Pandas-02 Notes

Content

- **Concatenation**
  - pd.concat()
    - axis for concat
    - Inner join
    - Outer join
- **Merging dataframes**
  - Concat v/s Merge
  - Outer join, Left join, Inner join
- **Intoduction to IMDB dataset**
  - Reading two datasets
- **Merging the dataframes**
  - `unique()` and `nunique()`
  - `isin()`
  - Use Left Outer Join and `merge()`
- **Feature Exploration**
  - Create new features
- **Fetching data using pandas**
  - Quering from dataframe - Masking, Filtering, `&` and `|`
  - String Methods - `contains()`, `startswith()`, `isin()`
  - Grouping
    - Split, Apply, Combine
    - `groupby()`
    - Group based Aggregates
    - `reset_index()`
    - Group based Filtering
    - Group based Transformation
    - `apply()`

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

Concatenating DataFrames

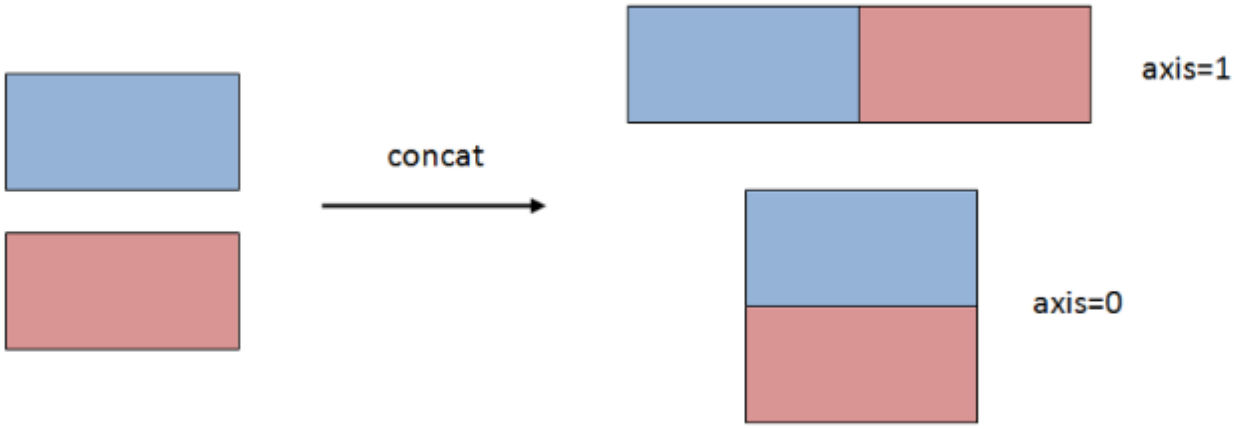
```
a = pd.DataFrame({'A':[10,30], 'B':[20,40]})
b = pd.DataFrame({'A':[10,30], 'C':[20,40]})
a
```

	A	B
0	10	20
1	30	40

```
b
```

	A	C
0	10	20
1	30	40

pd.concat()




```
pd.concat([a, b])
```

	A	B	C
0	10	20.0	NaN
1	30	40.0	NaN
0	10	NaN	20.0
1	30	NaN	40.0

Takeaways:

- By **default, `axis=0` for concatenation** (row-wise)
- Also the indices of the rows are preserved

```
pd.concat([a, b]).loc[0]
```

	A	B	C
0	10	20.0	NaN
0	10	NaN	20.0

▼ How can we get unique indices for each row ?

- By setting `ignore_index = True`

```
pd.concat([a, b], ignore_index = True)
```

	A	B	C
0	10	20.0	NaN
1	30	40.0	NaN
2	10	NaN	20.0
3	30	NaN	40.0

▼ What do we need to change to concatenate them column-wise?

- `axis=1`

```
pd.concat([a, b], axis=1)
```

	A	B	A	C
0	10	20	10	20
1	30	40	30	40

▼ Takeaway?

- It gives 2 columns with **different positional index**, but **same label**

We can also create a multi-indexed dataframe by mentioning the keys for each dataframe being concatenateed

```
pd.concat([a, b], keys=["x", "y"])
```

		A	B	C
x	**0**	10	20.0	NaN
	1	30	40.0	NaN
y	**0**	10	NaN	20.0
	1	30	NaN	40.0

- Which join can we use if we want a union of cols ?
  - Outer join
  - default for pd.concat

```
pd.concat([a, b], join="outer")
```

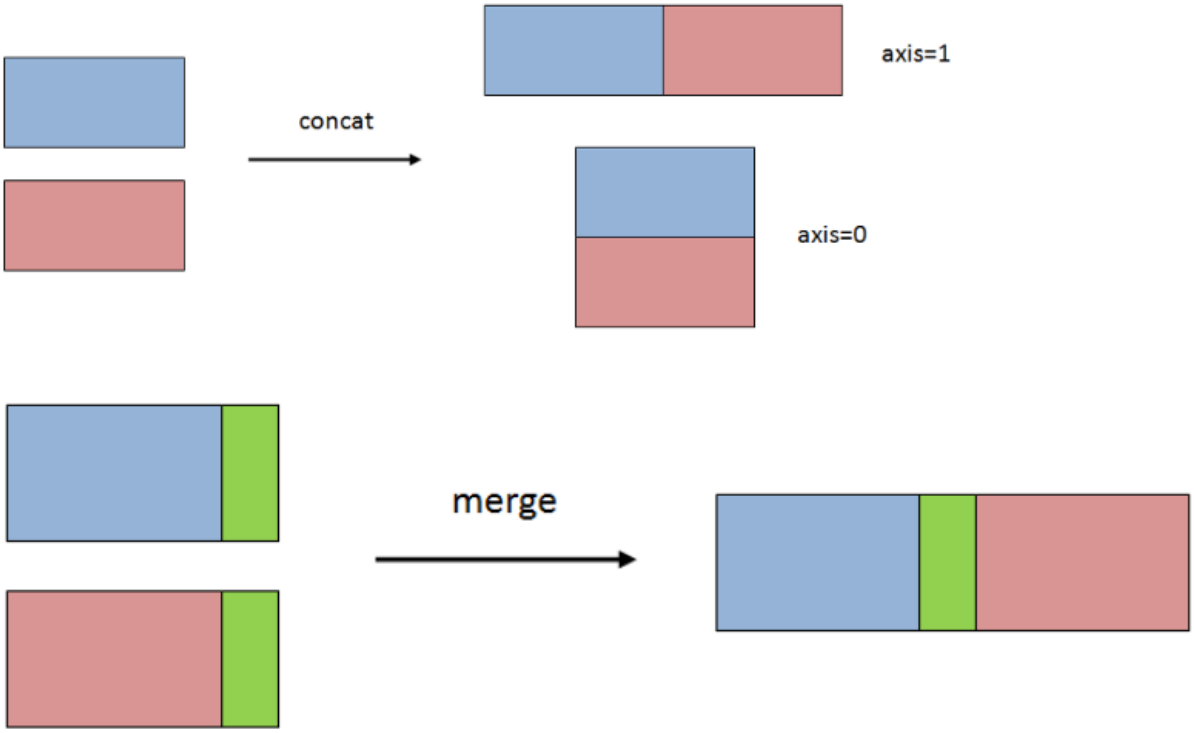
	A	B	C
0	10	20.0	NaN
1	30	40.0	NaN
0	10	NaN	20.0
1	30	NaN	40.0

- And what if we want an intersection of cols ?
  - inner join

```
pd.concat([a, b], join="inner")
```

	A
0	10
1	30
0	10
1	30

- whats the difference between concat and merge ?
  - `concat`
    - simply stacks multiple DataFrame together along an axis
  - `merge`
    - combines dataframes side-by-side based on values in shared columns



1. `users` --> **Stores the user details** - **IDs** and **Names of users**

```
users = pd.DataFrame({'userid':[1, 2, 3], 'name':['A', 'B', 'C']})
users
```

	userid	name
0	1	A
1	2	B
2	3	C

2. `msgs` --> **Stores the messages** users have sent - **User IDs** and **messages**

```
msgs = pd.DataFrame({'userid':[1, 1, 2], 'msg':['hello', 'bye', 'hi']})
msgs
```

	userid	msg
0	1	hello
1	1	bye
2	2	hi

- How can we get names of the person who have sent a message?

can we use pd.concat() for this ?

- pd.concat() does not work according to the values in the columns

Using pd.merge:

- Uses cols with same name as keys
- We can specify the cols to use as keys
- This is done through `on` parameter

pandas.merge

```
pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True,
indicator=False, validate=None)                                          [source]
```

Refer: https://pandas.pydata.org/docs/reference/api/pandas.merge.html

```
users.merge(msgs, on="userid")
```

	userid	name	msg
0	1	A	hello
1	1	A	bye
2	2	B	hi

But sometimes the column names might be different even if they contain the same data

How can we merge 2 dataframes in this situation ?

- Using the `left_on` and `right_on` keywords

```
users.rename(columns = {"userid": "id"}, inplace = True)
users.merge(msgs, left_on="id", right_on="userid") # this is inner join

# Notice that left_on is column from users
# right_on is column from msgs
```

	id	name	userid	msg
0	1	A	1	hello
1	1	A	1	bye
2	2	B	2	hi

Specifying type of joins to merge the dataframes

Lets say we want to find msg text of people only in the `users` table. Which join can we use for that ?

- `Inner join`
- It takes intersection of values in key cols
- Set by default in `pd.merge()`

```
users.merge(msgs, how = "inner", left_on = "id", right_on = "userid")
```

	id	name	userid	msg
0	1	A	1	hello
1	1	A	1	bye
2	2	B	2	hi

Now lets say we want a dataframe having all info of all the users.

- Using `outer` join
- It returns a join over the union of the input columns
- Replaces all missing values with `Na`

```
users.merge(msgs, how = "outer", left_on = "id", right_on = "userid")
```

	id	name	userid	msg
0	1	A	1.0	hello
1	1	A	1.0	bye
2	2	B	2.0	hi
3	3	C	NaN	NaN

And what if we want vals in key col of left dataframe ?

- We can use `left` join for that

```
users.merge(msgs, how = "left", left_on = "id", right_on = "userid")
```

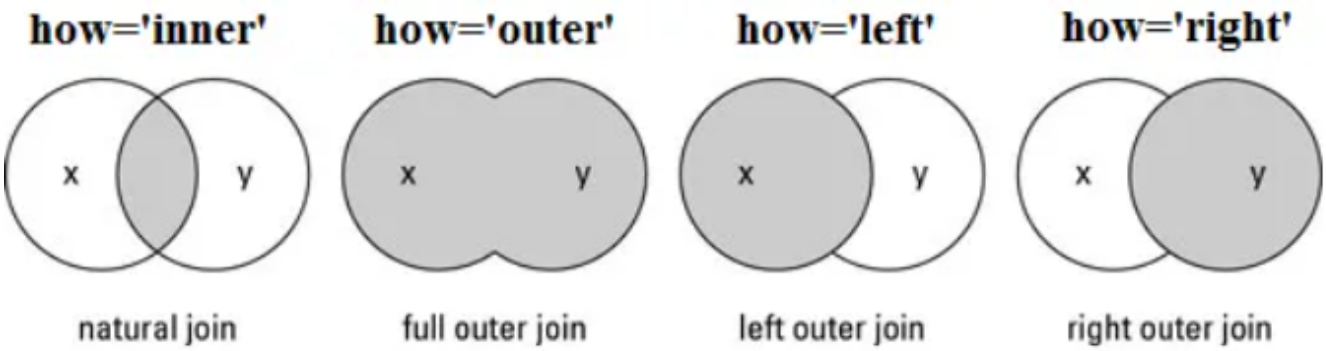
	id	name	userid	msg
0	1	A	1.0	hello
1	1	A	1.0	bye

Similarly, what if we want vals in key cols of only right dataframe ?

- Returns join over cols of right input

```
users.merge(msgs, how = "right", left_on = "id", right_on = "userid")
```

	id	name	userid	msg
0	1	A	1	hello
1	1	A	1	bye
2	2	B	2	hi



IMDB Movie Business Use-case

- The database contains info of several years about:
  - Movies
  - Rating
  - Director
  - Popularity
  - Revenue & Budget

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

```
!gdown 1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
```

```
Downloading...
From: https://drive.google.com/uc?id=1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd
```

```
To: /Users/anantm/Desktop/dsml-course/07-08-09-Pandas/movies.csv
100%|████████████████████████████████████████| 112k/112k [00:00<00:00, 1.86MB/s]
```

```
!gdown 1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
```

```
Downloading...
From: https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm
To: /Users/anantm/Desktop/dsml-course/07-08-09-Pandas/directors.csv
100%|████████████████████████████████████████| 65.4k/65.4k [00:00<00:00, 1.45MB/s]
```

Reading the dataset

```
movies = pd.read_csv('movies.csv')
#Top 5 rows
movies.head()
```

	Unnamed: 0	id	budget	popularity	revenue	title	vote_average	vote
0	0	43597	237000000	150	2787965087	Avatar	7.2	
1	1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	
2	2	43599	245000000	107	880674609	Spectre	6.3	
						The Dark		

what kind of questions can we ask from this dataset?

- **Top 10 most popular movies**
- **highest rated movies**
- **number of movies released per year**
- **find highest budget movies in a year**

Can we ask more interesting/deeper questions?

- Find the most productive director?
- Which director produces high budget films?
- Highest and lowest rated movies for every month in a particular year.

Notice, that we also get a column **Unnamed: 0** which represents nothing but the index of a row.

We can simply add one more argument `index_col=0` (treat first column as index) to get rid of this

The default value is `index_col=None`.

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

	id	budget	popularity	revenue	title	vote_average	vote_count	di
0	43597	237000000	150	2787965087	Avatar	7.2	11800	
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	
2	43599	245000000	107	880674609	Spectre	6.3	4466	

```
#Lets check the shape of dataset:
movies.shape
```

```
(1465, 11)
```

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

	director_name	id	gender
0	James Cameron	4762	Male
1	Gore Verbinski	4763	Male
2	Sam Mendes	4764	Male
3	Christopher Nolan	4765	Male
4	Andrew Stanton	4766	Male

```
directors.shape
```

```
(2349, 3)
```

Merging of both Dataframe:

So we want to include directors df info into movies df. How can we do this ?

- `merge()`

But, before merging, check if for all the movies in `movies` df, have their corresponding director details present in the `directors` df or not.

How do we get the number of unique directors in `movies` ?

We can do it using `nunique()`

- `unique()` gives **unique values** in a column
- `nunique()` gives **number of unique values** in a column

```
movies['director_id'].nunique()
```

```
199
```

```
directors['id'].nunique()
```

```
2349
```

- Movies Dataset: 1465 rows, but only 199 unique directors
- Directors Dataset: 2349 unique directors (= no of rows)

Inference?

- directors in `movies` is a subset of directors in `directors`
- need to check if we have details for 199 directors present in `directors` df also

How to check whether all the values in director_id column of movies is present in id column of director ?

- `isin()` method

Cant we do this using Python `in` ?

- We can, but this will work for one element at a time.
- We need to do this for all the values in the column
- The `isin()` method checks if the Dataframe column contains the specified value(s).

```
movies['director_id'].isin(directors['id'])
```

```
0       True
1       True
2       True
3       True
5       True
        ...
4736    True
4743    True
4748    True
4749    True
4768    True
Name: director_id, Length: 1465, dtype: bool
```

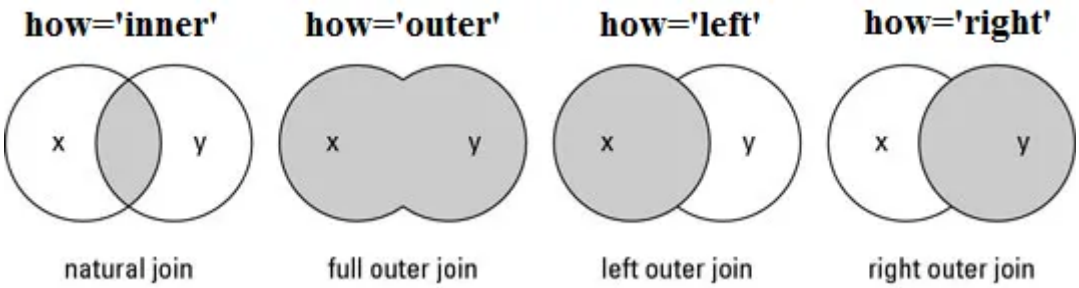
```
np.all(movies['director_id'].isin(directors['id']))
```

```
True
```

We can use `LEFT OUTER JOIN` to merge

Recall what will Left Outer Join do?

Left outer join will include all the rows of df `movies` and only those from `directors` that match with values of `movies` ['director_id']




```
# if column name is not same
# `left_on`: Specifies the key of the 1st dataframe
# `right_on`: Specifies the key of the 2nd dataframe
data = movies.merge(directors, how='left', left_on='director_id',right_on='id')
data
```

Since the columns with name `id` is present in both the df.

After merging:

- id_x : represents id values from movie df
- id_y: represents id values from directors df

Dropping redundant columns

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

	id_x	budget	popularity	revenue	title	vote_average	vote_count	ye
0	43597	237000000	150	2787965087	Avatar	7.2	11800	20
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	20
2	43599	245000000	107	880674609	Spectre	6.3	4466	20

1465 rows × 14 columns

Feature Exploration

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1465 entries, 0 to 1464
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id_x           1465 non-null   int64
 1   budget         1465 non-null   int64
 2   popularity     1465 non-null   int64
 3   revenue        1465 non-null   int64
 4   title          1465 non-null   object
 5   vote_average   1465 non-null   float64
 6   vote_count     1465 non-null   int64
 7   year           1465 non-null   int64
 8   month          1465 non-null   object
 9   day            1465 non-null   object
 10  director_name  1465 non-null   object
 11  gender         1341 non-null   object
dtypes: float64(1), int64(6), object(5)
memory usage: 148.8+ KB
```

```
data.describe()
```

	id_x	budget	popularity	revenue	vote_average	vote_cou
count	1465.000000	1.465000e+03	1465.000000	1.465000e+03	1465.000000	1465.000(
mean	45225.191126	4.802295e+07	30.855973	1.432539e+08	6.368191	1146.396(
std	1189.096396	4.935541e+07	34.845214	2.064918e+08	0.818033	1578.077(
min	43597.000000	0.000000e+00	0.000000	0.000000e+00	3.000000	1.000(
25%	44236.000000	1.400000e+07	11.000000	1.738013e+07	5.900000	216.000(
50%	45022.000000	3.300000e+07	23.000000	7.578164e+07	6.400000	571.000(
75%	45990.000000	6.600000e+07	41.000000	1.792469e+08	6.900000	1387.000(
max	48395.000000	3.800000e+08	724.000000	2.787965e+09	8.300000	13752.000(

```
data.describe(include=object)
```

	title	month	day	director_name	gender
count	1465	1465	1465	1465	1341
unique	1465	12	7	199	2
top	Zathura: A Space Adventure	Dec	Friday	Steven Spielberg	Male
freq	1	193	654	26	1309

- the range of values in the `revenue` and `budget` seem to be very high. So, it will be better to change the values into `million dollars USD`

▾ How will you change the values of `revenue` and `budget` into `million dollars USD?

```
data['revenue'] = (data['revenue']/1000000).round(2)
data
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	ye
0	43597	237000000	150	2787.97	Avatar	7.2	11800	20
1	43598	300000000	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	20
2	43599	245000000	107	880.67	Spectre	6.3	4466	20
3	43600	250000000	112	1084.94	The Dark Knight	7.6	9106	20

```
data['budget']=(data['budget']/1000000).round(2)
data.head()
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	mo
0	43597	237.0	150	2787.97	Avatar	7.2	11800	2009	
1	43598	300.0	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	
2	43599	245.0	107	880.67	Spectre	6.3	4466	2015	

Fetching queries from dataframe

Lets say we are interested in fetching all highly rates movies (ratings > 7)

Masking

```
data['vote_average'] > 7
```

```
0        True
1       False
2       False
3        True
4       False
        ...
1460     True
1461     True
1462    False
1463    False
1464    False
Name: vote_average, Length: 1465, dtype: bool
```

How do we get the row values ?

- By applying `.loc[]`
- This is known as **filtering**

```
data.loc[data['vote_average'] > 7]
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012
14	43616	250.00	120	956.02	The Hobbit: The Battle of the Five Armies	7.1	4760	2014
16	43619	250.00	94	958.40	The Hobbit: The Desolation of Smaug	7.6	4524	2013
19	43622	200.00	100	1845.03	Titanic	7.5	7562	1997
...	...	...	...	...	...	...	...	...
1456	48321	0.01	20	7.00	Eraserhead	7.5	485	1977
1457	48323	0.00	5	0.00	The Mighty	7.1	51	1998

- it always a safe option to create a copy of your dataframe using **copy()** and perform any analysis using the copy

```
df = data.copy(deep=True)
```

You can also perform the filtering without even using `loc`

```
df[df['vote_average'] > 7]
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012
14	43616	250.00	120	956.02	The Hobbit: The Battle of the Five Armies	7.1	4760	2014
16	43619	250.00	94	958.40	The Hobbit: The Desolation of Smaug	7.6	4524	2013
19	43622	200.00	100	1845.03	Titanic	7.5	7562	1997

But this is not recommended. Why ?

- It can create a confusion between implicit/explicit indexing used as discussed before
- `loc` is also much faster

We can also return only the subsets of columns

```
df.loc[df['vote_average'] > 7, ['title','vote_average']]
# These will be the only 2 columns printed out
```

Multiple conditions to filter rows

What if we want to fetch recently released (after 2014) highly rated latest movies?

Notes for applying mutliple conditions:

- Use elementwise operator `&` or `|`
- **we cannot use `and` or `or` with dataframe as a dataframe has multiple values**
- **for multiple conditions, we need to put each separate condition within parenthesis `()`**

```
df.loc[(df['vote_average'] > 7) & (df['year'] >= 2015)].head()
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
30	43641	190.0	102	1506.25	Furious 7	7.3	4176	2015
78	43724	150.0	434	378.86	Mad Max: Fury Road	7.2	9427	2015
106	43773	135.0	100	532.95	The Revenant	7.3	6396	2015
123	43807	108.0	107	290.16	The	7.0	7208	2015

Get all the movies which are alphabetically before movie title 'Avengers'

```
df.loc[df['title'] < 'Avengers']
# String comparisons like this (>, <, ==) are also possible
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
0	43597	237.0	150	2787.97	Avatar	7.2	11800	2009

String methods in pandas

How you can you filters row which has "The" in their movie titles?

- To apply a string method to a column, we will be using the `str` attribute of the Series object.
  - `Series.str.function()`
- `Series.str` can be used to access the values of the series as strings and apply several methods to it.
- First we would need to access that series (or column), then add `.str`, and finally add the specific method we want to use.
- `str.contains()` function is used to test if pattern is contained within a string of a Series

1405	47686	2.0	23	20.91	perros	7.6	521	2000

Find movies containing 'The' in their title

```
df['title'].str.contains('The')
```

```
0       False
1       False
2       False
3        True
4       False
        ...
1460     True
1461    False
1462    False
1463    False
1464    False
Name: title, Length: 1465, dtype: bool
```

```
df.loc[df['title'].str.contains('The')]
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012
9	43610	255.00	49	89.29	The Lone Ranger	5.9	2311	2013
11	43612	225.00	53	419.65	The Chronicles of Narnia: Prince Caspian	6.3	1630	2008
14	43616	250.00	120	956.02	The Hobbit: The Battle	7.1	4760	2014

If you want to search for movies that starts with "Batman"

- `.str.startswith()`

16	43619	250.00	94	958.40	The	7.6	4524	2013

```
df.loc[df['title'].str.startswith('Batman')]
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
5	43606	250.0	155	873.26	Batman v Superman: Dawn of Justice	5.7	7004	2016
74	43716	150.0	115	374.22	Batman Begins	7.5	7359	2005
128	43807	125.0	50	238.21	Batman & Robin	4.2	1418	1997
184	43896	100.0	48	336.53	Batman Forever	5.2	1498	1995

Find Top 5 most popular movies?

```
df.sort_values(['popularity'],ascending=False).head(5)
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
58	43692	165.0	724	675.12	Interstellar	8.1	10867	2014
78	43724	150.0	434	378.86	Mad Max: Fury Road	7.2	9427	2015

Get the list of movies directed by 'Christopher Nolan'

```
df.loc[df['director_name'] == 'Christopher Nolan',['title']]
```

	title
3	The Dark Knight Rises
45	The Dark Knight
58	Interstellar
59	Inception
74	Batman Begins
565	Insomnia
641	The Prestige
1341	Memento

- The string indicating "Christopher Nolan" could have contain trailing and leading spaces.
- The better way is to use string method - `contains`

Grouping

we want to know the number of movies released by a particular director

```
df.loc[df['director_name'] == 'Christopher Nolan',['title']].count()
```

```
title    8
dtype: int64
```

What if we have to do this all possible directors?

- `.value_counts()`

```
df["director_name"].value_counts()
```

```
Steven Spielberg    26
Martin Scorsese     19
Clint Eastwood      19
Woody Allen         18
```

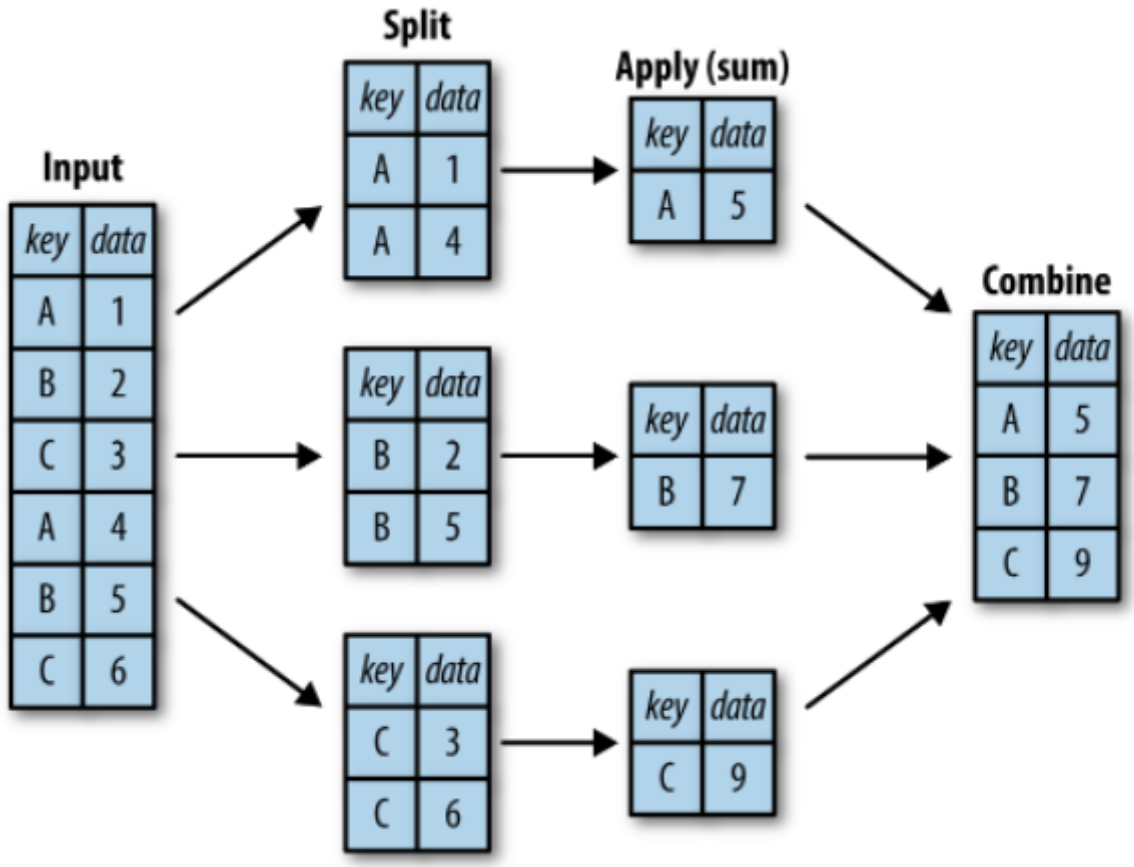
```
Robert Rodriguez    16
                    ..
Andrew Adamson       5
Mira Nair            5
Miguel Arteta        5
Sidney Lumet         5
Les Mayfield         5
Name: director_name, Length: 199, dtype: int64
```

Lets say, now you are interested in finding the highest budget movie of every director?

- Grouping

What is Grouping ?

- In simpler terms it could be understood through the terms - Split, apply, combine



- **Split**: Involves breaking up and grouping a DataFrame depending on the value of the specified key.
- **Apply**: Involves computing some function, usually an aggregate, transformation, or filtering, within the individual groups.
- **Combine**: Merges the results of these operations into an output array.

Note: All these steps are to understand the topic, not for real

Group based Aggregates

Now we want to know the count of movies for each director name

```
df.groupby('director_name')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f8918550850>
```

- it's a **DataFrameGroupBy type object**, **NOT a DataFrame** type object

What is `groupby('director_name')` doing?

- It is **grouping all rows in which director_name value is same**
- All the **rows having same director_name will be grouped together**

Now we want only 1 column (`budget`) of movies from the result of grouping and take it's `max`

```
df.groupby('director_name')['budget'].max()
```

```
director_name
Adam McKay                     100.0
Adam Shankman                   80.0
Alejandro González Iñárritu    135.0
Alex Proyas                    140.0
Alexander Payne                 30.0
                               ...
Wes Craven                      40.0
Wolfgang Petersen              175.0
Woody Allen                     30.0
Zack Snyder                    250.0
Zhang Yimou                     94.0
Name: budget, Length: 199, dtype: float64
```

Similarly, if `value_counts()` wasn't available, same thing can be done by groupby also

```
df.groupby('director_name')['title'].count()
```

```
director_name
Adam McKay                     6
Adam Shankman                  8
Alejandro González Iñárritu    6
Alex Proyas                    5
Alexander Payne                5
                              ..
Wes Craven                    10
Wolfgang Petersen              7
Woody Allen                   18
Zack Snyder                    7
Zhang Yimou                    6
Name: title, Length: 199, dtype: int64
```

Question: who is the **most productive director**?

Lets keep it simple for now, lets calculate who has directed maximum number of movies

```
df.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
```

Looks like `Steven Spielberg` has directed maximum number of movies

But does it make him the most productive director?

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

Lets calculate the number of active years of each director

How would you calculate active years for every director?

You would have to calculate both `min` and `max` of `year` and then subtract it.

- using `aggregate()` function

```
df_agg = df.groupby(['director_name'])[["title", "year"]].aggregate({"year":['min','max'],
df_agg
```

	year		title
	min	max	count
director_name			
Adam McKay	2004	2015	6
Adam Shankman	2001	2012	8
Alejandro González Iñárritu	2000	2015	6

- `director_name` column has turned into row labels
- We see some 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, still 2D
- Multi-level indexes are possible both for rows and columns

Zack Snyder	2004	2016	7

```
df_agg.columns
```

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

As we can, the level-1 column names are `year` and `title`.

if we print "year" column, it should give us both max and min.

```
df_agg["year"]
```

	min	max
director_name		
Adam McKay	2004	2015

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

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

Alex Proyas	1994 2016

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

	year_min	year_max	title_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

- How can we convert row labels into columns?
  - `reset_index()`

```
df_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
...	...	...	...	...
197	Zack Snyder	2004	2016	7

finding the most productive director

we can calculate rate of directing movies by `title_count` / `yrs_active`

```
df_agg["yrs_active"] = df_agg["year_max"] - df_agg["year_min"]
df_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

Now we can calculate the rate of directing movies and sort the values

```
df_agg["movie_per_yr"] = df_agg["title_count"] / df_agg["yrs_active"]
df_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

Group based Filtering

Question : How we find details of the movies by high budget directors?

Lets assume, any director who has created a >100M movie in past is a high budget director

question is **not asking us to give the name of the directors who have directed high budget movies**

[ ] ↳ *4 cells hidden*

Group based Transformation

Suppose, for every movie, we want to find out if it was an expensive movie for its director

How do we assess the budget of any movie wrt director?

we can subtract the average `budget` of a director from `budget` col, for each director

How can we do that ?

- Group data acc to `director_name`
- Calc its average `budget`
- Subtract it from the data of that `director_name`
- This process of changing data using group property is known as **Group based Transformation**

Just like `groupby().filter()`, we will use `grouby().transform()` function here

```
def sub_avg(x):
  x["budget"] -= x["budget"].mean()

df.groupby(['director_name']).transform(sub_avg)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_loc(s
tolerance)
   3079             try:
-> 3080                 return self._engine.get_loc(casted_key)
   3081             except KeyError as err:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index_class_helper.pxi in pandas._libs.index.Int64Engine._check_type()

KeyError: 'budget'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
```

13 frames

```
~/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in get_loc(s
tolerance)
   3080                 return self._engine.get_loc(casted_key)
   3081             except KeyError as err:
-> 3082                 raise KeyError(key) from err
   3083
   3084         if tolerance is not None:

KeyError: 'budget'
```

There is a keyerror, but `budget` column is present in our data

Does transform expect us to provide a column?

```
def inspect(x):
  print(x)
  print(type(x))
  raise

df.groupby(['director_name']).transform(inspect)
```

```
176    43882
323    44151
366    44236
505    44503
839    45301
916    45443
Name: id_x, dtype: int64
<class 'pandas.core.series.Series'>
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-78-5b8294aaf626> in <module>
      4   raise
      5
----> 6 df.groupby(['director_name']).transform(inspect)

                                 9 frames
<ipython-input-78-5b8294aaf626> in inspect(x)
```

Look at the data type of x: pandas Series

Hence transform() can never work with 2 or more cols

What should we do about our problem then?

We can pass a column

```
def sub_avg(x):
  x -= x.mean()
  return x

df.groupby(['director_name'])["budget"].transform(sub_avg)
```

```
0       130.300000
1       141.857143
2       150.142857
3       124.375000
4       174.004545
           ...
1460    -47.478947
1461    -11.976667
1462    -21.700000
1463    -10.890909
1464    -31.168750
Name: budget, Length: 1465, dtype: float64
```

We want to filter the movies whose budget was even higher than the average revenue of the director from his/her other movies

we can subtract the average `revenue` of a director from `budget` col, for each director

But we can't use `transform` here as it expects only one column

How can we do it ? => `.apply()`

- We need to group data acc to `director_name`

- Subtracting mean of `budget` from `revenue`

```
def func(x):
  x["risky"] = x["budget"] - x["revenue"].mean() >= 0
  return x
df_risky = df.groupby("director_name").apply(func)
df_risky
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009
1	43598	300.00	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007
2	43599	245.00	107	880.67	Spectre	6.3	4466	2015
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012
4	43602	258.00	115	890.87	Spider-Man 3	5.9	3576	2007
...	...	...	...	...	...	...	...	...
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978
1461	48370	0.03	19	3.15	Clerks	7.4	755	1994
1462	48375	0.00	7	0.00	Rampage	6.0	131	2009

```
df_risky.loc[df_risky["risky"]]
```

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year
7	43608	200.0	107	586.09	Quantum of Solace	6.1	2965	2008
12	43614	200.0	135	1045.71	Pirates of the Caribbean:	6.4	4948	2011

Note: `apply()` can be applied on any dataframe along any particular axis

- By default axis = 0

15	43618	200.0	37	310.67	Robin	6.2	1398	2010

```
df[['revenue', 'budget']].apply(np.sum, axis = 0)
```

```
revenue    209867.04
budget      70353.62
dtype: float64
```

```
df[['revenue', 'budget']].apply(np.sum, axis = 1)
```

```
0       3024.97
1       1261.00
2       1125.67
3       1334.94
4       1148.87
         ...
1460       0.32
1461       3.18
1462       0.00
1463       0.00
1464       2.26
Length: 1465, dtype: float64
```