First let's look at the tasks that we have.

Analyze a dataset to uncover trends, insights, and patterns that can help improve business decision-making. Additionally, create a synthetic dataset representing daily total sales by store and department.

Steps:

Data Cleaning:
Identify and handle missing values.
Standardize column names and formats (e.g., dates, currencies).
Remove duplicates and irrelevant columns.

Exploratory Data Analysis (EDA):
Provide a summary of the dataset.
Identify trends, correlations, and outliers.
Visualize key data points using charts.

Business Insights:
Identify the top 3 insights that could help improve the business.
Support findings with data and visuals.

Data cleaning

Before merging 3 data sources to have a new dataset, which will be more comfortable and easy to use, we must do data cleaning. Let's look for missing values, duplicated values, datatypes and so on.

First we import data to our jupyter notebook environment. Our data contained 3 datasets named as ‘Source1’, ‘Source2’, ‘Data2’.

Now let's analyze each df starting with df_source1.
The head of our dataframe:

	date	store	upc	retail_price
0	2024-12-21	473	9999312	603.36
1	2024-12-21	471	9999312	603.36
2	2024-12-21	493	9999312	603.36
3	2024-12-21	493	9999312	603.36
4	2024-12-21	471	9999312	603.36

Information about df_source1:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28098 entries, 0 to 28097
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          28098 non-null  object
 1   store         28098 non-null  int64
 2   upc           28098 non-null  int64
 3   retail_price  25892 non-null  float64
dtypes: float64(1), int64(2), object(1)
memory usage: 878.2+ KB
```

As we can see, we have 28098 records, and some missing values in retail_price column, let's analyze column, and decide how to handle those values.

We have 2206 missing values in retail_price column:

We can handle those 2206 missing values by assigning them the mean of the column.

Now let's check for duplicated values in df_source1:
It shows that we have 913 duplicated records.
I decided to drop duplicated values, keeping the first ones in dataset.

```
<class 'pandas.core.frame.DataFrame'>
Index: 27185 entries, 0 to 28097
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   date          27185 non-null  object 
 1   store         27185 non-null  int64  
 2   upc           27185 non-null  int64  
 3   retail_price  27185 non-null  float64
dtypes: float64(1), int64(2), object(1)
memory usage: 1.0+ MB
```

after handling missing values, let's look to column's datatypes.
As we see, 'date' column is a object type column, that will bother us in the future where we're going to analyze data based on its date. So let's convert it to datetime type.

Next step : analyzing df_source2.

	date	store	upc	family_code	price_type	unit_price	unit_cost
0	2024-12-20	493	7084781116	1432	R	3.39	-0.1
1	2024-12-18	470	7084781116	1432	R	3.69	-0.1
2	2024-12-24	493	7084781116	1432	R	3.39	-0.1
3	2024-12-21	470	7084781116	1432	R	3.39	-0.1
4	2024-12-23	470	7084781116	1432	R	3.39	-0.1

This is how our data looks like.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28098 entries, 0 to 28097
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         28098 non-null  datetime64[ns]
 1   store        28098 non-null  int64
 2   upc          28098 non-null  int64
 3   family_code  28098 non-null  object
 4   price_type   9366 non-null   object
 5   unit_price   14478 non-null  float64
 6   unit_cost    14478 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(2), object(2)
memory usage: 1.5+ MB
```

There is lot of more things here to do.

'price_type','unit_price','unit_cost', decide how to handle missing data correctly.

Now let's look for duplicated values, and drop them.
I handle duplicated values same way as in df_source1.
It shows it has 2824 duplicated values. Keeping the firs ones and dropping the other ones.

Now let's look at missing values in 'family_code' column.
Problem : it shows that it does not have missing values,but all we see are blank blocks, blocks are not empty, they are filled with whitespaces, so we need to cut all whitespaces out, and then try to convert family_code to numeric, but it's necessary to keep them as a string type, because the codes can begin with zeros. so we just need to handle missing values.
We replace '   ' 4 blank spaces, with np.nan values.
I searched in datasets for 'upc' codes, and was thinking that product that have same upc belong to the same family. So I wrote a function for finding family codes, but if there is not anything, that will still have its Nan value.
After appliyng that function, i droped every record that had Nan value.

Let's switch to 'unit_price' and 'unit_cost' columns.
For handling this kind of data I searched for a formula for unit cost and unit price, but beside retail price, they also required margin percentage, which we don't have in our data.
Filling missing values with the means of columns,

```
<class 'pandas.core.frame.DataFrame'>
Index: 25274 entries, 0 to 28097
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         25274 non-null  datetime64[ns]
 1   store        25274 non-null  int64
 2   upc          25274 non-null  int64
 3   family_code  20868 non-null  object
 4   price_type   8701 non-null   object
 5   unit_price   25274 non-null  float64
 6   unit_cost    25274 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(2), object(2)
memory usage: 1.5+ MB
```

let's dive deep in price_type column (not sure about these)

T (Transactional): Refers to a transactional price, such as the actual price at which an item was sold.

R (Retail): Represents the retail price, typically the listed price for an item in a store or catalog.

A (Average): Denotes the average price, possibly calculated over a period or from multiple sources.

(I am not very sure, is it right, the things I wrote above, because there were lot of words in that context under the letters ‘T’, ‘A’, ‘R’.)

I filled missing values with ‘R’s.

```
df_source2['price_type'].fillna('R', inplace=True)
```

now let's look at the data2 dataframe

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1897 entries, 0 to 1896
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   upc              1897 non-null   int64
 1   department_code  1891 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 29.8 KB
```

It seems like everything is okay with it. Just 6 missing values in department_code dataset, we can handle that easily.

Exploratory Data Analysis

Summary of datasets

df_source1

	date	store	upc	retail_price
count	27185	27185.000000	2.718500e+04	27185.000000
mean	2024-12-21 05:11:18.418245376	409.107523	1.243648e+10	2.852014
min	2024-12-18 00:00:00	65.000000	1.100000e+01	-8.000000
25%	2024-12-19 00:00:00	461.000000	2.100064e+09	1.000000
50%	2024-12-21 00:00:00	472.000000	4.400006e+09	1.000000
75%	2024-12-23 00:00:00	474.000000	7.069002e+09	3.062429
max	2024-12-24 00:00:00	493.000000	9.419199e+11	603.360000
std	NaN	149.185609	4.591158e+10	10.313845

df_source2

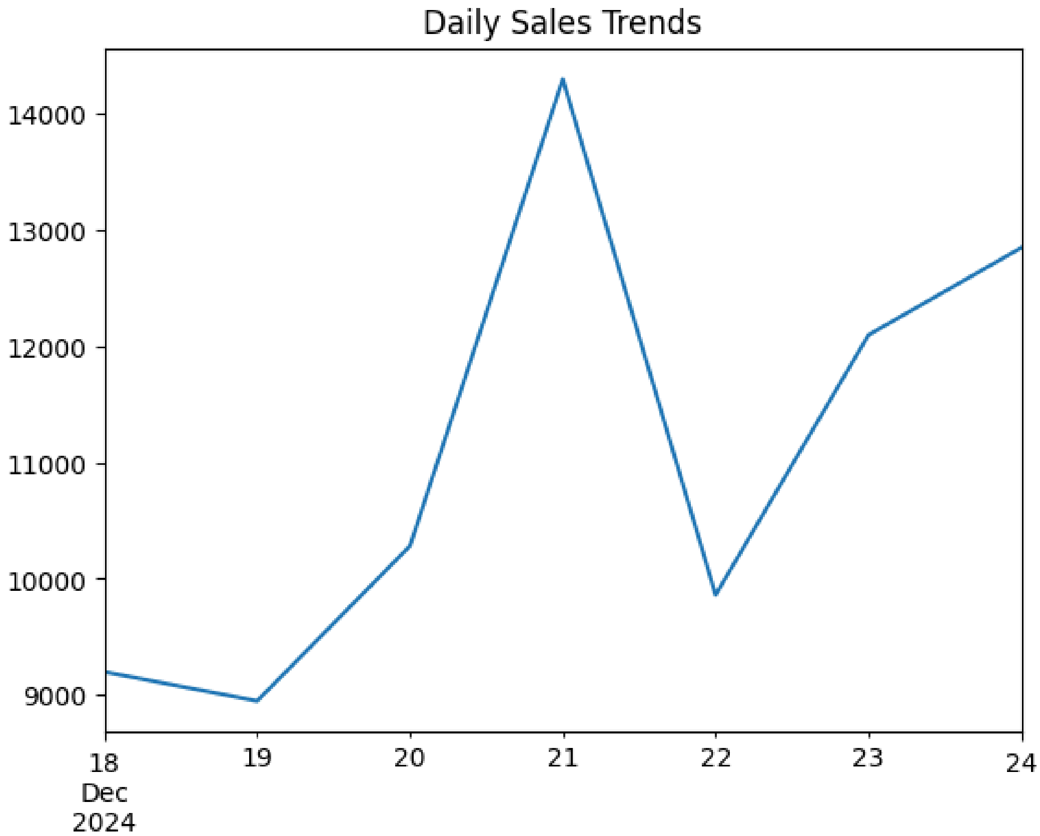
	date	store	upc	unit_price	unit_cost
count	25274	25274.000000	2.527400e+04	25274.000000	25274.000000
mean	2024-12-21 05:11:08.608055552	409.281435	1.242579e+10	4.712457	5.478711
min	2024-12-18 00:00:00	65.000000	1.100000e+01	-8.000000	-26.990000
25%	2024-12-19 00:00:00	461.000000	2.100064e+09	3.490000	3.000000
50%	2024-12-21 00:00:00	472.000000	4.400006e+09	4.712457	5.478711
75%	2024-12-23 00:00:00	474.000000	7.064002e+09	4.712457	5.478711
max	2024-12-24 00:00:00	493.000000	9.419199e+11	99.690000	418.560000
std	NaN	149.024570	4.601335e+10	3.357886	8.711650

df_data2

	upc	department_code
count	1.897000e+03	1891.000000
mean	1.657539e+10	29.957166
std	4.613947e+10	14.897809
min	1.100000e+01	1.000000
25%	2.800030e+09	31.000000
50%	4.400006e+09	32.000000
75%	7.402610e+09	33.000000
max	9.419199e+11	81.000000

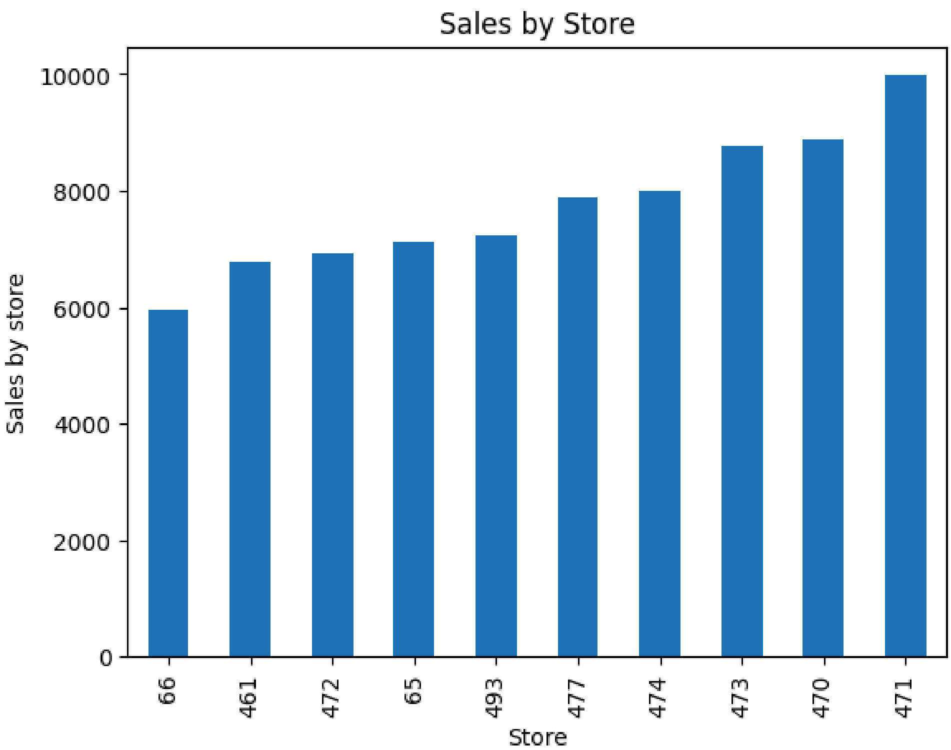
Task 1: Analyze sales trends over time.

1.Let’s analyze daily sales trends on our df_source1 dataframe

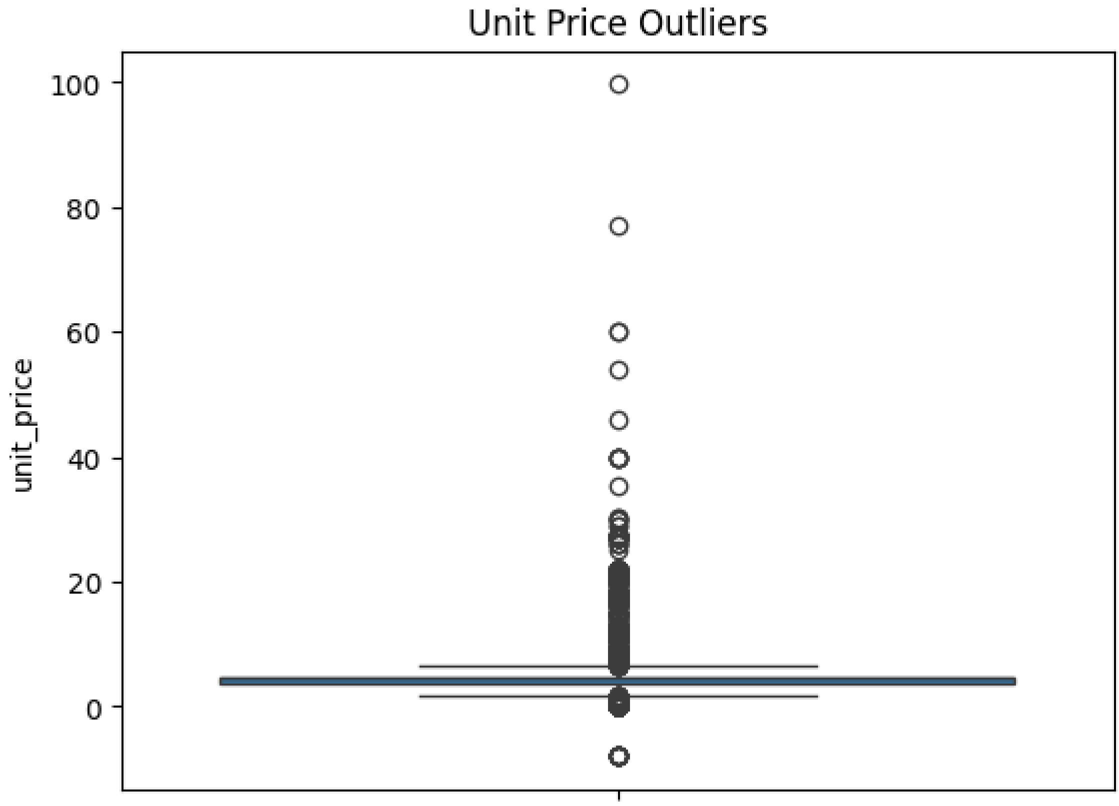


So we see that, in 21th of December, we have the most of daily sales.

2. Let's look to Store Performance.



Now let's look and find outliers, and make a new dataset, where we will include retail price, unit_cost and price, department code , store and so on



we also can use Z score for finding outliers

```
outliers
```

	date	store	upc	family_code	price_type	unit_price	unit_cost	zscore_unit_price
762	2024-12-20	470	1820096715	0000	R	27.49	23.25	6.783433
763	2024-12-19	474	1820096715	0000	R	27.49	23.45	6.783433
764	2024-12-19	473	1820096715	0000	R	27.49	23.45	6.783433
765	2024-12-18	474	1820096715	0000	R	27.49	23.45	6.783433
766	2024-12-18	66	1820096715	0000	R	27.49	24.25	6.783433
...	...	...	...	...	...	...	...	...
28024	2024-12-23	470	8066095715	NaN	T	16.99	13.20	3.656404
28025	2024-12-24	66	8066095715	NaN	T	17.39	13.60	3.775529
28026	2024-12-24	470	8066095715	NaN	T	16.99	13.20	3.656404
28027	2024-12-24	474	8066095715	NaN	T	16.49	13.20	3.507497
28028	2024-12-24	477	8066095715	NaN	T	17.39	13.60	3.775529

751 rows × 8 columns

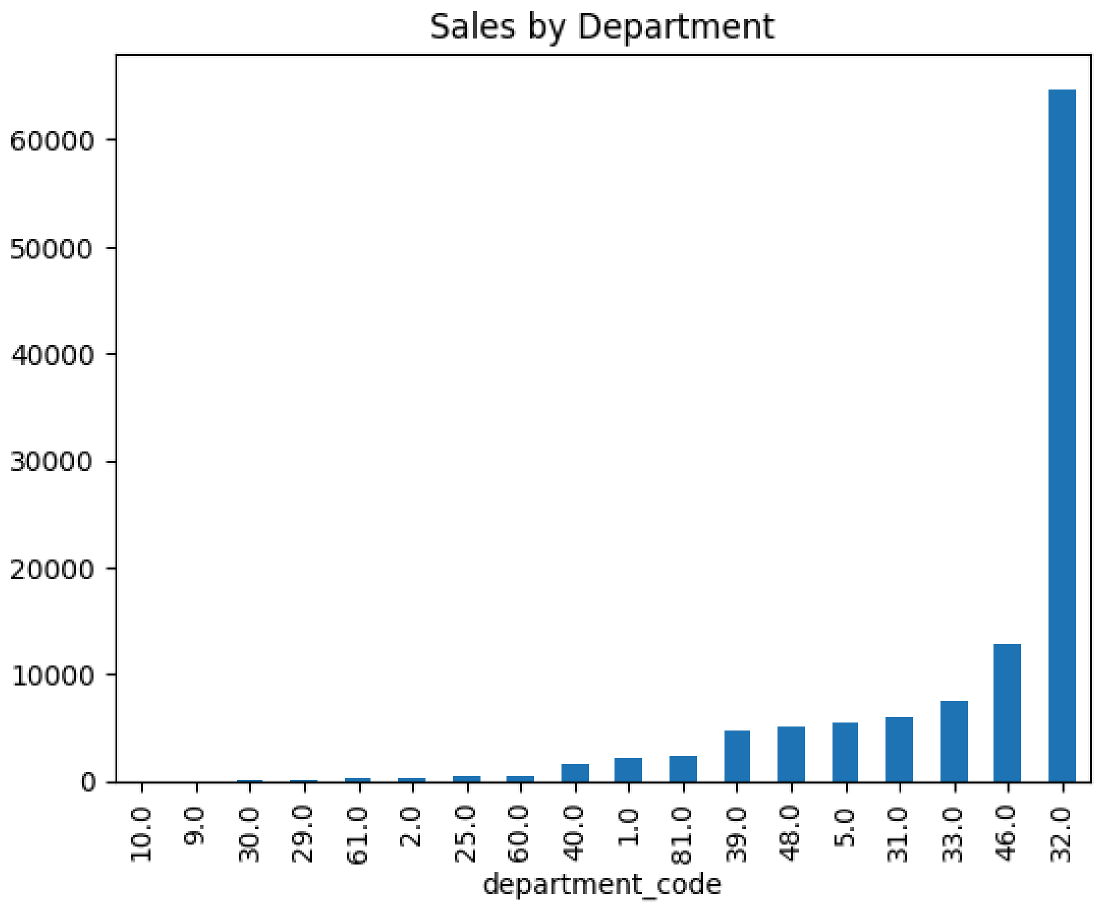
Now let's join datasets, records of each datasets are not equal, so, we need to merge only that records , that have same upc.

```
filtered_dataset
```

	date	store	upc	retail_price	family_code	price_type	unit_price	unit_cost	department_code
3	2024-12-22	461.0	961	39.990000	2617	T	29.990000	31.280000	81.0
4	2024-12-22	461.0	961	39.990000	2617	R	39.990000	31.280000	81.0
5	2024-12-20	471.0	961	39.990000	2617	T	29.990000	31.280000	81.0
6	2024-12-20	471.0	961	39.990000	2617	R	39.990000	31.280000	81.0
7	2024-12-20	472.0	961	39.990000	2617	T	29.990000	31.280000	81.0
...	...	...	...	...	...	...	...	...	...
42122	2024-12-21	471.0	941919903294	1.790000	0000	R	0.000000	0.000000	46.0
42123	2024-12-21	471.0	941919903294	1.000000	0000	R	4.712457	5.478711	46.0
42124	2024-12-21	471.0	941919903294	1.000000	0000	R	0.000000	0.000000	46.0
42125	2024-12-21	471.0	941919903294	3.062429	0000	R	4.712457	5.478711	46.0
42126	2024-12-21	471.0	941919903294	3.062429	0000	R	0.000000	0.000000	46.0

as we have our new dataset that consist everything which we need in it, let's continue our analysis and data visualization

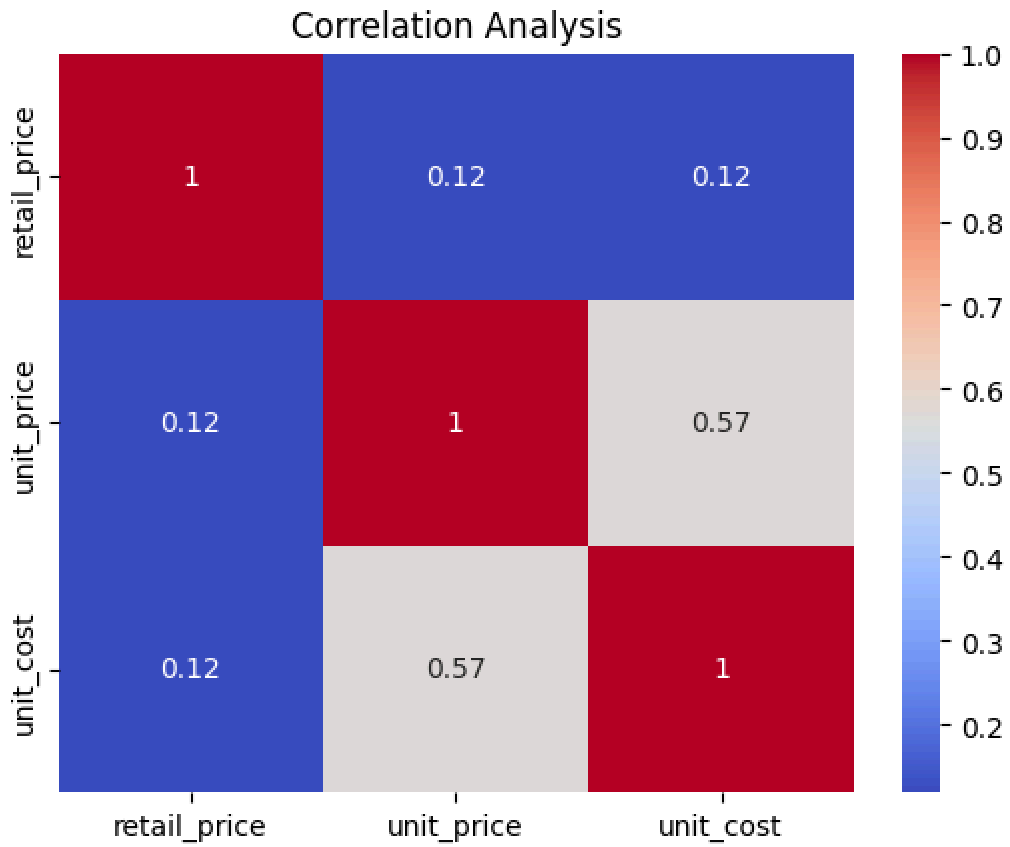
Department Performance Analyze which departments generate the most sales.



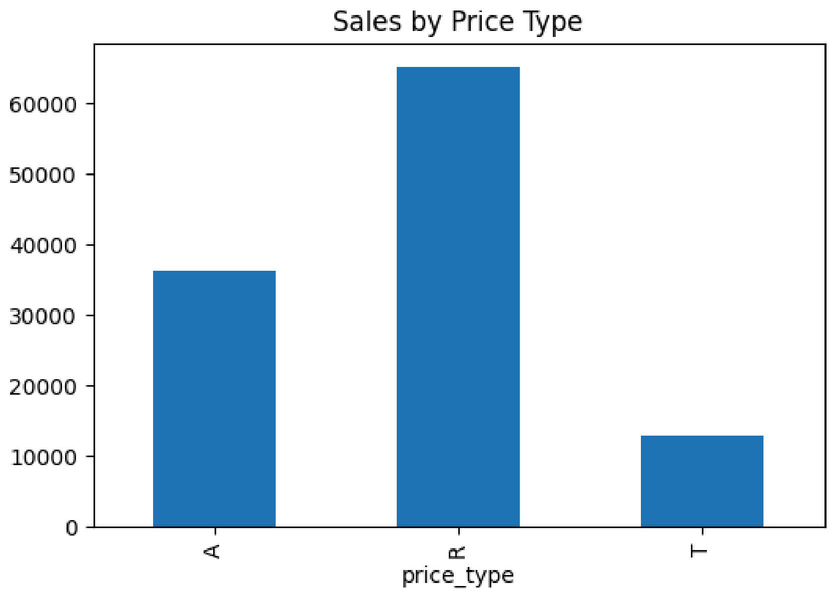
So department under department_code 32 does most of sales.

I wanted to replace department codes with their actual names, for us to be more clear, but i couldn't exactly find them.

Correlation Analysis



Now let's analyze trends by Price Types



Profit Margin analysis

Profit margin is a financial metric that shows the percentage of revenue that remains as profit after all expenses have been deducted. It helps measure how efficiently a company is managing its costs relative to its revenue.

	profit_margin
3	-1.290000
4	8.710000
5	-1.290000
6	8.710000
7	-1.290000
...	...
42122	0.000000
42123	-0.766254
42124	0.000000
42125	-0.766254
42126	0.000000

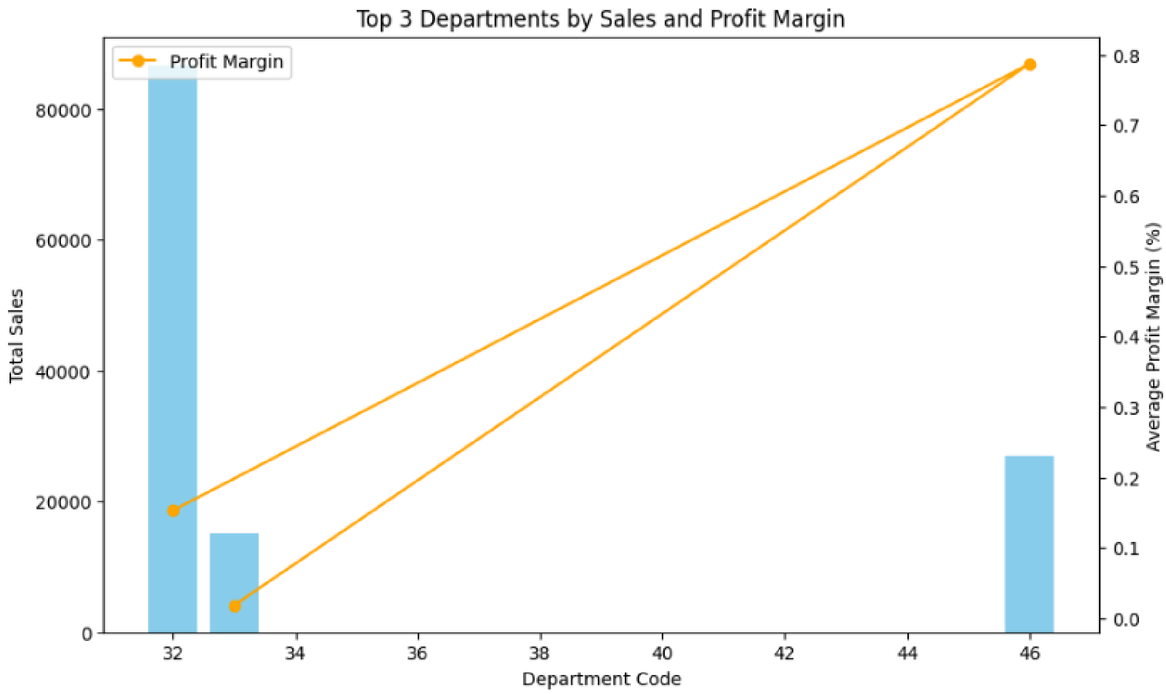
41626 rows × 1 columns

Determine which departments contribute the most to revenue or have the highest profit margins.

Group the data by department_code and calculate total sales and average profit margin for each department. Sort by total sales or profit margin to identify top-performing departments.

	department_code	total_sales	avg_profit_margin
9	32.0	86684.823699	0.153091
13	46.0	26999.293609	0.787093
10	33.0	15033.273166	0.018562

Visualization

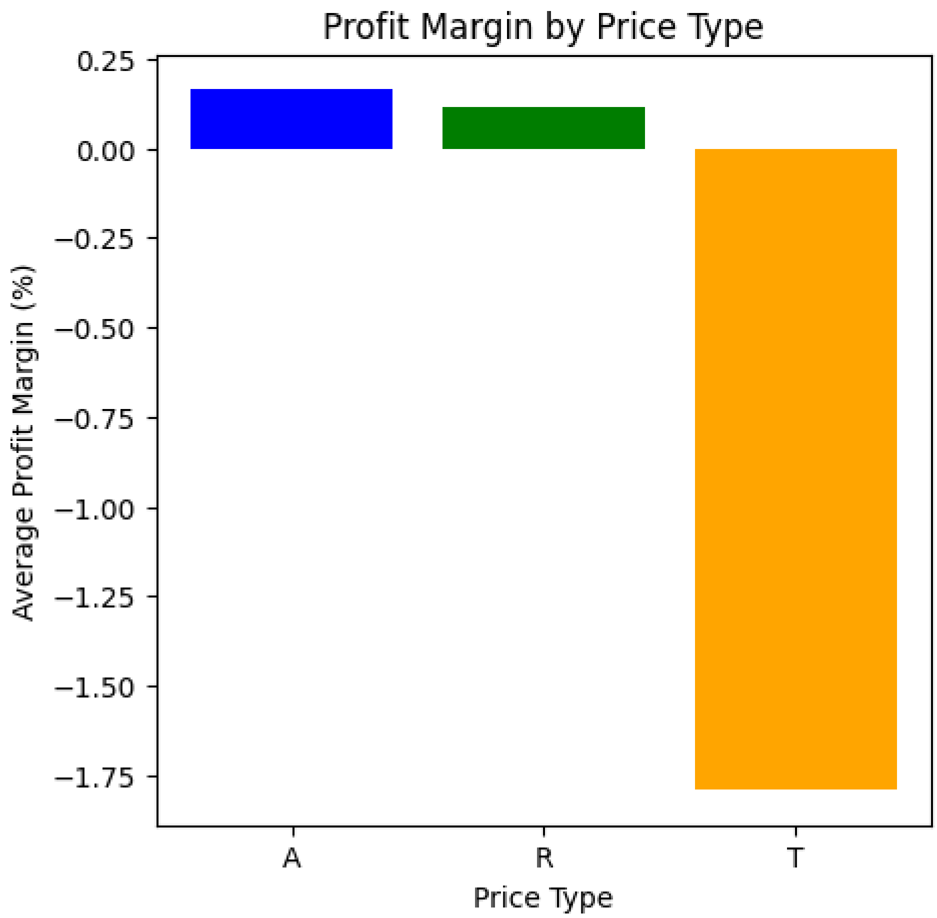


Insight: Price type impact on profit margin.

Let's group by data by price_type, and calculate the average profit margin for each category. Highlight which price type offers the best profability.

	price_type	avg_profit_margin
0	A	0.163857
1	R	0.113068
2	T	-1.791658

let's use a bar plot to compare avg_profit_margin between categories



Insight: Store performance analysis.

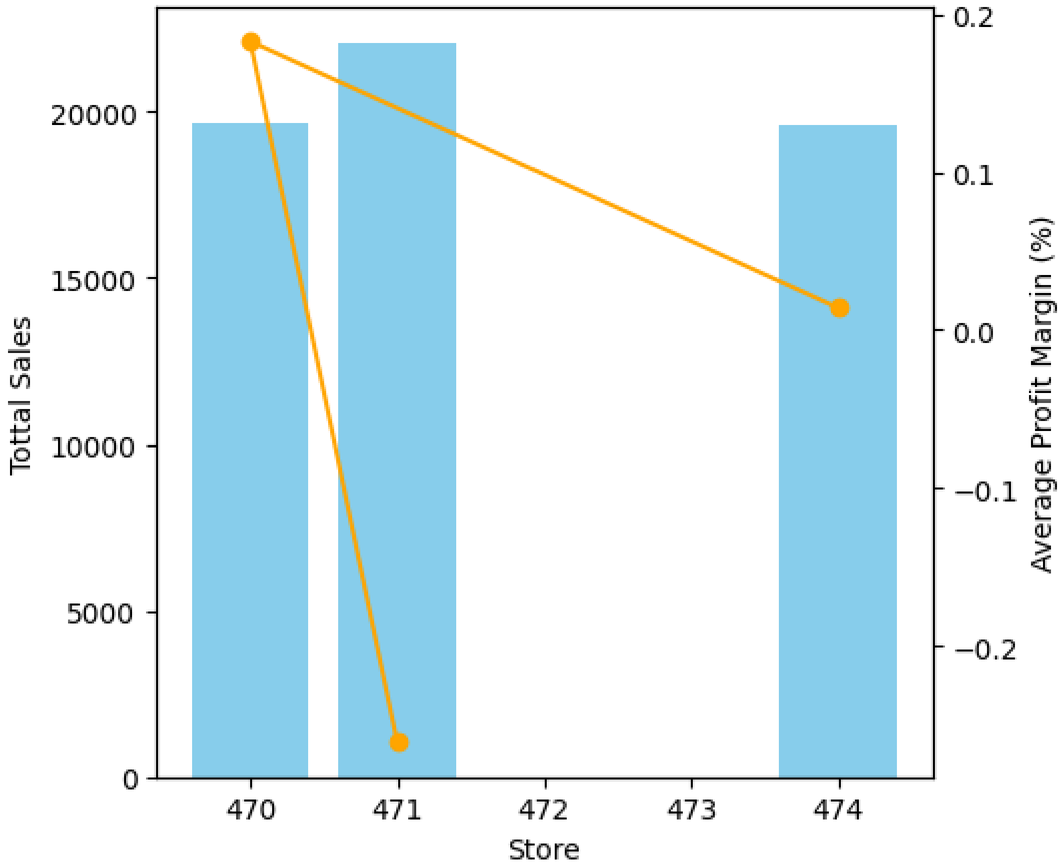
again the same thing, but with the store codes. Group data by store codes, and calculate total sales, and average profit_margin for each store.

	store	total_sales	avg_profit_margin
0	65.0	16317.194992	0.188662
1	66.0	12713.955801	-0.344004
2	461.0	16419.448643	0.297439
3	470.0	19632.200353	0.182724
4	471.0	22059.124237	-0.260607
5	472.0	17109.333962	0.118787
6	473.0	19423.933269	-0.140931
7	474.0	19601.441437	0.014012
8	477.0	17206.197161	-0.059575
9	493.0	17282.740586	0.034192

Top 3 stores

	store	total_sales	avg_profit_margin
4	471.0	22059.124237	-0.260607
3	470.0	19632.200353	0.182724
7	474.0	19601.441437	0.014012

Now let's visualize the data\



The task also included creating synthetic dataset representing daily total sales by store and department.

the dataset should include date, store, department code, daily_sales.

let's extract a dataset from our filtered_dataset

	date	store	department_code	daily_sales
0	2024-12-18	65.0	1.0	23.317371
1	2024-12-18	65.0	2.0	3.580000
2	2024-12-18	65.0	5.0	146.881599
3	2024-12-18	65.0	29.0	33.980000
4	2024-12-18	65.0	31.0	32.987200
...	...	...	...	...
922	2024-12-24	493.0	33.0	226.564228
923	2024-12-24	493.0	39.0	224.563885
924	2024-12-24	493.0	46.0	410.437941
925	2024-12-24	493.0	48.0	154.400000
926	2024-12-24	493.0	61.0	11.604914

927 rows × 4 columns

Because each store has products from various departments, counted them independently , daily sales for each department in each store.

.