


# Pandas ¶

Pandas is a library built using NumPy specifically for data analysis. You'll be using Pandas heavily for data manipulation, visualisation, building machine learning models, etc.

There are two main data structures in Pandas - Series and Dataframes. The default way to store data is dataframes, and thus manipulating dataframes quickly is probably the most important skill set for data analysis.

 source: <https://pandas.pydata.org/pandas-docs/stable/overview.html> (<https://pandas.pydata.org/pandas-docs/stable/overview.html>).

In this section, you will study:

1. The pandas Series (similar to a numpy array)
  - Creating a pandas series
  - Indexing series
2. Dataframes
  - Creating dataframes from dictionaries
  - Importing CSV data files as pandas dataframes
  - Reading and summarising dataframes
  - Sorting dataframes

## 1. The Pandas Series

A series is similar to a 1-D numpy array, and contains scalar values of the same type (numeric, character, datetime etc.). A dataframe is simply a table where each column is a pandas series.

### Creating Pandas Series

Series are one-dimensional array-like structures, though unlike numpy arrays, they often contain non-numeric data (characters, dates, time, booleans etc.)

You can create pandas series from array-like objects using `pd.Series()`.

```
In [1]: # import pandas, pd is an alias
import pandas as pd

# Creating a numeric pandas series
s = pd.Series([2, 4, 5, 6, 9])
print(s)
print(type(s))
```

```
0    2
1    4
2    5
3    6
4    9
dtype: int64
<class 'pandas.core.series.Series'>
```

Note that each element in the Series has an index, and the index starts at 0 as usual.

```
In [2]: # creating a series of characters
# notice that the 'dtype' here is 'object'
char_series = pd.Series(['a', 'b', 'af'])
char_series
```

```
Out[2]: 0    a
1    b
2   af
dtype: object
```

```
In [3]: # creating a series of type datetime
date_series = pd.date_range(start = '11-09-2017', end = '12-12-2017')
date_series
type(date_series)
```

```
Out[3]: pandas.core.indexes.datetimes.DatetimeIndex
```

## Indexing Series

Indexing series is exactly same as 1-D numpy arrays - index starts at 0.

```
In [4]: # Indexing pandas series: Same as indexing 1-d numpy arrays or lists
# accessing the fourth element
s[3]

# accessing elements starting index = 2 till the end
s[2:]
```

```
Out[4]: 2    5
3    6
4    9
dtype: int64
```

```
In [5]: # accessing the second and the fourth elements
# note that s[1, 3] will not work, you need to pass the indices [1, 3] as a list inside the original []
s[[1, 3]]
```

```
Out[5]: 1    4
        3    6
        dtype: int64
```

Usually, you will work with Series only as a part of dataframes. Let's study the basics of dataframes.

## The Pandas Dataframe

Dataframe is the most widely used data-structure in data analysis. It is a table with rows and columns, with rows having an index and columns having meaningful names.

### Creating dataframes from dictionaries

There are various ways of creating dataframes, such as creating them from dictionaries, JSON objects, reading from txt, CSV files, etc.

```
In [6]: # keys become column names
df = pd.DataFrame({'name': ['Vinay', 'Kushal', 'Aman', 'Saif'],
                  'age': [22, 25, 24, 28],
                  'occupation': ['engineer', 'doctor', 'data analyst', 'teacher']})
df
```

Out[6]:

	age	name	occupation
0	22	Vinay	engineer
1	25	Kushal	doctor
2	24	Aman	data analyst
3	28	Saif	teacher

### Importing CSV data files as pandas dataframes

For the upcoming exercises, we will use a dataset of a retail store having details about the orders placed, customers, product details, sales, profits etc.

```
In [7]: # reading a CSV file as a dataframe
market_df = pd.read_csv("../global_sales_data/market_fact.csv")
```

Usually, dataframes are imported as CSV files, but sometimes it is more convenient to convert dictionaries into dataframes. For e.g. when the raw data is in a JSON format (which is not uncommon), you can easily convert it into a dictionary, and then into a dataframe.

You will learn how to convert JSON objects to dataframes later.

## Reading and Summarising Dataframes

After you import a dataframe, you'd want to quickly understand its structure, shape, meanings of rows and columns etc. Further, you may want to look at summary statistics - such as mean, percentiles etc.

In [8]: *# Looking at the top and bottom entries of dataframes*  
market\_df.head()

Out[8]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87

In [9]: market\_df.tail()

Out[9]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	P
8394	Ord_5353	Prod_4	SHP_7479	Cust_1798	2841.4395	0.08	28	37
8395	Ord_5411	Prod_6	SHP_7555	Cust_1798	127.1600	0.10	20	-7
8396	Ord_5388	Prod_6	SHP_7524	Cust_1798	243.0500	0.02	39	-7
8397	Ord_5348	Prod_15	SHP_7469	Cust_1798	3872.8700	0.03	23	56
8398	Ord_5459	Prod_6	SHP_7628	Cust_1798	603.6900	0.00	47	13

Here, each row represents an order placed at a retail store. Notice the index associated with each row - starts at 0 and ends at 8398, implying that there were 8399 orders placed.

```
In [10]: # Looking at the datatypes of each column
market_df.info()

# Note that each column is basically a pandas Series of length 8399
# The ID columns are 'objects', i.e. they are being read as characters
# The rest are numeric (floats or int)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8399 entries, 0 to 8398
Data columns (total 10 columns):
Ord_id          8399 non-null object
Prod_id         8399 non-null object
Ship_id         8399 non-null object
Cust_id         8399 non-null object
Sales           8399 non-null float64
Discount        8399 non-null float64
Order_Quantity  8399 non-null int64
Profit          8399 non-null float64
Shipping_Cost   8399 non-null float64
Product_Base_Margin 8336 non-null float64
dtypes: float64(5), int64(1), object(4)
memory usage: 656.2+ KB
```

```
In [11]: # Describe gives you a summary of all the numeric columns in the dataset
market_df.describe()
```

Out[11]:

	Sales	Discount	Order_Quantity	Profit	Shipping_Cost	Product_Base_Margin
count	8399.000000	8399.000000	8399.000000	8399.000000	8399.000000	8336.000000
mean	1775.878179	0.049671	25.571735	181.184424	12.838557	0.512000
std	3585.050525	0.031823	14.481071	1196.653371	17.264052	0.135000
min	2.240000	0.000000	1.000000	-14140.700000	0.490000	0.350000
25%	143.195000	0.020000	13.000000	-83.315000	3.300000	0.380000
50%	449.420000	0.050000	26.000000	-1.500000	6.070000	0.520000
75%	1709.320000	0.080000	38.000000	162.750000	13.990000	0.590000
max	89061.050000	0.250000	50.000000	27220.690000	164.730000	0.850000

```
In [12]: # Column names
market_df.columns
```

```
Out[12]: Index(['Ord_id', 'Prod_id', 'Ship_id', 'Cust_id', 'Sales', 'Discount',
               'Order_Quantity', 'Profit', 'Shipping_Cost', 'Product_Base_Margin'],
              dtype='object')
```

```
In [13]: # The number of rows and columns
market_df.shape
```

```
Out[13]: (8399, 10)
```

```
In [14]: # You can extract the values of a dataframe as a numpy array using df.values
market_df.values
```

```
Out[14]: array([[ 'Ord_5446', 'Prod_16', 'SHP_7609', ..., -30.51, 3.6, 0.56],
                [ 'Ord_5406', 'Prod_13', 'SHP_7549', ...,  4.56, 0.93, 0.54],
                [ 'Ord_5446', 'Prod_4', 'SHP_7610', ..., 1148.9, 2.5, 0.59],
                ...,
                [ 'Ord_5388', 'Prod_6', 'SHP_7524', ..., -70.85, 5.35, 0.4],
                [ 'Ord_5348', 'Prod_15', 'SHP_7469', ..., 565.34, 30.0, 0.62],
                [ 'Ord_5459', 'Prod_6', 'SHP_7628', ..., 131.39, 4.86, 0.38]], dtype=object)
```

## Indices

An important concept in pandas dataframes is that of *row indices*. By default, each row is assigned indices starting from 0, and are represented at the left side of the dataframe.

```
In [15]: market_df.head()
```

```
Out[15]:
```

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51
1	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56
2	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
3	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34
4	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87

Now, arbitrary numeric indices are difficult to read and work with. Thus, you may want to change the indices of the df to something more meaningful.

Let's change the index to Ord\_id (unique id of each order), so that you can select rows using the order ids directly.

```
In [16]: # Setting index to Ord_id
market_df.set_index('Ord_id', inplace = True)
market_df.head()
```

Out[16]:

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	S
Ord_id								
Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.51	3.
Ord_5406	Prod_13	SHP_7549	Cust_1818	42.27	0.01	13	4.56	0.
Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90	2.
Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.89	0.09	43	729.34	1.
Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.15	0.08	35	1219.87	2.

Having meaningful row labels as indices helps you to select (subset) dataframes easily. You will study selecting dataframes in the next section.

## Sorting dataframes

You can sort dataframes in two ways - 1) by the indices and 2) by the values.

```
In [17]: # Sorting by index  
# axis = 0 indicates that you want to sort rows (use axis=1 for columns)  
market_df.sort_index(axis = 0, ascending = False)
```



Out[17]:

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
Ord_id							
Ord_999	Prod_15	SHP_1383	Cust_361	5661.0800	0.00	33	1055.47
Ord_998	Prod_8	SHP_1380	Cust_372	750.6600	0.00	33	120.05
Ord_998	Prod_5	SHP_1382	Cust_372	2149.3700	0.03	42	217.87
Ord_998	Prod_8	SHP_1381	Cust_372	254.3200	0.01	8	-117.39
Ord_997	Prod_14	SHP_1379	Cust_365	28761.5200	0.04	8	285.11
Ord_996	Prod_7	SHP_1378	Cust_371	35.6400	0.05	10	-0.71
Ord_996	Prod_5	SHP_1377	Cust_371	3202.2500	0.09	44	991.26
Ord_996	Prod_6	SHP_1378	Cust_371	81.2500	0.01	11	-44.54
Ord_996	Prod_13	SHP_1378	Cust_371	57.1700	0.08	18	-24.03
Ord_996	Prod_13	SHP_1378	Cust_371	41.9700	0.05	12	-37.03
Ord_995	Prod_4	SHP_1376	Cust_370	7965.9025	0.01	46	2311.96
Ord_994	Prod_1	SHP_1375	Cust_369	11103.7500	0.10	32	-522.59
Ord_993	Prod_1	SHP_1374	Cust_368	2645.8000	0.00	31	-684.78
Ord_993	Prod_5	SHP_1373	Cust_368	1140.2600	0.04	39	387.20
Ord_992	Prod_15	SHP_1370	Cust_368	5041.4600	0.06	43	-2136.66
Ord_992	Prod_11	SHP_1372	Cust_368	8817.7100	0.09	27	-2946.05
Ord_992	Prod_6	SHP_1371	Cust_368	282.9800	0.06	45	-237.47
Ord_991	Prod_4	SHP_1369	Cust_367	760.2485	0.04	16	29.80
Ord_990	Prod_11	SHP_1368	Cust_366	1042.0600	0.04	14	-931.25
Ord_990	Prod_6	SHP_1367	Cust_366	492.7100	0.02	24	67.01
Ord_99	Prod_8	SHP_136	Cust_72	311.4400	0.04	32	74.07
Ord_99	Prod_5	SHP_137	Cust_72	215.6500	0.01	17	77.38
Ord_989	Prod_3	SHP_1366	Cust_360	64.1300	0.10	10	-28.97
Ord_988	Prod_7	SHP_1364	Cust_360	90.9600	0.02	48	-30.11
Ord_988	Prod_5	SHP_1364	Cust_360	632.5500	0.05	27	165.35
Ord_988	Prod_5	SHP_1365	Cust_360	1997.1300	0.05	32	720.45
Ord_987	Prod_6	SHP_1362	Cust_365	905.3200	0.02	24	389.07
Ord_987	Prod_4	SHP_1363	Cust_365	1702.4990	0.08	16	162.11
Ord_986	Prod_3	SHP_1360	Cust_363	77.5700	0.00	19	-67.49
Ord_985	Prod_4	SHP_1359	Cust_362	1155.5835	0.04	24	-127.00
...	...	...	...	...	...	...	...

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
Ord_id							
Ord_1016	Prod_8	SHP_1406	Cust_383	138.0500	0.09	19	-100.49
Ord_1015	Prod_9	SHP_1405	Cust_370	124.0600	0.09	17	15.74
Ord_1014	Prod_5	SHP_1404	Cust_368	131.0900	0.01	10	10.94
Ord_1014	Prod_6	SHP_1404	Cust_368	1817.9000	0.06	33	700.31
Ord_1013	Prod_4	SHP_1403	Cust_382	1064.7865	0.10	12	-181.01
Ord_1012	Prod_3	SHP_1402	Cust_382	104.5100	0.04	13	-9.28
Ord_1011	Prod_3	SHP_1400	Cust_381	3465.9700	0.08	4	194.32
Ord_1011	Prod_8	SHP_1401	Cust_381	1307.8800	0.06	13	206.86
Ord_1010	Prod_11	SHP_1399	Cust_366	4051.8000	0.10	35	-684.07
Ord_1010	Prod_5	SHP_1398	Cust_366	195.2300	0.08	35	-72.46
Ord_101	Prod_1	SHP_139	Cust_73	300.2000	0.00	8	6.41
Ord_101	Prod_4	SHP_139	Cust_73	1364.8025	0.07	29	-183.68
Ord_1009	Prod_17	SHP_1396	Cust_366	2219.7900	0.10	30	-9.45
Ord_1009	Prod_6	SHP_1397	Cust_366	237.3600	0.07	25	-117.86
Ord_1008	Prod_5	SHP_1395	Cust_380	689.6700	0.07	38	51.47
Ord_1007	Prod_4	SHP_1394	Cust_379	2766.8180	0.03	47	846.95
Ord_1006	Prod_2	SHP_1393	Cust_378	1758.4100	0.06	18	238.52
Ord_1005	Prod_13	SHP_1392	Cust_377	35.4800	0.09	14	-9.19
Ord_1005	Prod_16	SHP_1392	Cust_377	15.0000	0.05	1	-11.02
Ord_1004	Prod_5	SHP_1390	Cust_365	199.5200	0.08	28	13.44
Ord_1004	Prod_3	SHP_1389	Cust_365	139.5400	0.01	26	-54.08
Ord_1004	Prod_7	SHP_1391	Cust_365	85.6600	0.04	39	6.67
Ord_1003	Prod_9	SHP_1387	Cust_376	49.0400	0.02	8	9.29
Ord_1003	Prod_11	SHP_1388	Cust_376	2320.3500	0.00	6	-237.62
Ord_1002	Prod_15	SHP_1386	Cust_375	1756.4600	0.01	15	-101.19
Ord_1001	Prod_5	SHP_1385	Cust_374	1981.2600	0.07	49	100.80
Ord_1000	Prod_6	SHP_1384	Cust_373	334.7100	0.01	25	31.74
Ord_100	Prod_8	SHP_138	Cust_58	121.1200	0.10	3	-118.82
Ord_10	Prod_3	SHP_13	Cust_10	80.6100	0.02	15	-4.72
Ord_1	Prod_1	SHP_1	Cust_1	261.5400	0.04	6	-213.25

8399 rows × 9 columns

```
In [18]: # Sorting by values

# Sorting in increasing order of Sales
market_df.sort_values(by='Sales').head()
```

Out[18]:

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	Shipp
Ord_id								
Ord_704	Prod_7	SHP_964	Cust_242	2.24	0.01	1	-1.97	0.70
Ord_149	Prod_3	SHP_7028	Cust_1712	3.20	0.09	1	-3.16	1.49
Ord_4270	Prod_7	SHP_5959	Cust_1450	3.23	0.06	2	-2.73	0.70
Ord_4755	Prod_13	SHP_6628	Cust_1579	3.41	0.06	1	-1.78	0.70
Ord_2252	Prod_3	SHP_3064	Cust_881	3.42	0.05	1	-2.91	1.49

```
In [19]: # Sorting in decreasing order of Shipping_Cost
market_df.sort_values(by='Shipping_Cost', ascending = False).head()
```

Out[19]:

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
Ord_id							
Ord_1751	Prod_15	SHP_2426	Cust_597	14740.510	0.00	46	3407.73
Ord_839	Prod_11	SHP_1361	Cust_364	12689.870	0.04	44	-169.23
Ord_1741	Prod_11	SHP_2411	Cust_595	15168.820	0.02	26	-1096.78
Ord_417	Prod_11	SHP_561	Cust_156	20333.816	0.02	45	-1430.45
Ord_1581	Prod_15	SHP_2184	Cust_519	2573.920	0.07	17	117.23

```
In [20]: # Sorting by more than two columns  
  
# Sorting in ascending order of Sales for each Product  
market_df.sort_values(by=['Prod_id', 'Sales'], ascending = False)
```

Out[20]:

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	S
Ord_id								
Ord_2197	Prod_9	SHP_2994	Cust_827	7522.80	0.04	48	3187.37	1!
Ord_4356	Prod_9	SHP_6074	Cust_1481	6831.72	0.01	41	3081.02	1!
Ord_262	Prod_9	SHP_358	Cust_66	6553.45	0.03	39	2969.81	1!
Ord_4059	Prod_9	SHP_5660	Cust_1378	5587.20	0.05	36	2254.16	1!
Ord_2973	Prod_9	SHP_6073	Cust_1480	5410.95	0.09	36	2077.91	1!
Ord_950	Prod_9	SHP_1315	Cust_334	4906.85	0.09	32	1907.94	1!
Ord_5112	Prod_9	SHP_7141	Cust_1729	4273.95	0.05	49	1340.07	1!
Ord_612	Prod_9	SHP_836	Cust_687	3872.38	0.10	50	1110.35	1!
Ord_3443	Prod_9	SHP_4773	Cust_1246	3849.17	0.06	46	1982.78	5.
Ord_3650	Prod_9	SHP_6795	Cust_1683	3353.54	0.07	22	1189.96	1!
Ord_4590	Prod_9	SHP_6385	Cust_1465	3212.97	0.03	37	1125.42	1!
Ord_2774	Prod_9	SHP_3809	Cust_1015	3191.24	0.00	38	1620.23	5.
Ord_793	Prod_9	SHP_1089	Cust_245	2833.19	0.06	34	1409.87	5.
Ord_3071	Prod_9	SHP_4265	Cust_1141	2728.42	0.06	45	664.15	1!
Ord_1290	Prod_9	SHP_1780	Cust_482	2609.53	0.00	27	911.66	1!
Ord_555	Prod_9	SHP_754	Cust_202	2568.71	0.10	43	590.77	1!
Ord_2853	Prod_9	SHP_3927	Cust_1056	2366.18	0.10	16	703.80	1!
Ord_4370	Prod_9	SHP_6092	Cust_1492	2325.42	0.02	25	739.91	1!
Ord_3239	Prod_9	SHP_4491	Cust_1205	2300.45	0.02	36	624.64	1!
Ord_2968	Prod_9	SHP_4092	Cust_1066	2160.83	0.08	36	485.02	1!
Ord_4989	Prod_9	SHP_6965	Cust_1700	2048.78	0.05	24	994.68	5.
Ord_4663	Prod_9	SHP_6498	Cust_1612	2026.42	0.03	50	655.36	9.
Ord_1152	Prod_9	SHP_1586	Cust_443	1554.53	0.00	16	474.66	1!
Ord_5370	Prod_9	SHP_7500	Cust_1809	1527.42	0.04	25	326.40	1!
Ord_3917	Prod_9	SHP_5439	Cust_1337	1504.01	0.05	42	211.04	1!
Ord_4595	Prod_9	SHP_6390	Cust_1523	1500.84	0.10	46	91.56	1!
Ord_3389	Prod_9	SHP_4698	Cust_1236	1484.89	0.03	41	220.99	1!
Ord_604	Prod_9	SHP_826	Cust_213	1406.64	0.02	50	424.36	8.
Ord_2737	Prod_9	SHP_3755	Cust_994	1400.91	0.09	49	-163.53	1!
Ord_2461	Prod_9	SHP_3373	Cust_941	1396.75	0.04	46	-75.70	1!
...	...	...	...	...	...	...	...	...

	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	S
Ord_id								
Ord_1666	Prod_1	SHP_2303	Cust_490	101.52	0.02	6	-16.37	5.
Ord_3693	Prod_1	SHP_5118	Cust_1287	96.03	0.10	6	-59.39	1.
Ord_2133	Prod_1	SHP_2910	Cust_808	96.01	0.01	11	-43.24	6.
Ord_3579	Prod_1	SHP_4956	Cust_1272	94.99	0.06	13	-106.29	9.
Ord_1654	Prod_1	SHP_2284	Cust_491	90.58	0.06	7	-27.83	6.
Ord_5479	Prod_1	SHP_7655	Cust_1805	88.94	0.05	5	-47.54	7.
Ord_3506	Prod_1	SHP_4857	Cust_395	79.93	0.00	2	-52.73	1.
Ord_1858	Prod_1	SHP_2820	Cust_720	73.86	0.08	7	-52.49	9.
Ord_75	Prod_1	SHP_99	Cust_50	73.07	0.08	4	-41.01	1.
Ord_5107	Prod_1	SHP_7132	Cust_1708	72.72	0.06	4	-36.61	11.
Ord_2725	Prod_1	SHP_3736	Cust_997	70.02	0.00	4	-30.63	8.
Ord_1933	Prod_1	SHP_2649	Cust_707	69.57	0.05	8	-36.06	6.
Ord_4613	Prod_1	SHP_6420	Cust_1569	68.03	0.02	4	-29.42	4.
Ord_3379	Prod_1	SHP_4685	Cust_1233	66.92	0.01	5	-23.29	6.
Ord_542	Prod_1	SHP_735	Cust_188	63.84	0.02	3	-47.97	1.
Ord_210	Prod_1	SHP_288	Cust_94	63.52	0.08	4	-8.47	4.
Ord_4618	Prod_1	SHP_6432	Cust_1575	58.90	0.03	5	-36.90	9.
Ord_212	Prod_1	SHP_291	Cust_88	58.33	0.02	3	-35.08	1.
Ord_2410	Prod_1	SHP_3307	Cust_963	57.48	0.00	3	-28.09	8.
Ord_4213	Prod_1	SHP_5875	Cust_1428	53.30	0.00	2	-46.20	1.
Ord_4079	Prod_1	SHP_5686	Cust_1386	52.59	0.05	4	-30.39	8.
Ord_362	Prod_1	SHP_481	Cust_144	47.01	0.09	1	-19.32	1.
Ord_130	Prod_1	SHP_3971	Cust_1064	40.06	0.07	6	-24.44	4.
Ord_1971	Prod_1	SHP_3185	Cust_919	32.60	0.03	2	-20.57	5.
Ord_1830	Prod_1	SHP_2524	Cust_634	29.65	0.05	1	-16.54	7.
Ord_3713	Prod_1	SHP_5145	Cust_1307	27.83	0.09	2	-22.14	9.
Ord_2746	Prod_1	SHP_3767	Cust_1030	22.61	0.03	1	-8.40	7.
Ord_439	Prod_1	SHP_587	Cust_136	18.73	0.05	1	-6.68	6.
Ord_2314	Prod_1	SHP_3171	Cust_899	18.16	0.03	1	-7.25	6.
Ord_286	Prod_1	SHP_387	Cust_90	18.15	0.04	1	-7.26	6.

8399 rows × 9 columns

