```
# install
!pip install pmdarima
!pip install surprise
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata (7.8 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.2.2)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.6.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.13.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.2.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (75.1.0)
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima) (3.5.
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (1.0.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=0.19->pmdarima)
Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 20.1 MB/s eta 0:00:00
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
Collecting surprise
  Downloading surprise-0.1-py2.py3-none-any.whl.metadata (327 bytes)
Collecting scikit-surprise (from surprise)
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 3.1 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.13.1)
Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp310-cp310-linux_x86_64.whl size=2357267 sha256=49d4af19eab6ce641af
  Stored in directory: /root/.cache/pip/wheels/4b/3f/df/6acbf0a40397d9bf3ff97f582cc22fb9ce66adde75bc71fd54
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.4 surprise-0.1
```

```
# Import Packages
import pandas as pd
import numpy as np


# Reading the dataset
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
csvFile = "/content/gdrive/MyDrive/Colab Notebooks/kz.csv"

# reading the csv file to different variables
df_data = pd.read_csv(csvFile)
data2 = pd.read_csv(csvFile)
data = pd.read_csv(csvFile)
data3 = pd.read_csv(csvFile)
```

```
Mounted at /content/gdrive
```

```
# Loading the dataset
df_data['event_time'] = df_data['event_time'].replace(" UTC","", regex=True)
df_data['event_time'] = pd.to_datetime(df_data['event_time'])
df_data
```

	event_time	order_id	product_id	category_id	category_code	brand	price	user_id
0	2020-04-24 11:50:39	2294359932054536986	1515966223509089906	2.268105e+18	electronics.tablet	samsung	162.01	1.515916e+18
1	2020-04-24 11:50:39	2294359932054536986	1515966223509089906	2.268105e+18	electronics.tablet	samsung	162.01	1.515916e+18
2	2020-04-24 14:37:43	2294444024058086220	2273948319057183658	2.268105e+18	electronics.audio.headphone	huawei	77.52	1.515916e+18
3	2020-04-24 14:37:43	2294444024058086220	2273948319057183658	2.268105e+18	electronics.audio.headphone	huawei	77.52	1.515916e+18
4	2020-04-24 19:16:21	2294584263154074236	2273948316817424439	2.268105e+18	NaN	karcher	217.57	1.515916e+18
...	...	...	...	...	...	...	...	...
2633516	2020-11-21 10:10:01	2388440981134693942	1515966223526602848	2.268105e+18	electronics.smartphone	oppo	138.87	1.515916e+18
2633517	2020-11-21 10:10:13	2388440981134693943	1515966223509089282	2.268105e+18	electronics.smartphone	apple	418.96	1.515916e+18

```
# Data transaformation
# Split the categories into sub categories
df_data[['l1_cat', 'l2_cat', 'l3_cat']] = df_data['category_code'].str.split('.',expand=True)

df_data
```

	event_time	order_id	product_id	category_id	category_code	brand	price	user_id	
0	2020-04-24 11:50:39	2294359932054536986	1515966223509089906	2.268105e+18	electronics.tablet	samsung	162.01	1.515916e+18	ele
1	2020-04-24 11:50:39	2294359932054536986	1515966223509089906	2.268105e+18	electronics.tablet	samsung	162.01	1.515916e+18	ele
2	2020-04-24 14:37:43	2294444024058086220	2273948319057183658	2.268105e+18	electronics.audio.headphone	huawei	77.52	1.515916e+18	ele
3	2020-04-24 14:37:43	2294444024058086220	2273948319057183658	2.268105e+18	electronics.audio.headphone	huawei	77.52	1.515916e+18	ele
4	2020-04-24 19:16:21	2294584263154074236	2273948316817424439	2.268105e+18	NaN	karcher	217.57	1.515916e+18	
...	...	...	...	...	...	...	...	...	
2633516	2020-11-21 10:10:01	2388440981134693942	1515966223526602848	2.268105e+18	electronics.smartphone	oppo	138.87	1.515916e+18	ele
2633517	2020-11-21 10:10:13	2388440981134693943	1515966223509089282	2.268105e+18	electronics.smartphone	apple	418.96	1.515916e+18	ele

```
# Describe Data + some data cleaning
df_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2633521 entries, 0 to 2633520
Data columns (total 11 columns):
 #   Column         Dtype
---  ------         -----
 0   event_time     datetime64[ns]
 1   order_id       int64
 2   product_id     int64
 3   category_id    float64
 4   category_code  object
 5   brand          object
 6   price          float64
 7   user_id        float64
 8   l1_cat         object
 9   l2_cat         object
 10  l3_cat         object
dtypes: datetime64[ns](1), float64(3), int64(2), object(5)
memory usage: 221.0+ MB
```

```
df_data.describe()
```

	event_time	order_id	product_id	category_id	price	user_id
count	2633521	2.633521e+06	2.633521e+06	2.201567e+06	2.201567e+06	5.641690e+05
mean	2020-01-16 19:57:05.412119808	2.361783e+18	1.674080e+18	2.273827e+18	1.540932e+02	1.515916e+18
min	1970-01-01 00:33:40	2.294360e+18	1.515966e+18	2.268105e+18	0.000000e+00	1.515916e+18
25%	2020-03-05 15:42:44	2.348807e+18	1.515966e+18	2.268105e+18	1.456000e+01	1.515916e+18
50%	2020-06-08 08:33:27	2.353254e+18	1.515966e+18	2.268105e+18	5.553000e+01	1.515916e+18
75%	2020-08-24 06:52:14	2.383131e+18	1.515966e+18	2.268105e+18	1.967400e+02	1.515916e+18
max	2020-11-21 10:10:30	2.388441e+18	2.388434e+18	2.374499e+18	5.092590e+04	1.515916e+18
std	NaN	1.716538e+16	3.102249e+17	2.353247e+16	2.419421e+02	2.379057e+07

```
# calculate percentage of missing value
df_data.isnull().sum()/len(df_data)
```

	0
event_time	0.000000
order_id	0.000000
product_id	0.000000
category_id	0.164021
category_code	0.232465
brand	0.192140
price	0.164021
user_id	0.785774
l1_cat	0.232465
l2_cat	0.232465
l3_cat	0.626633

dtype: float64

```
print("Min Date: ", df_data['event_time'].min())
print("Max Date: ", df_data['event_time'].max())
```

```
Min Date:  1970-01-01 00:33:40
Max Date:  2020-11-21 10:10:30
```

```
df_data['year'] = df_data['event_time'].dt.year
df_data.groupby(['year'])['year'].count()
```

year	year
1970	19631
2020	2613890

dtype: int64

```
df_data = df_data[df_data['year'] != 1970].reset_index(drop=True)
```

```
df_data = df_data[df_data['user_id'].notna()].reset_index(drop=True)
```

```
# Handling missing price values
products_median_prices = df_data.groupby(['product_id'])['price'].median().reset_index()
products_median_prices = pd.Series(df_data['price'].values, index=df_data['product_id']).to_dict()

# fill in missing prices with the median
df_data['price'] = df_data['price'].fillna(df_data['product_id'].map(products_median_prices))
df_data['price'].isna().sum()
```

```
0
```

```
Start coding or generate with AI.
```

```
# Import libraries - 3
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

# suppress scientific notation
np.set_printoptions(suppress=True)
pd.options.display.float_format = '{:.2f}'.format
```

```
data.head()
```

	event_time	order_id	product_id	category_id	category_code	brand	price	
0	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2268105426648171008.00	electronics.tablet	samsung	162.01	151591562544
1	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2268105426648171008.00	electronics.tablet	samsung	162.01	151591562544

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2633521 entries, 0 to 2633520
Data columns (total 8 columns):
 #   Column         Dtype
---  ------         -----
 0   event_time     object
 1   order_id       int64
 2   product_id     int64
 3   category_id    float64
 4   category_code  object
 5   brand          object
 6   price          float64
 7   user_id        float64
dtypes: float64(3), int64(2), object(3)
memory usage: 160.7+ MB
```

```
data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	
event_time	2633521	1316174	1970-01-01 00:33:40 UTC	19631	NaN	NaN	NaN	
order_id	2633521.00	NaN	NaN	NaN	2361782829757762048.00	17165379778976542.00	2294359932054536960.00	23
product_id	2633521.00	NaN	NaN	NaN	1674080384807513600.00	310224921942725248.00	1515966223509088512.00	15
category_id	2201567.00	NaN	NaN	NaN	2273827014269330176.00	23532467048797852.00	2268105388421284352.00	23
category_code	2021319	510	electronics.smartphone	357682	NaN	NaN	NaN	
brand	2127516	23021	samsung	358928	NaN	NaN	NaN	
price	2201567.00	NaN	NaN	NaN	154.09	241.94	0.00	
user_id	564169.00	NaN	NaN	NaN	1515915625486184960.00	23790565.29	1515915625439952128.00	15

```
# Data preprocessing
data = data.drop_duplicates()
```

```
data.isnull().sum()
```

	0
event_time	0
order_id	0
product_id	0
category_id	431953
category_code	612053
brand	505965
price	431953
user_id	2069351

dtype: int64

```
# Filter rows where 'event_time' contains '1970'
filtered_df = data[data['event_time'].str.contains('1970')]

filtered_df
```

	event_time	order_id	product_id	category_id	category_code	brand	price	use
28813	1970-01-01 00:33:40 UTC	2340102742254551453	1515966223509354098	2268105644970082560.00	NaN	pastel	53.22	
28814	1970-01-01 00:33:40 UTC	2340102742439100830	1515966223509117074	2268105427872907776.00	NaN	samsung	30.07	
28815	1970-01-01 00:33:40 UTC	2340102742439100830	1515966223509089955	2268105441009468160.00	appliances.kitchen.meat_grinder	moulinex	57.85	
28816	1970-01-01 00:33:40 UTC	2340102742439100830	1515966223509297118	2268105392925967104.00	appliances.environment.air_heater	ava	48.59	
28817	1970-01-01 00:33:40 UTC	2340102742439100830	1515966223509088552	2268105428166509056.00	electronics.smartphone	samsung	196.27	
...	...	...	...	...	...	...	...	

```
# exclude rows where 'event_time' contains '1970'
data = data[~data['event_time'].str.contains('1970')]
data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	
event_time	2613215	1316173	2020-04-09 16:30:01 UTC	349	NaN	NaN	NaN	
order_id	2613215.00	NaN	NaN	NaN	2361898823701765632.00	17170329471123940.00	2294359932054536960.00	2
product_id	2613215.00	NaN	NaN	NaN	1673993996657965568.00	310177500587615040.00	1515966223509088512.00	1
category_id	2185340.00	NaN	NaN	NaN	2273821966023597056.00	23520137742089068.00	2268105388421284352.00	2
category_code	2006141	509	electronics.smartphone	354747	NaN	NaN	NaN	
brand	2111921	22955	samsung	356346	NaN	NaN	NaN	
price	2185340.00	NaN	NaN	NaN	154.19	242.02	0.00	
user_id	562188.00	NaN	NaN	NaN	1515915625486215168.00	23805708.42	1515915625439952128.00	1

```
# remove empty rows in brand and user_id columns
data = data.dropna(subset=['brand', 'user_id']).reset_index(drop=True)
data.isnull().sum()
```

	0
event_time	0
order_id	0
product_id	0
category_id	0
category_code	115675
brand	0
price	0
user_id	0

dtype: int64

```
# split the category_code column into category and product columns
data[['category', 'product']] = data['category_code'].str.split('.', n=1, expand=True)

# fill empty cells in the new category and product columns with unknown
data['category'].fillna('unknown', inplace=True)
data['product'].fillna('unknown', inplace=True)

# drop the category_code column
data.drop('category_code', axis=1, inplace=True)

data.tail()
```

	event_time	order_id	product_id	category_id	brand	price	user_id	category
535060	2020-11-21 10:10:01 UTC	2388440981134693942	1515966223526602848	2268105428166509056.00	oppo	138.87	1515915625514888704.00	electronics
535061	2020-11-21 10:10:13 UTC	2388440981134693943	1515966223509089282	2268105428166509056.00	apple	418.96	1515915625514891264.00	electronics

```
data['category_id'] = data['category_id'].astype('int64')
data['user_id'] = data['user_id'].astype('int64')


# remove UTC from event_time
data['event_time'] = data['event_time'].str.replace('UTC', '')

# create date column
data['date'] = data.event_time.apply(lambda x: x.split(' ')[0])

# convert to datetime object
data['date'] = pd.to_datetime(data['date'])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 535065 entries, 0 to 535064
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   event_time   535065 non-null  object
 1   order_id     535065 non-null  int64
 2   product_id   535065 non-null  int64
 3   category_id  535065 non-null  int64
 4   brand        535065 non-null  object
 5   price        535065 non-null  float64
 6   user_id      535065 non-null  int64
 7   category     535065 non-null  object
 8   product      535065 non-null  object
 9   date         535065 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(4), object(4)
memory usage: 40.8+ MB
```

```
data.head()
```

	event_time	order_id	product_id	category_id	brand	price	user_id	category	
0	2020-04-24 11:50:39	2294359932054536986	1515966223509089906	2268105426648171008	samsung	162.01	1515915625441993984	electronics	
1	2020-04-24 14:37:43	2294444024058086220	2273948319057183658	2268105430162997760	huawei	77.52	1515915625447879424	electronics	audio.he
2	2020-04-24 19:16:21	2294584263154074236	2273948316817424439	2268105471367840000	karcher	217.57	1515915625443148032	unknown	

```
# create new columns - date, month, hour and day_of_week
data['month'] = data.date.dt.strftime('%b')
data['month_num'] = data.date.dt.month

data['week_day'] = data.date.dt.strftime('%a')
data['week_day_num'] = data.date.apply(lambda x: x.strftime('%w')).astype('int64')

data['hour'] = data.event_time.apply(lambda x: x.split(' ')[1].split(':')[0]).astype('int64')

del data['event_time']

data.head()
```

	order_id	product_id	category_id	brand	price	user_id	category	product	da
0	2294359932054536986	1515966223509089906	2268105426648171008	samsung	162.01	1515915625441993984	electronics	tablet	202 04-
1	2294444024058086220	2273948319057183658	2268105430162997760	huawei	77.52	1515915625447879424	electronics	audio.headphone	202 04-
2	2294584263154074236	2273948316817424439	2268105471367840000	karcher	217.57	1515915625443148032	unknown	unknown	202 04-
3	2295716521449619559	1515966223509261697	2268105442636858112	maestro	39.33	1515915625450382848	furniture	kitchen.table	202 04-
4	2295740594749702229	1515966223509104892	2268105428166509056	apple	1387.01	1515915625448766464	electronics	smartphone	202 04-

```
data.describe().T
```

	count	mean	min	25%	50%	
order_id	535065.00	2370567694359658496.00	2294359932054536960.00	2354295804434317824.00	2376796926830969856.00	238844098113459
product_id	535065.00	1692647560995175168.00	1515966223509088512.00	1515966223509104896.00	1515966223509261824.00	151596622352732
category_id	535065.00	2273071589655354368.00	2268105388421284352.00	2268105406549066752.00	2268105428166509056.00	226810543932335
price	535065.00	214.71	0.00	24.98	99.51	
user_id	535065.00	1515915625486156800.00	1515915625439952128.00	1515915625466993664.00	1515915625486697984.00	151591562551153
date	535065	2020-08-07 01:08:19.383812608	2020-01-05 00:00:00	2020-06-28 00:00:00	2020-08-16 00:00:00	2020-10-09
month_num	535065.00	7.73	1.00	6.00	8.00	
week_day_num	535065.00	3.03	0.00	1.00	3.00	
hour	535065.00	9.54	0.00	6.00	9.00	

```
# General Analysis
# total unique users
total_users = data['user_id'].nunique()
print(f'The total unique users in the dataset are: {total_users: 0,}')
```

```
The total unique users in the dataset are:  97,098
```

```
# total unique orders
total_orders = data['order_id'].nunique()
print(f'The total unique orders in the dataset are: {total_orders: 0,}')
```

```
The total unique orders in the dataset are:  388,742
```

```
# total sales
total_sales = round(data['price'].sum())
print(f'The total sales in $ is: {total_sales: 0,}')
```

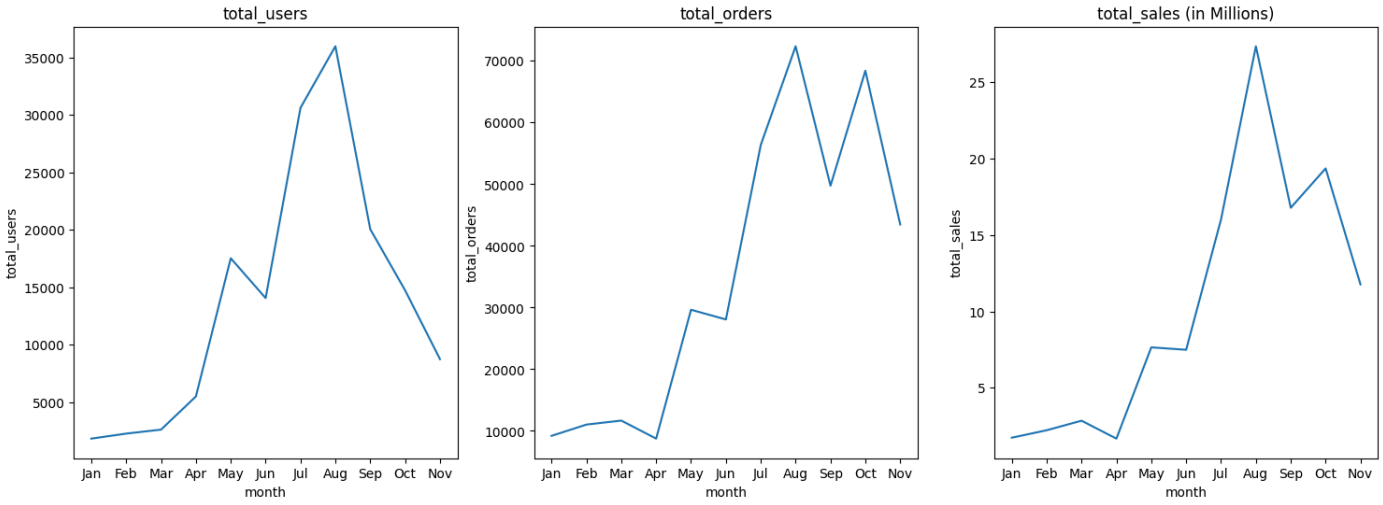
```
The total sales in $ is:  114,881,330
```

```
# Time Analysis
df_month = data.groupby(['month_num', 'month']).agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales= ('price', 'sum')
).sort_values(by='month_num', ascending=True).reset_index(level='month_num', drop=True)

df_month = df_month.reset_index()
df_month
```

	month	total_users	total_orders	total_sales
0	Jan	1823	9201	1729464.93
1	Feb	2259	11026	2216672.31
2	Mar	2606	11676	2841015.58
3	Apr	5495	8752	1669080.19
4	May	17527	29644	7644255.82
5	Jun	14059	28073	7486680.81
6	Jul	30628	56363	16019735.90
7	Aug	35989	72370	27362298.79
8	Sep	20062	49759	16785757.14
9	Oct	14736	68405	19361987.48
10	Nov	8744	43473	11764381.12

Next steps: Generate code with df_month | View recommended plots | New interactive sheet

```
plt.figure(figsize=(18, 6))

for i, col in enumerate(df_month.columns):
  if col != 'month':
    fig = plt.subplot(1, 3, i)
    sns.lineplot(data=df_month, x='month', y=col, )
    plt.title(f'{col}')

    if col == 'total_sales':
      fig.get_yaxis().set_major_formatter(matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

      # after plotting the data, format the labels
      current_values = plt.gca().get_yticks()

      plt.gca().set_yticklabels(['{:,.0f}'.format(x/1000000) for x in current_values])
      plt.title(f'{col} (in Millions)')

plt.show()
```



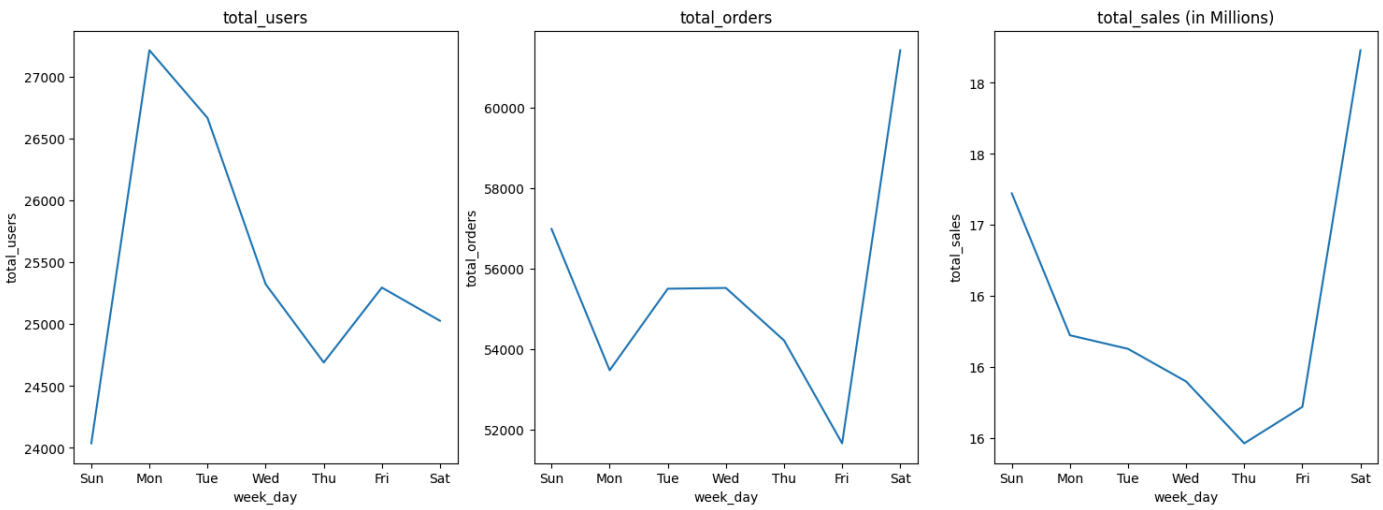

```
# Weekly Analysis
df_week = data.groupby(['week_day_num', 'week_day']).agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales=('price', 'sum')
).sort_values(by='week_day_num', ascending=True).reset_index(level='week_day_num', drop=True)

df_week = df_week.reset_index()
df_week
```

	week_day	total_users	total_orders	total_sales
0	Sun	24037	56979	17222096.04
1	Mon	27212	53470	16223122.63
2	Tue	26665	55495	16127420.47
3	Wed	25323	55514	15897601.88
4	Thu	24690	54211	15462231.02
5	Fri	25295	51655	15719070.48
6	Sat	25027	61418	18229787.55

Next steps: Generate code with df_week | View recommended plots | New interactive sheet

```
plt.figure(figsize=(18, 6))

for i, col in enumerate(df_week.columns):

    if col != 'week_day':

        fig = plt.subplot(1, 3, i)
        sns.lineplot(data=df_week, x='week_day', y=col, )
        plt.title(f'{col}')

        if col == 'total_sales':

            fig.get_yaxis().set_major_formatter(
            matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

            # after plotting the data, format the labels
            current_values = plt.gca().get_yticks()

            plt.gca().set_yticklabels(['{:,.0f}'.format(x/1000000) for x in current_values])
            plt.title(f'{col} (in Millions)')
```

```
plt.show()
```




```
# Daily analysis
df_date = data.groupby('date').agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales=('price', 'sum')
).reset_index()

df_date
```

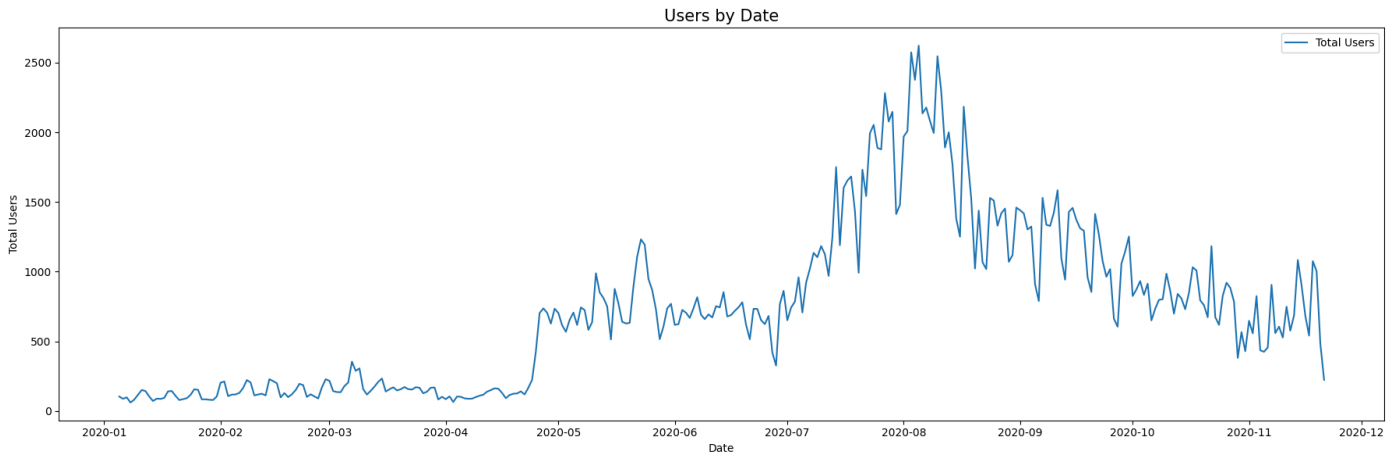
	date	total_users	total_orders	total_sales
0	2020-01-05	105	426	77216.52
1	2020-01-06	89	411	61146.03
2	2020-01-07	99	497	82381.21
3	2020-01-08	62	300	56344.74
4	2020-01-09	83	319	58278.71
...	...	...	...	...
317	2020-11-17	542	711	274243.17
318	2020-11-18	1076	5231	1147987.30
319	2020-11-19	1004	5012	1124056.05
320	2020-11-20	485	573	266572.62
321	2020-11-21	225	250	117368.16

322 rows × 4 columns

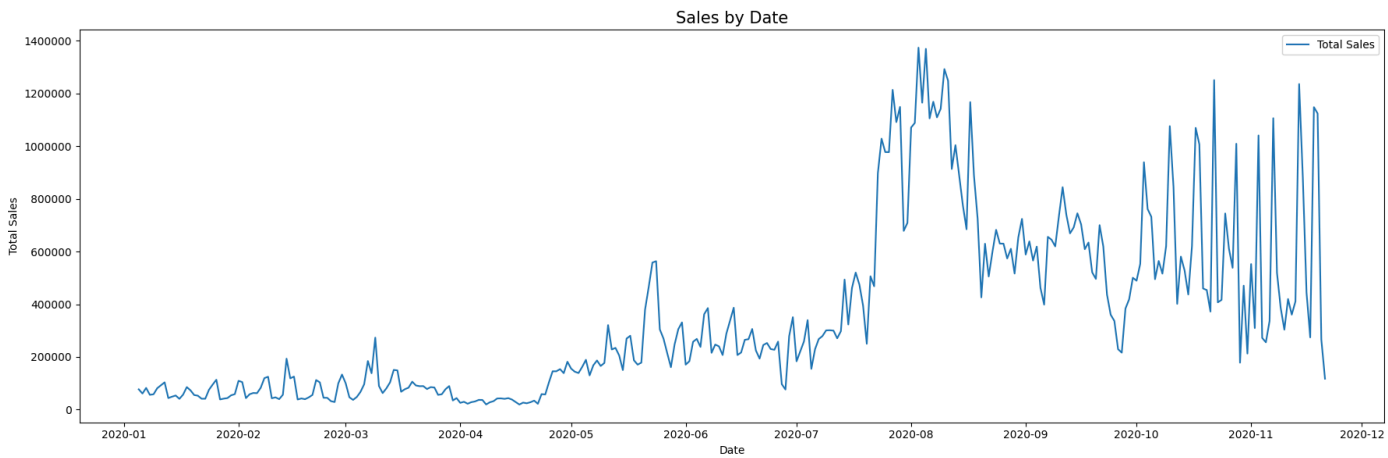
Next steps: Generate code with df_date | View recommended plots | New interactive sheet

```
fig, ax = plt.subplots(figsize=(18, 6))
xticks = df_date.date

ax.plot(df_date.date, df_date['total_users'])

ax.set_xlabel('Date')
ax.set_ylabel('Total Users')

ax.legend(['Total Users'])
plt.title('Users by Date', fontsize=15)
```

```
plt.tight_layout()
plt.show()
```

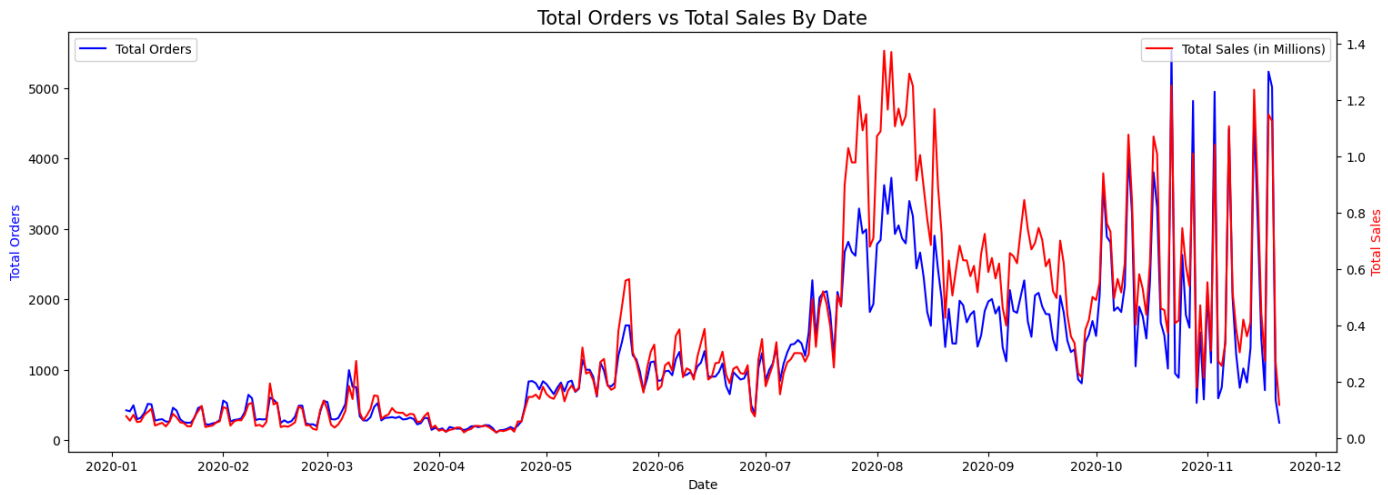



```
fig, ax = plt.subplots(figsize=(18, 6))
plt.ticklabel_format(style='plain')
xticks = df_date.date

ax.plot(df_date.date, df_date['total_sales'])

ax.set_xlabel('Date')
ax.set_ylabel('Total Sales')

ax.legend(['Total Sales'])
plt.title('Sales by Date', fontsize=15)

plt.tight_layout()
plt.show()
```




```
# Add a second y-axis
fig, ax1 = plt.subplots(figsize=(18, 6))
xticks = df_date.date

ax2 = ax1.twinx()
ax1.plot(df_date.date, df_date['total_orders'], color='b')
ax2.plot(df_date.date, df_date['total_sales'], color='r')
```

```
# format the second y-axis labels
ax2.get_yaxis().set_major_formatter(
matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels(['{:,.1f}'.format(x/1000000) for x in current_values])

ax1.set_xlabel('Date')
ax1.set_ylabel('Total Orders', color='b')
ax2.set_ylabel('Total Sales', color='r')

ax1.legend(['Total Orders'], loc='upper left')
ax2.legend(['Total Sales (in Millions)'], loc='upper right')

plt.title('Total Orders vs Total Sales By Date', fontsize=15)

plt.show()
```




```
# Customer Analysis
# average customer spending
avg_user_rev = round(data['price'].sum() / data['user_id'].nunique())
print(f'The Average Customer Sales in $: {avg_user_rev: 0,}')
```

```
The Average Customer Sales in $:  1,183
```

```
# average customer orders
avg_user_orders = round(data['order_id'].nunique() / data['user_id'].nunique(), 0)
print(f'The average customer orders are: {avg_user_orders}')
```

```
The average customer orders are: 4.0
```

```
# average order value
avg_order_value = round(data['price'].sum() / data['order_id'].nunique())
print(f'The average order value in $: {avg_order_value}')
```

```
The average order value in $: 296
```

```
# top 20 customers
top_20_customers = data.groupby('user_id').agg(
    total_orders=('order_id', 'count'),
    total_sales=('price', 'sum')
).reset_index().sort_values('total_orders', ascending=False).head(20)
top_20_customers
```

	user_id	total_orders	total_sales
92893	1515915625512763648	603	120965.01
92894	1515915625512763904	597	109908.68
91873	1515915625512422656	553	86242.12
93042	1515915625512817152	551	101644.54
90965	1515915625512118016	543	80672.60
90968	1515915625512118784	533	65778.93
91082	1515915625512155136	531	53172.39
91737	1515915625512376576	530	60860.53
90851	1515915625512084480	529	83895.33
90966	1515915625512118272	529	70619.40
91871	1515915625512422144	529	55012.37
90963	1515915625512117504	527	56608.55
90853	1515915625512084992	526	76463.18
91079	1515915625512154368	523	59301.30
91742	1515915625512377856	523	60585.40
91745	1515915625512378624	522	65813.57
91738	1515915625512376832	519	65520.74
93043	1515915625512817408	519	94095.53
94170	1515915625513284864	519	78927.92
90964	1515915625512117760	514	61345.83

Next steps: Generate code with top_20_customers View recommended plots New interactive sheet

```
# Product Analysis
# top 10 categories
print('Unique Categories', data['category'].nunique())
```

```
Unique Categories 14
```

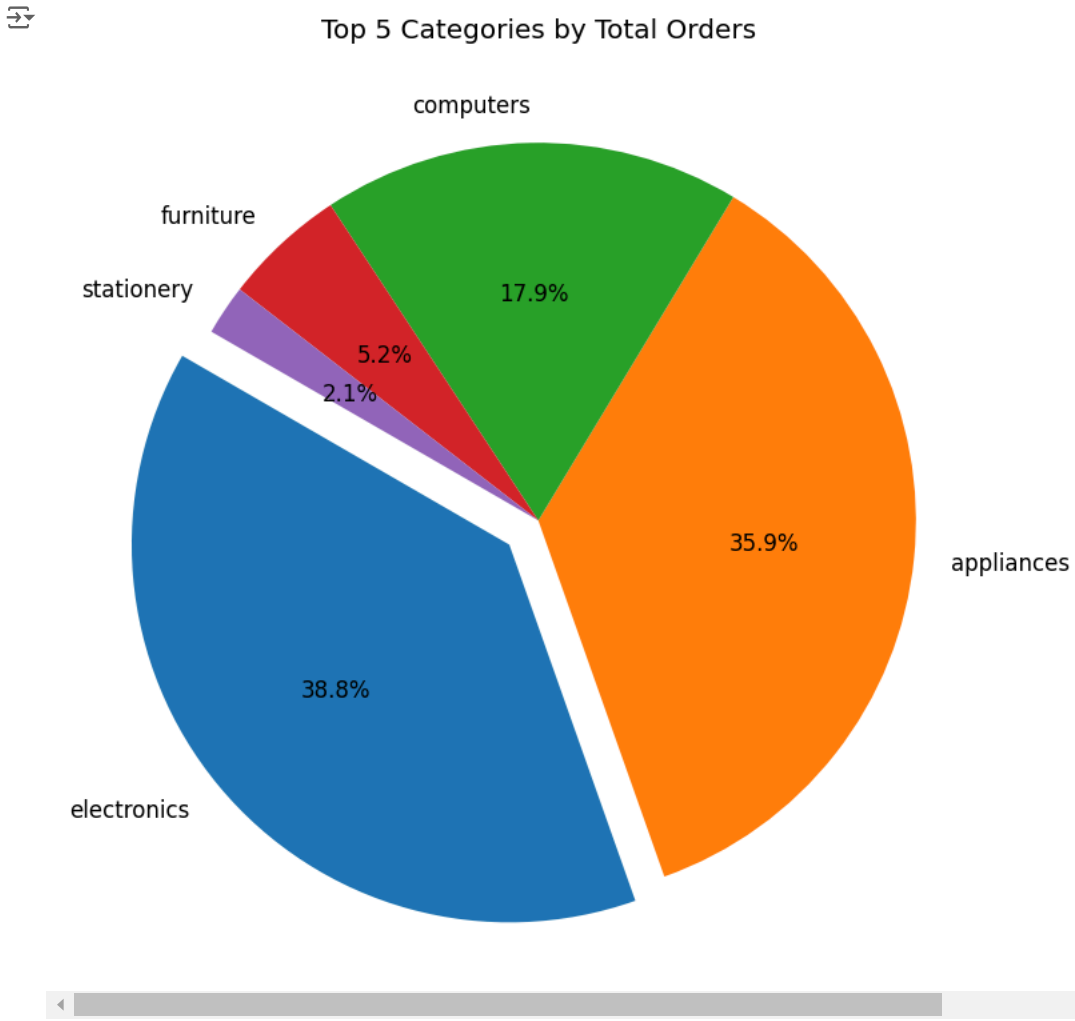
```
# by total orders
orders_category = data[data['category'] != 'unknown'].groupby('category').agg(
    total_orders=('order_id', 'count')
).reset_index().sort_values('total_orders', ascending=False).reset_index(drop=True)
orders_category.head(10)
```

	category	total_orders
0	electronics	156556
1	appliances	145217
2	computers	72378
3	furniture	21182
4	stationery	8676
5	construction	3959
6	accessories	3019
7	apparel	2664
8	kids	2275
9	auto	1366

Next steps: Generate code with orders_category View recommended plots New interactive sheet

```
# plot the pie plot for the top 5 categories by orders
plt.rcParams.update({'font.size': 12, 'figure.facecolor': 'white'})
# extract the top 5 categories as labels
```

```
labels = orders_category['category'][:5]
#only explode the 1st slice i.e. electronics
explode = (0.1, 0, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=orders_category.total_orders[:5], explode=explode, labels=labels, autopct='%1.1f%%', startangle=150)

plt.title('Top 5 Categories by Total Orders')
plt.tight_layout()
plt.show()
```

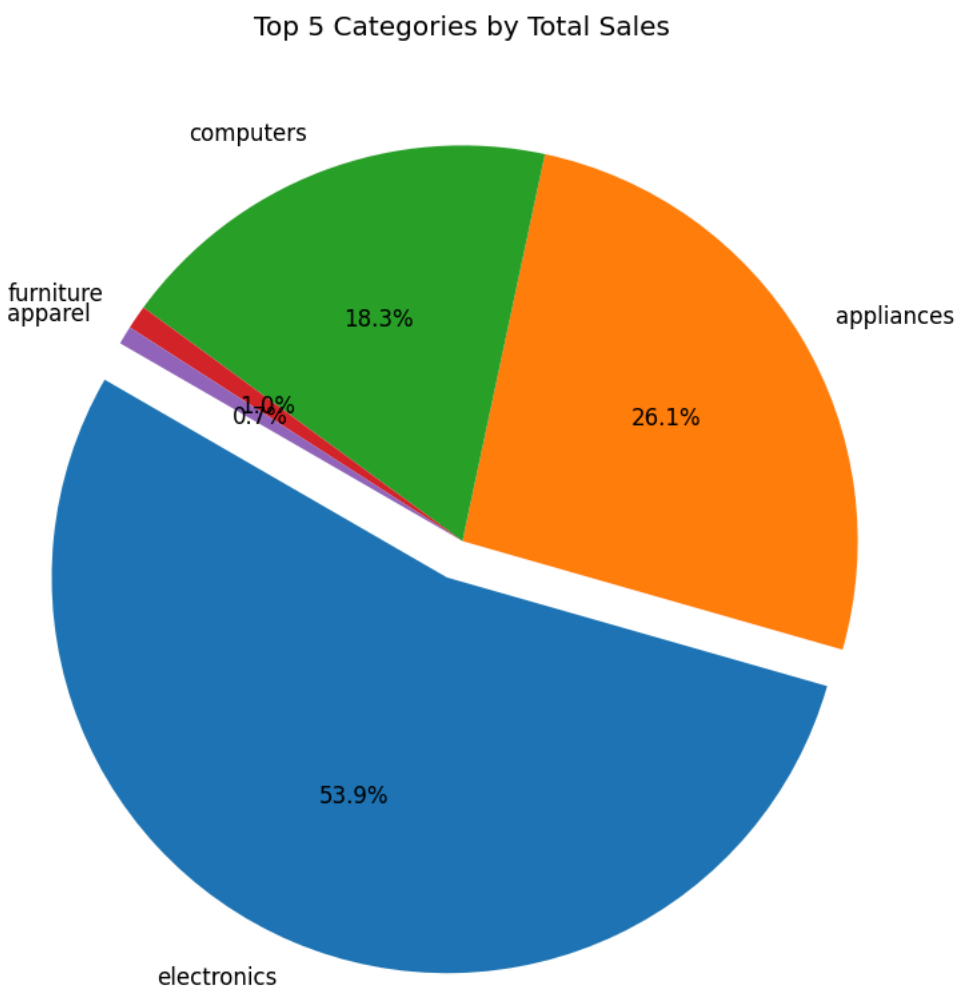



```
# by total sales
sales_category = data[data['category'] != 'unknown'].groupby('category').agg(
    total_sales=('price', 'sum')
).reset_index().sort_values('total_sales', ascending=False).reset_index(drop=True)
sales_category.head(10)
```

	category	total_sales
0	electronics	56713685.46
1	appliances	27437259.95
2	computers	19242876.00
3	furniture	1022587.13
4	apparel	787574.36
5	kids	549690.68
6	construction	331910.78
7	sport	243893.92
8	auto	119889.08
9	medicine	70498.46

Next steps: Generate code with sales_category | View recommended plots | New interactive sheet

```
# Plot the pie plot for the top 5 categories by sales
# extract the top 5 categories as labels
labels = sales_category['category'][:5]
# only "explode" the 1st slice (i.e. 'electronics')
explode = (0.1, 0, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=sales_category.total_sales[:5], explode=explode, labels=labels, autopct='%1.1f%%', startangle=150)

plt.title('Top 5 Categories by Total Sales')
plt.tight_layout()
plt.show()
```




```
# top 10 brands
print('The total number of brands sold: ', data['brand'].nunique())
```

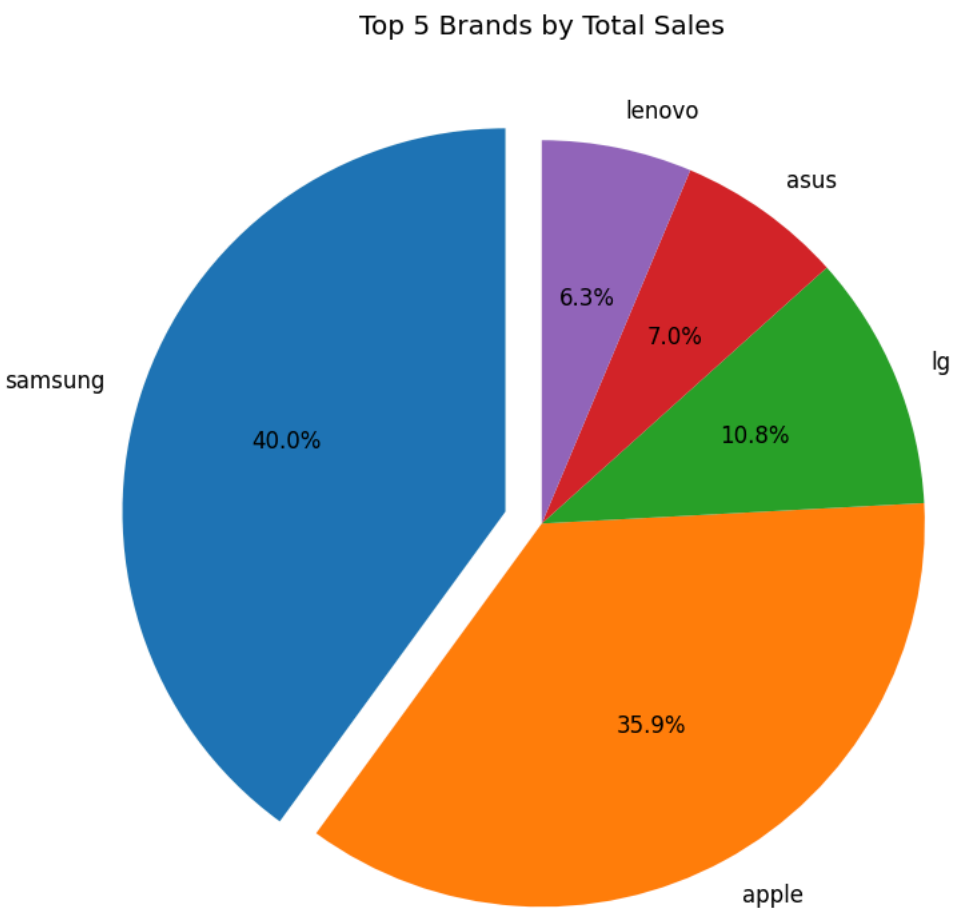
```
The total number of brands sold:  866
```

```
df_brand = data.groupby('brand').agg(
    total_users=('user_id', 'nunique'),
    total_orders = ('order_id', 'nunique'),
    total_sales = ('price', 'sum')
).reset_index().sort_values(by='total_sales', ascending=False).reset_index(drop=True)
df_brand.head(10)
```

	brand	total_users	total_orders	total_sales
0	samsung	35633	84685	28890299.34
1	apple	18762	34257	25929970.87
2	lg	8381	15572	7796270.97
3	asus	5711	8945	5074716.53
4	lenovo	4863	7853	4582558.86
5	bosch	4752	8419	3338757.25
6	hp	3936	6004	2496689.33
7	xiaomi	9107	14413	2390952.57
8	huawei	5722	9860	2218195.42
9	beko	4959	7300	2061907.61

Next steps: Generate code with df_brand | View recommended plots | New interactive sheet

```
# Plot the pie plot for top 5 brands by sales
# extract the top 5 catories as labels
labels = df_brand['brand'][:5]
# only "explode" the 1st slice (i.e. 'Samsung')
explode = (0.1, 0, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=df_brand.total_sales[:5], explode=explode,labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Top 5 Brands by Total Sales')
plt.tight_layout()
plt.show()
```



```
# top 10 products
print('The total number of unique products: ', data['product'].nunique())
print('The total number of product sold: ', data['product_id'].nunique())
```

```
The total number of unique products:  124
The total number of product sold:  19053
```

```
df_product = data.groupby(['product', 'brand']).agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales=('price', 'sum')
).reset_index().sort_values(by='total_sales', ascending=False).reset_index(drop=True)
df_product.head(10)
```

	product	brand	total_users	total_orders	total_sales
0	smartphone	apple	13795	23043	19163221.51
1	smartphone	samsung	23627	47792	16203110.72
2	notebook	asus	4832	7486	4699457.65
3	notebook	lenovo	4593	7430	4507154.41
4	video.tv	samsung	3640	6080	3903495.59
5	notebook	apple	1456	2249	3292571.10
6	video.tv	lg	3323	5533	3282657.46
7	kitchen.refrigerators	samsung	1995	2973	2191618.54
8	kitchen.washer	samsung	2956	4998	2058715.88
9	kitchen.washer	lg	3262	5327	1950491.25

Next steps: Generate code with df_product | View recommended plots | New interactive sheet

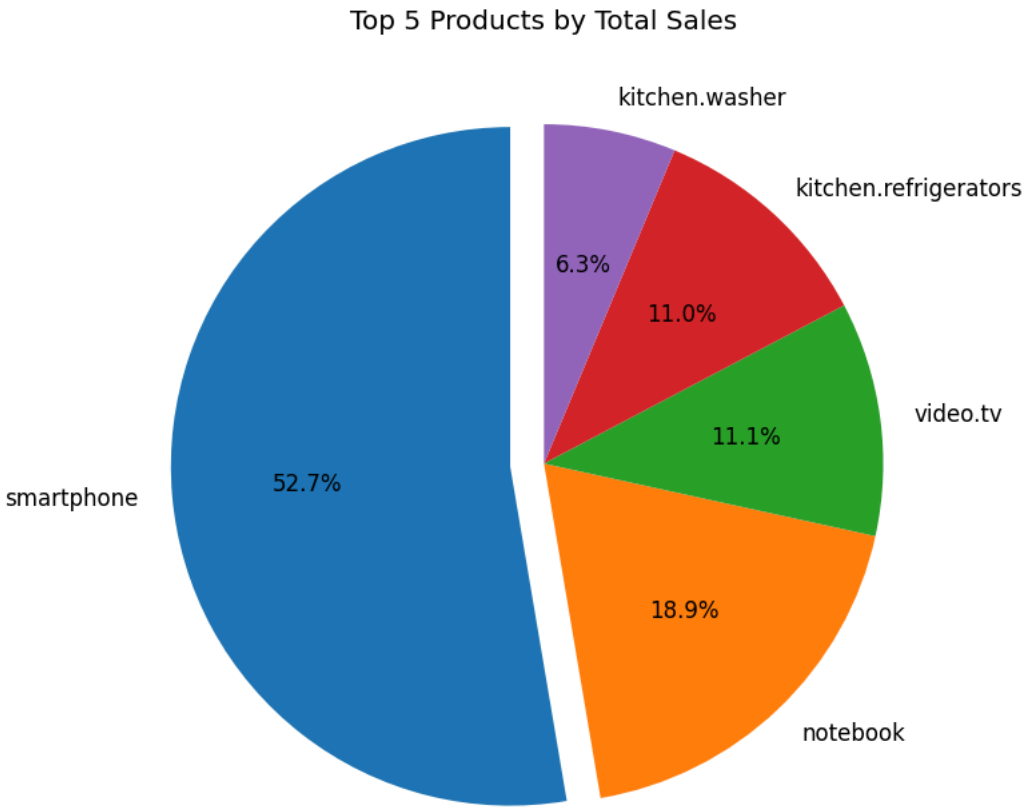
```
df_product_1 = data[data['product'] != 'unknown'].groupby('product').agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales=('price', 'sum')
).reset_index().sort_values(by='total_sales', ascending=False). reset_index(drop=True)
df_product_1.head(10)
```

	product	total_users	total_orders	total_sales
0	smartphone	41985	94980	41177570.76
1	notebook	12775	24617	14774573.47
2	video.tv	8645	17162	8716817.96
3	kitchen.refrigerators	10249	19248	8594453.16
4	kitchen.washer	7193	13865	4917272.74
5	environment.vacuum	7450	15427	2435306.62
6	tablet	4144	6279	2340078.84
7	clocks	3894	6197	2178261.94
8	environment.air_conditioner	4923	6933	1822777.41
9	kitchen.hood	3840	6959	1796502.32

Next steps: Generate code with df_product_1 | View recommended plots | New interactive sheet

```
# Plot the pie plot for top 5 products by sales
# extract the top 5 catories as labels
labels = df_product_1['product'][:5]
# only "explode" the 1st slice (i.e. 'smartphone')
explode = (0.1, 0, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=df_product_1.total_sales[:5], explode=explode, labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Top 5 Products by Total Sales')
```

```
plt.tight_layout()
plt.show()
```

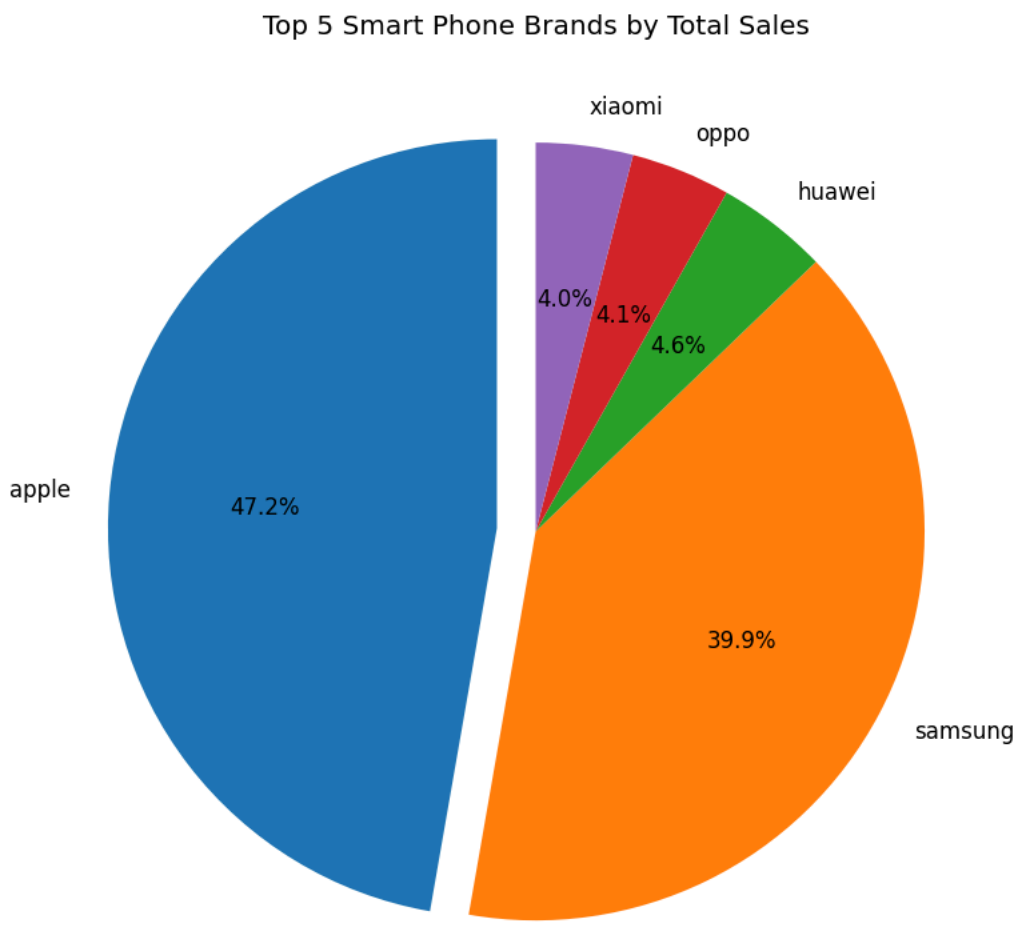



```
# Top 5 selling smart phone brands
df_smartphone = data[data['product'] == 'smartphone'].groupby('brand').agg(
    total_users=('user_id', 'nunique'),
    total_orders=('order_id', 'nunique'),
    total_sales=('price', 'sum')
).reset_index().sort_values(by='total_sales', ascending=False).reset_index(drop=True)
df_smartphone.head()
```

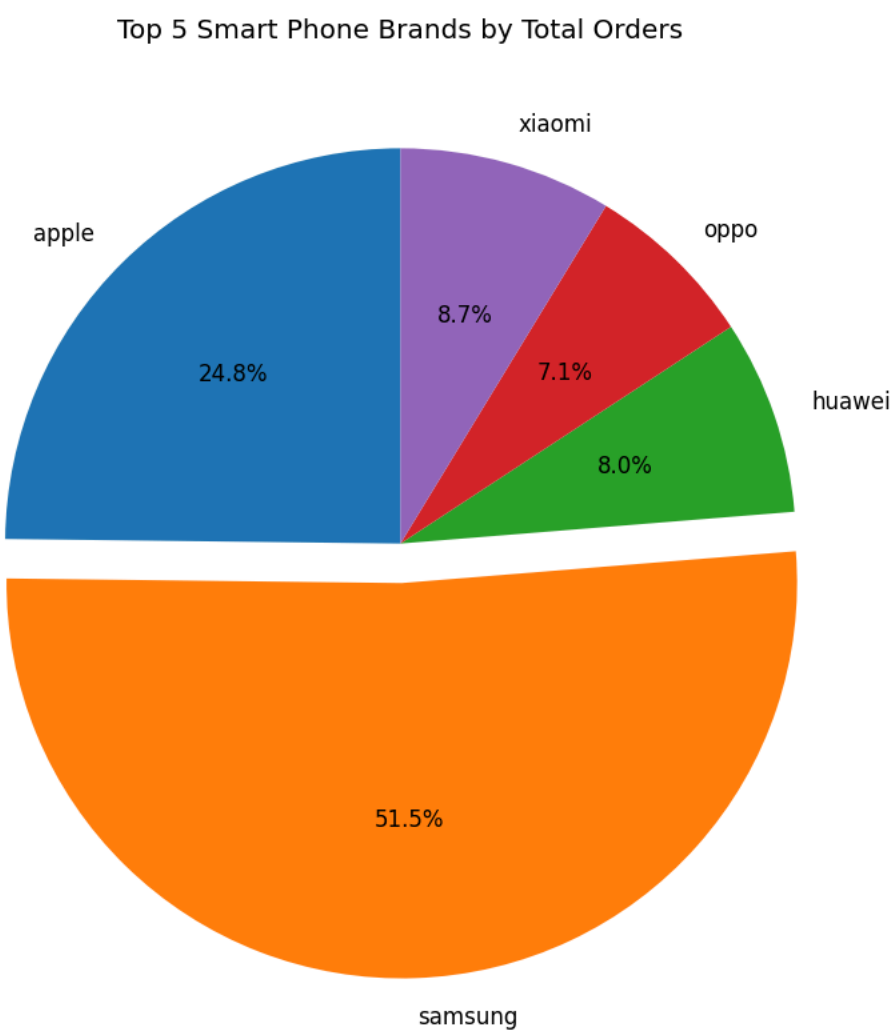
	brand	total_users	total_orders	total_sales
0	apple	13795	23043	19163221.51
1	samsung	23627	47792	16203110.72
2	huawei	4480	7386	1881632.04
3	oppo	3275	6555	1678580.92
4	xiaomi	5470	8081	1634833.59

Next steps: Generate code with df_smartphone | View recommended plots | New interactive sheet

```
# Plot the top 5 smart phone brands by total sales
# Extract the top 10 category codes as labels
labels = df_smartphone.brand[:5]
# only "explode" the 1st slice (i.e. 'Apple')
explode = (0.1, 0, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=df_smartphone.total_sales[:5], explode=explode,labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Top 5 Smart Phone Brands by Total Sales')
plt.tight_layout()
plt.show()
```




```
# Plot the top 5 smart phones by total orders
# Extract the top 10 category codes as labels
labels = df_smartphone.brand[:5]
# only "explode" the 2nd slice (i.e. 'Samsung')
explode = (0, 0.1, 0, 0, 0)

fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(x=df_smartphone.total_orders[:5], explode=explode,labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Top 5 Smart Phone Brands by Total Orders')
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
# Import libraries - 2
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error
from pandas.plotting import autocorrelation_plot

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.nonparametric.smoothers_lowess import lowess

import warnings
warnings.filterwarnings('ignore')


# drop all other columns except the two below
data2 = data2[['event_time', 'price']]
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2633521 entries, 0 to 2633520
Data columns (total 2 columns):
 #   Column      Dtype
---  ------      -----
 0   event_time  object
 1   price       float64
dtypes: float64(1), object(1)
```

```
memory usage: 40.2+ MB
```

```
# data preprocessing
data2 = data2.drop_duplicates()
data2.isnull().sum()
```

	0
event_time	0
price	241375

dtype: int64

```
# remove rows where 'price' contains NaNs
data2 = data2[~data2['price'].isnull()]
data2
```

	event_time	price
0	2020-04-24 11:50:39 UTC	162.01
2	2020-04-24 14:37:43 UTC	77.52
4	2020-04-24 19:16:21 UTC	217.57
5	2020-04-26 08:45:57 UTC	39.33
6	2020-04-26 09:33:47 UTC	1387.01
...	...	...
2633516	2020-11-21 10:10:01 UTC	138.87
2633517	2020-11-21 10:10:13 UTC	418.96
2633518	2020-11-21 10:10:30 UTC	12.48
2633519	2020-11-21 10:10:30 UTC	41.64
2633520	2020-11-21 10:10:30 UTC	53.22

2139643 rows × 2 columns

```
data2.isnull().sum()
```

	0
event_time	0
price	0

dtype: int64

```
data2.describe(include='all')
```

	event_time	price
count	2139643	2139643.00
unique	1299919	NaN
top	1970-01-01 00:33:40 UTC	NaN
freq	674	NaN
mean	NaN	155.76
std	NaN	243.61
min	NaN	0.00
25%	NaN	15.02
50%	NaN	57.85
75%	NaN	201.37
max	NaN	50925.90

```
# exclude event_time rows that contain 1970
data2 = data2[~data2['event_time'].str.contains('1970')]
#remove UTC from event_time
data2['event_time'] = data2['event_time'].str.replace('UTC','')
data2.describe(include='all')
```

	event_time	price
count	2138969	2138969.00
unique	1299918	NaN
top	2020-04-09 16:30:01	NaN
freq	155	NaN
mean	NaN	155.71
std	NaN	243.43
min	NaN	0.00
25%	NaN	15.02
50%	NaN	57.85
75%	NaN	201.37
max	NaN	50925.90

```
# create date column
data2['date'] = data2.event_time.apply(lambda x: x.split(' ')[0])

# convert to datetime object
data2['date'] = pd.to_datetime(data2['date'])
# delete event_time column
del data2['event_time']
data2.head()
```

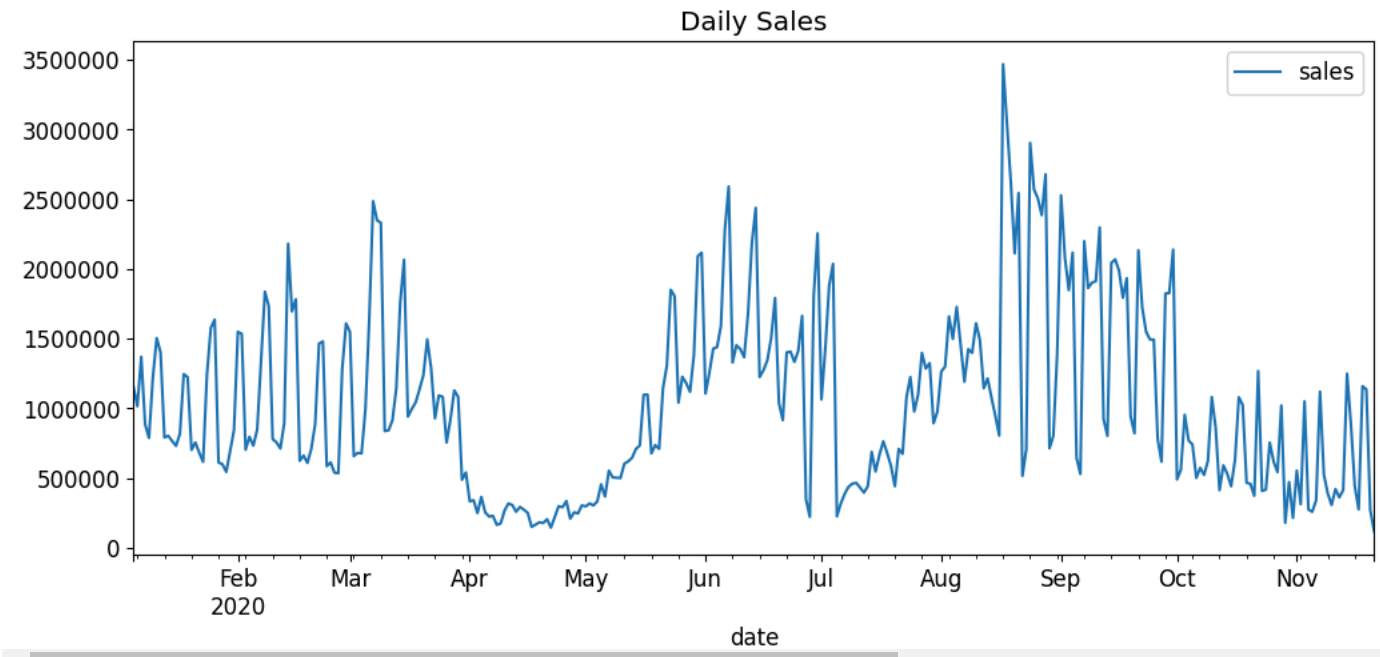
	price	date
0	162.01	2020-04-24
2	77.52	2020-04-24
4	217.57	2020-04-24
5	39.33	2020-04-26
6	1387.01	2020-04-26

```
# create a new time series
df = data2.groupby('date').agg(sales=('price', 'sum'))
df.head()
```

	sales
date	
2020-01-05	1151017.74
2020-01-06	1014544.86
2020-01-07	1369143.81
2020-01-08	886054.44
2020-01-09	787447.75

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# visualize the time series data
fig, ax = plt.subplots()
plt.ticklabel_format(style='plain')
fig = df.plot(figsize=(12,5), ax=ax)
plt.title('Daily Sales')
plt.show()
```




```
# Define RFM Dataset - 1
from pandas.tseries.offsets import MonthEnd

df_data['month_key'] = df_data['event_time'].dt.month
df_data[['event_time', 'month_key']]
```

	event_time	month_key
0	2020-04-24 11:50:39	4
1	2020-04-24 11:50:39	4
2	2020-04-24 14:37:43	4
3	2020-04-24 14:37:43	4
4	2020-04-24 19:16:21	4
...	...	...
562857	2020-11-21 10:10:01	11
562858	2020-11-21 10:10:13	11
562859	2020-11-21 10:10:30	11
562860	2020-11-21 10:10:30	11
562861	2020-11-21 10:10:30	11

562862 rows × 2 columns

```
# creating a new dataframe
df_month_keys = pd.DataFrame({"month_key":df_data['month_key'].unique(), 'key':0})
df_user_ids = pd.DataFrame({"user_id":df_data['user_id'].unique(), 'key':0})

df_rfm = df_month_keys.merge(df_user_ids, on='key', how='outer')
df_rfm = df_rfm.drop(columns=['key'])
df_rfm = df_rfm.sort_values(by=['user_id', 'month_key']).reset_index(drop=True)
df_rfm
```

	month_key	user_id
0	1	1515915625439952128.00
1	2	1515915625439952128.00
2	3	1515915625439952128.00
3	4	1515915625439952128.00
4	5	1515915625439952128.00
...	...	...
1080679	7	1515915625514891264.00
1080680	8	1515915625514891264.00
1080681	9	1515915625514891264.00
1080682	10	1515915625514891264.00
1080683	11	1515915625514891264.00

1080684 rows × 2 columns

```
# Recency
# User last month purchase
df_user_month_purchases = df_data[['month_key', 'user_id']].drop_duplicates()
df_user_month_purchases['last_purchase'] = df_user_month_purchases['month_key']

df_rfm = df_rfm.merge(df_user_month_purchases, how='left', on=['month_key', 'user_id'])

# filling the last_purchase month
user_ids = df_rfm[['user_id']]
df_rfm = df_rfm.groupby('user_id').ffill()
df_rfm['R_months_since_last_purchase'] = df_rfm['month_key'] - df_rfm['last_purchase']
df_rfm['user_id'] = user_ids
df_rfm.head(20)
```

	month_key	last_purchase	R_months_since_last_purchase	user_id
0	1	NaN	NaN	1515915625439952128.00
1	2	NaN	NaN	1515915625439952128.00
2	3	NaN	NaN	1515915625439952128.00
3	4	NaN	NaN	1515915625439952128.00
4	5	NaN	NaN	1515915625439952128.00
5	6	NaN	NaN	1515915625439952128.00
6	7	7.00	0.00	1515915625439952128.00
7	8	7.00	1.00	1515915625439952128.00
8	9	7.00	2.00	1515915625439952128.00
9	10	7.00	3.00	1515915625439952128.00
10	11	7.00	4.00	1515915625439952128.00
11	1	NaN	NaN	1515915625440038400.00
12	2	NaN	NaN	1515915625440038400.00
13	3	NaN	NaN	1515915625440038400.00
14	4	NaN	NaN	1515915625440038400.00
15	5	NaN	NaN	1515915625440038400.00
16	6	NaN	NaN	1515915625440038400.00
17	7	NaN	NaN	1515915625440038400.00
18	8	NaN	NaN	1515915625440038400.00
19	9	9.00	0.00	1515915625440038400.00

```
# Frequency
# user last month purchase order count
df_user_month_purchases = df_data.groupby(['month_key', 'user_id'])['order_id'].nunique().reset_index()
```

```
df_rfm = df_rfm.merge(df_user_month_purchases, how='left', on=['month_key', 'user_id'])

# filling the last_purchase month
user_ids = df_rfm[['user_id']]
df_rfm = df_rfm.groupby('user_id').ffill()
df_rfm['user_id'] = user_ids
df_rfm = df_rfm.rename(columns={"order_id":"F_last_monthly_purchases_count"})
df_rfm.head(20)
```

	month_key	last_purchase	R_months_since_last_purchase	F_last_monthly_purchases_count	user_id
0	1	NaN	NaN	NaN	1515915625439952128.00
1	2	NaN	NaN	NaN	1515915625439952128.00
2	3	NaN	NaN	NaN	1515915625439952128.00
3	4	NaN	NaN	NaN	1515915625439952128.00
4	5	NaN	NaN	NaN	1515915625439952128.00
5	6	NaN	NaN	NaN	1515915625439952128.00
6	7	7.00	0.00	1.00	1515915625439952128.00
7	8	7.00	1.00	1.00	1515915625439952128.00
8	9	7.00	2.00	1.00	1515915625439952128.00
9	10	7.00	3.00	1.00	1515915625439952128.00
10	11	7.00	4.00	1.00	1515915625439952128.00
11	1	NaN	NaN	NaN	1515915625440038400.00
12	2	NaN	NaN	NaN	1515915625440038400.00
13	3	NaN	NaN	NaN	1515915625440038400.00
14	4	NaN	NaN	NaN	1515915625440038400.00
15	5	NaN	NaN	NaN	1515915625440038400.00
16	6	NaN	NaN	NaN	1515915625440038400.00
17	7	NaN	NaN	NaN	1515915625440038400.00
18	8	NaN	NaN	NaN	1515915625440038400.00
19	9	9.00	0.00	1.00	1515915625440038400.00

```
# Monetary
# user last monthly purchase value
df_user_month_purchases = df_data.groupby(['month_key','user_id'])['price'].sum().reset_index()
df_rfm=df_rfm.merge(df_user_month_purchases, how='left', on=['month_key','user_id'])

# fill in  last purchase month
user_ids = df_rfm[['user_id']]
df_rfm = df_rfm.groupby('user_id').ffill()
df_rfm['user_id'] = user_ids
df_rfm = df_rfm.rename(columns={"price":"M_last_monthly_purchases_value"})
df_rfm.head(20)
```

	month_key	last_purchase	R_months_since_last_purchase	F_last_monthly_purchases_count	M_last_monthly_purchases_value	
0	1	NaN	NaN	NaN	NaN	151591562543
1	2	NaN	NaN	NaN	NaN	151591562543
2	3	NaN	NaN	NaN	NaN	151591562543
3	4	NaN	NaN	NaN	NaN	151591562543
4	5	NaN	NaN	NaN	NaN	151591562543
5	6	NaN	NaN	NaN	NaN	151591562543
6	7	7.00	0.00	1.00	416.64	151591562543
7	8	7.00	1.00	1.00	416.64	151591562543
8	9	7.00	2.00	1.00	416.64	151591562543
9	10	7.00	3.00	1.00	416.64	151591562543
10	11	7.00	4.00	1.00	416.64	151591562543
11	1	NaN	NaN	NaN	NaN	151591562544
12	2	NaN	NaN	NaN	NaN	151591562544
13	3	NaN	NaN	NaN	NaN	151591562544
14	4	NaN	NaN	NaN	NaN	151591562544
15	5	NaN	NaN	NaN	NaN	151591562544
16	6	NaN	NaN	NaN	NaN	151591562544
17	7	NaN	NaN	NaN	NaN	151591562544
18	8	NaN	NaN	NaN	NaN	151591562544
19	9	9.00	0.00	1.00	21.04	151591562544

```
# define RFM dataframe
# drop all missing values
df_rfm = df_rfm.dropna()
df_rfm = df_rfm[['user_id','month_key','R_months_since_last_purchase','F_last_monthly_purchases_count', 'M_last_monthly_purchases_value']]
df_rfm
```

	user_id	month_key	R_months_since_last_purchase	F_last_monthly_purchases_count	M_last_monthly_purchases_value
6	1515915625439952128.00	7	0.00	1.00	416.64
7	1515915625439952128.00	8	1.00	1.00	416.64
8	1515915625439952128.00	9	2.00	1.00	416.64
9	1515915625439952128.00	10	3.00	1.00	416.64
10	1515915625439952128.00	11	4.00	1.00	416.64
...	...	...	...	...	...
1080639	1515915625514887424.00	11	0.00	1.00	208.3
1080650	1515915625514887936.00	11	0.00	1.00	3472.2
1080661	1515915625514888704.00	11	0.00	3.00	752.4
1080672	1515915625514891008.00	11	0.00	1.00	925.6
1080683	1515915625514891264.00	11	0.00	1.00	418.9

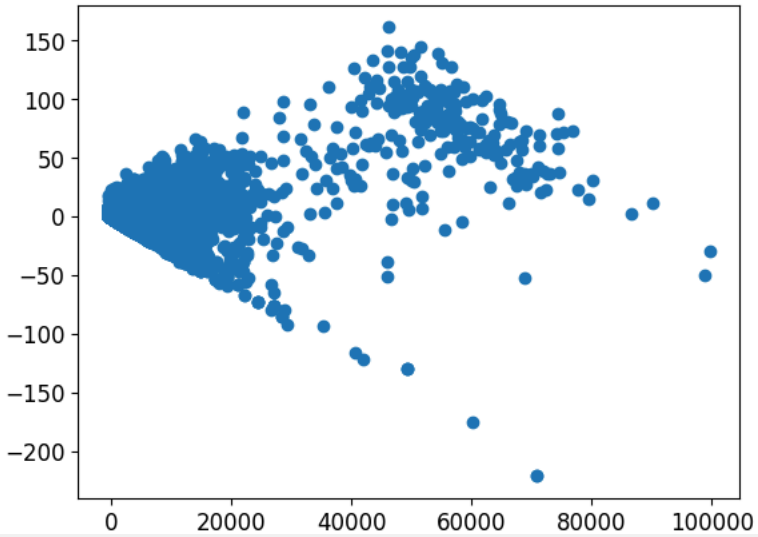
496504 rows × 5 columns

```
# visualizing PCA - Prinicpal Component Analysis
from sklearn.decomposition import PCA
X = df_rfm[['R_months_since_last_purchase','F_last_monthly_purchases_count','M_last_monthly_purchases_value']]
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
df_pca = pd.DataFrame(X_pca, columns=['pca_1','pca_2'])
df_pca
```

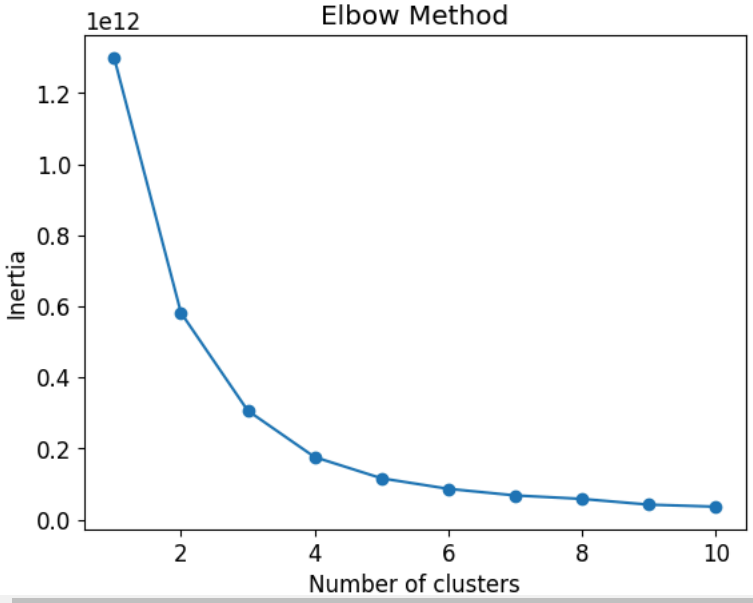
	pca_1	pca_2
0	-146.08	-0.39
1	-146.08	-0.38
2	-146.08	-0.37
3	-146.08	-0.36
4	-146.08	-0.35
...	...	...
496499	-354.41	0.30
496500	2909.47	-10.52
496501	189.78	0.50
496502	362.95	-2.07
496503	-143.76	-0.39

496504 rows × 2 columns

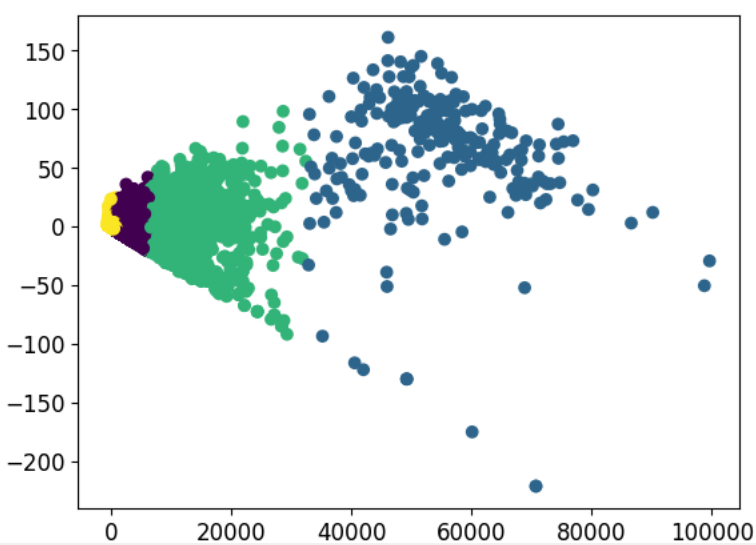
```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(df_pca['pca_1'], df_pca['pca_2'])
plt.show()
```




```
# K means clustering
from sklearn.cluster import KMeans
inertias = []
for i in range(1,11):
  kmeans = KMeans(n_clusters=i)
  kmeans.fit(X)
  inertias.append(kmeans.inertia_)
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```




```
kmeans = KMeans(n_clusters=4)
kmeans.fit(df_pca)
plt.scatter(df_pca['pca_1'], df_pca['pca_2'], c=kmeans.labels_)
plt.show()
```




```
# Analyze results of clustering
df_rfm['kmeans_labels'] = kmeans.labels_
df_rfm.groupby('kmeans_labels')[['R_months_since_last_purchase','F_last_monthly_purchases_count','M_last_monthly_purchases_value']].agg({"me
```

	R_months_since_last_purchase		F_last_monthly_purchases_count		M_last_monthly_purchases_value	
	count	mean	count	mean	count	mean
kmeans_labels						
0	44097	1.23	44097	4.06	44097	2363.64
1	237	0.02	237	244.30	237	55319.63
2	2024	0.73	2024	27.08	2024	11454.81
3	450146	1.74	450146	1.40	450146	308.49

Start coding or generate with AI.

```
# Machine Learning
# limiting the data in the dataset to perform machine learning operation with limited resources used
df = data3.head(1000)
df
```

	event_time	order_id	product_id	category_id	category_code	brand	price	
0	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2268105426648171008.00	electronics.tablet	samsung	162.01	151591562
1	2020-04-24 11:50:39 UTC	2294359932054536986	1515966223509089906	2268105426648171008.00	electronics.tablet	samsung	162.01	151591562
2	2020-04-24 14:37:43 UTC	2294444024058086220	2273948319057183658	2268105430162997760.00	electronics.audio.headphone	huawei	77.52	151591562
3	2020-04-24 14:37:43 UTC	2294444024058086220	2273948319057183658	2268105430162997760.00	electronics.audio.headphone	huawei	77.52	151591562
4	2020-04-24 19:16:21 UTC	2294584263154074236	2273948316817424439	2268105471367840000.00	NaN	karcher	217.57	151591562
...	...	...	...	...	...	...	...	...

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Inspect the dataset
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   event_time     1000 non-null   object
 1   order_id       1000 non-null   int64
 2   product_id     1000 non-null   int64
 3   category_id    1000 non-null   float64
 4   category_code  784 non-null    object
 5   brand          950 non-null    object
 6   price          1000 non-null   float64
 7   user_id        1000 non-null   float64
dtypes: float64(3), int64(2), object(3)
memory usage: 62.6+ KB
None
                 event_time             order_id           product_id  \
0  2020-04-24 11:50:39 UTC  2294359932054536986  1515966223509089906
1  2020-04-24 11:50:39 UTC  2294359932054536986  1515966223509089906
2  2020-04-24 14:37:43 UTC  2294444024058086220  2273948319057183658
3  2020-04-24 14:37:43 UTC  2294444024058086220  2273948319057183658
4  2020-04-24 19:16:21 UTC  2294584263154074236  2273948316817424439

              category_id               category_code    brand  price  \
0 2268105426648171008.00          electronics.tablet  samsung 162.01
1 2268105426648171008.00          electronics.tablet  samsung 162.01
2 2268105430162997760.00  electronics.audio.headphone   huawei  77.52
3 2268105430162997760.00  electronics.audio.headphone   huawei  77.52
4 2268105471367840000.00                         NaN  karcher 217.57

                 user_id
0 1515915625441993984.00
1 1515915625441993984.00
2 1515915625447879424.00
3 1515915625447879424.00
4 1515915625443148032.00
```

```
# Preprocess data
df = df.dropna(subset=['user_id', 'product_id', 'price'])  # Drop rows with missing user_id, product_id, or price
df['user_id'] = df['user_id'].astype(int)
df['product_id'] = df['product_id'].astype(int)


# added code
df.head()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   event_time     1000 non-null   object 
 1   order_id       1000 non-null   int64  
 2   product_id     1000 non-null   int64  
 3   category_id    1000 non-null   float64
 4   category_code  784 non-null    object 
 5   brand          950 non-null    object 
 6   price          1000 non-null   float64
 7   user_id        1000 non-null   int64  
dtypes: float64(2), int64(3), object(3)
memory usage: 62.6+ KB
```

```
df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	
event_time	1000	768	2020-04-30 05:04:53 UTC	16	NaN	NaN	NaN	
order_id	1000.00	NaN	NaN	NaN	2298508537828504320.00	534154845575678.44	2294359932054536960.00	2298235
product_id	1000.00	NaN	NaN	NaN	1682746760108694272.00	314195454751919104.00	1515966223509088512.00	1515966
category_id	1000.00	NaN	NaN	NaN	2271408141487000832.00	17726960141293160.00	2268105388421284352.00	2268105
category_code	784	67	electronics.smartphone	112	NaN	NaN	NaN	
brand	950	150	samsung	141	NaN	NaN	NaN	
price	1000.00	NaN	NaN	NaN	120.02	199.79	0.02	
user_id	1000.00	NaN	NaN	NaN	1515915625449569536.00	7989943.87	1515915625440946432.00	1515915

```
# Data preprocessing
df = df.drop_duplicates()
df.isnull().sum()
```

	0
event_time	0
order_id	0
product_id	0
category_id	0
category_code	213
brand	49
price	0
user_id	0

dtype: int64

```
# exclude rows where 'event_time' contains '1970'
df = df[~df['event_time'].str.contains('1970')]
df.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	
event_time	984	768	2020-04-30 05:04:53 UTC	16	NaN	NaN	NaN	
order_id	984.00	NaN	NaN	NaN	2298536380355673856.00	464266743452000.88	2294359932054536960.00	22982413
product_id	984.00	NaN	NaN	NaN	1681607103947265536.00	313424977112442944.00	1515966223509088512.00	15159662
category_id	984.00	NaN	NaN	NaN	2271353720612105216.00	17560725358318130.00	2268105388421284352.00	22681054
category_code	771	67	electronics.smartphone	109	NaN	NaN	NaN	
brand	935	150	samsung	138	NaN	NaN	NaN	
price	984.00	NaN	NaN	NaN	116.51	188.58	0.02	
user_id	984.00	NaN	NaN	NaN	1515915625449587712.00	8042060.83	1515915625440946432.00	15159156

```
# remove empty rows in brand and user_id columns
df = df.dropna(subset=['brand', 'user_id']).reset_index(drop=True)
df.isnull().sum()
```

	0
event_time	0
order_id	0
product_id	0
category_id	0
category_code	197
brand	0
price	0
user_id	0

dtype: int64

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from surprise import SVD, Dataset, Reader, KNNBasic
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
# Create train-test split for collaborative filtering
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)


# Collaborative Filtering
# Convert to Surprise format
reader = Reader(rating_scale=(df['price'].min(), df['price'].max()))
data = Dataset.load_from_df(df[['user_id', 'product_id', 'price']], reader)
trainset = data.build_full_trainset()


# Use SVD for collaborative filtering
collab_model = SVD()
collab_model.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7e0822cc3070>
```

```
print("Unique product IDs:", df['product_id'].unique())
```

Show hidden output

```
print("Unique users IDs:", df['user_id'].unique())
```

Show hidden output

```
# Test on a sample user-product pair
pred = collab_model.predict(uid=1515915625441990000, iid=1515966223509089906)
print(f"Collaborative Filtering Prediction: {pred.est}")
```

```
Collaborative Filtering Prediction: 122.59353994608634
```

```
testset = [tuple(x) for x in test_data[['user_id', 'product_id', 'price']].to_numpy()]

# Generate predictions for the testset
predictions = collab_model.test(testset)
true_ratings = [pred.r_ui for pred in predictions]  # True ratings
predicted_ratings = [pred.est for pred in predictions]  # Predicted ratings
rmse = np.sqrt(mean_squared_error(true_ratings, predicted_ratings))
print(f"RMSE: {rmse}")
```

```
RMSE: 313.0355957469271
```

```
# Content-Based Filtering
# Combine category_code and brand for content-based features
df['content'] = df['category_code'].fillna('') + " " + df['brand'].fillna('')


# Use TF-IDF to vectorize content
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['content'])


# Function to compute similarity on-the-fly
def content_based_recommendations_on_the_fly(product_id, top_n=5):
    # idx = df[df['product_id'] == product_id].index[0]
    if product_id not in df['product_id'].values:
        return []
    indices = df.index[df['product_id'] == product_id].tolist()
    if not indices:
        return []
    idx = indices[0]
    product_vector = tfidf_matrix[idx]
    sim_scores = linear_kernel(product_vector, tfidf_matrix).flatten()
    sim_scores = [(i, score) for i, score in enumerate(sim_scores)]
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:top_n + 1]
    product_indices = [i[0] for i in sim_scores]
    return df.iloc[product_indices]['product_id'].tolist()

# Test content-based filtering
print("Content-Based Recommendations:", content_based_recommendations_on_the_fly(product_id=1515966223509089906))
```

```
Content-Based Recommendations: [1515966223509106817, 1515966223509106817, 1515966223509089673, 1515966223509089673, 2273948222336532851]
```

```
from sklearn.neighbors import NearestNeighbors

# Train a nearest neighbor model
nn = NearestNeighbors(metric='cosine', algorithm='brute')
nn.fit(tfidf_matrix)

# Function to get recommendations
def ann_recommendations(product_id, top_n=5):
    idx = df[df['product_id'] == product_id].index[0]
    distances, indices = nn.kneighbors(tfidf_matrix[idx], n_neighbors=top_n + 1)
    return df.iloc[indices.flatten()[1:]]['product_id'].tolist()

# Test ANN recommendations
print("ANN Content-Based Recommendations:", ann_recommendations(product_id=1515966223509089906))
```