495K Followers · About Follow

Pandas equivalent of 10 useful SQL queries

... or Pandas for SQL developers

Dorian Lazar Mar 29 · 11 min read ★

Image by PublicDomainPictures @ Pixabay

In case you don't know, pandas is a python library for data manipulation and analysis.

In particular, it offers data structures and operations for manipulating numerical tables and time series. The name is derived from the term “panel data”, an econometrics term for data sets that include observations over multiple time periods for the same individuals.

Basically, it is a way of working with tables in python. In pandas tables of data are called `DataFrame` **s.**

As the title suggests, in this article I'll show you the pandas equivalents of some of the most useful SQL queries. This can serve both as an introduction to pandas for those who already know SQL or as a cheat sheet of common pandas operations you may need.

For the examples below I will use this dataset which consists of data about trending YouTube videos in the US. It consists of 40949 rows with 16 columns: video_id, trending_date, title, channel_title, category_id, publish_time, tags, views, likes, dislikes, comment_count, thumbnail_link, comments_disabled, ratings_disabled, video_error_or_removed, description.

```
import numpy as np
import pandas as pd

# Reading the csv file into a DataFrame
df = pd.read_csv('USvideos.csv')
```

Pandas operations, by default, don’t modify the data frame which you are working with; they just return other data frames which you need to assign to a variable or use parameter inplace=True if you want to save the changes. For most examples below we don’t change our original data frame, we just show the returned result.

1. SELECT

```
SELECT col1, col2, ... FROM table
```

The SELECT statement is used to select columns of data from a table.

To do the same thing in pandas we just have to use the array notation on the data frame and inside the square brackets pass a list with the column names you want to select.

```
df[['video_id', 'title']]
```

	video_id	title
0	2kyS6SvSYSE	WE WANT TO TALK ABOUT OUR MARRIAGE
1	1ZAPwfrtAFY	The Trump Presidency: Last Week Tonight with J...
2	5qpjK5DgCt4	Racist Superman | Rudy Mancuso, King Bach & Le...
3	puqaWrEC7tY	Nickelback Lyrics: Real or Fake?
4	d380meD0W0M	I Dare You: GOING BALD!?
...	...	...
40944	BZt0qjTWNhw	The Cat Who Caught the Laser
40945	1h7KV2sjUWY	True Facts : Ant Mutualism
40946	D6Oy4LfoqsU	I GAVE SAFIYA NYGAARD A PERFECT HAIR MAKEOVER ...
40947	oV0zkMe1K8s	How Black Panther Should Have Ended
40948	ooyjaVdt-jA	Official Call of Duty®: Black Ops 4 — Multipla...

40949 rows × 2 columns

The same thing can be made with the following syntax which makes easier to translate WHERE statements later:

```
df.loc[:, ['video_id', 'title']]
```

```
SELECT DISTINCT col1, col2, ... FROM table
```

The SELECT DISTINCT statement returns only unique rows form a table.

In a data frame there may be duplicate values. If you want to get only distinct rows (remove duplicates) it is as simple as calling the `.drop_duplicates()` method. Judging based on this method's name you may think that it removes duplicate rows from your initial data frame, but what it actually does is to return a new data frame with duplicate rows removed.

```
df.loc[:, ['channel_title']].drop_duplicates()
```

	channel_title
0	CaseyNeistat
1	LastWeekTonight
2	Rudy Mancuso
3	Good Mythical Morning
4	nigahiga
...	...
40366	HALO
40475	Ben Kronengold
40488	All Def Digital
40540	How It Should Have Ended
40554	Ubisoft North America

2207 rows × 1 columns

```
SELECT TOP number col1, col2, ... FROM table
```

or

```
SELECT col1, col2, ... FROM table LIMIT number
```

The TOP or LIMIT keyword in SQL is used to limit the number of returned rows from the top of the table.

In pandas this is very easy to do with `.head(number)` method. Pandas also has the `.tail(number)` method for showing the rows from the end of data frame.

```
df.loc[:, ['video_id', 'title']].head(5)
```

	video_id	title
0	2kyS6SvSYSE	WE WANT TO TALK ABOUT OUR MARRIAGE
1	1ZAPwfrtAFY	The Trump Presidency: Last Week Tonight with J...
2	5qpjK5DgCt4	Racist Superman | Rudy Mancuso, King Bach & Le...
3	puqaWrEC7tY	Nickelback Lyrics: Real or Fake?
4	d380meD0W0M	I Dare You: GOING BALD!?

```
df.loc[:, ['video_id', 'title']].tail(5)
```

	video_id	title
40944	BZt0qjTWNhw	The Cat Who Caught the Laser
40945	1h7KV2sjUWY	True Facts : Ant Mutualism
40946	D6Oy4LfoqsU	I GAVE SAFIYA NYGAARD A PERFECT HAIR MAKEOVER ...
40947	oV0zkMe1K8s	How Black Panther Should Have Ended
40948	ooyjaVdt-jA	Official Call of Duty®: Black Ops 4 — Multipla...

SQL's MIN(), MAX(), COUNT(), AVG(), and SUM() functions are pretty straightforward to translate to pandas:

```
SELECT MIN(col) FROM table
```

```
df.loc[:, ['views']].min()
```

	MIN(views)
0	549

```
SELECT MAX(col) FROM table
```

```
  df.loc[:, ['views']].max()
```

	MAX(views)
0	225211923

```
SELECT COUNT(col) FROM table
```

```
  df.loc[:, ['views']].count()
```

	COUNT(views)
0	40949

```
SELECT AVG(col) FROM table
```

```
  df.loc[:, ['views']].mean()
```

	AVG(views)
0	2.360785e+06

```
SELECT SUM(col) FROM table
```

```
df.loc[:, ['views']].sum()
```

	SUM(views)
0	96671770152

Now, what if we want to do something like this:
`SELECT MAX(likes), MIN(dislikes) FROM table` ? We need to do this in more steps:

```
new_df = df.loc[:, ['likes']].max().rename({'likes': 'MAX(likes)'})
new_df['MIN(dislikes)'] = df.loc[:, ['dislikes']].min().values[0]
```

	MAX(likes)	MIN(dislikes)
0	5613827	0

2. WHERE

```
SELECT col1, col2, ... FROM table WHERE condition
```

The WHERE clause is used to extract only the rows that fulfill a specified condition.

Recall the syntax we used so far for selecting columns:

```
df.loc[:, ['col1', 'col2']]
```

Inside the square brackets of `.loc` there is place for two parameters; so far we only used the second one which is used to specify what columns you want to select. Guess for what is the first parameter? Is for selecting rows. Pandas data frames expect a list of row indices or boolean flags based on which it extracts the rows we need. So far we used only the `:` symbol which means "return all rows". If we want to extract only rows with indices from 50 to 80 we can use `50:80` in that place.

For extracting rows based on some condition, most often we will pass there an array of boolean flags returned by some (vectorized) boolean operation. The rows on positions where we will have False will not be included in the result, only those rows with True on their positions will be returned.

Using equality and inequality operators ==, <, <=, >, >=, != in conditions is straightforward. For example, to return only rows that have number of likes >= 1000000 we can use:

```
df.loc[df['likes'] >= 1000000, ['video_id', 'title']]
```

	video_id	title
70	2Vv-BfVoq4g	Ed Sheeran - Perfect (Official Music Video)
336	2Vv-BfVoq4g	Ed Sheeran - Perfect (Official Music Video)
574	2Vv-BfVoq4g	Ed Sheeran - Perfect (Official Music Video)
801	TyHvyGVs42U	Luis Fonsi, Demi Lovato - Échame La Culpa
1005	TyHvyGVs42U	Luis Fonsi, Demi Lovato - Échame La Culpa
...	...	...
40823	VY1eFxgRR-k	Selena Gomez - Back To You
40824	J2HytHu5VBI	we broke up
40855	D_6QmL6rExk	BTS (방탄소년단) 'FAKE LOVE' Official MV (Extended ...
40869	xTlNMmZKwpA	Cardi B, Bad Bunny & J Balvin - I Like It [Off...
40896	nQySbNGu4g0	[CHOREOGRAPHY] BTS (방탄소년단) 'FAKE LOVE' Dance P...

369 rows × 2 columns

Note that the reason for which we could do what we did above (`df['likes'] >= 1000000`) is that pandas has overwritten the default behavior for >= operator so that it applies the operator element-wise and returns an array of booleans of the shape that we need (number of rows).
But the operators **and, or, not** don't work like that. So, we will use **&** instead of **and**, **|** instead of **or**, **~** instead of **not**.

```
df.loc[(df['likes'] >= 1000000) & (df['dislikes'] <= 5000),
['video_id', 'title']].drop_duplicates()
```

	video_id	title
1278	ixxR3ZoqnF0	BTS (방탄소년단) 'MIC Drop (Steve Aoki Remix)' Offi...
14755	J41qe-TM1DY	JONGHYUN 종현 '빛이 나 (Shinin')' MV
20823	Vm4Xn5vjqRA	CRISTIANO RONALDO E FRED, O GRANDE ENCONTRO
35156	2tDKp41nrw8	BTS (방탄소년단) 'FAKE LOVE' Official Teaser 2
37987	nQySbNGu4g0	[CHOREOGRAPHY] BTS (방탄소년단) 'FAKE LOVE' Dance P...

```
SELECT col1, col2, ... FROM table WHERE colN IS NOT NULL
```

In SQL you can use `IS NULL` or `IS NOT NULL` to get rows that contain/don't contain null values.

How to check for null values in pandas?
We will use `isnull(array-like)` function from pandas package to do that. Note that this is not a method of data frame objects, don't use `df.isnull(...)` ; instead do `pd.isnull(df['column'])` . So be careful.

The example below returns all rows where the description is not null.

	video_id	title
0	2kyS6SvSYSE	WE WANT TO TALK ABOUT OUR MARRIAGE
1	1ZAPwfrtAFY	The Trump Presidency: Last Week Tonight with J...
2	5qpjK5DgCt4	Racist Superman | Rudy Mancuso, King Bach & Le...
3	puqaWrEC7tY	Nickelback Lyrics: Real or Fake?
4	d380meD0W0M	I Dare You: GOING BALD!?
...	...	...
40760	dS5Thrl-4Kc	CRAYOLA MAKEUP | HIT OR MISS?
40761	JGm9Y_hFqNk	First Take reacts: Kyrie Irving says contract ...
40762	6h8QgZF5Qu4	Drop the Mic w/ Ashton Kutcher & Sean Diddy Combs
40764	mpnshdmtE2Y	Carla Makes BA Smashburgers | From the Test Ki...
40766	yz7Xq3T0YPs	Katherine Langford on 13 Reasons Why, Australi...

6365 rows × 2 columns

```
SELECT col1, col2, ... FROM table WHERE colN LIKE pattern
```

The LIKE keyword can be used in a WHERE clause to test if a column matches a pattern.

In pandas we can use python's native **re** module for regular expressions to accomplish the same thing, or even more as the python's re module allows for a richer set of patterns to be tested rather than SQL's LIKE.

We will create a function `like(x, pattern)` where x is an array-like object and pattern is a string containing the pattern which we want to test for. This function will first compile the pattern into a regular expression object, then we can use the `.fullmatch(val)` method to test the `val` 's value against our pattern. In order to apply this test to each element in our x vector we will use numpy's `vectorize(func)` function to create a vector equivalent for our operation of regex matching. Finally we apply this vectorized function to our x input vector. Then all we need to do is to pass `like(df['column'], pattern)` as first parameter in `.loc[]` .

```
import re

def like(x, pattern):
    r = re.compile(pattern)
    vlike = np.vectorize(lambda val: bool(r.fullmatch(val)))
    return vlike(x)
```

As an example the below code returns all videos that contains the word 'math' in their description.

```
df_notnull = df.loc[~pd.isnull(df['description']), :]
df_notnull.loc[like(df_notnull['description'], '.* math .*'),
['video_id', 'title']].drop_duplicates()
```

	video_id	title
1905	WKcYTvyoidw	Parenting Habits That Could Keep Children From...
26016	DjjBg8hTaPo	How Much Food Is There On Earth?

3. ORDER BY

```
SELECT col1, col2, ... FROM table ORDER BY col1, col2 ASC|DESC
```

This SQL keyword is used to sort the results in ascending or descending order.

It is straightforward to translate this to pandas, you just call the `.sort_values(by=['col1', ...], ascending=True/False)` method on a data frame.

```
df.loc[df['likes'] >= 2000000, ['video_id', 'title']
].sort_values(by=['title'], ascending=True).drop_duplicates()
```

	video_id	title
34708	ffxKSjUwKdU	Ariana Grande - No Tears Left To Cry
28879	kX0vO4vlJuU	BTS (방탄소년단) 'Euphoria : Theme of LOVE YOURSELF...
35976	7C2z4GqqS5E	BTS (방탄소년단) 'FAKE LOVE' Official MV
2201	kTlv5_Bs8aw	BTS (방탄소년단) 'MIC Drop (Steve Aoki Remix)' Offi...
34444	p8npDG2ulKQ	BTS (방탄소년단) LOVE YOURSELF 轉 Tear 'Singularity'...
36505	VYOjWnS4cMY	Childish Gambino - This Is America (Official V...
1950	TyHvyGVs42U	Luis Fonsi, Demi Lovato - Échame La Culpa
39814	aJOTlE1K90k	Maroon 5 - Girls Like You ft. Cardi B
3400	6ZfuNTqbHE8	Marvel Studios' Avengers: Infinity War Officia...
5236	FlsCjmMhFmw	YouTube Rewind: The Shape of 2017 | #YouTubeRe...
22040	OK3GJ0WIQ8s	j-hope 'Daydream (백일몽)' MV
40412	J2HytHu5VBI	we broke up

4. GROUP BY

```
SELECT col1, col2, ... FROM table GROUP BY colN
```

The GROUP BY statement groups rows that have the same value for a specific column. It is often used with aggregate functions (MIN, MAX, COUNT, SUM, AVG).

In pandas it is as simple as calling the `.groupby(['col1', ...])` method, followed by a call to one of `.min()`, `.max()`, `.count()`, `.sum`, `.mean()` methods.

```
df.loc[:, ['channel_title', 'views', 'likes', 'dislikes']
].groupby(['channel_title']).sum()
```

channel_title	views	likes	dislikes
12 News	177970	352	90
1MILLION Dance Studio	20959169	1399898	15030
1theK (원더케이)	57375949	3663362	63329
20th Century Fox	1082872611	24419452	488761
2CELLOS	432186	22900	245
...	...	...	...
ワーナー ブラザース 公式チャンネル	7389323	154962	6128
圧倒的不審者の極み!	11417717	157447	12315
杰威爾音樂 JVR Music	400530463	3400678	229439
郭韋辰	26964	99	2
영국남자 Korean Englishman	5963784	147154	2579

2207 rows × 3 columns

5. HAVING

```
SELECT col1, col2, ... FROM table GROUP BY colN HAVING condition
```

The HAVING keyword is used to filter the results based on group-level conditions.

In pandas we have the `.filter(func)` method that can be called after a `groupby()` call. We need to pass to this method a function that takes a data frame of a group as a parameter and returns a boolean value that decides whether this group is included in the results or not.
But if we want to do more things at once in pandas, e.g. apply aggregate functions on columns and filter results based on group-level conditions, we need to do this in more steps. Whereas in SQL we could have done this in only one query.

In the example below we want to group by *channel_title*, allow only channels that have

at least 100 videos in the table, and apply average function on *views*, *likes*, and *dislikes*.

In SQL this would be:

```
SELECT channel_title, AVG(views), AVG(likes), AVG(dislikes)
FROM videos_table
GROUP BY channel_title
HAVING COUNT(video_id) >= 100;
```

In pandas:

```
g = df.groupby(['channel_title'])
g = g.filter(lambda x: x['video_id'].count() > 100)
g = g.loc[:, ['channel_title', 'views', 'likes', 'dislikes']]
g = g.groupby(['channel_title']).mean()
```

channel_title	views	likes	dislikes
20th Century Fox	8.021279e+06	180884.829630	3620.451852
ABC News	4.540516e+05	3627.186992	2276.097561
AsapSCIENCE	5.162413e+06	86787.370968	5019.612903
Binging with Babish	1.074592e+06	33679.839286	820.625000
Bon Appétit	8.502353e+05	22298.470199	712.410596
...	...	...	...
WWE	2.789609e+06	42768.108280	2400.515924
Warner Bros. Pictures	4.434285e+06	39682.760000	5070.613333
Washington Post	1.604080e+06	13914.634783	7038.773913
You Suck At Cooking	7.727879e+05	35612.719626	1015.813084
jacksfilms	1.348708e+06	94536.297297	2718.081081

69 rows × 3 columns

6. INSERT

```
INSERT INTO table (column1, column2, ...) VALUES (value1, value2, ...)
```

This SQL statement is used to insert new rows in the table.

In pandas we can use the `.append()` method to append a new data frame at the end of an existing one. We will use `ignore_index=True` in order to continue indexing from the last row in the old data frame.

```
new_row = pd.DataFrame({'video_id': ['EkZGBdY0vlg'],
                        'channel_title': ['Professor Leonard'],
                        'title': ['Calculus 3 Lecture 13.3: Partial
Derivatives']})
df = df.append(new_row, ignore_index=True)
```

40947	oV0zkMe1K8s	18.14.06	How Black Panther Should Have Ended	How It Should Have Ended	1.0	2018-05-17T17:00:04.000Z
40948	ooyjaVdt-jA	18.14.06	Official Call of Duty®: Black Ops 4 — Multipla...	Call of Duty	20.0	2018-05-17T17:09:38.000Z
40949	EkZGBdY0vlg	NaN	Calculus 3 Lecture 13.3: Partial Derivatives	Professor Leonard	NaN	NaN

40950 rows × 16 columns

7. DELETE

```
DELETE FROM table WHERE condition
```

DELETE statement is used to delete existing rows from a table based on some condition.

In pandas we can use `.drop()` method to remove the rows whose indices we pass in. Unlike other methods this one doesn't accept boolean arrays as input. So we must convert our condition's output to indices. We can do that with `np.where()` function.

In the example below we deleted all the rows where *channel_title != '3Blue1Brown'*.

```
df = df.drop(np.where(df['channel_title'] != '3Blue1Brown')[0])
```

	video_id	trending_date	title	channel_title	category_id	publish_time	tags
15541	spUNpyF58BY	18.01.02	But what is the Fourier Transform? A visual i...	3Blue1Brown	27.0	2018-01-26T18:47:48.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
23992	bcPTiiiYDs8	18.16.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
24206	bcPTiiiYDs8	18.17.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
24420	bcPTiiiYDs8	18.18.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
24633	bcPTiiiYDs8	18.19.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
24853	bcPTiiiYDs8	18.20.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
25070	bcPTiiiYDs8	18.21.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
25281	bcPTiiiYDs8	18.22.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
25494	bcPTiiiYDs8	18.23.03	How pi was almost 6.283185...	3Blue1Brown	27.0	2018-03-14T13:05:32.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...
26630	b7FxPsqfkOY	18.29.03	How to solve 2D equations using color	3Blue1Brown	27.0	2018-03-24T14:42:51.000Z	Mathematics|"three blue one brown"|"3 blue 1 b...

8. ALTER

```
ALTER TABLE table ADD column
```

This SQL statement adds new columns.

In pandas we can do this by: `df['new_column'] = array-like`.

Below we add a new column 'like_ratio':

```
df['like_ratio'] = df['likes'] / (df['likes'] + df['dislikes'])
```

ratings_disabled	video_error_or_removed	description	like_ratio
False	False	SHANTELL'S CHANNEL - https://www.youtube.com/s...	0.950970
False	False	One year after the presidential election, John...	0.940521
False	False	WATCH MY PREVIOUS VIDEO ▶ \n\nSUBSCRIBE ▶ http...	0.964729
False	False	Today we find out if Link is a Nickelback amat...	0.938550
False	False	I know it's been a while since we did this sho...	0.985181
...	...	...	...
False	False	NaN	0.993674

```
ALTER TABLE table DROP COLUMN column
```

This SQL statement deletes a column.

`del df['column']` is how we do this in pandas.

For example, deleting 'comments_disabled' column would be:

```
del df['comments_disabled']
```

9. UPDATE

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

The UPDATE statement is used to change values in our table based on some condition.

For doing this in python we can use numpy's `where()` function. We also saw this function a few lines above when we used it to convert a boolean array to indices array. That is what this function does when given just one parameter. This function can receive 3 arrays of the same size as parameters, first one being a boolean array. Let's call them c, x, y. It returns an array of the same size filled with elements from x and y chosen in this way: if c[i] is true choose x[i] else choose y[i].
To modify a data frame column we can do:

```
df['column'] = np.where(condition, new_values, df['column'])
```

In the example below we increase the number of likes by 100 where channel_title == 'Veritasium'.

This is how the data looks before:

```
df.loc[df['channel_title'] == 'Veritasium', ['title', 'likes']]
```

Increment likes by 100 for Veritasium channel:

```
df['likes'] = np.where(df['channel_title'] == 'Veritasium',
df['likes']+100, df['likes'])
```

After we ran the above query:

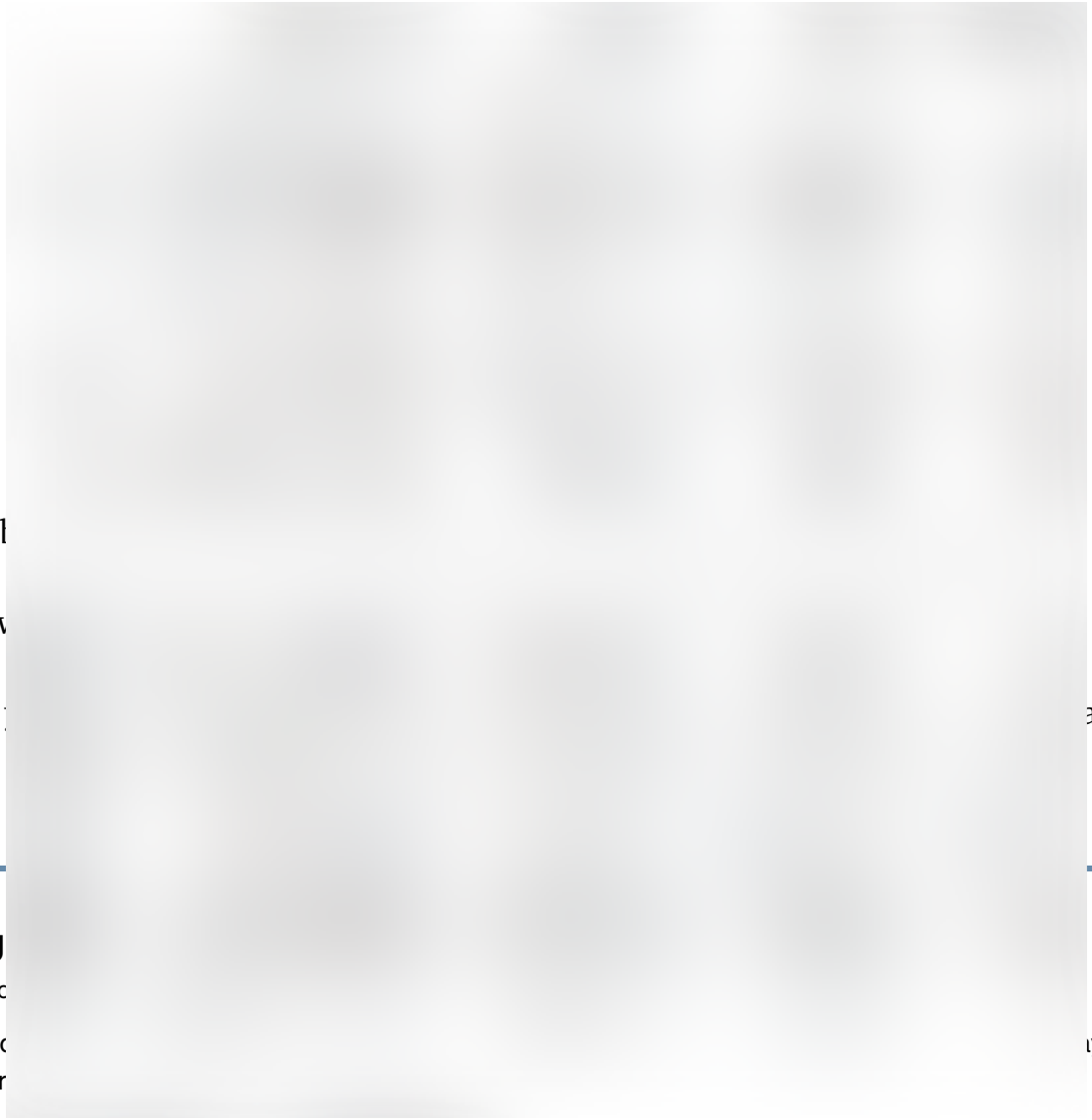
10. JOIN

A JOIN clause is used to combine rows from two or more tables based on a related column between them.

In order to show examples of joins I need at least two tables, so I will split the data frame used so far into two smaller tables.

```
df_titles = df.loc[:, ['video_id', 'title']].drop_duplicates()
```

```
df_stats = df.loc[:, ['video_id', 'views', 'likes', 'dislikes']
].groupby('video_id').max()
# transform video_id from index to column
df_stats = df_stats.reset_index()
```



A notel

If you v

I hope article please

- INNER JOIN: returns rows that have matching values in both tables
- FULL (OUTER) JOIN: returns rows that have matching values in any of the tables
- LEFT JOIN: returns all rows from the left table, and the matched rows from the right one

- RIGHT JOIN: returns all rows from the right table, and the matched rows from the left one

Images from w3schools

To specify which type of join you want in pandas you can use the **how** parameter in `.join()` method. This parameter can be one of: 'inner', 'outer', 'left', 'right'.

Below is an example of inner join of the two data frames above on 'video_id' column. The other types of joins are done in the same way, just change the 'how' parameter accordingly.

Inner join in SQL:

```
SELECT *
FROM df_titles
INNER JOIN df_stats
ON df_titles.video_id = df_stats.video_id;
```

Inner join in pandas:

```
df_titles.join(df_stats.set_index('video_id'), on='video_id', how='inner')
```