Indexing and Selecting Data

In this section, you will:

- Select rows from a dataframe
- Select columns from a dataframe
- Select subsets of dataframes

Selecting Rows

Selecting rows in dataframes is similar to the indexing you have seen in numpy arrays. The syntax `df[start_index:end_index]` will subset rows according to the start and end indices.

In [1]:
```
import numpy as np
import pandas as pd

market_df = pd.read_csv("../global_sales_data/market_fact.csv")
market_df.head()
```

Out[1]:

	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

Notice that, by default, pandas assigns integer labels to the rows, starting at 0.

In [2]:
```
# Selecting the rows from indices 2 to 6
market_df[2:7]
```

Out[2]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit
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
5	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
6	Ord_31	Prod_12	SHP_41	Cust_26	14.76	0.01	5	1.32

In [3]:
```
# Selecting alternate rows starting from index = 5
market_df[5::2].head()
```

Out[3]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Pr
5	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.0200	0.03	23	-47.6
7	Ord_4725	Prod_4	SHP_6593	Cust_1641	3410.1575	0.10	48	1137
9	Ord_4725	Prod_6	SHP_6593	Cust_1641	57.2200	0.07	8	-27.7
11	Ord_1925	Prod_6	SHP_2637	Cust_708	465.9000	0.05	38	79.3
13	Ord_2207	Prod_11	SHP_3093	Cust_839	3364.2480	0.10	15	-693.

Selecting Columns

There are two simple ways to select a single column from a dataframe - `df['column_name']` and `df.column_name`.

In [4]:
```
# Using df['column']
sales = market_df['Sales']
sales.head()
```

Out[4]:
```
0     136.81
1      42.27
2    4701.69
3    2337.89
4    4233.15
Name: Sales, dtype: float64
```

In [5]:
```
# Using df.column
sales = market_df.Sales
sales.head()
```

Out[5]:
```
0     136.81
1      42.27
2    4701.69
3    2337.89
4    4233.15
Name: Sales, dtype: float64
```

In [6]:
```
# Notice that in both these cases, the resultant is a Series object
print(type(market_df['Sales']))
print(type(market_df.Sales))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

Selecting Multiple Columns

You can select multiple columns by passing the list of column names inside the `[]`: `df[['column_1', 'column_2', 'column_n']]`.

For instance, to select only the columns Cust_id, Sales and Profit:

In [7]:
```
# Select Cust_id, Sales and Profit:
market_df[['Cust_id', 'Sales', 'Profit']].head()
```

Out[7]:

	Cust_id	Sales	Profit
0	Cust_1818	136.81	-30.51
1	Cust_1818	42.27	4.56
2	Cust_1818	4701.69	1148.90
3	Cust_1818	2337.89	729.34
4	Cust_1818	4233.15	1219.87

Notice that in this case, the output is itself a dataframe.

In [8]:
```
type(market_df[['Cust_id', 'Sales', 'Profit']])
```

Out[8]:
```
pandas.core.frame.DataFrame
```

In [9]:
```
# Similarly, if you select one column using double square brackets,
# you'll get a df, not Series

type(market_df[['Sales']])
```

Out[9]:
```
pandas.core.frame.DataFrame
```

Selecting Subsets of Dataframes

Until now, you have seen selecting rows and columns using the following ways:

- Selecting rows: `df[start:stop]`
- Selecting columns: `df['column']` or `df.column` or `df[['col_x', 'col_y']]`
  - `df['column']` or `df.column` return a series
  - `df[['col_x', 'col_y']]` returns a dataframe

But pandas does not prefer this way of indexing dataframes, since it has some ambiguity. For instance, let's try and select the third row of the dataframe.

In [10]:
```
# Trying to select the third row: Throws an error
market_df[2]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2521             try:
-> 2522                 return self._engine.get_loc(key)
   2523             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 2

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-10-481b7b400f8c> in <module>()
      1 # Trying to select the third row: Throws an error
----> 2 market_df[2]

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/frame.py in __getitem__(self, key)
   2137             return self._getitem_multilevel(key)
   2138         else:
-> 2139             return self._getitem_column(key)
   2140
   2141     def _getitem_column(self, key):

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/frame.py in _getitem_column(self, key)
   2144         # get column
   2145         if self.columns.is_unique:
-> 2146             return self._get_item_cache(key)
   2147
   2148         # duplicate columns & possible reduce dimensionality

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/generic.py in _get_item_cache(self, item)
   1840         res = cache.get(item)
   1841         if res is None:
-> 1842             values = self._data.get(item)
   1843             res = self._box_item_values(item, values)
   1844             cache[item] = res

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/internals.py in get(self, item, fastpath)
   3836
   3837             if not isna(item):
-> 3838                 loc = self.items.get_loc(item)
   3839             else:
```

```
   3840                 indexer = np.arange(len(self.items))[isna(self.items)
]

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-package
s/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2522                 return self._engine.get_loc(key)
   2523             except KeyError:
-> 2524                 return self._engine.get_loc(self._maybe_cast_indexer(
key))
   2525
   2526         indexer = self.get_indexer([key], method=method, tolerance=to
lerance)

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHas
hTable.get_item()

KeyError: 2
```

Pandas throws an error because it is confused whether the `[2]` is an *index* or a *label*. Recall from the previous section that you can change the row indices.

In [ ]:
```
# Changing the row indices to Ord_id
market_df.set_index('Ord_id').head()
```

Now imagine you had a column with entries `[2, 4, 7, 8 ...]`, and you set that as the index. What should `df[2]` return? The second row, or the row with the index value = 2?

Taking an example from this dataset, say you decide to assign the `Order_Quantity` column as the index.

In [ ]:
```
market_df.set_index('Order_Quantity').head()
```

Now, what should `df[13]` return - the 14th row, or the row with index label 13 (i.e. the second row)?

Because of this and similar other ambiguities, pandas provides **explicit ways** to subset dataframes - position based indexing and label based indexing, which we'll study next.