Pandas - 4

Content

- **Multi-indexing**
- **Restructuring data**
  - pd.melt()
  - pd.pivot()
  - pd.pivot_table()
  - pd.cut()

Importing Data

```
import pandas as pd
import numpy as np

!gdown 1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
!gdown 1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm

movies = pd.read_csv('movies.csv', index_col=0)
directors = pd.read_csv('directors.csv',index_col=0)

data = movies.merge(directors, how='left', left_on='director_id',right_on='id')
data.drop(['director_id','id_y'],axis=1,inplace=True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
To: /content/movies.csv
100% 112k/112k [00:00<00:00, 15.4MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
To: /content/directors.csv
100% 65.4k/65.4k [00:00<00:00, 86.7MB/s]
```

Multi-Indexing

Task: Which director according would be considered as most productive ?

- Should we decide based on the **number of movies** released by a director?

Or

- consider **quality into consideration also?**

Or

- consider the amount of business the movie is doing?

To simplify,

Lets calculate who has directed maximum number of movies

```
data.groupby(['director_name'])['title'].count().sort_values(ascending=False)
```

```
director_name
Steven Spielberg    26
Clint Eastwood      19
Martin Scorsese     19
Woody Allen         18
Robert Rodriguez    16
                    ..
Paul Weitz           5
John Madden          5
Paul Verhoeven       5
John Whitesell       5
Kevin Reynolds       5
Name: title, Length: 199, dtype: int64
```

`Steven Spielberg` has directed maximum number of movies. **But does it make `Steven` the most productive director?**

Chances are, he might be **active for more years** than other directors

calculating active years for **every director**?

We can subtract both `min` and `max` of `year`

```
data_agg = data.groupby(['director_name'])[["year", "title"]].aggregate({"year":['min','max'], "title": "count"})
data_agg
```

	year		title
	min	max	count
director_name			
Adam McKay	2004	2015	6
Adam Shankman	2001	2012	8
Alejandro González Iñárritu	2000	2015	6
Alex Proyas	1994	2016	5
Alexander Payne	1999	2013	5
...	...	...	...
Wes Craven	1984	2011	10
Wolfgang Petersen	1981	2006	7
Woody Allen	1977	2013	18
Zack Snyder	2004	2016	7
Zhang Yimou	2002	2014	6

199 rows × 3 columns

Notice,

- `director_name` column has turned into **row labels**
- There are multiple levels for the column names

This is called **Multi-index Dataframe**

What is Multi-index Dataframe ?

- It can have **multiple indexes along a dimension**
  - no of dimensions remain same though => 2D
- Multi-level indexes are **possible both for rows and columns**

```
data_agg.columns #Printing the columns for better clarity
```

```
MultiIndex([( 'year',   'min'),
            ( 'year',   'max'),
            ('title', 'count')],
           )
```

> The level-1 column names are `year` and `title`

What would happen if we print the col `year` of this multi-index dataframe?

```
data_agg["year"]
```

director_name	min	max
Adam McKay	2004	2015
Adam Shankman	2001	2012
Alejandro González Iñárritu	2000	2015
Alex Proyas	1994	2016
Alexander Payne	1999	2013
...	...	...
Wes Craven	1984	2011
Wolfgang Petersen	1981	2006
Woody Allen	1977	2013
Zack Snyder	2004	2016
Zhang Yimou	2002	2014

199 rows × 2 columns

- How can we convert multi-level back to only one level of columns?

Example: `year_min`, `year_max`, `title_count`

```
data_agg = data.groupby(['director_name'])[["year","title"]].aggregate(
    {"year":['min', 'max'], "title": "count"}) #The column names are not aligned properly
```

```
data_agg.columns = ['_'.join(col) for col in data_agg.columns]
data_agg
```

director_name	year_min	year_max	title_count
Adam McKay	2004	2015	6
Adam Shankman	2001	2012	8
Alejandro González Iñárritu	2000	2015	6
Alex Proyas	1994	2016	5
Alexander Payne	1999	2013	5
...	...	...	...
Wes Craven	1984	2011	10
Wolfgang Petersen	1981	2006	7
Woody Allen	1977	2013	18
Zack Snyder	2004	2016	7
Zhang Yimou	2002	2014	6

199 rows × 3 columns

Since these were tuples, we can just join them

```
# another more simplified method

data.groupby('director_name')[['year', 'title']].aggregate(
    year_max=('year','max'),
    year_min=('year','min'),
    title_count=('title','count')
)
```

director_name	year_max	year_min	title_count
Adam McKay	2015	2004	6
Adam Shankman	2012	2001	8
Alejandro González Iñárritu	2015	2000	6
Alex Proyas	2016	1994	5
Alexander Payne	2013	1999	5
...	...	...	...
Wes Craven	2011	1984	10
Wolfgang Petersen	2006	1981	7
Woody Allen	2013	1977	18
Zack Snyder	2016	2004	7
Zhang Yimou	2014	2002	6

199 rows × 3 columns

Columns look good, but we may want to turn back the row labels into a proper column as well

- converting row labels into a column using `reset_index`

```
data_agg.reset_index()
```

	director_name	year_min	year_max	title_count
0	Adam McKay	2004	2015	6
1	Adam Shankman	2001	2012	8
2	Alejandro González Iñárritu	2000	2015	6
3	Alex Proyas	1994	2016	5
4	Alexander Payne	1999	2013	5
...	...	...	...	...
194	Wes Craven	1984	2011	10
195	Wolfgang Petersen	1981	2006	7
196	Woody Allen	1977	2013	18
197	Zack Snyder	2004	2016	7
198	Zhang Yimou	2002	2014	6

199 rows × 4 columns

- Using the new features, can we find the most productive director?

First calculate how many years the director has been active.

```
data_agg["yrs_active"] = data_agg["year_max"] - data_agg["year_min"]
data_agg
```

director_name	year_min	year_max	title_count	yrs_active
Adam McKay	2004	2015	6	11
Adam Shankman	2001	2012	8	11
Alejandro González Iñárritu	2000	2015	6	15
Alex Proyas	1994	2016	5	22
Alexander Payne	1999	2013	5	14
...	...	...	...	...
Wes Craven	1984	2011	10	27
Wolfgang Petersen	1981	2006	7	25
Woody Allen	1977	2013	18	36
Zack Snyder	2004	2016	7	12
Zhang Yimou	2002	2014	6	12

199 rows × 4 columns

Then calculate rate of directing movies by `title_count` / `yrs_active`

```
data_agg["movie_per_yr"] = data_agg["title_count"] / data_agg["yrs_active"]
data_agg
```

director_name	year_min	year_max	title_count	yrs_active	movie_per_yr
Adam McKay	2004	2015	6	11	0.545455
Adam Shankman	2001	2012	8	11	0.727273
Alejandro González Iñárritu	2000	2015	6	15	0.400000
Alex Proyas	1994	2016	5	22	0.227273
Alexander Payne	1999	2013	5	14	0.357143
...	...	...	...	...	...
Wes Craven	1984	2011	10	27	0.370370
Wolfgang Petersen	1981	2006	7	25	0.280000
Woody Allen	1977	2013	18	36	0.500000
Zack Snyder	2004	2016	7	12	0.583333
Zhang Yimou	2002	2014	6	12	0.500000

199 rows × 5 columns

Now finally sort the values

```
data_agg.sort_values("movie_per_yr", ascending=False)
```

director_name	year_min	year_max	title_count	yrs_active	movie_per_yr
Tyler Perry	2006	2013	9	7	1.285714
Jason Friedberg	2006	2010	5	4	1.250000
Shawn Levy	2002	2014	11	12	0.916667
Robert Rodriguez	1992	2014	16	22	0.727273
Adam Shankman	2001	2012	8	11	0.727273
...	...	...	...	...	...
Lawrence Kasdan	1985	2012	5	27	0.185185
Luc Besson	1985	2014	5	29	0.172414
Robert Redford	1980	2010	5	30	0.166667
Sidney Lumet	1976	2006	5	30	0.166667
Michael Apted	1980	2010	5	30	0.166667

199 rows × 5 columns

Conclusion:

> **Tyler Perry** turns out to be the **truly most productive director**

Importing our data

- For this topic we will be using **data of few drugs** being developed by **PFizer**

Link: https://drive.google.com/file/d/173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ/view?usp=sharing

```
!gdown 173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ
```

```
Downloading...
From: https://drive.google.com/uc?id=173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ
To: /content/Pfizer_1.csv
100% 1.51k/1.51k [00:00<00:00, 8.41MB/s]
```

What is the data about?

- Temperature (K)
- Pressure (P)

are recorded after an **interval of 1 hour** everyday to monitor the drug stability in a drug development test

==> These data points are thus used to **identify the optimal set of values of parameters** for the stability of the drugs

Now, Let's explore this dataset

```
data = pd.read_csv('Pfizer_1.csv')
data
```

	Date	Drug_Name	Parameter	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:30:00	7:30:00	8:30:00	9:30:00
0	15-10-2020	diltiazem hydrochloride	Temperature	23.0	22.0	NaN	21.0	21.0	22	23.0	21.0	22.0
1	15-10-2020	diltiazem hydrochloride	Pressure	12.0	13.0	NaN	11.0	13.0	14	16.0	16.0	24.0
2	15-10-2020	docetaxel injection	Temperature	NaN	17.0	18.0	NaN	17.0	18	NaN	NaN	23.0
3	15-10-2020	docetaxel injection	Pressure	NaN	22.0	22.0	NaN	22.0	23	NaN	NaN	27.0
4	15-10-2020	ketamine hydrochloride	Temperature	24.0	NaN	NaN	27.0	NaN	26	25.0	24.0	23.0
5	15-10-2020	ketamine hydrochloride	Pressure	8.0	NaN	NaN	7.0	NaN	9	10.0	11.0	10.0
6	16-10-2020	diltiazem hydrochloride	Temperature	34.0	35.0	36.0	36.0	37.0	38	37.0	38.0	39.0
7	16-10-2020	diltiazem hydrochloride	Pressure	18.0	19.0	20.0	21.0	22.0	23	24.0	25.0	25.0
8	16-10-2020	docetaxel injection	Temperature	46.0	47.0	NaN	48.0	48.0	49	50.0	52.0	55.0
9	16-10-2020	docetaxel injection	Pressure	23.0	24.0	NaN	25.0	26.0	27	28.0	29.0	28.0
10	16-10-2020	ketamine hydrochloride	Temperature	8.0	9.0	10.0	NaN	11.0	12	12.0	11.0	NaN
11	16-10-2020	ketamine hydrochloride	Pressure	12.0	12.0	13.0	NaN	15.0	15	15.0	15.0	NaN
12	17-10-2020	diltiazem hydrochloride	Temperature	20.0	19.0	19.0	18.0	17.0	16	15.0	NaN	13.0
13	17-10-2020	diltiazem hydrochloride	Pressure	3.0	4.0	4.0	4.0	6.0	8	9.0	NaN	9.0
14	17-10-2020	docetaxel injection	Temperature	12.0	13.0	14.0	15.0	16.0	17	18.0	19.0	20.0
15	17-10-2020	docetaxel injection	Pressure	20.0	22.0	22.0	22.0	22.0	23	25.0	26.0	27.0
16	17-10-2020	ketamine hydrochloride	Temperature	13.0	14.0	15.0	16.0	17.0	18	19.0	20.0	21.0
17	17-10-2020	ketamine hydrochloride	Pressure	8.0	9.0	10.0	11.0	11.0	12	12.0	11.0	12.0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 15 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       18 non-null     object 
 1   Drug_Name  18 non-null     object 
 2   Parameter  18 non-null     object 
 3   1:30:00    16 non-null     float64
 4   2:30:00    16 non-null     float64
 5   3:30:00    12 non-null     float64
 6   4:30:00    14 non-null     float64
 7   5:30:00    16 non-null     float64
 8   6:30:00    18 non-null     int64  
 9   7:30:00    16 non-null     float64
 10  8:30:00    14 non-null     float64
 11  9:30:00    16 non-null     float64
 12  10:30:00   18 non-null     int64  
 13  11:30:00   16 non-null     float64
 14  12:30:00   18 non-null     int64  
dtypes: float64(9), int64(3), object(3)
memory usage: 2.2+ KB
```

Melting in Pandas

As we saw earlier, the dataset has 18 rows and 15 columns

If you notice further, you'll see:

- The **columns are `1:30:00`, `2:30:00`, `3:30:00`, ... so on**
- `Temperature` and `Pressure` **of each date** is **in a separate row**

Can we restructure our data into a better format?

Maybe we can have a column for `time`, with `timestamps` as the column value

Where will the Temperature/Pressure values go?

We can similarly create one column containing the values of these parameters

==> **"Melt" timestamp columns into two columns** - timestamp and corresponding values

How can we restructure our data into having every row corresponding to a single reading?

```
pd.melt(data, id_vars=['Date', 'Parameter', 'Drug_Name'])
```

	Date	Parameter	Drug_Name	variable	value
0	15-10-2020	Temperature	diltiazem hydrochloride	1:30:00	23.0
1	15-10-2020	Pressure	diltiazem hydrochloride	1:30:00	12.0
2	15-10-2020	Temperature	docetaxel injection	1:30:00	NaN
3	15-10-2020	Pressure	docetaxel injection	1:30:00	NaN
4	15-10-2020	Temperature	ketamine hydrochloride	1:30:00	24.0
...	...	...	...	...	...
211	17-10-2020	Pressure	diltiazem hydrochloride	12:30:00	14.0
212	17-10-2020	Temperature	docetaxel injection	12:30:00	23.0
213	17-10-2020	Pressure	docetaxel injection	12:30:00	28.0
214	17-10-2020	Temperature	ketamine hydrochloride	12:30:00	24.0
215	17-10-2020	Pressure	ketamine hydrochloride	12:30:00	15.0

216 rows × 5 columns

This converts our data from `wide` to `long` format

Notice the `id_vars` are set of variables which remain unmelted

How does `pd.melt()` work?

- Pass in the **DataFrame**
- Pass in the **column names to not melt**

But we can provide better names to these new columns

How can we rename the columns "variable" and "value" as per our original dataframe?

```
data_melt = pd.melt(data,id_vars = ['Date', 'Drug_Name', 'Parameter'],
            var_name = "time",
            value_name = 'reading')
```

```
data_melt
```

	Date	Drug_Name	Parameter	time	reading
0	15-10-2020	diltiazem hydrochloride	Temperature	1:30:00	23.0
1	15-10-2020	diltiazem hydrochloride	Pressure	1:30:00	12.0
2	15-10-2020	docetaxel injection	Temperature	1:30:00	NaN
3	15-10-2020	docetaxel injection	Pressure	1:30:00	NaN
4	15-10-2020	ketamine hydrochloride	Temperature	1:30:00	24.0
...	...	...	...	...	...
211	17-10-2020	diltiazem hydrochloride	Pressure	12:30:00	14.0
212	17-10-2020	docetaxel injection	Temperature	12:30:00	23.0
213	17-10-2020	docetaxel injection	Pressure	12:30:00	28.0
214	17-10-2020	ketamine hydrochloride	Temperature	12:30:00	24.0
215	17-10-2020	ketamine hydrochloride	Pressure	12:30:00	15.0

216 rows × 5 columns

Conclusion

- The labels of the timestamp columns are conviniently **melted into a single column** - `time`
- It retained all values in column `reading`
- The labels of columns such as `1:30:00`, `2:30:00` have now become categories of the variable column
- The **values from columns we are melting** are stored in **value** column

Pivot

Now suppose we want to convert our data back to **wide format**

The reason could be to maintain the structure for storing or some other purpose.

Notice:

- The variables `Date`, `Drug_Name` and `Parameter` will remain same
- The column names will be extracted from the column `time`
- The values will be extracted from the column `readings`

How can we restructure our data back to the original wide format, before it was melted?

```
data_melt.pivot(index=['Date','Drug_Name','Parameter'],  # Column to use to make new frame's index
                columns = 'time',                 # Column to use to make new frame's columns
                values='reading')                  # Columns to use for populating new frame's values.
```

			time	10:30:00	11:30:00	12:30:00	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:30:
Date	Drug_Name	Parameter										
15-10-2020	diltiazem hydrochloride	Pressure		18.0	19.0	20.0	12.0	13.0	NaN	11.0	13.0	14
		Temperature		20.0	20.0	21.0	23.0	22.0	NaN	21.0	21.0	22
	docetaxel injection	Pressure		26.0	29.0	28.0	NaN	22.0	22.0	NaN	22.0	23
		Temperature		23.0	25.0	25.0	NaN	17.0	18.0	NaN	17.0	18
	ketamine hydrochloride	Pressure		9.0	9.0	11.0	8.0	NaN	NaN	7.0	NaN	9
		Temperature		22.0	21.0	20.0	24.0	NaN	NaN	27.0	NaN	26
16-10-2020	diltiazem hydrochloride	Pressure		24.0	NaN	27.0	18.0	19.0	20.0	21.0	22.0	23
		Temperature		40.0	NaN	42.0	34.0	35.0	36.0	36.0	37.0	38
	docetaxel injection	Pressure		28.0	29.0	30.0	23.0	24.0	NaN	25.0	26.0	27
		Temperature		56.0	57.0	58.0	46.0	47.0	NaN	48.0	48.0	49
	ketamine hydrochloride	Pressure		16.0	17.0	18.0	12.0	12.0	13.0	NaN	15.0	15
		Temperature		13.0	14.0	15.0	8.0	9.0	10.0	NaN	11.0	12
17-10-2020	diltiazem hydrochloride	Pressure		11.0	13.0	14.0	3.0	4.0	4.0	4.0	6.0	8
		Temperature		14.0	11.0	10.0	20.0	19.0	19.0	18.0	17.0	16
	docetaxel injection	Pressure		28.0	29.0	28.0	20.0	22.0	22.0	22.0	22.0	23
		Temperature		21.0	22.0	23.0	12.0	13.0	14.0	15.0	16.0	17
	ketamine hydrochloride	Pressure		13.0	14.0	15.0	8.0	9.0	10.0	11.0	11.0	12
		Temperature		22.0	23.0	24.0	13.0	14.0	15.0	16.0	17.0	18

Notice,

`pivot()` is the exact opposite of melt

We are getting **multiple indices** here, but we can get single index again using `reset_index`

```
data_melt.pivot(index=['Date','Drug_Name','Parameter'],
                columns = 'time',
                values='reading').reset_index()
```

time	Date	Drug_Name	Parameter	10:30:00	11:30:00	12:30:00	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:
0	15-10-2020	diltiazem hydrochloride	Pressure	18.0	19.0	20.0	12.0	13.0	NaN	11.0	13.0	
1	15-10-2020	diltiazem hydrochloride	Temperature	20.0	20.0	21.0	23.0	22.0	NaN	21.0	21.0	
2	15-10-2020	docetaxel injection	Pressure	26.0	29.0	28.0	NaN	22.0	22.0	NaN	22.0	
3	15-10-2020	docetaxel injection	Temperature	23.0	25.0	25.0	NaN	17.0	18.0	NaN	17.0	
4	15-10-2020	ketamine hydrochloride	Pressure	9.0	9.0	11.0	8.0	NaN	NaN	7.0	NaN	
5	15-10-2020	ketamine hydrochloride	Temperature	22.0	21.0	20.0	24.0	NaN	NaN	27.0	NaN	
6	16-10-2020	diltiazem hydrochloride	Pressure	24.0	NaN	27.0	18.0	19.0	20.0	21.0	22.0	
7	16-10-2020	diltiazem hydrochloride	Temperature	40.0	NaN	42.0	34.0	35.0	36.0	36.0	37.0	
8	16-10-2020	docetaxel injection	Pressure	28.0	29.0	30.0	23.0	24.0	NaN	25.0	26.0	
9	16-10-2020	docetaxel injection	Temperature	56.0	57.0	58.0	46.0	47.0	NaN	48.0	48.0	
10	16-10-2020	ketamine hydrochloride	Pressure	16.0	17.0	18.0	12.0	12.0	13.0	NaN	15.0	
11	16-10-2020	ketamine hydrochloride	Temperature	13.0	14.0	15.0	8.0	9.0	10.0	NaN	11.0	
12	17-10-2020	diltiazem hydrochloride	Pressure	11.0	13.0	14.0	3.0	4.0	4.0	4.0	6.0	
13	17-10-2020	diltiazem hydrochloride	Temperature	14.0	11.0	10.0	20.0	19.0	19.0	18.0	17.0	
14	17-10-2020	docetaxel injection	Pressure	28.0	29.0	28.0	20.0	22.0	22.0	22.0	22.0	
15	17-10-2020	docetaxel injection	Temperature	21.0	22.0	23.0	12.0	13.0	14.0	15.0	16.0	
16	17-10-2020	ketamine hydrochloride	Pressure	13.0	14.0	15.0	8.0	9.0	10.0	11.0	11.0	
17	17-10-2020	ketamine hydrochloride	Temperature	22.0	23.0	24.0	13.0	14.0	15.0	16.0	17.0	

```
data_melt.head()
```

	Date	Drug_Name	Parameter	time	reading
0	15-10-2020	diltiazem hydrochloride	Temperature	1:30:00	23.0
1	15-10-2020	diltiazem hydrochloride	Pressure	1:30:00	12.0
2	15-10-2020	docetaxel injection	Temperature	1:30:00	NaN
3	15-10-2020	docetaxel injection	Pressure	1:30:00	NaN
4	15-10-2020	ketamine hydrochloride	Temperature	1:30:00	24.0

Now if you notice,

We are **using 2 rows** to log readings for a single experiment.

Can we further restructure our data into dividing the Parameter column into T/P?

A format like:

```
Date | time | Drug_Name | Pressure | Temperature
```

would be really suitable

- We want to **split one single column into multiple columns**

How can we divide the Parameter column again?

+ Code + Text

```
data_tidy = data_melt.pivot(index=['Date','time', 'Drug_Name'],
                                    columns = 'Parameter',
                                    values='reading')

data_tidy
```

Date	time	Parameter / Drug_Name	Pressure	Temperature
15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0
		docetaxel injection	26.0	23.0
		ketamine hydrochloride	9.0	22.0
	11:30:00	diltiazem hydrochloride	19.0	20.0
		docetaxel injection	29.0	25.0

We can use `reset_index()` to remove the multi-index

```
data_tidy = data_tidy.reset_index()
data_tidy
```

	Parameter	Date	time	Drug_Name	Pressure	Temperature
0		15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0
1		15-10-2020	10:30:00	docetaxel injection	26.0	23.0
2		15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0
3		15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0
4		15-10-2020	11:30:00	docetaxel injection	29.0	25.0
...		...	...	...	...	...
103		17-10-2020	8:30:00	docetaxel injection	26.0	19.0
104		17-10-2020	8:30:00	ketamine hydrochloride	11.0	20.0
105		17-10-2020	9:30:00	diltiazem hydrochloride	9.0	13.0
106		17-10-2020	9:30:00	docetaxel injection	27.0	20.0
107		17-10-2020	9:30:00	ketamine hydrochloride	12.0	21.0

108 rows × 5 columns

We can rename our `index` column from `Parameter` to simply `None`

```
data_tidy.columns.name = None
```

```
data_tidy.head()
```

	Date	time	Drug_Name	Pressure	Temperature
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0

Pivot Table

Now suppose we want to find some insights, like **mean temperature day wise**

- Can we use pivot to find the day-wise mean value of temperature for each drug?

```
data_tidy.pivot(index=['Drug_Name'],
                columns = 'Date',
                values=['Temperature'])
```