In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
df=pd.read_csv("retail_sales_50k_messy.csv")
df.head()
```

Out[2]:

	order_id	order_date	city	region	category	product	quantity	unit_price	disc
0	100000	06/25/2023	Bengaluru	South	Beauty	Face Wash	1.0	11521.49	
1	100001	2024-05-17	Kolkata	East	Beauty	Face Wash	2.0	22032.78	
2	100002	NaN	Mumbai	West	Fashion	T-Shirt	-1.0	48903.88	
3	100003	2024-02-03	MUMBAI	NaN	Fashion	Jacket	1.0	13340.77	
4	100004	12/12/2023	Bangalore	NaN	Beauty	Moisturizer	3.0	3788.15	

In [3]:
```python
df.info()
df.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   order_id          50000 non-null  int64  
 1   order_date        37513 non-null  object 
 2   city              45501 non-null  object 
 3   region            31694 non-null  object 
 4   category          50000 non-null  object 
 5   product           50000 non-null  object 
 6   quantity          41829 non-null  float64
 7   unit_price        49484 non-null  float64
 8   discount_percent  41574 non-null  float64
 9   final_unit_price  34742 non-null  float64
 10  revenue           34742 non-null  float64
 11  payment_method    40117 non-null  object 
 12  customer_type     33383 non-null  object 
 13  sales_channel     50000 non-null  object 
dtypes: float64(5), int64(1), object(8)
memory usage: 5.3+ MB
```

Out[3]:
```
(50000, 14)
```

In [4]:
```python
df.isnull().sum()
```

Out[4]:
```
order_id                0
order_date          12487
city                 4499
region              18306
category                0
product                 0
quantity             8171
unit_price            516
discount_percent     8426
final_unit_price    15258
revenue             15258
payment_method       9883
customer_type       16617
sales_channel           0
dtype: int64
```

In [5]:
```python
df.duplicated().sum()
```

Out[5]:
```
np.int64(0)
```

In [6]:
```python
df.describe()
```

Out[6]:

	order_id	quantity	unit_price	discount_percent	final_unit_price	
count	50000.000000	41829.000000	49484.000000	41574.000000	34742.000000	3.47
mean	124939.148800	1.793756	27782.986800	9.966085	25046.888110	4.50
std	14466.366909	1.725868	32843.593955	7.093827	29643.317647	8.48
min	100000.000000	-1.000000	200.320000	0.000000	160.870000	-4.70
25%	112407.750000	1.000000	12622.932500	5.000000	11307.677500	5.42
50%	124932.500000	2.000000	25355.000000	10.000000	22725.705000	3.23
75%	137471.250000	3.000000	37914.737500	15.000000	33914.985000	7.42
max	149999.000000	4.000000	498326.780000	20.000000	498326.780000	1.70

In [8]:
```python
df.drop_duplicates(subset="order_id", inplace=True)
```

In [9]:
```python
df['city'] = df['city'].str.lower().str.strip()
df['city'] = df['city'].replace({
    'mumbai ': 'mumbai',
    'delhi ': 'delhi',
    'bangalore ': 'bangalore'
})
```

In [10]:
```python
df['order_date'] = pd.to_datetime(df['order_date'], errors='coerce')
```

In [11]:
```python
df = df[df['order_date'].notnull()]
```

In [13]:
```
df['quantity'].fillna(df['quantity'].median(), inplace=True)
df['final_unit_price'].fillna(df['final_unit_price'].median(), inplace=True)
df['discount_percent'].fillna(0, inplace=True)
df['customer_type'].fillna('unknown', inplace=True)
```

```
C:\Users\shubh\AppData\Local\Temp\ipykernel_8140\2715588028.py:1: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['quantity'].fillna(df['quantity'].median(), inplace=True)
C:\Users\shubh\AppData\Local\Temp\ipykernel_8140\2715588028.py:2: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['final_unit_price'].fillna(df['final_unit_price'].median(), inplace=True)
C:\Users\shubh\AppData\Local\Temp\ipykernel_8140\2715588028.py:3: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['discount_percent'].fillna(0, inplace=True)
C:\Users\shubh\AppData\Local\Temp\ipykernel_8140\2715588028.py:4: FutureWarning: A v
alue is trying to be set on a copy of a DataFrame or Series through chained assignme
nt using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['customer_type'].fillna('unknown', inplace=True)
```

In [14]:
```
df = df[df['quantity'] >0
```

```
In [16]: Q1 = df['final_unit_price'].quantile(0.25)
         Q3 = df['final_unit_price'].quantile(0.75)
         IQR = Q3 - Q1

         df = df[(df['final_unit_price'] >= Q1 - 1.5*IQR) &
                 (df['final_unit_price'] <= Q3 + 1.5*IQR)]
```

```
In [19]: df['revenue'] = df['quantity'] * df['final_unit_price'] * (1 - df['discount_percent
```

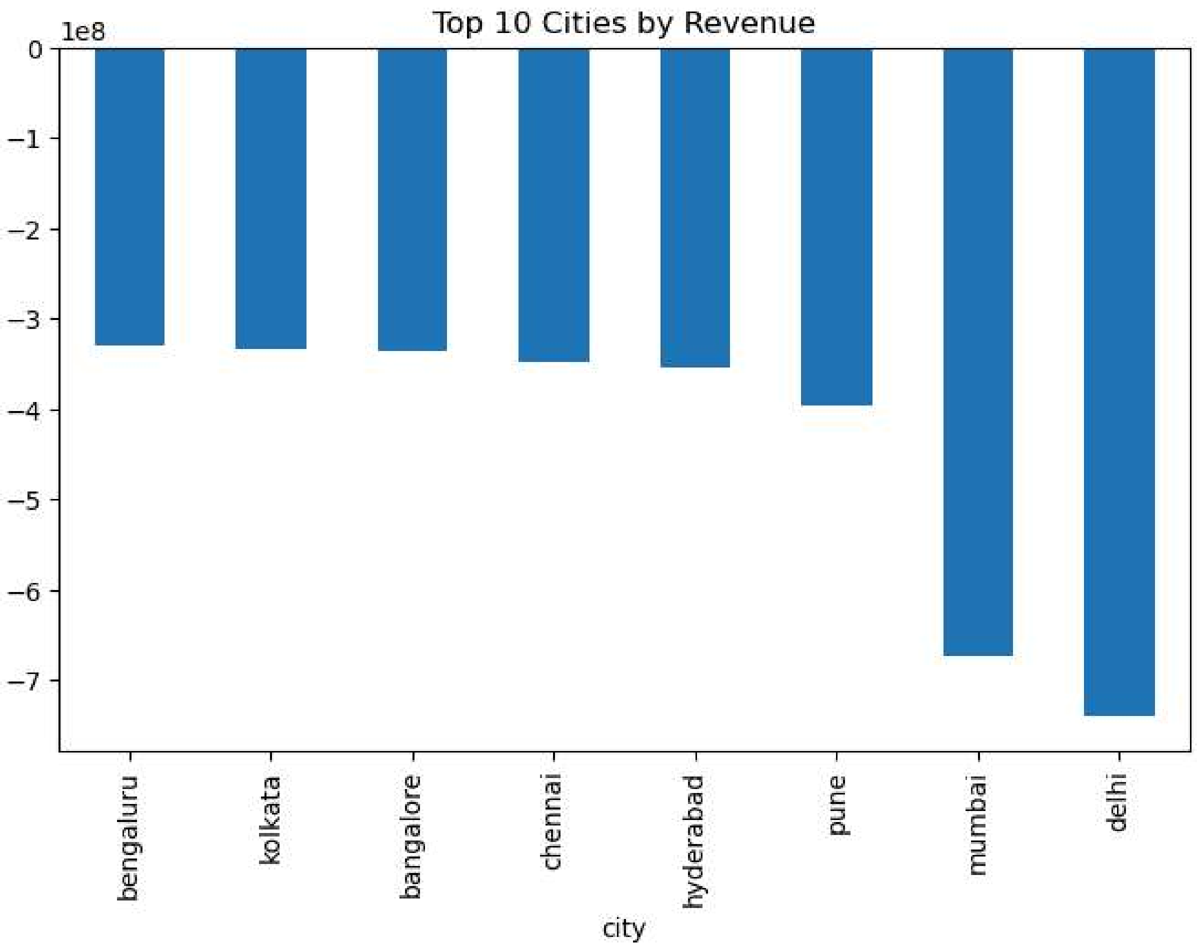
```
In [20]: df['revenue'].sum()
```

```
Out[20]: np.float64(-3836330085.55)
```

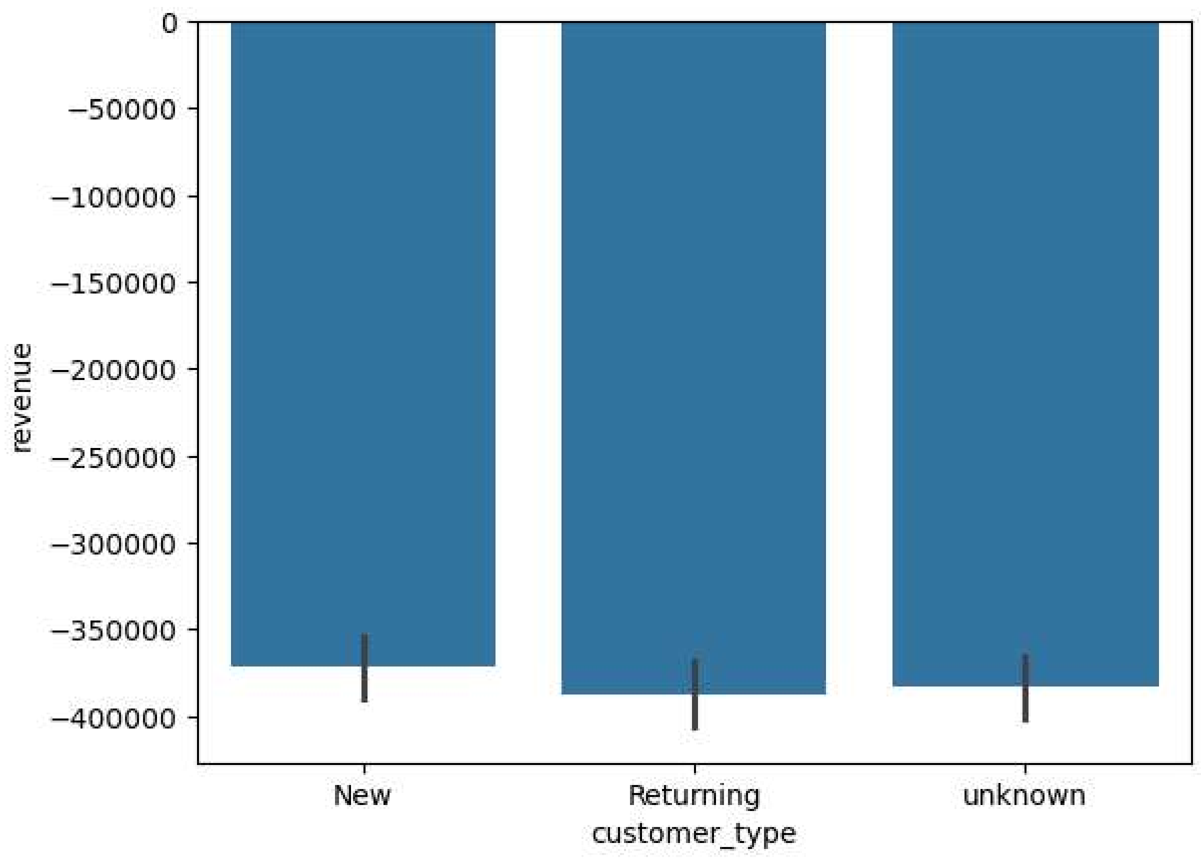
```
In [21]: city_sales = df.groupby('city')['revenue'].sum().sort_values(ascending=False)

         city_sales.head()
```

```
Out[21]: city
         bengaluru   -3.309103e+08
         kolkata     -3.335179e+08
         bangalore   -3.353444e+08
         chennai     -3.485642e+08
         hyderabad   -3.538006e+08
         Name: revenue, dtype: float64
```

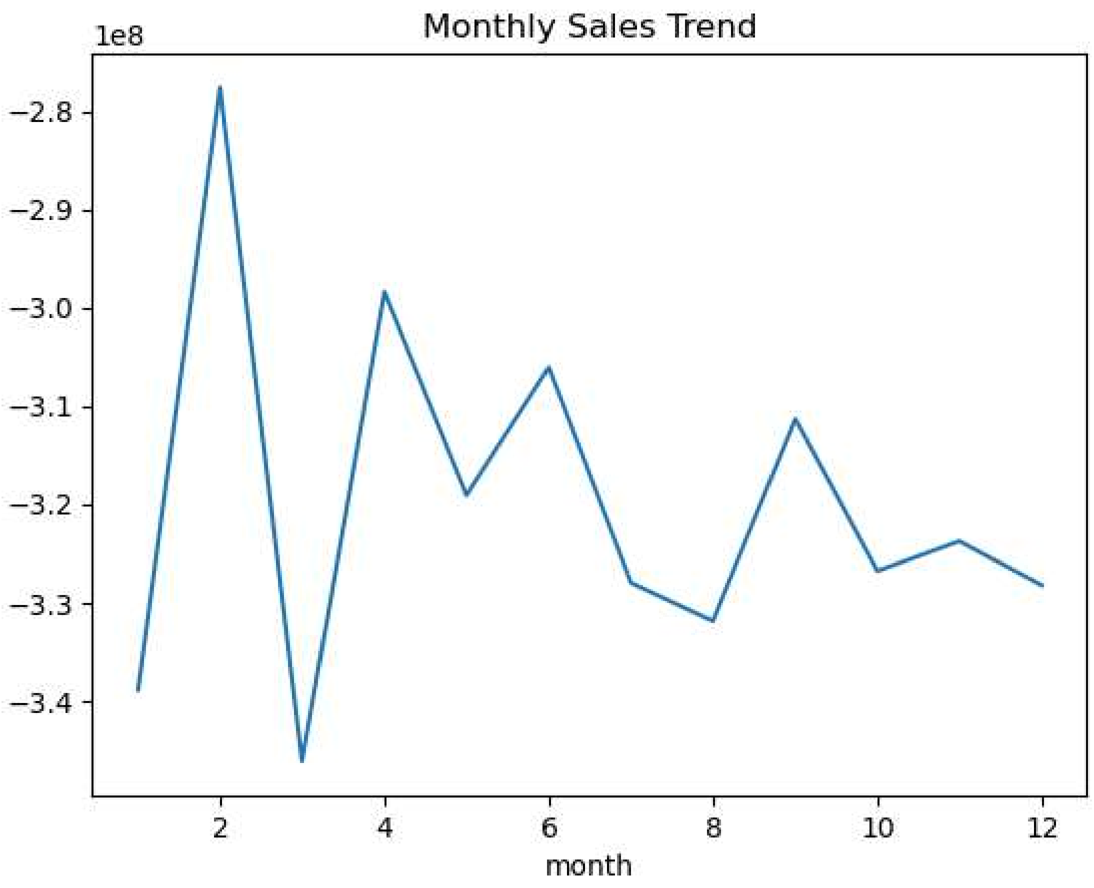
```
In [22]: city_sales.head(10).plot(kind='bar', figsize=(8,5))
         plt.title("Top 10 Cities by Revenue")
         plt.show()
```



```
In [23]: sns.barplot(x='customer_type', y='revenue', data=df)
         plt.show()
```



In [24]:
```
df['month'] = df['order_date'].dt.month

monthly_sales = df.groupby('month')['revenue'].sum()
monthly_sales.plot()
plt.title("Monthly Sales Trend")
plt.show()
```



```
In [25]: df.to_csv("cleaned_retail_sales.csv", index=False)
```

```
In [ ]:
```