Pandas-02

Outline

- **Concatenation**
  - pd.concat()
    - axis for concat
    - Inner join
    - Outer join
- **Merging dataframes**
  - Concat v/s Merge
  - Outer join, Left join, Inner join
  - Quiz 1
- **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()`
  - Quiz 2, Quiz 3
  - Grouping
    - Split, Apply, Combine
    - `groupby()`
    - Group based Aggregates
    - `reset_index()`
    - Group based Filtering
    - Group based Transformation
    - `apply()`
    - Quiz 4

Today's Agenda

- Today's, we'll continue looking at **Pandas** library
- We will look at how to concatenate and merge two dataframe
- We'll also look at **fetching data from the dataframe** for IMDB dataset which is our buisness for today

In [2]:

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

Concatenating DataFrames

- We can **join 2 or more DataFrames to form a single DataFrame**
- Let's start by creating 2 DataFrames

In [6]:

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

Out[6]:

	A	B
0	10	20
1	30	40

In [3]:

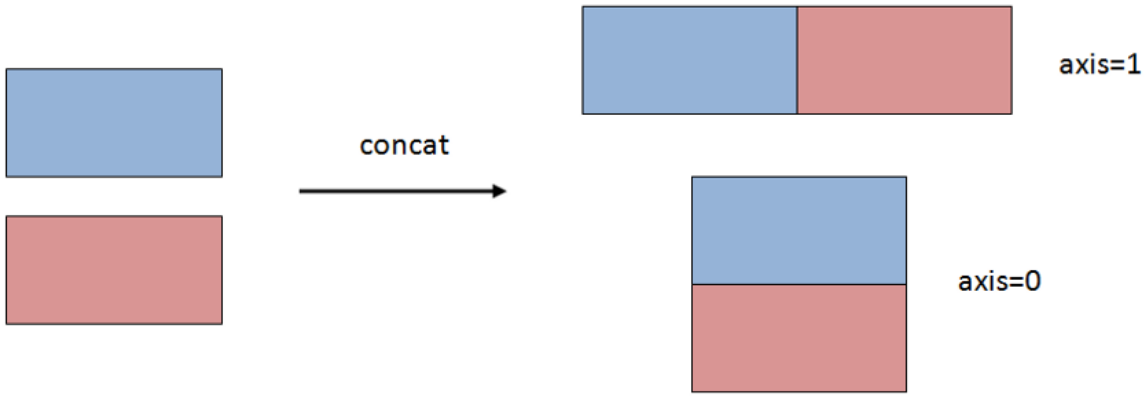
```
1 b
```

Out[3]:

	A	C
0	10	20
1	30	40

We just use `pd.concat()`

- Pass in a list of DataFrames that we want to combine



In [8]:

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

Out[8]:

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

Notice a few things here:

- By **default, `axis=0` for concatenation**
- These **means concatenation is done row-wise**
- **Column `A`** in both DataFrames is **combined into a single column**
  - Column **name matching**
- It concatenated in such a way as if
  - **DataFrame `a`** did **NOT have any values in Column `C`**
  - **DataFrame `b`** did **NOT have any values in Column `B`**
- Also the indices of the rows are preserved

In [5]:

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

Out[5]:

	A	B	C
0	10	20.0	NaN
0	10	NaN	20.0

We obviously want the indices to be unique for each row

How can we do this ?

- By setting `ignore_index = True`

In [6]:

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

Out[6]:

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

We can concatenate column-wise as well

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

- `axis=1`

In [7]:

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

Out[7]:

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

As you can see here:

- **Column `A` is NOT combined as one**
- 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

In [8]:

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

Out[8]:

		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

Also By default, the entries for which no data is available are filled with NA values

We can change this behaviour by specifying the type of `join` that should be used to combine data

Which join can we use if we want a union of cols ?

- Outer join
- Set as default by pd.concat

In [9]:

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

Out[9]:

	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 ?

- We need to use the inner join for that
- There will be no null values in any cell

In [10]:

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

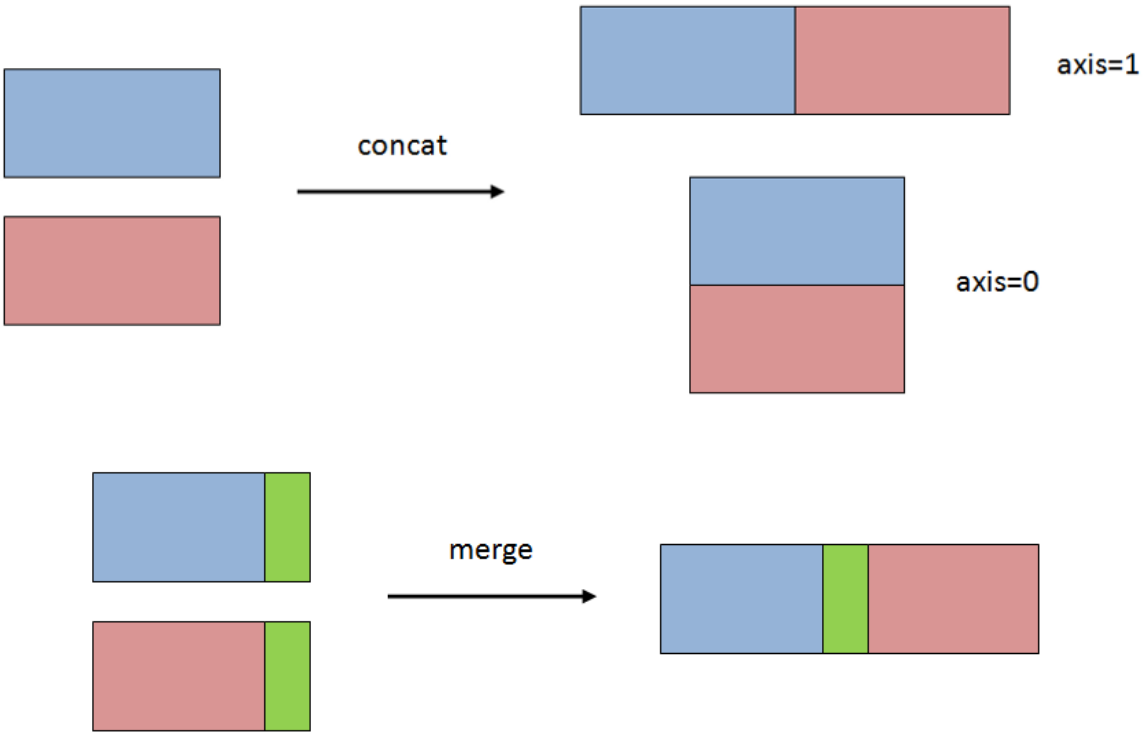
Out[10]:

	A
0	10
1	30
0	10
1	30

So far we have only concatenated and not merged data

Bur 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



Lets explore merging in more detail

- This **works like join in SQL**
- Lets see what this means

Let's create 2 DataFrames

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

In [2]:

```
import pandas as pd

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

Out[2]:

	userid	name
0	1	A
1	2	B
2	3	C

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

In [3]:

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

Out[3]:

	userid	msg
0	1	hello
1	1	bye
2	2	hi

Now suppose you want to know the name of the person who sent a message

How can we do that ?

- We need to create a new dataframe
- It will take data from both `msgs` and `users`

So should can we use pd.concat() for this ?

- No
- pd.concat() does not work according to the values in the columns

How can we do this then ?

- Using pd.merge()

How does it work ?

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

In [13]:

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

Out[13]:

	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

For eg:

- Dataframe 1: col for employees name might be `name`
- Dataframe 2: col for employees name might be `employee`

How can we merge the 2 dataframes in this situation ?

- Using the `left_on` and `right_on` keywords
- `left_on` : Specifies the key of the 1st dataframe
- `right_on` : Specifies the key of the 2nd dataframe

Lets see how it works

In [4]:

```
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
```

Out[4]:

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

In above codes we have skipped one 1 important part

Specifying type of joins to merge the dataframes

Where does it become relevant ?

- Notice that `users` has a userid = 3 but `msgs` does not
- When we merge these dataframes the userid = 3 is not included
- Only the userid common in both dataframes is shown

- **What if we want to change this behaviour ?**
  - This is where joins can be used

There are different types of joins

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()
- Lets code it now

In [5]:

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

Out[5]:

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

Now lets say we want 1 dataframe having all info of all the users. How can we do that ?

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

In [16]:

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

Out[16]:

	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

In [17]:

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

Out[17]:

	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

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

- Returns join over cols of right input

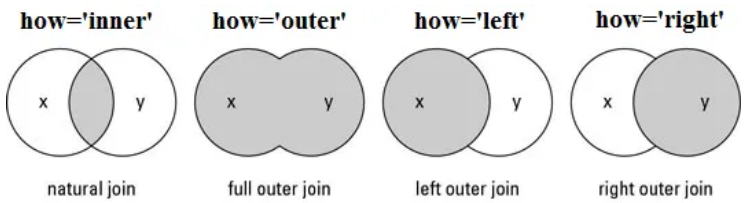
In [18]:

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

Out[18]:

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

Lets visualise these joins using a venn diagram



Quiz:

1. For the concat(), if the axis=1, it will join the dataframes

a. vertically

b. horizontally

Ans: horizontally

IMDB Movie Business Use-case

Imagine that you are working as a Data Scientist for an Analytics firm

- And you are analysing some movie trends for a client
- IMDB has online database of information related to movies
- The database contains info of several years about:
  - Movies
  - Rating
  - Director
  - Popularity
  - Revenue & Budget

In [6]:

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

Lets download and read the IMDB dataset

- Lets first download the dataset
- File1: https://drive.google.com/file/d/1s2TkjSpzNc4SyxqRrQleZyDIHIc7bxnd/view?usp=sharing (https://drive.google.com/file/d/1s2TkjSpzNc4SyxqRrQleZyDIHIc7bxnd/view?usp=sharing)
- File2: https://drive.google.com/file/d/1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm/view?usp=sharing (https://drive.google.com/file/d/1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm/view?usp=sharing)

In [8]:

```
!pip install gdown
!gdown 1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd

```

```
Collecting gdown
  Downloading gdown-4.6.0-py3-none-any.whl (14 kB)
Requirement already satisfied: tqdm in c:\users\shelendra\anaconda3\lib\site-packages (from gdown) (4.62.3)
Requirement already satisfied: requests[socks] in c:\users\shelendra\anaconda3\lib\site-packages (from gdown) (2.26.0)
Requirement already satisfied: beautifulsoup4 in c:\users\shelendra\anaconda3\lib\site-packages (from gdown) (4.10.0)
Requirement already satisfied: filelock in c:\users\shelendra\anaconda3\lib\site-packages (from gdown) (3.3.1)
Requirement already satisfied: six in c:\users\shelendra\anaconda3\lib\site-packages (from gdown) (1.16.0)
Requirement already satisfied: soupsieve>1.2 in c:\users\shelendra\anaconda3\lib\site-packages (from beautifulsoup4->gd
own) (2.2.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\shelendra\anaconda3\lib\site-packages (from requests[socks]->gd
own) (3.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\shelendra\anaconda3\lib\site-packages (from requests[sock
s]->gdown) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\shelendra\anaconda3\lib\site-packages (from reques
ts[socks]->gdown) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\shelendra\anaconda3\lib\site-packages (from requests[s
ocks]->gdown) (1.26.7)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in c:\users\shelendra\anaconda3\lib\site-packages (from requests
[socks]->gdown) (1.7.1)
Requirement already satisfied: colorama in c:\users\shelendra\anaconda3\lib\site-packages (from tqdm->gdown) (0.4.4)
Installing collected packages: gdown
Successfully installed gdown-4.6.0

[notice] A new release of pip available: 22.1.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Downloading...
From: https://drive.google.com/uc?id=1s2TkjSpzNc4SyxqRrQleZyDIHlc7bxnd (https://drive.google.com/uc?id=1s2TkjSpzNc4Syxq
RrQleZyDIHlc7bxnd)
To: C:\Users\Shelendra\movies.csv

  0%|          | 0.00/112k [00:00<?, ?B/s]
100%|##########| 112k/112k [00:00<00:00, 37.5MB/s]
```

In [9]:

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

```
Downloading...
From: https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGLVUQurbHDvStePlEJm (https://drive.google.com/uc?id=1Ws-_s1fHZ9nHfGL
VUQurbHDvStePlEJm)
To: C:\Users\Shelendra\directors.csv

  0%|          | 0.00/65.4k [00:00<?, ?B/s]
100%|##########| 65.4k/65.4k [00:00<00:00, 494kB/s]
100%|##########| 65.4k/65.4k [00:00<00:00, 490kB/s]
```

Reading the dataset

Here we have two csv files 'movies.csv' and 'directors.csv' in the data.

Lets read both the csv files using pd.read_csv()

In [10]:

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

Out[10]:

	Unnamed: 0	id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day
0	0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday
1	1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	4763	2007	May	Saturday
2	2	43599	245000000	107	880674609	Spectre	6.3	4466	4764	2015	Oct	Monday
3	3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106	4765	2012	Jul	Monday
4	5	43602	258000000	115	890871626	Spider-Man 3	5.9	3576	4767	2007	May	Tuesday

So what kind of questions can we ask from this dataset?

- Since have `popularity` , can find, "Top 10 most popular movies"
- Or using `vote_average` , I can find some highest rated movies also
- Since, `year` is given, we can find nnumber of movies released per year
- And may be using both `budget` and `year` , I can find highest budget movies in a year

But can we ask more interesting/deeper questions?

- Do you think we can find the most productive director?
- Which director produces high budget films?
- Highest and lowest rated movies for every month in a particular year

So, lets explore the dataset

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

Inorder to get rid of this column-

We can simply add one more argument `index_col=0`

The default value is index_col=None.

If we set index_col=0 we're explicitly stating to treat the first column as the index

In [11]:

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

Out[11]:

	id	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	4763	2007	May	Saturday
2	43599	245000000	107	880674609	Spectre	6.3	4466	4764	2015	Oct	Monday
3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106	4765	2012	Jul	Monday
5	43602	258000000	115	890871626	Spider-Man 3	5.9	3576	4767	2007	May	Tuesday

In [12]:

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

Out[12]:

```
(1465, 11)
```

- The movies df contains 1465 rows,11 columns

Lets read another dataset:

In [13]:

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

Out[13]:

	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

In [14]:

```
directors.shape
```

Out[14]:

```
(2349, 3)
```

Now directors df contains:

- 2349 rows,3 columns

Summary

1. Movie dataset contains info about movies, release, popularity, ratings and the director ID
2. Director dataset contains detailed info about the director

Now, if we want to know the details about Director of a particular movie, we may have to join these datasets

Merging of both Dataframe:

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

- We can do it using `merge()`
- Recall what is `merge()` ?

  As you have seen in previous lecture:
  - The merge() method enables us to combine two dataframes together,
  - But it requires specifying the columns as a merge key.

Question: So on which column we should merge the dfs ?

- We will merge both the dataframes into single dataframe on the ID columns (representing unique director) in both the datasets
- If you look at the values of columns `director_id` of movies df and `id` of directors df
- You will observe that values of id in movies are referred or taken from directors df

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

For that, lets first check number of unique director values in our `movies` data

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

Lets first check for movie df

In [15]:

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

Out[15]:

```
199
```

Movie df contains 199 unique values of director id

Lets check number of unique diretors in `directors` df

In [16]:

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

Out[16]:

```
2349
```

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

What can we infer from this?

- Looks like directors in `movies` is a subset of directors in `directors`
- But, still 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 ?

We can do it using `isin()` method of pandas

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).

In [17]:

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

Out[17]:

```
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
```

- Notice that this is like a boolean "mask"
- It returns a df similar to the original df,
- where rows are checked and a boolean series is returned which is True
- wherever value of movies['director_id'] is present in directors['id'].

Lets see if there is any False here.

In [18]:

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

Out[18]:

```
True
```

- If you know SQL, you can relate it to the concept of primary key
- Else we can say all the values of `director_id` of movies df is referred from id column in `directors` df

Lets finally merge the dataframes

Do we need to keep all the rows for movies?

Yes

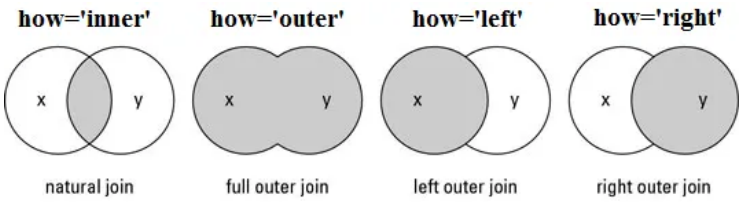
Do we need to keep all the rows of directors?

No, only the ones for which we have a corresponding row in movies

###Question: So which `merge()` method do you think we should apply here ?

We can use LEFT OUTER JOIN

Recall what will Left Outer Join do?



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

Lets perform merge() using LEFT OUTER JOIN

In [19]:

```
# 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
```

Out[19]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	director_id	year	month	day	director_name	id_y	gen
0	43597	237000000	150	2787965087	Avatar	7.2	11800	4762	2009	Dec	Thursday	James Cameron	4762	M
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	4763	2007	May	Saturday	Gore Verbinski	4763	M
2	43599	245000000	107	880674609	Spectre	6.3	4466	4764	2015	Oct	Monday	Sam Mendes	4764	M
3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106	4765	2012	Jul	Monday	Christopher Nolan	4765	M
4	43602	258000000	115	890871626	Spider-Man 3	5.9	3576	4767	2007	May	Tuesday	Sam Raimi	4767	M
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1460	48363	0	3	321952	The Last Waltz	7.9	64	4809	1978	May	Monday	Martin Scorsese	4809	M
1461	48370	27000	19	3151130	Clerks	7.4	755	5369	1994	Sep	Tuesday	Kevin Smith	5369	M
1462	48375	0	7	0	Rampage	6.0	131	5148	2009	Aug	Friday	Uwe Boll	5148	M
1463	48376	0	3	0	Slacker	6.4	77	5535	1990	Jul	Friday	Richard Linklater	5535	M
1464	48395	220000	14	2040920	El Mariachi	6.6	238	5097	1992	Sep	Friday	Robert Rodriguez	5097	N

1465 rows × 14 columns

Notice, two stranger id columns `id_x` and `id_y` .

What do you think they are?

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

Question: Do you think any column is redundant here and can be dropped?

- `id_y` is redundant as it is same as `director_id`
- But we dont require `director_id` further

So we can simply drop these features

In [20]:

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

Out[20]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237000000	150	2787965087	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
1	43598	300000000	139	961000000	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male
2	43599	245000000	107	880674609	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male
3	43600	250000000	112	1084939099	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
4	43602	258000000	115	890871626	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male

Feature Exploration

Lets explore all the features in the merged dataset

In [21]:

```
1 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
```

- Looks like only `gender` column has missing values (will come later)

Lets describe these features to know more about their range of values

In [22]:

```
1 data.describe()
```

Out[22]:

	id_x	budget	popularity	revenue	vote_average	vote_count	year
count	1465.000000	1.465000e+03	1465.000000	1.465000e+03	1465.000000	1465.000000	1465.000000
mean	45225.191126	4.802295e+07	30.855973	1.432539e+08	6.368191	1146.396587	2002.615017
std	1189.096396	4.935541e+07	34.845214	2.064918e+08	0.818033	1578.077438	8.680141
min	43597.000000	0.000000e+00	0.000000	0.000000e+00	3.000000	1.000000	1976.000000
25%	44236.000000	1.400000e+07	11.000000	1.738013e+07	5.900000	216.000000	1998.000000
50%	45022.000000	3.300000e+07	23.000000	7.578164e+07	6.400000	571.000000	2004.000000
75%	45990.000000	6.600000e+07	41.000000	1.792469e+08	6.900000	1387.000000	2009.000000
max	48395.000000	3.800000e+08	724.000000	2.787965e+09	8.300000	13752.000000	2016.000000

In [23]:

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

Out[23]:

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

- Generally budget and revenue for Hollywood movies is in million dollars
- But notice, 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?

In [24]:

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

```

Out[24]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237000000	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
1	43598	300000000	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male
2	43599	245000000	107	880.67	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male
3	43600	250000000	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
4	43602	258000000	115	890.87	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1460	48363	0	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male
1461	48370	27000	19	3.15	Clerks	7.4	755	1994	Sep	Tuesday	Kevin Smith	Male
1462	48375	0	7	0.00	Rampage	6.0	131	2009	Aug	Friday	Uwe Boll	Male
1463	48376	0	3	0.00	Slacker	6.4	77	1990	Jul	Friday	Richard Linklater	Male
1464	48395	220000	14	2.04	El Mariachi	6.6	238	1992	Sep	Friday	Robert Rodriguez	NaN

1465 rows × 12 columns

Similarly, we can do it for 'budget' as well

In [25]:

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

Out[25]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.0	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
1	43598	300.0	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male
2	43599	245.0	107	880.67	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male
3	43600	250.0	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
4	43602	258.0	115	890.87	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male

Now, finally our dataset is ready for analysis

Lets query the dataset to ask some interesting questions

Fetching queries from dataframe

Lets say we are interested in fetching all highly rates movies

Say, I define any movie to be high rated when ratings > 7

How can we do this we get movies with ratings > 7?

- Have you ever come across or seen SQL?
- For now, just know that **SQL is used to fetch data from databases**
- We can use basic **Pandas operations** in a similar way to **fetch desired data from loaded data**

Masking

Lets first create a mask to filter such movies

- In SQL: SELECT * FROM movies WHERE vote_average > 7
- In pandas:

In [26]:

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

Out[26]:

```
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
```

But we still don't know the row values ... Only that which row satisfied the condtion

How do we get the row values then ?

- By applying `movies.loc`
- This is known as filtering

In [29]:

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

Out[29]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
14	43616	250.00	120	956.02	The Hobbit: The Battle of the Five Armies	7.1	4760	2014	Dec	Wednesday	Peter Jackson	Male
16	43619	250.00	94	958.40	The Hobbit: The Desolation of Smaug	7.6	4524	2013	Dec	Wednesday	Peter Jackson	Male
19	43622	200.00	100	1845.03	Titanic	7.5	7562	1997	Nov	Tuesday	James Cameron	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1456	48321	0.01	20	7.00	Eraserhead	7.5	485	1977	Mar	Saturday	David Lynch	Male
1457	48323	0.00	5	0.00	The Mighty	7.1	51	1998	Oct	Friday	Peter Chelsom	Male
1458	48335	0.06	27	3.22	Pi	7.1	586	1998	Jul	Friday	Darren Aronofsky	Male
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male
1461	48370	0.03	19	3.15	Clerks	7.4	755	1994	Sep	Tuesday	Kevin Smith	Male

301 rows × 12 columns

- Here, we will **only get those rows for which `data['vote_average'] > 7`**
- If we save the result back in `data` , **all other rows will be deleted**
- We might **still need the original data to work with**
- So it always a safe option to create a copy of your dataframe using **copy()** and perform any analysis using the copy

In [28]:

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

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

In [29]:

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

Out[29]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
14	43616	250.00	120	956.02	The Hobbit: The Battle of the Five Armies	7.1	4760	2014	Dec	Wednesday	Peter Jackson	Male
16	43619	250.00	94	958.40	The Hobbit: The Desolation of Smaug	7.6	4524	2013	Dec	Wednesday	Peter Jackson	Male
19	43622	200.00	100	1845.03	Titanic	7.5	7562	1997	Nov	Tuesday	James Cameron	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1456	48321	0.01	20	7.00	Eraserhead	7.5	485	1977	Mar	Saturday	David Lynch	Male
1457	48323	0.00	5	0.00	The Mighty	7.1	51	1998	Oct	Friday	Peter Chelsom	Male
1458	48335	0.06	27	3.22	Pi	7.1	586	1998	Jul	Friday	Darren Aronofsky	Male
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male
1461	48370	0.03	19	3.15	Clerks	7.4	755	1994	Sep	Tuesday	Kevin Smith	Male

301 rows × 12 columns

But this is not recommended. Why ?

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

Now this is how we can start quering our data based on the conditions we want

We can also return only the subsets of columns

- Works **just like slicing**

In [30]:

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

Out[30]:

	title	vote_average
0	Avatar	7.2
3	The Dark Knight Rises	7.6
14	The Hobbit: The Battle of the Five Armies	7.1
16	The Hobbit: The Desolation of Smaug	7.6
19	Titanic	7.5
...	...	...
1456	Eraserhead	7.5
1457	The Mighty	7.1
1458	Pi	7.1
1460	The Last Waltz	7.9
1461	Clerks	7.4

301 rows × 2 columns

So far we saw only single condition for filtering

What if we have multiple conditions to filter rows?

What if we want to filter highly rated latest movies?

Notice that two conditions are involved here

1. Movies need to be highly rated i.e.. > 7
2. It should be recent, say released after 2014

- In SQL: SELECT * FROM movies WHERE vote_average > 7 AND year >= 2015
- We can **use AND operator b/w multiple conditions**

Recall how we apply mutliple conditions in numpy ?

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

In [31]:

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

Out[31]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
30	43641	190.0	102	1506.25	Furious 7	7.3	4176	2015	Apr	Wednesday	James Wan	Male
78	43724	150.0	434	378.86	Mad Max: Fury Road	7.2	9427	2015	May	Wednesday	George Miller	Male
106	43773	135.0	100	532.95	The Revenant	7.3	6396	2015	Dec	Friday	Alejandro González Iñárritu	Male
162	43867	108.0	167	630.16	The Martian	7.6	7268	2015	Sep	Wednesday	Ridley Scott	Male
312	44128	75.0	48	108.15	The Man from U.N.C.L.E.	7.1	2265	2015	Aug	Thursday	Guy Ritchie	Male

####Question: Get me all the movies which are alphabetically before movie title 'Avengers'

In [32]:

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

Out[32]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.0	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
23	43629	200.0	78	1025.49	Alice in Wonderland	6.4	4645	2010	Mar	Wednesday	Tim Burton	Male
40	43656	200.0	45	769.65	2012	5.6	4903	2009	Oct	Saturday	Roland Emmerich	Male
41	43657	200.0	39	325.23	A Christmas Carol	6.6	1095	2009	Nov	Wednesday	Robert Zemeckis	Male
69	43709	155.0	39	167.30	Alexander	5.6	927	2004	Nov	Sunday	Oliver Stone	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1390	47491	3.0	7	20.97	A Room with a View	6.9	156	1985	Dec	Friday	James Ivory	Male
1395	47575	3.0	3	0.00	Amnesiac	4.1	52	2015	Aug	Friday	Michael Polish	Male
1405	47686	2.0	23	20.91	Amores perros	7.6	521	2000	Jun	Friday	Alejandro González Iñárritu	Male
1432	47970	0.0	3	0.00	All the Real Girls	5.9	30	2003	Aug	Friday	David Gordon Green	Male
1440	48155	0.0	0	0.00	Alleluia! The Devil's Carnival	6.0	2	2016	Mar	Tuesday	Darren Lynn Bousman	Male

125 rows × 12 columns

String methods in pandas

What kind of questions can be use string methods for?

Find rows which contains a particular string

Lets say you want to filter movies that contain pattern or substring 'The' in it

####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.

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.
- First we would need to access that series (or column), then add .str, and finally add the specific method we want to use.

Here coming back to our task:

- we can find the pattern 'The' in movies using str.contains()
- str.contains() function is used to test if pattern is contained within a string of a Series

In [33]:

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

Out[33]:

```
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
```

This will result in Series of True and False.

Again, we can use it as mask or fancy indexing, can use .loc to extract rows:

In [34]:

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

Out[34]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
9	43610	255.00	49	89.29	The Lone Ranger	5.9	2311	2013	Jul	Wednesday	Gore Verbinski	Male
11	43612	225.00	53	419.65	The Chronicles of Narnia: Prince Caspian	6.3	1630	2008	May	Thursday	Andrew Adamson	Male
14	43616	250.00	120	956.02	The Hobbit: The Battle of the Five Armies	7.1	4760	2014	Dec	Wednesday	Peter Jackson	Male
16	43619	250.00	94	958.40	The Hobbit: The Desolation of Smaug	7.6	4524	2013	Dec	Wednesday	Peter Jackson	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1440	48155	0.00	0	0.00	Alleluia! The Devil's Carnival	6.0	2	2016	Mar	Tuesday	Darren Lynn Bousman	Male
1443	48192	0.35	35	29.40	The Evil Dead	7.3	894	1981	Oct	Thursday	Sam Raimi	Male
1449	48244	0.25	6	0.06	The Canyons	4.1	75	2013	Jul	Monday	Paul Schrader	NaN
1457	48323	0.00	5	0.00	The Mighty	7.1	51	1998	Oct	Friday	Peter Chelsom	Male
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male

327 rows × 12 columns

Do you remember another handy string method we learnt earlier - `startswith`

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

In [35]:

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

Out[35]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
5	43606	250.0	155	873.26	Batman v Superman: Dawn of Justice	5.7	7004	2016	Mar	Wednesday	Zack Snyder	Male
74	43716	150.0	115	374.22	Batman Begins	7.5	7359	2005	Jun	Friday	Christopher Nolan	Male
128	43807	125.0	50	238.21	Batman & Robin	4.2	1418	1997	Jun	Friday	Joel Schumacher	Male
184	43896	100.0	48	336.53	Batman Forever	5.2	1498	1995	May	Wednesday	Joel Schumacher	Male
257	44025	80.0	59	280.00	Batman Returns	6.6	1673	1992	Jun	Friday	Tim Burton	Male
704	44956	35.0	44	411.35	Batman	7.0	2096	1989	Jun	Friday	Tim Burton	Male

So, from whateve we have learnt so far, we can answer a couple of questions

Question: How will you find Top 5 most popular movies?

- we can simply sort our data based on values of column 'popularity'

In [36]:

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

Out[36]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
58	43692	165.0	724	675.12	Interstellar	8.1	10867	2014	Nov	Wednesday	Christopher Nolan	Male
78	43724	150.0	434	378.86	Mad Max: Fury Road	7.2	9427	2015	May	Wednesday	George Miller	Male
119	43796	140.0	271	655.01	Pirates of the Caribbean: The Curse of the Bla...	7.5	6985	2003	Jul	Wednesday	Gore Verbinski	Male
120	43797	125.0	206	752.10	The Hunger Games: Mockingjay - Part 1	6.6	5584	2014	Nov	Tuesday	Francis Lawrence	Male
45	43662	185.0	187	1004.56	The Dark Knight	8.2	12002	2008	Jul	Wednesday	Christopher Nolan	Male

Okay, so who is your favorite Director?

My favourite director is **Christopher Nolan**

Lets say I am interested in getting names of all movies directed by my favourite director

Question: How will get list of movies directed by 'Christopher Nolan'?

In [37]:

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

Out[37]:

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

But do remember that we got lucky here.

- The string indicating "Christopher Nolan" could have been something else as well.
- The better way is to use string method - contains as we did earlier.

Quiz2:

1. How to get the details of movies released in the month of 'Jan' or 'Nov' from the dataset ?

   a. df.loc[(df['month']='Jan') | (df['month']='Nov')]

   b. df.loc[(df['month']=='Jan') || (df['month']=='Nov')]

   c. df.loc[df['month']=='Jan'| df['month']=='Nov']

   d. df.loc[(df['month']=='Jan') | (df['month']=='Nov')]

   Ans: d

Quiz3:

2. How to filter those records where movies released in the year(2015,2016,2012) from the above dataset ?

   a. df['year'].isin([2015, 2016, 2012])

   b. df['year'].in([2015, 2016, 2012])

   c. df['year']==([2015, 2016, 2012])

Ans:a

Grouping

Question :Now lets say we want to know the number of movies released by a particular director

In [38]:

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

Out[38]:

```
title    8
dtype: int64
```

What if we have to do this all possible directors?

I can probably do a value_counts() of the directors

In [39]:

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

Out[39]:

```
Steven Spielberg      26
Martin Scorsese       19
Clint Eastwood        19
Woody Allen           18
Ridley Scott          16
                      ..
Tim Hill               5
Jonathan Liebesman     5
Roman Polanski         5
Larry Charles          5
Nicole Holofcener      5
Name: director_name, Length: 199, dtype: int64
```

Let me complicate a question a bit more.

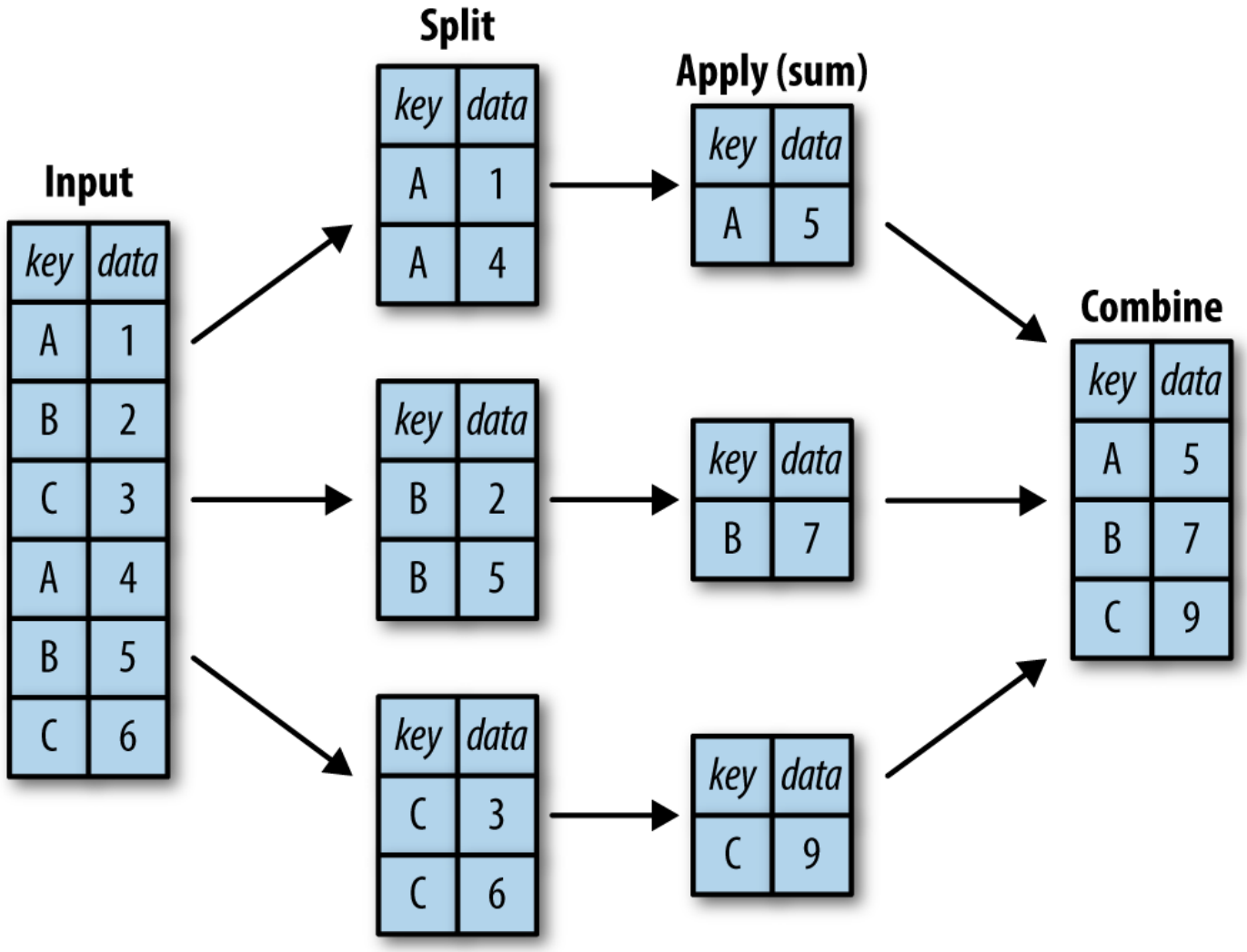
Lets say, now you are interested in finding the highest budget movie of every director?

Think about it, we will have to some group our rows director wise.

This can be be better solved by **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

In [40]:

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

Out[40]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000028586B66190>
```

- As you can see, 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`

In [41]:

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

Out[41]:

```
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, you could have done this by groupby also

In [42]:

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

Out[42]:

```
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
```

Now, lets say, you want to find who is the **most productive director**

Question: Which director according to you would be considered as most productive ?

Will you decide based on the number of movies released by a director?

Or will consider quality into consideration also?

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

In [43]:

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

Out[43]:

```
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.

We can calculate multiple aggregates in a single go using `aggregate()` function

In [44]:

```
df.head(10)
```

Out[44]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.0	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
1	43598	300.0	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male
2	43599	245.0	107	880.67	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male
3	43600	250.0	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
4	43602	258.0	115	890.87	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male
5	43606	250.0	155	873.26	Batman v Superman: Dawn of Justice	5.7	7004	2016	Mar	Wednesday	Zack Snyder	Male
6	43607	270.0	57	391.08	Superman Returns	5.4	1400	2006	Jun	Wednesday	Bryan Singer	Male
7	43608	200.0	107	586.09	Quantum of Solace	6.1	2965	2008	Oct	Thursday	Marc Forster	Male
8	43609	200.0	145	1065.66	Pirates of the Caribbean: Dead Man's Chest	7.0	5246	2006	Jun	Tuesday	Gore Verbinski	Male
9	43610	255.0	49	89.29	The Lone Ranger	5.9	2311	2013	Jul	Wednesday	Gore Verbinski	Male

In [ ]:

```

```

In [46]:

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

Out[46]:

	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

Hm, the output looks somewhat strange

- Notice that `director_name` column has turned into row labels, but thats fine.
- 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
- We have seen multi-level index for rows in the last class

Lets try to understand this better by printing the columns

In [45]:

```
df_agg.columns
```

Out[45]:

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

As we can, the level-1 column names are `year` and `title`. Lets try to print one of them

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

In [46]:

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

Out[46]:

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

In [47]:

```
df_agg.columns
```

Out[47]:

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

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

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

Notice that these are tuples, and we can just join them

In [47]:

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

Out[47]:

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

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

How can we convert row labels into columns?

- We can use `reset_index()`

- Each **row gets assigned a label number**

In [48]:

```
df_agg.reset_index()
```

Out[48]:

	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

Looks, PRETTY GOOD.

Now, using the new features, can we find the most productive director?

We need to know for how many years the director has been active.

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

So, lets calculate `yrs_active` first

In [49]:

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

Out[49]:

	year_min	year_max	title_count	yrs_active
director_name				
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

In [50]:

```
df_agg["movie_per_yr"] = df_agg["title_count"] / df_agg["yrs_active"]
df_agg.sort_values("movie_per_yr", ascending=False)
```

Out[50]:

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

- As we can see, "Tyler Perry" turns out to be the truly most productive director
- He has directed only 9 movies as compared to

So, we have seen couple of examples of how we can use grouping to calculate aggregates.

Group based Filtering

Apart from using aggregate functions, filtering operations that can also be performed using groupyby()

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

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

Lets first quickly see, how we would have solved that

In [49]:

```
df_dir_budget = df.groupby("director_name")["budget"].max().reset_index()
df_dir_budget
```

Out[49]:

	director_name	budget
0	Adam McKay	100.0
1	Adam Shankman	80.0
2	Alejandro González Iñárritu	135.0
3	Alex Proyas	140.0
4	Alexander Payne	30.0
...	...	...
194	Wes Craven	40.0
195	Wolfgang Petersen	175.0
196	Woody Allen	30.0
197	Zack Snyder	250.0
198	Zhang Yimou	94.0

199 rows × 2 columns

In [50]:

```
df_dir_budget = df.groupby("director_name")["budget"].max().reset_index()
#df_dir_budget
df_dir_budget.loc[df_dir_budget["budget"] >= 100, "director_name"]
```

Out[50]:

```
0                      Adam McKay
2      Alejandro González Iñárritu
3                     Alex Proyas
5                  Andrew Adamson
10                        Ang Lee
                  ...
187                   Tom Shadyac
188                    Tom Tykwer
189                    Tony Scott
195               Wolfgang Petersen
197                    Zack Snyder
Name: director_name, Length: 85, dtype: object
```

So what is the question asking us to do?

We want to filter the rows based on some group property (director's max budget movie)

- The cases when we want to filter rows based on group properties (aggragates) is called **Group Based |Filtering**
- We can group it by director and then use `groupby().filter` function
- **Rows from groups are filtered if they do not satisfy the boolean criterion** specified by func.

Now to filter values, we need to pass a filtering function to `filter()`

In [53]:

```
#df.groupby('director_name').filter(lambda x: x["budget"].max() >= 100)
```

Out[53]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male
1	43598	300.00	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male
2	43599	245.00	107	880.67	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male
4	43602	258.00	115	890.87	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male
...	...	...	...	...	...	...	...	...	...	...	...	...
1450	48267	0.40	33	100.00	Mad Max	6.6	1213	1979	Apr	Thursday	George Miller	Male
1451	48268	0.20	13	4.51	Swingers	6.8	253	1996	Oct	Friday	Doug Liman	Male
1452	48274	0.00	5	2.61	Three	6.3	31	2010	Dec	Thursday	Tom Tykwer	Male
1458	48335	0.06	27	3.22	Pi	7.1	586	1998	Jul	Friday	Darren Aronofsky	Male
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male

679 rows × 12 columns

- Notice, in the output, we have some low budget movies like MadMax.
- But they were filtered because their directors have directed high budget movies as well in the past

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?

May be 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

In [51]:

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

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

~\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

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

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)
C:\Users\SHELEN~1\AppData\Local\Temp/ipykernel_23004/3490489248.py in <module>
```

Looks like there is a keyerror, but we are sure that `budget` column is present in our data

Does transform expect us to provide a column?

Lets try to inspect the code

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

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

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

In [55]:

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

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

Out[55]:

```
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
```

Notice how some numbers for the movies which are of higher budget, the result will positive and for others, it will be negative

Lets make the problem a little more interesting and challenging

What if we want to subtract 1 col's mean from the other col mean?

What if we want to do the transformation of a column using some column's agrregate

Lets say, we want to filter the movies whose budget was even higher than the average revenue of the director from his other movies

How do we filter movies whose making were big risk to directors?

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

But we can use `transform` here as it expects only one column

We can use `apply()` function here

How can we do it ?

- We again need to group data acc to `director_name`
- Subtracting mean of `budget` from `revenue`
- To do so we would need to apply a custom function
- We can do so using the apply() method

In [56]:

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

Out[56]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender	risky
0	43597	237.00	150	2787.97	Avatar	7.2	11800	2009	Dec	Thursday	James Cameron	Male	False
1	43598	300.00	139	961.00	Pirates of the Caribbean: At World's End	6.9	4500	2007	May	Saturday	Gore Verbinski	Male	False
2	43599	245.00	107	880.67	Spectre	6.3	4466	2015	Oct	Monday	Sam Mendes	Male	False
3	43600	250.00	112	1084.94	The Dark Knight Rises	7.6	9106	2012	Jul	Monday	Christopher Nolan	Male	False
4	43602	258.00	115	890.87	Spider-Man 3	5.9	3576	2007	May	Tuesday	Sam Raimi	Male	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1460	48363	0.00	3	0.32	The Last Waltz	7.9	64	1978	May	Monday	Martin Scorsese	Male	False
1461	48370	0.03	19	3.15	Clerks	7.4	755	1994	Sep	Tuesday	Kevin Smith	Male	False
1462	48375	0.00	7	0.00	Rampage	6.0	131	2009	Aug	Friday	Uwe Boll	Male	False
1463	48376	0.00	3	0.00	Slacker	6.4	77	1990	Jul	Friday	Richard Linklater	Male	False
1464	48395	0.22	14	2.04	El Mariachi	6.6	238	1992	Sep	Friday	Robert Rodriguez	NaN	False

1465 rows × 13 columns

Lets see if there are any risky movies

In [81]:

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

Out[81]:

	id_x	budget	popularity	revenue	title	vote_average	vote_count	year	month	day	director_name	gender	risky
7	43608	200.0	107	586.09	Quantum of Solace	6.1	2965	2008	Oct	Thursday	Marc Forster	Male	True
12	43614	380.0	135	1045.71	Pirates of the Caribbean: On Stranger Tides	6.4	4948	2011	May	Saturday	Rob Marshall	Male	True
15	43618	200.0	37	310.67	Robin Hood	6.2	1398	2010	May	Wednesday	Ridley Scott	Male	True
20	43624	209.0	64	303.03	Battleship	5.5	2114	2012	Apr	Wednesday	Peter Berg	Male	True
24	43630	210.0	3	459.36	X-Men: The Last Stand	6.3	3525	2006	May	Wednesday	Brett Ratner	Male	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1347	47224	5.0	7	3.26	The Sweet Hereafter	6.8	103	1997	May	Wednesday	Atom Egoyan	Male	True
1349	47229	5.0	3	4.84	90 Minutes in Heaven	5.4	40	2015	Sep	Friday	Michael Polish	Male	True
1351	47233	5.0	6	0.00	Light Sleeper	5.7	15	1992	Aug	Friday	Paul Schrader	NaN	True
1356	47263	15.0	10	0.00	Dying of the Light	4.5	118	2014	Dec	Thursday	Paul Schrader	NaN	True
1383	47453	3.5	4	0.00	In the Name of the King III	3.3	19	2013	Dec	Friday	Uwe Boll	Male	True

131 rows × 13 columns

Yes, there are some 131 movies whose budget was even greater than average earnings of movies from the same director.

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

What does this mean ?

- The custom func will be apllied on each row if axis = 0 and on each col if axis = 1
- By default axis = 0

- Lets create a new dataframe to understand this

In [82]:

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

Out[82]:

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

All rows in both cols were added

In [83]:

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

Out[83]:

```
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
```

One row of `revenue_Mdollars` was added to same row of `budget_Mdollars`

Quiz-4

1. For each year , how you will get the average, lowest, and highest value of ratings of movies?

a. df.groupby('year').agg({'vote_average': ['mean', 'min', 'max']})

b. df.groupby['year'].agg({'vote_average': ['mean', 'min', 'max']})

c. df.groupby('year').agg(('vote_average': ['mean', 'min', 'max']))

Ans: a