1. 순수익이 높은 카테고리
2. 손해가 발생하는 카테고리
    1. 손해가 있다고 무조건 버리기 보단, 어떤 구조 때문에 손해가 발생하는지 확인 필요
        1. 배송 카테고리 참고
            1. 기대 배송 <> 수익 간 상관관계
                1. 매장을 증설하거나 물류센터 구축
            2. 배송 비용
                1. 운임 비용이 비싸서 손해가 발생한다면
                    1. 가까운 지역에서 구매를 유도
            3. 무게와 거리의 효율성 체크

1. 순수익이 높은 카테고리

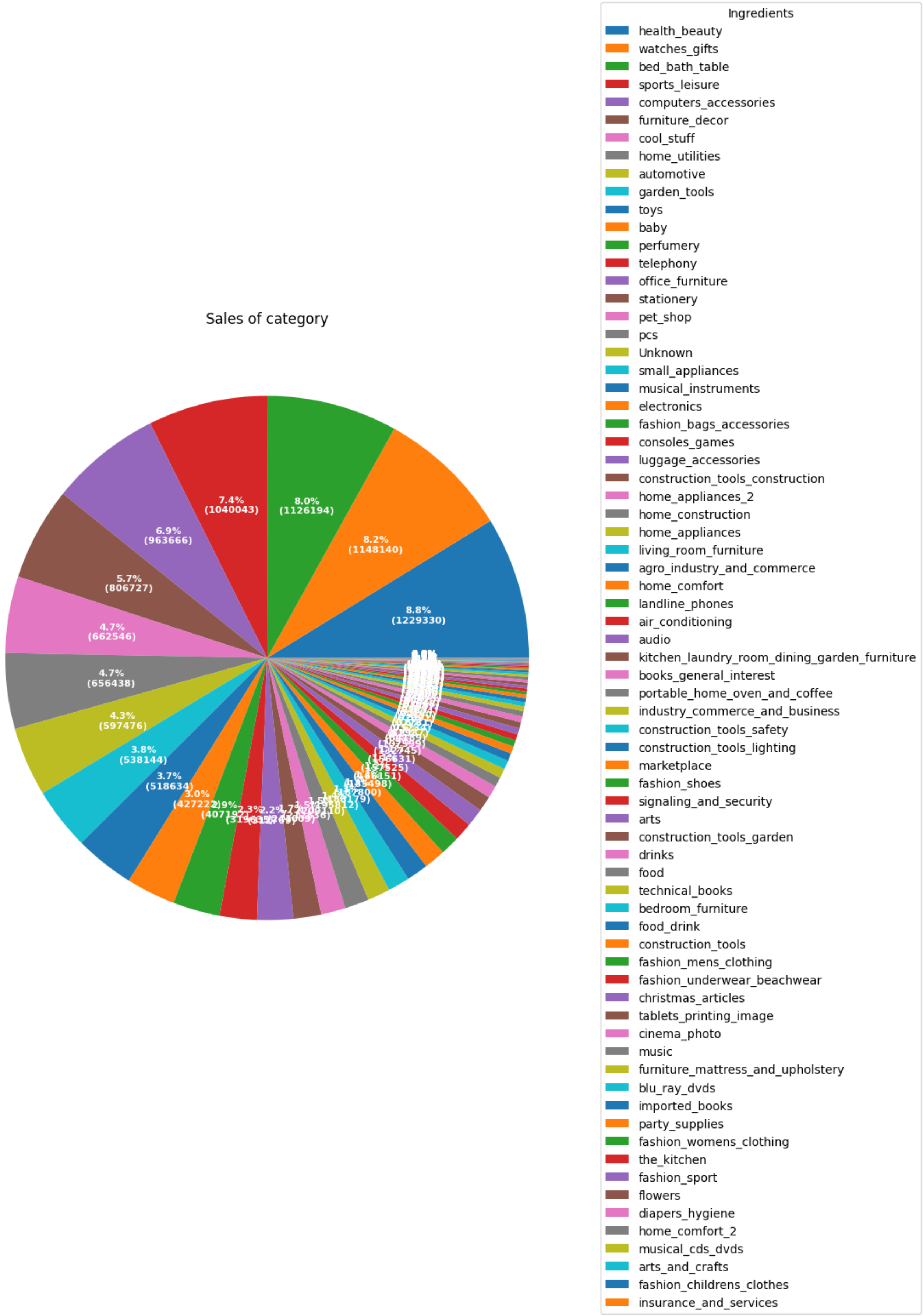
```
- 데이터 정제 과정
```

```
SELECT *
FROM (
        select pp.Product_category_name, oi.Product_id, oi.Order_id ,
oi.Order_item_id, oi.Seller_id
        from (
        select p.Product_id, p.Product_category_name
        from products p
        ) pp
        join
        order_items oi
        on oi.Product_id = pp.Product_id
) as poi
JOIN payments
ON payments.Order_id = poi.Order_id
order by Payment_value desc;

-- Products_order_items_payments_202404271813.csv 파일 추출
```

```
# SQL에서 추출한 csv파일 표시
poip_df = pd.read_csv('Products_order_items_payments_202404271813.csv')

# Order_item_id 8까지 존재하는 샘플 확인
poip_df[poip_df.Order_id == 'ORDER_11847']
```

```
# 데이터 분류를 위해 Order_item_id이 1인 항목만 추
Order_item_id = poip_df[poip_df.Order_item_id==1]


# 카테고리별 Payment_value 값 담기
from tqdm import tqdm
for item in category:
    locals()[f'{item}_value'] = []

for idx, row in tqdm(Order_item_id.iterrows()):
    for item in category:
        if Order_item_id.iloc[idx]['Product_category_name'] != item:
            continue
        else:
            locals()[f'{item}_value'].append(Order_item_id.iloc[idx]
['Payment_value'])
            break


# 카테고리별 수익 합산
for item in category:
    locals()[f'{item}'] = []
    locals()[item] = pd.DataFrame(locals()[f"{item}_value"]).sum()
```

```
# 카테고리 이름 목록 추출
category = products.Product_category_name.unique()


# 카테고리별 판매 수량, 내림차순
number_of_sales_by_category = pd.DataFrame({"count":category_cnt},
index=category).sort_values(by='count', ascending=False)
```


Sales of category
8.8%
(1229330)
8.2%
(1148140)
8.0%
(1126194)
7.4%
(1040043)
6.9%
(963666)
5.7%
(806727)
4.7%
(662546)
4.7%
(656438)
4.3%
(597476)
3.8%
(538144)
3.7%
(518634)
3.0%
(427222)
2.9%
(407192)
2.3%
2.2%
Ingredients
health_beauty
watches_gifts
bed_bath_table
sports_leisure
computers_accessories
furniture_decor
cool_stuff
home_utilities
automotive
garden_tools
toys
baby
perfumery
telephony
office_furniture
stationery
pet_shop
pcs
Unknown
small_appliances
musical_instruments
electronics
fashion_bags_accessories
consoles_games
luggage_accessories
construction_tools_construction
home_appliances_2
home_construction
home_appliances
living_room_furniture
agro_industry_and_commerce
home_comfort
landline_phones
air_conditioning
audio
kitchen_laundry_room_dining_garden_furniture
books_general_interest
portable_home_oven_and_coffee
industry_commerce_and_business
construction_tools_safety
construction_tools_lighting
marketplace
fashion_shoes
signaling_and_security
arts
construction_tools_garden
drinks
food
technical_books
bedroom_furniture
food_drink
construction_tools
fashion_mens_clothing
fashion_underwear_beachwear
christmas_articles
tablets_printing_image
cinema_photo
music
furniture_mattress_and_upholstery
blu_ray_dvds
imported_books
party_supplies
fashion_womens_clothing
the_kitchen
fashion_sport
flowers
diapers_hygiene
home_comfort_2
musical_cds_dvds
arts_and_crafts
fashion_childrens_clothes
insurance_and_services

SQL 조회

```
SELECT *
FROM
(
SELECT B.ORDER_ID, B.CUSTOMER_ID, A.CUSTOMER_UNIQUE_ID,
A.CUSTOMER_ZIPCODE_PREFIX
FROM customers A
JOIN orders B
ON A.Customer_id = B.Customer_id
) C
JOIN payments p
ON C.ORDER_ID = p.Order_id
order by Payment_value desc;
```

PRICE + FREIGHT_VALUE = 결제금액

```
sum_value = []
for idx, row in order_items.iterrows():
    sum_value.append(order_items.iloc[idx]['Price'] + order_items.iloc[idx]
['Freight_value'])
order_items['sum_value'] = sum_value
```

위 데이터가 잘못된거 같아서 다시 추출

```
-- order_items + payments + products
-- count : 91971
select P.Product_id, P.Product_category_name, C.Order_id, C.pay_installments,
C.pay_value
from products P
join (
        select A.Order_id, A.Order_item_id, A.Product_id, A.Seller_id,
        B.Payment_installments as pay_installments,
        B.Payment_value as pay_value
        from order_items A
        join
                (
                select Order_id, Payment_type, Payment_installments,
Payment_value
                from payments order by Payment_value desc
                ) B
        on A.Order_id = B.order_id
        where Order_item_id = 1
        order by pay_value desc
        ) C
on P.Product_id = C.Product_id
order by C.pay_value desc;
```

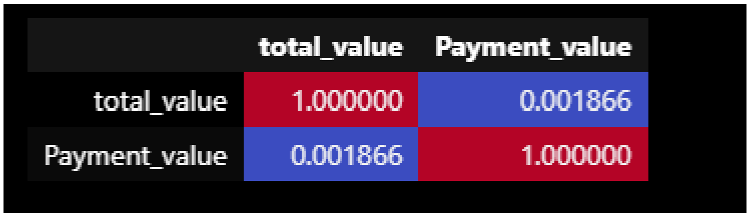
pandas [order by]

```
# order by = sort_values
category_merge = category_merge.sort_values(by='div_value', ascending=False)

# 카테고리별, 가격별 내림차순
temp4.sort_values(by=['Product_category_name', 'Payment_value'], axis=0,
ascending=False)
```

상관관계 구하기
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

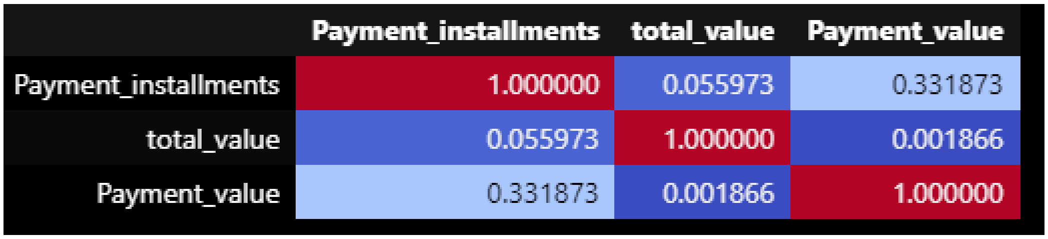
pandas 상관계수

```

# 필요시 설치
# pip install Jinja2

temp3 = pd.DataFrame(  {'total_value': temp2['total_value'],'Payment_value':
temp2['Payment_value']})

corr = temp3.corr()
corr.style.background_gradient(cmap='coolwarm', axis=None)
```



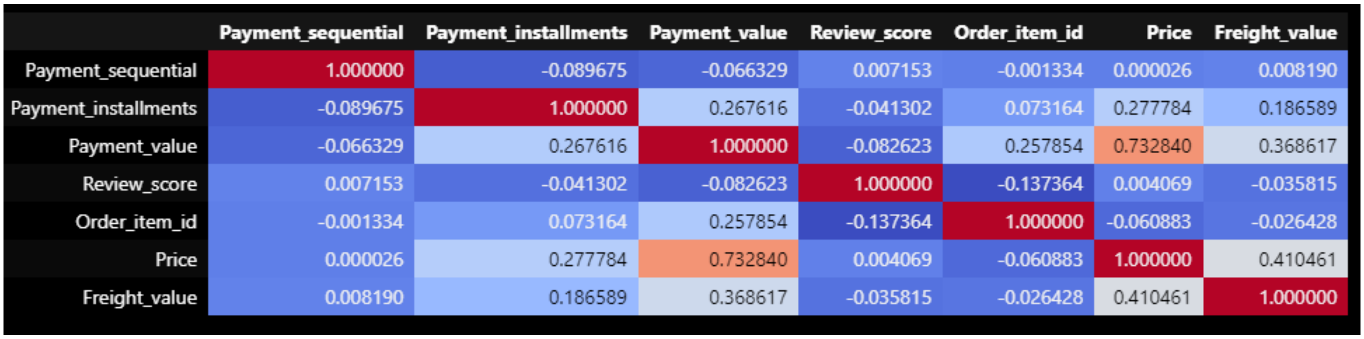
	total_value	Payment_value
total_value	1.000000	0.001866
Payment_value	0.001866	1.000000

```
corr2 = temp4.corr(numeric_only=True)
corr2.style.background_gradient(cmap='coolwarm', axis=None)
```



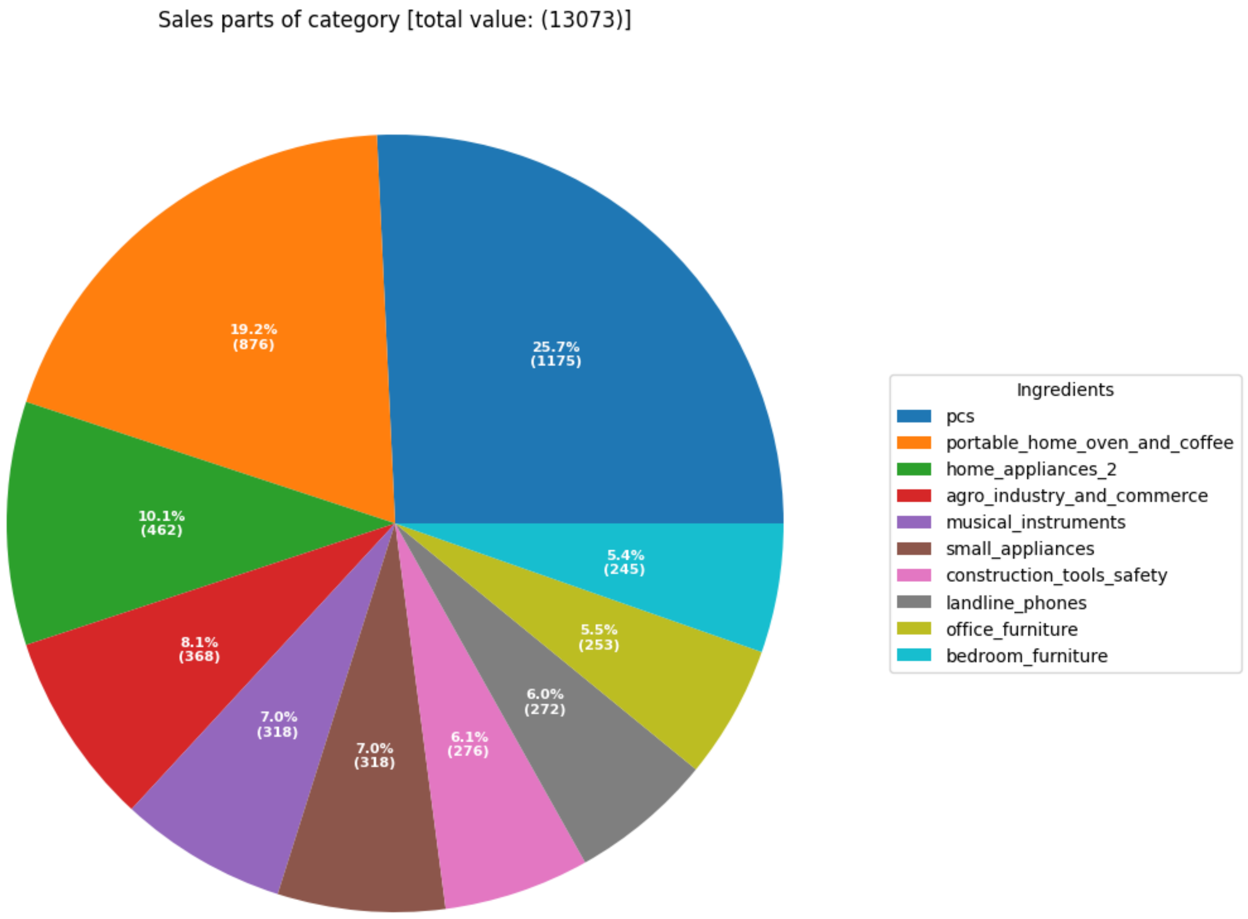
	Payment_installments	total_value	Payment_value
Payment_installments	1.000000	0.055973	0.331873
total_value	0.055973	1.000000	0.001866
Payment_value	0.331873	0.001866	1.000000

```
pr = payments.merge(reviews, how='inner', on='Order_id')
pro = pr.merge(order_items, how='inner', on='Order_id')
pro.merge(products, how='inner',
on='Product_id').corr(numeric_only=True).style.background_gradient(cmap='coolwarm', axis=None)
```



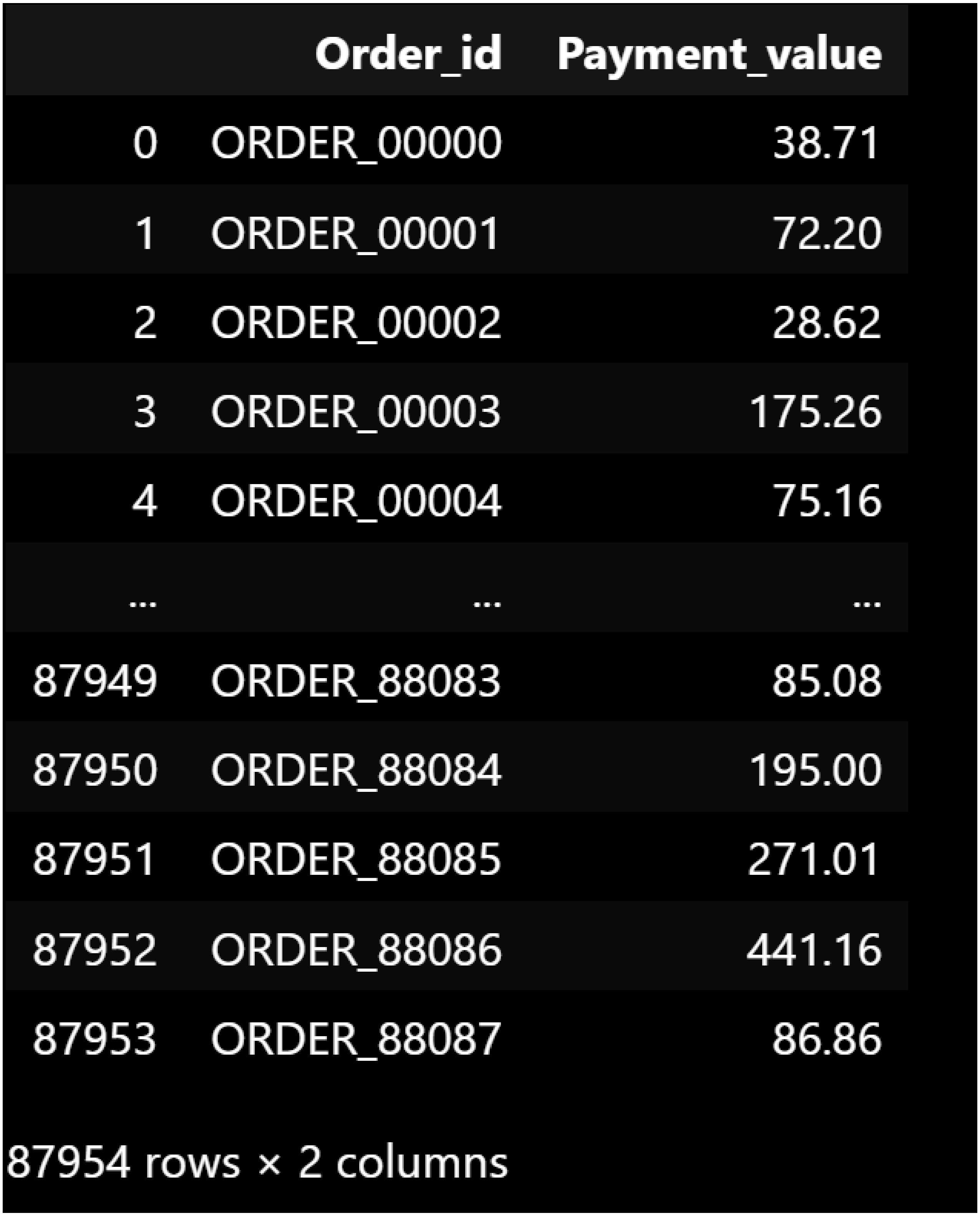
	Payment_sequential	Payment_installments	Payment_value	Review_score	Order_item_id	Price	Freight_value
Payment_sequential	1.000000	-0.089675	-0.066329	0.007153	-0.001334	0.000026	0.008190
Payment_installments	-0.089675	1.000000	0.267616	-0.041302	0.073164	0.277784	0.186589
Payment_value	-0.066329	0.267616	1.000000	-0.082623	0.257854	0.732840	0.368617
Review_score	0.007153	-0.041302	-0.082623	1.000000	-0.137364	0.004069	-0.035815
Order_item_id	-0.001334	0.073164	0.257854	-0.137364	1.000000	-0.060883	-0.026428
Price	0.000026	0.277784	0.732840	0.004069	-0.060883	1.000000	0.410461
Freight_value	0.008190	0.186589	0.368617	-0.035815	-0.026428	0.410461	1.000000

전체 결제 금액 대비 카테고리별 1회 결제 금액



payments

```
# payments['Order_id'] payment_value 합산
pay_sum = payments.groupby(['Order_id'])['Payment_value'].sum().reset_index()


# 결제 금액과 할부 횟수 관계
```



	Order_id	Payment_value
0	ORDER_00000	38.71
1	ORDER_00001	72.20
2	ORDER_00002	28.62
3	ORDER_00003	175.26
4	ORDER_00004	75.16
...	...	...
87949	ORDER_88083	85.08
87950	ORDER_88084	195.00
87951	ORDER_88085	271.01
87952	ORDER_88086	441.16
87953	ORDER_88087	86.86

87954 rows × 2 columns