

## ▼ Pandas Practice

This notebook is dedicated to practicing different tasks with pandas. The solutions are available in a solutions notebook, however, you should always try to figure them out yourself first.

It should be noted there may be more than one different way to answer a question or complete an exercise.

Exercises are based off (and directly taken from) the quick introduction to pandas notebook.

Different tasks will be detailed by comments or text.

For further reference and resources, it's advised to check out the [pandas documentation](#).

```
# Import pandas
import pandas as pd
```

```
# Create a series of three different colours
colors = pd.Series(['red', 'green', 'blue'])
```

```
# View the series of different colours
colors
```

```
0      red
1    green
2     blue
dtype: object
```

```
# Create a series of three different car types and view it
```

Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#)

```
0      suv
1    sedan
2  hatchback
dtype: object
```

```
# Combine the Series of cars and colours into a DataFrame
cars = pd.DataFrame({'colors': colors, 'car_types': car_types})
cars
```

	colors	car_types
0	red	suv
1	green	sedan
2	blue	hatchback

✓ 0s completed at 10:06 AM



```
cars.dtypes
```

```
colors      object
car_types   object
dtype: object
```

```
cars_category = cars.astype({'colors': 'category', 'car_types': 'category'})
cars_category
```

	colors	car_types
0	red	suv
1	green	sedan
2	blue	hatchback

```
cars_category.dtypes
```

```
colors      category
car_types   category
dtype: object
```

```
# Import "../data/car-sales.csv" and turn it into a DataFrame
sales = pd.read_csv("car-sales.csv")
```

**Note:** Since you've imported [../data/car-sales.csv](#) as a DataFrame, we'll now refer to this DataFrame as 'the car sales DataFrame'.

```
# Export the DataFrame you created to a .csv file
```

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

```
# Find the different datatypes of the car data DataFrame
cars.dtypes
```

```
colors      object
car_types   object
dtype: object
```

```
# Describe your current car sales DataFrame using describe()
sales.describe()
```

	Odometer (KM)	Doors
count	10.000000	10.000000
mean	78601.400000	4.000000

```

mean    70001.400000    4.000000
std      61983.471735    0.471405
min      11179.000000    3.000000
25%      35836.250000    4.000000
50%      57369.000000    4.000000
75%      96384.500000    4.000000
max     213095.000000    5.000000

```

```
# Get information about your DataFrame using info()
sales.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Make             10 non-null    object
1   Colour           10 non-null    object
2   Odometer (KM)    10 non-null    int64
3   Doors            10 non-null    int64
4   Price            10 non-null    object
dtypes: int64(2), object(3)
memory usage: 528.0+ bytes

```

What does it show you?

```
# Create a Series of different numbers and find the mean of them
nums = pd.Series([5,7,20,9])
nums.mean()
```

```
10.25
```

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

```
nums.sum()
```

```
41
```

```
sales.head()
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	\$4,000.00
1	Honda	Red	87899	4	\$5,000.00
2	Toyota	Blue	32549	3	\$7,000.00
3	BMW	Black	11179	5	\$22,000.00
4	Nissan	White	213095	4	\$25,000.00



```
└─ Nissan      White      213095      └─ $3,500.00
```

```
# List out all the column names of the car sales DataFrame
sales.columns
```

```
Index(['Make', 'Colour', 'Odometer (KM)', 'Doors', 'Price'],
      dtype='object')
```



```
# Find the length of the car sales DataFrame
len(sales)
```

```
10
```

```
sales.shape[0]
```

```
10
```


```
# Show the first 5 rows of the car sales DataFrame
sales.head(5)
```

	Make	Colour	Odometer (KM)	Doors	Price	
0	Toyota	White	150043	4	\$4,000.00	
1	Honda	Red	87899	4	\$5,000.00	
2	Toyota	Blue	32549	3	\$7,000.00	
3	BMW	Black	11179	5	\$22,000.00	
4	Nissan	White	213095	4	\$3,500.00	



```
# Show the first 7 rows of the car sales DataFrame
sales.head(7)
```

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

0	Toyota	White	150043	4	\$4,000.00	
1	Honda	Red	87899	4	\$5,000.00	
2	Toyota	Blue	32549	3	\$7,000.00	
3	BMW	Black	11179	5	\$22,000.00	
4	Nissan	White	213095	4	\$3,500.00	
5	Toyota	Green	99213	4	\$4,500.00	
6	Honda	Blue	45698	4	\$7,500.00	

```
# Show the bottom 5 rows of the car sales DataFrame
sales.tail(5)
```

	Make	Colour	Odometer (KM)	Doors	Price	
5	Toyota	Green	99213	4	\$4,500.00	
6	Honda	Blue	45698	4	\$7,500.00	
7	Honda	Blue	54738	4	\$7,000.00	
8	Toyota	White	60000	4	\$6,250.00	
9	Nissan	White	31600	4	\$9,700.00	

```
# Use .loc to select the row at index 3 of the car sales DataFrame
sales.loc[3]
```

```
Make          BMW
Colour        Black
Odometer (KM)  11179
Doors          5
Price         $22,000.00
Name: 3, dtype: object
```

```
# Use .iloc to select the row at position 3 of the car sales DataFrame
sales.iloc[3]
```

```
Make          BMW
Colour        Black
Odometer (KM)  11179
Doors          5
Price         $22,000.00
Name: 3, dtype: object
```

Notice how they're the same? Why do you think this is?

Check the pandas documentation for [.loc](#) and [.iloc](#). Think about a different situation each could be used for and try them out

Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#)

```
# Select the "Odometer (KM)" column from the car sales DataFrame
sales['Odometer (KM)']
```

```
0    150043
1     87899
2     32549
3     11179
4    213095
5     99213
6     45698
7     54738
8     60000
9     31600
Name: Odometer (KM), dtype: int64
```

```
# Find the mean of the "Odometer (KM)" column in the car sales DataFrame
```

```
sales['Odometer (KM)'].mean()
```

```
78601.4
```

```
# Select the rows with over 100,000 kilometers on the Odometer
sales.loc[sales['Odometer (KM)'] > 100000]
```

	Make	Colour	Odometer (KM)	Doors	Price	
0	Toyota	White	150043	4	\$4,000.00	
4	Nissan	White	213095	4	\$3,500.00	

```
# Create a crosstab of the Make and Doors columns
pd.crosstab(sales['Make'], sales['Doors'])
```

Doors	3	4	5	
Make				
<b>BMW</b>	0	0	1	
<b>Honda</b>	0	3	0	
<b>Nissan</b>	0	2	0	
<b>Toyota</b>	1	3	0	

```
sales.head()
```

	Make	Colour	Odometer (KM)	Doors	Price	
0	Toyota	White	150043	4	\$4,000.00	
1	Honda	Red	87899	4	\$5,000.00	

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

4	Nissan	White	213095	4	\$3,500.00
---	--------	-------	--------	---	------------

```
# Group columns of the car sales DataFrame by the Make column and find the average
sales.groupby('Make').mean()
```

```
<ipython-input-80-6bf863db20de>:2: FutureWarning: The default value of num
sales.groupby('Make').mean()
```

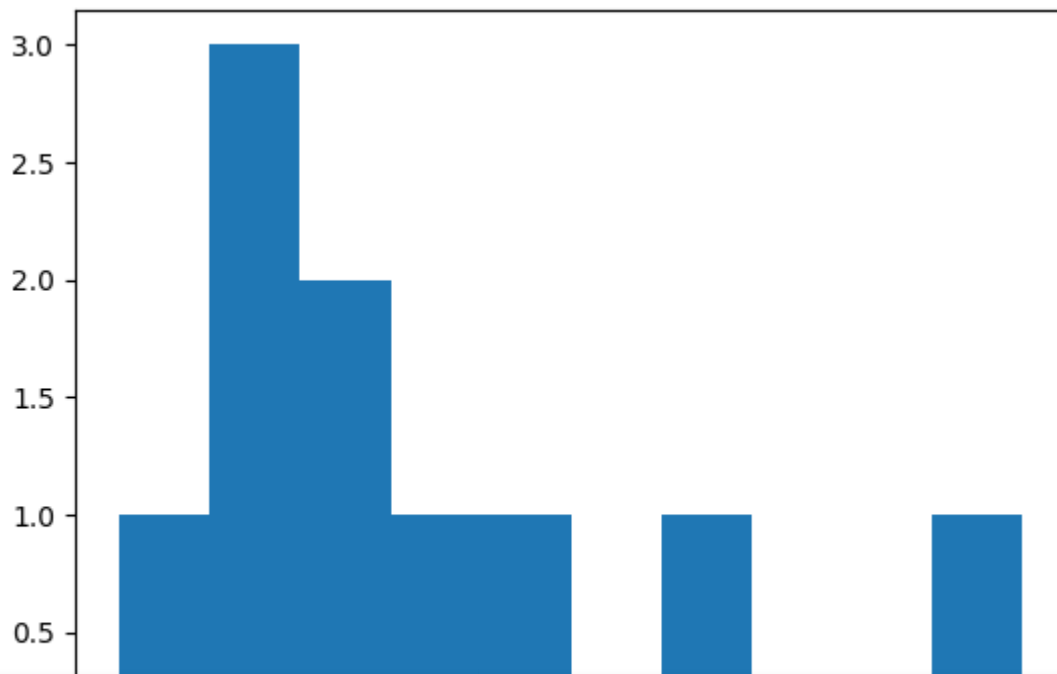
	Odometer (KM)	Doors	
Make			
<b>BMW</b>	11179.000000	5.00	
<b>Honda</b>	62778.333333	4.00	

<b>Nissan</b>	122347.500000	4.00
<b>Toyota</b>	85451.250000	3.75

```
# Import Matplotlib and create a plot of the Odometer column
# Don't forget to use %matplotlib inline
from matplotlib import pyplot as plt
%matplotlib inline

# Create a histogram of the Odometer column using hist()
plt.hist(sales['Odometer (KM)'])

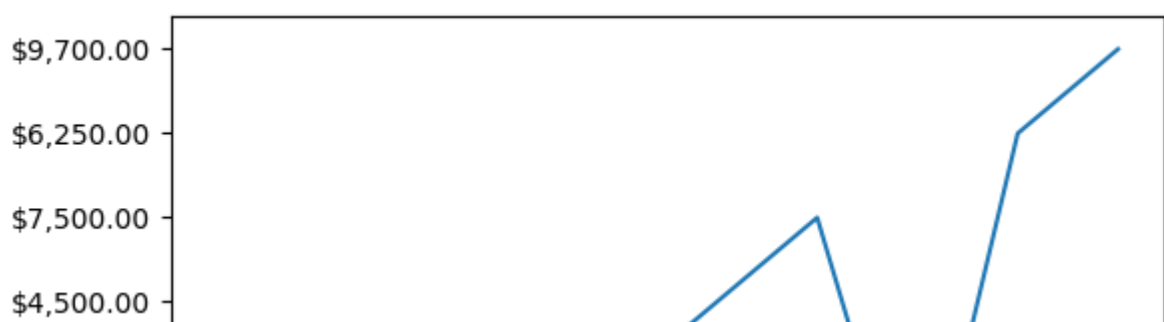
(array([1., 3., 2., 1., 1., 0., 1., 0., 0., 1.]),
 array([ 11179. ,  31370.6,  51562.2,  71753.8,  91945.4, 112137. ,
        132328.6, 152520.2, 172711.8, 192903.4, 213095. ]),
 <BarContainer object of 10 artists>)
```

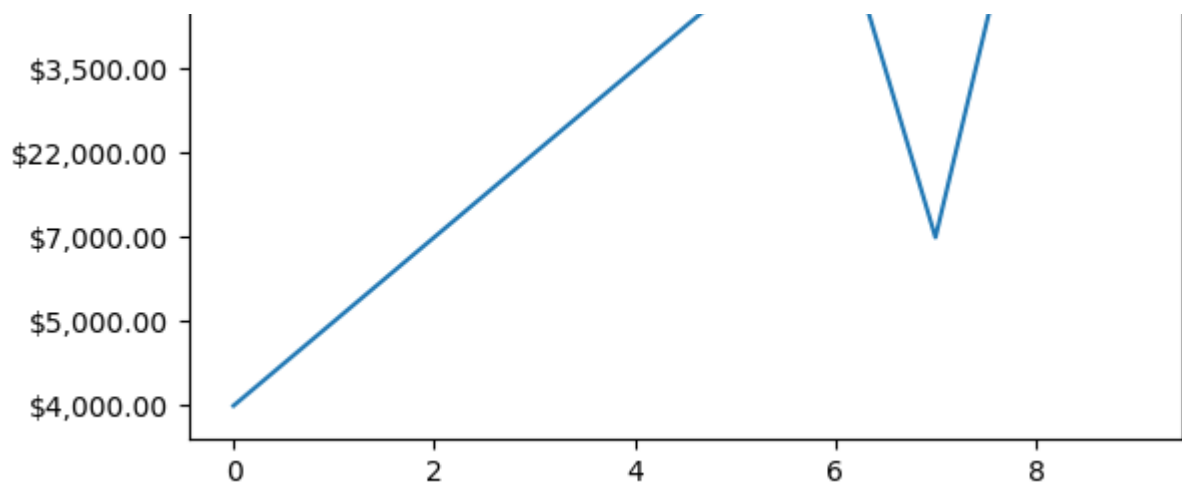


Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#)

```
# Try to plot the Price column using plot()
plt.plot(sales['Price'])
```

[<matplotlib.lines.Line2D at 0x7e751aca25f0>]





Why didn't it work? Can you think of a solution?

You might want to search for "how to convert a pandas string column to numbers".

And if you're still stuck, check out this [Stack Overflow question and answer on turning a price column into integers](#).

See how you can provide the example code there to the problem here.

```
sales.head()
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	\$4,000.00
1	Honda	Red	87899	4	\$5,000.00
2	Toyota	Blue	32549	3	\$7,000.00
3	BMW	Black	11179	5	\$22,000.00
4	Nissan	White	213095	4	\$3,500.00

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

```
# use str.replace("[\$,\\.]", "")
```

```
# refer the link to study Punctuation replcemenet https://blog.enterprisedna.co/
```

```
sales['Price'].replace("[\$,\\.]", "", regex=True, inplace=True)
```

```
sales.head()
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	400000
1	Honda	Red	87899	4	500000
2	Toyota	Blue	32549	3	700000
3	BMW	Black	11179	5	2200000
4	Nissan	White	213095	4	350000



```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Make            10 non-null     object
1   Colour          10 non-null     object
2   Odometer (KM)    10 non-null     int64
3   Doors           10 non-null     int64
4   Price           10 non-null     object
dtypes: int64(2), object(3)
memory usage: 528.0+ bytes
```

```
# Check the changes to the price column
sales['Price'].head()
```

```
0    400000
1    500000
2    700000
3   2200000
4    350000
Name: Price, dtype: object
```

```
# Remove the two extra zeros at the end of the price column
sales.head()
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	400000
1	Honda	Red	87899	4	500000
2	Toyota	Blue	32549	3	700000
4	Nissan	White	213095	4	350000

Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#)

```
# Check the changes to the Price column
sales['Price'].replace(".{2}$", "", regex=True, inplace=True)
```

```
sales['Price']
```

```
0    4000
1    5000
2    7000
3   22000
4    3500
5    4500
6    7500
7    7000
```

```
/      /000
8      6250
9      9700
Name: Price, dtype: object
```

```
# Change the datatype of the Price column to integers
sales['Price'] = sales['Price'].astype('int64')
```

```
sales['Price']

0      4000
1      5000
2      7000
3     22000
4      3500
5      4500
6      7500
7      7000
8      6250
9      9700
Name: Price, dtype: int64
```

```
# Lower the strings of the Make column
sales['Make'].str.lower()
```

```
0      toyota
1      honda
2      toyota
3       bmw
4      nissan
5      toyota
6      honda
7      honda
8      toyota
9      nissan
Name: Make, dtype: object
```

Automatic saving failed. This file was updated remotely or in another tab.

[Show diff](#)

```
0      Toyota
1      Honda
2      Toyota
3       BMW
4      Nissan
5      Toyota
6      Honda
7      Honda
8      Toyota
9      Nissan
Name: Make, dtype: object
```

If you check the car sales DataFrame, you'll notice the Make column hasn't been lowered.

How could you make these changes permanent?

Try it out.

```
# Make lowering the case of the Make column permanent
sales['Make'] = sales['Make'].str.lower()
```

```
# Check the car sales DataFrame
sales['Make']
```

```
0    toyota
1     honda
2    toyota
3      bmw
4    nissan
5    toyota
6     honda
7     honda
8    toyota
9    nissan
Name: Make, dtype: object
```

## Extensions

For more exercises, check out the pandas documentation, particularly the [10-minutes to pandas section](#).

One great exercise would be to retype out the entire section into a Jupyter Notebook of your own.

Get hands-on with the code and see what it does.

The next place you should check out are the [top questions and answers on Stack Overflow for pandas](#). Often, these contain some of the most useful and common pandas functions. Be sure to play around with the different filters!

Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#) mistakes.

Ask questions, get things wrong, take note of the things you do most often. And don't worry if you keep making the same mistake, pandas has many ways to do the same thing and is a big library. So it'll likely take a while before you get the hang of it.

[Colab paid products](#) - [Cancel contracts here](#)

Automatic saving failed. This file was updated remotely or in another tab. [Show diff](#)

```
import pandas as pd
```

Double-click (or enter) to edit



### ▼ Combining two data frames

Question 1: Some of the orders are stored in another csv file named `megamart_new_sales`. Read the csv file, store it in a data frame and add it to the `megamart_sales` data frame. Find the total sales value of the category 'Office Supplies' after combining the dataframes



- a) 7970    <= Correct
- b) 6964
- c) 7494
- d) 6076

```
megamart_sales = pd.read_csv("MegaMart_sales.csv")
megamart_new_sales = pd.read_csv("MegaMart_newsales.csv")
```

```
megamart_sales.describe()
```

	Discount	Sales	Profit	Quantity	
count	59.000000	59.000000	59.000000	59.000000	
mean	0.094915	300.830508	62.355932	3.898305	
std	0.184226	403.498158	132.271894	2.233714	
min	0.000000	10.000000	-98.000000	1.000000	
25%	0.000000	38.500000	3.000000	2.000000	
50%	0.000000	98.000000	15.000000	3.000000	
75%	0.100000	403.000000	104.000000	5.000000	
max	0.600000	1908.000000	820.000000	12.000000	

```
megamart_new_sales.describe()
```

	Discount	Sales	Profit	Quantity	
count	9.000000	9.000000	9.000000	9.000000	
mean	0.044444	141.000000	30.000000	3.888889	
std	0.133333	102.701022	31.622777	1.833333	
min	0.000000	52.000000	1.000000	2.000000	



✓ 0s completed at 11:08 AM

● ×

50%	0.000000	97.000000	19.000000	4.000000
75%	0.000000	136.000000	40.000000	5.000000
max	0.400000	339.000000	102.000000	7.000000

#your code here

```
megamart_final_sales = pd.concat([megamart_sales, megamart_new_sales])
megamart_final_sales.describe()
```

	Discount	Sales	Profit	Quantity	
count	68.000000	68.000000	68.000000	68.000000	
mean	0.088235	279.676471	58.073529	3.897059	
std	0.178325	381.021543	124.044686	2.172691	
min	0.000000	10.000000	-98.000000	1.000000	
25%	0.000000	45.500000	3.000000	2.000000	
50%	0.000000	97.500000	15.500000	3.000000	
75%	0.100000	315.000000	74.250000	5.000000	
max	0.600000	1908.000000	820.000000	12.000000	

megamart\_final\_sales.head()

	Order ID	Product Name	Discount	Sales	Profit	Quantity	Category
0	AZ-2011-1029887	Novimex Color Coded Labels, 5000 Label Set	0.0	26	7	2	Office Supplies
1	AZ-2011-107716	Deflect-O Desk Chair	0.0	85	15	2	Furniture

```
megamart_final_sales[megamart_final_sales['Category'] == 'Office Supplies']['Sales']
```

7970

Dataframe.duplicated() method of Pandas.

Syntax : DataFrame.duplicated(subset = None, keep = 'first') Parameters: subset: This Takes a column or list of column label. It's default value is None. After passing columns, it will consider them only for duplicates. keep: This Controls how to consider duplicate value. It has only three distinct value and default is 'first'.

If 'first', This considers first value as unique and rest of the same values as duplicate. If 'last', This considers last value as unique and rest of the same values as duplicate. If 'False', This considers all of the same values as duplicates. Returns: Boolean Series denoting duplicate rows.



## Dropping duplicates

Question 2: There are some duplicate rows in the data frame. Drop these rows and calculate the total sales value of the category Office Supplies.

- a) 7156
- b) 6496
- c) 6964    <= Correct
- d) 6023

#your code here

```
megamart_final_sales.drop_duplicates(inplace=True)
megamart_final_sales.describe()
```

	Discount	Sales	Profit	Quantity	
<b>count</b>	61.000000	61.000000	61.000000	61.000000	
<b>mean</b>	0.080328	277.344262	55.393443	3.852459	
<b>std</b>	0.171094	387.918586	125.951880	2.204813	
<b>min</b>	0.000000	10.000000	-98.000000	1.000000	
<b>25%</b>	0.000000	52.000000	6.000000	2.000000	
<b>50%</b>	0.000000	97.000000	15.000000	3.000000	
<b>75%</b>	0.000000	307.000000	65.000000	5.000000	
<b>max</b>	0.600000	1908.000000	820.000000	12.000000	

```
megamart_final_sales[megamart_final_sales['Category'] == 'Office Supplies']['Sales']
```

6964

`df.drop_duplicates()` `DataFrame.drop_duplicates(subset=None, *, keep='first', inplace=False, ignore_index=False)` [source] Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

Parameters `subset` column label or sequence of labels, optional Only consider certain columns for identifying duplicates, by default use all of the columns.

`keep={'first', 'last', False}`, default 'first' Determines which duplicates (if any) to keep.

'first' : Drop duplicates except for the first occurrence.

'last' : Drop duplicates except for the last occurrence.

False : Drop all duplicates.

`inplacebool`, default False Whether to modify the DataFrame rather than creating a new one.

`ignore_indexbool`, default False If True, the resulting axis will be labeled 0, 1, ..., n - 1.

Returns DataFrame or None DataFrame with duplicates removed or None if `inplace=True`.

```
megamart_final_sales.shape
```

```
(61, 8)
```

```
megamart_final_sales.groupby('Category')['Sales'].sum()
```

```
Category
Furniture      2460
Office Supplies 6964
Technology     7494
Name: Sales, dtype: int64
```

## Best category-sub category

Question 3: Find the most profitable category and sub category combination based on the net profit.

- a) Furniture-Bookcases
- b) Office supplies-Appliances
- c) Office supplies-Storage
- d) Technology-Phones <= Correct

```
#your code here
```

```
megamart_final_sales.groupby(['Category', 'Sub-Category'])['Sales'].sum().idxmax
```

```
('Technology', 'Phones')
```

## Invalid order IDs

Question 4: How many invalid order IDs are there in the data frame? An order id is of the form AZ-2011-Y where Y represents a whole number. A Order ID is said to be valid only if Y



consists of 7 digits. Find the number of invalid order IDs in the data frame.

- a)6
- b)7 Correct
- c)8
- d)9

```
megamart_final_sales.head()
```

	Order ID	Product Name	Discount	Sales	Profit	Quantity	Category
0	AZ-2011-1029887	Novimex Color Coded Labels, 5000 Label Set	0.0	26	7	2	Office Supplies
1	AZ-2011-107716	Deflect-O Door Stop	0.0	85	15	2	Furniture

```
#your code here
```

```
def filterValidOrderId(orderid):  
    orderidValidationPart = orderid.split('-')[2]  
    # print(orderidValidationPart)  
    return len(orderidValidationPart) != 7
```

```
megamart_final_sales_invalid_orderid = megamart_final_sales.loc[megamart_final_sales['Order ID'].apply(filterValidOrderId) == False]  
megamart_final_sales_invalid_orderid
```

	Order ID	Product Name	Discount	Sales	Profit	Quantity	Category
1	AZ-2011-107716	Deflect-O Door Stop, Ergonomic	0.0	85	15	2	Furniture
9	AZ-2011-122598	Avery Removable Labels, Alphabetical	0.0	32	6	3	Office Supplies
17	AZ-2011-130330	Office Star Chairmat, Adjustable	0.1	307	99	5	Furniture

```
len(megamart_final_sales_invalid_orderid)
```

```
7
```

Double-click (or enter) to edit

## Occurence of furniture in top 25 sales

Question 5: Find the top 25 orders based on sales value and find the number of orders which belong to furniture category.

- a)2
- b)3
- c)4
- d)5 <= Correct

#your code here

```
top25_orders = megamart_final_sales.sort_values(by=['Sales'], ascending=False).head(25)
top25_orders.loc[top25_orders['Category'] == 'Furniture']
```

	Order ID	Product Name	Discount	Sales	Profit	Quantity	Category
<b>31</b>	AZ-2011-144325	Bush Stackable Bookrack, Pine	0.0	630	132	5	Furniture
<b>12</b>	AZ-2011-1253407	Safco Stackable Bookrack, Pine	0.1	541	156	4	Furniture

```
len(top25_orders.loc[top25_orders['Category'] == 'Furniture'])
```

5

## And operation

Question 6: Among the orders with sales>250 and profit>50, find the product name of the fourth highest order based on sales value.

- a)Motorola Headset, with Caller ID => Correct
- b)Panasonic Printer, Durable
- c)Hoover Microwave, Red
- d)Fellowes Lockers, Industrial

#your code here

```
orders_in_range = megamart_final_sales.loc[(megamart_final_sales['Sales'] > 250)
orders_in_range_sorted = orders_in_range.sort_values(by=['Sales'], ascending=False)
orders_in_range_sorted.iloc[3]['Product Name']
```

'Motorola Headset, with Caller ID'

## Column manipulation

Question 7: Remove the orders with negative profit by dropping the corresponding rows with negative Profit. Find the product that makes the lowest profit per Quantity in the Technology category.

- a) Nokia Audio Dock, with Caller ID
- b) Logitech Keyboard, Programmable
- c) Motorola Headset, with Caller ID
- d) Belkin Flash Drive, Bluetooth

```
megamart_final_sales.head()
```

	Order ID	Product Name	Discount	Sales	Profit	Quantity	Category
0	AZ-2011-1029887	Novimex Color Coded Labels, 5000 Label Set	0.0	26	7	2	Office Supplies
1	AZ-2011-107716	Deflect-O Door Stop, Ergonomic	0.0	85	15	2	Furniture
		Belkin Flash					

```
len(megamart_final_sales)
```

61

```
final_sales["Profit"]/megamart_final_sales["Quantity"]
megamart_final_sales["Profit"]>0) & (megamart_final_sales["Category"]=="Technology")][0]
```

'Novimex Steel Folding Chair, Red'

[Colab paid products](#) - [Cancel contracts here](#)