Book2 15 Laptop (12th Gen Core i5/ 8GB/ 512GB SSD/ Win11)
```

Рисунок 3.3.4 — Результат дослідження помилки запису “eration Intel Core”

Унікальні значення лінійок процесорів та результат виправлення помилок у колонці Processor\_name зображено на рисунку 3.3.5.

```
sorted(df['Processor_name'].unique())
```

```
[ 'AMD Athlon',
  'AMD Athlon Pro',
  'AMD Athlon Silver',
  'AMD Ryzen 3',
  'AMD Ryzen 5',
  'AMD Ryzen 7',
  'AMD Ryzen 9',
  'Apple M1',
  'Apple M2',
  'Apple M2 Apple M2 Chip',
  'Apple M3',
  'Apple M3 Max',
  'Apple M3 Pro',
  'HiSilicon Kirin 9006C 9006C',
  'Intel ',
  'Intel Atom Quad',
  'Intel Celeron ',
  'Intel Celeron Dual',
  'Intel Core 3',
  'Intel Core 5',
  'Intel Core 7',
  'Intel Core i3-1115G4',
  'Intel Core Ultra',
  'Intel Core i3',
  'Intel Core i5',
  'Intel Core i7',
  'Intel Core i9',
  'Intel Pentium ',
  'Intel Pentium Gold',
  'Intel Pentium Silver',
  'MediaTek',
  'MediaTek Kompanio',
  'Mediatek',
  'Microsoft SQ1 SQ1',
  'Qualcomm X Elite',
  'eration Intel Core']
```

```
sorted(df['Processor_name'].unique())
```

```
[ 'AMD Athlon',
  'AMD Athlon Pro',
  'AMD Athlon Silver',
  'AMD Ryzen 3',
  'AMD Ryzen 5',
  'AMD Ryzen 7',
  'AMD Ryzen 9',
  'Apple M1',
  'Apple M2',
  'Apple M3',
  'HiSilicon Kirin',
  'Intel',
  'Intel Atom Quad',
  'Intel Celeron',
  'Intel Celeron Dual',
  'Intel Core Ultra',
  'Intel Core i3',
  'Intel Core i5',
  'Intel Core i7',
  'Intel Core i9',
  'Intel Pentium',
  'Intel Pentium Gold',
  'Intel Pentium Silver',
  'MediaTek Kompanio',
  'Microsoft',
  'Qualcomm X']
```

Рисунок 3.3.5 — Результат виправлення помилок у Processor\_name

Колонка Processor\_variant також має декілька проблем. Деякі суфікси назв моделей процесору у нижньому регістрі, деякі - у верхньому, наявний зайвий символ \u200b, а пропущені моделі процесорів для i5, 7, 9 слід замінити на найпопулярнішу модель серед лінійки i5, 7 і 9 відповідно.

Для вирішення цих помилок всі літери у назвах процесорів було переведено у верхній регістр, символ \u200b видалено, а для пропущених моделей процесорів лінійки i5, 7, 9 було знайдено моду по лінійці.

У колонці RAM\_type було знайдено дві помилки – типи LPDDR5X та LPDDR4X були неправильно написані. Унікальні значення колонки до і після виправлення цих помилок зображено на рисунку 3.3.6.

```
sorted(df['RAM_type'].unique())
```

```
sorted(df['RAM_type'].unique())
```

Left cell output:

```
['DDR3',
 'DDR4',
 'DDR5',
 'DDR6',
 'LPDDR3',
 'LPDDR4',
 'LPDDR4X',
 'LPDDR5',
 'LPDDR5X',
 'LPDDR4X',
 'PDDR5X',
 'Unified']
```

Right cell output:

```
['DDR3',
 'DDR4',
 'DDR5',
 'DDR6',
 'LPDDR3',
 'LPDDR4',
 'LPDDR4X',
 'LPDDR5',
 'LPDDR5X',
 'Unified']
```

Рисунок 3.3.6 — Результат виправлення помилок у RAM\_type

Записи колонки Storage\_type містили зайві пробіли та деякі неточності. Унікальні значення характеристики та результат виправлення на рисунку 3.3.7.

```
[36] sorted(df['Storage_type'].unique())
```

```
[ ' Hard Disk', ' NVMe SSD', ' SSD', 'Hard Disk & SSD']
```

```
df['Storage_type'] = df['Storage_type'].str.strip()
df['Storage_type'] = df['Storage_type'].replace('Hard Disk', 'HDD', regex = True)
```

```
[51] sorted(df['Storage_type'].unique())
```

```
['HDD', 'HDD & SSD', 'NVMe SSD', 'SSD']
```

Рисунок 3.3.7 — Результат виправлення помилок у Storage\_type

Характеристика Graphics\_name до виправлення містила найбільшу кількість помилок та неточностей. Одні й ті самі моделі графічних процесорів були записані по-різному. Було помічено наступні основні помилки:

- 1) наявні зайві символи: тире, пробіли, ®, \u200e
- 2) характеристики "Graphics" та "Integrated" зайві
- 3) є проблеми із написанням назв виробників (наприклад, INTEL)
- 4) в деяких характеристиках відсутні пробіли



- 5) є дублікати характеристик або ці характеристики пропущені
- 6) всі записи, що містять назву Intel Iris слід уніфікувати до Intel Iris Xe
- 7) не вказано виробника відеядер Apple

Всі ці проблеми було виправлено завдяки умовним конструкціям та циклам мови Python. В результаті виправлення все моделі графічних процесорів записані правильно та не мають написаних по-іншому дублікатів.

Ознака `Operating_system` до виправлення також мала деякі неточності в назвах. Ці неточності було виправлено. Значення до виправлення і результат виправлення зображені на рисунку 3.3.8.

Before	After
'Android 11 OS'	'Android 11 OS'
'Chrome OS'	'Chrome OS'
'DOS 3.0 OS'	'DOS OS'
'DOS OS'	'Jio OS'
'Linux OS'	'Linux OS'
'Mac 10.15.3\t OS'	'Mac OS'
'Mac Catalina OS'	'Prime OS'
'Mac OS'	'Windows 10 OS'
'Prime OS'	'Windows 11 OS'
'Ubuntu OS'	
'Windows 10 OS'	
'Windows 11 OS'	
'jio'	

Рисунок 3.3.8 — Результат виправлення помилок у `Operating_system`

На цьому етап виправлення помилок і роботи з даними в цілому завершено. Дані було проаналізовано, пропуски у даних були заповнені, помилки виправлені. Датасет готовий до інтелектуального аналізу.

Для підготовки даних до аналізу було використано бібліотеки `Pandas` та `re`. Весь код процесу обробки даних наведено в Додатку А.



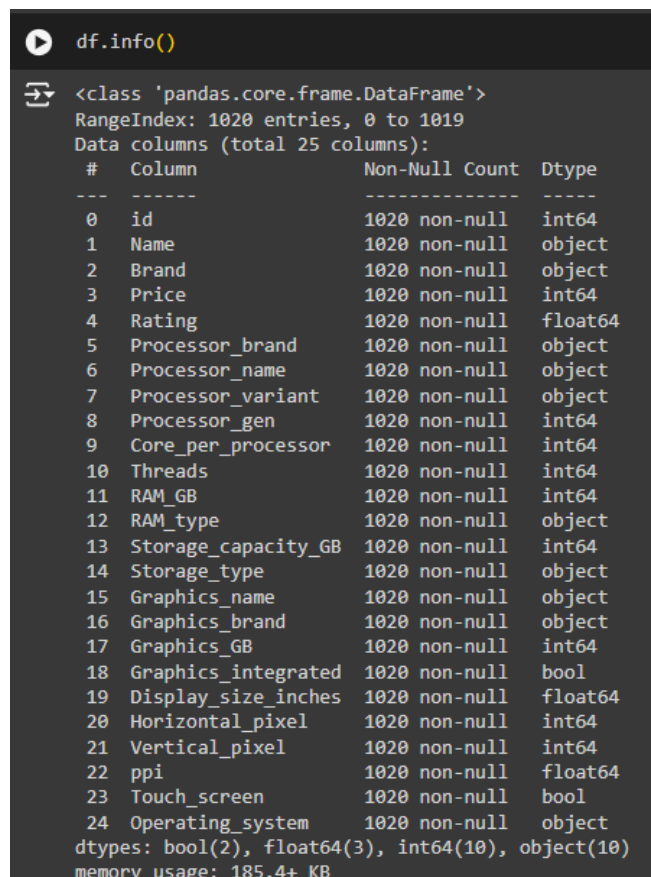
## 4. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ

Мета цієї курсової роботи – це передбачення ціни на ноутбук на основі його характеристик. Ціль інтелектуального аналізу полягає у визначенні характеристик ноутбука, що впливають найбільше на ціну, а також у пошуці, навчанні та покращенні моделей машинного навчання.

### 4.1. Обґрунтування вибору факторів

Для прогнозування ціни на ноутбуки на основі його ключових характеристик слід обрати, які характеристики найбільше впливають на ціну девайсу, і які з них найкраще підходять для навчання моделей.

В ході роботи з даними датасет було підготовлено до аналізу даних. У фінальному вигляді файл з даними містить 25 колонок (рис. 4.1.1) з різними типами даних: 10 колонок з типом object, 10 з типом int64, 3 з типом float64 і 2 колонки з типом bool. Очевидно, що використання всіх цих характеристик негативно вплине на точність прогнозу та продуктивність програмного забезпечення.



```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 25 columns):
#   Column                                Non-Null Count  Dtype  
---  -
0   id                                    1020 non-null   int64  
1   Name                                1020 non-null   object  
2   Brand                               1020 non-null   object  
3   Price                               1020 non-null   int64  
4   Rating                              1020 non-null   float64 
5   Processor_brand                     1020 non-null   object  
6   Processor_name                      1020 non-null   object  
7   Processor_variant                   1020 non-null   object  
8   Processor_gen                       1020 non-null   int64  
9   Core_per_processor                  1020 non-null   int64  
10  Threads                             1020 non-null   int64  
11  RAM_GB                              1020 non-null   int64  
12  RAM_type                            1020 non-null   object  
13  Storage_capacity_GB                 1020 non-null   int64  
14  Storage_type                        1020 non-null   object  
15  Graphics_name                       1020 non-null   object  
16  Graphics_brand                      1020 non-null   object  
17  Graphics_GB                         1020 non-null   int64  
18  Graphics_integrated                 1020 non-null   bool    
19  Display_size_inches                 1020 non-null   float64 
20  Horizontal_pixel                    1020 non-null   int64  
21  Vertical_pixel                      1020 non-null   int64  
22  ppi                                 1020 non-null   float64 
23  Touch_screen                        1020 non-null   bool    
24  Operating_system                    1020 non-null   object  
dtypes: bool(2), float64(3), int64(10), object(10)
memory usage: 185.4+ KB
```

Рисунок 4.1.1 — Структура даних перед аналізом

На першому етапі дослідження характеристик для наочного уявлення структури даних та виявлення прихованих закономірностей або аномалій побудовано різні за призначенням діаграми. Це спрощує подальше прийняття рішень. На рисунках 4.1.2 – 4.1.15 зображено результати візуалізації даних.

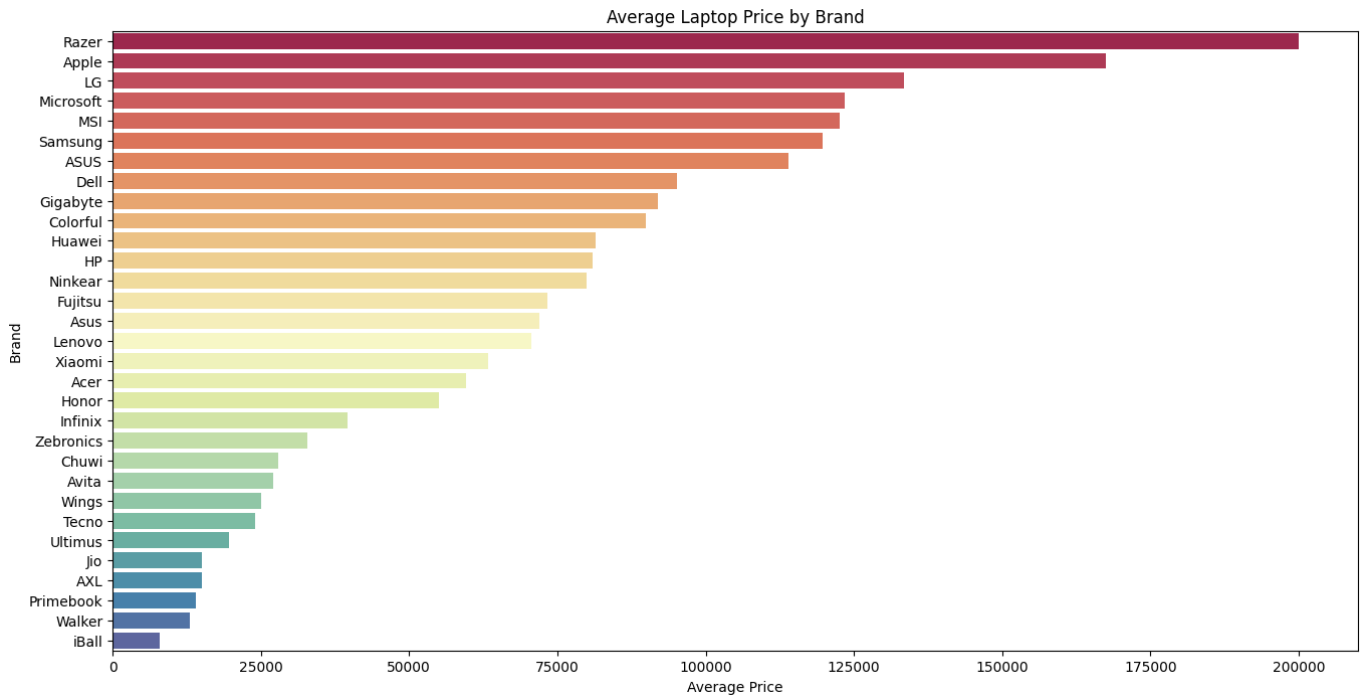


Рисунок 4.1.2 — Середня ціна ноутбука для кожного бренду

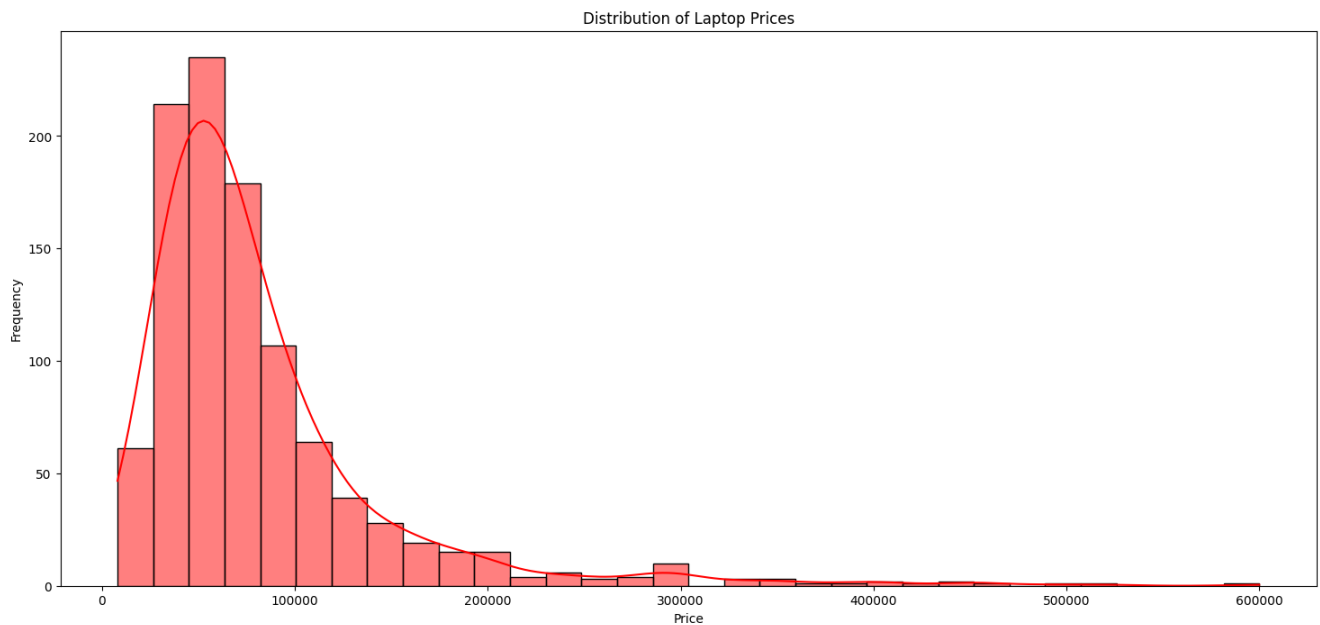


Рисунок 4.1.3 — Гістограма цін ноутбуків

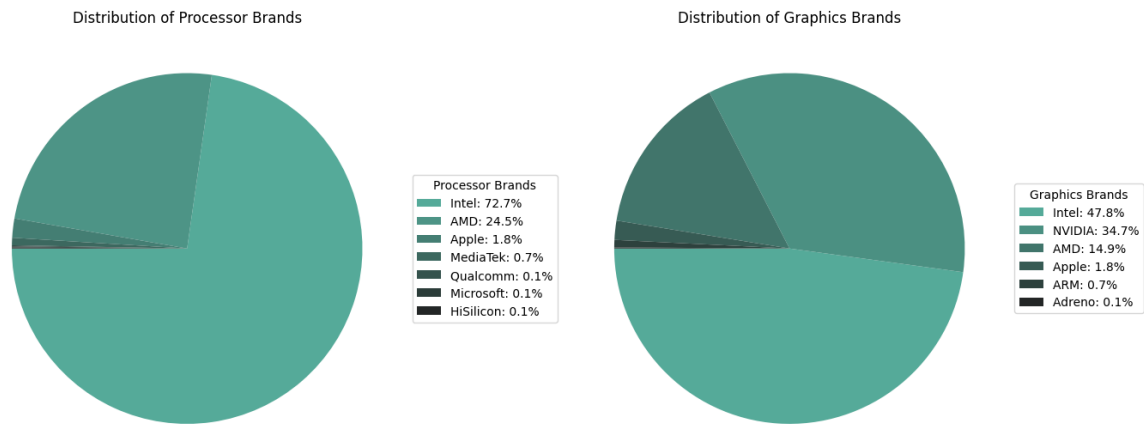


Рисунок 4.1.4 — Кругові діаграми розподілу ЦП та ГП за виробником

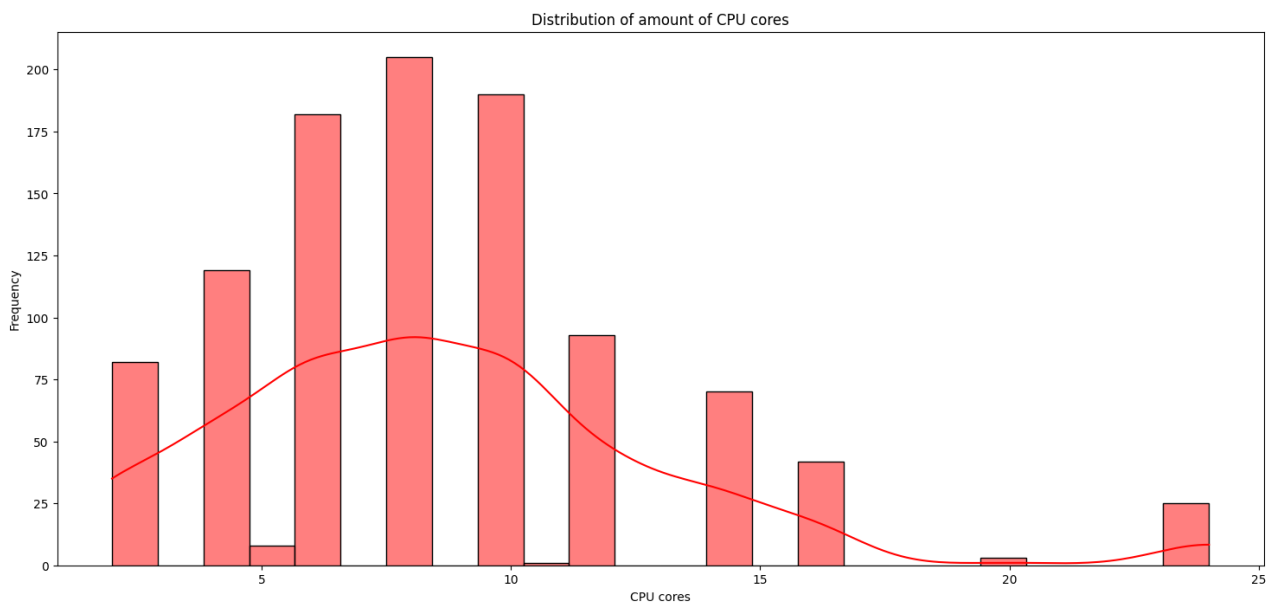


Рисунок 4.1.5 — Гістограма кількості ядер процесора

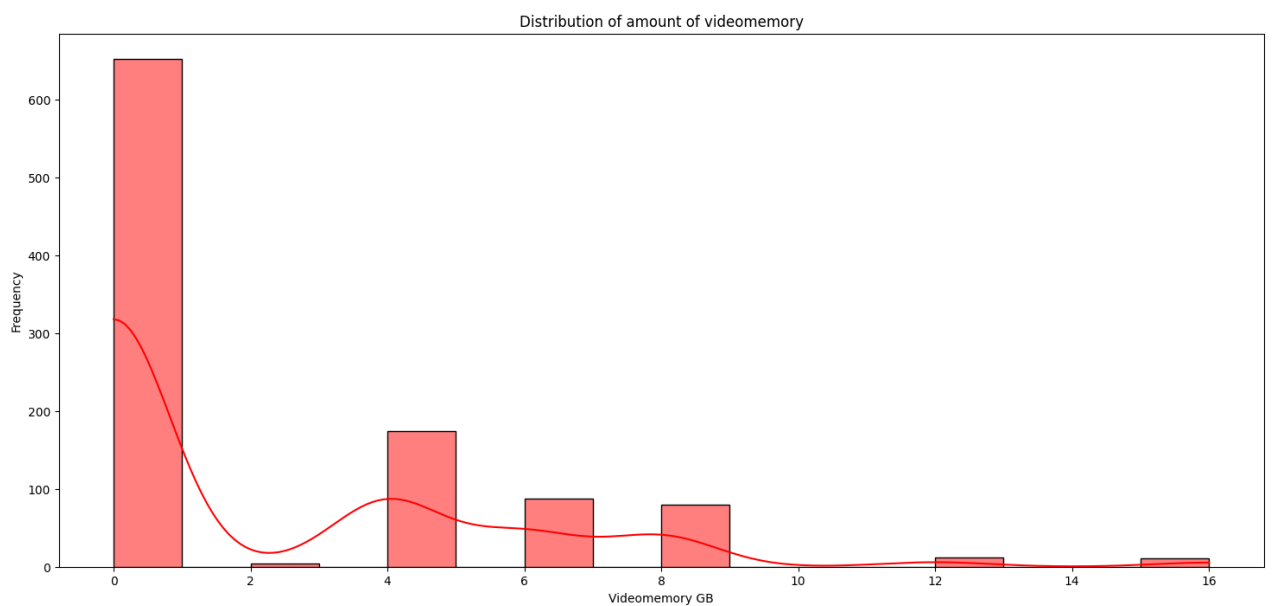


Рисунок 4.1.6 — Гістограма розподілу об'єму відеопам'яті

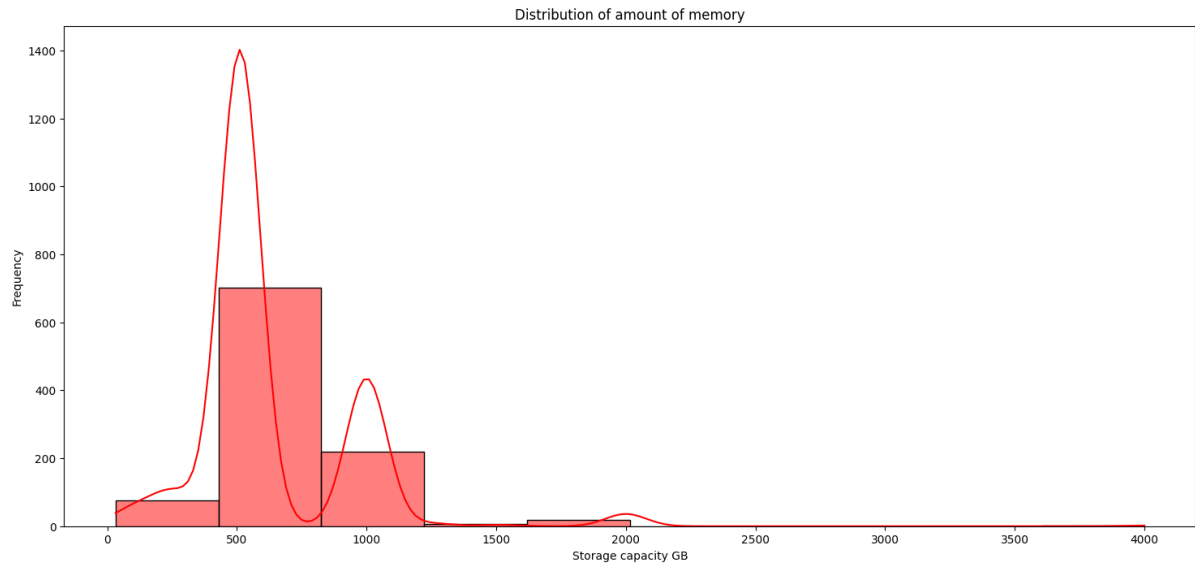


Рисунок 4.1.7 — Гістограма об’єму накопичувача

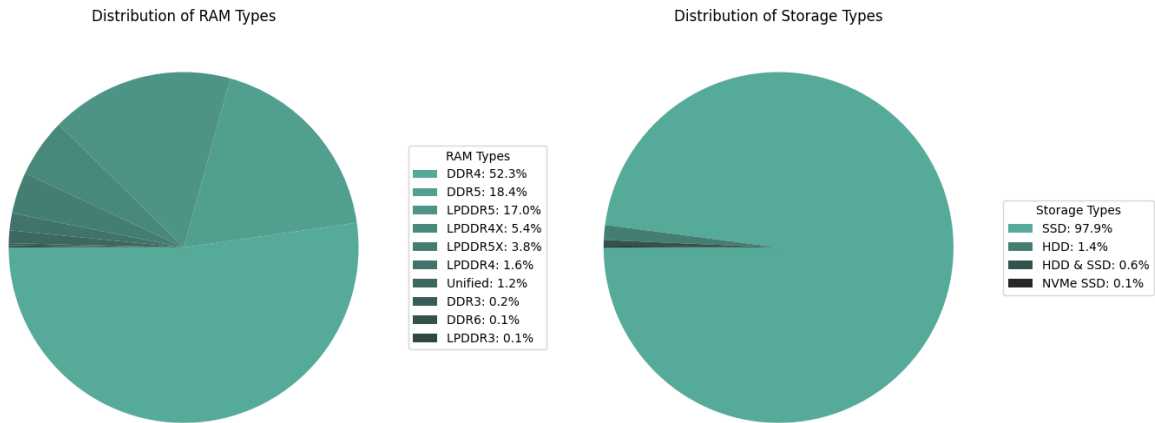


Рисунок 4.1.8 — Кругові діаграми розподілу типів ОЗП та накопичувача

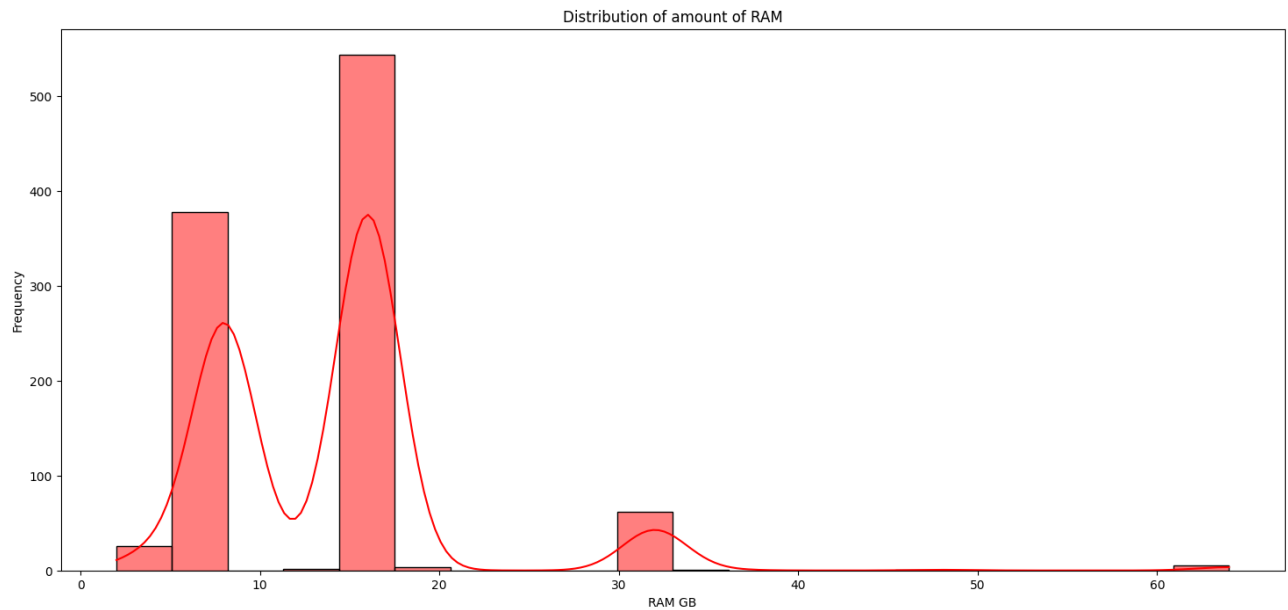


Рисунок 4.1.9 — Гістограма об’єму оперативної пам’яті

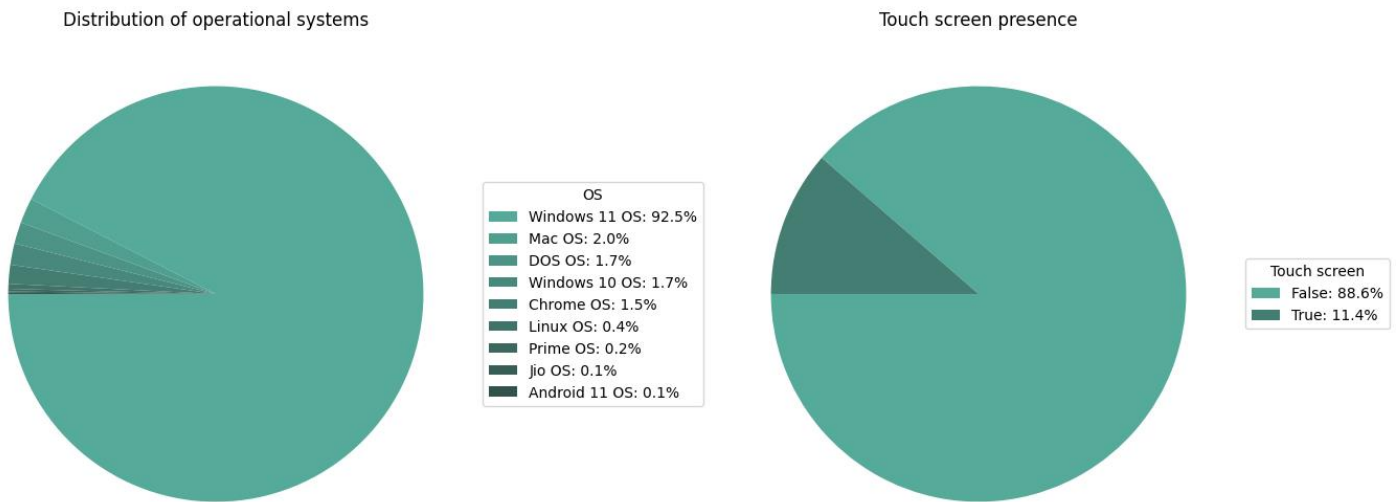


Рисунок 4.1.10 — Кругові діаграми розподілу типів операційної системи та наявності сенсорного екрану

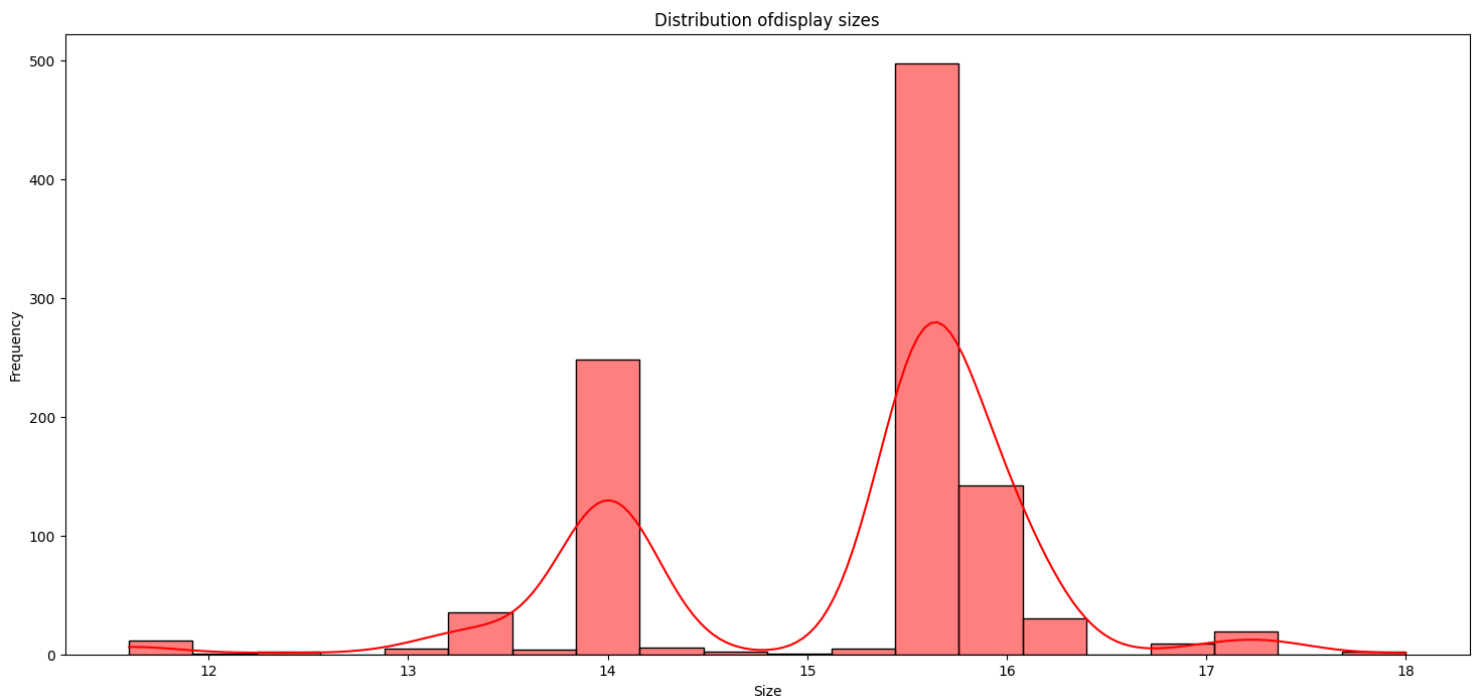


Рисунок 4.1.11 — Гістограма діагоналі екрану

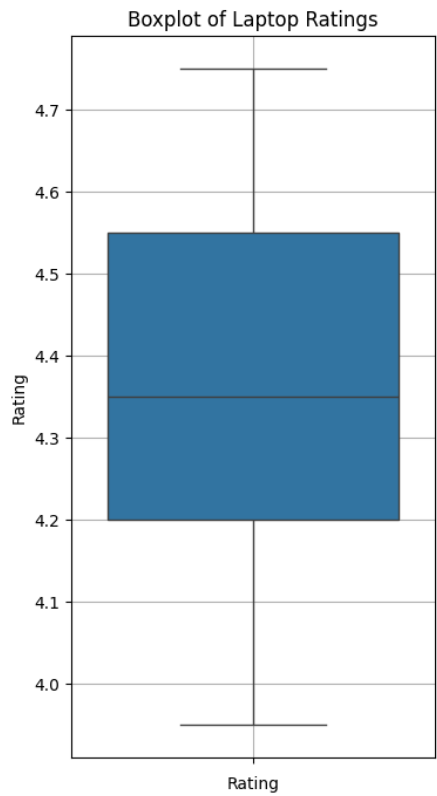


Рисунок 4.1.12 — Коробковий графік оцінок на ноутбуки

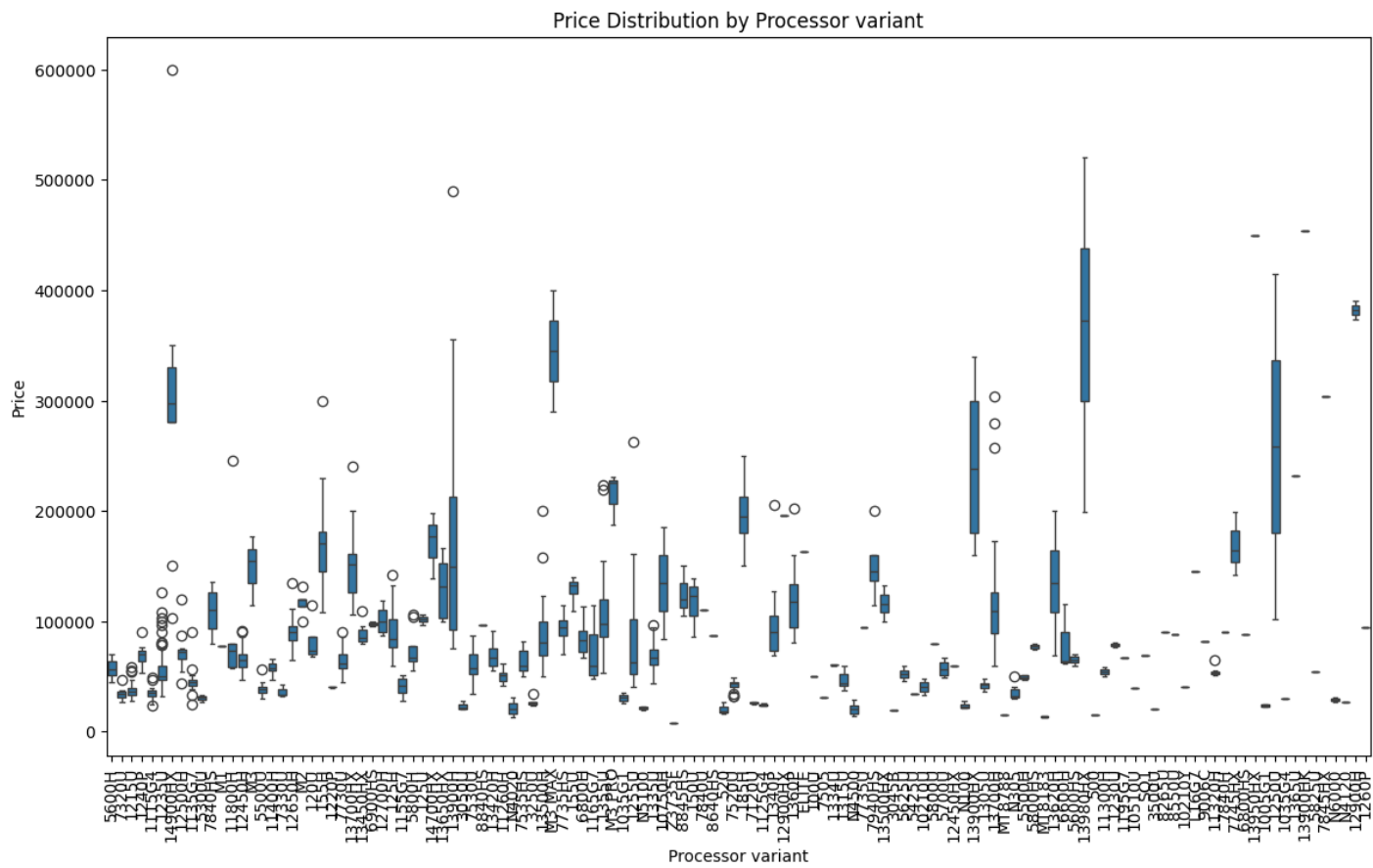


Рисунок 4.1.13 — Коробковий графік ціни на моделі процесорів

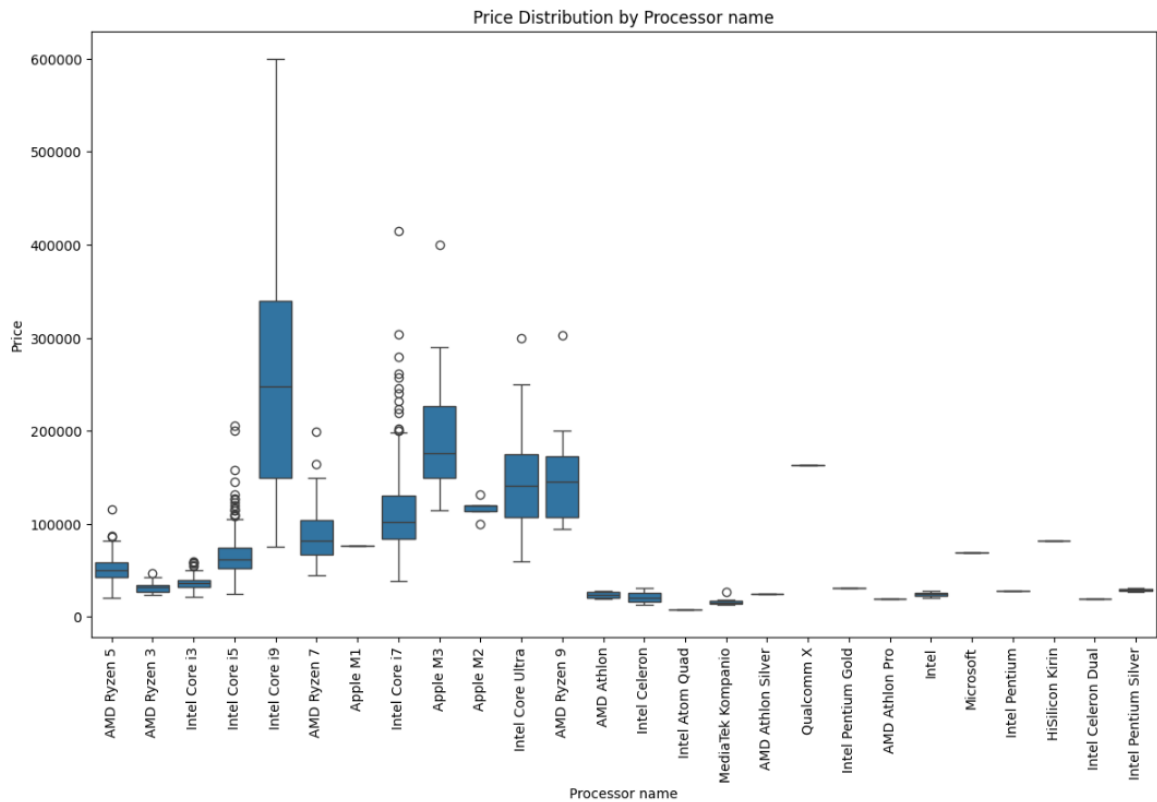


Рисунок 4.1.14 — Коробковий графік ціни на лінійки процесорів

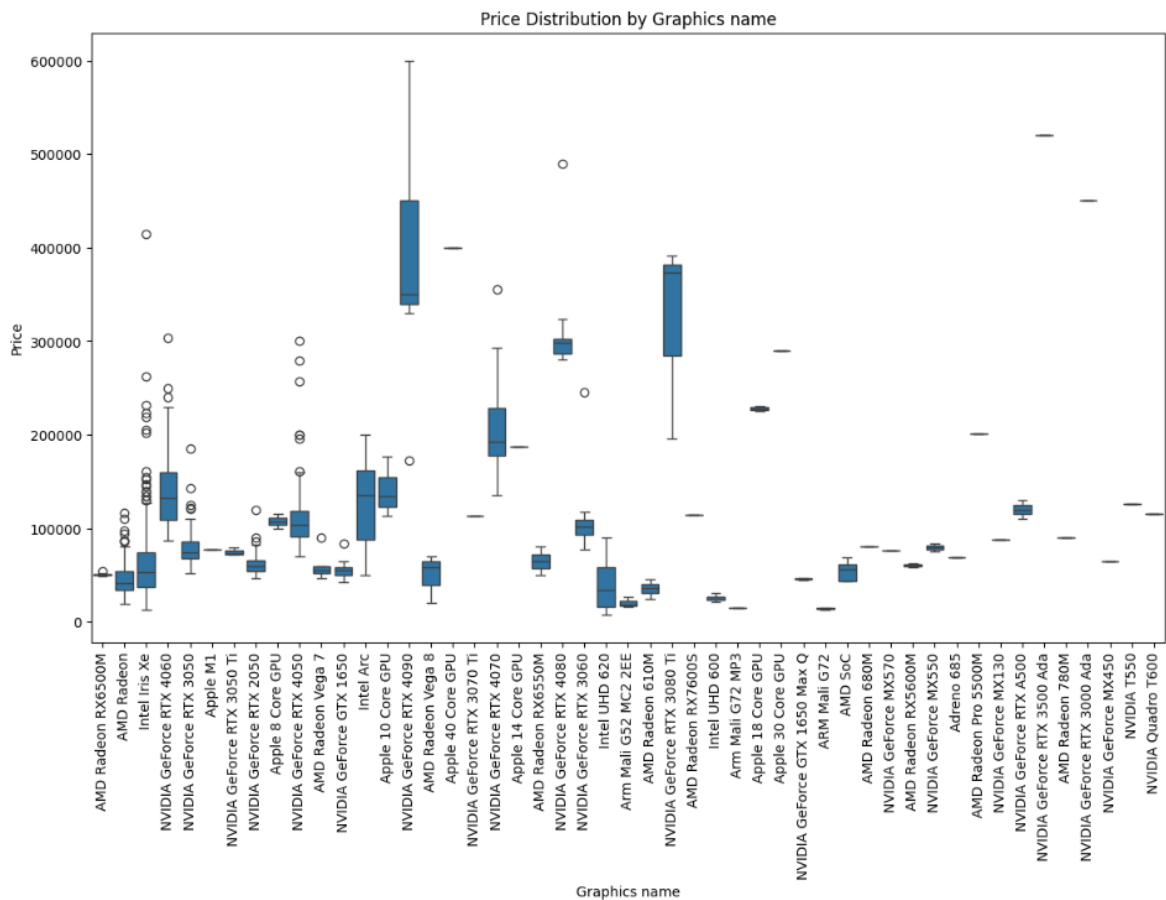


Рисунок 4.1.15 — Коробковий графік ціни на лінійки графічних процесорів

Результати візуалізації даних були використані для прийняття висновків щодо використання тих чи інших предикторів при побудові моделей.

Після візуалізації даних було проведено кореляційний аналіз, який допоміг вибрати декілька найбільш придатних факторів. Для цього аналізу було використано бібліотеку `scipy`. Спочатку було обраховано матрицю коефіцієнтів кореляції Пірсона і візуалізовано її у вигляді теплової карти, яка зображена на рисунку 4.1.16. Ця карта дозволяє виділити колонки “Core\_per\_processor”, “Threads”, “RAM\_GB”, “Storage\_capacity\_GB”, “Graphics\_GB”, “ppi”, “Horizontal\_pixel” та “Vertical\_pixel” як потенційні предиктори ціни.

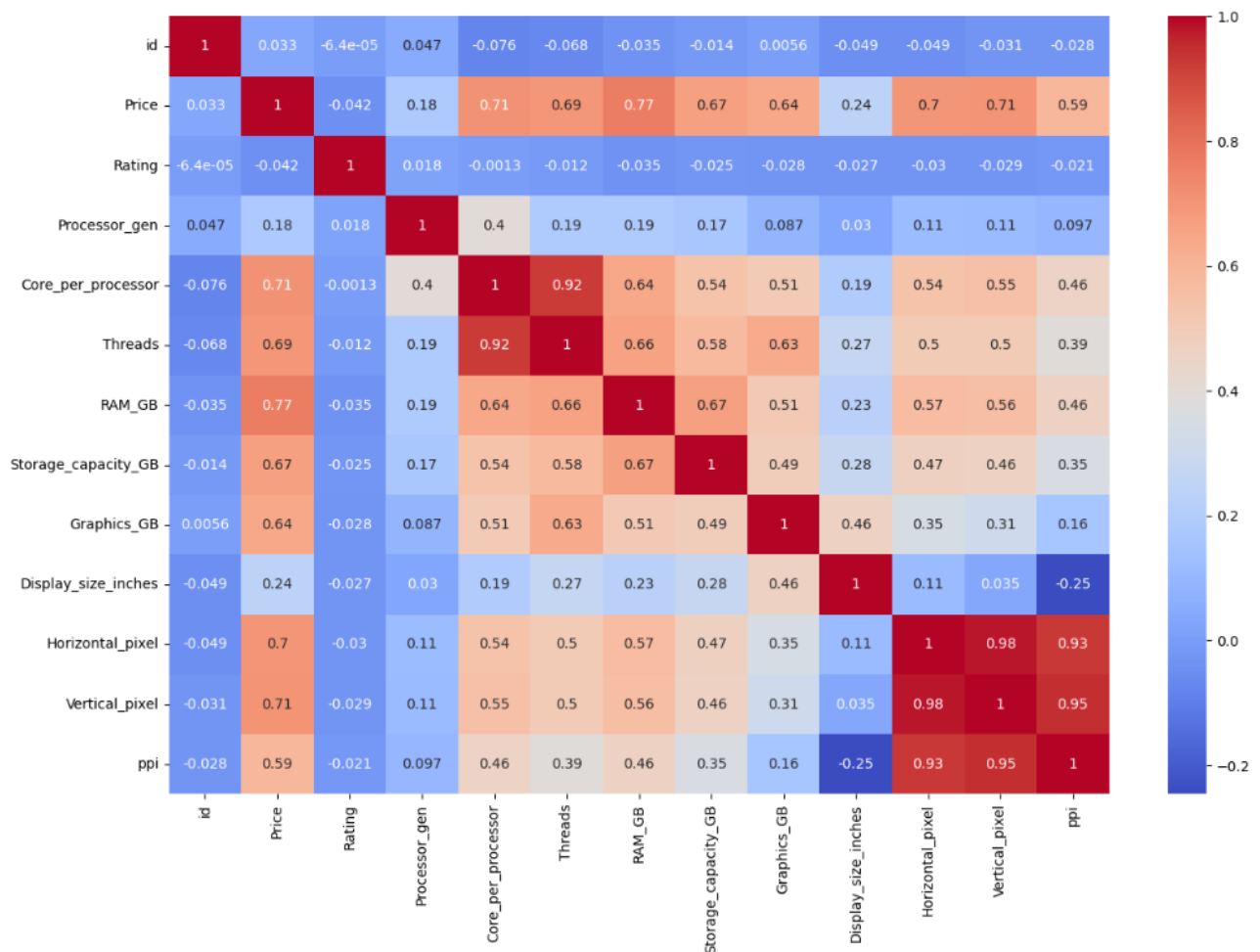


Рисунок 4.1.16 — Теплова карта коефіцієнтів кореляції Пірсона

Коефіцієнт кореляції Пірсона вимагає нормальності розподілу досліджуваних характеристик. За допомогою критерію Шапіро-Уилка обрані ознаки було перевірено на нормальність розподілу (рис. 4.1.17)



```

Analyzing predictor: Core_per_processor
Shapiro test: statistic = 0.9154046177864075, p-value = 2.0561309240343755e-23

Analyzing predictor: Threads
Shapiro test: statistic = 0.917628824710846, p-value = 3.9710731009741047e-23

Analyzing predictor: RAM_GB
Shapiro test: statistic = 0.6817833185195923, p-value = 5.369887819169845e-40

Analyzing predictor: Storage_capacity_GB
Shapiro test: statistic = 0.6395811438560486, p-value = 8.550723229310034e-42

Analyzing predictor: Graphics_GB
Shapiro test: statistic = 0.6878281831741333, p-value = 1.007389262107718e-39

Analyzing predictor: Horizontal_pixel
Shapiro test: statistic = 0.6342617273330688, p-value = 5.2212380780742684e-42

Analyzing predictor: Vertical_pixel
Shapiro test: statistic = 0.6966042518615723, p-value = 2.5565485364926968e-39

Analyzing predictor: ppi
Shapiro test: statistic = 0.7369532585144043, p-value = 2.508091299284713e-37

```

Рисунок 4.1.17 — Результати проведення тесту Шапіро-Уилка

За проведенням аналізом очевидно, що жодна з досліджуваних характеристик не розподілена нормально. Здогадку про потенційні предиктори перевірено за допомогою теплової карти, що побудована на основі матриці коефіцієнтів кореляції Спірмена (рис. 4.1.18).

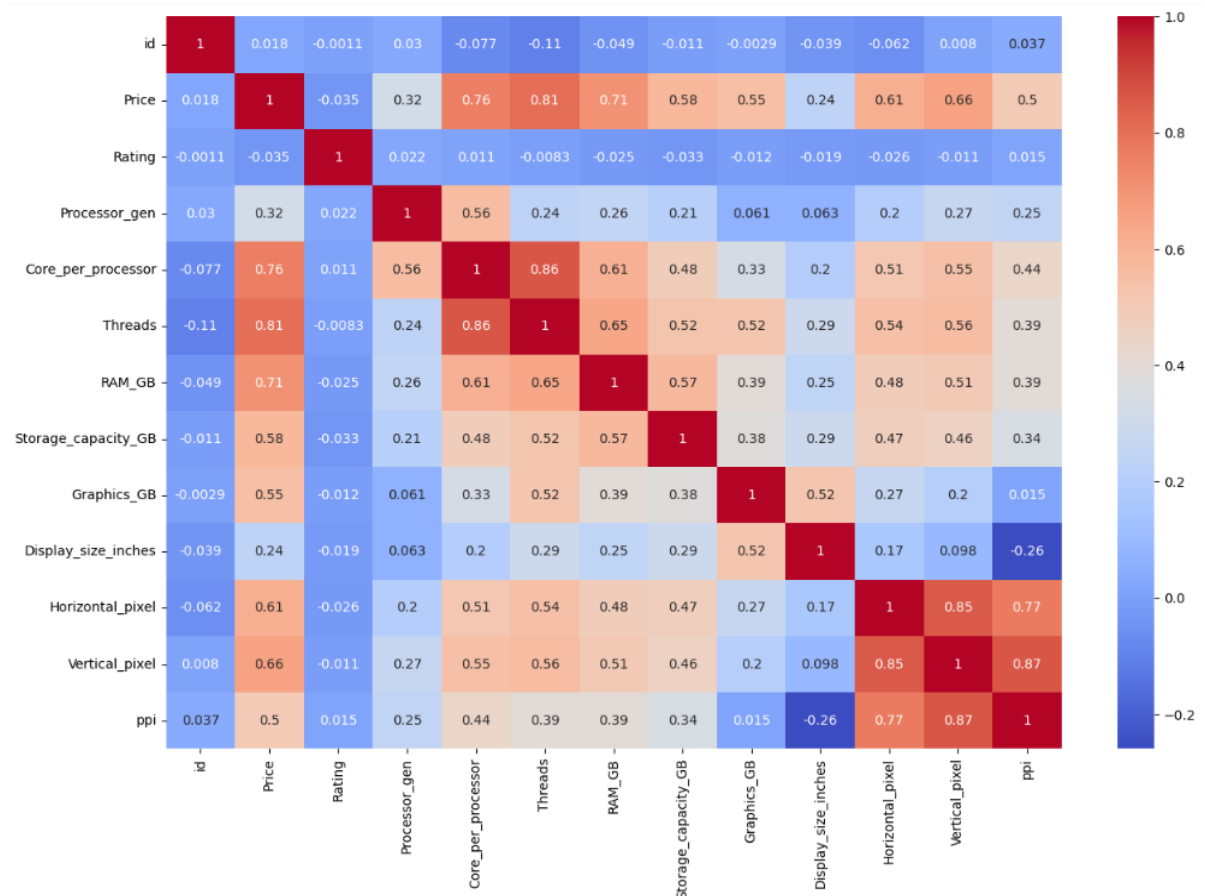


Рисунок 4.1.18 — Теплова карта коефіцієнтів кореляції Спірмена

Здогадки про важливість обраних потенційних факторів підтвердилися завдяки тепловій карті кореляції Спірмена (рис. 4.1.18), гістограмі кількості ядер процесора (рис. 4.1.5), гістограмі розподілу об'єму відеопам'яті (рис. 4.1.6), гістограмі об'єму накопичувача (рис. 4.1.7) та гістограмі об'єму оперативної пам'яті (рис. 4.1.9).

На рисунку 4.1.19 візуалізовано графіки розсіювання обраних факторів

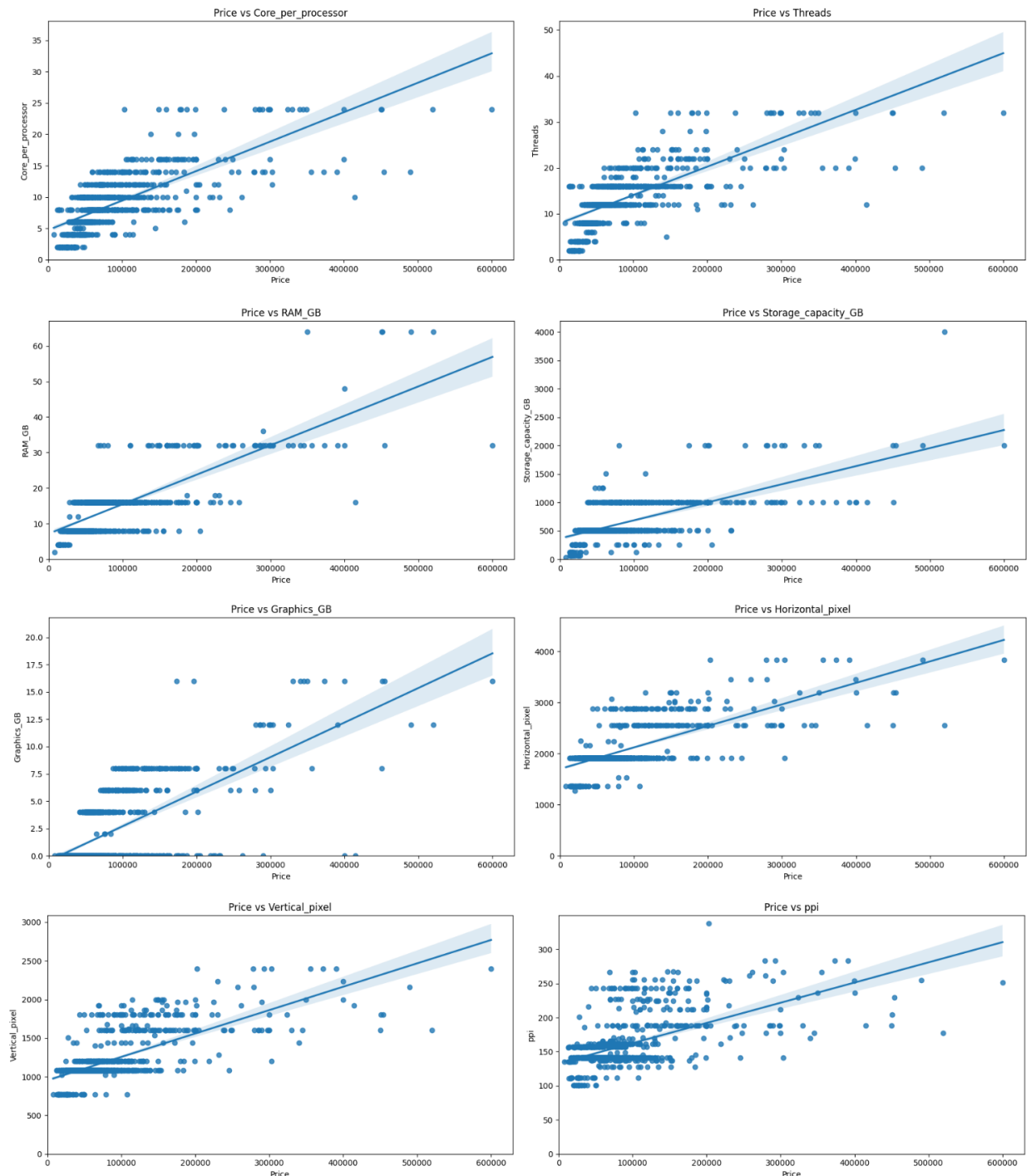


Рисунок 4.1.19 — Графіки розсіювання потенційних факторів

Висновок кореляційного аналізу було сформульовано так:

- 1) В якості потенційних предикторів ціни 'Price' обрано фактори "Core\_per\_processor", "Threads", "RAM\_GB", "Storage\_capacity\_GB", "Graphics\_GB", "Horizontal\_pixel", "Vertical\_pixel" та "ppi".
- 2) Кореляція між 'Price' та 'Core\_per\_processor', 'Threads', 'RAM\_GB' сильна
- 3) Кореляція між 'Price' та 'Storage\_capacity\_GB', 'Graphics\_GB', 'Horizontal\_pixel', 'Vertical\_pixel' помітна
- 4) Кореляція між 'Price' та 'ppi' помірна

Для зменшення кількості числових предикторів було проведено факторний аналіз за допомогою бібліотеки sklearn. Цей метод використовується для вивчення прихованих структурних відносин між змінними в наборі даних. Він допомагає виявити приховані змінні, які пояснюють кореляції між вихідними змінними. Основні цілі факторного аналізу – скорочення розмірності, виявлення прихованих факторів та покращення інтерпретованості даних.

Перед безпосереднім факторним аналізом було проведено оцінку кількості факторів за допомогою scree plot (рис. 4.1.20). За «правилом ліктя» обрано кількість факторів, яка дорівнює 3.

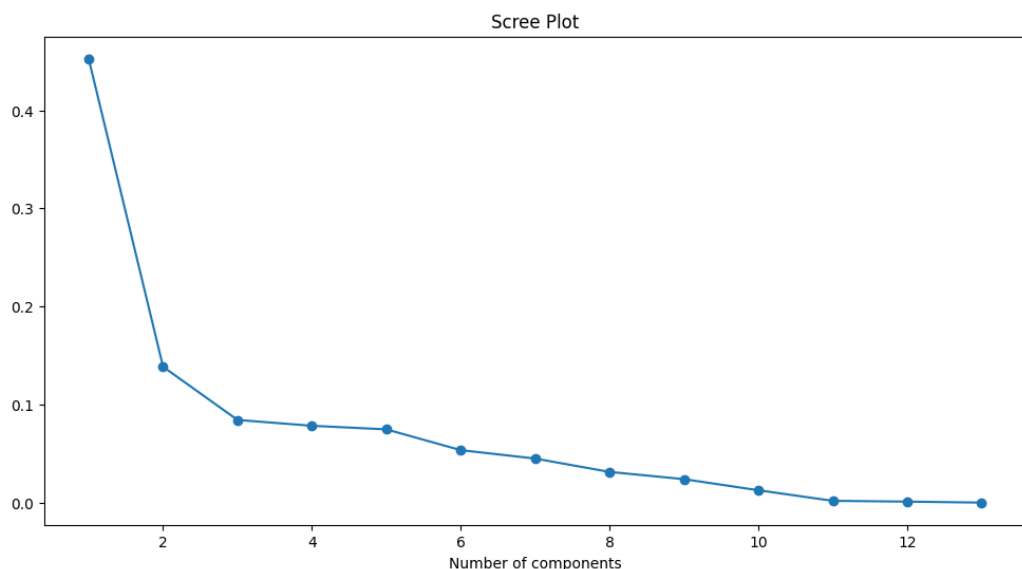


Рисунок 4.1.20 — Scree plot

Результати факторного аналізу зображено на рисунку 4.1.21. Було зроблено наступні висновки:

- 1) Фактор 1 описує характеристики екрану (ppi, Vertical\_pixel, Horizontal\_pixel)
- 2) Фактор 2 описує характеристики компонентів (Core\_per\_processor, Threads)
- 3) Фактор 3 описує характеристики пам'яті (RAM\_GB, Storage\_capacity\_GB, Graphics\_GB).

Факторний аналіз дозволив зменшити кількість факторів у 2 рази. Замість 8 підходящих предикторів обрано 4: 'Threads', 'Vertical\_pixel', 'RAM\_GB' та 'Graphics\_GB'.

	Factor1	Factor2	Factor3
id	-0.013976	-0.052738	-0.004384
Price	0.539530	0.444661	0.579417
Rating	-0.029543	0.024078	-0.052533
Processor_gen	0.032482	0.342945	0.000427
Core_per_processor	0.278271	0.941977	0.267739
Threads	0.312166	0.703479	0.456584
RAM_GB	0.412262	0.445816	0.511621
Storage_capacity_GB	0.319207	0.361198	0.527059
Graphics_GB	0.126510	0.296222	0.722183
Display_size_inches	-0.137249	0.046317	0.602014
Horizontal_pixel	0.911834	0.206739	0.251260
Vertical_pixel	0.935684	0.234432	0.186565
ppi	0.979855	0.232828	-0.093314

Рисунок 4.1.21 — Результати факторного аналізу

Після визначення числових факторів, слід знайти найкращі категоріальні. Для цього було проведено дисперсійний аналіз за допомогою бібліотеки scіру та критерію Краскела-Уолліса. Результат аналізу зображено на рисунку 4.1.22.

Висновки аналізу можна зробити наступні:

- 1) Ознаки "Name" та "Processor\_variant" для аналізу не підходять, адже вони мають багато значень, що дуже рідко зустрічаються (рис. 4.1.13). Використання таких факторів точно призведе до перенавчення моделей, тому їх слід проігнорувати

- 2) Ознаки "Graphics\_integrated", "Touch\_screen", "Processor\_brand", "Storage\_type" явно не підходять для аналізу через невелику різницю між групами. Їх слід проігнорувати
- 3) Ознаки "Processor\_name", "Graphics\_name", "RAM\_type", "Graphics\_brand" та "Brand" можуть стати чудовими предикторами
- 4) Ознака "Operating\_system" має нижчі показники H\_statistic і потенційно може бути предиктором

Feature	H_statistic	p_value
Name	1019.000000	4.941086e-01
Processor_variant	888.820201	1.983983e-115
Processor_name	740.309818	4.545905e-140
Graphics_name	542.856813	5.248217e-85
RAM_type	398.561313	2.777476e-80
Graphics_brand	324.795907	4.652696e-68
Brand	239.137874	1.869251e-34
Operating_system	105.940749	2.594336e-19
Graphics_integrated	82.000614	1.360444e-19
Touch_screen	76.411183	2.303431e-18
Processor_brand	68.915485	6.822289e-13
Storage_type	37.591181	3.449734e-08

Рисунок 4.1.22 — Результати дисперсійного аналізу

Отже, після візуалізації даних, виконання кореляційного, факторного та дисперсійного аналізу було обрано наступні фактори для передбачення відгуку:

- 1) 'Threads'
- 2) 'Vertical\_pixel'
- 3) 'RAM\_GB'
- 4) 'Graphics\_GB'
- 5) 'RAM\_type'
- 6) 'Brand'
- 7) 'Processor\_name'
- 8) 'Graphics\_brand'

#### 4.2. Обґрунтування вибору методів аналізу даних

Для вирішення задачі передбачення ціни на ноутбук на основі його ключових компонентів було обрано наступні методи інтелектуального аналізу даних: Linear Regression, Ridge Regression, Random Forest. Вибір кожної з цих моделей необхідно обґрунтувати.

Лінійна регресія – це найбільш фундаментальний та широко використовуваний метод машинного навчання. Вона застосовується для моделювання залежності між залежною змінною та однією або декількома незалежними змінними шляхом підбору лінійної функції, яка найкраще описує ці залежності. Переваги лінійної регресії наступні:

- 1) Лінійна регресія проста для розуміння та використання. Результат легко інтерпретується, адже коефіцієнти моделі показують, наскільки зміна незалежної змінної впливає на відгук.
- 2) Класична модель лінійної регресії не вимагає гіперпараметрів для підбору, що спрощує процес налаштування моделі.
- 3) Лінійна регресія може бути швидко навчена навіть на великих наборах даних, що робить її придатною для завдань, які потребують швидкого отримання результатів.

Але лінійна регресія має наступні недоліки:

- 1) Якщо залежність між відгуком та факторами нелінійна, точність моделі може сильно впасти.
- 2) Лінійна регресія достатньо чутлива до викидів та перенавчання.
- 3) Ненормальність розподілу залишків може призвести до зменшення надійності моделі
- 4) Лінійна регресія чутлива до мультиколінеарності. Висока кореляція між незалежними змінними може зробити оцінки коефіцієнтів нестабільними та важко інтерпретованими.

Гребенева регресія є розширенням лінійної. Відрізняється вона від класичної лінійної регресії регуляризацією через додавання штрафів за величину коефіцієнтів моделі. За контроль степеню регуляризації відповідає гіперпараметр  $\alpha$ . Така зміна допомагає уникати перенавчання.

Переваги гребеневої регресії наступні:

- 1) Зниження чутливості до мультиколінеарності. Гребенева регресія додає штраф за величину коефіцієнтів, що зменшує їх значення та стабілізує оцінки.
- 2) Регуляризація допомагає уникнути перенавчання, покращуючи узагальнюючу здатність моделі.
- 3) Гребенева регресія менш чутлива до шуму та викидів у даних порівняно з лінійною регресією

Але гребенева регресія має свої недоліки:

- 1) Результати складніше інтерпретуються.
- 2) Підбір гіперпараметру регуляризації може бути довгим і затратним.
- 3) Гребенева регресія чутлива до масштабу ознак. Перед застосуванням моделі необхідно стандартизувати дані, щоб штраф рівномірно розподілявся між ознаками.
- 4) Як і лінійна регресія, гребенева модель може погано справлятися із нелінійними залежностями і втрачати точність.

Вибір гребеневої регресії може значно збільшити точність передбачення в порівнянні з лінійною. Регуляризація здатна зменшити чутливість до недоліків в навчальних даних.

Random Forest – це універсальний ансамблевий метод машинного навчання, що використовується як для завдань класифікації, так і регресії. Метод побудований на основі створення великої кількості дерев рішень, об'єднаних разом для покращення точності та стійкості моделі.

Ця модель має наступні переваги:

- 1) Завдяки методу Bootstrap Aggregating та випадковості ознак модель стійка до перенавчання.
- 2) Випадковий ліс менш чутливий до шуму та викидів у даних, оскільки усереднює результати дерев, знижуючи вплив аномалій.
- 3) Модель має високу точність передбачень.
- 4) Модель чудово справляється з даними, що містять велику кількість ознак.

Випадковий ліс має і свої недоліки

- 1) Основний недолік – це час навчання. Навчання великої кількості дерев може бути дуже затратним і займати значний час, особливо на великих наборах даних.
- 2) Модель важко інтерпретується.
- 3) Модель вимагає великого обсягу пам'яті

Вибір випадкового лісу в якості моделі передбачення ціни може покращити результати прогнозування, якщо у даних є проблема з лінійністю залежності відгуку від факторів або якщо існує проблема мультиколінеарності. Випадковий ліс достатньо стійкий до шуму та викидів у даних, що може зробити його надійним вибором при вирішенні задачі регресії.

#### 4.3. Аналіз отриманих результатів для моделі Linear Regression

Для аналізу даних використовувалася бібліотека `sklearn` і її модулі `metrics`, `preprocessing`, `model_selection`, `linear_model` та `ensemble`.

Всі обрані моделі було навчено на 40% вхідних даних. Розподілення даних на тренувальні та навчальні відбувалося завдяки функції `train_test_split` модулю `sklearn.model_selection`. Для навчання моделей на категоріальних факторах використовувався метод One-hot encoding.

Для оцінки моделей використовувалися оцінки MAPE (Mean Absolute Percentage Error), RMSE (Root Mean Squared Error) та  $R^2$ .



Для створення та навчання моделі лінійної регресії було використано модуль `sklearn.linear_model`. Модель навчено на навчальних даних, а завдяки тестовим даним оцінено її точність. Результати оцінки на рисунку 4.3.1. Для візуальної оцінки точності було використано діаграму розподілу (рис. 4.3.2). Вона допомагає оцінити, наскільки точно модель вгадувала ціну ноутбуків.

```
model_lr = LinearRegression()
model_lr.fit(x_train, y_train)

y_pred = model_lr.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mape = mean_absolute_percentage_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}')
```

MAPE = 0.20507357024092224  
RMSE = 25128.30362838063  
R^2 = 0.8630080019656385

Рисунок 4.3.1 — Результати оцінки точності моделі лінійної регресії

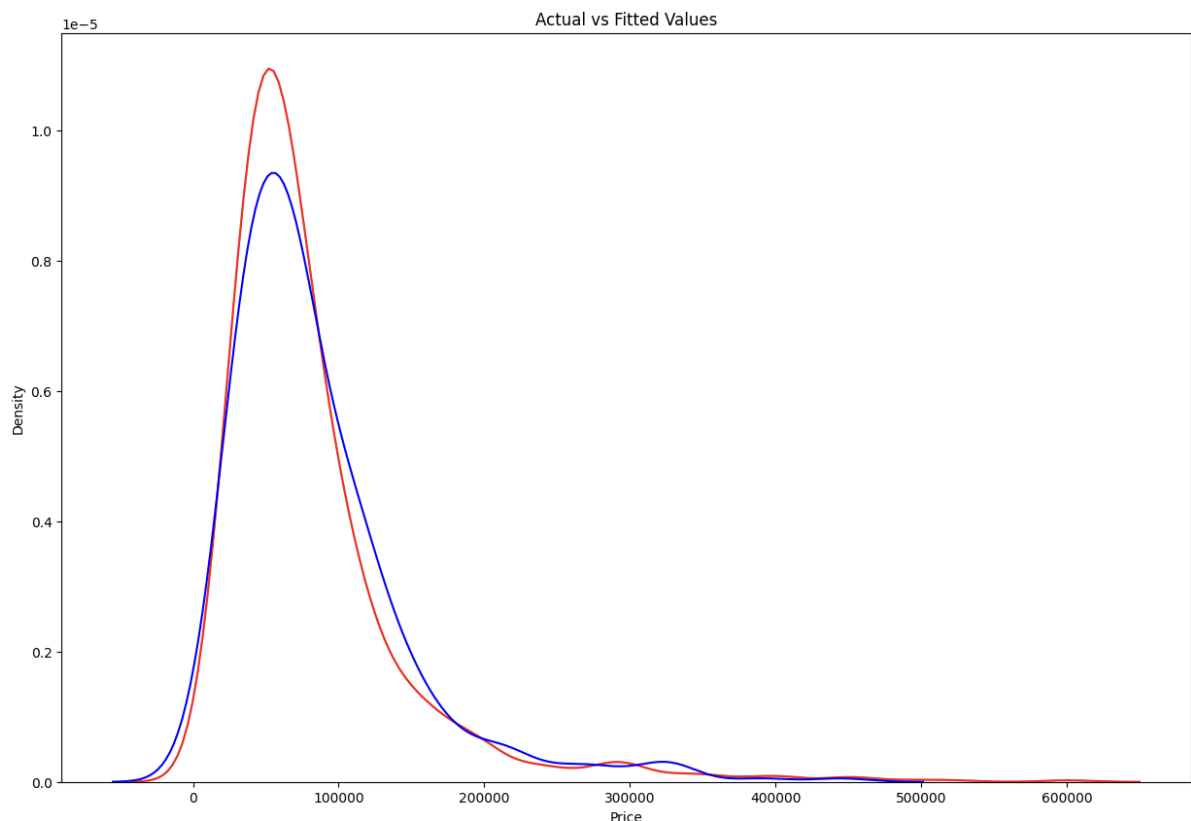
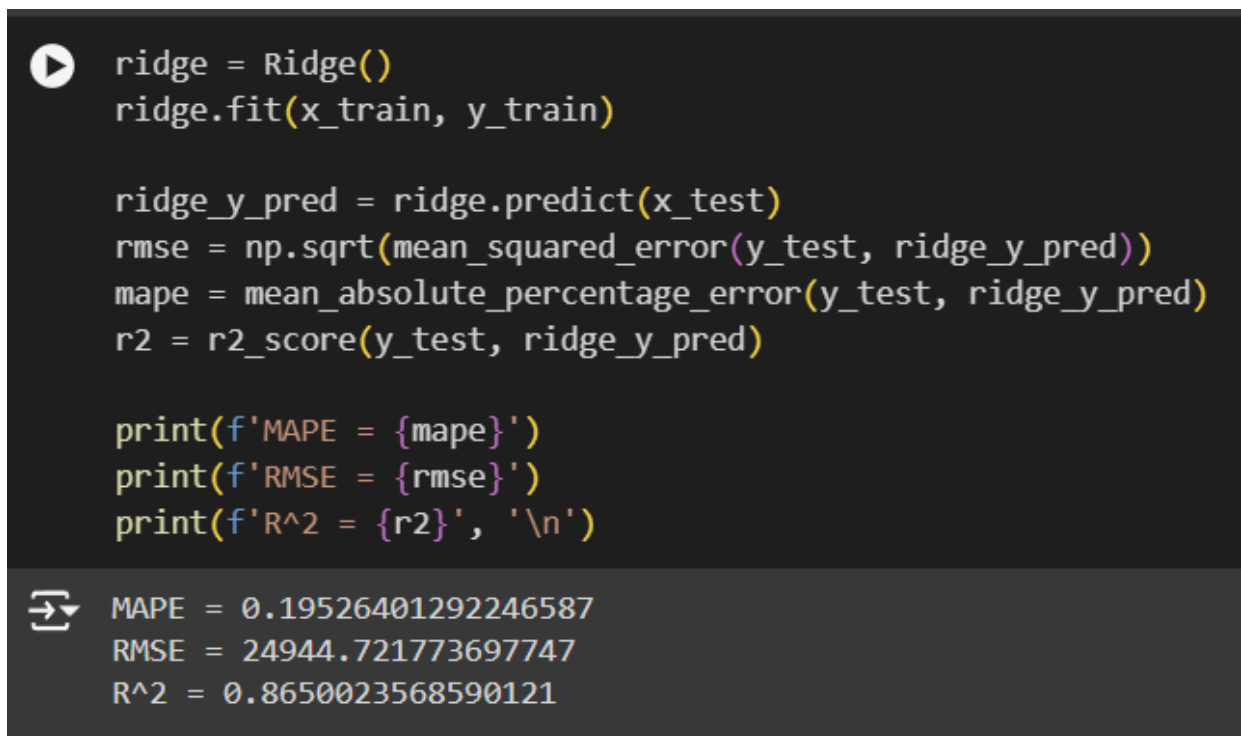


Рисунок 4.3.2 — Графік розподілу лінійної регресії

Висновок можна зробити наступний: лінійна регресія непогано справляється з передбаченням ціни на ноутбук, хоча графік розподілу показує, що модель має значні неточності на деяких проміжках ціни. Отримано наступні оцінки точності:  $\text{MAPE} = 20.5\%$ ,  $\text{RMSE} = 25128$ ,  $R^2 = 0.863$ .

#### 4.4. Аналіз отриманих результатів для моделі Ridge Regression

Для створення та навчання моделі гребеневої регресії було використано модуль `sklearn.linear_model`. Модель було навчено на тих самих даних, що і лінійну регресію. Результат навчання та оцінки точності зображено на рисунку 4.4.1. Отримано наступні результати оцінок точності:  $\text{MAPE} = 19.5\%$ ,  $\text{RMSE} = 24944$ ,  $R^2 = 0.865$ . На рисунку 4.4.2 зображено графік розподілу гребеневої регресії.



```

▶ ridge = Ridge()
  ridge.fit(x_train, y_train)

  ridge_y_pred = ridge.predict(x_test)
  rmse = np.sqrt(mean_squared_error(y_test, ridge_y_pred))
  mape = mean_absolute_percentage_error(y_test, ridge_y_pred)
  r2 = r2_score(y_test, ridge_y_pred)

  print(f'MAPE = {mape}')
  print(f'RMSE = {rmse}')
  print(f'R^2 = {r2}', '\n')
⇨ MAPE = 0.19526401292246587
   RMSE = 24944.721773697747
   R^2 = 0.8650023568590121

```

Рисунок 4.4.1 — Результати оцінки точності моделі гребеневої регресії

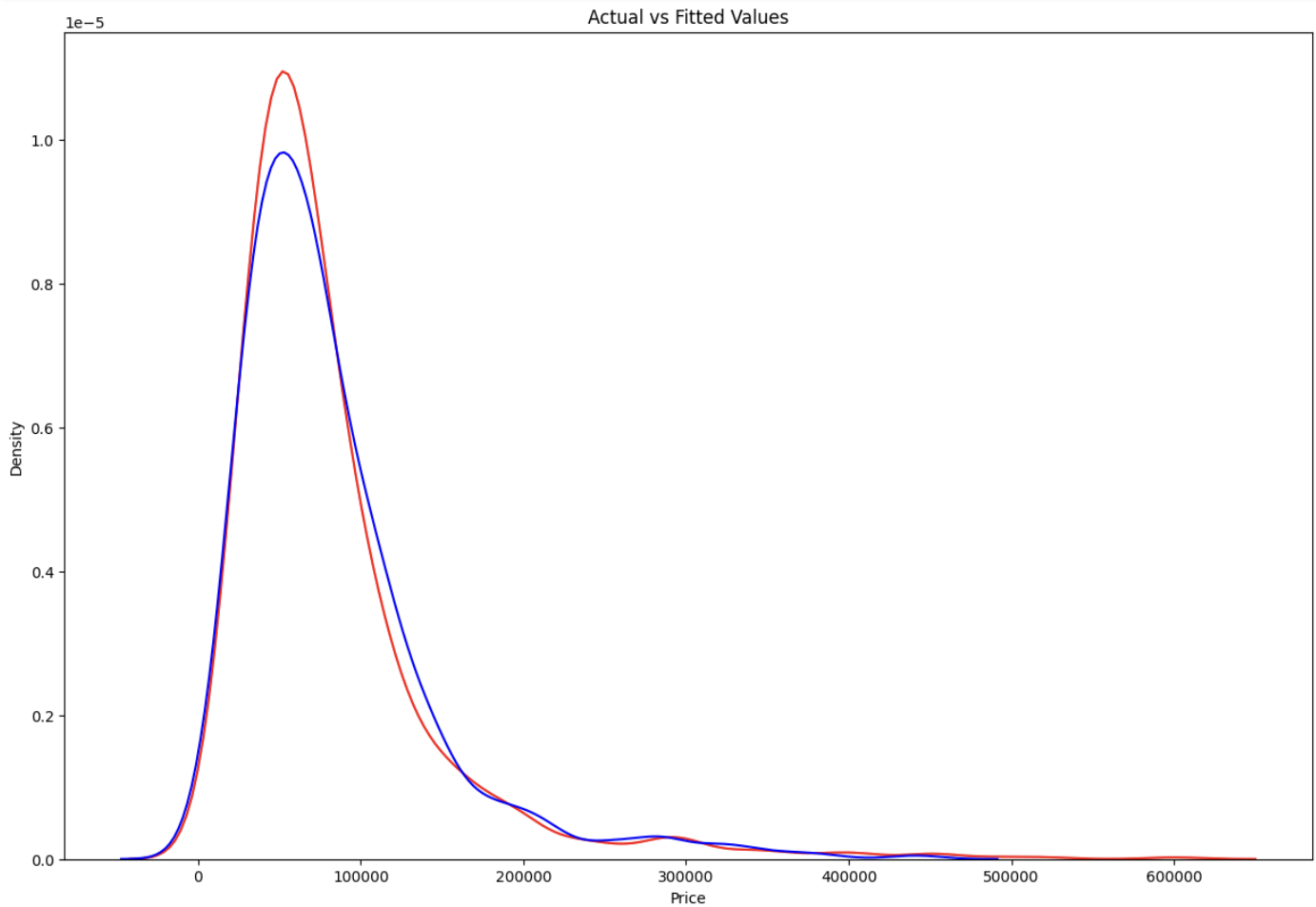


Рисунок 4.4.2 — Графік розподілу гребеневої регресії

Було здійснено спробу покращити результати точності завдяки використанню стандартизації та поліноміальної трансформації даних, а також підбору гіперпараметру  $\alpha$  через пошук по сітці (Grid Search) з модулю `sklearn.model_selection`.

Такі дії помітно покращили результати точності (рис. 4.4.3). Отримано наступні результати оцінок точності:  $\text{MAPE} = 17.1\%$ ,  $\text{RMSE} = 23082$ ,  $R^2 = 0.884$ . На рисунку 4.4.4 зображено графік розподілу гребеневої регресії.

```
[ ] poly = PolynomialFeatures(degree = 2, include_bias = False)
x_poly = poly.fit_transform(x_transformed)
x_poly_scaled = StandardScaler().fit_transform(x_poly)

x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(x_poly_scaled, y, test_size = 0.4, random_state = 1)

[ ] param_grid = {'alpha': np.logspace(-2, 10, 13)}
ridge = Ridge()
grid_search = GridSearchCV(estimator = ridge, param_grid = param_grid, cv = 5, scoring = 'neg_mean_squared_error', n_jobs = -1, verbose = 2)
grid_search.fit(x_train_p, y_train_p)

best_ridge = grid_search.best_estimator_
ridge_y_pred = best_ridge.predict(x_test_p)
rmse = np.sqrt(mean_squared_error(y_test_p, ridge_y_pred))
mape = mean_absolute_percentage_error(y_test_p, ridge_y_pred)
r2 = r2_score(y_test_p, ridge_y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}')
```


 Fitting 5 folds for each of 13 candidates, totalling 65 fits  
 MAPE = 0.1710438291815642  
 RMSE = 23082.171489261153  
 R^2 = 0.8844094887999928

Рисунок 4.4.3 — Результати оцінки точності моделі гребеневої регресії

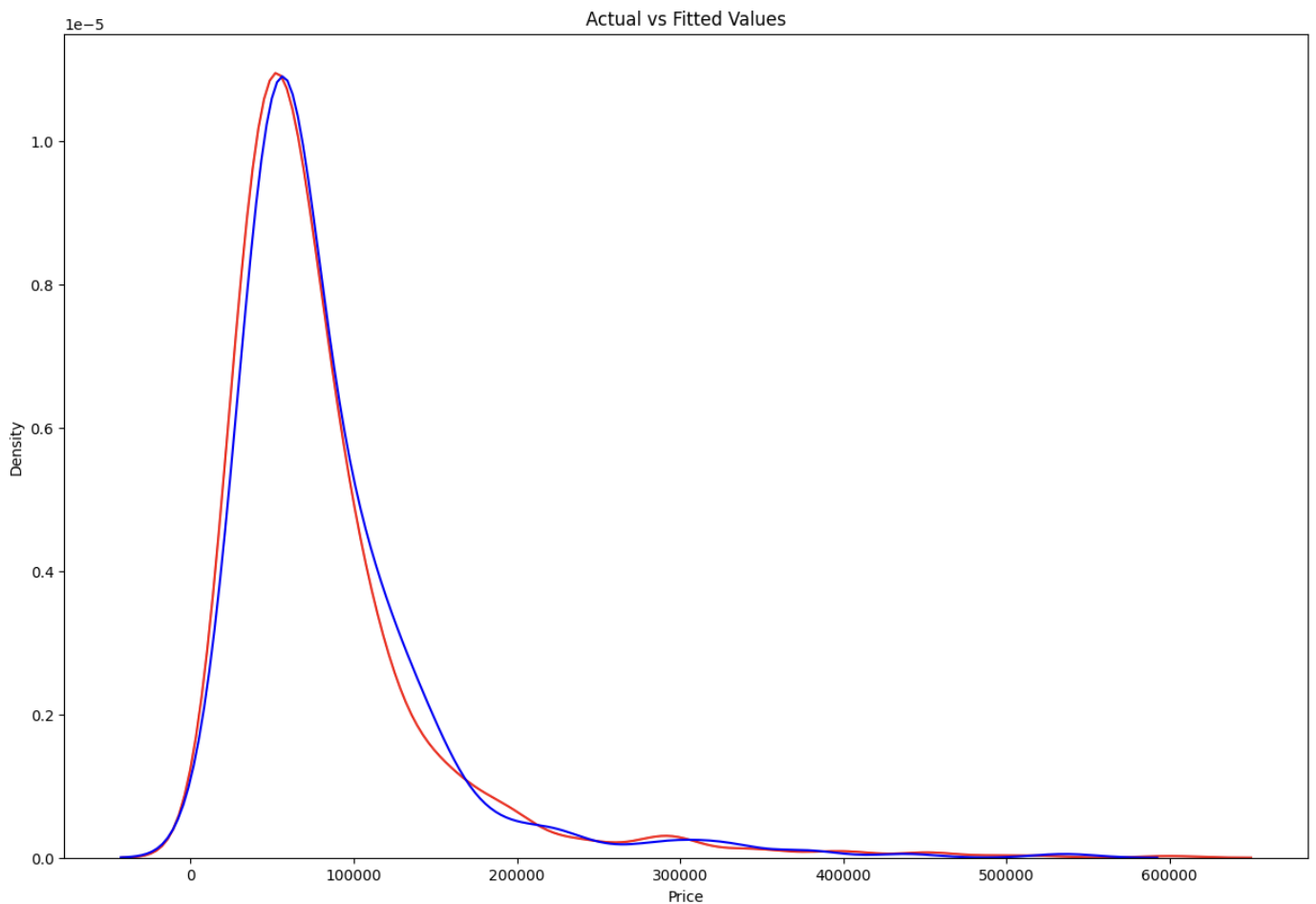


Рисунок 4.4.4 — Графік розподілу гребеневої регресії

#### 4.5. Аналіз отриманих результатів для моделі Random Forest

Для створення та навчання моделі випадкового лісу було використано модуль `sklearn.linear_model`. Модель було навчено на тих самих даних, що і лінійну регресію. Результат навчання та оцінки точності зображено на рисунку 4.5.1. Отримано наступні результати оцінок точності:  $\text{MAPE} = 17.3\%$ ,  $\text{RMSE} = 26949$ ,  $R^2 = 0.842$ . На рисунку 4.5.2 зображено графік розподілу моделі.

```
[ ] model_rf = RandomForestRegressor(n_estimators = 100, random_state = 1)
    model_rf.fit(x_train, y_train)

    y_pred = model_rf.predict(x_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mape = mean_absolute_percentage_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f'MAPE = {mape}')
    print(f'RMSE = {rmse}')
    print(f'R^2 = {r2}')
```

MAPE = 0.17313189287210343  
 RMSE = 26949.834627046614  
 R^2 = 0.8424272672598802

Рисунок 4.5.1 — Результати оцінки точності моделі випадкового лісу

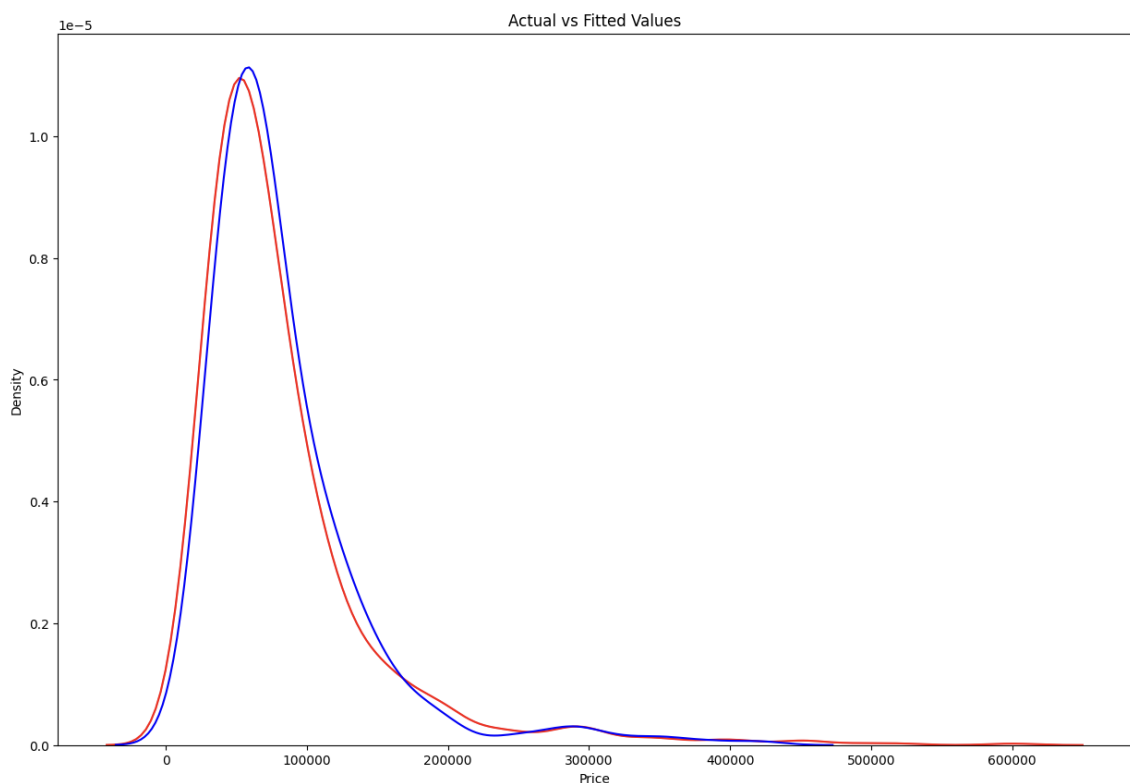


Рисунок 4.5.2 — Графік розподілу моделі Random Forest

Було здійснено спробу покращити результати точності завдяки підбору гіперпараметрів `n_estimators` (кількості дерев), `max_depth` (максимальної глибини дерева), `min_samples_split` (мінімальна кількість зразків, необхідних для розбиття вузла), `min_samples_leaf` (мінімальна кількість зразків, необхідних для формування листа) та `max_features` (максимальна кількість ознак, що розглядатимуться для розбиття вузла) через пошук по сітці (Grid Search) з модулю `sklearn.model_selection`.

Такі дії жодним чином не покращили результати точності (рис. 4.5.3). Отримано наступні результати оцінок точності:  $\text{MAPE} = 17.3\%$ ,  $\text{RMSE} = 26949$ ,  $R^2 = 0.842$ .

Друга спроба покращити результат полягала у стандартизації та поліноміальній трансформації даних. Для цього при навчанні моделі було використано тренувальні дані, використані при навчанні гребеневої регресії. Після цих дій точність моделі піросла (рис. 4.5.4). Отримано наступні результати оцінок точності:  $\text{MAPE} = 17.2\%$ ,  $\text{RMSE} = 26105$ ,  $R^2 = 0.852$ . Візуально оцінити покращення можна на рисунку 4.5.5.

```

param_grid = {
    'n_estimators': [50, 100, 150, 200, 500, 700, 1000],
    'max_depth': [5, 10, 20, 30, None],
    'min_samples_split': [2, 5, 7, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

rf = RandomForestRegressor(random_state = 1)

grid_search = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 4, n_jobs = -1, verbose = 2, scoring = 'neg_mean_squared_error')
grid_search.fit(x_train, y_train)

best_rf = grid_search.best_estimator_

best_rf_y_pred = best_rf.predict(x_test)
best_rf_rmse = np.sqrt(mean_squared_error(y_test, best_rf_y_pred))
best_rf_mape = mean_absolute_percentage_error(y_test, best_rf_y_pred)
best_rf_r2 = r2_score(y_test, best_rf_y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}', '\n')
print(f'Best parameters: {grid_search.best_params_}')

```


 Fitting 4 folds for each of 1680 candidates, totalling 6720 fits  
 MAPE = 0.17313189287210343  
 RMSE = 26949.834627046614  
 R^2 = 0.8424272672598802  
 Best parameters: {'max\_depth': 30, 'max\_features': 'sqrt', 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 150}

Рисунок 4.5.3 — Спроба підібрати оптимальні параметри Random Forest

```
[ ] model_rf = RandomForestRegressor(n_estimators = 100, random_state = 1)
model_rf.fit(x_train_p, y_train_p)

y_pred = model_rf.predict(x_test_p)
rmse = np.sqrt(mean_squared_error(y_test_p, y_pred))
mape = mean_absolute_percentage_error(y_test_p, y_pred)
r2 = r2_score(y_test_p, y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}')
```

MAPE = 0.17166573075668498  
 RMSE = 26105.378550344216  
 R^2 = 0.8521474400882985

Рисунок 4.5.4 — Результати оцінки точності моделі Random Forest

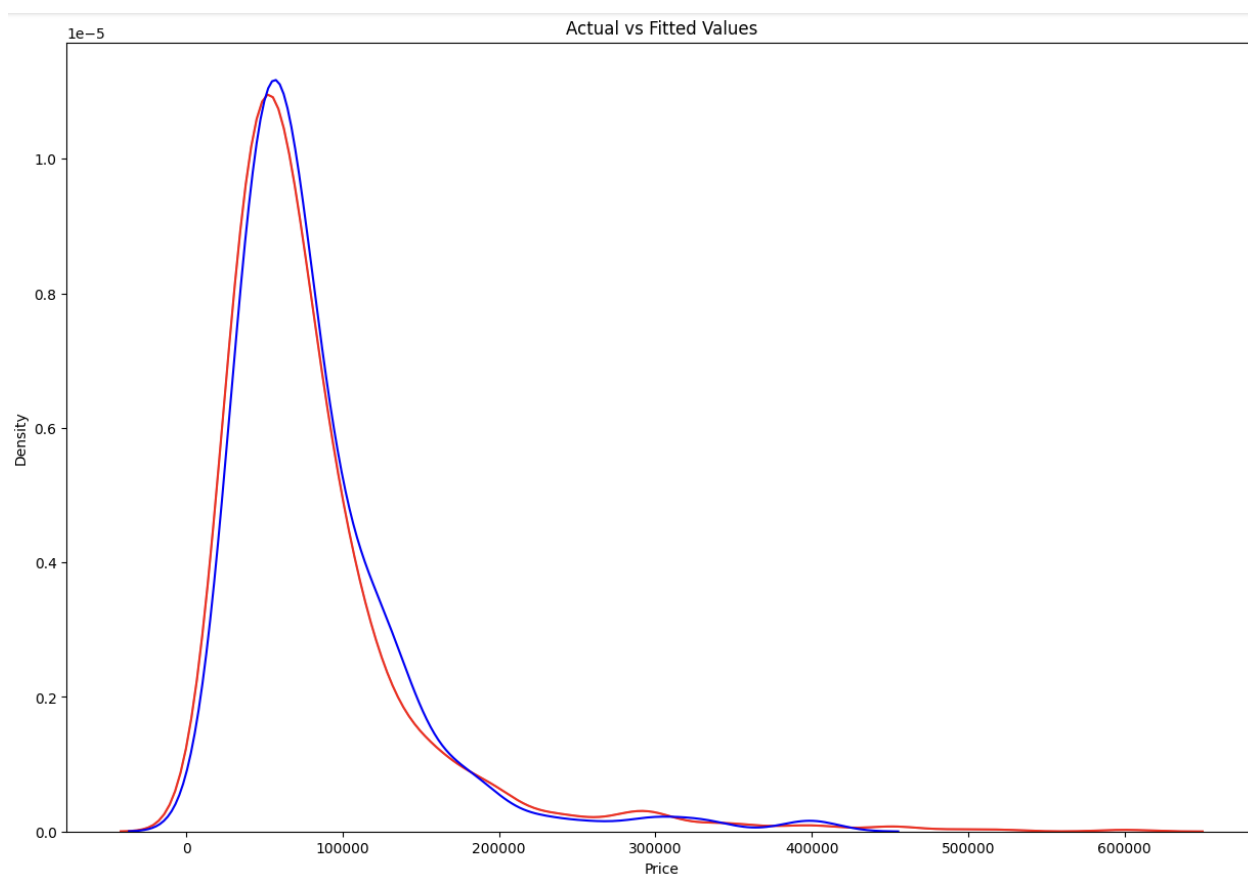


Рисунок 4.5.5 — Графік розподілу моделі Random Forest

4.6. Порівняння отриманих результатів моделей

Після проведення регресійного аналізу обрані моделі слід порівняти. Для цього було зіставлено оцінки якості моделей, а також діаграми розподілу для візуальної оцінки точності моделей.

У таблиці 4.6.1. наведено оцінки якості моделей для порівняння. На рисунку 4.6.1. зображено зіставлені діаграми розподілу для візуальної оцінки.

Таблиця 4.6.1 — Порівняння оцінок моделей

Назва моделі	MAPE	RMSE	R <sup>2</sup>
Лінійна регресія	20.5%	25128	0.863
Гребенева регресія	17.1%	23082	0.884
Випадковий ліс	17.1%	26105	0.852

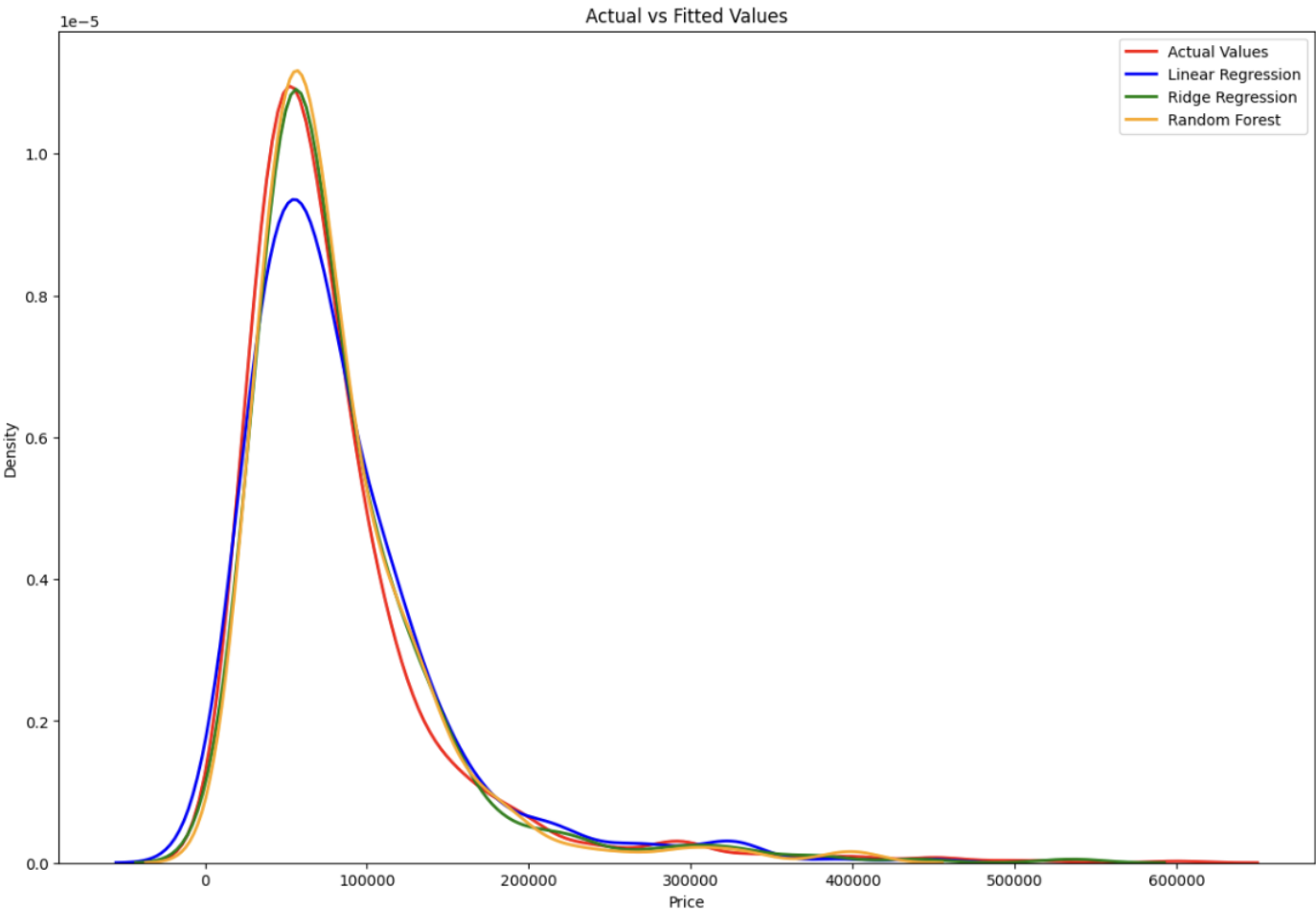


Рисунок 4.6.1 — Графік розподілу обраних моделей



З отриманих даних очевидно, що найкраще із задачею передбачення ціни на ноутбук на основі його ключових компонентів справилася модель гребеневої регресії із пошуком гіперпараметру  $\alpha$  по сітці, стандартизацією та поліноміальною трансформацією даних. Коефіцієнт детермінації у цієї моделі найбільший, а середня абсолютна відсоткова та середня квадратична помилки найменші. Спираючись на графік розподілу, можна зробити той самий висновок: найточнішою виявилася гребенева регресія. На такий результат, скоріше за все, вплинули ключові якості моделі: регуляризація, стійкість до мультиколінеарності, параметр  $\alpha$ , що дозволяє контролювати складність моделі, забезпечуючи оптимальний баланс між зміщенням та дисперсією.

На другому місці по якості передбачення виявилася модель Random Forest. Вона має ту саму оцінку MAPE, але нижчу  $R^2$  та вищу RMSE. Рисунок 4.6.1 дозволяє оцінити високу точність моделі. Скоріше за все, складніша модель випадкового лісу виявилася гіршою за гребеневу регресію через переважання лінійних залежностей у даних. В такому випадку простіші лінійні моделі можуть виявитися кращими за більш складні.

Найгіршою виявилася звичайна лінійна регресія. Хоча результати прогнозування за допомогою цієї моделі все ще високі, графік на рисунку 4.6.1 дозволяє помітити значні неточності передбачення на деяких проміжках ціни. Оцінка MAPE лінійної регресії найбільша серед протестованих моделей, хоча оцінки  $R^2$  та RMSE трохи кращі за оцінки Random Forest.

## ВИСНОВКИ

Метою цієї курсової роботи було створення застосунку для передбачення ціни на ноутбук на основі його ключових компонентів. В ході виконання роботи було обрано дані[1] для аналізу, які було підготовлено до аналізу завдяки заповненню пропусків та виправленню помилок у записах та типах даних. Після обробки даних, на етапі інтелектуального аналізу, завдяки кореляційному та факторному аналізу було обрано числові фактори, а завдяки дисперсійному аналізу було обрано категоріальні предиктори, що могли б бути використані при передбаченні ціни на ноутбук.

Після обрання факторів, що найсильніше впливають на відгук, було побудовано 3 моделі машинного навчання, а саме: Linear Regression, Ridge Regression, Random Forest. Точність кожної моделі було досліджено за допомогою наступних метрик: MAPE (Mean Absolute Percentage Error), RMSE (Root Mean Squared Error) та  $R^2$ . Точність моделей було візуалізовано через діаграми розподілу.

Після оцінки результатів, була спроба покращити моделі гребеневої регресії та випадкового лісу завдяки стандартизації та поліноміальній трансформації даних, а також підбору параметрів. Наслідком цього стало підвищення точності обох моделей.

За результатами оцінок, з передбаченням ціни на ноутбук найкраще справляється гребенева регресія зі стандартизацією та поліноміальною трансформацією даних, а також підбором гіперпараметру  $\alpha$  за допомогою пошуку по сітці. Оцінки цієї моделі наступні: MAPE = 17.1%, RMSE = 23082,  $R^2 = 0.884$ . Моделі лінійної регресії та випадкового лісу показали трохи гірші результати, хоча із задачею передбачення справляються все ще чудово.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Laptop sales price prediction 2024 dataset. Kaggle. URL:  
<https://www.kaggle.com/datasets/siddiquifaiznaeem/laptop-sales-price-prediction-dataset-2024> (дата звернення: 31.05.2024).
2. Using matplotlib. Matplotlib. URL:  
<https://matplotlib.org/stable/users/index.html> (дата звернення: 25.05.2024).
3. An introduction to seaborn. Seaborn. URL:  
<https://seaborn.pydata.org/tutorial/introduction.html>  
(дата звернення: 25.05.2024).
4. User Guide. Supervised learning. Scikit-learn. URL:  
[https://scikit-learn.org/stable/supervised\\_learning.html](https://scikit-learn.org/stable/supervised_learning.html)  
(дата звернення: 25.05.2024).
5. User Guide. Pandas. URL:  
[https://pandas.pydata.org/docs/user\\_guide/index.html#user-guide](https://pandas.pydata.org/docs/user_guide/index.html#user-guide)  
(дата звернення: 25.05.2024).
6. Категориальные признаки. Habr. URL:  
<https://habr.com/ru/articles/666234/> (дата звернення: 01.06.2024).

**ДОДАТОК А - ПРОГРАМНИЙ КОД ОБРОБКИ ДАНИХ**

*Тексти програмного коду першочергової обробки даних,  
очистки, заповнення пропусків та виправлення помилок*

---

(Найменування програми (документа))

*SSD*

---

(Вид носія даних)

*11 арк, 195 Кб*

---

(Обсяг програми (документа), арк, Кб)

*студента групи ІІІ-22 ІІ курсу*

*Нижника Д.С.*

```

import pandas as pd

import re


from google.colab import drive
from google.colab import files


drive.mount('/content/drive')

filename = "/content/drive/My Drive/Laptop_prices.csv"

df = pd.read_csv(filename)

df.head(5)


df.info()


df = df.rename(columns={df.columns[0]: 'id'})
df = df.rename(columns={'Graphics_integreted': 'Graphics_integrated'})


filtered_df = df[df['Processor_variant'].isna()]
result_df = filtered_df[['id', 'Name', 'Processor_brand', 'Processor_name']]
result_df


for index, row in df.iterrows():
    if row['Processor_brand'] == 'Apple' and pd.isna(row['Processor_variant']):
        parts = row['Processor_name'].split('Apple')
        if len(parts) > 2:
            new_variant = parts[2].replace('Chip', '').strip()
        else:
            new_variant = parts[1].replace('Chip', '').strip()
        df.at[index, 'Processor_variant'] = new_variant

```

```

for index, row in df.iterrows():
    if 'MediaTek' in row['Processor_brand'] and pd.isna(row['Processor_variant']):
        name_parts = row['Name'].split()
        for i in range(len(name_parts)):
            if 'MediaTek' in name_parts[i]:
                if i + 1 < len(name_parts):
                    next_part = name_parts[i+1].strip('/')
                    if not any(char.isdigit() for char in next_part):
                        if i + 2 < len(name_parts):
                            df.at[index, 'Processor_variant'] = name_parts[i + 2].strip('/')
                        else:
                            df.at[index, 'Processor_variant'] = next_part
                            break

filtered_df = df[(df['Processor_brand'] == 'Apple') | (df['Processor_brand'] == 'MediaTek')]
result_df = filtered_df[['id', 'Name', 'Processor_brand', 'Processor_name', 'Processor_variant']]
result_df

mode_per_brand = df.groupby('Processor_brand')['Processor_gen'].agg(lambda x:
pd.Series.mode(x).iloc[0] if not x.mode().empty else None)

for brand, mode in mode_per_brand.items():
    if mode is not None:
        df.loc[(df['Processor_brand'] == brand) & (df['Processor_gen'].isna()), 'Processor_gen'] =
mode

for index, row in df[df['Processor_brand'] == 'Apple'].iterrows():
    match = re.search(r'M(\d+)', row['Processor_name'])
    if match:
        df.at[index, 'Processor_gen'] = int(match.group(1))

```

```

df.loc[df['Processor_gen'].isna(), 'Processor_gen'] = 1

mode_per_var = df.groupby('Processor_variant')['Core_per_processor'].agg(lambda x:
pd.Series.mode(x).iloc[0] if not x.mode().empty else None)

for brand, mode in mode_per_var.items():

    if mode is not None:

        df.loc[(df['Processor_variant'] == brand) & (df['Core_per_processor'].isna()),
'Core_per_processor'] = mode

overall_mode = df['Core_per_processor'].mode().iloc[0] if not
df['Core_per_processor'].mode().empty else None

if overall_mode is not None:

    df['Core_per_processor'].fillna(overall_mode, inplace=True)

df = df.drop(columns=['Total_processor', 'Execution_units'])

mode_per_brand = df.groupby('Core_per_processor')['Threads'].agg(lambda x:
pd.Series.mode(x).iloc[0] if not x.mode().empty else None)

for brand, mode in mode_per_brand.items():

    if mode is not None:

        df.loc[(df['Core_per_processor'] == brand) & (df['Threads'].isna()), 'Threads'] = mode

for index, row in df.iterrows():

    if row['Processor_brand'] == 'Apple' and pd.isna(row['Threads']):

        df.at[index, 'Threads'] = df.at[index, 'Core_per_processor']

    elif pd.isna(row['Threads']):

        df.at[index, 'Threads'] = df.at[index, 'Core_per_processor'] + df.at[index,
'Core_per_processor']

for index, row in df.iterrows():

    if row['Processor_brand'] == 'Apple' and pd.isna(row['RAM_type']):

```

```

df.at[index, 'RAM_type'] = 'Unified'

mode_per_GB = df.groupby('RAM_GB')['RAM_type'].agg(lambda x: pd.Series.mode(x).iloc[0]
if not x.mode().empty else None)

for ram, mode in mode_per_GB.items():

    if mode is not None:

        df.loc[(df['RAM_GB'] == ram) & (df['RAM_type'].isna()), 'RAM_type'] = mode

overall_mode_name = df['Graphics_name'].mode().iloc[0] if not
df['Graphics_name'].mode().empty else None

if overall_mode_name is not None:

    df['Graphics_name'].fillna(overall_mode_name, inplace=True)

    filtered_df = df[df['Graphics_name'] == overall_mode_name]

    overall_mode_brand = filtered_df['Graphics_brand'].mode().iloc[0] if not
    filtered_df['Graphics_brand'].mode().empty else None

    if overall_mode_brand is not None:

        df.loc[df['Graphics_name'] == overall_mode_name, 'Graphics_brand'] =
df.loc[df['Graphics_name'] == overall_mode_name,
'Graphics_brand'].fillna(overall_mode_brand)

    overall_mode_integrated = filtered_df['Graphics_integrated'].mode().iloc[0] if not
    filtered_df['Graphics_integrated'].mode().empty else None

    if overall_mode_integrated is not None:

        df.loc[df['Graphics_name'] == overall_mode_name, 'Graphics_integrated'] =
df.loc[df['Graphics_name'] == overall_mode_name,
'Graphics_integrated'].fillna(overall_mode_integrated)

for index, row in df.iterrows():

    if pd.isna(row['Graphics_GB']):

        df.at[index, 'Graphics_GB'] = 0

df.info()

```



```

df['Threads'] = df['Threads'].astype('int64')
df['Graphics_GB'] = df['Graphics_GB'].astype('int64')
df['Processor_gen'] = df['Processor_gen'].astype('int64')
df['Core_per_processor'] = df['Core_per_processor'].astype('int64')
df['Graphics_integrated'] = df['Graphics_integrated'].astype('bool')

df.describe()

df = df.drop(columns=['Low_Power_Cores', 'Energy_Efficient_Units'])
filtered_df = df[df['Horizontal_pixel'] == 1080]
filtered_df[['Vertical_pixel', 'Horizontal_pixel']].head(10)

for index, row in df.iterrows():
    if row['Horizontal_pixel'] < row['Vertical_pixel']:
        df.at[index, 'Horizontal_pixel'], df.at[index, 'Vertical_pixel'] = row['Vertical_pixel'],
row['Horizontal_pixel']

for index, row in df.iterrows():
    if row['Horizontal_pixel'] < row['Vertical_pixel']:
        print(df.loc[index])

sorted(df['Processor_brand'].unique())
sorted(df['Processor_name'].unique())

for index, row in df.iterrows():
    processor_name = row['Processor_name'].replace(" ", "").strip()
    if 'Apple M3' in processor_name:
        processor_name = 'Apple M3'
    elif 'Apple M2 Apple M2 Chip' in processor_name:
        processor_name = 'Apple M2'
    df.at[index, 'Processor_variant'] = 'M2'

```

```

elif processor_name.startswith("Intel ") and processor_name.endswith(" "):
    processor_name = processor_name.strip()
elif 'Intel Core I3-1115G4' in processor_name:
    processor_name = 'Intel Core i3'
    df.at[index, 'Processor_variant'] = '1115G4'
elif 'HiSilicon Kirin 9006C 9006C' in processor_name:
    processor_name = 'HiSilicon Kirin'
    df.at[index, 'Processor_variant'] = '9006C'
elif 'Intel Core 3' in processor_name:
    processor_name = processor_name.replace('3', 'i3')
elif 'Intel Core 5' in processor_name:
    processor_name = processor_name.replace('5', 'i5')
elif 'Intel Core 7' in processor_name:
    processor_name = processor_name.replace('7', 'i7')
elif 'MediaTek' in processor_name or 'Mediatek' in processor_name:
    processor_name = 'MediaTek Kompanio'
elif 'Microsoft SQ1 SQ1' in processor_name:
    processor_name = 'Microsoft'
    df.at[index, 'Processor_variant'] = 'SQ1'
elif 'Qualcomm X Elite' in processor_name:
    processor_name = 'Qualcomm X'
    df.at[index, 'Processor_variant'] = 'Elite'

df.at[index, 'Processor_name'] = processor_name

filtered_df = df[df['Processor_name'] == 'eration Intel Core']
filtered_df[['Name', 'Processor_name', 'Processor_variant']]

```

```

for word in filtered_df['Name']:

    print(word)


index = df[df['Processor_name'] == 'eration Intel Core'].index

if not index.empty:

    df.at[index[0], 'Processor_name'] = 'Intel Core i5'

    df.at[index[0], 'Processor_variant'] = '1235U'

    df.at[index[0], 'Processor_gen'] = 12

    df.at[index[0], 'Core_per_processor'] = 10

    df.at[index[0], 'Threads'] = 12


sorted(df['Processor_name'].unique())

sorted(df['Processor_variant'].unique())


df['Processor_variant'] = df['Processor_variant'].replace('6900HS\u200b', '6900HS', regex =
True)

mode_value = df.loc[df['Processor_name'].str.contains('Intel Core i5', na = False),
'Processor_variant'].mode()[0]

if pd.notna(mode_value):

    df['Processor_variant'] = df['Processor_variant'].replace('i5', mode_value, regex = True)


    mode_value = df.loc[df['Processor_name'].str.contains('Intel Core i7', na = False),
'Processor_variant'].mode()[0]

if pd.notna(mode_value):

    df['Processor_variant'] = df['Processor_variant'].replace('i7', mode_value, regex = True)


    mode_value = df.loc[df['Processor_name'].str.contains('Intel Core i9', na = False),
'Processor_variant'].mode()[0]

if pd.notna(mode_value):

```

```

df['Processor_variant'] = df['Processor_variant'].replace('i9', mode_value, regex = True)

df['Processor_variant'] = df['Processor_variant'].str.upper()


sorted(df['Processor_variant'].unique())

sorted(df['RAM_type'].unique())

df['RAM_type'] = df['RAM_type'].replace('PDDR5X', 'LPDDR5X', regex = True)


sorted(df['Storage_type'].unique())

df['Storage_type'] = df['Storage_type'].str.strip()

df['Storage_type'] = df['Storage_type'].replace('Hard Disk', 'HDD', regex = True)


sorted(df['Graphics_brand'].unique())

df['Graphics_brand'] = df['Graphics_brand'].replace('Radeon', 'AMD', regex = True)


sorted(df['Graphics_name'].unique())

for index, row in df.iterrows():
    name = row['Graphics_name']
    name = name.replace("-", " ")
    name = name.replace("@", " ")
    name = name.replace("\u200e", "")
    name = name.replace("INTEL", "")
    name = name.replace(" Graphics", "")
    name = name.replace("Integrated", " ")
    name = name.replace("Raedon", "Radeon")
    name = name.replace("Nvidia", "NVIDIA")
    name = name.replace("3050 GPU", "3050")
    name = name.replace("Geforce", "GeForce")
    name = name.replace("GEFORCE", "GeForce")
    name = name.replace("GTX 3050", "RTX 3050")

```

```

name = name.replace("GTX 2050", "RTX 2050")

name = name.replace("UHD 620", "Intel UHD 620")

name = name.replace("Intel HD", "Intel UHD 620")

name = re.sub(r"\s+", " ", name)

name = re.sub(r"RX\s+", "RX", name)

name = re.sub(r"RTX(?! \w)", r"RTX ", name)

name = re.sub(r"GTX(?! \w)", r"GTX ", name)

name = re.sub(r"(?!s)(Ti)", r" \1", name)

name = re.sub(r"(?!NVIDIA\s)(GeForce)", r"NVIDIA \1", name)

name = re.sub(r"(?!AMD\s)(Radeon)", r"AMD \1", name)

name = re.sub(r'\b(AMD Radeon)\b(?:\s+\1\b|)+', r'\1', name)

name = re.sub(r'\b(NVIDIA)\b.*\b\1\b', r'\1', name)

name = re.sub(r'\b(Radeon)\b.*\b\1\b', r'\1', name)

name = re.sub(r'\b(Intel)\b.*\b\1\b', r'\1', name)

name = re.sub(r'\b(AMD)\b.*\b\1\b', r'\1', name)

name = re.sub(r'\b(RTX)\b.*\b\1\b', r'\1', name)

if 'GeForce' not in name:

    name = re.sub(r"(RTX|GTX)", r"GeForce \1", name)

if "Core GPU" in name and not name.startswith("Apple"):

    name = "Apple " + name

elif "Intel ARC" in name:

    name = "Intel Arc"

elif "Iris" in name or "iris" in name:

    name = "Intel Iris Xe"

elif name == "AMD Radeon 7":

    name = "AMD Radeon Vega 7"

df.at[index, 'Graphics_name'] = name.strip()

```

```

for index, row in df.iterrows():

    name = row['Graphics_name']

    if name == 'AMD Vega':

        vega_mode = df[df['Graphics_name'].str.contains("Vega")]['Graphics_name'].mode()[0]

        name = vega_mode

    elif name == 'AMD':

        filtered = df[df['Graphics_name'].str.contains("AMD") &
~df['Graphics_name'].isin(["AMD"])]['Graphics_name']

        amd_mode = filtered.mode()[0]

        name = amd_mode

    elif name == 'Intel':

        filtered = df[df['Graphics_name'].str.contains("Intel") &
~df['Graphics_name'].isin(["Intel"])]['Graphics_name']

        int_mode = filtered.mode()[0]

        name = int_mode

    elif name == 'NVIDIA':

        filtered = df[df['Graphics_name'].str.contains("NVIDIA") &
~df['Graphics_name'].isin(["NVIDIA"])]['Graphics_name']

        nv_mode = filtered.mode()[0]

        name = nv_mode

    elif name == 'Intel UHD':

        filtered = df[df['Graphics_name'].str.contains("Intel") & ~df['Graphics_name'].isin(["Intel
UHD"])]['Graphics_name']

        int_mode = filtered.mode()[0]

        name = int_mode

df.at[index, 'Graphics_name'] = name.strip()

sorted(df['Graphics_name'].unique())

sorted(df['Operating_system'].unique())

```

```
for index, row in df.iterrows():

    name = row['Operating_system']

    if "Mac" in name:

        name = "Mac OS"

    if "jio" in name:

        name = "Jio OS"

    if "Ubuntu" in name:

        name = "Linux OS"

    if "DOS" in name:

        name = "DOS OS"

    df.at[index, 'Operating_system'] = name.strip()


sorted(df['Operating_system'].unique())


df.to_csv('Laptop_prices_clean.csv', index = False)

files.download("Laptop_prices_clean.csv")
```

**ДОДАТОК Б - ПРОГРАМНИЙ КОД АНАЛІЗУ ДАНИХ**

*Тексти програмного коду інтелектуального аналізу даних.*

*Методи аналізу – Linear Regression, Ridge Regression,  
Random Forest*

---

(Найменування програми (документа))

*SSD*

---

(Вид носія даних)

*14 арк, 2383 Кб*

---

(Обсяг програми (документа), арк, Кб)

*студента групи ІІІ-22 ІІ курсу*

*Нижника Д.С.*



```
import pandas as pd

import numpy as np


from google.colab import drive

drive.mount('/content/drive')

filename = "/content/drive/My Drive/Laptop_prices_clean.csv"

df = pd.read_csv(filename)

df.info()


import matplotlib.pyplot as plt

import seaborn as sns


mean_price_by_brand = df.groupby('Brand')['Price'].mean().reset_index()

mean_price_by_brand = mean_price_by_brand.sort_values(by = 'Price', ascending = False)


plt.figure(figsize = (16, 8))

sns.barplot(x = 'Price', y = 'Brand', data = mean_price_by_brand, palette = 'Spectral')

plt.title('Average Laptop Price by Brand')

plt.xlabel('Average Price')

plt.ylabel('Brand')

plt.show()


plt.figure(figsize = (18, 8))

sns.histplot(df['Price'], bins = 32, kde = True, color = 'red')

plt.title('Distribution of Laptop Prices')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.show()
```

```

processor_brand_counts = df['Processor_brand'].value_counts()

graphics_brand_counts = df['Graphics_brand'].value_counts()

processor_labels = [f'{brand}: {count / sum(processor_brand_counts) * 100:.1f}% ' for brand,
count in zip(processor_brand_counts.index, processor_brand_counts)]

graphics_labels = [f'{brand}: {count / sum(graphics_brand_counts) * 100:.1f}% ' for brand, count
in zip(graphics_brand_counts.index, graphics_brand_counts)]

fig, axes = plt.subplots(1, 2, figsize = (14, 8))

colors_1 = sns.color_palette('dark:#5A9_r', 7)

wedges_processor = axes[0].pie(processor_brand_counts, colors = colors_1, startangle = 180)

axes[0].legend(wedges_processor[0], processor_labels, title = 'Processor Brands', loc = 'center
left', bbox_to_anchor = (1, 0, 0.5, 1))

axes[0].set_title('Distribution of Processor Brands')

colors_2 = sns.color_palette('dark:#5A9_r', 6)

wedges_graphics = axes[1].pie(graphics_brand_counts, colors = colors_2, startangle = 180)

axes[1].legend(wedges_graphics[0], graphics_labels, title = 'Graphics Brands', loc = 'center left',
bbox_to_anchor = (1, 0, 0.5, 1))

axes[1].set_title('Distribution of Graphics Brands')

plt.tight_layout()

plt.show()

plt.figure(figsize = (18, 8))

sns.histplot(df['Core_per_processor'], bins = 24, kde = True, color = 'red')

plt.title('Distribution of amount of CPU cores')

plt.xlabel('CPU cores')

plt.ylabel('Frequency')

plt.show()

```

```
plt.figure(figsize = (18, 8))

sns.histplot(df['Graphics_GB'], bins = 16, kde = True, color = 'red')

plt.title('Distribution of amount of videomemory')

plt.xlabel('Videomemory GB')

plt.ylabel('Frequency')

plt.show()
```

```
plt.figure(figsize = (18, 8))

sns.histplot(df['Storage_capacity_GB'], bins = 10, kde = True, color = 'red')

plt.title('Distribution of amount of memory')

plt.xlabel('Storage capacity GB')

plt.ylabel('Frequency')

plt.show()
```

```
ram_type_counts = df['RAM_type'].value_counts()

storage_type_counts = df['Storage_type'].value_counts()
```

```
ram_labels = [f'{ram}: {count / sum(ram_type_counts) * 100:.1f}% ' for ram, count in
zip(ram_type_counts.index, ram_type_counts)]
```

```
storage_labels = [f'{storage}: {count / sum(storage_type_counts) * 100:.1f}% ' for storage, count
in zip(storage_type_counts.index, storage_type_counts)]
```

```
fig, axes = plt.subplots(1, 2, figsize = (14, 8))
```

```
colors_1 = sns.color_palette('dark:#5A9_r', 13)
```

```
wedges_ram = axes[0].pie(ram_type_counts, colors = colors_1, startangle = 180)
```

```
axes[0].legend(wedges_ram[0], ram_labels, title = 'RAM Types', loc = 'center left',
bbox_to_anchor = (1, 0, 0.5, 1))
```

```
axes[0].set_title('Distribution of RAM Types')
```

```

colors_2 = sns.color_palette('dark:#5A9_r', 4)

wedges_storage = axes[1].pie(storage_type_counts, colors = colors_2, startangle = 180)

axes[1].legend(wedges_storage[0], storage_labels, title = 'Storage Types', loc = 'center left',
bbox_to_anchor = (1, 0, 0.5, 1))

axes[1].set_title('Distribution of Storage Types')

```

```
plt.tight_layout()
```

```
plt.show()
```

```
plt.figure(figsize = (18, 8))
```

```
sns.histplot(df['RAM_GB'], bins = 20, kde = True, color = 'red')
```

```
plt.title('Distribution of amount of RAM')
```

```
plt.xlabel('RAM GB')
```

```
plt.ylabel('Frequency')
```

```
plt.show()
```

```
os_type_counts = df['Operating_system'].value_counts()
```

```
touch_screen_bool_counts = df['Touch_screen'].value_counts()
```

```
os_type_labels = [f'{os}: {count / sum(os_type_counts) * 100:.1f}%' for os, count in
zip(os_type_counts.index, os_type_counts)]
```

```
touch_screen_bool_labels = [f'{storage}: {count / sum(touch_screen_bool_counts) * 100:.1f}%'
for storage, count in zip(touch_screen_bool_counts.index, touch_screen_bool_counts)]
```

```
fig, axes = plt.subplots(1, 2, figsize = (14, 8))
```

```
colors_1 = sns.color_palette('dark:#5A9_r', 13)
```

```
wedges_os = axes[0].pie(os_type_counts, colors = colors_1, startangle = 180)
```

```
axes[0].legend(wedges_os[0], os_type_labels, title = 'OS', loc = 'center left', bbox_to_anchor =
(1, 0, 0.5, 1))
```

```
axes[0].set_title('Distribution of operational systems')
```

```
colors_2 = sns.color_palette('dark:#5A9_r', 4)
```

```
wedges_storage = axes[1].pie(touch_screen_bool_counts, colors = colors_2, startangle = 180)
```

```
axes[1].legend(wedges_storage[0], touch_screen_bool_labels, title = 'Touch screen', loc = 'center  
left', bbox_to_anchor = (1, 0, 0.5, 1))
```

```
axes[1].set_title('Touch screen presence')
```

```
plt.tight_layout()
```

```
plt.show()
```

```
plt.figure(figsize = (18, 8))
```

```
sns.histplot(df['Display_size_inches'], bins = 20, kde = True, color = 'red')
```

```
plt.title('Distribution of display sizes')
```

```
plt.xlabel('Size')
```

```
plt.ylabel('Frequency')
```

```
plt.show()
```

```
fig, ax = plt.subplots(figsize = (4, 8))
```

```
sns.boxplot(df['Rating'])
```

```
ax.set_title('Boxplot of Laptop Ratings')
```

```
ax.set_xlabel('Rating')
```

```
ax.grid(True)
```

```
plt.show()
```

```
plt.figure(figsize = (15, 10))
```

```
sns.heatmap(df[df.select_dtypes(include=[np.number]).columns].corr(), annot = True, cmap =  
'coolwarm')
```

```
plt.show()
```

```
from scipy.stats import shapiro
```

```
predictors = ['Core_per_processor', 'Threads', 'RAM_GB', 'Storage_capacity_GB',
'Graphics_GB', 'Horizontal_pixel', 'Vertical_pixel', 'ppi']
```

```
for predictor in predictors:
```

```
    print(f"\nAnalyzing predictor: {predictor}")
```

```
    stat, p_value = shapiro(df[predictor])
```

```
    print(f"Shapiro test: statistic = {stat}, p-value = {p_value}")
```

```
spearman_corr_matrix = df[df.select_dtypes(include=[np.number]).columns].corr(method =
'spearman')
```

```
plt.figure(figsize = (15, 10))
```

```
sns.heatmap(spearman_corr_matrix, annot = True, cmap = 'coolwarm')
```

```
plt.show()
```

```
plt.figure(figsize = (18, 20))
```

```
for i, predictor in enumerate(predictors, 1):
```

```
    plt.subplot(4, 2, i)
```

```
    sns.regplot(x = 'Price', y = predictor, data = df)
```

```
    plt.title(f'Price vs {predictor}')
```

```
    plt.ylim(0,)
```

```
    plt.xlim(0,)
```

```
plt.tight_layout()
```

```
plt.show()
```

```
from scipy.stats import kruskal
```

```
non_numeric_columns = df.select_dtypes(exclude = [np.number]).columns
```

```
kruskal_results = []
```

```
for column in non_numeric_columns:
```

```
    groups = [group['Price'].values for name, group in df.groupby(column)]
```

```
    if len(groups) > 1:
```

```
        h, p = kruskal(*groups)
```

```
        kruskal_results.append((column, h, p))
```

```
kruskal_df = pd.DataFrame(kruskal_results, columns = ['Feature', 'H_statistic', 'p_value'])
```

```
kruskal_df = kruskal_df.set_index('Feature')
```

```
print(kruskal_df.sort_values(by='H_statistic', ascending=False))
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
```

```
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(df[df.select_dtypes(include=[np.number]).columns])
```

```
pca = PCA()
```

```
pca.fit(scaled_data)
```

```
plt.figure(figsize = (12, 6))
```

```
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_, marker  
= 'o')
```

```
plt.title('Scree Plot')
```

```
plt.xlabel('Number of components')
plt.show()
```

```
!pip install factor_analyzer
```

```
from factor_analyzer import FactorAnalyzer
```

```
fa = FactorAnalyzer(n_factors = 3, rotation = 'varimax')
```

```
fa.fit(scaled_data)
```

```
loadings = fa.loadings_
```

```
loadings_df = pd.DataFrame(loadings, index = df.select_dtypes(include = [np.number]).columns,
columns=[f'Factor{i + 1}' for i in range(loadings.shape[1])])
```

```
print(loadings_df)
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
```

```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler, OneHotEncoder
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.linear_model import LinearRegression, Ridge
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
selected_columns = ['Threads', 'Vertical_pixel', 'RAM_GB', 'Graphics_GB', 'RAM_type', 'Brand',
'Processor_name', 'Graphics_brand']
```

```
x = df[selected_columns]
```

```
y = df['Price']
```

```
categorical_cols = x.select_dtypes(include = ['object']).columns
```

```
onehot_encoder = OneHotEncoder(sparse = False, drop = 'first')
```



```

x_encoded = pd.DataFrame(onehot_encoder.fit_transform(x[categorical_cols]), columns =
onehot_encoder.get_feature_names_out(categorical_cols))

x_numeric = x.drop(columns = categorical_cols)

x_transformed = pd.concat([x_numeric.reset_index(drop = True), x_encoded.reset_index(drop =
True)], axis = 1)

x_train, x_test, y_train, y_test = train_test_split(x_transformed, y, test_size = 0.4, random_state =
1)

model_lr = LinearRegression()

model_lr.fit(x_train, y_train)

y_pred = model_lr.predict(x_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

mape = mean_absolute_percentage_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f'MAPE = {mape}')

print(f'RMSE = {rmse}')

print(f'R^2 = {r2}')

Y_hat_0 = model_lr.predict(x_test)

plt.figure(figsize = (15, 10))

ax1 = sns.distplot(df['Price'], hist = False, color = "r", label = "Actual Values")

sns.distplot(Y_hat_0, hist = False, color = "b", label = "Fitted Values", ax = ax1)

plt.title("Actual vs Fitted Values")

plt.xlabel("Price")

```

```
plt.show()
```

```
plt.close()
```

```
ridge = Ridge()
```

```
ridge.fit(x_train, y_train)
```

```
ridge_y_pred = ridge.predict(x_test)
```

```
rmse = np.sqrt(mean_squared_error(y_test, ridge_y_pred))
```

```
mape = mean_absolute_percentage_error(y_test, ridge_y_pred)
```

```
r2 = r2_score(y_test, ridge_y_pred)
```

```
print(f'MAPE = {mape}')
```

```
print(f'RMSE = {rmse}')
```

```
print(f'R^2 = {r2}', '\n')
```

```
Y_hat_1 = ridge.predict(x_transformed)
```

```
plt.figure(figsize = (15, 10))
```

```
ax1 = sns.distplot(df['Price'], hist = False, color = "r", label = "Actual Values")
```

```
sns.distplot(Y_hat_1, hist = False, color = "b", label = "Fitted Values", ax = ax1)
```

```
plt.title("Actual vs Fitted Values")
```

```
plt.xlabel("Price")
```

```
plt.show()
```

```
plt.close()
```

```
poly = PolynomialFeatures(degree = 2, include_bias = False)
```

```

x_poly = poly.fit_transform(x_transformed)

x_poly_scaled = StandardScaler().fit_transform(x_poly)

x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(x_poly_scaled, y, test_size = 0.4,
random_state = 1)

param_grid = {'alpha': np.logspace(-2, 10, 13)}

ridge = Ridge()

grid_search = GridSearchCV(estimator = ridge, param_grid = param_grid, cv = 5, scoring =
'neg_mean_squared_error', n_jobs = -1, verbose = 2)

grid_search.fit(x_train_p, y_train_p)

best_ridge = grid_search.best_estimator_

ridge_y_pred = best_ridge.predict(x_test_p)

rmse = np.sqrt(mean_squared_error(y_test_p, ridge_y_pred))

mape = mean_absolute_percentage_error(y_test_p, ridge_y_pred)

r2 = r2_score(y_test_p, ridge_y_pred)

print(f'MAPE = {mape}')

print(f'RMSE = {rmse}')

print(f'R^2 = {r2}')

Y_hat_2 = best_ridge.predict(x_test_p)

plt.figure(figsize = (15, 10))

ax1 = sns.distplot(df['Price'], hist = False, color = "r", label = "Actual Values")

sns.distplot(Y_hat_2, hist = False, color = "b", label = "Fitted Values", ax = ax1)

```

```
plt.title("Actual vs Fitted Values")

plt.xlabel("Price")


plt.show()

plt.close()


model_rf = RandomForestRegressor(n_estimators = 100, random_state = 1)

model_rf.fit(x_train, y_train)


y_pred = model_rf.predict(x_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

mape = mean_absolute_percentage_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f'MAPE = {mape}')

print(f'RMSE = {rmse}')

print(f'R^2 = {r2}')


Y_hat_3 = model_rf.predict(x_test)


plt.figure(figsize = (15, 10))

ax1 = sns.distplot(df['Price'], hist = False, color = "r", label = "Actual Values")

sns.distplot(Y_hat_3, hist = False, color = "b", label = "Fitted Values", ax = ax1)


plt.title("Actual vs Fitted Values")

plt.xlabel("Price")


plt.show()

plt.close()
```

```

param_grid = {
    'n_estimators': [50, 100, 150, 200, 500, 700, 1000],
    'max_depth': [5, 10, 20, 30, None],
    'min_samples_split': [2, 5, 7, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

rf = RandomForestRegressor(random_state=1)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=4, n_jobs=-1,
verbose=2, scoring='neg_mean_squared_error')

grid_search.fit(x_train, y_train)

best_rf = grid_search.best_estimator_

best_rf_y_pred = best_rf.predict(x_test)

best_rf_rmse = np.sqrt(mean_squared_error(y_test, best_rf_y_pred))

best_rf_mape = mean_absolute_percentage_error(y_test, best_rf_y_pred)

best_rf_r2 = r2_score(y_test, best_rf_y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}', '\n')
print(f'Best parameters: {grid_search.best_params_}')

model_rf = RandomForestRegressor(n_estimators = 100, random_state = 1)

model_rf.fit(x_train_p, y_train_p)

y_pred = model_rf.predict(x_test_p)

```

```

rmse = np.sqrt(mean_squared_error(y_test_p, y_pred))
mape = mean_absolute_percentage_error(y_test_p, y_pred)
r2 = r2_score(y_test_p, y_pred)

print(f'MAPE = {mape}')
print(f'RMSE = {rmse}')
print(f'R^2 = {r2}')

Y_hat_4 = model_rf.predict(x_test_p)

plt.figure(figsize = (15, 10))
ax1 = sns.distplot(df['Price'], hist = False, color = "r", label = "Actual Values")
sns.distplot(Y_hat_4, hist = False, color = "b", label = "Fitted Values", ax = ax1)
plt.title("Actual vs Fitted Values")
plt.xlabel("Price")

plt.show()
plt.close()

plt.figure(figsize = (15, 10))
ax1 = sns.kdeplot(df['Price'], color="red", label="Actual Values", lw=2)
sns.kdeplot(Y_hat_0, color="blue", label="Linear Regression", ax=ax1, lw=2)
sns.kdeplot(Y_hat_2, color="green", label="Ridge Regression", ax=ax1, lw=2)
sns.kdeplot(Y_hat_4, color="orange", label="Random Forest", ax=ax1, lw=2)
plt.title("Actual vs Fitted Values")
plt.xlabel("Price")
plt.legend()
plt.show()
plt.close()

```