Pandas 4

Content

- Multi-indexing
- Melting
  - `pd.melt()`
- Pivoting
  - `pd.pivot()`
  - `pd.pivot_table()`
- Binning
  - `pd.cut()`

Multi-Indexing

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

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='
data.drop(['director_id','id_y'],axis=1,inplace=True)
```

Which director according to you should be considered as most productive?

- Should we decide based on the **number of movies** directed?
- Or take the **quality of the movies** into consideration as well?
- Or maybe look at the the **amount of business** the movie is doing?

To simplify, let's calculate who has directed maximum number of movies.

In [2]:
```
data.groupby(['director_name'])['title'].count().sort_values(ascending=Fals
```

Out[2]:
```
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 the other directors.

Calculating the active years for every director?

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

In [3]:
```
data_agg = data.groupby(['director_name'])[["year", "title"]].aggregate({"ye
data_agg
```

Out[3]:

	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 a **Multi-index DataFrame**.

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

In [4]:
```
data_agg.columns
```

Out[4]:
```
MultiIndex([( 'year',   'min'),
            ( 'year',   'max'),
            ('title', 'count')],
           )
```

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

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

In [5]:
```
data_agg["year"]
```

Out[5]:

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?

- e.g. `year_min`, `year_max`, `title_count`

In [6]:
```
data_agg = data.groupby(['director_name'])[["year","title"]].aggregate(
    {"year":['min', 'max'], "title": "count"})
```

In [7]:
```
data_agg.columns = ['_'.join(col) for col in data_agg.columns]
data_agg
```

Out[7]:

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.

In [8]:
```
data.groupby('director_name')[['year', 'title']].aggregate(
    year_max=('year','max'),
    year_min=('year','min'),
    title_count=('title','count')
)
```

Out[8]:

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

The 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` -

In [9]:
```
data_agg.reset_index()
```

Out[9]:

	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?

1. First calculate how many years the director has been active.

In [10]:
```
data_agg["yrs_active"] = data_agg["year_max"] - data_agg["year_min"]
data_agg
```

Out[10]:

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

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

In [11]:
```
data_agg["movie_per_yr"] = data_agg["title_count"] / data_agg["yrs_active"]
data_agg
```

Out[11]:

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

1. Finally, sort the values.

In [12]:
```
data_agg.sort_values("movie_per_yr", ascending=False)
```

Out[12]:

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 truly the most productive director.

PFizer data

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

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

What is the data about?

- Temperature (K)
- Pressure (P)

The data is recorded after an **interval of 1 hour** everyday to monitor the drug stability in a drug development test.

These data points are therefore used to **identify the optimal set of values of parameters** for the stability of the drugs.

In [13]:
```
data = pd.read_csv('Pfizer_1.csv')
data
```

Out[13]:

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

	Date	Drug_Name	Parameter	1:30:00	2:30:00	3:30:00	4:30:00	5:30:00	6:30:00
17	17-10-	ketamine hydrochloride	Pressure	8.0	9.0	10.0	11.0	11.0	12

In [14]:
```
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

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" the timestamp column into two columns** - timestamp and corresponding values

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

In [15]:
```
pd.melt(data, id_vars=['Date', 'Parameter', 'Drug_Name'])
```

Out[15]:

	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 that the `id_vars` are set of variables which remain unmelted.

How does `pd.melt()` work?

- Pass in the **DataFrame**.
- Pass in the **column names that we don't want to 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?

In [16]:
```
data_melt = pd.melt(data,id_vars = ['Date', 'Drug_Name', 'Parameter'],
                    var_name = "time",
                    value_name = 'reading')
data_melt
```

Out[16]:

	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 the values in `reading` column.
- 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 the `value` column.

Pivoting

Now suppose we want to convert our data back to the **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?

In [17]:
```
data_melt.pivot(index=['Date','Drug_Name','Parameter'],  # Columns used to r
                columns = 'time',                        # Column used to ma
                values='reading')                        # Column used for p
```

Out[17]:

		time	10:30:00	11:30:00	12:30:00	1:30:00	2:30:00	3:30:0(
Date	Drug_Name	Parameter						
15-10-2020	diltiazem hydrochloride	Pressure	18.0	19.0	20.0	12.0	13.0	NaN
		Temperature	20.0	20.0	21.0	23.0	22.0	NaN
	docetaxel injection	Pressure	26.0	29.0	28.0	NaN	22.0	22.(
		Temperature	23.0	25.0	25.0	NaN	17.0	18.(
	ketamine hydrochloride	Pressure	9.0	9.0	11.0	8.0	NaN	NaN
		Temperature	22.0	21.0	20.0	24.0	NaN	NaN
16-10-2020	diltiazem hydrochloride	Pressure	24.0	NaN	27.0	18.0	19.0	20.(
		Temperature	40.0	NaN	42.0	34.0	35.0	36.(
	docetaxel injection	Pressure	28.0	29.0	30.0	23.0	24.0	NaN
		Temperature	56.0	57.0	58.0	46.0	47.0	NaN
	ketamine hydrochloride	Pressure	16.0	17.0	18.0	12.0	12.0	13.(
		Temperature	13.0	14.0	15.0	8.0	9.0	10.(
17-10-2020	diltiazem hydrochloride	Pressure	11.0	13.0	14.0	3.0	4.0	4.(
		Temperature	14.0	11.0	10.0	20.0	19.0	19.(
	docetaxel injection	Pressure	28.0	29.0	28.0	20.0	22.0	22.(
		Temperature	21.0	22.0	23.0	12.0	13.0	14.(
	ketamine hydrochloride	Pressure	13.0	14.0	15.0	8.0	9.0	10.(
		Temperature	22.0	23.0	24.0	13.0	14.0	15.(

Notice that `pivot()` is the exact opposite of `melt()`.

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

In [18]:
```
data_melt.pivot(index=['Date','Drug_Name','Parameter'],
                columns = 'time',
                values='reading').reset_index()
```

Out[18]:

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

time	Date	Drug_Name	Parameter	10:30:00	11:30:00	12:30:00	1:30:00	2:30:00	3:3
17	17-10-	ketamine hydrochloride	Temperature	22.0	23.0	24.0	13.0	14.0	

In [19]:
```
data_melt.head()
```

Out[19]:

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

How can we divide the `Parameter` column again?

In [20]:
```
data_tidy = data_melt.pivot(index=['Date','time', 'Drug_Name'],
                            columns = 'Parameter',
                            values='reading')
data_tidy
```

Out[20]:

			Parameter	Pressure	Temperature
Date	time	Drug_Name			
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
...	...	...		...	...
17-10-2020	8:30:00	docetaxel injection		26.0	19.0
		ketamine hydrochloride		11.0	20.0
	9:30:00	diltiazem hydrochloride		9.0	13.0
		docetaxel injection		27.0	20.0
		ketamine hydrochloride		12.0	21.0

108 rows × 2 columns

Notice that a **multi-index** dataframe has been created.

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

In [21]:
```
data_tidy = data_tidy.reset_index()
data_tidy
```

Out[21]:

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`.

In [22]:
```
data_tidy.columns.name = None
data_tidy.head()
```

Out[22]:

	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?

In [23]:
```
data_tidy.pivot(index=['Drug_Name'],
                columns = 'Date',
                values=['Temperature'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[23], line 1
----> 1 data_tidy.pivot(index=['Drug_Name'],
      2                 columns = 'Date',
      3                 values=['Temperature'])

File ~/anaconda3/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:8567, in DataFrame.pivot(self, index, columns, values)
   8561 @Substitution("")
   8562 @Appender(_shared_docs["pivot"])
   8563 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self"])
   8564 def pivot(self, index=None, columns=None, values=None) -> DataFrame:
   8565     from pandas.core.reshape.pivot import pivot
-> 8567     return pivot(self, index=index, columns=columns, values=values)

File ~/anaconda3/lib/python3.11/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/pivot.py:540, in pivot(data, index, columns, values)
    536         indexed = data._constructor_sliced(data[values]._values, index=multiindex)
    537 # error: Argument 1 to "unstack" of "DataFrame" has incompatible type "Union
    538 # [List[Any], ExtensionArray, ndarray[Any, Any], Index, Series]"; expected
    539 # "Hashable"
--> 540 return indexed.unstack(columns_listlike)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:9112, in DataFrame.unstack(self, level, fill_value)
   9050 """
   9051 Pivot a level of the (necessarily hierarchical) index labels.
   9052
   (...)
   9108 dtype: float64
   9109 """
   9110 from pandas.core.reshape.reshape import unstack
-> 9112 result = unstack(self, level, fill_value)
   9114 return result.__finalize__(self, method="unstack")

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/reshape.p
```

```
y:476, in unstack(obj, level, fill_value)
    474 if isinstance(obj, DataFrame):
    475     if isinstance(obj.index, MultiIndex):
--> 476         return _unstack_frame(obj, level, fill_value=fill_value)
    477     else:
    478         return obj.T.stack(dropna=False)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/reshape.p
y:499, in _unstack_frame(obj, level, fill_value)
    497 def _unstack_frame(obj: DataFrame, level, fill_value=None):
    498     assert isinstance(obj.index, MultiIndex)  # checked by caller
--> 499     unstacker = _Unstacker(obj.index, level=level, constructor=obj.
_constructor)
    501     if not obj._can_fast_transpose:
    502         mgr = obj._mgr.unstack(unstacker, fill_value=fill_value)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/reshape.p
y:137, in _Unstacker.__init__(self, index, level, constructor)
    129 if num_cells > np.iinfo(np.int32).max:
    130     warnings.warn(
    131         f"The following operation may generate {num_cells} cells "
    132         f"in the resulting pandas object.",
    133         PerformanceWarning,
    134         stacklevel=find_stack_level(),
    135     )
--> 137 self._make_selectors()

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/reshape.p
y:189, in _Unstacker._make_selectors(self)
    186 mask.put(selector, True)
    188 if mask.sum() < len(self.index):
--> 189     raise ValueError("Index contains duplicate entries, cannot resh
ape")
    191 self.group_index = comp_index
    192 self.mask = mask

ValueError: Index contains duplicate entries, cannot reshape
```

Why did we get an error?

- We need to find the **average** of temperature values throughout a day.
- If you notice, the error shows **duplicate entries**.

Hence, the index values should be unique entry for each row.

What can we do to get our required mean values then?

In [24]:
```
pd.pivot_table(data_tidy, index='Drug_Name', columns='Date', values=['Tempe
```

Out[24]:

	Temperature		
Date	15-10-2020	16-10-2020	17-10-2020
Drug_Name			
diltiazem hydrochloride	21.454545	37.454545	15.636364
docetaxel injection	20.750000	51.454545	17.500000
ketamine hydrochloride	23.555556	11.500000	18.500000

This function is similar to `pivot()`, with an extra feature of an aggregator.

How does `pivot_table()` work?

- The initial parameters are same as what we use in `pivot()`.
- As an extra parameter, we pass the **type of aggregator**.

Note:

- We could have done this using `groupby` too.
- In fact, `pivot_table` uses `groupby` in the backend to group the data and perform the aggregration.
- The only difference is in the type of output we get using both the functions.

Similarly, what if we want to find the minimum values of temperature and pressure on a particular date?

In [25]:
```
pd.pivot_table(data_tidy, index='Drug_Name', columns='Date', values=['Tempe
```

Out[25]:

	Pressure			Temperature		
Date	**15-10-2020**	**16-10-2020**	**17-10-2020**	**15-10-2020**	**16-10-2020**	**17-10-2020**
Drug_Name						
diltiazem hydrochloride	11.0	18.0	3.0	20.0	34.0	10.0
docetaxel injection	22.0	23.0	20.0	17.0	46.0	12.0
ketamine hydrochloride	7.0	12.0	8.0	20.0	8.0	13.0

Binning

Sometimes, we would want our data to be in **categorical** form instead of **continuous/numerical**.

- Let's say, instead of knowing specific test values of a month, I want to know its type.
- Depending on the level of granularity, we want to have - Low, Medium, High, Very High.

How can we derive bins/buckets from continous data?

- use `pd.cut()`

Let's try to use this on our `Temperature` column to categorise the data into bins.

But to define categories, let's first check `min` and `max` temperature values.

In [26]:
```
data_tidy
```

Out[26]:

	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

In [27]:
```
print(data_tidy['Temperature'].min(), data_tidy['Temperature'].max())
```

```
8.0 58.0
```

Here,

- Min value = 8
- Max value = 58

Lets's keep some buffer for future values and take the range from 5-60 (instead of 8-58).

We'll divide this data into **4 bins** of 10-15 values each.

In [28]:
```
temp_points = [5, 20, 35, 50, 60]

temp_labels = ['low','medium','high','very_high'] # labels define the severi
```

In [29]:
```
data_tidy['temp_cat'] = pd.cut(data_tidy['Temperature'], bins=temp_points,
data_tidy.head()
```

Out[29]:

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

In [30]:
```
data_tidy['temp_cat'].value_counts()
```

Out[30]:
```
low          45
medium       30
high         15
very_high     5
Name: temp_cat, dtype: int64
```

Note: By default, `pd.cut()` creates intervals of the form (x, y] — which includes the right endpoint but excludes the left one.

In [ ]: