Lambda Functions and Pivot Tables

Until now, we have not made any changes or modifications to the data. In this section, we will:

- Use lambda functions to create new and alter existing columns
- Use pandas pivot tables as an alternative to `df.groupby()` to summarise data

Let's first read all the files and create a `master_df`.

In [1]:
```
# Loading libraries and files
import numpy as np
import pandas as pd

market_df = pd.read_csv("../global_sales_data/market_fact.csv")
customer_df = pd.read_csv("../global_sales_data/cust_dimen.csv")
product_df = pd.read_csv("../global_sales_data/prod_dimen.csv")
shipping_df = pd.read_csv("../global_sales_data/shipping_dimen.csv")
orders_df = pd.read_csv("../global_sales_data/orders_dimen.csv")

# Merging the dataframes to create a master_df
df_1 = pd.merge(market_df, customer_df, how='inner', on='Cust_id')
df_2 = pd.merge(df_1, product_df, how='inner', on='Prod_id')
df_3 = pd.merge(df_2, shipping_df, how='inner', on='Ship_id')
master_df = pd.merge(df_3, orders_df, how='inner', on='Ord_id')

master_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_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.58

5 rows × 22 columns

Lambda Functions

Say you want to create a new column indicating whether a given order was profitable or not (1/0).

You need to apply a function which returns 1 if Profit > 0, else 0. This can be easily done using the `apply()` method on a column of the dataframe.

In [2]:
```
# Create a function to be applied
def is_positive(x):
    return x > 0

# Create a new column
master_df['is_profitable'] = master_df['Profit'].apply(is_positive)
master_df.head()
```

Out[2]:

	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_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.58

5 rows × 23 columns

The same can be done in just one line of code using lambda functions.

In [3]:
```
# Create a new column using a lambda function
master_df['is_profitable'] = master_df['Profit'].apply(lambda x: x > 0)
master_df.head()
```

Out[3]:

	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_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.90
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.64
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.12
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.58

5 rows × 23 columns

Now you can use the new column to compare the percentage of profitable orders across groups.

In [4]:
```
# Comparing percentage of profitable orders across customer segments
by_segment = master_df.groupby('Customer_Segment')
by_segment.is_profitable.mean()
```

Out[4]:
```
Customer_Segment
CONSUMER          0.500910
CORPORATE         0.481469
HOME OFFICE       0.498524
SMALL BUSINESS    0.496346
Name: is_profitable, dtype: float64
```

In [5]:
```
# Comparing percentage of profitable orders across product categories
by_category = master_df.groupby('Product_Category')
by_category.is_profitable.mean()
```

Out[5]:
```
Product_Category
FURNITURE          0.465197
OFFICE SUPPLIES    0.466161
TECHNOLOGY         0.573366
Name: is_profitable, dtype: float64
```

In FURNITURE, 46% orders are profitable, compared to 57% in TECHNOLOGY.

In [6]:
```
# You can also use apply and lambda to alter existing columns
# E.g. you want to see Profit as one decimal place
# apply the round() function
master_df['Profit'] = master_df['Profit'].apply(lambda x: round(x, 1))
master_df.head()
```

Out[6]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	S
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.5	3
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.9	2
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.6	6
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.1	3
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.6	3

5 rows × 23 columns

You sometimes need to create new columns using existing columns, for instance, say you want a column `Profit / Order_Quantity`.

In [7]:
```
# Creating a column Profit / Order_Quantity
master_df['profit_per_qty'] = master_df['Profit'] / master_df['Order_Quantity']
]
master_df.head()
```

Out[7]:

	Ord_id	Prod_id	Ship_id	Cust_id	Sales	Discount	Order_Quantity	Profit	S
0	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.81	0.01	23	-30.5	3
1	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.69	0.00	26	1148.9	2
2	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.02	0.03	23	-47.6	6
3	Ord_2978	Prod_16	SHP_4112	Cust_1088	305.05	0.04	27	23.1	3
4	Ord_5484	Prod_16	SHP_7663	Cust_1820	322.82	0.05	35	-17.6	3

5 rows × 24 columns

Pivot Tables

You may want to use pandas pivot tables as an alternative to `groupby()`. They provide Excel-like functionalities to create aggregate tables.

In [8]:
```
# Read documentation
help(pd.DataFrame.pivot_table)
```

```
Help on function pivot_table in module pandas.core.frame:

pivot_table(self, values=None, index=None, columns=None, aggfunc='mean', fill
_value=None, margins=False, dropna=True, margins_name='All')
    Create a spreadsheet-style pivot table as a DataFrame. The levels in
    the pivot table will be stored in MultiIndex objects (hierarchical
    indexes) on the index and columns of the result DataFrame

    Parameters
    ----------
    values : column to aggregate, optional
    index : column, Grouper, array, or list of the previous
        If an array is passed, it must be the same length as the data. The
        list can contain any of the other types (except list).
        Keys to group by on the pivot table index.  If an array is passed,
        it is being used as the same manner as column values.
    columns : column, Grouper, array, or list of the previous
        If an array is passed, it must be the same length as the data. The
        list can contain any of the other types (except list).
        Keys to group by on the pivot table column.  If an array is passed,
        it is being used as the same manner as column values.
    aggfunc : function or list of functions, default numpy.mean
        If list of functions passed, the resulting pivot table will have
        hierarchical columns whose top level are the function names
        (inferred from the function objects themselves)
    fill_value : scalar, default None
        Value to replace missing values with
    margins : boolean, default False
        Add all row / columns (e.g. for subtotal / grand totals)
    dropna : boolean, default True
        Do not include columns whose entries are all NaN
    margins_name : string, default 'All'
        Name of the row / column that will contain the totals
        when margins is True.

    Examples
    --------
    >>> df = pd.DataFrame({"A": ["foo", "foo", "foo", "foo", "foo",
    ...                          "bar", "bar", "bar", "bar"],
    ...                    "B": ["one", "one", "one", "two", "two",
    ...                          "one", "one", "two", "two"],
    ...                    "C": ["small", "large", "large", "small",
    ...                          "small", "large", "small", "small",
    ...                          "large"],
    ...                    "D": [1, 2, 2, 3, 3, 4, 5, 6, 7]})
    >>> df
         A    B      C  D
    0  foo  one  small  1
    1  foo  one  large  2
    2  foo  one  large  2
    3  foo  two  small  3
    4  foo  two  small  3
    5  bar  one  large  4
    6  bar  one  small  5
    7  bar  two  small  6
    8  bar  two  large  7
```

```
>>> table = pivot_table(df, values='D', index=['A', 'B'],
...                     columns=['C'], aggfunc=np.sum)
>>> table
... # doctest: +NORMALIZE_WHITESPACE
C        large  small
A   B
bar one    4.0    5.0
    two    7.0    6.0
foo one    4.0    1.0
    two    NaN    6.0

Returns
-------
table : DataFrame

See also
--------
DataFrame.pivot : pivot without aggregation that can handle
    non-numeric data
```

The general syntax is `pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', ...)`.

- `data` is a dataframe
- `values` contains the column to aggregate
- `index` is the row in the pivot table
- `columns` contains the columns you want in the pivot table
- `aggfunc` is the aggregate function

Let's see some examples.

In [9]:
```
# E.g. Compare average Sales across customer segments
master_df.pivot_table(values = 'Sales', index = 'Customer_Segment', aggfunc =
'mean')
```

Out[9]:

	Sales
Customer_Segment	
CONSUMER	1857.859965
CORPORATE	1787.680389
HOME OFFICE	1754.312931
SMALL BUSINESS	1698.124841

In [10]:
```
# E.g. compare total number of profitable orders across regions
# Note that since is_profitable is 1/0, we can directly compute the sum
master_df.pivot_table(values = 'is_profitable', index = 'Region', aggfunc = 's
um')
```

Out[10]:

	is_profitable
Region	
ATLANTIC	544.0
NORTHWEST TERRITORIES	194.0
NUNAVUT	38.0
ONTARIO	916.0
PRARIE	852.0
QUEBEC	360.0
WEST	969.0
YUKON	262.0

In [11]:
```
# Grouping by both rows and columns
# Compare the total profit across product categories and customer segments
# Since there are two categorical variables, we use both rows (index) and colu
mns
master_df.pivot_table(values = 'Profit',
                      index = 'Product_Category',
                      columns = 'Customer_Segment',
                      aggfunc = 'sum')
```

Out[11]:

Customer_Segment	CONSUMER	CORPORATE	HOME OFFICE	SMALL BUSINESS
Product_Category				
FURNITURE	42728.5	22008.3	23978.6	28717.5
OFFICE SUPPLIES	88532.4	203038.8	121145.6	105306.8
TECHNOLOGY	156700.1	374701.1	173230.6	181684.1

You don't necessarily need to specify all four arguments, since `pivot_table()` has some smart defaults. For instance, if you just provide `columns`, it will compute the **mean of all the numeric columns** across each column. For e.g.:

In [12]:
```
# Computes the mean of all numeric columns across categories
# Notice that the means of Order_IDs are meaningless
master_df.pivot_table(columns = 'Product_Category')
```

Out[12]:

Product_Category	FURNITURE	OFFICE SUPPLIES	TECHNOLOGY
Discount	0.049287	0.050230	0.048746
Order_ID_x	30128.711717	30128.122560	29464.891525
Order_ID_y	30128.711717	30128.122560	29464.891525
Order_Quantity	25.709977	25.656833	25.266344
Product_Base_Margin	0.598555	0.461270	0.556305
Profit	68.116531	112.369544	429.208668
Sales	3003.822820	814.048178	2897.941008
Shipping_Cost	30.883811	7.829829	8.954886
is_profitable	0.465197	0.466161	0.573366
profit_per_qty	-3.607020	1.736175	-52.274216