Pivot Tables

Pivoting data can sometimes help clarify relationships and connections.

Full documentation on a variety of related pivot methods:
https://pandas.pydata.org/docs/user_guide/reshaping.html

Data

In [1]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
df = pd.read_csv("D:\\Study\\Programming\\python\\Python course from udemy\\[GigaCourse.
Com] Udemy - 2022 Python for Machine Learning & Data Science Masterclass\\01 - Introducti
on to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\03-Pandas\\Sales_Funnel_CRM.csv")
df
```

Out[3]:

	Account Number	Company	Contact	Account Manager	Product	Licenses	Sale Price	Status
0	2123398	Google	Larry Pager	Edward Thorp	Analytics	150	2100000	Presented
1	2123398	Google	Larry Pager	Edward Thorp	Prediction	150	700000	Presented
2	2123398	Google	Larry Pager	Edward Thorp	Tracking	300	350000	Under Review
3	2192650	BOBO	Larry Pager	Edward Thorp	Analytics	150	2450000	Lost
4	420496	IKEA	Elon Tusk	Edward Thorp	Analytics	300	4550000	Won
5	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Analytics	300	2800000	Under Review
6	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Prediction	150	700000	Presented
7	1216870	Microsoft	Will Grates	Edward Thorp	Tracking	300	350000	Under Review
8	2200450	Walmart	Will Grates	Edward Thorp	Analytics	150	2450000	Lost
9	405886	Apple	Cindy Phoner	Claude Shannon	Analytics	300	4550000	Won
10	470248	Exxon Mobile	Cindy Phoner	Claude Shannon	Analytics	150	2100000	Presented
11	698032	ATT	Cindy Phoner	Claude Shannon	Tracking	150	350000	Under Review
12	698032	ATT	Cindy Phoner	Claude Shannon	Prediction	150	700000	Presented
13	902797	CVS Health	Emma Gordian	Claude Shannon	Tracking	450	490000	Won
14	2046943	Salesforce	Emma Gordian	Claude Shannon	Analytics	750	7000000	Won
15	2169499	Cisco	Emma Gordian	Claude Shannon	Analytics	300	4550000	Lost
16	2169499	Cisco	Emma Gordian	Claude Shannon	GPS Positioning	300	350000	Presented

The pivot() method

The pivot method reshapes data based on column values and reassignment of the index. Keep in mind, it doesn't always make sense to pivot data. In our machine learning lessons, we will see that our data doesn't need to be pivoted. Pivot methods are mainly for data analysis,visualization, and exploration.

Here is an image showing the idea behind a pivot() call:

In [5]:

```
help(pd.pivot)
```

```
Help on function pivot in module pandas.core.reshape.pivot:

pivot(data: 'DataFrame', index: 'IndexLabel | None' = None, columns: 'IndexLabel | None'
= None, values: 'IndexLabel | None' = None) -> 'DataFrame'
    Return reshaped DataFrame organized by given index / column values.

    Reshape data (produce a "pivot" table) based on column values. Uses
    unique values from specified `index` / `columns` to form axes of the
    resulting DataFrame. This function does not support data
    aggregation, multiple values will result in a MultiIndex in the
    columns. See the :ref:`User Guide <reshaping>` for more on reshaping.

    Parameters
    ----------
    data : DataFrame
    index : str or object or a list of str, optional
        Column to use to make new frame's index. If None, uses
        existing index.

        .. versionchanged:: 1.1.0
           Also accept list of index names.

    columns : str or object or a list of str
        Column to use to make new frame's columns.

        .. versionchanged:: 1.1.0
           Also accept list of columns names.

    values : str, object or a list of the previous, optional
        Column(s) to use for populating new frame's values. If not
        specified, all remaining columns will be used and the result will
        have hierarchically indexed columns.

    Returns
    -------
    DataFrame
        Returns reshaped DataFrame.

    Raises
    ------
    ValueError:
        When there are any `index`, `columns` combinations with multiple
        values. `DataFrame.pivot_table` when you need to aggregate.

    See Also
    --------
    DataFrame.pivot_table : Generalization of pivot that can handle
        duplicate values for one index/column pair.
    DataFrame.unstack : Pivot based on the index values instead of a
        column.
    wide_to_long : Wide panel to long format. Less flexible but more
        user-friendly than melt.

    Notes
    -----
    For finer-tuned control, see hierarchical indexing documentation along
    with the related stack/unstack methods.

    Reference :ref:`the user guide <reshaping.pivot>` for more examples.

    Examples
    --------
    >>> df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two',
    ...                            'two'],
    ...                    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
    ...                    'baz': [1, 2, 3, 4, 5, 6],
    ...                    'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
    >>> df
        foo   bar  baz  zoo
    0   one   A    1    x
    1   one   B    2    y
    2   one   C    3    z
    3   two   A    4    q
    4   two   B    5    w
    5   two   C    6    t

    >>> df.pivot(index='foo', columns='bar', values='baz')
    bar  A   B   C
    foo
    one  1   2   3
    two  4   5   6

    >>> df.pivot(index='foo', columns='bar')['baz']
    bar  A   B   C
    foo
    one  1   2   3
    two  4   5   6

    >>> df.pivot(index='foo', columns='bar', values=['baz', 'zoo'])
          baz       zoo
    bar   A  B  C   A  B  C
    foo
    one   1  2  3   x  y  z
    two   4  5  6   q  w  t
```

You could also assign a list of column names or a list of index names.

```
>>> df = pd.DataFrame({
...        "lev1": [1, 1, 1, 2, 2, 2],
...        "lev2": [1, 1, 2, 1, 1, 2],
...        "lev3": [1, 2, 1, 2, 1, 2],
...        "lev4": [1, 2, 3, 4, 5, 6],
...        "values": [0, 1, 2, 3, 4, 5]})
>>> df
    lev1 lev2 lev3 lev4 values
0   1    1    1    1    0
1   1    1    2    2    1
2   1    2    1    3    2
3   2    1    2    4    3
4   2    1    1    5    4
5   2    2    2    6    5

>>> df.pivot(index="lev1", columns=["lev2", "lev3"],values="values")
lev2    1         2
lev3    1    2    1    2
lev1
1     0.0  1.0  2.0  NaN
2     4.0  3.0  NaN  5.0

>>> df.pivot(index=["lev1", "lev2"], columns=["lev3"],values="values")
      lev3    1    2
lev1  lev2
   1     1  0.0  1.0
         2  2.0  NaN
   2     1  4.0  3.0
         2  NaN  5.0
```

A ValueError is raised if there are any duplicates.

```
>>> df = pd.DataFrame({"foo": ['one', 'one', 'two', 'two'],
...                    "bar": ['A', 'A', 'B', 'C'],
...                    "baz": [1, 2, 3, 4]})
>>> df
   foo bar  baz
0  one   A    1
1  one   A    2
2  two   B    3
3  two   C    4
```

Notice that the first two rows are the same for our `index`

```
    and `columns` arguments.

    >>> df.pivot(index='foo', columns='bar', values='baz')
    Traceback (most recent call last):
       ...
    ValueError: Index contains duplicate entries, cannot reshape
```

Note: Common Point of Confusion: Students often just randomly pass in index,column, and value choices in an attempt to see the changes. This often just leads to formatting errors. You should first go through this checklist BEFORE running a pivot():

- **What question are you trying to answer?**
- **What would a dataframe that answers the question look like? Does it need a pivot()**
- **What you want the resulting pivot to look like? Do you need all the original columns?**

In [20]:

```
df
```

Out[20]:

	Account Number	Company	Contact	Account Manager	Product	Licenses	Sale Price	Status
0	2123398	Google	Larry Pager	Edward Thorp	Analytics	150	2100000	Presented
1	2123398	Google	Larry Pager	Edward Thorp	Prediction	150	700000	Presented
2	2123398	Google	Larry Pager	Edward Thorp	Tracking	300	350000	Under Review
3	2192650	BOBO	Larry Pager	Edward Thorp	Analytics	150	2450000	Lost
4	420496	IKEA	Elon Tusk	Edward Thorp	Analytics	300	4550000	Won
5	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Analytics	300	2800000	Under Review
6	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Prediction	150	700000	Presented
7	1216870	Microsoft	Will Grates	Edward Thorp	Tracking	300	350000	Under Review
8	2200450	Walmart	Will Grates	Edward Thorp	Analytics	150	2450000	Lost
9	405886	Apple	Cindy Phoner	Claude Shannon	Analytics	300	4550000	Won
10	470248	Exxon Mobile	Cindy Phoner	Claude Shannon	Analytics	150	2100000	Presented
11	698032	ATT	Cindy Phoner	Claude Shannon	Tracking	150	350000	Under Review
12	698032	ATT	Cindy Phoner	Claude Shannon	Prediction	150	700000	Presented
13	902797	CVS Health	Emma Gordian	Claude Shannon	Tracking	450	490000	Won
14	2046943	Salesforce	Emma Gordian	Claude Shannon	Analytics	750	7000000	Won
15	2169499	Cisco	Emma Gordian	Claude Shannon	Analytics	300	4550000	Lost
16	2169499	Cisco	Emma Gordian	Claude Shannon	GPS Positioning	300	350000	Presented

What type of question does a pivot help answer?

Imagine we wanted to know, how many licenses of each product type did Google purchase? Currently the way the data is formatted is hard to read. Let's pivot it so this is clearer, we will take a subset of the data for the question at hand.

In [22]:

```
# Let's take a subset, otherwise we'll get an error due to duplicate rows and data
Licenses = df[['Company','Product','Licenses']]
Licenses
```

Out[22]:

	Company	Product	Licenses
0	Google	Analytics	150
1	Google	Prediction	150
2	Google	Tracking	300
3	BOBO	Analytics	150
4	IKEA	Analytics	300
5	Tesla Inc.	Analytics	300
6	Tesla Inc.	Prediction	150
7	Microsoft	Tracking	300
8	Walmart	Analytics	150
9	Apple	Analytics	300
10	Exxon Mobile	Analytics	150
11	ATT	Tracking	150
12	ATT	Prediction	150
13	CVS Health	Tracking	450
14	Salesforce	Analytics	750
15	Cisco	Analytics	300
16	Cisco	GPS Positioning	300

In [11]:

```
pd.pivot(data =Licenses,index='Company',columns='Product',values='Licenses')
```

Out[11]:

Product	Analytics	GPS Positioning	Prediction	Tracking
Company				
Google	150.0	NaN	150.0	300.0
ATT	NaN	NaN	150.0	150.0
Apple	300.0	NaN	NaN	NaN
BOBO	150.0	NaN	NaN	NaN
CVS Health	NaN	NaN	NaN	450.0
Cisco	300.0	300.0	NaN	NaN
Exxon Mobile	150.0	NaN	NaN	NaN
IKEA	300.0	NaN	NaN	NaN
Microsoft	NaN	NaN	NaN	300.0
Salesforce	750.0	NaN	NaN	NaN
Tesla Inc.	300.0	NaN	150.0	NaN
Walmart	150.0	NaN	NaN	NaN

The pivot_table() method

Similar to the pivot() method, the pivot_table() can add aggregation functions to a pivot call.

In [24]:

```
df
```

Out[24]:

	Account Number	Company	Contact	Account Manager	Product	Licenses	Sale Price	Status
0	2123398	Google	Larry Pager	Edward Thorp	Analytics	150	2100000	Presented
1	2123398	Google	Larry Pager	Edward Thorp	Prediction	150	700000	Presented
2	2123398	Google	Larry Pager	Edward Thorp	Tracking	300	350000	Under Review
3	2192650	BOBO	Larry Pager	Edward Thorp	Analytics	150	2450000	Lost
4	420496	IKEA	Elon Tusk	Edward Thorp	Analytics	300	4550000	Won
5	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Analytics	300	2800000	Under Review
6	636685	Tesla Inc.	Elon Tusk	Edward Thorp	Prediction	150	700000	Presented
7	1216870	Microsoft	Will Grates	Edward Thorp	Tracking	300	350000	Under Review
8	2200450	Walmart	Will Grates	Edward Thorp	Analytics	150	2450000	Lost
9	405886	Apple	Cindy Phoner	Claude Shannon	Analytics	300	4550000	Won
10	470248	Exxon Mobile	Cindy Phoner	Claude Shannon	Analytics	150	2100000	Presented
11	698032	ATT	Cindy Phoner	Claude Shannon	Tracking	150	350000	Under Review
12	698032	ATT	Cindy Phoner	Claude Shannon	Prediction	150	700000	Presented
13	902797	CVS Health	Emma Gordian	Claude Shannon	Tracking	450	490000	Won
14	2046943	Salesforce	Emma Gordian	Claude Shannon	Analytics	750	7000000	Won
15	2169499	Cisco	Emma Gordian	Claude Shannon	Analytics	300	4550000	Lost
16	2169499	Cisco	Emma Gordian	Claude Shannon	GPS Positioning	300	350000	Presented

In [23]:

```
# Notice Account Number sum() doesn't make sense to keep/use
pd.pivot_table(data=df,index = 'Company', aggfunc='sum')
```

Out[23]:

	Account Number	Licenses	Sale Price
Company			
Google	6370194	600	3150000
ATT	1396064	300	1050000
Apple	405886	300	4550000
BOBO	2192650	150	2450000
CVS Health	902797	450	490000
Cisco	4338998	600	4900000
Exxon Mobile	470248	150	2100000
IKEA	420496	300	4550000
Microsoft	1216870	300	350000
Salesforce	2046943	750	7000000
Tesla Inc.	1273370	450	3500000
Walmart	2200450	150	2450000

In [26]:

```
# Same thing we can do by groupby method
df.groupby('Company').sum()
```

Out[26]:

	Account Number	Licenses	Sale Price

	Account Number	Licenses	Sale Price
Company			
Google	6370194	600	3150000
ATT	1396064	300	1050000
Apple	405886	300	4550000
BOBO	2192650	150	2450000
CVS Health	902797	450	490000
Cisco	4338998	600	4900000
Exxon Mobile	470248	150	2100000
IKEA	420496	300	4550000
Microsoft	1216870	300	350000
Salesforce	2046943	750	7000000
Tesla Inc.	1273370	450	3500000
Walmart	2200450	150	2450000

In [30]:

```
# We have seen that it also sum account number that doesnt mean anything, here we remove
account number column
pd.pivot_table(data=df,index = 'Company', aggfunc='sum', values=['Licenses','Sale Price'
])
```

Out[30]:

	Licenses	Sale Price
Company		
Google	600	3150000
ATT	300	1050000
Apple	300	4550000
BOBO	150	2450000
CVS Health	450	490000
Cisco	600	4900000
Exxon Mobile	150	2100000
IKEA	300	4550000
Microsoft	300	350000
Salesforce	750	7000000
Tesla Inc.	450	3500000
Walmart	150	2450000

In [33]:

```
# Same thing by groupby function
df.groupby('Company').sum()[['Licenses','Sale Price']]
```

Out[33]:

	Licenses	Sale Price
Company		
Google	600	3150000
ATT	300	1050000
Apple	300	4550000
BOBO	150	2450000
CVS Health	450	490000

	Licenses	Sale Price
Company		
Cisco	600	4900000
Exxon Mobile	150	2100000
IKEA	300	4550000
Microsoft	300	350000
Salesforce	750	7000000
Tesla Inc.	450	3500000
Walmart	150	2450000

In [44]:

```
# Here outside index is Account Manger and Inner index is Contact , we count on sum of sales
pd.pivot_table(data= df, index =['Account Manager','Contact'] ,values=['Sale Price'], aggfunc='sum')
```

Out[44]:

		Sale Price
Account Manager	**Contact**	
Claude Shannon	**Cindy Phoner**	7700000
	Emma Gordian	12390000
Edward Thorp	**Elon Tusk**	8050000
	Larry Pager	5600000
	Will Grates	2800000

Columns are optional - they provide an additional way to segment the actual values you care about. The aggregation functions are applied to the values you list.

In [43]:

```
#If we want product wise we can add columns function
pd.pivot_table(data= df, index =['Account Manager','Contact'] ,values=['Sale Price'],columns=['Product'] ,aggfunc='sum')
```

Out[43]:

		Sale Price			
	Product	**Analytics**	**GPS Positioning**	**Prediction**	**Tracking**
Account Manager	**Contact**				
Claude Shannon	**Cindy Phoner**	6650000.0	NaN	700000.0	350000.0
	Emma Gordian	11550000.0	350000.0	NaN	490000.0
Edward Thorp	**Elon Tusk**	7350000.0	NaN	700000.0	NaN
	Larry Pager	4550000.0	NaN	700000.0	350000.0
	Will Grates	2450000.0	NaN	NaN	350000.0

In [55]:

```
# If we wany both sum and mean we can do that , here focus on aggfunction we added np.mean and np.sum in square brackets
pd.pivot_table(data= df, index =['Account Manager','Contact'] ,values=['Sale Price']
               ,columns=['Product'] ,aggfunc=[np.sum,np.mean])
```

Out[55]:

		sum	mean
		Sale Price	Sale Price

	Product	sum Analytics Sale Price	GPS Positioning	Prediction	Tracking	mean Analytics Sale Price	GPS Positioning	Prediction	Tracking
Account Manager	Product Contact	Analytics	GPS Positioning	Prediction	Tracking	Analytics	GPS Positioning	Prediction	Tracking
Claude Shannon Account Manager	Cindy Phoner Contact	6650000.0	NaN	700000.0	350000.0	3325000.0	NaN	700000.0	350000.0
Edward Thorp	Emma Gordian	11550000.0	350000.0	NaN	490000.0	5775000.0	350000.0	NaN	490000.0
	Elon Tusk	7350000.0	NaN	700000.0	NaN	3675000.0	NaN	700000.0	NaN
	Larry Pager	4550000.0	NaN	700000.0	350000.0	2275000.0	NaN	700000.0	350000.0
	Will Grates	2450000.0	NaN	NaN	350000.0	2450000.0	NaN	NaN	350000.0

In [56]:

```
# we change product from column to row as well
pd.pivot_table(df,index=['Account Manager','Contact','Product'],aggfunc=[np.sum],values=
['Sale Price'])
```

Out[56]:

			sum
			Sale Price
Account Manager	Contact	Product	
Claude Shannon	Cindy Phoner	Analytics	6650000
		Prediction	700000
		Tracking	350000
Edward Thorp	Emma Gordian	Analytics	11550000
		GPS Positioning	350000
		Tracking	490000
	Elon Tusk	Analytics	7350000
		Prediction	700000
	Larry Pager	Analytics	4550000
		Prediction	700000
		Tracking	350000
	Will Grates	Analytics	2450000
		Tracking	350000

In [57]:

```
# we can add grand total by adding margins
pd.pivot_table(data= df, index =['Account Manager','Contact']
               ,values=['Sale Price'],columns=['Product'],margins=True ,aggfunc=[np.sum,
np.mean])
```

Out[57]:

		sum					mean			
		Sale Price					Sale Price			
	Product	Analytics	GPS Positioning	Prediction	Tracking	All	Analytics	GPS Positioning	Prediction	Tracking
Account Manager	Contact									
Claude Shannon	Cindy Phoner	6650000.0	NaN	700000.0	350000.0	7700000	3.325000e+06	NaN	700000.0	350000.0

		sum					mean			
		Sale Price					Sale Price			
Account Manager	Contact Product	Analytics	GPS Positioning	Prediction	Tracking	All	Analytics	GPS Positioning	Prediction	Tracking
	Emma Gordian	11550000.0	350000.0	NaN	490000.0	12390000	5.775000e+06	350000.0	NaN	490000.0
Edward Thorp	Elon Tusk	7350000.0	NaN	700000.0	NaN	8050000	3.675000e+06	NaN	700000.0	NaN
	Larry Pager	4550000.0	NaN	700000.0	350000.0	5600000	2.275000e+06	NaN	700000.0	350000.0
	Will Grates	2450000.0	NaN	NaN	350000.0	2800000	2.450000e+06	NaN	NaN	350000.0
All		32550000.0	350000.0	2100000.0	1540000.0	36540000	3.616667e+06	350000.0	700000.0	385000.0

In [58]:

```
pd.pivot_table(df,index=['Account Manager','Contact','Product'],aggfunc=[np.sum],values=
['Sale Price'],margins=True)
```

Out[58]:

			sum
			Sale Price
Account Manager	Contact	Product	
Claude Shannon	Cindy Phoner	Analytics	6650000
		Prediction	700000
		Tracking	350000
	Emma Gordian	Analytics	11550000
		GPS Positioning	350000
		Tracking	490000
Edward Thorp	Elon Tusk	Analytics	7350000
		Prediction	700000
	Larry Pager	Analytics	4550000
		Prediction	700000
		Tracking	350000
	Will Grates	Analytics	2450000
		Tracking	350000
All			36540000

In [ ]: