База даних IKEA.

1. Завантажте цей набір даних IKEA.
2. Виконайте дослідницький аналіз для набору даних, включаючи описову статистику та візуалізації. Опишіть результати.
3. На основі EDA та вашого здорового глузду виберіть дві гіпотези, які ви хочете перевірити / проаналізувати. Для кожної гіпотези наведіть нульову гіпотезу та інші можливі альтернативні гіпотези, розробіть тести, щоб розрізняти їх, та виконайте їх. Опишіть результати.
4. Навчіть модель передбачати ціну на меблі.
   - Вкажіть, які стовпці слід виключити з моделі і чому.
   - Створіть конвеєр перехресної перевірки для навчання та оцінки моделі, включаючи (за необхідності) такі кроки, як заповнення пропущених значень та нормалізація.
   - Запропонуйте методи підвищення продуктивності моделі. Опишіть результати.

1. Попередній перегляд даних

Завантажимо дані з використанням бібліотек **requests** та **pandas.**

```
import requests
import pandas as pd
data = requests.get(
    'https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-11-03/ikea.csv').text
with open('dataset.csv', 'w', encoding="utf-8") as f:#'w'
    f.write(data)
data = pandas.read_csv('dataset.csv', sep=',')
# Показує скільки рядків та колонок має набір даних
data.info()
```

Маємо таблицю, яка складається з 3694 рядків та з наступних 14 стовпчиків:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3694 entries, 0 to 3693
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        3694 non-null   int64  
 1   item_id           3694 non-null   int64  
 2   name              3694 non-null   object 
 3   category          3694 non-null   object 
 4   price             3694 non-null   float64
 5   old_price         3694 non-null   object 
 6   sellable_online   3694 non-null   bool   
 7   link              3694 non-null   object 
 8   other_colors      3694 non-null   object 
 9   short_description 3694 non-null   object 
 10  designer          3694 non-null   object 
 11  depth             2231 non-null   float64
 12  height            2706 non-null   float64
```

```
 13  width           3105 non-null   float64
dtypes: bool(1), float64(4), int64(2), object(7)
memory usage: 378.9+ KB
```

Щоб детальніше розглянути дані створимо базу даних **sql_step_project.db** та першу таблицю в ній **all_data.** Для цього використаємо бібліотеку **sqlite3**

```
conn = sqlite3.connect('sql_step_project.db', check_same_thread=False, )
cursor = conn.cursor()
def delete_table():
   cursor.execute("DROP TABLE IF EXISTS all_data")
delete_table()

columns = ','.join(["" + col + "" + ' ' + 'TEXT' for col in data.columns])

def creationandfilingDB():
   cursor.execute(f"""CREATE TABLE IF NOT EXISTS all_data ({columns})""")

   for x in data.values:
      cursor.execute("""INSERT INTO all_data VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)""",(x))
      conn.commit()

# Створення та наповнення бази даних
creationandfilingDB()
```

Опишемо дані, оцінимо їх якість.

Стовпчик ‘**Unnamed: 0’** містить порядкові номери даних. Це числа від 0 до 3693. Пропущених значень нема.

Стовпчик ‘**item_id’** містить коди товарів у числовому форматі. Серед значень трапляються повтори, що може вказувати на те, що таблиця містить дублікати. 2962 унікальних кодів. Пропущених значень нема.

```
#запит підрахунку кількості унікальних записів поля item_id
SELECT count(DISTINCT item_id)
FROM all_data;  --2962
```

Стовпчик ‘**name’** містить назви товарів в текстовому форматі. Також є повтори. 607 унікальних імен. Пропущених значень нема.

```
#запит підрахунку кількості унікальних записів поля name
SELECT count(DISTINCT name)
FROM all_data;  --607
```

Стовпчик ‘**category’** містить назви категорій, до яких належать товари в текстовому форматі. Є повтори даних, 17 унікальних категорій, пропущених значень нема.

```
#запит підрахунку кількості унікальних записів поля category
SELECT count(DISTINCT category)
FROM all_data;--17
```

Стовпчик ‘**price**’ містить ціну товару в числовому форматі (у вигляді дійсного числа через крапку, одне число після крапки). Містить повтори, 979 унікальних записів, пропущених значень немає.

```
#запит підрахунку кількості унікальних записів поля price
SELECT count(DISTINCT price)
FROM all_data;--979
```

Стовпчик ‘**old_price**’ містить також ціну товару але в трохи різних форматах: грошовому (починається з назви грошової одиниці SR, тисячі відділені комою); форматі, який також має позначення грошової одиниці, але через риску дробу вказується, ймовірно, ще кількість одиниць товару (10 таких записів). Всі комірки стовпчика заповнені, але стовпчик також містить комірки текстового формату (з текстом ‘No old price’), що, ймовірно може означати відсутність ціни. Таких записів 3040.

```
#запит підрахунку кількості записів поля old_price, які містять текст
#'No old price'
SELECT COUNT(*)
FROM all_data
WHERE old_price='No old price'   -- 3040
```

Стовпчик **‘sellable_online’** містить інформацію про продаж товару онлайн, логічний тип в числовому форматі (0 або 1). 28 записів містять значення 0. Пропущених значень немає.

```
#запит підрахунку кількості записів поля sellable_online,
#які містять ‘0’
SELECT count(*)
FROM all_data
WHERE sellable_online='0';--28
```

Стовпчик **‘link’** містить інформацію з посиланнями на інтернет сторінки товарів. Містить повтори. 2962 унікальних посилань. Пропущених значень немає.

```
#запит підрахунку кількості унікальних записів поля link
SELECT count(DISTINCT link)
FROM all_data;--2962
```

Стовпчик **‘other_colors’** містить інформацію про наявність товару іншого кольору, логічний тип в текстовому форматі (yes або no). 1512 записів містять значення ‘Yes’. Пропущених значень немає

```
#запит підрахунку кількості записів поля other_colors,
#які дорівнюють 'Yes'
SELECT count(*)
FROM all_data
WHERE other_colors='Yes';--1512
```

Стовпчик ‘**short_description**’ містить короткий опис товару у текстовому форматі. Є повтори, 1706 унікальних записів. Пропусків нема.

```
#запит підрахунку кількості унікальних записів поля short_description
SELECT count(DISTINCT short_description)
FROM all_data;--1706
```

Стовпчик ‘**designer**’ містить імена та назви дизайнерів в текстовому форматі, є повтори, об’єднання декількох назв. 381 унікальний запис. Пропусків нема.

```
#запит підрахунку кількості унікальних записів поля designer
SELECT count(DISTINCT designer)
FROM all_data;--381
```

Але стовпчик містить також записи з цифрами та текстом, які, ймовірно, випадкові і не мають відношення до поля ‘**designer**’, таких записів 143

```
#запит підрахунку кількості записів поля designer, які є випадковими
SELECT count(*)
FROM all_data--381
WHERE designer like '%0%' OR designer like '%9%' -143
```

Тому кількість унікальних записів без цих випадкових буде дорівнювати- 331.

```
#запит підрахунку кількості унікальних випадковими записів поля designer
SELECT count(DISTINCT designer)
FROM all_data
WHERE designer not like '%0%' OR designer not like '%9%'—331
```

Стовпчик **‘depth’** містить значення розміру товару, а саме його довжину, в числовому форматі (у вигляді дійсного числа через крапку, одне число після крапки). Є пропуски, 2231 не пусте значення.

Стовпчик **‘height’** містить значення розміру товару, а саме його висоту, в числовому форматі (у вигляді дійсного числа через крапку, одне число після крапки). Є пропуски, 2706 не пусте значення.

Стовпчик **‘width’** містить значення розміру товару, а саме його ширину, в числовому форматі (у вигляді дійсного числа через крапку, одне число після крапки). Є пропуски, 3105 не пусте значення.

Висновок з попереднього перегляду даних. Перед подальшим статистичним дослідженням оцінимо якість набору даних за допомогою Data Quality Framework.

Data Quality Criteria	*Data Quality Metrics*	*Data Quality Score*
Accuracy	Error rate	0.3%
Completeness	Field completeness	12.96%
Consistency	Field consistency	wasn’t found
Validity	Format validation	0.03%
Uniqueness	Duplication rate	19.82%

Error rate: відсоток записів з неправильними даними. Неправильні дані є у стовпчику ‘**designer**’ -143, тобто

$$143/(3694*13)*100\%=0.3\%$$

Field completeness: відсоток записів, які містять неповні дані для всіх полів. Такі дані містять стовпчики ‘**old_price**’- 3040, ‘**designer**’-143 (ті, що з помилкою), **‘depth’** – 1463, **‘height’** – 988, **‘width’** – 589, тобто

(3040+143+1463+988+589)/(3694*13)*100%=12.96%

Field consistency: відсоток записів, які містять протирічливі, непослідовні дані для всіх полів. Такі дані не були виявлені.

Format validation: відсоток записів, які містять неочікуваний формат даних для всіх полів. Такі дані містить стовпчик ‘**old_price**’ – 10, тобто

10/(3694*13)*100%=0.03

Duplication rate: відсоток записів, які містять дублікати. Таблиця містить 732 дублікати:

732/3694*100%=19.82%

У подальшому продовжимо процес очищення, стандартизації та трансформації даних з метою їх аналізу.

2. Статистичний аналіз даних

Для подальшого аналізу даних за допомогою Python. Перетворимо дані наступним чином.

Приберемо дані, що повторюються. Для цього створимо нову таблицю **all_data_without_duplicate,** також в цю таблицю не будемо включати стовпчики ‘**sellable_online’, ‘link’, ‘other_colors’,** ‘**short_description**’ оскільки подальший їх аналіз не вважаю доцільним для даного набору даних.

```
#запит створення таблиці all_data_without_duplicate без дублікатів
CREATE TABLE all_data_without_duplicate AS
   SELECT `Unnamed: 0`, item_id, name, category, price, old_price, other_colors, designer,
depth, height, width
     FROM  ( SELECT *, ROW_NUMBER() OVER(PARTITION BY item_id, name,
                price) AS RowNum
                      FROM all_data)q
 WHERE q.RowNum == 1
```

Приберемо зайві символи з стовпчика **‘old_price’**, перетворивши значення до числового вигляду

```
#запит зміни даних поля old_price
UPDATE all_data_without_duplicate
SET old_price = REPLACE(old_price, ',', '');

UPDATE all_data_without_duplicate
SET old_price = REPLACE(old_price, 'pack', '');

UPDATE all_data_without_duplicate
SET old_price = REPLACE(old_price, 'SR', '')
```

І для зручності замінимо значення **'No old price'** на `0` у цьому ж стовпчику.

```
#запит зміни даних поля old_price
UPDATE all_data_without_duplicate
SET old_price ='0'
WHERE old_price == 'No old price'
```

Приберемо випадкові записи зі стовпчика ‘**designer**’:

```
#запит зміни даних поля designer
UPDATE all_data_without_duplicate
SET designer=''
WHERE designer like '%0%' OR designer like '%9%'
```

1. Завантажимо дані стовпчика `**category**` до python як дата фрейм **exer1** та знайдемо їх статистичні характеристики.

```
# запит вибору колонки category
cursor.execute("""SELECT category
          FROM all_data_without_duplicate""")
exer1= cursor.fetchall()
exer1=pd.DataFrame(exer1,columns=['category'])
# описові статистики множини category
print(exer1.describe())
```

	category
count	**2962**
unique	**17**
top	**Bookcases & shelving units**
freq	**548**

Модою даних відповідно результатів є категорія ‘Bookcases & shelving units’, яка зустрічається 548 разів, для того, щоб детальніше розглянути весь розподіл побудуємо його гістограму (Рис.1)

Bookcases & shelving units	**548**
Chairs	**438**
Sofas & armchairs	**380**
Tables & desks	**370**
Wardrobes	**220**
Beds	**208**
Outdoor furniture	**197**
Cabinets & cupboards	**187**
Chests of drawers & drawer units	**111**
TV & media furniture	**89**
Children's furniture	**84**
Bar furniture	**47**
Trolleys	**23**
Nursery furniture	**22**
Café furniture	**18**
Sideboards, buffets & console tables	**10**
Room dividers	**10**

```
# кількість товарів в кожній category
count_category =
exer1['category'].value_counts()
# сортування за спаданням
count_category =
count_category.sort_values(ascending=False)

#побудова гістограми category
fig,ax=plt.subplots(figsize=(6,6))
sns.barplot(x=count_category.values,
            y=count_category.index)
ax.set_ylabel('category')
ax.set_xlabel('Count')
ax.set_title('the distribution histogram')
```

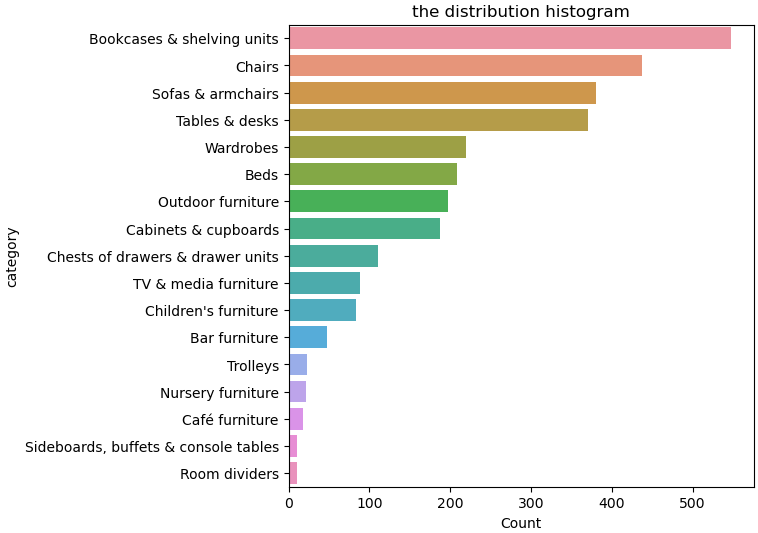

Рис.1 Гістограма розподілу змінної ‘category’

Гістограма на рис. 1 показує, що кількості товарів за категоріями розподілені нерівномірно. Якщо впорядкувати дані за спаданням кількості товарів в кожній категорії, то ще 3 категорії, крім моди мають достатньо великі значення (‘Chairs’ - 438, ‘Sofas & armchairs’- 380, ‘Tables & desks’- 370), а найменше значення мають категорії: ‘Room dividers’, ‘Sideboards, buffets & console tables’ -10, що може говорити про певні пріоритети у виборі товарів для продажу або про загальну варіативність певних категорій товарів.

2. Завантажимо дані стовпчика `**price**` до python як дата фрейм **exer2** та знайдемо їх статистичні характеристики.

```
# запит вибору колонки price
cursor.execute("""SELECT price
        FROM all_data_without_duplicate""")
exer2= cursor.fetchall()
```

	price
count	**2962.00**
mean	**1108.72**
std	**1393.58**

```
exer2=pd.DataFrame(exer2, columns=['price'])
exer2['price'] = exer2['price'].astype(float)
# описові статистики множини price
print(exer2.describe().round(2))

# знайдемо mode для price
from statistics import mode

mode = mode(exer2['price'])
print(mode)
```

min 3.00
25% 200.00
50% 570.00
75% 1475.00
max 9585.00

Mode 395.0

Побудуємо гістограму (рис. 2) та щільність розподілу **‘price’** (рис. 3).

Графік гістограми (рис. 2) показує, що дані мають асиметричний розподіл, з більшістю значень, які знаходяться в діапазоні від 0 до 500. Графік щільності розподілу (рис. 3) також показує, що дані мають одну вершину, моду, яка дорівнює 395. Наявність асиметрії також підтверджується значною різницею між значеннями моди (395), медіани(570) та середньої (1108.72), це може також говорити про наявність викидів. З іншого боку, значний розмах даних (9585-3=9582) може також означати, що дані сильно розсіяні навколо свого середнього значення, про це свідчить і середньо квадратичне відхилення (1393.58), яке більше за середню. Це потребує додаткового вивчення. Побудуємо графік **boxplot** (рис. 4) для дослідження даних на викиди. З графіку (рис.4) можна побачити значну кількість викидів, що говорить про необхідність додаткового дослідження цих даним, наприклад, дослідження ціни (**price**) за категоріями (**category**) та за дизайнерами (**designer**).

```
fig,ax=plt.subplots()
ax.hist(exer2['price'], bins=20)
ax.set_xlabel('Price')
ax.set_ylabel('Count')
ax.set_title('the distribution
histogram')
```

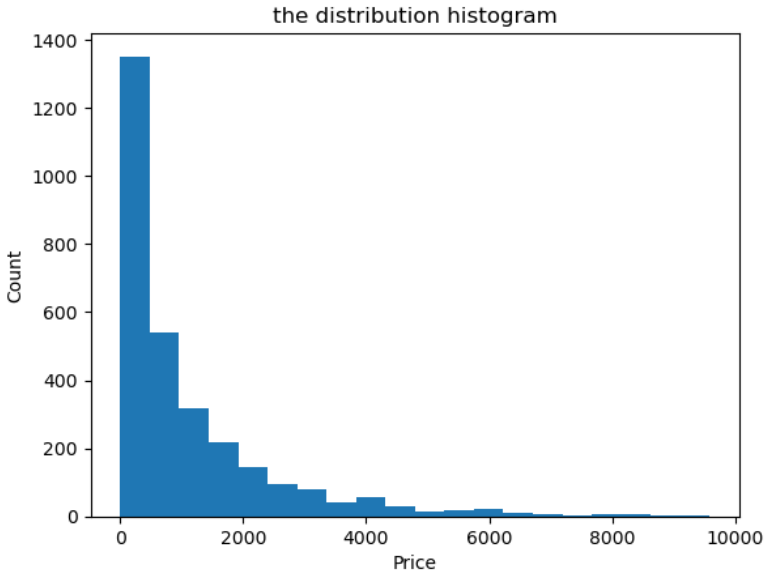

Рис. 2. Гістограма розподілу змінної ‘price’

```
fig,ax=plt.subplots()
sns.kdeplot(exer2['price'])
ax.set_title('the distribution
density')
plt.show()
```

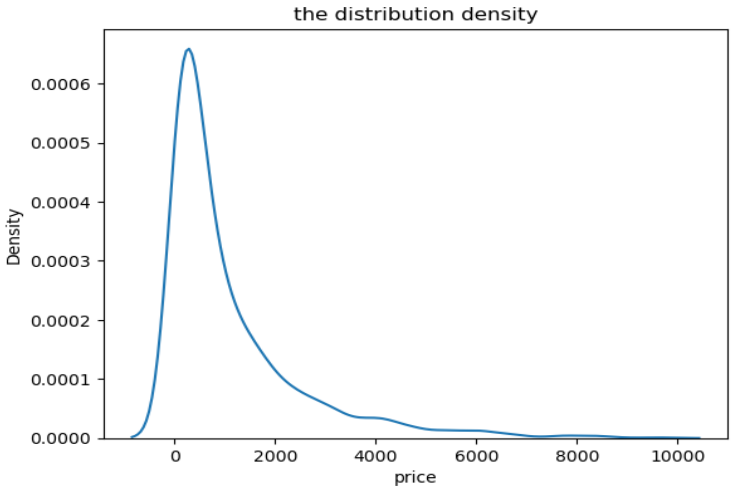

Рис. 3. Графік щільності розподілу змінної ‘price’

```
#побудова boxplot price
fig,ax=plt.subplots()
sns.boxplot(y ='price', data=exer2)
ax.set_title('the boxplot of the distribution')
plt.show()
```

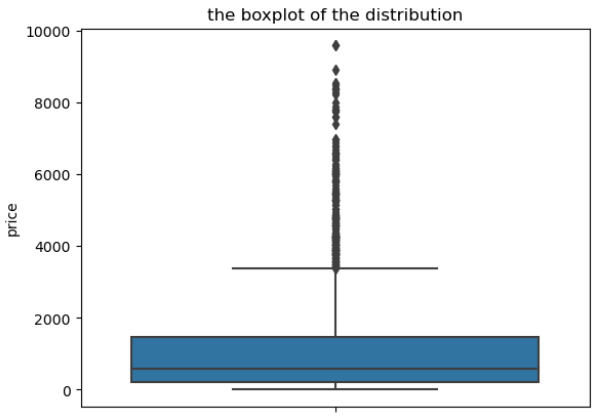

Рис. 4. Графік boxplot розподілу змінної ‘price’

3. Завантажимо дані стовпчика `**designer**` до python як дата фрейм **exer1_1** та знайдемо їх статистичні характеристики.

```
# запит вибору колонки designer
cursor.execute("""SELECT designer
            FROM all_data_without_duplicate
            WHERE designer!="")
exer1_1= cursor.fetchall()
exer1_1=pd.DataFrame(exer1_1,columns=['designer'])
# описові статистики множини category
print(exer1_1.describe())
```

	designer
count	2860
unique	279
top	IKEA of Sweden
freq	683

IKEA of Sweden	683
Ehlén Johansson	136
Francis Cayouette	131
Ola Wihlborg	128
Jon Karlsson	106
...	
E Thomasson/P Süssmann	1
Ehlén Johansson/K Hagberg/M Hagberg/IKEA of Sweden	1
Ola Wihlborg/Ehlén Johansson/IKEA of Sweden	1
Mia Lagerman/IKEA of Sweden/Wiebke Braasch	1
K Hagberg/M Hagberg/Francis Cayouette	1

Name: designer, Length: 279, dtype: int64

Модою даних з великим відривом відповідно результатів є дизайнер ‘IKEA of Sweden’, який зустрічається 683 разів, але при цьому є достатня кількість дизайнерів (113), які зустрічаються в таблиці лише раз.

```
# кількість товарів за designer
count_designer = exer1_1['designer'].value_counts()
min_value = count_designer.min()

# кількість desiner, які зустрічаються лише раз
num_min_designers = (count_designer == min_value).sum()
print(num_min_designers)
```

113

Побудуємо гістограму розподілу, але для дизайнерів які зустрічаються в даних більше 10 разів (рис.5).

Гістограма (рис. 5) показує, що кількість товарів за дизайнерами розподілена достатньо монотонно починаючи з другого по кількості товарів дизайнера (‘Ehlén Johansson’ - 136).

```
count_designer =
count_designer[count_designer.values>10].sort_values(
ascending=False)

#побудова гістограму designer
fig,ax=plt.subplots(figsize=(5,10))
sns.barplot(x=count_designer.values,
y=count_designer.index)
ax.set_ylabel('Designer')
ax.set_xlabel('Count')
ax.set_title('the distribution histogram')
plt.show()
```

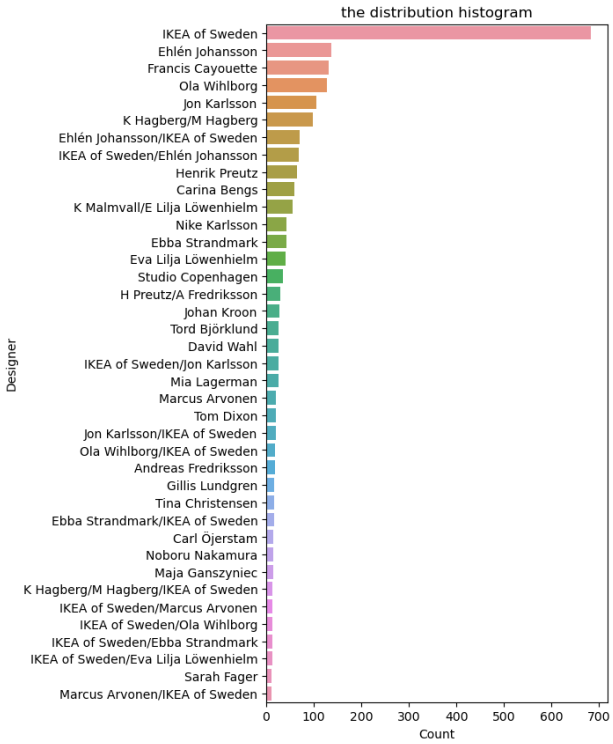

Рис. 5. Гістограма розподілу змінної ‘designer’

4. Завантажимо не нульові дані стовпчика `**old_price**` до python як дата фрейм **exer2_1** та знайдемо їх статистичні характеристики.

```
# запит вибору колонки old_price
cursor.execute("""SELECT old_price
            FROM all_data_without_duplicate
            WHERE old_price!='0'""")
exer2_1= cursor.fetchall()
exer2_1=pd.DataFrame(exer2_1,columns=['old_price'])
exer2_1['old_price'] = exer2_1['old_price'].astype(float)
# описові статистики множини old_price
print(exer2_1.describe().round(2))
```

	old_price
count	**574.00**
mean	**1633.06**
std	**1814.51**
min	**2.50**
25%	**400.00**
50%	**995.00**
75%	**2172.50**
max	**9985.00**

```
# знайдемо mode для price
from statistics import mode

mode = mode(exer2_1['old_price'])
print('mode ',mode)
```

mode 595.0

Побудуємо гістограму (рис. 6) та щільність розподілу (рис. 7) ‘**old_price’.**

Графік гістограми (рис. 6) показує, що дані мають асиметричний розподіл, з більшістю значень, які знаходяться в діапазоні від 0 до 400. Немонотонність (численні піки) може бути ілюстрацією неповноти даних. Графік щільності розподілу (рис. 7) також показує, що дані мають одну вершину, моду, яка дорівнює 595. Наявність асиметрії також підтверджується значною різницею між значеннями моди (595), медіани (995) та середньої (1633.06), це може також говорити про наявність викидів. З іншого боку, значний розмах даних (9985-2,5=9982,5) може також означати, що дані сильно розсіяні навколо свого середнього значення, про це свідчить і середньо квадратичне відхилення (1814.51), яке більше за середню. Це потребує додаткового вивчення.

```
# побудова гістограми розподілу
old_price
fig,ax=plt.subplots()
ax.hist(exer2_1['old_price'],
bins=30)
ax.set_xlabel('Old_price')
ax.set_ylabel('Count')
ax.set_title('the distribution
histogram')
```

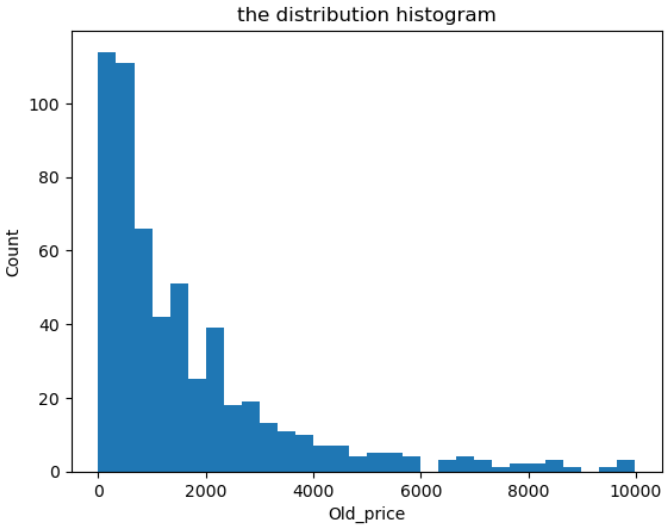

Рис. 6. Гістограма розподілу змінної ‘old_price’

```
# графік щільності old_price
fig,ax=plt.subplots()
sns.kdeplot(exer2_1['old_price'])
ax.set_title('the distribution
density')
plt.show()
```

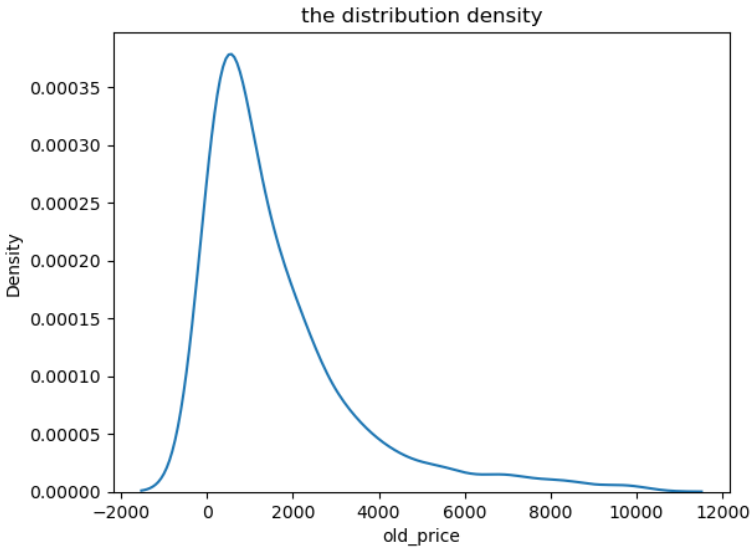

Рис. 7. Графік щільності розподілу змінної ‘old_price’

Побудуємо графік **boxplot** (рис.8) для дослідження даних на викиди. З графіку можна побачити значну кількість викидів.

Візуальна схожість розподілів даних ‘**price**’ та ‘**old_price**’ може говорити про їх залежність, що потребує додаткового дослідження.

Далі подивимося на існуючі дані у взаємодії.

5. Розглянемо розподіл даних **‘price’** за **‘category’**. Графік boxplot (рис. 9) для кожної категорії виглядає по-іншому, це може означати, що фактор **‘category’** має вплив на змінну **‘price’**.

```
#побудова boxplot old_price
fig,ax=plt.subplots()
sns.boxplot(y ='old_price', data=exer2_1)
ax.set_title('the boxplot of the distribution')
plt.show()
```

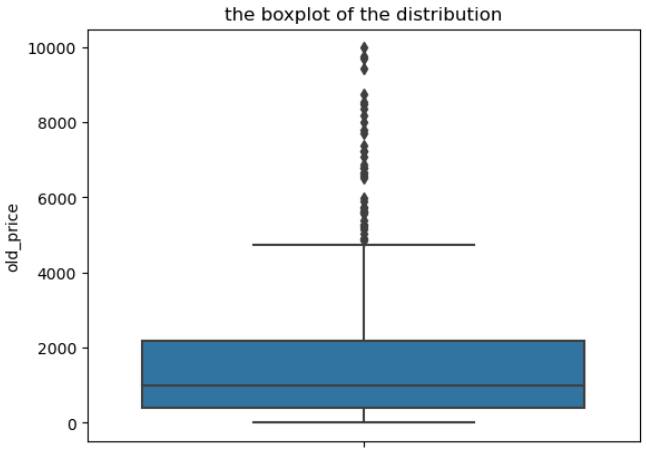

Рис. 8. Графік boxplot розподілу змінної ‘old_price’

У такому випадку, можна припустити, що ці категорії різняться за середнім значенням змінної **‘price’** і, отже, ця якісна змінна може бути значущим фактором у визначенні ціноутворення. Але оскільки кількість товарів по категоріям розподілена не рівномірно та є значна кількість викидів, це припущення потребує проведення додаткових статистичних тестів.

```
# запит вибору колонок category, price
cursor.execute("""SELECT category, price
            FROM all_data_without_duplicate""")
exer3= cursor.fetchall()
exer3=pd.DataFrame(exer3,columns=['category','pric
)
exer3['price'] = exer3['price'].astype(float)

#побудова boxplot price за category
plt.subplots(figsize=(15,6))
sns.boxplot(x='category',y ='price', data=exer3)
plt.xticks(rotation=90)
plt.show()
```

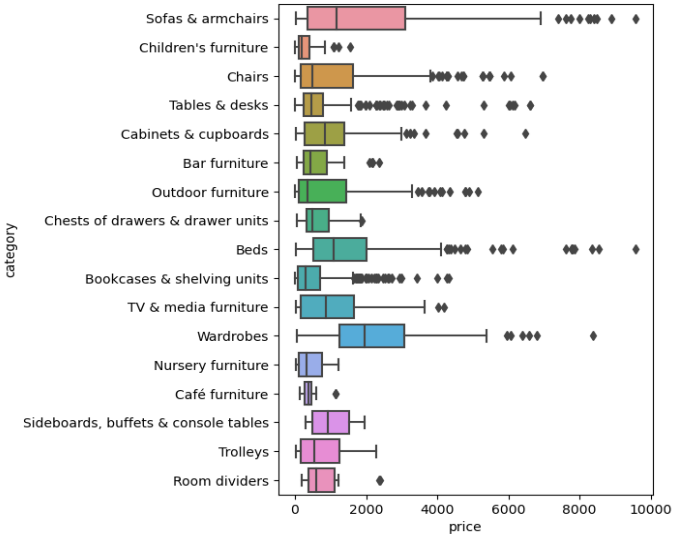

Рис.9 Графік boxplot розподілу ціни за категоріями

```
print(exer3.groupby(['category'])['price'].mean().
round(2)
```

Category	**mean**
Bar furniture	**679.55**
Beds	**1647.43**
Bookcases & shelving units	**519.42**
Cabinets & cupboards	**1044.82**

Café furniture	**426.72**
Chairs	**1097.12**
Chests of drawers & drawer units	**657.49**
Children's furniture	**286.18**
Nursery furniture	**431.77**
Outdoor furniture	**919.76**
Room dividers	**912.60**
Sideboards, buffets & console tables	**1013.00**
Sofas & armchairs	**1968.16**
TV & media furniture	**1045.65**
Tables & desks	**760.13**
Trolleys	**748.87**
Wardrobes	**2249.02**

6. Розглянемо дані стовпчика `**designer**`, після вилучення дублікатів та очищення він містить 102 пусті комірки. Давайте поглянемо, в які саме категорії потрапили пусті значення стовпчика `**designer**`.

```
# запит вибору колонок category,
COUNT_empty_value,
де пусті значеня designer
cursor.execute("""SELECT category,
COUNT(designer)
AS COUNT_empty_value
            FROM all_data_without_duplicate
            WHERE designer=''
            GROUP BY category
            ORDER BY COUNT_empty_value
DESC""")
exer6= cursor.fetchall()
exer6=pd.DataFrame(exer6,columns=['category',
'COUNT_empty_value'])
```

	category	COUNT_empty_value
0	**Bookcases & shelving units**	**26**
1	**Chairs**	**23**
2	**Sofas & armchairs**	**20**
3	**Tables & desks**	**14**
4	**Cabinets & cupboards**	**5**
5	**Beds**	**5**
6	**Children's furniture**	**4**
7	**TV & media furniture**	**3**
8	**Outdoor furniture**	**1**
9	**Chests of drawers & drawer units**	**1**

```
# кількість пустих комірок
print(exer6['COUNT_empty_value'].sum())
```

102

Тепер розглянемо не пусті значення за отриманими категоріями і виберемо по 3 дизайнера в кожній категорії, які найбільше представляють товари таблиці. Та побудуємо графіки розсіювання дизайнерів за вибраними категоріями (рис. 10-11).

```
# запит вибору колонок category, designer,
за категоріями, які містять пусті значення
стовпчика designer
cursor.execute("""SELECT category, designer, coun
            FROM designer_for_empty_3""")
exer6_1= cursor.fetchall()
exer6_1=pd.DataFrame(exer6_1,columns=
['category','designer','coun'])

# графік розподілу кількості дизайнерів за
категоріями
sns.scatterplot(x='count',y ='category',
hue='designer',data=exer6_1, s=125)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

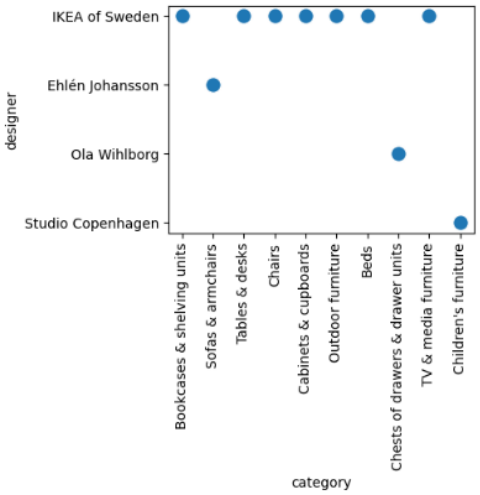

Рис.10 Графік розсіювання дизайнерів, які найчастіше зустрічаються за 9 категоріями

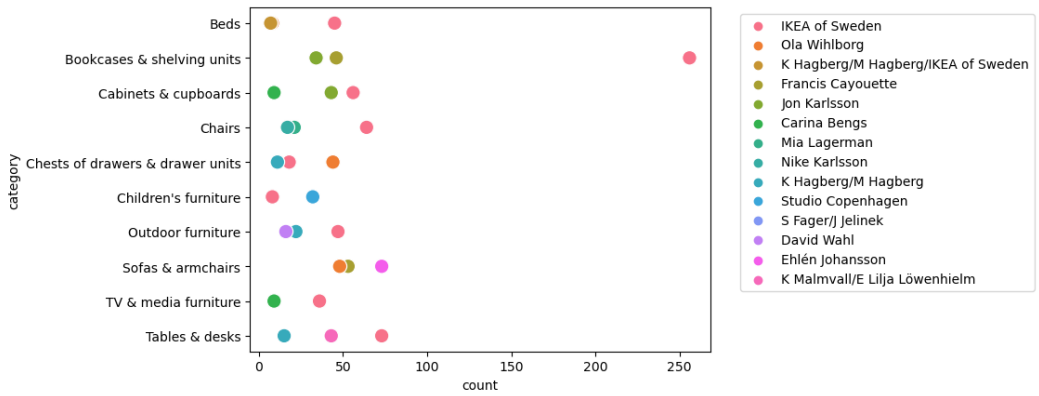

Рис.11 Графік розсіювання 3 перших дизайнерів, які найчастіше зустрічаються за 9 категоріями

З графіку (рис. 11) видно, що лідерами по кількості товарів є всього 4 дизайнера і найпершим з них є IKEA of Sweden, це можна чітко побачити на рис. 10, графіку розподілу дизайнера за категоріями з максимальним представленням у даних.

Даний результат можна використати, наприклад для заповнення пропусків даних стовпчика `**designer**`.

7. Розглянемо розподіл даних **‘price’** за **‘designer’**. Графік boxplot (рис. 12) для кожного дизайнера виглядає по-іншому, це може означати, що фактор **‘designer’** має вплив на змінну **‘price’**.

У такому випадку, можна припустити, що дизайнери різняться за середнім значенням змінної **‘price’** і, отже, ця якісна змінна може бути значущим фактором у визначенні ціноутворення. Але оскільки кількість товарів за дизайнерами розподілена не рівномірно та є значна кількість викидів, це припущення потребує проведення додаткових статистичних тестів.

```
# запит вибору колонок category,designer, price
cursor.execute("""SELECT category, designer,
price
        FROM all_data_without_duplicate
        WHERE designer!="""")
exer7= cursor.fetchall()
exer7=pd.DataFrame(exer7,columns=
['category','designer','price'])
exer7['price'] = exer7['price'].astype(float)
# опис колонки price за category
print(exer7.groupby(['designer'])['price'].mean().r
ound(2))


#побудова boxplot price за designer
plt.subplots(figsize=(5,50))
sns.boxplot(x='price',y ='designer', data=exer7)
plt.show()
```

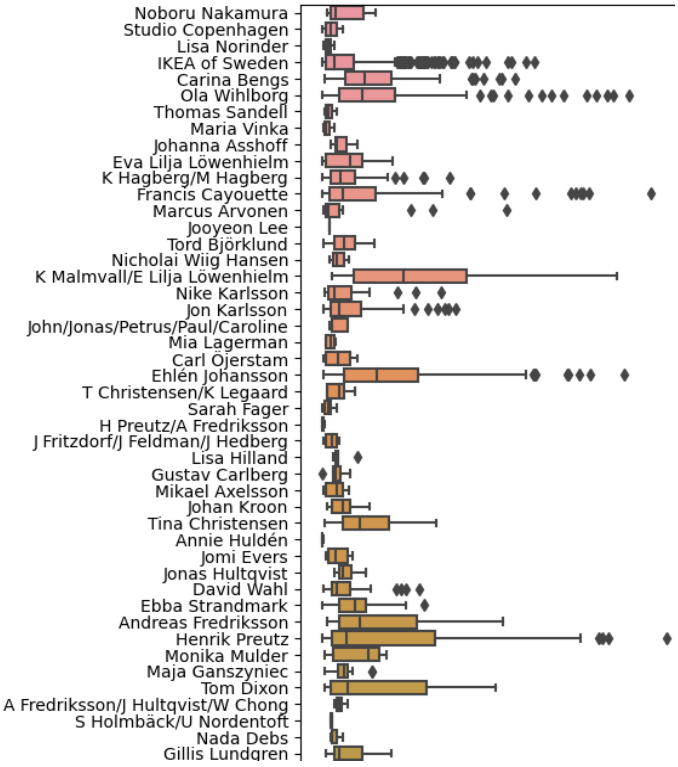

Рис.12 Графік boxplot розподілу ціни за дизайнерами

8. Розглянемо дві номінальні змінні **‘price’** та **‘old_price’** подивимося чи є між ними взаємозв’язок для цього побудуємо графік scatterplot (рис. 13).

Графік (рис.13) показує, що між змінними **‘price’** та **‘old_price’** може бути лінійна залежність. Для того, щоб переконатися в цьому знайдемо коефіцієнт кореляції між цими змінними.

```
corr = exer5_1.corr()
corr
```

	price	old_price
price	1.000000	0.993705
old_price	0.993705	1.000000

Коефіцієнт кореляції близький до 1, що дає підстави стверджувати, що між даними є лінійна залежність. Це може бути корисним, наприклад для відновлення пропущених значень в стовпчику **‘old_price’** з допомогою даних **‘price’.**

```
# запит вибору колонки category,price, old_price
cursor.execute("""SELECT item_id,category, price, old_price
            FROM all_data_without_duplicate""")
exer5= cursor.fetchall()
exer5=pandas.DataFrame(exer5,columns=['item_id','category','price','old_price'])
exer5['old_price'] = exer5['old_price'].astype(float)
exer5['price'] = exer5['price'].astype(float)

exer5_1=exer5[exer5['old_price']!=0]
#побудова scatterplot price за old_price
sns.scatterplot(x='price',y ='old_price', data=exer5_1)
plt.show()
```

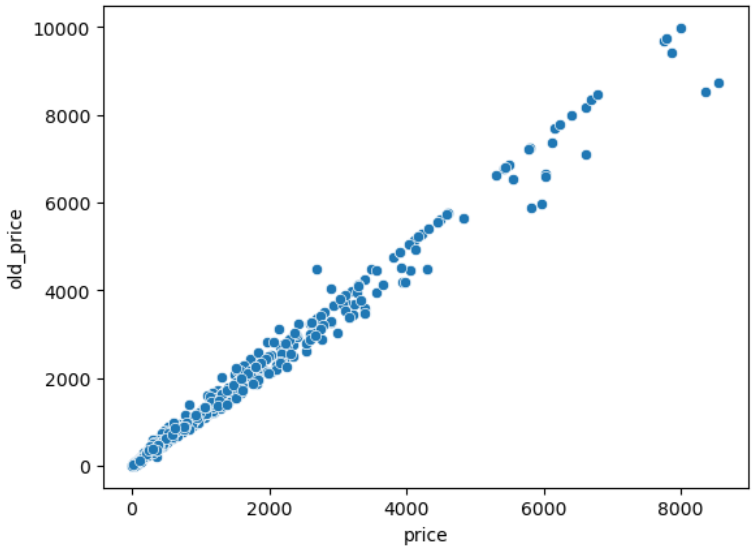

Рис.13 Графік розсіювання price та old_price

9. Розглянемо залежність таких стовпчиків, як: **‘category’**, **‘price’**, **‘depth’**, **‘height’**, **‘width’**. Оскільки дані стовпчиків **‘depth’**, **‘height’**, **‘width’** містять пусті значення, побудуємо розподіли окремо для кожного з цих стовпчиків за даними стовпика **‘category’** (рис.14, 15, 16).

```
# запит вибору колонок category, depth, height, width,price
cursor.execute("""SELECT category, price, depth, height, width
            FROM all_data_without_duplicate""")
exer3_1= cursor.fetchall()
exer3_1=pd.DataFrame(exer3_1,columns=['category','price','depth', 'height', 'width'])
# print('ТОП 10 експортерів кави з набору даних:')
print(exer3_1)

exer3_1['depth'] = exer3_1['depth'].fillna(value=numpy.nan).astype(float)
exer3_1['height'] = exer3_1['height'].fillna(value=numpy.nan).astype(float)
exer3_1['width'] = exer3_1['width'].fillna(value=numpy.nan).astype(float)
exer3_1['price'] = exer3_1['price'].astype(float)
```

```
#побудова boxplot depth за category
plt.subplots(figsize=(15,6))
sns.boxplot(x='category',y ='depth', data=exer3_1)
plt.xticks(rotation=90)
plt.show()
```

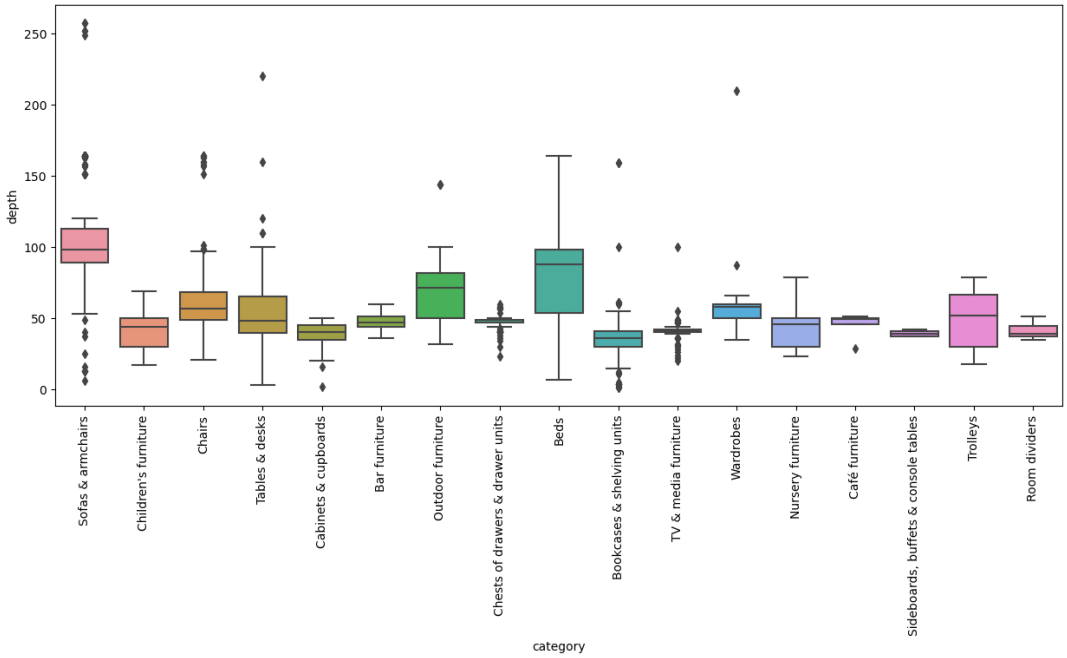

Рис. 14. Графік boxplot розподілу depth за category

```
#побудова boxplot height за category
plt.subplots(figsize=(15,6))
sns.boxplot(x='category',y ='height', data=exer3_1)
plt.xticks(rotation=90)
plt.show()
```

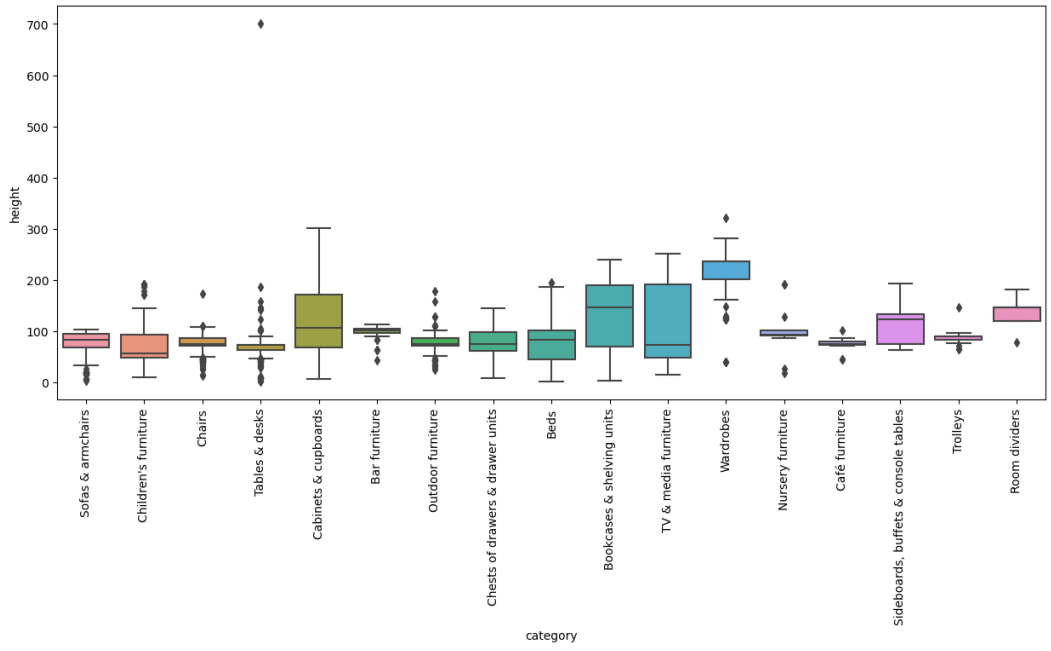

Рис. 15. Графік boxplot розподілу height за category

```
#побудова boxplot width за category
plt.subplots(figsize=(15,6))
sns.boxplot(x='category',y ='width', data=exer3_1)
plt.xticks(rotation=90)
plt.show()
```

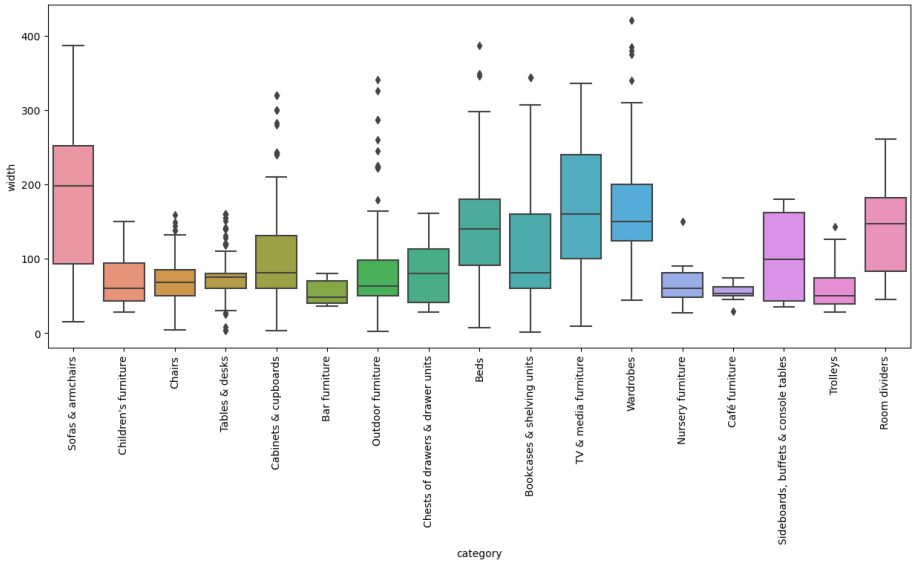

Рис. 16. Графік boxplot розподілу width за category

Графіки Boxplot **‘depth’**, **‘height’**, **‘width’** для кожної категорії виглядають по-іншому, це може означати, що фактор **‘category’** має вплив на змінні **‘depth’**, **‘height’**, **‘width’**.

У такому випадку, можна припустити, що категорії різняться за середнім значенням змінних **‘depth’**, **‘height’**, **‘width’** і, отже, ці якісна змінна може бути значущим фактором для розпізнавання розмірів товарів. Перевіримо це.

Знайдемо середні значення для непустих комірок по стовпчиках **‘depth’**, **‘height’**, **‘width’, ‘price’,** за категоріями, потім розрахуємо за цими даними середні об’єми **‘volume’**. Побудуємо barplot (рис. 17) поля **‘volume’** за категоріями.

```
# знаходження середніх
avg_parametr = exer3_1.groupby('category').agg(mean_depth=('depth', lambda x: x.mean(skipna=True)),
                                               mean_height=('height', lambda x: x.mean(skipna=True)),
                                               mean_width=('width', lambda x: x.mean(skipna=True)))
#знаходження об'єму
avg_parametr['volume'] = avg_parametr.apply(lambda row: row.mean_depth * row.mean_height * row.mean_width, axis=1)
avg_parametr['volume'] = avg_parametr.groupby('category')['volume'].prod()
avg_parametr = avg_parametr.sort_values(by='volume', ascending=False)

#побудова barplot volume за category
plt.subplots(figsize=(15,6))
sns.barplot(x=avg_parametr.index, y=avg_parametr["volume"])
plt.xticks(rotation=90)
plt.xlabel("Category")
plt.ylabel("Volume")
plt.show()
```

Даний графік (рис. 17) підтверджує, що середні об’ємів різняться для різних категорій. Цей вплив можна, наприклад використати для заповнення пропусків в даних стовпчиків **‘depth’**, **‘height’**, **‘width’.** Але оскільки кількість товарів за категоріями розподілена не рівномірно та є значна кількість викидів, це припущення потребує проведення додаткових статистичних тестів.

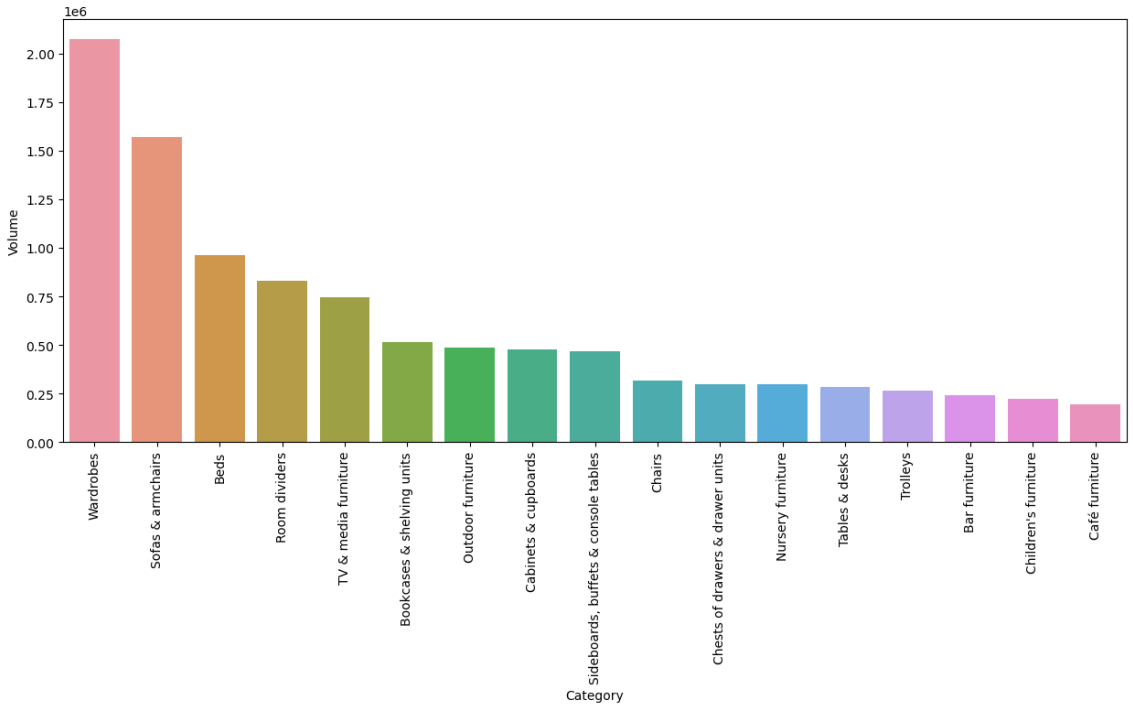

Рис. 17. Графік barplot розподілу середнього об’єму за category

10. Також, використовуючи попередні розрахунки ми можемо подивитися чи є кореляція між ціною товару та його розміром. Для цього побудуємо графік розсіювання між даними середньої ціни на середнього об’єму за категоріями товарів (рис. 18). Та обчислимо коефіцієнт кореляції для цих змінних.

```
#знаходження середньої ціни за категоріями
avg_parametr['price'] = exer3_1.groupby('category')['price'].mean()
#побудова scatterplot price та volume
sns.scatterplot(x='price',y ='volume', hue='category', data=avg_parametr)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
#знаходження коефіцієнта кореляції
corr = avg_parametr[['price', 'volume']].corr().iloc[0, 1]
```

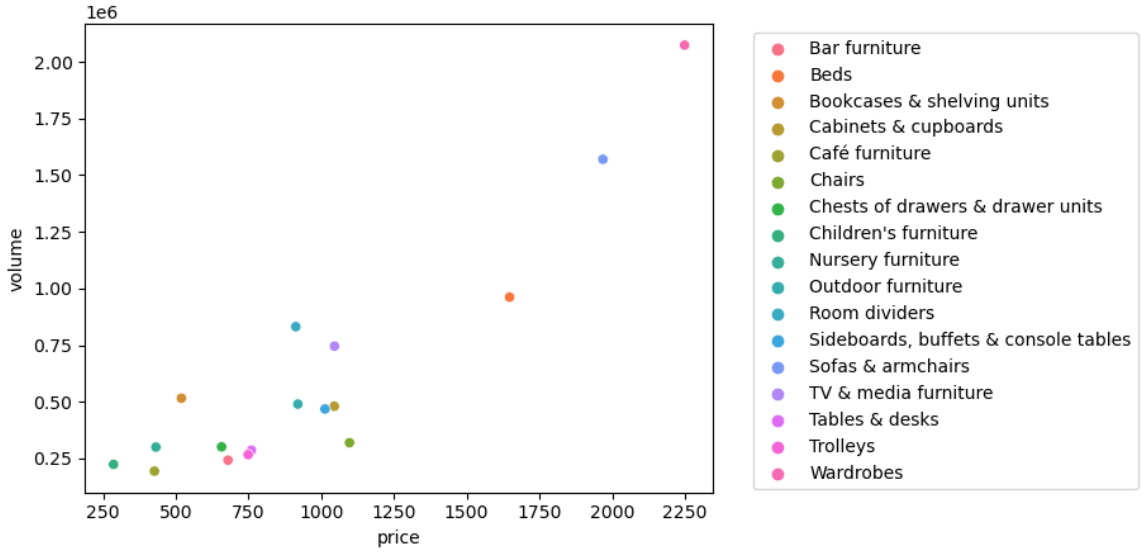

Рис. 18. Графік розсіювання середньої ціни та об'єму за category

На графіку розсіювання можна спостерігати лінійну залежність, а коефіцієнт кореляції (0.9116683492815336) прямує до 1, що може говорити про пряму лінійну залежність між ціноутворенням товару і його розміром, що можна використати для передбачення ціни.

Висновок зі статистичного аналізу даних. Нормативні дані стовпчиків **'price', 'old_price', 'depth', 'height', 'width'** не розподілені ні нормально, ні рівномірно, містять значну кількість викидів. Відповідно попередніх статистичних досліджень дані стовпчиків **'price', 'old_price'** мають лінійну залежність, також ціна лінійно залежна від розмірів товару. Попередній розгляд розподілів нормативних даних за факторними даними **'category'** та **'designer'** дає можливість припускати, що є значний вплив цих факторів на ціноутворення, на яке також може впливати розмір товару.

У подальшому попрацюємо над відновленням відсутніх даних для того, щоб більш якісно зробити додаткові статистичні тести для перевірки наших припущень та здійснити передбачення.

3. Відновлення відсутніх даних

1. Природа наявності даних, які не відповідають числовим значенням у стовпику **‘old_price’** важко пояснити, а текст ‘No old price’ говорить про її невипадковість. Також значна кількість таких даних 3040 з 3694 не дозволяє просто замінити їх середніми.

Використовуючи виявлену лінійну залежність між даними стовпчиків **‘price’** та **‘old_price’** побудуємо модель лінійної регресії та застосуємо її для заповнення пропущених числових значень в даних стовпчика **‘old_price’**.

```
# запит вибору колонки category,price, old_price
cursor.execute("""SELECT item_id,category, price, old_price
                FROM all_data_without_duplicate""")
exer5= cursor.fetchall()
exer5=pd.DataFrame(exer5,columns=['item_id','category','price','old_price'])
exer5['old_price'] = exer5['old_price'].astype(float)
exer5['price'] = exer5['price'].astype(float)

#запам’ятовування окремо даних з 'old_price', які не дорівнюють і які дорівнюють 0
exer5_1=exer5[exer5['old_price']!=0]
exer5_2=exer5[exer5['old_price']==0]

#розподілення даних на тренувальні та тестові
X=exer5_1['price'].values.reshape(-1, 1)
y=exer5_1['old_price']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=42)

# побудова моделі лінійної регресії
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)

#тестування моделі
y_pred=model.predict(X_test)
```

```
# графічне зображення моделі

plt.scatter(X_train, y_train, color='blue',
label='Training Dat')
plt.scatter(X_test, y_test, color='red', label='Test
        Data')
# Line of best fit for the test data
plt.plot(X_test, y_pred, color='green',
        label='Regression Line')
plt.xlabel('price')
plt.ylabel('old_price')
plt.title('Scatter plot of price vs old_price')
plt.legend()
plt.show()
```

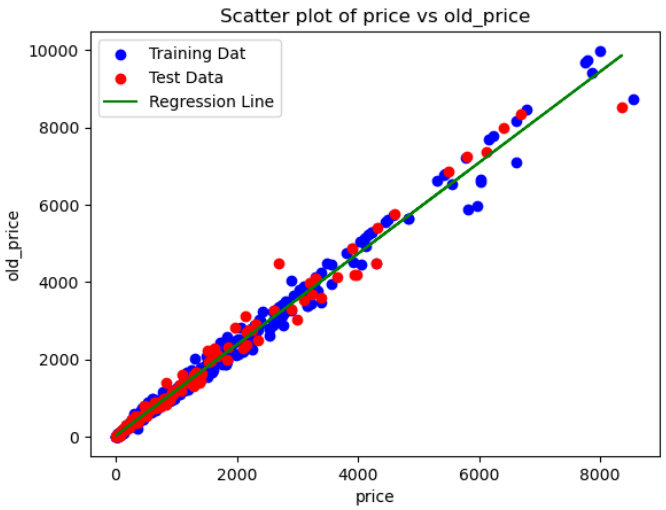

Рис. 19. Графік розсіювання, лінійної регресії price, old_price (навчання, тестування)

Перевіримо модель.

```
# перевірка моделі
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

0.98331845721843

Коефіцієнт детермінації (R-squared) близький до 1, це означає, що залежність між двома змінними може бути дійсно лінійною. Використаємо побудовану модель для заповнення даних, яких не вистачає шляхом передбачення. За аргумент функції передбачення візьмемо дані з відсутніми значеннями стовпчика **‘old_price’**

```
plot_x=exer5_2['price'].values.reshape(-1, 1)
plot_y=model.predict(plot_x)
```

Візуалізуємо результат.

```
plt.scatter(X, y, color='blue', label='Existing old_price Data')
plt.scatter(plot_x, plot_y, color='red', label='Non-existent old_price Data')

# Line of best fit for the test data
plt.plot(plot_x, plot_y, color='green', label='Regression Line')
plt.xlabel('price')
plt.ylabel('old_price')
plt.title('Scatter plot of price vs old_price')
plt.legend()
plt.show()
```

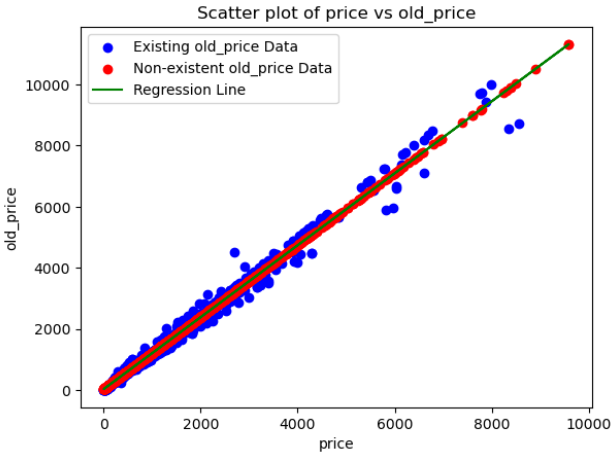

Рис. 20. Графік розсіювання, лінійної регресії price, old_price (передбачення)

Запишемо отримані значення даних **‘old_price’** назад до бази даних.

```
# запишимо передбачені значення до датафрейму exer5
exer5.loc[exer5['old_price'] == 0.0, 'old_price'] = plot_y.reshape(-1, 1)

# запишемо передбачені значення стовпчика old_price до таблиці NEW_data
conn = sqlite3.connect('sql_step_project.db')
conn.execute('BEGIN TRANSACTION')
for index, row in exer5.iterrows():
    item_id=row['item_id']
    category = row['category']
    old_price = row['old_price']
    conn.execute('UPDATE NEW_data SET old_price_predict = ? WHERE item_id = ? AND
 category = ?', (old_price, item_id, category))
conn.commit()
```

2. Причиною відсутності даних у стовпчику **‘designer’** є очищення даних, після їх помилкового заповнення, тобто має випадковий характер.

Для відновлення відсутніх даних використаємо попередній статистичний аналіз і заповнимо пропуски дизайнерами, які є найбільш представлені за кожною категорією (рис. 10).

```
# запит вибору стовпчика max кількість (designer який найчастіше зустрічається), також
 стовпчики category, designer
cursor.execute("""SELECT category, designer, max
            FROM designer_for_empty""")
exer6_2= cursor.fetchall()
exer6_2=pd.DataFrame(exer6_2,columns=['category','designer','max'])

# запит вибору колонок item_id, category , designer, де пусті
 значеня designer
cursor.execute("""SELECT item_id, category , designer
              FROM all_data_without_duplicate
              WHERE designer="
             """)
exer6_3= cursor.fetchall()
exer6_3=pd.DataFrame(exer6_3,columns=['item_id', 'category' , 'designer'])

# заповнення пропусків дизайнерами, які найчастіше зустрічаються у базі за кожною категорією
for index, row in exer6_3.iterrows():
   category = row['category']
   designer = exer6_2.loc[exer6_2['category'] == category, 'designer'].iloc[0]
   exer6_3.at[index, 'designer'] = designer

# запис заповнених значень стовпчика designer до таблиці NEW_data
conn = sqlite3.connect('sql_step_project.db')
conn.execute('BEGIN TRANSACTION')
for index, row in exer6_3.iterrows():
   item_id=row['item_id']
   category = row['category']
   designer = row['designer']
   conn.execute('UPDATE NEW_data SET designer_new = ? WHERE item_id = ? AND category =
?', (designer, item_id, category))
conn.commit()
```

3. Припускаємо, що пропуски в даних стовпчиків **‘depth’**, **‘height’**, **‘width’** мають випадковий характер. Заповнимо ці пропуски середніми або медіанами по кожній категорії. Середніми будемо заповнювати у випадку, коли розподіл за категорією виглядає хоча б приблизно нормальним і не містить викидів, а медіанами у протилежному випадку.

Для цього розглянемо графіки boxplot з попереднього статистичного аналізу для кожного стовпчика **‘depth’**, **‘height’**, **‘width’** (рис 14, 15, 16).

Стовпчик **‘depth’** відповідно до графіку boxplot (рис. 14)**:** заміняємо середніми в категоріях ‘Children's furniture’, ‘Bar furniture’, ‘Outdoor furniture’, ‘Trolleys’ пропуски у всіх інших категоріях замінимо медіанами.

```
# запит вибору колонок item_id, category, depth, де depth Null
cursor.execute("""SELECT item_id, category, depth
            FROM all_data_without_duplicate
            WHERE depth IS NULL""")
exer4= cursor.fetchall()
exer4=pd.DataFrame(exer4,columns=['item_id','category','depth'])
```

```
# заміна пустих значень середніми
exer4.loc[exer4['category']=="Children's furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="Children's furniture", 'mean_depth'].values[0]
exer4.loc[exer4['category']=="Bar furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="Bar furniture", 'mean_depth'].values[0]
```

```
exer4.loc[exer4['category']=="Outdoor furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="Outdoor furniture", 'mean_depth'].values[0]
exer4.loc[exer4['category']=="Trolleys", 'depth'] = avg_parametr.loc[avg_parametr.index=="Trolleys", 'mean_depth'].values[0]

                    # заміна пустих значень медіанами
exer4.loc[exer4['category']=="Sofas & armchairs", 'depth'] = avg_parametr.loc[avg_parametr.index=="Sofas & armchairs", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Cabinets & cupboards", 'depth'] = avg_parametr.loc[avg_parametr.index=="Cabinets & cupboards", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Chairs", 'depth'] = avg_parametr.loc[avg_parametr.index=="Chairs", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Tables & desks", 'depth'] = avg_parametr.loc[avg_parametr.index=="Tables & desks", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Chests of drawers & drawer units", 'depth'] = avg_parametr.loc[avg_parametr.index=="Chests of drawers & drawer units", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Beds", 'depth'] = avg_parametr.loc[avg_parametr.index=="Beds", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Bookcases & shelving units", 'depth'] = avg_parametr.loc[avg_parametr.index=="Bookcases & shelving units", 'median_depth'].values[0]
exer4.loc[exer4['category']=="TV & media furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="TV & media furniture", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Wardrobes", 'depth'] = avg_parametr.loc[avg_parametr.index=="Wardrobes", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Nursery furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="Nursery furniture", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Café furniture", 'depth'] = avg_parametr.loc[avg_parametr.index=="Café furniture", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Sideboards, buffets & console tables", 'depth'] = avg_parametr.loc[avg_parametr.index=="Sideboards, buffets & console tables", 'median_depth'].values[0]
exer4.loc[exer4['category']=="Room dividers", 'depth'] = avg_parametr.loc[avg_parametr.index=="Room dividers", 'median_depth'].values[0]
                    # запишемо заповнені значення стовпчика depth до таблиці NEW_data
                    conn = sqlite3.connect('sql_step_project.db')
                    conn.execute('BEGIN TRANSACTION')
                    for index, row in exer4.iterrows():
                        item_id=row['item_id']
                        category = row['category']
                        depth = row['depth']
                        conn.execute('UPDATE NEW_data SET depth_new = ? WHERE item_id = ? AND
                    category = ?', (depth, item_id, category))
                    conn.commit()
```

Стовпчик **‘height’** відповідно до графіку boxplot (рис.15)**:** заміняємо середніми в категоріях ‘Cabinets & cupboards’, ‘Chests of drawers & drawer units’, ‘Bookcases & shelving units’, пропуски у всіх інших категоріях замінимо медіанами.

```
                    # запит вибору колонок item_id, category, height, де height Null
                    cursor.execute("""SELECT item_id, category, height
                                FROM all_data_without_duplicate
                                WHERE height IS NULL""")
                    exer4_1= cursor.fetchall()
                    exer4_1=pd.DataFrame(exer4_1,columns=['item_id','category','height'])

                    # заміна пустих значень середніми
exer4_1.loc[exer4_1['category']=="Cabinets & cupboards", 'height'] = avg_parametr.loc[avg_parametr.index=="Cabinets & cupboards", 'mean_height'].values[0]
exer4_1.loc[exer4_1['category']=="Chests of drawers & drawer units", 'height'] = avg_parametr.loc[avg_parametr.index=="Chests of drawers & drawer units", 'mean_height'].values[0]
```

```
exer4_1.loc[exer4_1['category']=="Bookcases & shelving units", 'height'] = avg_parametr.loc[avg_parametr.index=="Bookcases & shelving units", 'mean_height'].values[0]

# заміна пустих значень медіанами
exer4_1.loc[exer4_1['category']=="Children's furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="Children's furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Bar furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="Bar furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Outdoor furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="Outdoor furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Trolleys", 'height'] = avg_parametr.loc[avg_parametr.index=="Trolleys", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Sofas & armchairs", 'height'] = avg_parametr.loc[avg_parametr.index=="Sofas & armchairs", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Chairs", 'height'] = avg_parametr.loc[avg_parametr.index=="Chairs", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Tables & desks", 'height'] = avg_parametr.loc[avg_parametr.index=="Tables & desks", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Beds", 'height'] = avg_parametr.loc[avg_parametr.index=="Beds", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="TV & media furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="TV & media furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Wardrobes", 'height'] = avg_parametr.loc[avg_parametr.index=="Wardrobes", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Nursery furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="Nursery furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Café furniture", 'height'] = avg_parametr.loc[avg_parametr.index=="Café furniture", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Sideboards, buffets & console tables", 'height'] = avg_parametr.loc[avg_parametr.index=="Sideboards, buffets & console tables", 'median_height'].values[0]
exer4_1.loc[exer4_1['category']=="Room dividers", 'height'] = avg_parametr.loc[avg_parametr.index=="Room dividers", 'median_height'].values[0]
```

```
# запишемо заповнені значення стовпчика height до таблиці NEW_data
conn = sqlite3.connect('sql_step_project.db')
conn.execute('BEGIN TRANSACTION')
for index, row in exer4_1.iterrows():
    item_id=row['item_id']
    category = row['category']
    height = row['height']
    conn.execute('UPDATE NEW_data SET height_new = ? WHERE item_id = ?
 AND category = ?', (height, item_id, category))
conn.commit()
```

Стовпчик **‘width’** відповідно до графіку boxplot (рис.16)**:** заміняємо середніми в категоріях ‘Beds’, ‘Chairs’, ‘TV & media furniture’, пропуски у всіх інших категоріях замінимо медіанами.

```
# запит вибору колонок item_id, category, width, де width Null
cursor.execute("""SELECT item_id, category, width
        FROM all_data_without_duplicate
        WHERE width IS NULL""")
exer4_2= cursor.fetchall()
exer4_2=pd.DataFrame(exer4_2,columns=['item_id','category','width'])

# заміна пустих значень середніми
exer4_2.loc[exer4_2['category']=="Beds", 'width'] = avg_parametr.loc[avg_parametr.index=="Beds", 'mean_width'].values[0]
exer4_2.loc[exer4_2['category']=="Chairs", 'width'] = avg_parametr.loc[avg_parametr.index=="Chairs", 'mean_width'].values[0]
```

```
exer4_2.loc[exer4_2['category']=="TV & media furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="TV & media furniture", 'mean_width'].values[0]

                    # заміна пустих значень медіанами
exer4_2.loc[exer4_2['category']=="Cabinets & cupboards", 'width'] = avg_parametr.loc[avg_parametr.index=="Cabinets & cupboards", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Chests of drawers & drawer units", 'width'] = avg_parametr.loc[avg_parametr.index=="Chests of drawers & drawer units", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Bookcases & shelving units", 'width'] = avg_parametr.loc[avg_parametr.index=="Bookcases & shelving units", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Children's furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="Children's furniture", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Bar furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="Bar furniture", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Outdoor furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="Outdoor furniture", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Trolleys", 'width'] = avg_parametr.loc[avg_parametr.index=="Trolleys", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Sofas & armchairs", 'width'] = avg_parametr.loc[avg_parametr.index=="Sofas & armchairs", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Tables & desks", 'width'] = avg_parametr.loc[avg_parametr.index=="Tables & desks", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Wardrobes", 'width'] = avg_parametr.loc[avg_parametr.index=="Wardrobes", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Nursery furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="Nursery furniture", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Café furniture", 'width'] = avg_parametr.loc[avg_parametr.index=="Café furniture", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Sideboards, buffets & console tables", 'width'] = avg_parametr.loc[avg_parametr.index=="Sideboards, buffets & console tables", 'median_width'].values[0]
exer4_2.loc[exer4_2['category']=="Room dividers", 'width'] = avg_parametr.loc[avg_parametr.index=="Room dividers", 'median_width'].values[0]

                    # запишемо заповнені значення стовпчика height до таблиці NEW_data
                    conn = sqlite3.connect('sql_step_project.db')
                    conn.execute('BEGIN TRANSACTION')
                    for index, row in exer4_2.iterrows():
                        item_id=row['item_id']
                        category = row['category']
                        width = row['width']
                        conn.execute('UPDATE NEW_data SET width_new = ? WHERE item_id = ? AND
                    category = ?', (width, item_id, category))
                    conn.commit()
```

Висновки по відновленню відсутніх даних. На основі моделі лінійної регресії були заповнені дані стовпчика ‘old_price’ (у стовпчик ‘old_price_predict’). Втрачені дані стовпчика ‘designer’ заповнені найчастіше представленими за кожною категорією (у стовпчик ‘designer_new’). Пропущені дані стовпчиків **‘depth’**, **‘height’**, **‘width’** заповнені середніми або медіанами за кожною категорією, в залежності від властивостей розподілу за цією категорією.

4. Модель передбачення ціни

Для створення моделі передбачення ціни я взяла наступні змінні **‘name’, ‘category’, ‘price’, ‘old_price_predict’, ‘designer_new’, ‘depth_new’, ‘height_new’, ‘width_new’**, де **‘old_price_predict’, ‘designer_new’, ‘depth_new’, ‘height_new’, ‘width_new’** – дані, які були відновлені.

```
# запит вибору стовпчиків необхідний для стоворення моделі
cursor.execute("""SELECT name,category, price, old_price_predict,designer_new, depth_new,
 height_new, width_new
            FROM NEW_data""")
exer8= cursor.fetchall()
exer8=pd.DataFrame(exer8,columns=['name','category', 'price',
 'old_price_predict','designer_new', 'depth_new',
                          'height_new', 'width_new'])
```

Моделі будувала на основі DecisionTreeRegressor та KNeighborsRegressor. В процесі тестування моделей на основі усіх попередньо перерахованих даних виникла проблема пов’язана з тим, що факторні змінні **‘name’, ‘designer_new’** мають забагато різних варіант, видавалося повідомлення, що деякі значення факторних змінних не відомі моделі. Тому, щоб створити працюючі моделі, я зменшила кількість вхідних змінних, прибравши **‘name’, ‘designer_new’.**

```
# вибір даних для створення моделі
X=exer8[['category','old_price_predict', 'depth_new','height_new', 'width_new']]
y=exer8['price']

# розбиття даних на тренувальні та тестувальні
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=42)
```

Створимо модель передбачення на основі DecisionTreeRegressor:

```
# створення моделі передбачення на основі DecisionTreeRegressor
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
numeric_transformer = Pipeline(steps=[
       ('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[
      ('onehot', OneHotEncoder())])
column_preprocessing = ColumnTransformer(transformers=[
   ('numeric', numeric_transformer,['old_price_predict', 'depth_new','height_new',
 ‘width_new’]),
   (‘categorical’, categorical_transformer,[‘category’])])
clf = Pipeline(steps=[
   (‘preprocessing’, column_preprocessing),
   (‘clf’, DecisionTreeRegressor())])
```

Виберемо найкращу глибину для моделі DecisionTreeRegressor з використанням методу GridSearchCV

```
# перевірка та підбір найкращої стратегії (max_depth)
from sklearn.model_selection import GridSearchCV
gridsearch=GridSearchCV(estimator=clf,
                param_grid = {'clf__max_depth':
[None,1,2,3,4,5,6,7,8,9,10,20,30,40,50,100]
})
gridsearch.fit(X_train, y_train)
print(gridsearch.best_params_)
print(gridsearch.best_score_)
clf.fit(X_train, y_train)
mean_squared_error(y_test,clf.predict(X_test))
```

```
{'clf__max_depth': 7}
gridsearch.best_score
0.9940952577297868

mean_squared_error
4631.457408906883
```

Найкраще значення max_depth=6. Побудуємо модель з використанням цього значення параметру та за допомогою техніки cross-validation перевіримо ефективність отриманої моделі.

```
# створення моделі передбачення на основі DecisionTreeRegressor з найкращою
стратегією, max_depth=6
numeric_transformer = Pipeline(steps=[
      ('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[
     ('onehot', OneHotEncoder())])
column_preprocessing = ColumnTransformer(transformers=[
   ('numeric', numeric_transformer,['old_price_predict', 'depth_new','height_new',
'width_new']),
   ('categorical', categorical_transformer,['category'])])
clf = Pipeline(steps=[
   ('preprocessing', column_preprocessing),
   ('clf', DecisionTreeRegressor(max_depth=6))])
```

```
 # перехресна перевірка
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X, y, cv=5)
print(scores)
print(scores.mean())
clf.fit(X_train,y_train)
mean_squared_error(y_test,clf.predict(X_test))
```

```
[0.99645889, 0.99614562, 0.99620771, 0.99369414,
0.99587524]
scores.mean
0.9956763206430062

mean_squared_error
6251.934538140453
```

Значення оцінки прямує до одиниці, це означає, що точність отриманого результату на основі цієї моделі достатньо висока.

Створимо модель передбачення на основі KNeighborsRegressor:

```
# створення моделі передбачення на основі KNeighborsRegressor
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
numeric_transformer = Pipeline(steps=[
      ('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[
     ('onehot', OneHotEncoder())])
column_preprocessing = ColumnTransformer(transformers=[
```

```
    ('numeric', numeric_transformer,['old_price_predict',
'depth_new','height_new', 'width_new']),
    ('categorical', categorical_transformer,['category'])])
model = Pipeline(steps=[
    ('preprocessing', column_preprocessing),
    ('reg', KNeighborsRegressor())])
```

Виберемо найкращий параметр для KNeighborsRegressor, який визначає кількість найближчих сусідів.

```
 # перевірка та підбір найкращої стратегії
# Find the best value of "k"
from sklearn.metrics import accuracy_score
for k in range(1, 21):
    numeric_transformer = Pipeline(steps=[('scaler',
StandardScaler())])
    categorical_transformer = Pipeline(steps=[('onehot',
OneHotEncoder())])
    column_preprocessing =
ColumnTransformer(transformers=[
    ('numeric', numeric_transformer,['old_price_predict',
'depth_new','height_new', 'width_new']),
    ('categorical', categorical_transformer,['category'])])
    model = Pipeline(steps=[
    ('preprocessing', column_preprocessing),
    ('reg', KNeighborsRegressor(n_neighbors=k))])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    msr =
mean_squared_error(y_test,model.predict(X_test))
    print("k:", k, "mean_squared_error:", msr)
```

k: 1 mean_squared_error: 48750.41018893388
k: 2 mean_squared_error: 44387.504298245614
k: 3 mean_squared_error: 47037.34362273205
k: 4 mean_squared_error: 50628.56895748987
k: 5 mean_squared_error: 51702.30704831309
k: 6 mean_squared_error: 56414.93213337831
k: 7 mean_squared_error: 60971.98071910546
k: 8 mean_squared_error: 64781.97206224696
k: 9 mean_squared_error: 67683.93469652288
k: 10 mean_squared_error: 68919.45061538462
k: 11 mean_squared_error: 72749.57781811489
k: 12 mean_squared_error: 74327.66305967911
k: 13 mean_squared_error: 77093.87382195817
k: 14 mean_squared_error: 79958.86660717728
k: 15 mean_squared_error: 80638.83855174689
k: 16 mean_squared_error: 82402.12315383561
k: 17 mean_squared_error: 84698.76982764338
k: 18 mean_squared_error: 87980.42413334499
k: 19 mean_squared_error: 89546.41890781716
k: 20 mean_squared_error: 92572.67792179486

Найкраще значення n_neighbors =2. Побудуємо модель з використанням цього значення параметру та за допомогою техніки cross-validation перевіримо ефективність отриманої моделі.

```
# створення моделі передбачення на основі KNeighborsRegressor з найкращою стратегією,
n_neighbors=2
numeric_transformer = Pipeline(steps=[
      ('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[
      ('onehot', OneHotEncoder())])
column_preprocessing = ColumnTransformer(transformers=[
    ('numeric', numeric_transformer,['old_price_predict', 'depth_new','height_new', 'width_new']),
    ('categorical', categorical_transformer,['category'])])
model = Pipeline(steps=[
    ('preprocessing', column_preprocessing),
    ('reg', KNeighborsRegressor(n_neighbors=2))])
```

```
# перехресна перевірка
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print(scores)
print(scores.mean())
model.fit(X_train,y_train)
mean_squared_error(y_test,model.predict(X_test))
```

[0.963896, 0.97760489, 0.97834837, 0.973317
22, 0.97305159]
scores.mean
0.9732436125082495
mean_squared_error
44387.504298245614

Значення оцінки прямує до одиниці, це означає, що точність отриманого передбачення з допомогою цієї моделі також є достатньо висока, але трохи менша за попередню модель.

Застосуємо отримані моделі для передбачення ціни на вигаданих вхідних даних.

```
# дані для передбачення
new_data = pd.DataFrame({
    'category': ['Chairs'],
    'old_price_predict': [320],
    'depth_new': [50],
    'height_new': [80],
    'width_new': [40]
})
# передбачення за допомого моделі DecisionTreeRegressor
Price_new=clf.predict(new_data)
Price_new.round(1)
```

array([238.6])

```
# передбачення за допомого моделі KNeighborsRegressor
Price_new_1=model.predict(new_data)
Price_new_1.round(1)
```

array([198.5])

Висновок з побудови моделей передбачення. Порівнявши дві побудовані моделі DecisionTreeRegressor та KNeighborsRegressor передбачення ціни за двома метриками: cross_val_score та mean_squared_error, можна побачити, що значення cross_val_score для моделі DecisionTreeRegressor (0.9940952577297868) більше, ніж для другої моделі (0.9732436125082495). Це може означати, що перша модель краще підходить для даної задачі, оскільки вона має більший середній показник точності при крос-валідації. А при порівнянні за допомогою mean_squared_error, можна побачити, що перша модель (6251.934538140453) має меншу середньоквадратичну помилку, ніж KNeighborsRegressor модель (44387.504298245614). Це може означати, що перша модель краще передбачає значення цільової змінної для тестової вибірки. Таким чином DecisionTreeRegressor модель є точнішою та має меншу середньоквадратичну помилку передбачення ціни товару, ніж KNeighborsRegressor модель.

Зменшивши варіацію змінної **‘designer’** можна б було побудувати модель передбачення ціни з використанням і цього фактору, але це не входило в задачі мого дослідження.

Загальні висновки до проекту

Відповідно Data Quality Framework набор вхідних даних має наступні характеристики.

Data Quality Criteria	*Data Quality Metrics*	*Data Quality Score*
Accuracy	Error rate	0.3%
Completeness	Field completeness	12.96%
Consistency	Field consistency	wasn’t found
Validity	Format validation	0.03%
Uniqueness	Duplication rate	19.82%

Нормативні дані стовпчиків **‘price’, ‘old_price’, ‘depth’, ‘height’, ‘width’** не розподілені ні нормально, ні рівномірно, містять значну кількість викидів. Відповідно попередніх статистичних досліджень дані стовпчиків **‘price’, ‘old_price’** мають лінійну залежність, також ціна лінійно залежна від розмірів товару. Попередній розгляд розподілів нормативних даних за факторними даними **‘category’** та **‘designer’** дає можливість припускати, що є значний вплив цих факторів на ціноутворення, на яке також може впливати розмір товару. Дані припущення потребують окремої детальної перевірки. Наприклад, якщо застосувати двофакторний дисперсійний аналіз для перевірки залежності змінних **‘category’** та **‘designer’** на **‘price’,** тест дає наступний результат.

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Задаємо формулу для моделі
formula = 'price ~ C(category) + C(designer) + C(category):C(designer)'

# Створюємо модель та обчислюємо ANOVA
model = ols(formula, data=exer6).fit()
table = sm.stats.anova_lm(model, typ=2)

# Виводимо результати ANOVA
print(table)
```

	sum_sq	df	F	PR(>F)
C(category)	**-1.459431e+01**	**16.0**	**-9.353310e-07**	**1.000000e+00**
C(designer)	**-6.537808e+09**	**278.0**	**-2.411510e+01**	**1.000000e+00**
C(category):C(designer)	**2.232818e+10**	**4448.0**	**5.147428e+00**	**1.266065e-168**
Residual	**2.244935e+09**	**2302.0**	**NaN**	**NaN**

Значення *p* для ефекту взаємодії (категорія: дизайнер) дуже мале (1,266065e-168), що означає значний вплив взаємодії між категорією та дизайнером на ціну. Іншими словами, вплив дизайнера на ціну залежить від категорії, і навпаки.

p-значення для основного ефекту категорії та дизайнера дорівнює 1, що означає, що немає суттєвої різниці в ціні між категоріями чи дизайнерами після контролю за іншими факторами. Однак цей результат слід інтерпретувати з обережністю, оскільки значний ефект взаємодії свідчить про те, що зв’язок між ціною, категорією та дизайнером є складнішим, ніж просте порівняння засобів.

Підсумовуючи, тест ANOVA показує, що існує значний вплив взаємодії між категорією та дизайнером на ціну, але немає істотного основного впливу категорії чи дизайнера на ціну.

А оскільки це дослідження потребує детального попереднього аналізу, додаткових статистичних тестів, кожної групи категорія-дизайнер, які як було описано, мають не нормальні, не рівні за розмірами, з великою кількістю викидів розподіли, то стверджувати, що вплив значення змінних полів **‘category’** та **‘designer’** на змінну **поля ‘price’** є статистично доведеним не варто.

Були заповнені дані стовпчиків **‘old_price’** (у стовпчик ‘old_price_predict’), **‘designer’** (у стовпчик ‘designer_new’), **‘depth’**, **‘height’**, **‘width’** (у стовпчики **‘depth_new’**, **‘height_new’**, **‘width_new’**).

Були побудовані моделі передбачення ціни на основі DecisionTreeRegressor та KNeighborsRegressor. Виявлено, що DecisionTreeRegressor модель є точнішою та має меншу середньоквадратичну помилку передбачення ціни товару, ніж KNeighborsRegressor модель.

Щоб покращити якість передбачення моделі варто б було навчити її враховувати такі факторні змінні як **‘designer’, ‘name’.**

В самому наборі вхідних даних також хотілось би мати деякі додаткові характеристики товарів, які впливають на його ціну, наприклад собівартість товарів, вартість матеріалів, роботи, тощо.