

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: «Аналіз та прогнозування цін на ноутбуки на основі різних
показників»

Студентки 2 курсу групи ІП-23

Спеціальності: 121

«Інженерія програмного забезпечення»

Піроженко Дар'ї Валеріївни

«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 ІІІ-23

Семестр 4

ЗАВДАННЯ

на курсову роботу студентки

Піроженко Дар'ї Валеріївни

1.Тема роботи **Аналіз та прогнозування цін на ноутбуки на основі різних показників**

2.Строк здачі студенткою закінченої роботи 29.05.2024

3. Вхідні дані до роботи методичні вказівки до курсової роботи, обрані дані з сайту https://www.kaggle.com/datasets/muhammetvarl/laptop-price?resource=download&select=laptop_price.csv

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

1.Постановка задачі

2.Аналіз предметної області

3.Розробка сховища даних

4.Інтелектуальний аналіз даних

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

6.Дата видачі завдання 16.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	28.03.2024	
2.	Визначення зовнішніх джерел даних	10.04.2024	
3.	Пошук та вивчення літератури з питань курсової роботи	25.04.2024	
4.	Обробка та аналіз даних	03.05.2024	
5.	Обґрунтування методів інтелектуального аналізу даних	09.05.2024	
6.	Застосування та порівняння ефективності методів інтелектуального аналізу даних	14.05.2024	
7.	Підготовка пояснювальної записки	28.05.2024	
8.	Здача курсової роботи на перевірку	29.05.2024	
9.	Захист курсової роботи	03.06.2024	

Студентка

(підпис)

Піроженко Дар'я Валеріївна

(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)

"03" червня 2024 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 54 сторінки, 54 рисунки, 10 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, обробка та аналіз даних, реалізація програмного забезпечення для роботи з даними, їх подальшого аналізу та прогнозування.

Дана курсова робота включає в себе: опис створення програмного забезпечення для інтелектуального аналізу даних, їх графічного відображення та прогнозування за допомогою різних моделей.

МОДЕЛЬ ПРОГНОЗУВАННЯ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, MULTIPLE LINEAR REGRESSION, RIDGE REGRESSION, GRADIENT BOOSTING REGRESSOR, RANDOM FOREST REGRESSOR.

ЗМІСТ

ВСТУП.....	5
1. ПОСТАНОВКА ЗАДАЧІ.....	6
2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	7
3. Робота з даними.....	9
3.1 Опис обраних даних.....	9
3.2 Підготовка даних для аналізу.....	9
3.3 Перевірка даних.....	13
3.4 Візуальний аналіз.....	15
3.5 Поділ даних.....	23
4. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ.....	24
4.1 Обґрунтування вибору методів інтелектуального аналізу даних.....	24
4.2 Аналіз отриманих результатів для методу Multiple Linear regression.....	26
4.3 Аналіз отриманих результатів для методу Ridge Regression.....	29
4.4 Аналіз отриманих результатів для методу Gradient Boosting Regressor.....	31
4.5 Аналіз отриманих результатів для методу Random Forest Regressor.....	35
4.6 Порівняння отриманих результатів методів.....	39
ВИСНОВКИ.....	41
ПЕРЕЛІК ПОСИЛАНЬ.....	42
ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ.....	43

ВСТУП

Ноутбуки є невід’ємною частиною нашого повсякденного життя, вони використовуються у різних сферах діяльності: від навчання і роботи до розваг і творчості. Вони забезпечують доступ до інформації, комунікацій та розвитку в будь-який час і в будь-якому місці. Завдяки своїй портативності, ноутбуки стають незамінними в умовах сучасного швидкоплинного ритму життя. Вибір ноутбука – це важливе рішення, яке потребує уважного підходу. Тому важливо знати які фактори найбільше впливають на ціну, щоб знати на що орієнтуватись при виборі такого пристрою.

В рамках даної курсової роботи було проаналізовано дані про характеристики ноутбуків. На основі отриманих даних використано чотири методи для прогнозування цін на ноутбуки.

Для проведення аналізу використовуватимемо дані, зібрані та опубліковані на сайті kaggle.

У даній курсовій роботі було використано наступні технології: Python[1], Pandas[2], Matplotlib[3], Sklearn[4], Seaborn[5], NumPy[6].

1. ПОСТАНОВКА ЗАДАЧІ

Метою цієї курсової роботи є прогнозування цін на ноутбуки на основі аналізу технічних характеристик.

Наш підхід передбачає детальний аналіз та попередню обробку даних з датасету. Для досягнення мети роботи необхідно обрати найоптимальніші методи інтелектуального аналізу даних і встановити критерії оцінки якості моделей. Далі слід провести валідацію і тестування розроблених моделей для перевірки їхньої точності та ефективності. Якщо буде потрібно, внести корекції та вдосконалити моделі на основі отриманих результатів. Після цього, щоб узагальнити результати, необхідно порівняти ефективність різних методів на навчальних та тестових вибірках даних.

Вхідними даними є назва компанії, тип ноутбуку, діагональ, наявна інформація про розширення екрану, центральний та графічний процесор, Ram, загальну пам'ять пристрою, операційн усистему, вагу та ціну ноутбука.

Цінність даного дослідження полягає в тому, що на воно дає розуміння які характеристики найбільше впливають на ціну ноутбука, а також дозволяє прогнозувати ціни на основі необхідних ознак.

Для прогнозування буде використано методи Linear Regression, Ridge Regression, Gradient Boosting Regressor та Random Forest Regressor. Для кожного методу необхідно проаналізувати результати та в кінці порівняти

2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Ноутбуки – це портативні персональні комп'ютери, які широко використовуються в усьому світі. Вони стали невід'ємною частиною нашого життя, пропонуючи зручність, портативність і потужність.

Ноутбуки використовуються для різних цілей, таких як:

- Робота: ноутбуки широко використовуються в діловому середовищі для таких завдань, як обробка текстів, електронна пошта, презентації та робота з веб-браузерами.
- Навчання: ноутбуки є незамінним інструментом для студентів, які використовують їх для конспектування лекцій, досліджень, написання робіт та доступу до навчальних ресурсів.
- Розваги: ноутбуки використовуються для перегляду фільмів, телешоу, прослуховування музики, ігор та інших розваг.
- Ігри: ігрові ноутбуки спеціально розроблені для ігор і мають потужні графічні процесори та інші компоненти, необхідні для забезпечення плавної та чіткої графіки.

В залежності від цілі для якої буде використовуватись ноутбук, він буде мати відповідні характеристики, що будуть найкраще виконувати поставлені задачі.

Наразі багато компаній, представлених на ринку, змагаються в інноваціях, прагнучи створити сучасні ноутбуки. Проте зі зростанням кількості конкурентів у технологічній індустрії, зокрема у виробництві ноутбуків, недостатньо лише інновацій. Компанії також повинні розробляти більш ефективні стратегії для виживання на ринку.

Однією з таких стратегій є визначення факторів, які впливають на споживачів під час покупки ноутбука. За словами дослідників, існує чотири основні мотиватори для споживачів при виборі інноваційних продуктів:

- (1) функціональні потреби – бажання споживачів отримати функціональні переваги від продукту,
- (2) гедонічні потреби – бажання споживачів отримати задоволення або насолоду від продукту,
- (3) соціальні потреби – прагнення споживачів до підвищення статусу та престижу, а також бажання володіти унікальним продуктом, що відрізняється від інших,
- (4) когнітивні потреби – цікавість споживачів до використання продукту. Водночас кожен споживач має свої унікальні міркування при виборі продукту.

Постійний розвиток процесорів, графічних карт, пам'яті та інших компонентів дозволяє створювати більш потужні та енергоефективні ноутбуки. Також важливими є інновації у сфері дисплеїв (роздільна здатність, частота оновлення) та батарей (тривалість роботи). Споживачі ж шукають оптимальне співвідношення ціни та якості. Тому необхідно розуміти, що найбільше впливає на ціноутворення на ринку ноутбуків.

У програмному забезпеченні буде реалізовано наступну функціональність, що включає в себе:

- завантаження вибірки даних;
- обробка та дослідження завантажених даних;
- інтелектуальний аналіз даних;
- використання моделей прогнозування даних;
- прогнозування цін на ноутбуки;
- візуалізація отриманих результатів та їх аналіз.

3. Робота з даними

3.1 Опис обраних даних

Для виконання поставленого завдання було обрано датасет Laptop Price на сайті Kaggle, що складається з 1300 рядків і 13 стовпців, а саме: Company, Product TypeName, Inches, ScreenResolution, Cpu, Ram, Memory, GPU, OpSys, Weight, Price_euros. Цей набір даних є джерелом інформації для вивчення характеристик різноманітних ноутбуків. Стовпці мають наступну інформацію:

- Company – компанія, що виготовила ноутбук
- Product – модель ноутбуку
- TypeName – тип ноутбуку (Notebook, Ultrabook, Gaming, і т. д.)
- Inches — діагональ екрану
- ScreenResolution – інформація про розширення екрану
- Cpu – інформація про центральний процесор ноутбуку
- Ram – кількість RAM
- Memory – інформація про наявну пам'ять, а також її вид (Hard Disk / SSD Memory)
- GPU – інформація про графічний процесор
- OpSys – вказана наявна операційна система або її відсутність
- Weight – вага ноутбуку
- Price_euros – ціна ноутбуку в євро

3.2 Підготовка даних для аналізу

Зчитасмо дані з файлу в датафрейм, виведемо перші 5 рядків датафрейму, щоб перевірити коректність імпорту даних, та основну інформацію (рис. 3.1) про нього, таку можливість нам надає Python бібліотека pandas.

```
from google.colab import drive
drive.mount('/content/drive')
filename = "/content/drive/My Drive/laptop_price.csv"
df = pd.read_csv(filename, encoding='windows-1251')
df.head()
```

Mounted at /content/drive

	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros
0	1	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8GB	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37kg	1339.69
1	2	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8GB	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34kg	898.94
2	3	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8GB	256GB SSD	Intel HD Graphics 620	No OS	1.86kg	575.00
3	4	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16GB	512GB SSD	AMD Radeon Pro 455	macOS	1.83kg	2537.45
4	5	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8GB	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37kg	1803.60

Рисунок 3.1 – Вигляд датафрейму.

Також перевіримо типи даних для кожної колонки (рис. 3.2)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   laptop_ID              1303 non-null  int64
1   Company                1303 non-null  object
2   Product                1303 non-null  object
3   TypeName               1303 non-null  object
4   Inches                 1303 non-null  float64
5   ScreenResolution       1303 non-null  object
6   Cpu                    1303 non-null  object
7   Ram                    1303 non-null  object
8   Memory                 1303 non-null  object
9   Gpu                    1303 non-null  object
10  OpSys                  1303 non-null  object
11  Weight                 1303 non-null  object
12  Price_euros            1303 non-null  float64
dtypes: float64(2), int64(1), object(10)
memory usage: 132.5+ KB
```

Рисунок 3.2 – Загальна інформація про датафрейм.

На даному етапі можна помітити, що дані було зчитано коректно, але є проблема з тим, що в таких колонках як Ram та Weight присутні одиниці вимірювання, тому вони відображаються як тип object. Приберемо їх, для того щоб ці колонки стали числовими.

У колонці ScreenResolution бачимо, що є дуже багато інформації, яку можна кодувати (рис 3.3).

```
df['ScreenResolution'].value_counts()
```

ScreenResolution	
Full HD 1920x1080	507
1366x768	281
IPS Panel Full HD 1920x1080	230
IPS Panel Full HD / Touchscreen 1920x1080	53
Full HD / Touchscreen 1920x1080	47
1600x900	23
Touchscreen 1366x768	16
Quad HD+ / Touchscreen 3200x1800	15
IPS Panel 4K Ultra HD 3840x2160	12
IPS Panel 4K Ultra HD / Touchscreen 3840x2160	11
4K Ultra HD / Touchscreen 3840x2160	10
4K Ultra HD 3840x2160	7
Touchscreen 2560x1440	7
IPS Panel 1366x768	7

Рисунок 3.3 – Приклади значень, наведених в колонці ScreenResolution.

Кодування категоріальних змінних дозволить перетворити їх у числові значення. Створимо колонку Touchscreen та використаємо кодування порядковим числом - цей метод просто присвоює кожній категорії числове значення. Це означає, що категорії будуть перетворені на числа, де кожен ноутбук матиме значення 0 або 1. Нехай 0 буде інтерпретувати як "відсутність сенсорного екрану", а 1 - як "наявність".

Аналогічно створимо колонку Ips та відмітимо 1, якщо у монітора наявна Ips матриця та 0, якщо відсутня. Спостерігаємо відповідні зміни (рис 3.4).

```
df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:1 if 'Touchscreen' in x else 0)
df['Ips'] = df['ScreenResolution'].apply(lambda x:1 if 'IPS' in x else 0)
df.head()
```

	laptop_ID	Company	Product	TypeName	Inches	ScreenResolution	Cpu	Ram	Memory	Gpu	OpSys	Weight	Price_euros	Touchscreen	Ips
0	1	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 2.3GHz	8	128GB SSD	Intel Iris Plus Graphics 640	macOS	1.37	1339.69	0	1
1	2	Apple	Macbook Air	Ultrabook	13.3	1440x900	Intel Core i5 1.8GHz	8	128GB Flash Storage	Intel HD Graphics 6000	macOS	1.34	898.94	0	0
2	3	HP	250 G6	Notebook	15.6	Full HD 1920x1080	Intel Core i5 7200U 2.5GHz	8	256GB SSD	Intel HD Graphics 620	No OS	1.86	575.00	0	0
3	4	Apple	MacBook Pro	Ultrabook	15.4	IPS Panel Retina Display 2880x1800	Intel Core i7 2.7GHz	16	512GB SSD	AMD Radeon Pro 455	macOS	1.83	2537.45	0	1
4	5	Apple	MacBook Pro	Ultrabook	13.3	IPS Panel Retina Display 2560x1600	Intel Core i5 3.1GHz	8	256GB SSD	Intel Iris Plus Graphics 650	macOS	1.37	1803.60	0	1

Next steps: [View recommended plots](#)

Рисунок 3.4 – Бачимо новостворені колонки Touchscreen та Ips.

Розрахуємо кількість пікселів на дюйм, що є більш важливими даними ніж розширення екрану, враховуючи те, що у нас відома діагональ та запишемо ці дані в колонку Ppi (Pixels Per Inch) (рис 3.5).

```

new = df['ScreenResolution'].str.split('x',n=1,expand=True)
df['X_res'] = new[0]
df['Y_res'] = new[1]
df['X_res'] = df['X_res'].str.replace(',', '').str.findall(r'(\d+\.\d+)?').apply(lambda x:x[0])
df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')

df['Ppi'] = (((df['X_res']**2) + (df['Y_res']**2))**0.5/df['Inches']).astype('float')
df.drop(columns=['ScreenResolution', 'X_res', 'Y_res'],inplace=True)

df['Ppi']

```

	Ppi
0	226.983005
1	127.677940
2	141.211998
3	220.534624
4	226.983005
...	...
1298	157.350512
1299	276.053530
1300	111.935204
1301	100.454670
1302	100.454670

Name: Ppi, Length: 1303, dtype: float64

Рисунок 3.5 – Розрахунок для колонки Ppi.

Далі систематизуємо назви для центральних процесорів (рис. 3.6).

```

df['Cpu brand'] = df['Cpu Name'].apply(fetch_processor)
df.drop(columns=['Cpu', 'Cpu Name'],inplace=True)
df['Cpu brand']

```

	Cpu brand
0	Intel Core i5
1	Intel Core i5
2	Intel Core i5
3	Intel Core i7
4	Intel Core i5
...	...
1298	Intel Core i7
1299	Intel Core i7
1300	Other Intel Processor
1301	Intel Core i7
1302	Other Intel Processor

Name: Cpu brand, Length: 1303, dtype: object

Рисунок 3.6 – Новий вигляд колонки Cpu brand.

У колонці Memory представлена інформація про всі наявні типи пам'яті для ноутбука. Декомпозуємо цю колонку на відповідні типи пам'яті (рис. 3.7).

	HDD	SSD	Hybrid	Flash_Storage
0	128	0	0	0
0	0	0	0	128
0	256	0	0	0
0	512	0	0	0
0	256	0	0	0

Рисунок 3.7 – Кількість HDD, SSD, Hybrid, Flash storage для кожного ноутбука.

Залишимо тільки компанію графічного процесора, для кращої систематизації. А також приведемо назви операційних систем до одного вигляду (рис 3.8).

```
def cat_os(inp):
    if inp == 'Windows 10' or inp == 'Windows 7' or inp == 'Windows 10 S':
        return 'Windows'
    elif inp == 'macOS' or inp == 'Mac OS X':
        return 'Mac'
    elif inp == 'Linux':
        return 'Linux'
    elif inp == 'No OS':
        return 'No OS'
    else:
        return 'Others'

df['Os'] = df['OpSys'].apply(cat_os)
df.drop(columns=['OpSys'], inplace=True)
df['Os']
```

0	Mac
1	Mac
2	No OS
3	Mac
4	Mac
...	
1298	Windows
1299	Windows
1300	Windows
1301	Windows
1302	Windows

Name: Os, Length: 1302, dtype: object

Рисунок 3.8 – Новий вигляд колонки Os.

Також видалимо колонки, в яких замало даних або які непотрібні для аналізу (рис. 3.9).

```
df.drop(columns=['Gpu', 'Hybrid', 'Flash_Storage', 'laptop_ID'], inplace=True)
```

Рисунок 3.9 – Видалення непотрібних колонок.

3.3 Перевірка даних

Виведемо загальну інформацію про новоутворений датафрейм (рис. 3.10).

`df.describe()`

	Inches	Ram	Weight	Price_euros	Touchscreen	Ips	Ppi	HDD	SSD
count	1302.000000	1302.000000	1302.000000	1302.000000	1302.000000	1302.000000	1302.000000	1302.000000	1302.000000
mean	15.019278	8.385561	2.039416	1124.043894	0.146697	0.27957	146.568497	414.101382	183.874040
std	1.424861	5.085166	0.665274	699.158856	0.353940	0.44896	43.069016	515.889348	186.969314
min	10.100000	2.000000	0.690000	174.000000	0.000000	0.00000	90.583402	0.000000	0.000000
25%	14.000000	4.000000	1.500000	599.000000	0.000000	0.00000	127.335675	0.000000	0.000000
50%	15.600000	8.000000	2.040000	978.000000	0.000000	0.00000	141.211998	0.000000	256.000000
75%	15.600000	8.000000	2.300000	1488.435000	0.000000	1.00000	157.350512	1000.000000	256.000000
max	18.400000	64.000000	4.700000	6099.000000	1.000000	1.00000	352.465147	2000.000000	1024.000000

`df.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 1302 entries, 0 to 1302
Data columns (total 15 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Company     1302 non-null   object
1   Product     1302 non-null   object
2   TypeName    1302 non-null   object
3   Inches      1302 non-null   float64
4   Ram         1302 non-null   int32
5   Weight      1302 non-null   float32
6   Price_euros 1302 non-null   float64
7   Touchscreen 1302 non-null   int64
8   Ips         1302 non-null   int64
9   Ppi         1302 non-null   float64
10  Cpu brand   1302 non-null   object
11  HDD         1302 non-null   int64
12  SSD         1302 non-null   int64
13  Gpu brand   1302 non-null   object
14  Os          1302 non-null   object
dtypes: float32(1), float64(3), int32(1), int64(4), object(6)
memory usage: 152.6+ KB
```

Рисунок 3.10 – Загальна інформація про новоутворений датасет.

Перевіряємо на пропущені значення (рис 3.11).

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

```
Company      0
Product      0
TypeName     0
Inches       0
Ram          0
Weight       0
Price_euros  0
Touchscreen  0
Ips          0
Ppi          0
Cpu brand    0
HDD          0
SSD          0
Gpu brand    0
Os           0
dtype: int64
```

Рисунок 3.11 – Пропущених значень не виявлено.

Перевіряємо наявність дублікатів та видаляємо їх (рис 3.12).

```
[82] df.duplicated().sum()
```

```
28
```

```
df.drop_duplicates(inplace=True)
```

Рисунок 3.12 – Видалення дублікатів.

3.4 Візуальний аналіз

Виходячи з висновків вище можна стверджувати, що дані чисті та готові до подальшого аналізу. Тепер можемо подивитись розподіл кількості ноутбуків для кожного виду (рис. 3.13). Для відображення графіків використовуємо бібліотеки `matplotlib` та `seaborn`.

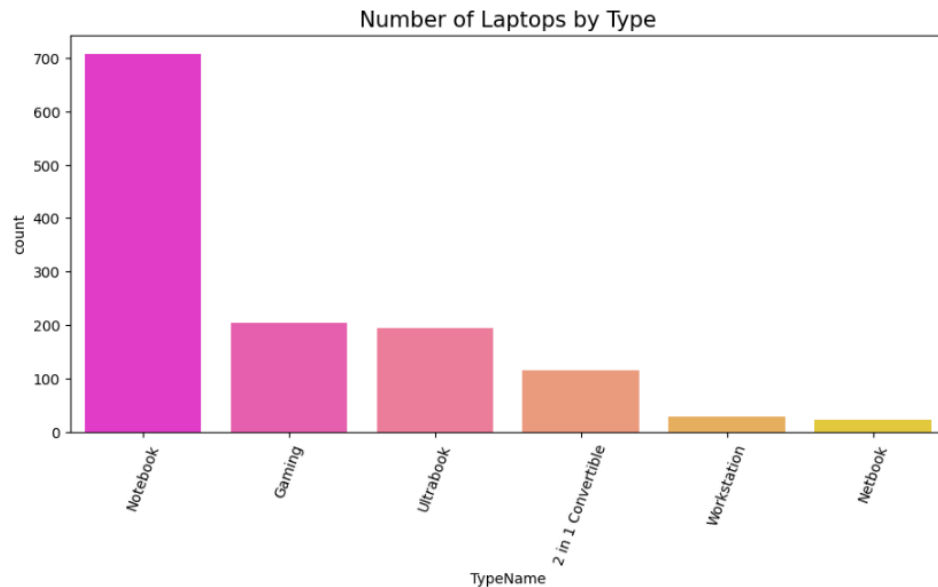


Рисунок 3.13 – Кількість ноутбуків для кожного виду.

Також за допомогою діаграми розмаху подивимось на розподіл ціни на ноутбуки за цими ж видами (рис 3.15).

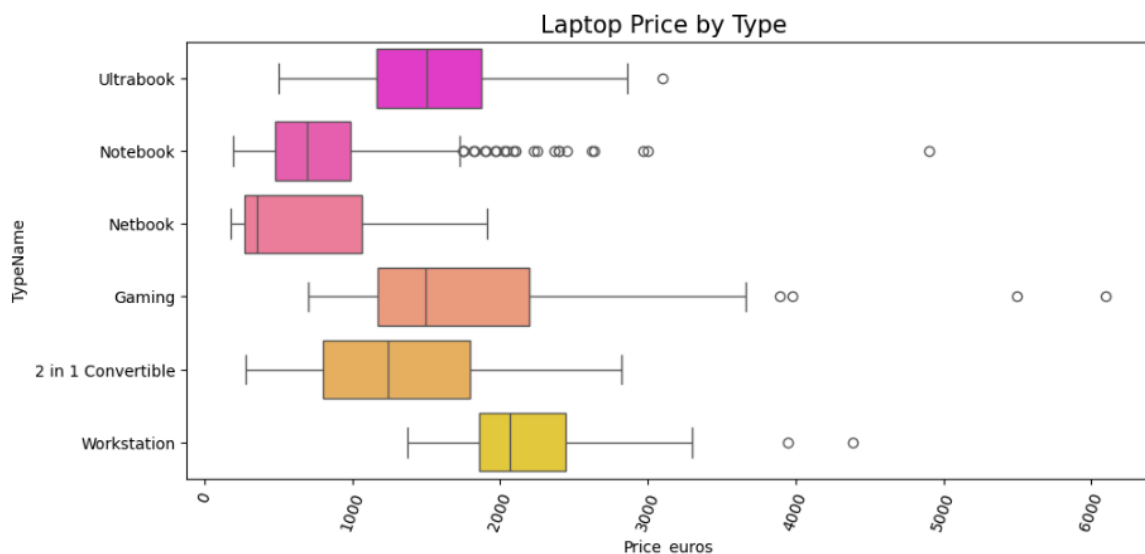


Рисунок 3.15 – Розмах цін ноутбуків для кожного виду.

На основі даних діаграм можна зробити висновок, що ціновий діапазон коливається від 200 до 6000 євро, але більшість ноутбуків коштують менше 4000 євро. Не викликає здивування, що найпоширенішим типом є стандартний ноутбук, а найменш популярним — нетбук.

Далі розглянемо як в даному датасеті представлені компанії, що виготовляють ноутбуки (рис. 3.16).

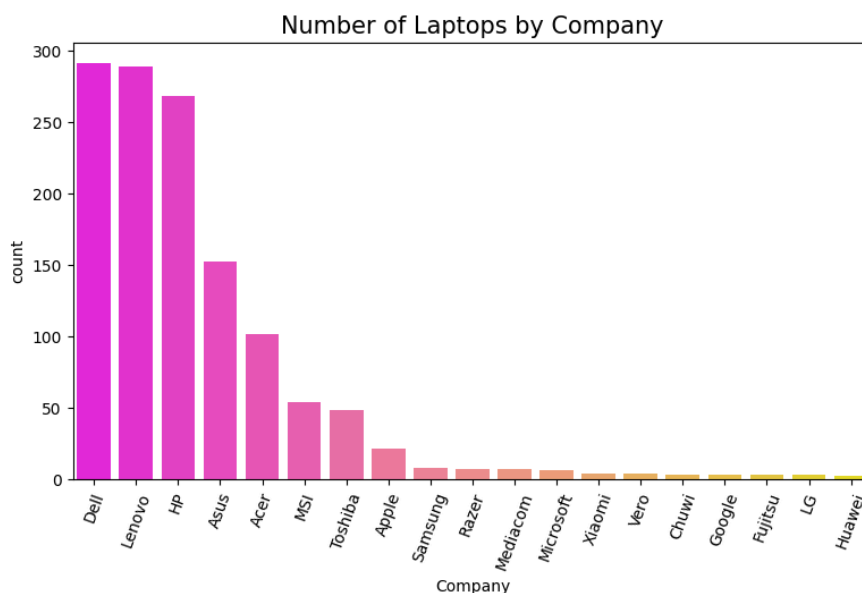


Рисунок 3.16 – Кількість ноутбуків для кожної компанії.

Також розглянемо діапазон цін різних компаній (рис. 3.17).

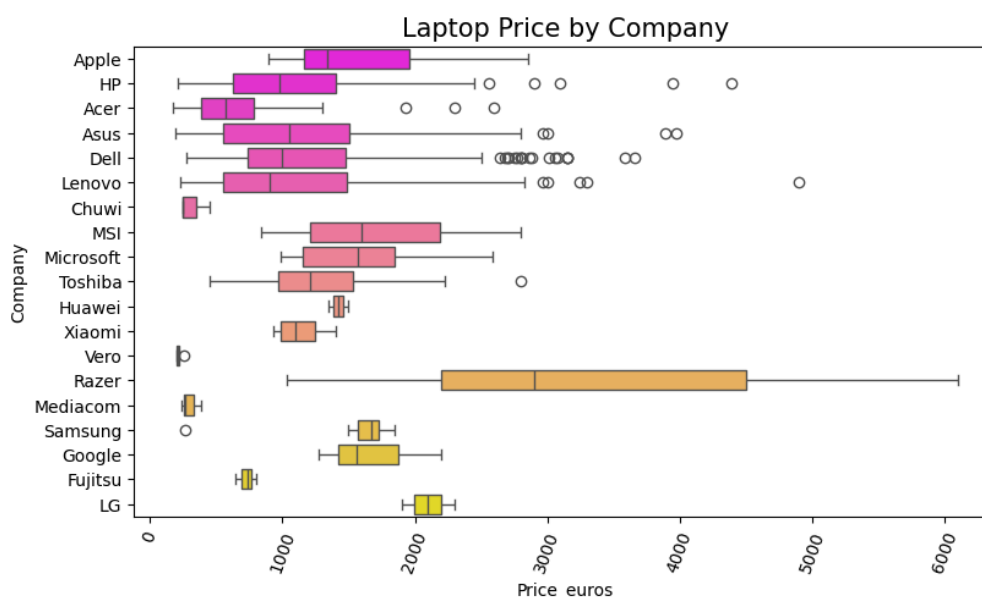


Рисунок 3.17 – Розмах цін ноутбуків для кожної компанії.

Як бачимо ноутбуки Dell, Lenovo, HP, Asus і Acer є найпоширенішими у нашому датасеті. Ноутбуків Samsung, Razer, Mediacom, Microsoft, Xiaomi, Vero, Chuwi, Google, Fujitsu, LG і Huawei у датасеті менше 10. Razer є найдорожчим ноутбуком (що закономірно, адже ця компанія спеціалізується на ігрових ноутбуках, що є найдорожчими).

Наступним кроком розглянемо графічні процесори, які представлені брендами Intel, Nvidia та AMD. Аналогічно розглянемо кількість ноутбуків, що використовують відповідні графічні процесори, а також розмах цін на них (рис. 3.18).

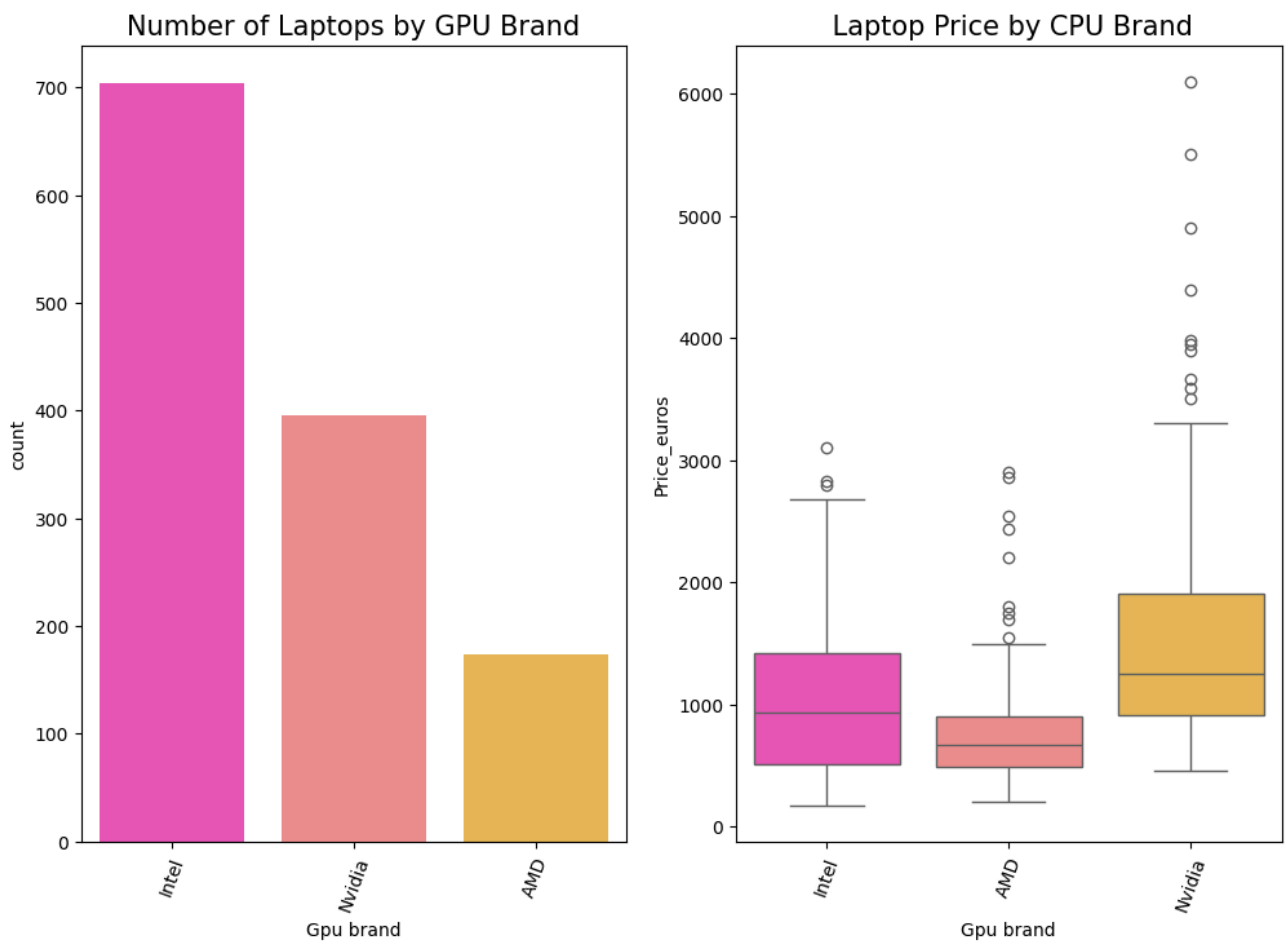


Рисунок 3.18 – Кількість ноутбуків для кожної компанії графічних процесорів та розмах цін ноутбуків для кожної з них.

Наступним кроком розглянемо виробників центральних процесорів (рис.3.19), а також розглянемо як представлені операційні системи в нашому датасеті.

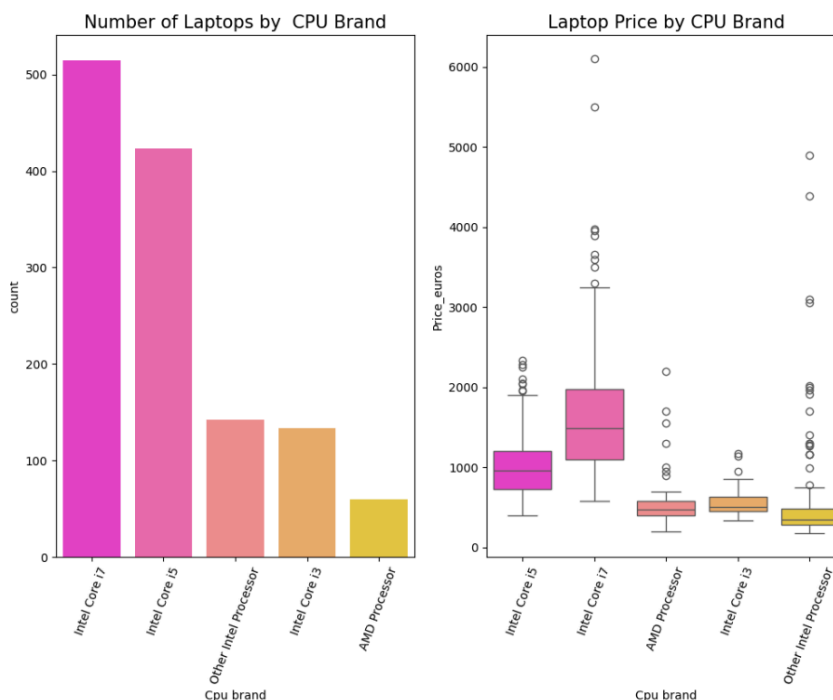


Рисунок 3.19 – Кількість ноутбуків для кожної компанії центральних процесорів та розмах цін ноутбуків для кожної з них.

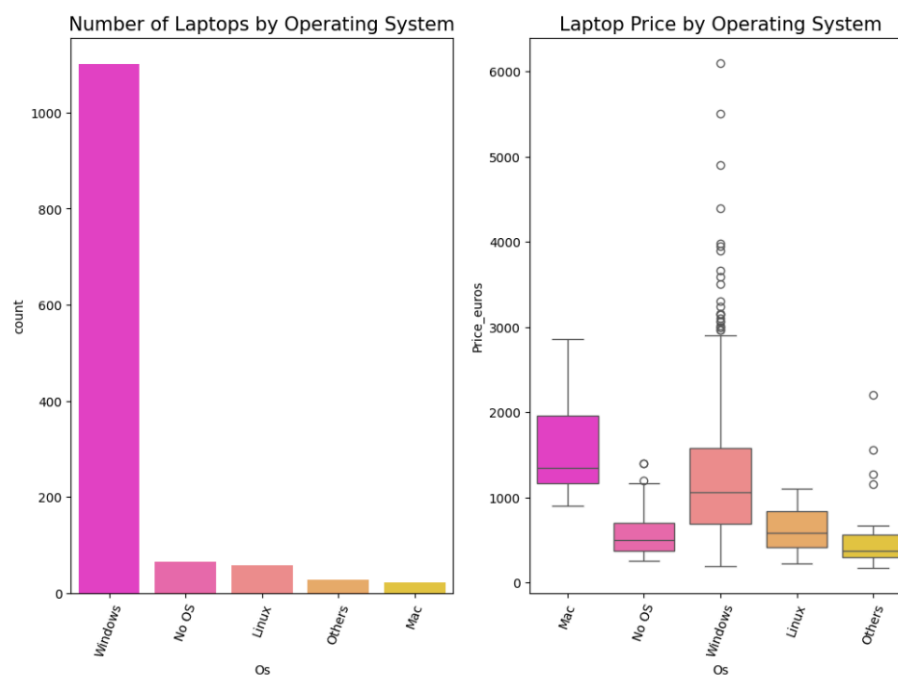


Рисунок 3.19 – Кількість ноутбуків для кожної операційної системи та розмах цін ноутбуків для кожної з них.

Найпоширенішими моделями графічних процесорів Intel. Nvidia є найдорожчим брендом графічних процесорів, тоді як AMD – найдешевшим. Intel є найпоширенішим брендом процесорів. У датасеті представлено 5 операційних систем: Windows, Mac, Chrome, Linux та Android. Також є ноутбуки без операційної системи (No OS). Ноутбуки з операційною системою Mac мають найвищу середню ціну, тоді як ноутбуки з Linux – найнижчу.

Не менш важливим при виборі ноутбуку є кількість Ram. Тому побудуємо відповідні графіки для дослідження, яка кількість оперативної пам'яті є найбільш поширеною та як розподілена ціна (рис. 3.20).

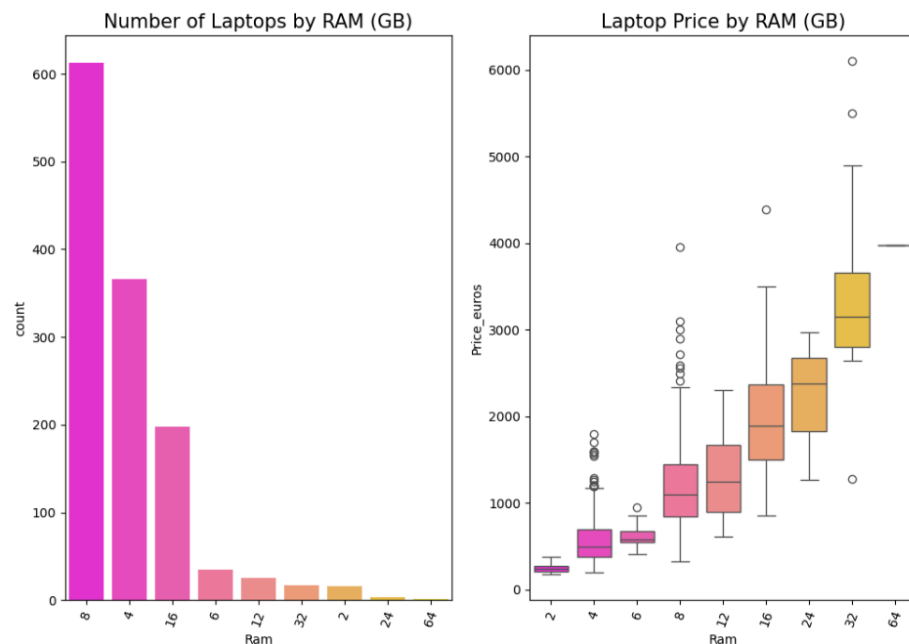


Рисунок 3.19 – Кількість ноутбуків для розмірності оперативної пам'яті та розмах цін ноутбуків в залежності від її кількості.

Обсяг оперативної пам'яті варіюється від 8 ГБ до 64 ГБ, причому найбільш поширеним є обсяг у 8 ГБ. Існує очевидний зв'язок між обсягом оперативної пам'яті та ціною: ціна зростає зі збільшенням обсягу оперативної пам'яті.

За допомогою кореляційної матриці (рис. 3.20) виявимо ступінь взаємозв'язку між різними змінними в наборі даних.

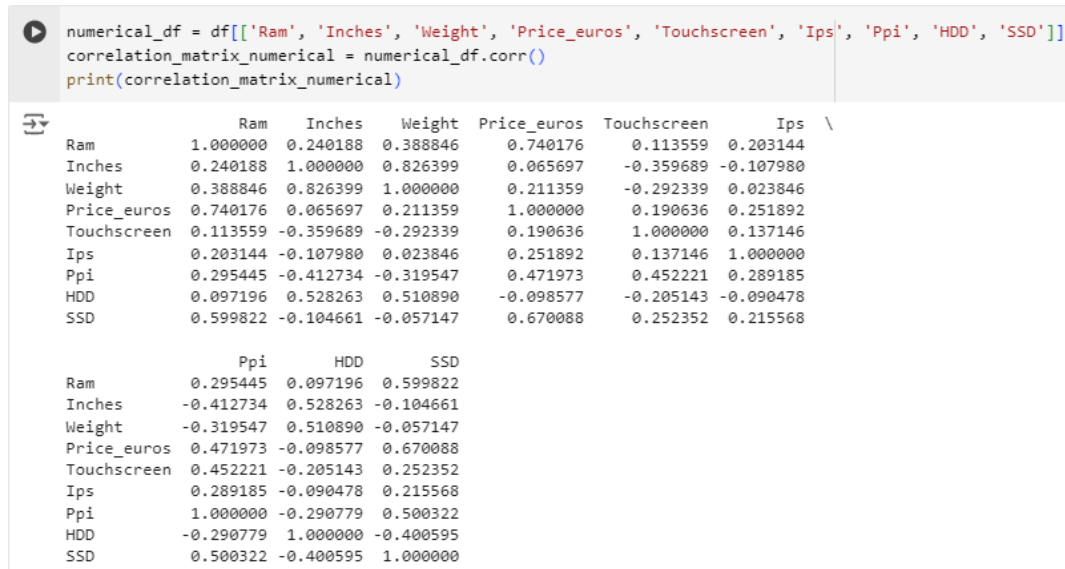


Рисунок 3.20 – Матриця кореляцій.

Як бачимо, з ціною гарно корелюють такі показники, як обсяг оперативної пам'яті та кількість SSD, помірно та слабо корелюють кількість пікселів на дюйм, вага та наявність IPS дисплею, майже не впливають на ціну такі ознаки як діагональ, наявність сенсорного екрану та HDD.

Також як ми вже впевнелись з попередніх діаграм, деякі якісні ознаки також можуть впливати на ціну. Для того, щоб перевірити, чи, наприклад тип ноутбуку впливає на ціну, скористаємося дисперсійним аналізом (ANOVA) це статистичний метод, який використовується для перевірки наявності суттєвих відмінностей між середніми значеннями двох або більше груп. Згрупуємо дані, щоб побачити чи впливають різні типи ноутбуків на ціну. Скористаємося функцією 'f_oneway' з модуля 'stats' для отримання F-test score та P-value (рис. 3.21).



Рисунок 3.21 – F-test score та P-value.

Але такий варіант дисперсійного аналізу дасть коректний результат, коли дані мають нормальний розподіл в групах. Перевіримо, чи нормально розподілені дані в кожній з груп, скористаємось функцією 'shapiro' з модуля 'stats' (рис. 3.22).

```
[128] print("rpyна 'Ultrabook'", stats.shapiro(grouped_test2.get_group('Ultrabook')['Price_euros']))
      print("rpyна 'Notebook'", stats.shapiro(grouped_test2.get_group('Notebook')['Price_euros']))
      print("rpyна 'Netbook'", stats.shapiro(grouped_test2.get_group('Netbook')['Price_euros']))
      print("rpyна 'Gaming'", stats.shapiro(grouped_test2.get_group('Gaming')['Price_euros']))
      print("rpyна '2 in 1 Convertible'", stats.shapiro(grouped_test2.get_group('2 in 1 Convertible')['Price_euros']))
      print("rpyна 'Workstation'", stats.shapiro(grouped_test2.get_group('Workstation')['Price_euros']))

rpyна 'Ultrabook' ShapiroResult(statistic=0.9831055998802185, pvalue=0.01956580951809883)
rpyна 'Notebook' ShapiroResult(statistic=0.840705394744873, pvalue=5.233807992956017e-26)
rpyна 'Netbook' ShapiroResult(statistic=0.7425427436828613, pvalue=5.261397018330172e-05)
rpyна 'Gaming' ShapiroResult(statistic=0.8571395874023438, pvalue=6.35895307764256e-13)
rpyна '2 in 1 Convertible' ShapiroResult(statistic=0.9625328183174133, pvalue=0.0025242732372134924)
rpyна 'Workstation' ShapiroResult(statistic=0.8799748420715332, pvalue=0.0033573973923921585)
```

Рисунок 3.22 –P-value для кожного типу.

P-value майже 0 або не перевищує рівня значущості 0.05, отже в усіх групах дані мають розподіл що відрізняється від нормального. Тому для дисперсійного аналізу скористаємось непараметричним аналогом ANOVA - тестом Краскела-Уоліса (рис. 3.23).

```
result = stats.kruskal(grouped_test2.get_group('Ultrabook')['Price_euros'], grouped_test2.get_group('Notebook')['Price_euros'],
                      grouped_test2.get_group('Netbook')['Price_euros'], grouped_test2.get_group('Gaming')['Price_euros'],
                      grouped_test2.get_group('2 in 1 Convertible')['Price_euros'], grouped_test2.get_group('Workstation')['Price_euros'])

print(result)

KruskalResult(statistic=545.9793387702136, pvalue=9.441917882554493e-116)
```

Рисунок 3.23 – Результат тесту Краскела-Уоліса.

Це чудовий результат із високим показником тесту, який показує сильну кореляцію, і P-value майже 0, що передбачає впевнену статистичну значущість.

Отже, якісні ознаки також впливають на ціноутворення ноутбуків. Тому за допомогою TargetEncoder призначимо кожній якісній ознаці відповідні значення(рис.3.24) і отримаємо такий суто числовий датафрейм (рис. 3.25).

```
!pip install category_encoders
from category_encoders import TargetEncoder

df2 = df.copy()
encoder = TargetEncoder()
features_object = ['Company', 'Product', 'TypeName', 'Gpu brand', 'Cpu brand', 'Os']

for col in features_object:
    df2[col] = encoder.fit_transform(df2[col], df2['Price_euros'])
```

Рисунок 3.24 – Використовуємо TargetEncoder.

df2

	Company	Product	TypeName	Inches	Ram	Weight	Price_euros	Touchscreen	Ips	Ppi	Cpu_brand	HDD	SSD	Gpu_brand	Os
0	1360.483088	1369.367509	1556.676019	13.3	8	1.37	1339.69	0	1	226.983005	1015.015248	0	128	1020.367713	1360.483088
1	1360.483088	1120.232308	1556.676019	13.3	8	1.34	898.94	0	0	127.677940	1015.015248	0	0	1020.367713	1360.483088
2	1080.314664	784.792911	788.744781	15.6	8	1.86	575.00	0	0	141.211998	1015.015248	0	256	1020.367713	593.420644
3	1360.483088	1369.367509	1556.676019	15.4	16	1.03	2537.45	0	1	220.534624	1612.277612	0	512	778.026625	1360.483088
4	1360.483088	1369.367509	1556.676019	13.3	8	1.37	1803.60	0	1	226.983005	1015.015248	0	256	1020.367713	1360.483088
...
1270	1093.862215	1070.634166	1295.140901	14.0	4	1.80	638.00	1	1	157.350512	1612.277612	0	128	1020.367713	1202.132652
1271	1093.862215	1191.515088	1295.140901	13.3	16	1.30	1499.00	1	1	276.053530	1612.277612	0	512	1020.367713	1202.132652
1272	1093.862215	998.690949	788.744781	14.0	2	1.50	229.00	0	0	111.935204	565.843499	0	0	1020.367713	1202.132652
1273	1080.314664	1087.027834	788.744781	15.6	6	2.19	764.00	0	0	100.454670	1612.277612	1000	0	778.026625	1202.132652
1274	1123.829758	1035.634986	788.744781	15.6	4	2.20	369.00	0	0	100.454670	565.843499	500	0	1020.367713	1202.132652

1274 rows x 15 columns

Рисунок 3.25 – Конвертування категоріальних змінних.

На основі цієї таблиці створимо нову кореляційну матрицю, яка відображатиме кореляцію і між якісними ознаками також (рис. 3.26).

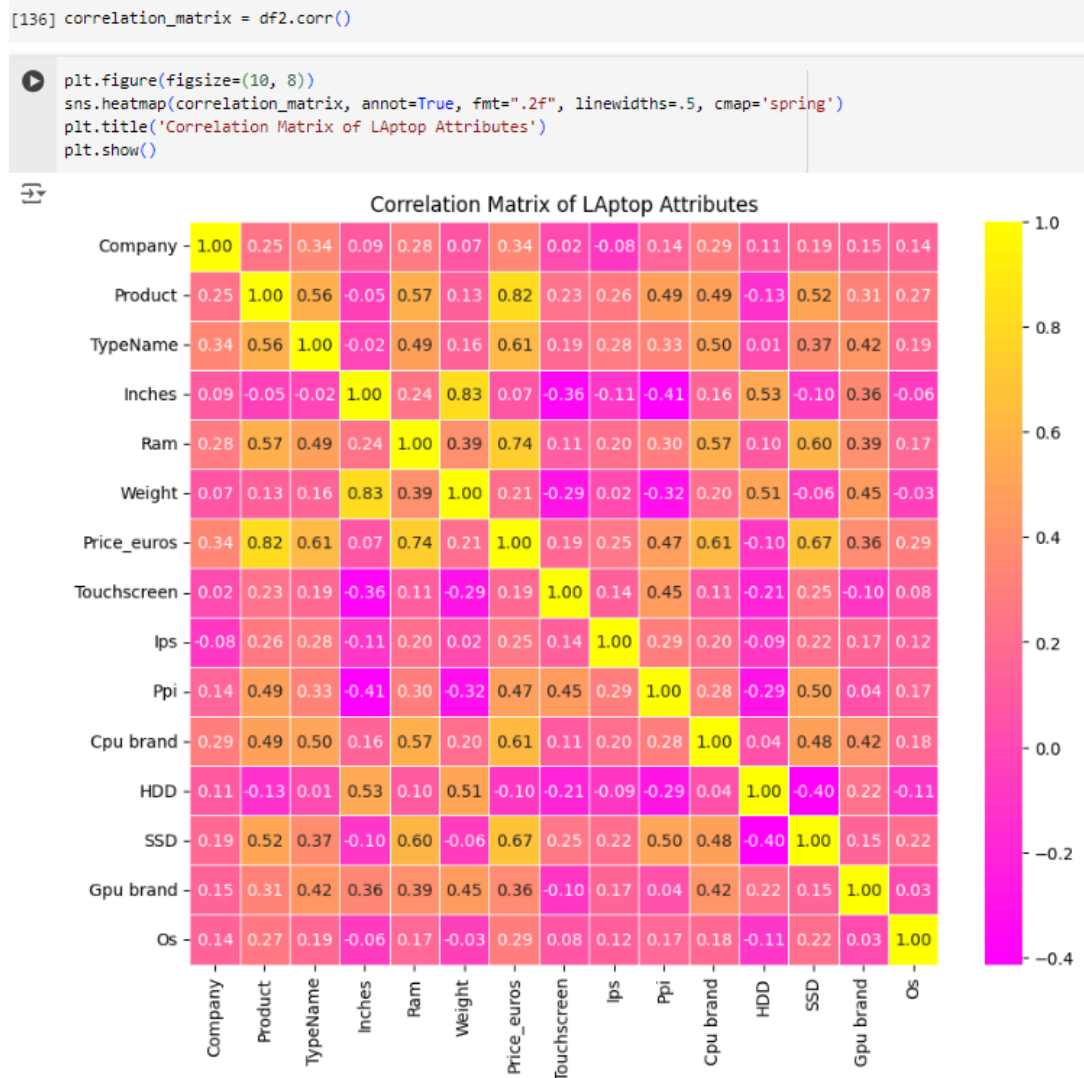


Рисунок 3.26 – Матриця кореляцій всіх ознак.

З даної діаграми видно, що якісні ознаки доволі сильно впливають на ціну, тому їх варто залишити для подальшого дослідження.

3.5 Поділ даних

На цьому етапі формуємо навчальні та тестові вибірки для використання з різними методами інтелектуального аналізу даних. Одним із поширених методів розбиття даних є випадковий розподіл, при якому дані випадково поділяються на дві групи: навчальну та тестову. Набір даних (X) є колекцією даних, яка буде використовуватися для роботи з методами. Цільова змінна (y) – це те, що модель намагається передбачити або пояснити. Таким чином, розділимо наш набір даних на 80% для тренування та 20% для тестування (рис. 3.37). Також при попередньому спостереженні ми виявили, що такі колонки, як Inches, HDD і Touchscreen не мають рішучого впливу на ціну, тому можемо видалити дані ознаки.

```
features_to_delete = ['Inches', 'HDD', 'Touchscreen']
df_model = df2.drop(columns=features_to_delete)

X = df_model.drop(['Price_euros'], axis=1)
y = df_model['Price_euros']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
print("y_train: ", y_train.shape)
print("y_test: ", y_test.shape)
```

```
X_train: (1019, 11)
X_test: (255, 11)
y_train: (1019,)
y_test: (255,)
```

Рисунок 3.27 – Формування тренувальних та тестових вибірок.

4. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ

4.1 Обґрунтування вибору методів інтелектуального аналізу даних

У курсовій роботі для прогнозування цін на ноутбуки було обрано такі методи інтелектуального аналізу: Multiple Linear regression[7], Ridge Regression[8], Gradient Boosting regressor[9] та Random Forest Regressor[10]. Спираючись на характеристики ноутбуків за допомогою математичних обчислень можна передбачити ціни на них, що є актуальним для покупців, аналітиків ринку та виробників.

Linear Regression – є найпоширенішим методом предиктивного аналізу. Вона використовує лінійні взаємозв'язки між залежною змінною (цільовою) та однією або більше незалежними змінними (предикторами) для прогнозування майбутнього значення цільової змінної. Прогноз базується на припущенні, що зв'язок між цільовою змінною та предикторами є залежним або причинним.

Вона шукає оптимальну лінію, яка мінімізує суму квадратів різниць між передбаченими та фактичними значеннями. Цей метод використовується в різних галузях, таких як економіка та фінанси, для аналізу та прогнозування тенденцій даних. Лінійну регресію можна розширити до множинної лінійної регресії, що включає кілька незалежних змінних, а також до логістичної регресії, яка підходить для задач бінарної класифікації.

Множинна лінійна регресія, яка буде використовуватись в даному дослідженні, є технікою для розуміння взаємозв'язку між однією залежною змінною та кількома незалежними змінними. Формулювання для множинної лінійної регресії схоже на просту лінійну регресію, з тією невеликою зміною, що замість одного коефіцієнта тепер будуть коефіцієнти для всіх використаних змінних.

Ridge Regression - була розроблена як можливе рішення для усунення неточностей оцінок методу найменших квадратів, коли в лінійних регресійних

моделях є мультиколінеарні незалежні змінні, шляхом створення оцінювача гребеневої регресії (RR). Це забезпечує більш точну оцінку параметрів гребеневої регресії, оскільки її дисперсія та середньоквадратична помилка часто є меншими за оцінки методу найменших квадратів.

Вона застосовується у багатьох галузях, включаючи економетрику, хімію та інженерію. Особливо корисною вона є для пом'якшення проблеми мультиколінеарності в лінійній регресії, яка часто виникає в моделях з великою кількістю параметрів.

Загалом, метод забезпечує підвищену ефективність оцінки параметрів в обмін на допустиму кількість зміщення). Гребенева регресія надає більш точні оцінки параметрів, оскільки її дисперсія та середньоквадратична помилка часто є меншими за відповідні показники методу найменших квадратів.

Gradient Boosting Regressor - техніка машинного навчання, яка поєднує прогнози кількох слабких моделей, зазвичай рішень дерев, послідовно. Вона спрямована на підвищення загальної ефективності прогнозування шляхом оптимізації ваг моделі на основі помилок попередніх ітерацій, поступово зменшуючи помилки прогнозу та покращуючи точність моделі. Ця техніка найчастіше використовується для лінійної регресії.

Gradient Boosting regressor допомагає мінімізувати помилку зміщення моделі. Основна ідея цього алгоритму полягає у послідовному побудові моделей, де кожна наступна модель намагається зменшити помилки попередньої моделі.

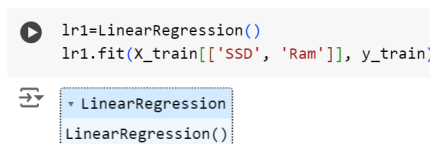
Random Forest Regressor - є методом ансамблевого навчання, який поєднує прогнози кількох дерев рішень для отримання більш точного та стабільного прогнозу. Це тип алгоритму може використовуватися як для класифікаційних, так і для регресійних задач.

Кожне дерево рішень має високу дисперсію, але коли ми комбінуємо всі дерева паралельно, то результуюча дисперсія є низькою, оскільки кожне дерево рішень навчається на своїй вибірці даних. Таким чином, вихідний результат не залежить від одного дерева рішень, а від багатьох дерев. У випадку задачі

класифікації, кінцевий результат визначається за допомогою методу більшості голосів. У випадку задачі регресії, кінцевий результат є середнім значенням усіх вихідних даних.

4.2 Аналіз отриманих результатів для методу Multiple Linear regression

Спершу створимо об'єкт множинної лінійної регресії, використовуючи ознаки SSD та Ram (рис. 4.1), обираємо саме їх, адже бачили на матриці кореляцій, що вони мають доволі сильну кореляцію з ціною.



```
lr1=LinearRegression()
lr1.fit(X_train[['SSD', 'Ram']], y_train)
```

LinearRegression
LinearRegression()

Рисунок 4.1 – Об'єкт лінійної регресії.

Спробуємо дану модель на тренувальних даних, потім перевіримо точність прогнозування на тренувальних та тестових вибірках (рис. 4.2).



```
lr1.score(X_test[['SSD', 'Ram']], y_test)
```

0.6213069884541926

```
lr1.score(X_train[['SSD', 'Ram']], y_train)
```

0.629068871757581

Рисунок 4.2 – Перевіряємо точність моделі.

Ми маємо задовільну точність для тренувальних даних, що свідчить про те, що модель доволі добре обраховує все, але нас більше цікавлять тестові дані, які також мають добрий результат.

Переглянемо та порівняємо справжні дані та ті, що передбачила модель (рис. 4.3).

```

train_pred=lr1.predict(X_train[['SSD', 'Ram']])
train_pred[0:5]

array([ 721.88221179,  864.59249479, 1186.48873088, 1034.65953036,
        721.88221179])

[195] pred=lr1.predict(X_test[['SSD', 'Ram']])
pred[0:5]

array([ 560.93409375,  560.93409375, 1186.48873088, 1186.48873088,
        1186.48873088])

[196] results = pd.DataFrame({'Actual': y_test, 'Predicted': pred})
results[0:5]

```

	Actual	Predicted
309	469.00	560.934094
622	776.00	560.934094
184	1199.00	1186.488731
705	795.00	1186.488731
522	1279.73	1186.488731

Рисунок 4.3 – Справжні дані та передбачені моделлю.

Для візуалізації моделі множинної лінійної регресії використовуємо діаграму розподілу. Розглянемо розподіл прогнозованих значень для навчальних даних (рис. 4.4.) та тестових даних (рис.4.5).

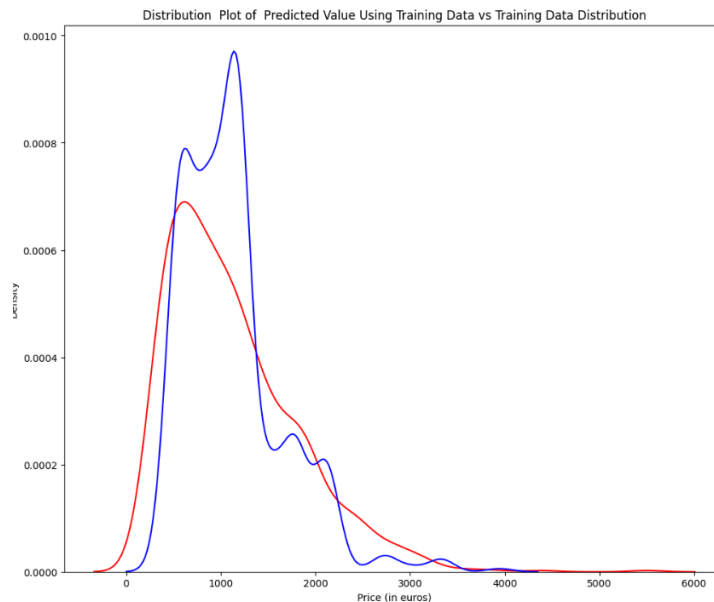


Рисунок 4.4 – Діаграма прогнозованих значень на основі навчальних даних порівняно з фактичними значеннями навчальних даних.

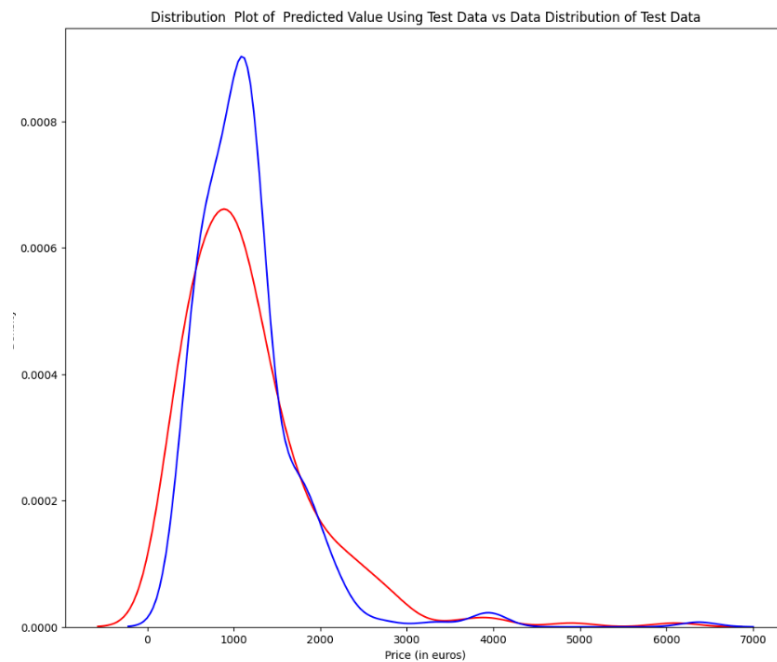


Рисунок 4.5 – Діаграма прогнозованих значень на основі тестових даних порівняно з фактичними значеннями тестових даних.

Як бачимо модель задовільно передає результат, тому проведемо обрахунок помилок MSE (Mean Squared Error) і MAE (Mean Absolute Error) (рис. 4.6), що є важливою частиною оцінки результатів моделі машинного навчання. MSE вимірює середню квадратичну помилку між фактичними та прогнозованими значеннями. MAE вимірює середню абсолютну помилку між фактичними та прогнозованими значеннями. Він також використовується для оцінки точності прогнозів у регресійних моделях. MAE вказує на середню абсолютну величину помилки прогнозу, а не на квадратичну величину, як у випадку MSE.

```
print("MSE :",mean_squared_error(y_test, pred))
print("MAE: ",mean_absolute_error(y_test, pred))
```

MSE : 228136.89328929197
MAE: 329.83738180497687

Рисунок 4.6 – Обрахунок MSE і MAE.

4.3 Аналіз отриманих результатів для методу Ridge Regression

Для початку, щоб аналізувати результати роботи методу Ridge Regression , необхідно обрати найкращу з можливих моделей, для цього ми використали модуль, що перебирає всі можливі варіанти параметрів та дає зрозуміти, які параметри найкраще вирішують поставлену задачу (рис. 4.7).

```
from sklearn.model_selection import GridSearchCV
rr=Ridge()

parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 1000000]}]

rdg_grid_search = GridSearchCV(rr, parameters1, cv=5)

rdg_grid_search.fit(X_train, y_train)

GridSearchCV
  estimator: Ridge
    Ridge

rdg=rdg_grid_search.best_estimator_

rdg.fit(X_train, y_train)

Ridge
Ridge(alpha=100)
```

Рисунок 4.7 – Створюємо об'єкт гребеневої регресії з найкращими параметрами.

Далі перевіримо точність прогнозування на тренувальних та тестових вибірках (рис. 4.8).

```
print('Train accuracy = ', rdg.score(X_train, y_train))
print('Test accuracy = ', rdg.score(X_test, y_test))

Train accuracy = 0.8306459216819193
Test accuracy = 0.8137942600547221
```

Рисунок 4.8 – Перевіряємо точність моделі.

Ми маємо гарну точність для тренувальних даних, що свідчить про те, що модель доволі добре обраховує все. Аналогічно для тестових даних точність досить гарна. Переглянемо та порівняємо справжні дані та ті, що передбачила модель (рис. 4.9).

```
rdg_train_pred = rdg.predict(X_train)
rdg_train_pred[0:5]
```

```
array([ 694.8040171 , 250.15697359, 1282.40226716, 1067.7249455 ,
        862.42274157])
```

```
rdg_pred = rdg.predict(X_test)
rdg_pred[0:5]
```

```
array([ 54.07359124, 639.00423993, 970.16978407, 1002.79099095,
        1105.94546112])
```

```
results = pd.DataFrame({'Actual': y_test, 'Predicted': rdg_pred})
results[0:5]
```

	Actual	Predicted
309	469.00	54.073591
622	776.00	639.004240
184	1199.00	970.169784
705	795.00	1002.790991
522	1279.73	1105.945461

Рисунок 4.9 – Перевіряємо точність моделі.

Аналогічно до множинної лінійної регресії скористаємося діаграмою розподілу. Відповідно розглянемо розподіл прогнозованих значень для навчальних даних (рис. 4.10) та тестових даних (рис.4.11).

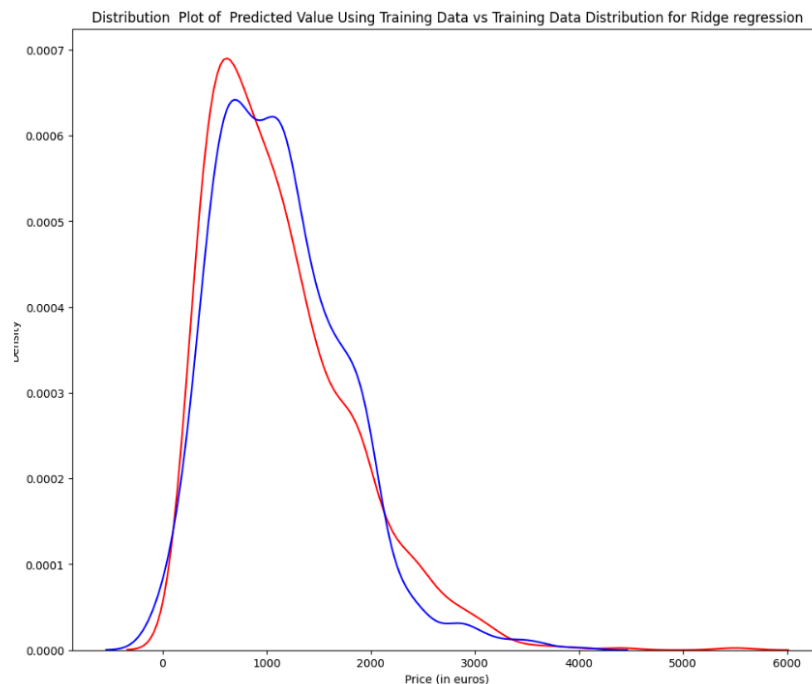


Рисунок 4.10 – Діаграма прогнозованих значень на основі навчальних даних порівняно з фактичними значеннями навчальних даних.

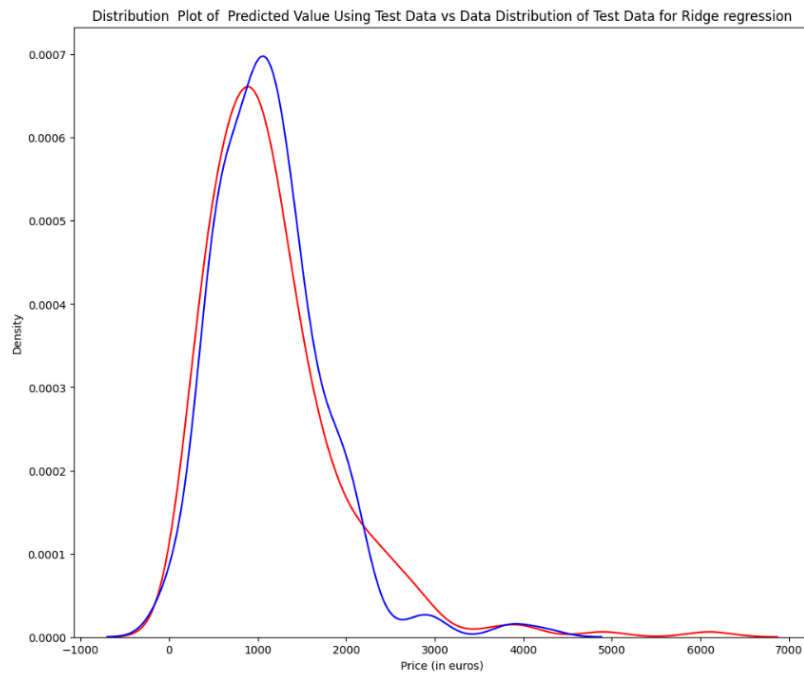


Рисунок 4.11 – Діаграма прогнозованих значень на основі тестових даних порівняно з фактичними значеннями тестових даних.

Як бачимо гребенева прогресія краще передає результат, тому також проведемо обрахунок помилок MSE (Mean Squared Error) і MAE (Mean Absolute Error) (рис. 4.12).

```
print("MSE :",mean_squared_error(y_test, rdg_pred))
print("MAE: ",mean_absolute_error(y_test, rdg_pred))

MSE : 112176.34793508991
MAE: 209.48455270322435
```

Рисунок 4.12 – Обрахунок MSE і MAE.

Бачимо, що гребенева регресія краще прогнозує дані, а також її точність вища, Mean Squared Error та Mean Absolute Error нижчі, ніж у множинної лінійної регресії, що стверджує, що дана модель справляється з прогнозуванням краще за попередню, але ще є місце для вдосконалення.

4.4 Аналіз отриманих результатів для методу Gradient Boosting Regressor

Знову ж таки, перед тим як аналізувати результати роботи методу Gradient Boosting Regressor, необхідно обрати найкращу з можливих моделей, для цього ми

використали модуль, що перебирає всі можливі варіанти параметрів та дає зрозуміти, які параметри найкраще вирішують поставлену задачу (рис. 4.13).

Визначаємо словник `parameters1`, який міститиме параметри, що використовуватимуться для налаштування моделі. Кожен ключ відповідає параметру, а значення цього ключа - це список значень, які потрібно перевірити під час налаштування моделі за допомогою перехресної перевірки:

`N_estimators` - цей параметр визначає кількість дерев у градієнтному бустингу. Більше дерев може дати кращі результати, але також може збільшити час навчання та ризик перенавчання. Тому ми перевіряємо декілька значень, щоб знайти оптимальну кількість дерев. Значення, які перевіряються: [50, 100, 200]

`Learning_rate` - цей параметр визначає швидкість навчання моделі. Низький `learning_rate` дозволяє моделі більш точно адаптуватися до даних, але може знадобитися більше дерев для досягнення оптимальної точності. Значення, які перевіряються: [0.01, 0.1, 0.2]

`Max_depth` - цей параметр визначає максимальну глибину кожного дерева у градієнтному бустингу. Глибокі дерева можуть дати кращі результати на тренувальному наборі даних, але можуть призвести до перенавчання. Значення, які перевіряються: [3, 4, 5]

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
import pandas as pd

gbr = GradientBoostingRegressor()

parameters1 = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

gbr_grid_search = GridSearchCV(gbr, parameters1, cv=5)

gbr_grid_search.fit(X_train, y_train)
```

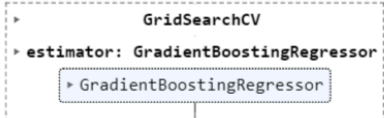


Рисунок 4.13 – Створюємо об'єкт Gradient Boosting Regressor з найкращими параметрами.

Далі тренуємо модель, використовуючи найкращі параметри, так наприклад бачимо, що максимальна глибина дерева в найкращому випадку має бути 4 (рис. 4.14).

```
gbr = gbr_grid_search.best_estimator_

gbr.fit(X_train, y_train)
```

▼ GradientBoostingRegressor

```
GradientBoostingRegressor(max_depth=4)
```

Рисунок 4.14 – Тренування моделі Gradient Boosting Regressor.

Розрахуємо точність прогнозування на тренувальних та тестових вибірках (рис. 4.15).

```
print('Train accuracy = ', gbr.score(X_train, y_train))
print('Test accuracy = ', gbr.score(X_test, y_test))
```

Train accuracy = 0.9751823167841553
Test accuracy = 0.9249317825767336

Рисунок 4.15 – Перевіряємо точність моделі Gradient Boosting Regressor.

Ми дуже гарну точність для тренувальних даних, що свідчить про те, що модель дуже добре все обраховує. Аналогічно для тестових даних точність є високою. Переглянемо та порівняємо справжні дані та ті, що передбачила модель (рис. 4.16).

```
gbr_train_pred = gbr.predict(X_train)
print(gbr_train_pred[0:5])
```

[601.75683457 637.8331872 1370.53572255 813.45859508 969.44090417]

```
gbr_pred = gbr.predict(X_test)
print(gbr_pred[0:5])
```

[401.10586871 628.11789229 1065.28678891 762.87539792 1097.95781949]

```
results = pd.DataFrame({'Actual': y_test, 'Predicted': gbr_pred})
print(results[0:5])
```

	Actual	Predicted
309	469.00	401.105869
622	776.00	628.117892
184	1199.00	1065.286789
705	795.00	762.875398
522	1279.73	1097.957819

Рисунок 4.16 – Перевіряємо точність моделі Gradient Boosting Regressor.

Аналогічно до попередньо розглянутих регресій, скористаємося діаграмою розподілу. Відповідно розглянемо розподіл прогнозованих значень для навчальних даних (рис. 4.17) та тестових даних (рис.4.18).

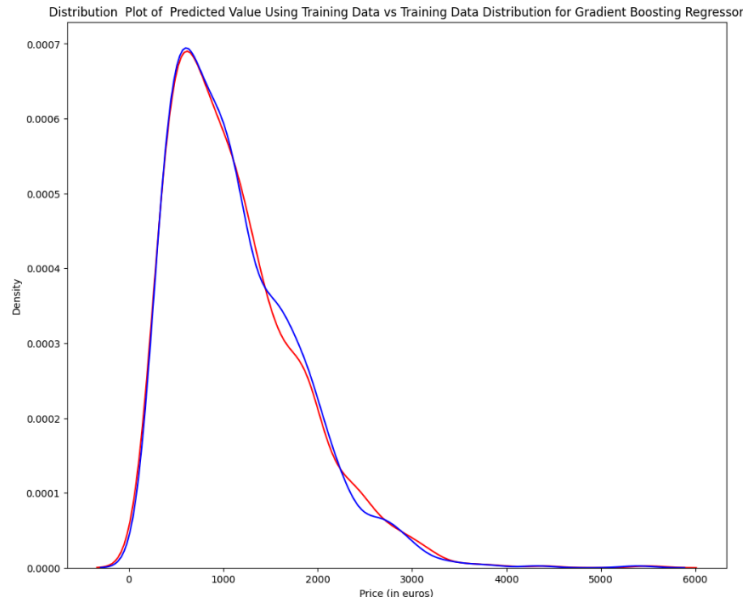


Рисунок 4.17 – Діаграма прогнозованих значень на основі навчальних даних порівняно з фактичними значеннями навчальних даних Gradient Boosting Regressor.

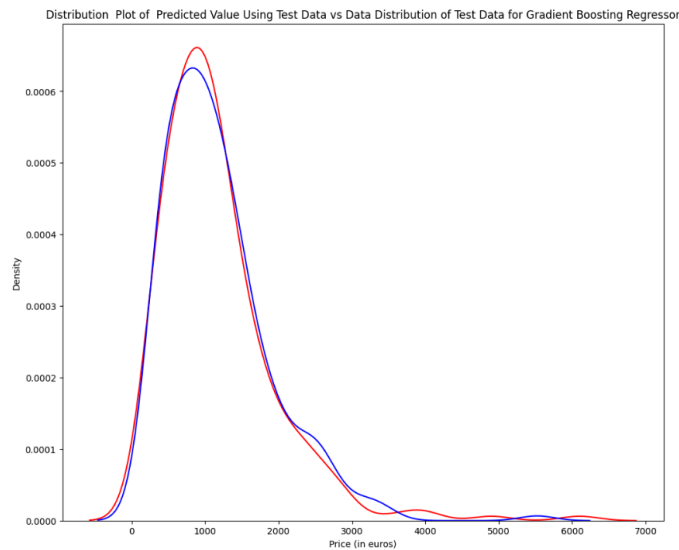


Рисунок 4.18 – Діаграма прогнозованих значень на основі тестових даних порівняно з фактичними значеннями тестових даних Gradient Boosting Regressor.

Як бачимо Gradient Boosting Regressor ще краще передає результат, тому також проведемо обрахунок помилок MSE (Mean Squared Error) і MAE (Mean Absolute Error) (рис. 4.19).

```
print("MSE :",mean_squared_error(y_test, gbr_pred))
print("MAE: ",mean_absolute_error(y_test, gbr_pred))

MSE : 45223.517164476405
MAE: 115.36244195883211
```

Рисунок 4.19 – Обрахунок MSE і MAE для Gradient Boosting Regressor.

Як можемо спостерігати значення MSE і MAE покращуються, в порівнянні з минулими моделями, що свідчить про гарну якість даної моделі.

4.5 Аналіз отриманих результатів для методу Random Forest Regressor

Перед навчанням моделі треба попередньо обрати найкращу модель за найкращим параметрами для моделі RandomForestRegressor за допомогою GridSearchCV (рис.4.20):

Max_depth - цей параметр визначає максимальну глибину кожного дерева в лісі. Глибші дерева можуть вловлювати більш складні патерни в даних, але можуть призвести до перенавчання. Тому важливо знайти баланс між глибиною дерев і ризиком перенавчання. Значення, які перевіряються: [10, 20, 30]

Max_features - цей параметр визначає максимальну кількість ознак, які розглядаються для розщеплення при кожному вузлі дерева. Менше значення може зменшити ризик перенавчання, в той час як більше значення може дати кращі результати на тренувальному наборі. Значення, які перевіряються: [5, 10, 20]

N_estimators - цей параметр визначає кількість дерев у випадковому лісі. Більше дерев може поліпшити продуктивність моделі, але також збільшує час навчання і може призвести до перенавчання, якщо їх занадто багато. Значення, які перевіряються: [10, 50, 100, 150]

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 0)

parameters2 = {
    'max_depth': [10,20,30],
    'max_features' : [5,10,20],
    'n_estimators': [10,50,100,150]
}

rf_grid_search = GridSearchCV(rf, param_grid=parameters2, cv=5)
rf_grid_search.fit(X_train, y_train)
```

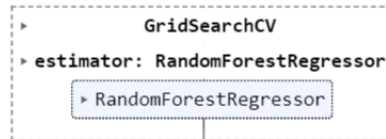


Рисунок 4.20 – Створюємо об’єкт Random Forest Regressor з найкращими параметрами.

Як бачимо, найкраща глибина виявилась 10, максимальна кількість ознак – 5 і кількість дерев – 150 (рис. 4.21), тому тренуємо саме таку модель.

```
rf = rf_grid_search.best_estimator_
rf.fit(X_train, y_train)
```

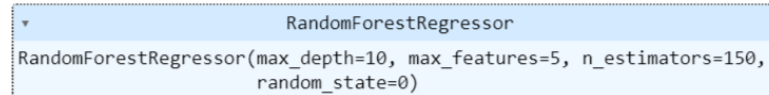


Рисунок 4.21 – Тренування моделі Random Forest Regressor.

Розрахуємо точність прогнозування на тренувальних та тестових вибірках (рис. 4.22).

```
print('Train accuracy = ', rf.score(X_train, y_train))
print('Test accuracy = ', rf.score(X_test, y_test))
```

```
Train accuracy = 0.9777972046941231
Test accuracy = 0.9015771489519901
```

Рисунок 4.22 – Перевіряємо точність моделі Random Forest Regressor.

Знову ж таки маємо гарну точність для тренувальних даних, що свідчить про те, що модель дуже добре все обраховує. Аналогічно для тестових даних точність є високою. Переглянемо та порівняємо справжні дані та ті, що передбачила модель (рис. 4.23).

```
rf_train_pred = rf.predict(X_train)
rf_train_pred[0:5]
```

```
array([ 612.455924 , 688.834313 , 1334.27835866, 823.59145387,
        1002.35671598])
```

```
rf_pred=rf.predict(X_test)
rf_pred[0:5]
```

```
array([ 447.07888353, 651.08284042, 1078.56396759, 787.83440694,
        1148.17888032])
```

```
results = pd.DataFrame({'Actual': y_test, 'Predicted': rf_pred})
results[0:5]
```

	Actual	Predicted
309	469.00	447.078884
622	776.00	651.082840
184	1199.00	1078.563968
705	795.00	787.834407
522	1279.73	1148.178880

Рисунок 4.23 – Перевіряємо точність моделі Random Forest Regressor.

Аналогічно до попередньо розглянутих регресій, скористаємося діаграмою розподілу. Відповідно розглянемо розподіл прогнозованих значень для навчальних даних (рис. 4.24) та тестових даних (рис.4.25).

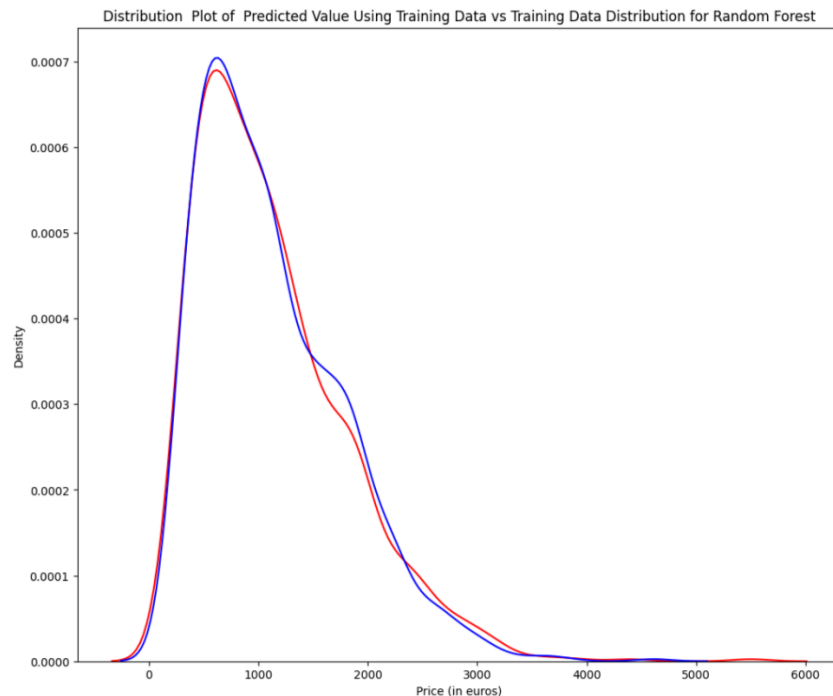


Рисунок 4.24 – Діаграма прогнозованих значень на основі навчальних даних порівняно з фактичними значеннями навчальних даних Random Forest Regressor.

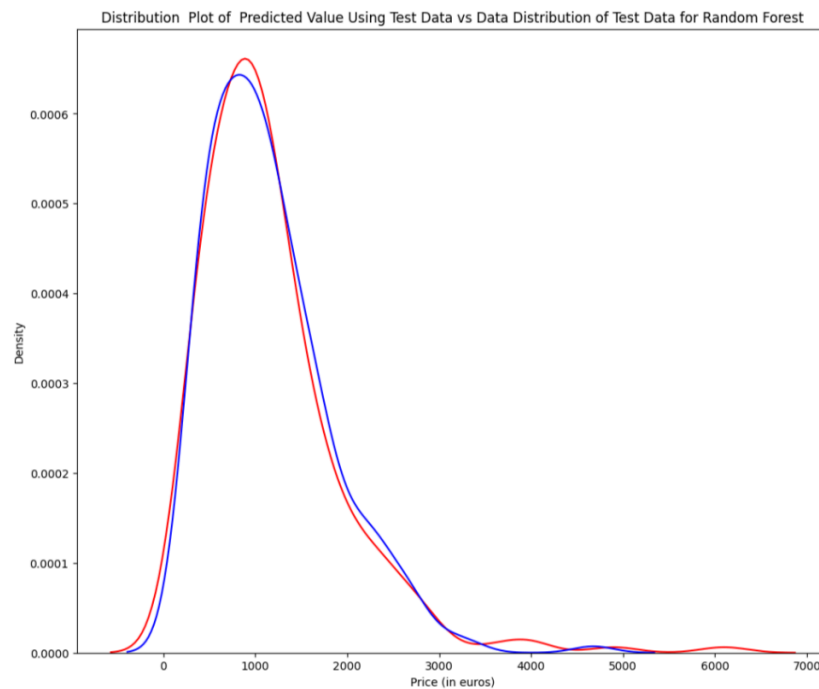


Рисунок 4.25 – Діаграма прогнозованих значень на основі тестових даних порівняно з фактичними значеннями тестових даних Random Forest Regressor.

Як бачимо Random Forest Regressor також дуже добре прогнозує ціну, тому також проведемо обрахунок помилок MSE (Mean Squared Error) і MAE (Mean Absolute Error) (рис. 4.26).

```
print("MSE :", mean_squared_error(y_test, rf_pred))
print("MAE: ", mean_absolute_error(y_test, rf_pred))

MSE : 59293.10228121713
MAE: 115.50885649492464
```

Рисунок 4.26 – Обрахунок MSE і MAE для Random Forest Regressor.

Як можемо спостерігати значення MSE і MAE показують гарний результат, що свідчить про гарну якість даної моделі.

4.6 Порівняння отриманих результатів методів

Отримані результати свідчать про роботу даних алгоритмів для прогнозування цін на навчальних і тестових даних. Підіб'ємо підсумки, порівнявши результати роботи моделей. Для цього обчислимо для кожної моделі MSE, MAE, R^2 (рис. 4.27).

MSE (Mean Squared Error): середньоквадратична помилка – це середнє значення квадратів різниць між фактичними та прогнозованими значеннями. Недоліком є те, що дана оцінка сильно карає великі помилки, оскільки помилки підносяться до квадрату.

MAE (Mean Absolute Error): середня абсолютна помилка – це середнє значення абсолютних різниць між фактичними та прогнозованими значеннями. Даний метод оцінки менш чутливий до викидів, оскільки використовує абсолютні значення помилок і не так сильно карає великі помилки, як MSE.

R^2 (Coefficient of Determination): коефіцієнт детермінації – це показник, який визначає, яку частку дисперсії залежної змінної пояснює модель. Інтерпретується як відсоток варіації залежної змінної, який пояснюється незалежними змінними моделі.

Ці метрики допоможуть оцінити точність і ефективність регресійної моделі, дозволяючи вибрати найбільш підходящу модель для прогнозування.


```

results = {
    'Multiple Linear Regression': {
        'RMSE': np.sqrt(mean_squared_error(y_test, pred)),
        'MAE': mean_absolute_error(y_test, pred),
        'R²': r2_score(y_test, pred)
    },
    'Ridge Regression': {
        'RMSE': np.sqrt(mean_squared_error(y_test, rdg_pred)),
        'MAE': mean_absolute_error(y_test, rdg_pred),
        'R²': r2_score(y_test, rdg_pred)
    },
    'Gradient Boosting Regressor': {
        'RMSE': np.sqrt(mean_squared_error(y_test, gbr_pred)),
        'MAE': mean_absolute_error(y_test, gbr_pred),
        'R²': r2_score(y_test, gbr_pred)
    },
    'Random Forest Regressor': {
        'RMSE': np.sqrt(mean_squared_error(y_test, rf_pred)),
        'MAE': mean_absolute_error(y_test, rf_pred),
        'R²': r2_score(y_test, rf_pred)
    }
}

results_df = pd.DataFrame(results).T
print(results_df)

```

	RMSE	MAE	R²
Multiple Linear Regression	477.636780	329.837382	0.621307
Ridge Regression	334.927377	209.484553	0.813794
Gradient Boosting Regressor	213.110769	115.253500	0.924612
Random Forest Regressor	243.501750	115.508856	0.901577

Рис. 4.27 - Обрахунок MSE, MAE, R^2 для кожної моделі.

Як бачимо Multiple Linear regression дає 62% точності для прогнозування даних, Ridge Regression – 81%, Gradient Boosting Regressor – 92% та Random Forest Regressor – 90%.

Проаналізувавши результати, можна зробити висновок про перевагу Gradient Boosting Regressor та Random Forest Regressor над Multiple Linear Regression та Ridge Regression. Згідно з отриманими результатами дослідження, Gradient Boosting Regressor виявився найкращою моделлю з точки зору прогнозування. Також він показав найнижчі значення MSE та MAE, що свідчить про меншу середню квадратичну та середню абсолютну помилку порівняно з іншими моделями. Крім того, коефіцієнт детермінації R^2 для Gradient Boosting Regressor виявився найвищим, що підтверджує його кращу здатність до прогнозування вихідної змінної, що ми і намагалися досягти в даному дослідженні.

Отже, на основі аналізу метрик оцінки моделей регресії в цьому дослідженні можна визначити Gradient Boosting Regressor як найефективнішу модель.

ВИСНОВКИ

У даній курсовій роботі було проведено дослідження методів інтелектуального аналізу для прогнозування цін на ноутбуки за допомогою різних моделей.

Було вивчено теоретичні характеристики ноутбуків, що можуть впливати на ціноутворення, а також методи, які були використані для виконання поставленої мети. В ході дослідження застосували чотири методи для розробки моделей: Multiple Linear Regression, Ridge Regression, Gradient Boosting Regressor та Random Forest Regressor. Для цього завантажили та попередньо обробили дані про ноутбуки, а потім розділили їх на навчальну та тестову вибірки. В результаті було встановлено, що Gradient Boosting Regressor та Random Forest Regressor мають високі результати прогнозування цін на ноутбуки, в той час, як Multiple Linear Regression, Ridge Regression гірше справляються з поставленою задачею.

Після цього було проведено розрахунок MSE, MAE, R^2 для кожної моделі, що показало, Gradient Boosting Regressor є найкращою моделлю для прогнозування в даному прикладі, дуже близькі за показниками результати також показав метод Random Forest Regressor. У цих обох модолей малі MSE, MAE, і доволі високе (більше 90%) R^2 .

Отже, поставлені задачі були виконані, було зроблено певні висновки та обрано найкращий метод для подальшого прогнозування цін на ноутбуки – Gradient Boosting Regressor.

ПЕРЕЛІК ПОСИЛАНЬ

1. Документація мови програмування Python. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.python.org/doc/> .
2. Бібліотека Pandas. [Електронний ресурс] – Режим доступу до ресурсу: <https://pandas.pydata.org/docs/>
3. Бібліотека Matplotlib. [Електронний ресурс] – Режим доступу до ресурсу: <https://matplotlib.org/stable/>
4. Бібліотека Sklearn. [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/user_guide.html
5. Бібліотека Seaborn.[Електронний ресурс] – Режим доступу до ресурсу: <https://seaborn.pydata.org/introduction.html>
6. Бібліотека Numpy. [Електронний ресурс] – Режим доступу до ресурсу: <https://wiki.python.org/moin/NumPy>
7. Multiple Linear Regression. [Електронний ресурс] —Режим доступу до ресурсу:
<https://corporatefinanceinstitute.com/resources/data-science/multiple-linear-regression/>
8. Ridge Regression. [Електронний ресурс] —Режим доступу до ресурсу:
https://en.wikipedia.org/wiki/Ridge_regression
9. Gradient Boosting Regressor. [Електронний ресурс] —Режим доступу до ресурсу:
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>
- 10.Random Forest Regressor. [Електронний ресурс] —Режим доступу до ресурсу:
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

*Тексти програмного коду аналіз
та прогнозування цін на ноутбуки*

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

1.57 МБ

(Обсяг програми (документа))

Студентки групи ІП-23 2 курсу

Піроженко Д. В.

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import RobustScaler, LabelEncoder

from google.colab import drive
drive.mount('/content/drive')
filename = "/content/drive/My Drive/laptop_price.csv"
df = pd.read_csv(filename, encoding='windows-1251')

df.head()

df.info()

df['Ram'] = df['Ram'].str.replace('GB',"")
df['Weight'] = df['Weight'].str.replace('kg',"")
df['Ram'] = df['Ram'].astype('int32')
df['Weight'] = df['Weight'].astype('float32')

df.head()

df['ScreenResolution'].value_counts()

df['Touchscreen'] = df['ScreenResolution'].apply(lambda x:1 if 'Touchscreen' in x else 0)
df['Ips'] = df['ScreenResolution'].apply(lambda x:1 if 'IPS' in x else 0)
df.head()

new = df['ScreenResolution'].str.split('x',n=1,expand=True)
df['X_res'] = new[0]
df['Y_res'] = new[1]
df['X_res'] = df['X_res'].str.replace(',',"").str.findall(r'(\d+\.\d+)').apply(lambda x:x[0])
df['X_res'] = df['X_res'].astype('int')
df['Y_res'] = df['Y_res'].astype('int')

df['Ppi'] = (((df['X_res']**2) + (df['Y_res']**2))*0.5/df['Inches']).astype('float')
df.drop(columns=['ScreenResolution', 'X_res', 'Y_res'],inplace=True)

df['Ppi']

df['Cpu Name'] = df['Cpu'].apply(lambda x:" ".join(x.split()[0:3]))

def fetch_processor(text):
    if text == 'Intel Core i7' or text == 'Intel Core i5' or text == 'Intel Core i3':
        return text

```

```

else:
    if text.split()[0] == 'Intel':
        return 'Other Intel Processor'
    else:
        return 'AMD Processor'

df['Cpu brand'] = df['Cpu Name'].apply(fetch_processor)
df.drop(columns=['Cpu','Cpu Name'],inplace=True)
df['Cpu brand']

df.head()

df['Memory'] = df['Memory'].astype(str).replace('\.0', '', regex=True)
df["Memory"] = df["Memory"].str.replace('GB', '')
df["Memory"] = df["Memory"].str.replace('TB', '000')
new = df["Memory"].str.split("+", n = 1, expand = True)

df["first"] = new[0]
df["first"] = df["first"].str.strip()

df["second"] = new[1]

df["Layer1HDD"] = df["first"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer1SSD"] = df["first"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer1Hybrid"] = df["first"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer1Flash_Storage"] = df["first"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['first'] = df['first'].str.replace(r'\D', '')

df["second"].fillna("0", inplace = True)

df["Layer2HDD"] = df["second"].apply(lambda x: 1 if "HDD" in x else 0)
df["Layer2SSD"] = df["second"].apply(lambda x: 1 if "SSD" in x else 0)
df["Layer2Hybrid"] = df["second"].apply(lambda x: 1 if "Hybrid" in x else 0)
df["Layer2Flash_Storage"] = df["second"].apply(lambda x: 1 if "Flash Storage" in x else 0)

df['second'] = df['second'].str.replace(r'\D', '')
df["first"] = df["first"].str.replace('SSD', '')
df["first"] = df["first"].str.replace('HDD', '')
df["first"] = df["first"].str.replace('Flash Storage', '')
df["first"] = df["first"].str.replace('Hybrid', '')

df["second"] = df["second"].str.replace('HDD', '')
df["second"] = df["second"].str.replace('SSD', '')
df["second"] = df["second"].str.replace('Hybrid', '')

df["first"] = df["first"].astype(int)
df["second"] = df["second"].astype(int)

```

```

df["HDD"]=(df["first"]*df["Layer1HDD"]+df["second"]*df["Layer2HDD"])
df["SSD"]=(df["first"]*df["Layer1SSD"]+df["second"]*df["Layer2SSD"])
df["Hybrid"]=(df["first"]*df["Layer1Hybrid"]+df["second"]*df["Layer2Hybrid"])
df["Flash_Storage"]=(df["first"]*df["Layer1Flash_Storage"]+df["second"]*df["Layer2Flash_Storage"])

df.drop(columns=['first', 'second', 'Layer1HDD', 'Layer1SSD', 'Layer1Hybrid',
                'Layer1Flash_Storage', 'Layer2HDD', 'Layer2SSD', 'Layer2Hybrid',
                'Layer2Flash_Storage'],inplace=True)

df.drop(columns=['Memory'],inplace=True)
df.head()

df['Gpu brand'] = df['Gpu'].apply(lambda x:x.split()[0])

df = df[df['Gpu brand'] != 'ARM']

df.drop(columns=['Gpu','Hybrid','Flash_Storage', 'laptop_ID'],inplace=True)

def cat_os(inp):
    if inp == 'Windows 10' or inp == 'Windows 7' or inp == 'Windows 10 S':
        return 'Windows'
    elif inp == 'macOS' or inp == 'Mac OS X':
        return 'Mac'
    elif inp == 'Linux':
        return 'Linux'
    elif inp == 'No OS':
        return 'No OS'
    else:
        return 'Others'

df['Os'] = df['OpSys'].apply(cat_os)
df.drop(columns=['OpSys'],inplace=True)
df['Os']

df.describe()

df.info()

df.isnull().sum()

df.duplicated().sum()

df.drop_duplicates(inplace=True)

fig, axes = plt.subplots(1,2, figsize=(24, 5))

```

```

for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)

sns.countplot(x="TypeName",ax=axes[0],palette='spring',data=df,order=df["TypeName"].value_counts().index)
sns.boxplot(x='Price_euros',y="TypeName",ax=axes[1], palette='spring', data = df)

axes[0].set_title("Number of Laptops by Type",fontsize=15)
axes[1].set_title("Laptop Price by Type ",fontsize=15)
plt.show()

fig, axes = plt.subplots(1,2, figsize=(20, 5))
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)

sns.countplot(x="Company",ax=axes[0],palette='spring',data=df,order=df["Company"].value_counts().index)
sns.boxplot(x='Price_euros',y="Company",palette='spring',data = df)

axes[0].set_title("Number of Laptops by Company",fontsize=15)
axes[1].set_title("Laptop Price by Company",fontsize=15)

fig, axes = plt.subplots(1,2, figsize=(12, 8))
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)

sns.countplot(x="Gpu brand",ax=axes[0],palette='spring',data=df,order=df["Gpu brand"].value_counts().index)
sns.boxplot(x='Gpu brand',y="Price_euros",palette='spring',ax=axes[1],data = df)

axes[0].set_title("Number of Laptops by GPU Brand",fontsize=15)
axes[1].set_title(" Laptop Price by CPU Brand",fontsize=15)

plt.show()

fig, axes = plt.subplots(1,2, figsize=(12, 8))
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)

sns.countplot(x="Cpu brand",ax=axes[0],palette='spring',data=df,order=df["Cpu brand"].value_counts().index)
sns.boxplot(x="Cpu brand",y="Price_euros",palette='spring',ax=axes[1],data = df)

axes[0].set_title("Number of Laptops by CPU Brand",fontsize=15)
axes[1].set_title("Laptop Price by CPU Brand",fontsize=15)

```



```

plt.show()

fig, axes = plt.subplots(1,2, figsize=(12, 8))
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)
sns.countplot(x="Os",ax=axes[0],palette='spring',data=df,order=df["Os"].value_counts().index)
sns.boxplot(x="Os",y='Price_euros',palette='spring',ax=axes[1],data = df)

axes[0].set_title("Number of Laptops by Operating System",fontsize=15)
axes[1].set_title("Laptop Price by Operating System",fontsize=15)
plt.show()

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))
for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=70)

sns.countplot(x="Ram",ax=axes[0],palette='spring',order=df["Ram"].value_counts().index,data=df)
sns.boxplot(x="Ram",y="Price_euros",palette='spring',data = df)

axes[0].set_title("Number of Laptops by RAM (GB)",fontsize=15)
axes[1].set_title("Laptop Price by RAM (GB)",fontsize=15)

plt.show()

numerical_df = df[['Ram', 'Inches', 'Weight', 'Price_euros', 'Touchscreen', 'Ips', 'Ppi', 'HDD', 'SSD']]
correlation_matrix_numerical = numerical_df.corr()
print(correlation_matrix_numerical)

df_gptest = df[['TypeName', 'Price_euros']]
grouped_test2=df_gptest[['TypeName', 'Price_euros']].groupby(['TypeName'])

from scipy import stats
f_val, p_val = stats.f_oneway(grouped_test2.get_group('Ultrabook')['Price_euros'],
grouped_test2.get_group('Notebook')['Price_euros'], grouped_test2.get_group('Netbook')['Price_euros'],
grouped_test2.get_group('Gaming')['Price_euros'], grouped_test2.get_group('2 in 1 Convertible')['Price_euros'],
grouped_test2.get_group('Workstation')['Price_euros'])

print( "ANOVA results: F=", f_val, ", P =", p_val)

print("группа 'Ultrabook'", stats.shapiro(grouped_test2.get_group('Ultrabook')['Price_euros']))
print("группа 'Notebook'", stats.shapiro(grouped_test2.get_group('Notebook')['Price_euros']))
print("группа 'Netbook'", stats.shapiro(grouped_test2.get_group('Netbook')['Price_euros']))
print("группа 'Gaming'", stats.shapiro(grouped_test2.get_group('Gaming')['Price_euros']))
print("группа '2 in 1 Convertible'", stats.shapiro(grouped_test2.get_group('2 in 1 Convertible')['Price_euros']))
print("группа 'Workstation'", stats.shapiro(grouped_test2.get_group('Workstation')['Price_euros']))

```

```

result = stats.kruskal(grouped_test2.get_group('Ultrabook')['Price_euros'],
grouped_test2.get_group('Notebook')['Price_euros'],
                        grouped_test2.get_group('Netbook')['Price_euros'],
grouped_test2.get_group('Gaming')['Price_euros'],
                        grouped_test2.get_group('2 in 1 Convertible')['Price_euros'],
grouped_test2.get_group('Workstation')['Price_euros'])

```

```

print(result)

```

```

!pip install category_encoders
from category_encoders import TargetEncoder

```

```

df2 = df.copy()
encoder = TargetEncoder()
features_object = ['Company', 'Product', 'TypeName', 'Gpu brand', 'Cpu brand', 'Os']

```

```

for col in features_object:
    df2[col] = encoder.fit_transform(df2[col], df2['Price_euros'])

```

```

df2

```

```

correlation_matrix = df2.corr()

```

```

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", linewidths=.5, cmap='spring')
plt.title('Correlation Matrix of LAptop Attributes')
plt.show()

```

```

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import
RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor, ExtraTreesRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor

```

```

features_to_delete = ['Inches', 'HDD', 'Touchscreen']
df_model = df2.drop(columns=features_to_delete)

```

```

X = df_model.drop(['Price_euros'], axis=1)
y = df_model['Price_euros']

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
print("y_train: ", y_train.shape)
print("y_test: ", y_test.shape)

lr1=LinearRegression()
lr1.fit(X_train[['SSD', 'Ram']], y_train)

lr1.score(X_test[['SSD', 'Ram']], y_test)
lr1.score(X_train[['SSD', 'Ram']], y_train)

train_pred=lr1.predict(X_train[['SSD', 'Ram']])
train_pred[0:5]

pred=lr1.predict(X_test[['SSD', 'Ram']])
pred[0:5]

results = pd.DataFrame({'Actual': y_test, 'Predicted': pred})
results[0:5]

def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.distplot(RedFunction, hist=False, color="r", label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", label=BlueName, ax=ax1)

    plt.title(Title)
    plt.xlabel('Price (in euros)')

    plt.show()
    plt.close()

Title = 'Distribution Plot of Predicted Value Using Training Data vs Training Data Distribution'
DistributionPlot(y_train, train_pred, "Actual Values (Train)", "Predicted Values (Train)", Title)

Title='Distribution Plot of Predicted Value Using Test Data vs Data Distribution of Test Data'
DistributionPlot(y_test, pred,"Actual Values (Test)","Predicted Values (Test)",Title)

print("MSE :",mean_squared_error(y_test, pred))
print("MAE: ",mean_absolute_error(y_test, pred))

from sklearn.model_selection import GridSearchCV

```

```

rr=Ridge()

parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 1000000]}]
rdg_grid_search = GridSearchCV(rr, parameters1, cv=5)
rdg_grid_search.fit(X_train, y_train)

rdg=rdg_grid_search.best_estimator_
rdg.fit(X_train, y_train)

print('Train accuracy = ', rdg.score(X_train, y_train))
print('Test accuracy = ', rdg.score(X_test, y_test))

rdg_train_pred = rdg.predict(X_train)
rdg_train_pred[0:5]

rdg_pred = rdg.predict(X_test)
rdg_pred[0:5]

results = pd.DataFrame({'Actual': y_test, 'Predicted': rdg_pred})
results[0:5]

Title = 'Distribution Plot of Predicted Value Using Training Data vs Training Data Distribution for Ridge
regression'
DistributionPlot(y_train, rdg_train_pred, "Actual Values (Train)", "Predicted Values (Train)", Title)

Title='Distribution Plot of Predicted Value Using Test Data vs Data Distribution of Test Data for Ridge
regression'
DistributionPlot(y_test, rdg_pred,"Actual Values (Test)","Predicted Values (Test)",Title)

print("MSE :",mean_squared_error(y_test, rdg_pred))
print("MAE: ",mean_absolute_error(y_test, rdg_pred))

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
import pandas as pd

gbr = GradientBoostingRegressor()

parameters1 = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

gbr_grid_search = GridSearchCV(gbr, parameters1, cv=5)

gbr_grid_search.fit(X_train, y_train)

```

```

gbr = gbr_grid_search.best_estimator_

gbr.fit(X_train, y_train)

print('Train accuracy = ', gbr.score(X_train, y_train))
print('Test accuracy = ', gbr.score(X_test, y_test))

gbr_train_pred = gbr.predict(X_train)
print(gbr_train_pred[0:5])

gbr_pred = gbr.predict(X_test)
print(gbr_pred[0:5])

results = pd.DataFrame({'Actual': y_test, 'Predicted': gbr_pred})
print(results[0:5])

Title = 'Distribution Plot of Predicted Value Using Training Data vs Training Data Distribution for Gradient
Boosting Regressor'
DistributionPlot(y_train, gbr_train_pred, "Actual Values (Train)", "Predicted Values (Train)", Title)

Title='Distribution Plot of Predicted Value Using Test Data vs Data Distribution of Test Data for Gradient
Boosting Regressor'
DistributionPlot(y_test, gbr_pred,"Actual Values (Test)","Predicted Values (Test)",Title)

print("MSE :",mean_squared_error(y_test, gbr_pred))
print("MAE: ",mean_absolute_error(y_test, gbr_pred))

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 0)

parameters2 = {
    'max_depth': [10,20,30],
    'max_features' : [5,10,20],
    'n_estimators': [10,50,100,150]
}

rf_grid_search = GridSearchCV(rf, param_grid=parameters2, cv=5)
rf_grid_search.fit(X_train, y_train)

rf = rf_grid_search.best_estimator_
rf.fit(X_train, y_train)

print('Train accuracy = ', rf.score(X_train, y_train))
print('Test accuracy = ', rf.score(X_test, y_test))

rf_train_pred = rf.predict(X_train)
rf_train_pred[0:5]

```

```
rf_pred=rf.predict(X_test)
rf_pred[0:5]
```

```
results = pd.DataFrame({'Actual': y_test, 'Predicted': rf_pred})
results[0:5]
```

```
Title = 'Distribution Plot of Predicted Value Using Training Data vs Training Data Distribution for Random Forest'
```

```
DistributionPlot(y_train, rf_train_pred, "Actual Values (Train)", "Predicted Values (Train)", Title)
```

```
Title='Distribution Plot of Predicted Value Using Test Data vs Data Distribution of Test Data for Random Forest'
```

```
DistributionPlot(y_test, rf_pred, "Actual Values (Test)", "Predicted Values (Test)", Title)
```

```
print("MSE :",mean_squared_error(y_test, rf_pred))
print("MAE: ",mean_absolute_error(y_test, rf_pred))
```

```
results = {
    'Multiple Linear Regression': {
        'RMSE': np.sqrt(mean_squared_error(y_test, pred)),
        'MAE': mean_absolute_error(y_test, pred),
        'R²': r2_score(y_test, pred)
    },
    'Ridge Regression': {
        'RMSE': np.sqrt(mean_squared_error(y_test, rdg_pred)),
        'MAE': mean_absolute_error(y_test, rdg_pred),
        'R²': r2_score(y_test, rdg_pred)
    },
    'Gradient Boosting Regressor': {
        'RMSE': np.sqrt(mean_squared_error(y_test, gbr_pred)),
        'MAE': mean_absolute_error(y_test, gbr_pred),
        'R²': r2_score(y_test, gbr_pred)
    },
    'Random Forest Regressor': {
        'RMSE': np.sqrt(mean_squared_error(y_test, rf_pred)),
        'MAE': mean_absolute_error(y_test, rf_pred),
        'R²': r2_score(y_test, rf_pred)
    }
}
```

```
results_df = pd.DataFrame(results).T
print(results_df)
```