Pandas - 5

Content

- Dealing with Missing Values
  - None and nan values
  - isna() and isnull()
- String method in pandas
- Handling datetime
- Writing to a file

Importing and preparing data

```
import pandas as pd
import numpy as np

!gdown 173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ

data = pd.read_csv('Pfizer_1.csv')
data_melt = pd.melt(data,id_vars = ['Date', 'Drug_Name', 'Parameter'],
            var_name = "time",
            value_name = 'reading')
data_tidy = data_melt.pivot(index=['Date','time', 'Drug_Name'],
                                      columns = 'Parameter',
                                      values='reading')
data_tidy = data_tidy.reset_index()
data_tidy.columns.name = None
```

```
Downloading...
From: https://drive.google.com/uc?id=173A59xh2mnpmljCCB9bhC4C5eP2IS6qZ
To: /content/Pfizer_1.csv
100% 1.51k/1.51k [00:00<00:00, 6.05MB/s]
```

```
data.head()
```

	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

```
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

```
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

Handling Missing Values

If you notice, there are many "NaN" values in our data

```
data.head()
```

	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

What are these "NaN" values?

They are basically **missing values**

What are missing values?

A Missing Value signifies an **empty cell/no data**

There can be 2 kinds of missing values:

1. `None`
2. `NaN` (short for Not a Number)

Whats the difference between the "None" and "NaN"?

The diff mainly lies in their datatype

```
type(None)
```

```
NoneType
```

```
type(np.nan)
```

```
float
```

None type is for missing values in a column with **non-number entries**

- E.g.-strings

NaN occurs for columns with **number entries**

Note:

Pandas uses these values nearly **interchangeably**, converting between them where appropriate, based on column datatype

```
pd.Series([1, np.nan, 2, None])
```

```
0    1.0
1    NaN
2    2.0
3    NaN
dtype: float64
```

For **numerical** types, Pandas changes **None to NaN** type

```
pd.Series(["1", "np.nan", "2", None])
```

```
0         1
1    np.nan
2         2
3      None
dtype: object
```

```
pd.Series(["1", "np.nan", "2", np.nan])
```

```
0         1
1    np.nan
```

```
2        2
3      NaN
dtype: object
```

For **object** type, the **None is preserved** and not changed to NaN

Now we have the basic idea about missing values

How to know the count of missing values for each row/column?

```
data.isna().head()
```

	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	10:30:00	11
0	False	False	False	False	False	True	False	False	False	False	False	False	False	
1	False	False	False	False	False	True	False	False	False	False	False	False	False	
2	False	False	False	True	False	False	True	False	False	True	True	False	False	
3	False	False	False	True	False	False	True	False	False	True	True	False	False	
4	False	False	False	False	True	True	False	True	False	False	False	False	False	

We can also use isnull to get the same results

```
data.isnull().head()
```

	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	10:30:00	11
0	False	False	False	False	False	True	False	False	False	False	False	False	False	
1	False	False	False	False	False	True	False	False	False	False	False	False	False	
2	False	False	False	True	False	False	True	False	False	True	True	False	False	
3	False	False	False	True	False	False	True	False	False	True	True	False	False	
4	False	False	False	False	True	True	False	True	False	False	False	False	False	

But, why do we have two methods, "isna" and "isnull" for the same operation?

isnull() is just an alias for isna()

```
pd.isnull
```

```
<function pandas.core.dtypes.missing.isna(obj: 'object') -> 'bool | npt.NDArray[np.bool_] | NDFrame'>
```

```
pd.isna
```

```
<function pandas.core.dtypes.missing.isna(obj: 'object') -> 'bool | npt.NDArray[np.bool_] | NDFrame'>
```

As we can see, function signature is same for both

`isna()` returns a **boolean dataframe**, with each cell as a boolean value

This value corresponds to **whether the cell has a missing value**

On top of this, we can use `.sum()` to find the count

```
data.isna().sum()
```

```
Date         0
Drug_Name    0
Parameter    0
1:30:00      2
2:30:00      2
3:30:00      6
4:30:00      4
5:30:00      2
6:30:00      0
7:30:00      2
8:30:00      4
9:30:00      2
10:30:00     0
11:30:00     2
```

```
12:30:00     0
dtype: int64
```

This gives us the total number of missing values in each column

Can we also get the number of missing values in each row?

```
data.isna().sum(axis=1)
```

```
0     1
1     1
2     4
3     4
4     3
5     3
6     1
7     1
8     1
9     1
10    2
11    2
12    1
13    1
14    0
15    0
16    0
17    0
dtype: int64
```

Note:

By default the value is `axis=0` in sum()

We have identified the null count, but how do we deal with them?

We have two options:

- delete the rows/columns containing the null values
- fill the missing values with some data/estimate

Let's first look at deleting the rows

How can we drop rows containing null values?

```
data.dropna()
```

	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
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

Rows with **even a single missing value** have been deleted

What if we want to delete the columns having missing value?

```
data.dropna(axis=1)
```

	Date	Drug_Name	Parameter	6:30:00	10:30:00	12:30:00
0	15-10-2020	diltiazem hydrochloride	Temperature	22	20	21
1	15-10-2020	diltiazem hydrochloride	Pressure	14	18	20
2	15-10-2020	docetaxel injection	Temperature	18	23	25
3	15-10-2020	docetaxel injection	Pressure	23	26	28
4	15-10-2020	ketamine hydrochloride	Temperature	26	22	20
5	15-10-2020	ketamine hydrochloride	Pressure	9	9	11
6	16-10-2020	diltiazem hydrochloride	Temperature	38	40	42
7	16-10-2020	diltiazem hydrochloride	Pressure	23	24	27
8	16-10-2020	docetaxel injection	Temperature	49	56	58
9	16-10-2020	docetaxel injection	Pressure	27	28	30
10	16-10-2020	ketamine hydrochloride	Temperature	12	13	15
11	16-10-2020	ketamine hydrochloride	Pressure	15	16	18
12	17-10-2020	diltiazem hydrochloride	Temperature	16	14	10
13	17-10-2020	diltiazem hydrochloride	Pressure	8	11	14
14	17-10-2020	docetaxel injection	Temperature	17	21	23
15	17-10-2020	docetaxel injection	Pressure	23	28	28
16	17-10-2020	ketamine hydrochloride	Temperature	18	22	24
17	17-10-2020	ketamine hydrochloride	Pressure	12	13	15

=> Every column which had even a single missing value has been deleted

But what are the problems with deleting rows/columns?

One of the major problems:

- loss of data

Instead of dropping, it would be better to **fill the missing values with some data**

How can we fill the missing values with some data?

```
data.fillna(0).head()
```

	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	0.0	21.0	21.0	22	23.0	21.0	22.0
1	15-10-2020	diltiazem hydrochloride	Pressure	12.0	13.0	0.0	11.0	13.0	14	16.0	16.0	24.0
2	15-10-2020	docetaxel injection	Temperature	0.0	17.0	18.0	0.0	17.0	18	0.0	0.0	23.0
3	15-10-2020	docetaxel injection	Pressure	0.0	22.0	22.0	0.0	22.0	23	0.0	0.0	27.0
4	15-10-2020	ketamine hydrochloride	Temperature	24.0	0.0	0.0	27.0	0.0	26	25.0	24.0	23.0

What is fillna(0) doing?

It fills all missing values with 0

We can do the same on a particular column too

```
data['2:30:00'].fillna(0)
```

```
0     22.0
1     13.0
2     17.0
3     22.0
4      0.0
5      0.0
6     35.0
7     19.0
8     47.0
9     24.0
10     9.0
11    12.0
12    19.0
13     4.0
```

```
14    13.0
15    22.0
16    14.0
17     9.0
Name: 2:30:00, dtype: float64
```

What other values can we use to fill the missing values ?

We can use some **kind of estimator** too

- An estimator like **mean or median**

How would you calculate the mean of the column `2:30:00`?

```
data['2:30:00'].mean()
```

```
18.8125
```

Now let's fill the NaN values with the mean value of the column

```
data['2:30:00'].fillna(data['2:30:00'].mean())
```

```
0     22.0000
1     13.0000
2     17.0000
3     22.0000
4     18.8125
5     18.8125
6     35.0000
7     19.0000
8     47.0000
9     24.0000
10     9.0000
11    12.0000
12    19.0000
13     4.0000
14    13.0000
15    22.0000
16    14.0000
17     9.0000
Name: 2:30:00, dtype: float64
```

But this doesn't feel right. What could be wrong with this?

Can we use the mean of all compounds as average for our estimator?

- **Different drugs** have **different characteristics**
- We can't simply do an average and fill the null values

Then what could be a solution here?

We could fill the null values of **respective compounds with their respective means**

How can we form a column with mean temperature of respective compounds?

We can use `apply` that we learnt earlier

Let's first create a function to calculate the mean

```
def temp_mean(x):
  x['Temperature_avg'] = x['Temperature'].mean() # We will name the new col Temperature_avg
  return x
```

Now we can form a new column based on the average values of temperature for each drug

```
data_tidy=data_tidy.groupby(["Drug_Name"], group_keys=False).apply(temp_mean)
data_tidy
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0	30.387097
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0	30.387097
...	...	...	...	...	...	...
103	17-10-2020	8:30:00	docetaxel injection	26.0	19.0	30.387097
104	17-10-2020	8:30:00	ketamine hydrochloride	11.0	20.0	17.709677
105	17-10-2020	9:30:00	diltiazem hydrochloride	9.0	13.0	24.848485
106	17-10-2020	9:30:00	docetaxel injection	27.0	20.0	30.387097
107	17-10-2020	9:30:00	ketamine hydrochloride	12.0	21.0	17.709677

108 rows × 6 columns

Now we fill the null values in Temperature using this new column!

```
data_tidy['Temperature'].fillna(data_tidy["Temperature_avg"], inplace=True)
data_tidy
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0	30.387097
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0	30.387097
...	...	...	...	...	...	...
103	17-10-2020	8:30:00	docetaxel injection	26.0	19.0	30.387097
104	17-10-2020	8:30:00	ketamine hydrochloride	11.0	20.0	17.709677
105	17-10-2020	9:30:00	diltiazem hydrochloride	9.0	13.0	24.848485
106	17-10-2020	9:30:00	docetaxel injection	27.0	20.0	30.387097
107	17-10-2020	9:30:00	ketamine hydrochloride	12.0	21.0	17.709677

108 rows × 6 columns

```
data_tidy.isna().sum()
```

```
Date               0
time               0
Drug_Name          0
Pressure          13
Temperature        0
Temperature_avg    0
dtype: int64
```

Great!!

We have removed the null values of our Temperature column

Let's do the same for Pressure

```
def pr_mean(x):
  x['Pressure_avg'] = x['Pressure'].mean()
  return x
data_tidy=data_tidy.groupby(["Drug_Name"]).apply(pr_mean)
data_tidy['Pressure'].fillna(data_tidy["Pressure_avg"], inplace=True)
data_tidy
```

```
<ipython-input-27-df55c441df36>:4: FutureWarning: Not prepending group keys to the result index of transform-like apply.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  data_tidy=data_tidy.groupby(["Drug_Name"]).apply(pr_mean)
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0	30.387097	25.483871
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677	11.935484
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0	30.387097	25.483871
...	...	...	...	...	...	...	...
103	17-10-2020	8:30:00	docetaxel injection	26.0	19.0	30.387097	25.483871
104	17-10-2020	8:30:00	ketamine hydrochloride	11.0	20.0	17.709677	11.935484
105	17-10-2020	9:30:00	diltiazem hydrochloride	9.0	13.0	24.848485	15.424242
106	17-10-2020	9:30:00	docetaxel injection	27.0	20.0	30.387097	25.483871
107	17-10-2020	9:30:00	ketamine hydrochloride	12.0	21.0	17.709677	11.935484

108 rows × 7 columns

```
data_tidy.isna().sum()
```

```
Date              0
time              0
Drug_Name         0
Pressure          0
Temperature       0
Temperature_avg   0
Pressure_avg      0
dtype: int64
```

Pandas Cut

Sometimes, we would want our data to be in **categorical format instead of continous data**.

Lets say, instead of knowing specific test values of a month, I want to know its type. Depends on level of granularity we want to have - Low, Medium, High, V High

We could have defined more (or less) categories

> But how can bucketisation of continous data help?
>
> - Since, we can get the count of different categories
> - We can get a idea of the bin which category (range of values) most of the temperature values lie.

Let's try to use this on our max (temp) column to categorise the data into bins

But, to define categories, lets first check min and max temp values

```
data_tidy
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0	30.387097	25.483871
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677	11.935484
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0	30.387097	25.483871
...	...	...	...	...	...	...	...
103	17-10-2020	8:30:00	docetaxel injection	26.0	19.0	30.387097	25.483871
104	17-10-2020	8:30:00	ketamine hydrochloride	11.0	20.0	17.709677	11.935484
105	17-10-2020	9:30:00	diltiazem hydrochloride	9.0	13.0	24.848485	15.424242
106	17-10-2020	9:30:00	docetaxel injection	27.0	20.0	30.387097	25.483871
107	17-10-2020	9:30:00	ketamine hydrochloride	12.0	21.0	17.709677	11.935484

108 rows × 7 columns

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

```
8.0 58.0
```

Min value = 8, Max value is 58.

- Lets's keep some buffer for future values and take the range from 5-60(instead of 8-58)
- Lets divide this data into 4 bins of 10-15 values each

```
temp_points = [5, 20, 35, 50, 60]
temp_labels = ['low','medium','high','very_high'] # Here labels define the severity of the resultant output of the test
data_tidy['temp_cat'] = pd.cut(data_tidy['Temperature'], bins=temp_points, labels=temp_labels)
data_tidy.head()
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg	temp_cat
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242	low
1	15-10-2020	10:30:00	docetaxel injection	26.0	23.0	30.387097	25.483871	medium
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677	11.935484	medium
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242	low
4	15-10-2020	11:30:00	docetaxel injection	29.0	25.0	30.387097	25.483871	medium

```
data_tidy['temp_cat'].value_counts()
```

```
low          50
medium       38
high         15
very_high     5
Name: temp_cat, dtype: int64
```

String function and motivation for datetime

What kind of questions can we use string methods for?

Find rows which contains a particular string

Say,

How you can you filter rows containing "hydrochloride" in their drug name?

```
data_tidy.loc[data_tidy['Drug_Name'].str.contains('hydrochloride')].head()
```

	Date	time	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg	temp_cat
0	15-10-2020	10:30:00	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242	low
2	15-10-2020	10:30:00	ketamine hydrochloride	9.0	22.0	17.709677	11.935484	medium
3	15-10-2020	11:30:00	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242	low
5	15-10-2020	11:30:00	ketamine hydrochloride	9.0	21.0	17.709677	11.935484	medium
6	15-10-2020	12:30:00	diltiazem hydrochloride	20.0	21.0	24.848485	15.424242	medium

So in general, we will be using the following format:

```
> Series.str.function()
```

Series.str can be used to **access the values of the series as strings** and apply several methods to it.

Now suppose we want to form a new column based on the year of the experiments?

What can we do form a column containing the year?

```
data_tidy['Date'].str.split('-')
```

```
0      [15, 10, 2020]
1      [15, 10, 2020]
2      [15, 10, 2020]
3      [15, 10, 2020]
4      [15, 10, 2020]
           ...
103    [17, 10, 2020]
104    [17, 10, 2020]
105    [17, 10, 2020]
106    [17, 10, 2020]
107    [17, 10, 2020]
Name: Date, Length: 108, dtype: object
```

To extract the year we need to select the last element of each list

```
data_tidy['Date'].str.split('-').apply(lambda x:x[2])
```

```
0      2020
1      2020
2      2020
3      2020
4      2020
       ...
103    2020
104    2020
105    2020
106    2020
107    2020
Name: Date, Length: 108, dtype: object
```

But there are certain problems with this approach:

- The **dtype of the output is still an object**, we would prefer a number type
- The date format will always **not be in day-month-year**, it can vary

Thus, to work with such date-time type of data, we can use a special method of pandas

Datetime

How can we handle date-time data-types?

- We can do using the `to_datetime()` function of pandas
- It takes as input:
    - Array/Scalars with values having proper date/time format
    - `dayfirst` : Indicating if the day comes first in the date format used
    - `yearfirst` : Indicates if year comes first in the date format

Let's first merge our `Date` and `time` columns into a new timestamp column

```
data_tidy['timestamp'] = data_tidy['Date']+ " "+ data_tidy['time']
```

```
data_tidy.drop(['Date', 'time'], axis=1, inplace=True)
```

```
data_tidy.head()
```

	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg	temp_cat	timestamp
0	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242	low	15-10-2020 10:30:00
1	docetaxel injection	26.0	23.0	30.387097	25.483871	medium	15-10-2020 10:30:00
2	ketamine hydrochloride	9.0	22.0	17.709677	11.935484	medium	15-10-2020 10:30:00
3	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242	low	15-10-2020 11:30:00
4	docetaxel injection	29.0	25.0	30.387097	25.483871	medium	15-10-2020 11:30:00

Lets convert our `timestamp` col now

```
data_tidy['timestamp'] = pd.to_datetime(data_tidy['timestamp']) # will leave to explore how you can mention datetime format
```

```
data_tidy
```

	Drug_Name	Pressure	Temperature	Temperature_avg	Pressure_avg	temp_cat	timestamp
0	diltiazem hydrochloride	18.0	20.0	24.848485	15.424242	low	2020-10-15 10:30:00
1	docetaxel injection	26.0	23.0	30.387097	25.483871	medium	2020-10-15 10:30:00
2	ketamine hydrochloride	9.0	22.0	17.709677	11.935484	medium	2020-10-15 10:30:00
3	diltiazem hydrochloride	19.0	20.0	24.848485	15.424242	low	2020-10-15 11:30:00
4	docetaxel injection	29.0	25.0	30.387097	25.483871	medium	2020-10-15 11:30:00
...	...	...	...	...	...	...	...
103	docetaxel injection	26.0	19.0	30.387097	25.483871	low	2020-10-17 08:30:00
104	ketamine hydrochloride	11.0	20.0	17.709677	11.935484	low	2020-10-17 08:30:00
105	diltiazem hydrochloride	9.0	13.0	24.848485	15.424242	low	2020-10-17 09:30:00
106	docetaxel injection	27.0	20.0	30.387097	25.483871	low	2020-10-17 09:30:00
107	ketamine hydrochloride	12.0	21.0	17.709677	11.935484	medium	2020-10-17 09:30:00

108 rows × 7 columns

```
data_tidy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 108 entries, 0 to 107
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Drug_Name        108 non-null    object
 1   Pressure         108 non-null    float64
 2   Temperature      108 non-null    float64
 3   Temperature_avg  108 non-null    float64
 4   Pressure_avg     108 non-null    float64
 5   temp_cat         108 non-null    category
 6   timestamp        108 non-null    datetime64[ns]
dtypes: category(1), datetime64[ns](1), float64(4), object(1)
memory usage: 10.3+ KB
```

The **type of `timestamp` column** has been **changed to `datetime`** from `object`

Now, Let's look at a single timestamp using Pandas

- How can we **extract information** from a single **timestamp** using Pandas?

```
ts = data_tidy['timestamp'][0]
ts
```

```
Timestamp('2020-10-15 10:30:00')
```

Extracting individual information from date

```
ts.year, ts.month, ts.day, ts.month_name()
```

```
(2020, 10, 15, 'October')
```

... and so on

We can similarly extract minutes and seconds

This data parsing from string to date-time makes it easier to work with data

We can use this data from the columns as a whole using `.dt` object

```
data_tidy['timestamp'].dt
```

```
<pandas.core.indexes.accessors.DatetimeProperties object at 0x7c2e78c72b60>
```

- **`dt` gives properties of values in a column**
- From this **`DatetimeProperties` of column `'end'`**, we can **extract `year`**

```
data_tidy['timestamp'].dt.year
```

```
0      2020
1      2020
2      2020
3      2020
4      2020
       ...
103    2020
104    2020
105    2020
106    2020
107    2020
```