Member-only story

30 SQL Queries Explained Through Their Pandas Equivalents

SQL made much easier for people who love Pandas

Bex T. · Follow

Published in Towards Data Science

10 min read · Jun 9

Image by me with Midjourney

Motivation

In a world dominated by SQL since 1974, along came Pandas in 2008, offering attractive features like built-in visualization and flexible data handling. It quickly became the go-to tool for data exploration, overshadowing SQL.

But don't be fooled, SQL still holds its ground. It's the second most in-demand and third fastest-growing language for data science (see here). So, while Pandas steals the spotlight, SQL remains a vital skill for any data scientist.

Let's find out how easy it is to learn SQL when you already know Pandas.

Connecting to a Database

Setting up an SQL workspace and connecting to a sample database can be a real pain. First, you need to install your favorite SQL flavor (PostgreSQL, MySQL, etc.) and download an SQL IDE. Doing those here would draw us away the article's purpose, so we will use a shortcut.

Specifically, we will directly run SQL queries in a Jupyter Notebook without additional steps. All we need to do is install the `ipython-sql` package using pip:

```
pip install ipython-sql
```

After the installation, start a new Jupyter session and run this command in the notebook:

```
%load_ext sql
```

and you are all set!

To illustrate how basic SQL statements work, we will be using the Chinook SQLite database, which has 11 tables.

To load the dataset and its 11 tables as separate variables into our environment, we can run:

```
%sql sqlite:///data/chinook.db
```

The statement starts with `%sql` in-line magic command that tells the notebook interpreter we will be running SQL commands. It is followed by the path that the downloaded Chinook database is in.

The valid paths should always start with `sqlite:///` prefix for SQLite databases. Above, we are connecting to the database stored in the 'data' folder of the current directory. If you want to pass an absolute path, the prefix should start with four forward slashes - `sqlite:////`

If you wish to connect to a different database flavor, you can refer to this excellent article.

Taking a First Look at the Tables

The first thing we always do in Pandas is to use the `.head()` function to take a first look at the data. Let's learn how to do that in SQL:

```
1  %%sql
2
3  SELECT * FROM customers
4  LIMIT 5
```

6801.sql hosted with ❤ by GitHub view raw

CustomerId	FirstName	LastName	Company	Address	City	State	Country	PostalCode	Phone	Fax
1	Luís	Gonçalves	Embraer - Empresa Brasileira de Aeronáutica S.A.	Av. Brigadeiro Faria Lima, 2170	São José dos Campos	SP	Brazil	12227-000	+55 (12) 3923-5555	+55 (12) 3923-5566
2	Leonie	Köhler	None	Theodor-Heuss-Straße 34	Stuttgart	None	Germany	70174	+49 0711 2842222	None
3	François	Tremblay	None	1498 rue Bélanger	Montréal	QC	Canada	H2G 1A7	+1 (514) 721-4711	None
4	Bjørn	Hansen	None	Ullevålsveien 14	Oslo	None	Norway	0171	+47 22 44 22 22	None
5	František	Wichterlová	JetBrains s.r.o.	Klanova 9/506	Prague	None	Czech Republic	14700	+420 2 4172 5555	+420 2 4172 5555

The dataset is licensed for commercial use as well.

The first keyword in the above query is `SELECT`. It is equivalent to the brackets operator in Pandas, where we select specific columns. But, the SELECT keyword is followed by a * (asterisk). * is an SQL operator that selects everything (all rows and columns) from a table specified after the `FROM` keyword. LIMIT is used to minimize the output returned. So, the above query is equivalent to `df.head()` function.

If you don't want to select all columns, you can specify one or more column names after the SELECT keyword:

```
%%sql

SELECT Name, Composer, UnitPrice
FROM tracks
LIMIT 10
```

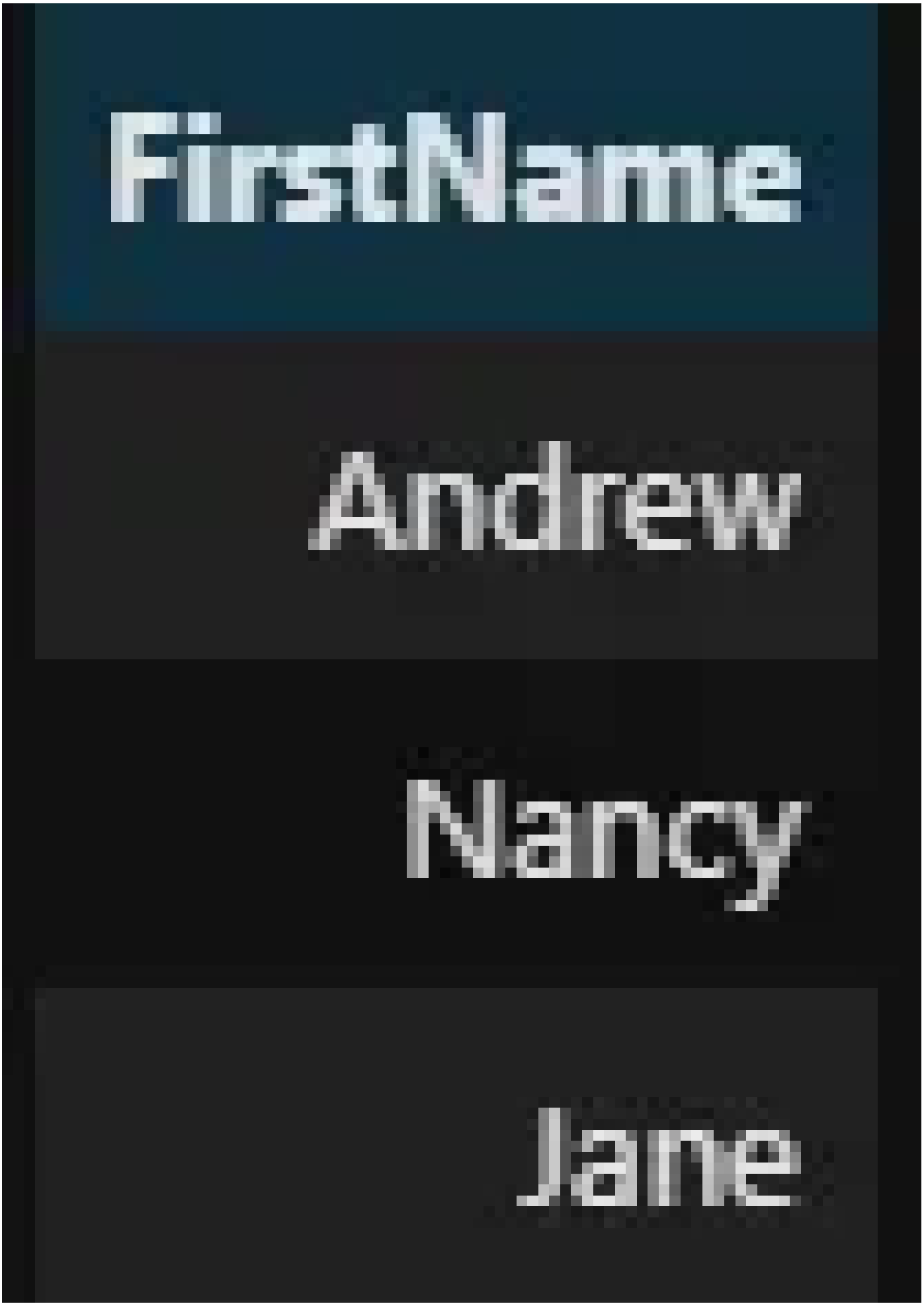
6802.sql hosted with ❤️ by GitHub view raw

Name	Composer	UnitPrice
For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	0.99
Balls to the Wall	None	0.99
Fast As a Shark	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	0.99
Restless and Wild	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	0.99
Princess of the Dawn	Deaffy & R.A. Smith-Diesel	0.99
Put The Finger On You	Angus Young, Malcolm Young, Brian Johnson	0.99
Let's Get It Up	Angus Young, Malcolm Young, Brian Johnson	0.99
Inject The Venom	Angus Young, Malcolm Young, Brian Johnson	0.99
Snowballed	Angus Young, Malcolm Young, Brian Johnson	0.99
Evil Walks	Angus Young, Malcolm Young, Brian Johnson	0.99

The equivalent Pandas operation is:

```
tracks[['Name', 'Composer', 'UnitPrice']].head(10)
```

Another useful keyword in SQL is `DISTINCT`. Adding this keyword before any column name returns its unique values:


FirstName
Andrew
Nancy
Jane


Margaret
Steve
Michael
Robert

Comments in SQL are written with double dashes.

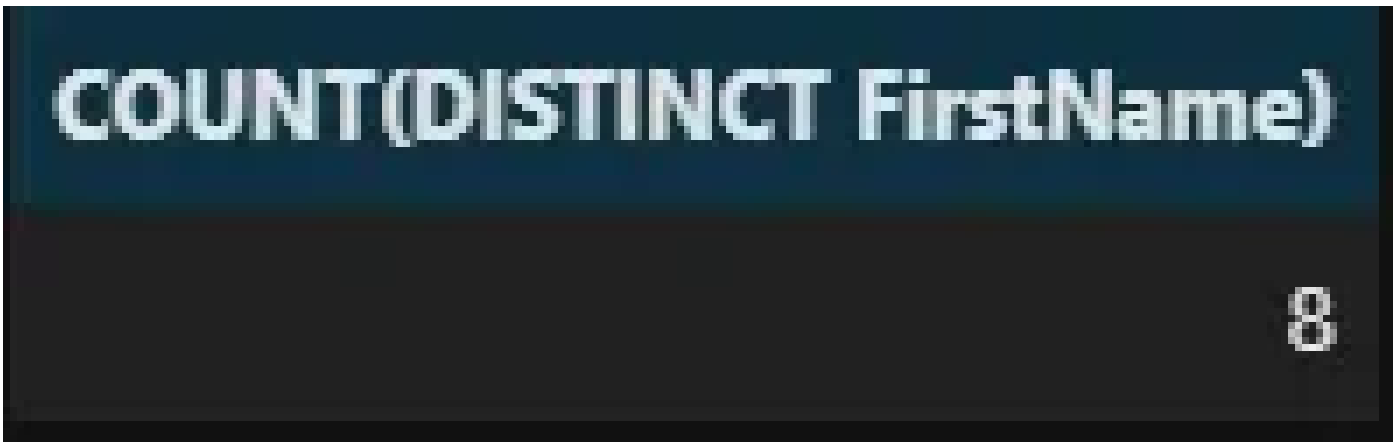
Counting the Number of Rows

Just like Pandas has `.shape` attribute on its DataFrames, SQL has a `COUNT` function to display the number of rows in a table:

```
%%sql

SELECT COUNT(*) FROM tracks
```

More helpful info would be counting the number of unique values in a particular column. We can do this by adding the DISTINCT keyword into COUNT:


COUNT(DISTINCT FirstName)
8

Filtering Results With WHERE Clauses

Just looking and counting rows is pretty lame. Let's see how we can filter rows based on conditions.

First, let's look at the songs which cost more than a dollar:

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
2819	Battlestar Galactica: The Story So Far	226	3	18	None	2622250	490750393	1.99
2820	Occupation / Precipice	227	3	19	None	5286953	1054423946	1.99
2821	Exodus, Pt. 1	227	3	19	None	2621708	475079441	1.99
2822	Exodus, Pt. 2	227	3	19	None	2618000	466820021	1.99
2823	Collaborators	227	3	19	None	2626626	483484911	1.99
2824	Torn	227	3	19	None	2631291	495262585	1.99
2825	A Measure of Salvation	227	3	18	None	2563938	489715554	1.99
2826	Hero	227	3	18	None	2713755	506896959	1.99
2827	Unfinished Business	227	3	18	None	2622038	528499160	1.99
2828	The Passage	227	3	18	None	2623875	490375760	1.99

Songs that cost more than a dollar.

Conditional statements are written in the WHERE clause, which always comes after FROM and before the LIMIT keywords. Using conditionals is pretty similar to how we do it in Pandas but I dare say the SQL version is more readable.

You can also use the COUNT function when using conditionals. For example, let's see the number of songs that are priced between 1 and 10 dollars:

As you can see, the SQL version is much more readable.

The number of songs that cost between 1 and 10 dollars.

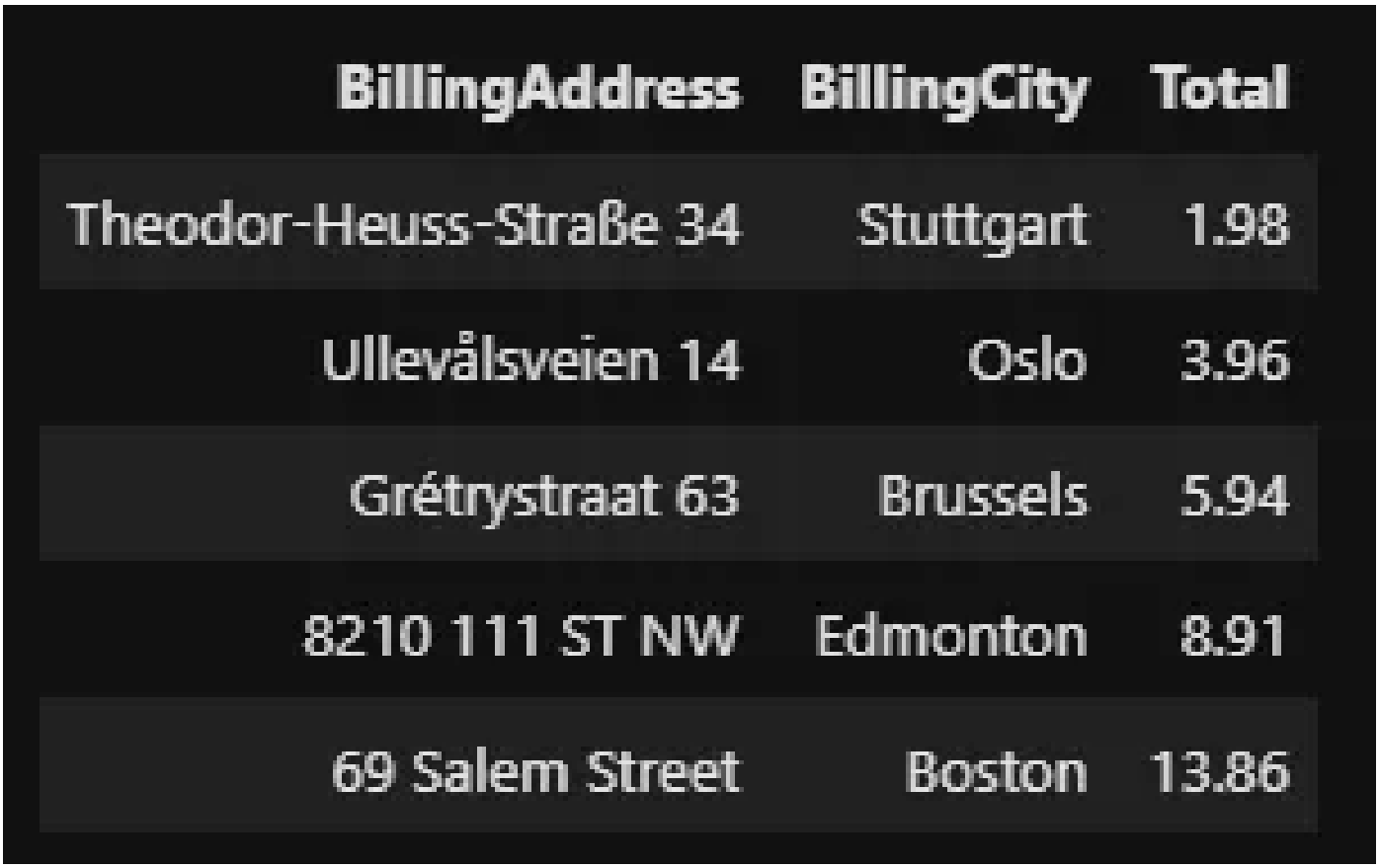
Above we chained two conditions with the boolean operator AND. Other boolean operators (OR, NOT) are used similarly.

Now, let's see all the invoices that have Paris *or* Berlin as a billing city:

BillingAddress	BillingCity	Total
Barbarossastraße 19	Berlin	1.98
8, Rue Hanovre	Paris	1.98
8, Rue Hanovre	Paris	13.86
Tauentzienstraße 8	Berlin	1.98
Barbarossastraße 19	Berlin	3.96

The invoices that are billed from either Berlin or Paris

The **equality** operator in SQL requires only one '=' (equal) sign. The **inequality** operator is represented with either '!=' or '<>' operators:



BillingAddress	BillingCity	Total
Theodor-Heuss-Straße 34	Stuttgart	1.98
Ullevålsveien 14	Oslo	3.96
Grétrystraat 63	Brussels	5.94
8210 111 ST NW	Edmonton	8.91
69 Salem Street	Boston	13.86

The invoices where the billing city isn't Berlin or Paris

Easier Filtering With BETWEEN And IN

Similar conditionals are used very often, and writing them out with simple booleans becomes cumbersome. For example, Pandas has `.isin()` function which checks if a value belongs to a list of values.

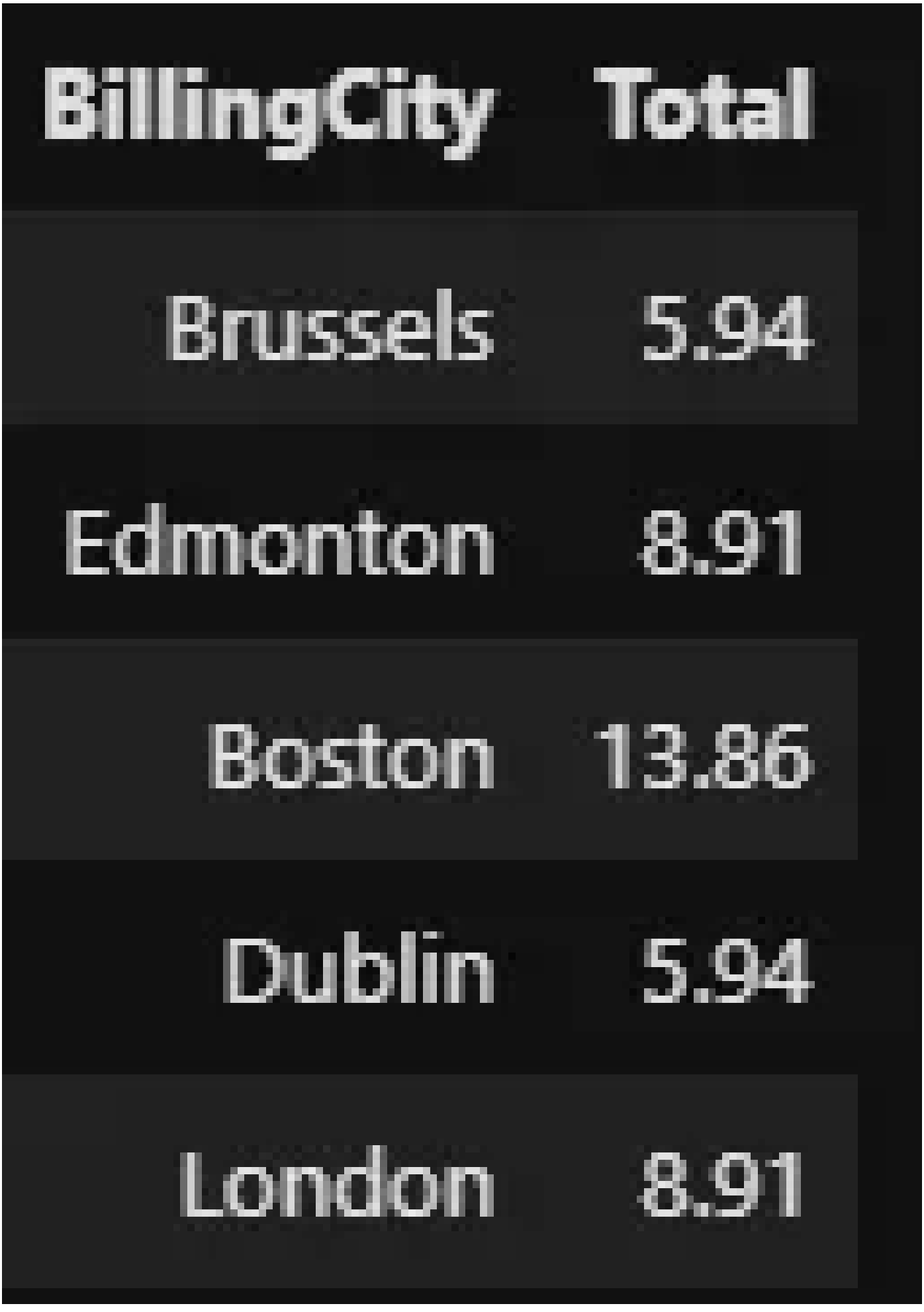
If we wanted to select invoices for five cities, we would have to write five chained conditions. Luckily, SQL supports a similar IN operator like `.isin()` so we don't have to:

InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
5	23	2009-01-11 00:00:00	69 Salem Street	Boston	MA	USA	2113	13.86
7	38	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	None	Germany	10779	1.98
8	40	2009-02-01 00:00:00	8, Rue Hanovre	Paris	None	France	75002	1.98
11	52	2009-02-06 00:00:00	202 Hoxton Street	London	None	United Kingdom	N1 5LH	8.91
19	40	2009-03-14 00:00:00	8, Rue Hanovre	Paris	None	France	75002	13.86

The list of values after IN should be given as a tuple, not as a list. You can also negate the condition with the NOT keyword:

InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
1	2	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	None	Germany	70174	1.98
2	4	2009-01-02 00:00:00	Ullevålsveien 14	Oslo	None	Norway	0171	3.96
3	8	2009-01-03 00:00:00	Grétrystraat 63	Brussels	None	Belgium	1000	5.94
4	14	2009-01-06 00:00:00	8210 111 ST NW	Edmonton	AB	Canada	T6G 2C7	8.91
6	37	2009-01-19 00:00:00	Berger Straße 10	Frankfurt	None	Germany	60316	0.99

Another common filtering operation on numeric columns is to select values within a range. For this, BETWEEN keyword can be used, which is equivalent to `pd.Series.between()`:


BillingCity Total
Brussels 5.94
Edmonton 8.91
Boston 13.86
Dublin 5.94
London 8.91

Choosing the invoices with billing amount between 5 and 15.

Checking For Nulls

Every data source has missing values, and databases are no exception. Just like there are several ways to explore the missing values in Pandas, there are specific keywords that check the existence of null values in SQL. The below query counts the number of rows with missing values in BillingState:

You can add the NOT keyword between IS and NULL to discard missing values of a particular column:

InvoiceDate	BillingCountry
2009-01-01 00:00:00	Germany
2009-01-02 00:00:00	Norway
2009-01-03 00:00:00	Belgium
2009-01-06 00:00:00	Canada
2009-01-11 00:00:00	USA
2009-01-19 00:00:00	Germany
2009-02-01 00:00:00	Germany
2009-02-01 00:00:00	France
2009-02-02 00:00:00	France
2009-02-03 00:00:00	Ireland

Better String Matching With LIKE

In the WHERE clause, we filtered columns based on exact text values. But often, we may want to filter textual columns based on a pattern. In Pandas and pure Python, we would use regular expressions for pattern matching, which are very powerful but regex requires time to master.

As an alternative, SQL offers a '%' wildcard as a placeholder to match any character 0 or more times. For example, the 'gr%' string matches' great,' 'groom,' 'greed,' and '%ex%' matches any text with 'ex' in the middle, etc. Let's see how to use it with SQL:

Name	Composer	UnitPrice
Balls to the Wall	None	0.99
Breaking The Rules	Angus Young, Malcolm Young, Brian Johnson	0.99
Bad Boy Boogie	AC/DC	0.99
Blind Man	Steven Tyler, Joe Perry, Taylor Rhodes	0.99
Bleed The Freak	Jerry Cantrell	0.99

The above query finds all songs that start with ‘B.’ The string that contains the wildcard should come after the LIKE keyword.

Now, let’s find all songs that contain the word ‘beautiful’ in their titles:

Name	Composer	UnitPrice
So Beautiful	Mick Hucknall	0.99
Beautiful Day	Adam Clayton, Bono, Larry Mullen, The Edge	0.99
Beautiful Boy	None	0.99
On the Beautiful Blue Danube	Johann Strauss II	0.99

You can also use other boolean operators next to LIKE:

Name	Composer	UnitPrice
Fallout	None	1.99
Further Instructions	None	1.99
Flashes Before Your Eyes	None	1.99
Fire + Water	None	1.99
Five Years Gone	None	1.99
Fire In Space	None	1.99

There are many other wildcards in SQL that have different functionalities. You can see the complete list and their usage here.

Aggregate Functions in SQL

It is also possible to perform basic arithmetic operations on columns. These operations are called aggregate functions in SQL, and the most common ones are `AVG`, `SUM`, `MIN`, `MAX` . Their functionality should be clear from their names:

SUM(Total)	MAX(Total)	MIN(Total)	AVG(Total)
2328.600000000004	25.86	0.99	5.651941747572825

Aggregate functions give only a single result for the column you used them on. This means you cannot aggregate across one column and select other unaggregated columns:

AVG(Total)	BillingCity	BillingAddress
5.651941747572825	Stuttgart	Theodor-Heuss-Straße 34

You can combine aggregate functions with conditionals using WHERE clauses just as easily:


AVG(Total) BillingCity
5.517142857142857 Paris

It is also possible to use arithmetic operators such as +, -, *, / on columns, and simple numbers. When used on columns, the operation is performed element-wise:

One thing to note about arithmetic operations: if you perform operations on integers only, SQL thinks that you are expecting an integer as the answer:

```
%%sql
```

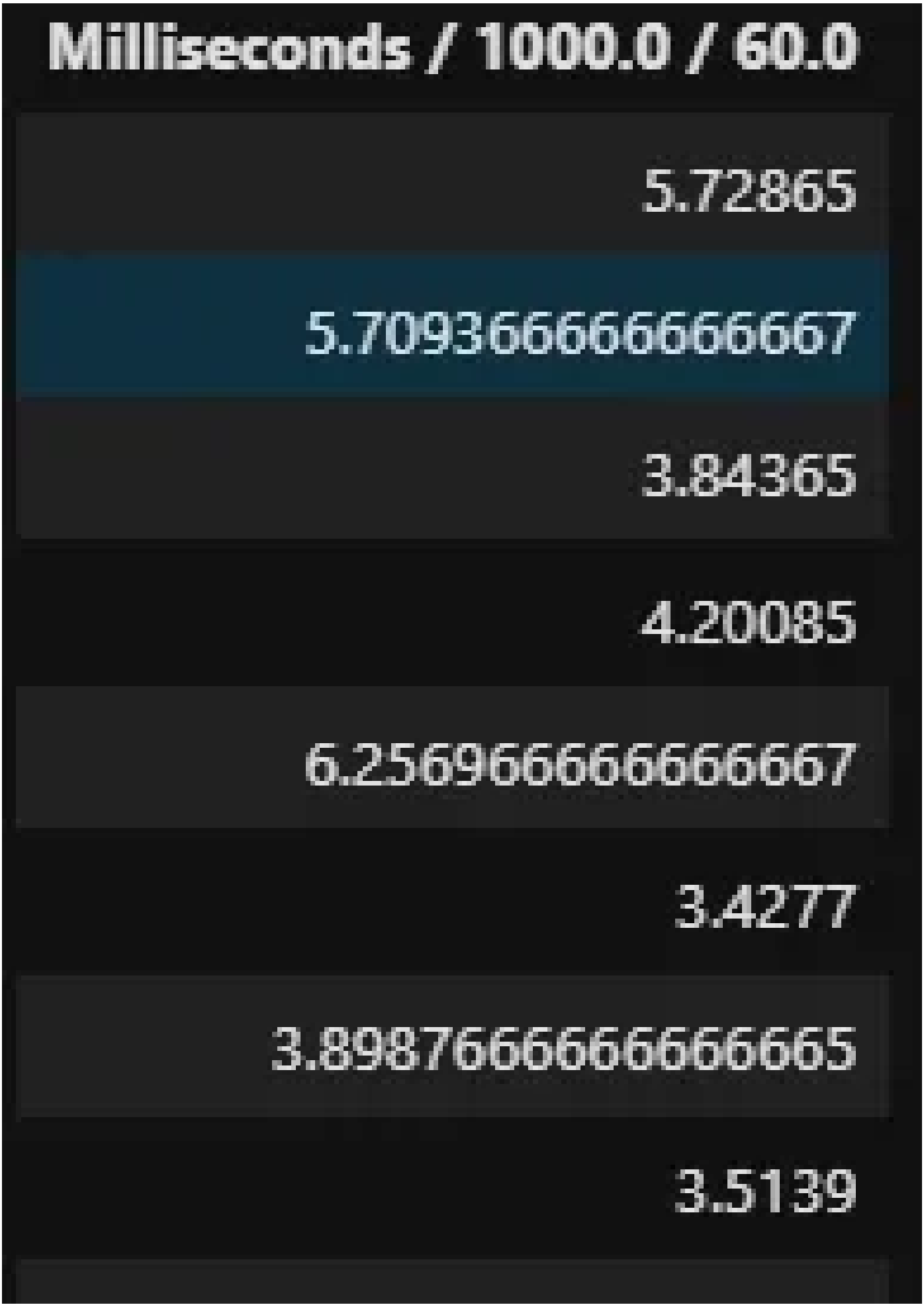
```
SELECT 10 / 3
```

Instead of returning 3.33..., the result is 3. To get a float result, you should use at least one float number in the query or use all floats to be safe:

```
%%sql
```

```
SELECT 10.0 / 3.0
```

Using this knowledge, let's calculate the average duration of a song in minutes:


Milliseconds / 1000.0 / 60.0
5.72865
5.709366666666667
3.84365
4.20085
6.256966666666667
3.4277
3.898766666666665
3.5139

If you pay attention to the above column, its name is written as "*the query used to generate that column*." Because of this behavior, using long calculations, such as finding the standard deviation or variance of a column, can be an issue because the column name will be as large as the query itself.

To avoid this, SQL allows aliasing, similar to aliasing import statements in Python. For example:

Using `as` keyword after a single item in a `SELECT` statement tells SQL that we are aliasing. Here are more examples:

sum_total	max_total	min_total	mean_total
2328.600000000004	25.86	0.99	5.6519417475728 25

You can use aliasing just as easily for columns with long names.

Ordering Results in SQL

Just like Pandas has `sort_values` method, SQL supports ordering columns via `ORDER BY` clause. Passing a column name after the clause sorts the results in ascending order:

Name	Composer	UnitPrice
Iron Man	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
Children Of The Grave	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
Paranoid	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
New Rhumba	A. Jamal	0.99
Astronomy	A.Bouchard/J.Bouchard/S.Pearlman	0.99
Hard To Handle	A.Isbell/A.Jones/O.Redding	0.99
Go Down	AC/DC	0.99
Dog Eat Dog	AC/DC	0.99
Let There Be Rock	AC/DC	0.99
Bad Boy Boogie	AC/DC	0.99

We order the tracks table in ascending order by the composer's name. Note that the ORDER BY statement should always come after the WHERE clause. It is also possible to pass two or more columns to ORDER BY:

Name	Composer	UnitPrice
Children Of The Grave	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
Iron Man	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
Paranoid	A. F. Iommi, W. Ward, T. Butler, J. Osbourne	0.99
New Rhumba	A. Jamal	0.99
Astronomy	A.Bouchard/J.Bouchard/S.Pearlman	0.99
Hard To Handle	A.Isbell/A.Jones/O.Redding	0.99
Bad Boy Boogie	AC/DC	0.99
Dog Eat Dog	AC/DC	0.99
Go Down	AC/DC	0.99
Hell Ain't A Bad Place To Be	AC/DC	0.99

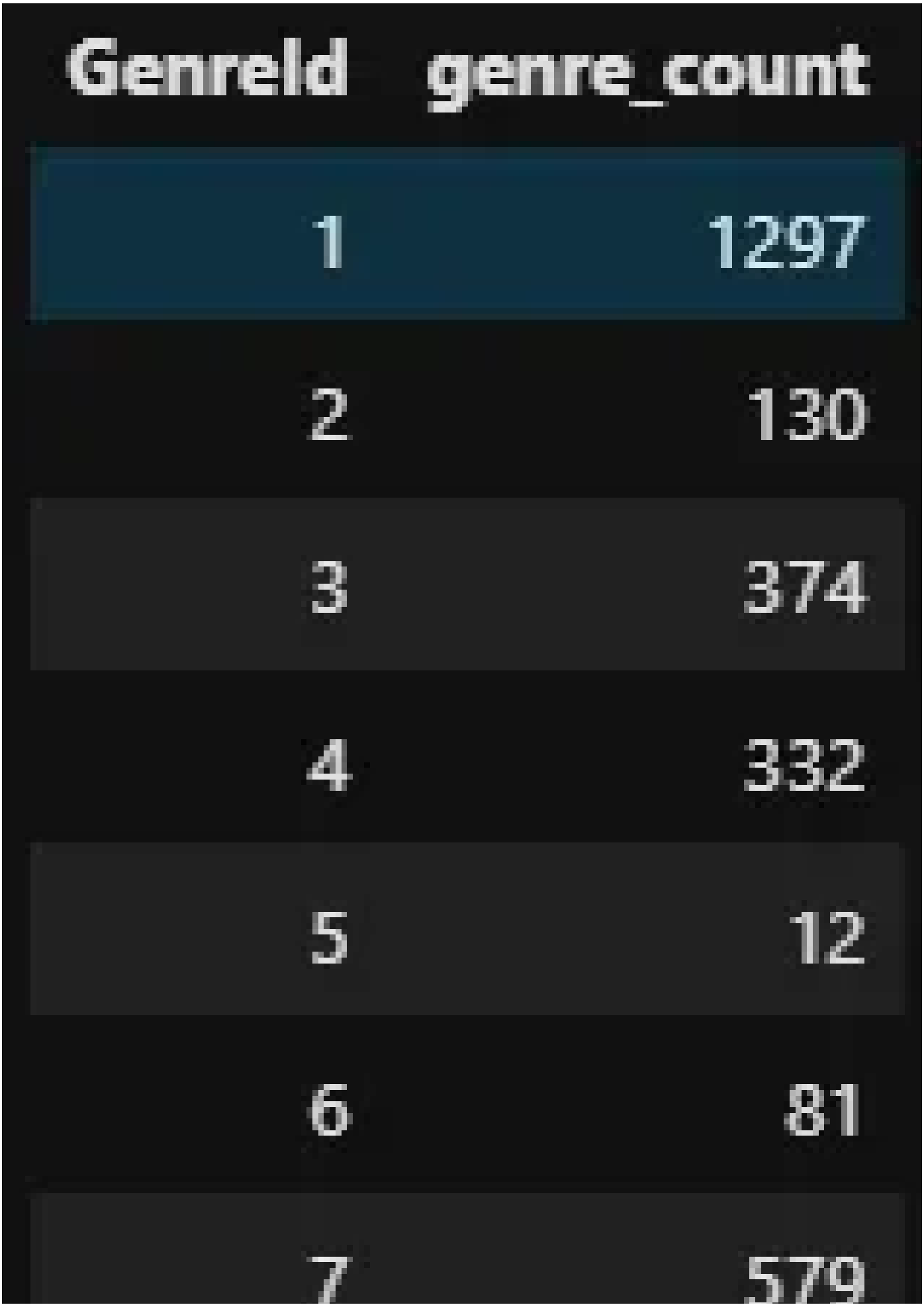
You can also reverse the ordering by passing the `DESC` keyword after each column name:

Name	Composer	UnitPrice
A Twist In The Tail	roger glover	0.99
Lick It Up	roger glover	0.99
One Man's Meat	roger glover	0.99
Ramshackle Man	roger glover	0.99
Solitaire	roger glover	0.99
Talk About Love	roger glover	0.99
Time To Kill	roger glover	0.99
Loves Been Good To Me	rod mckuen	0.99
For Once In My Life	orlando murden/ronald miller	0.99
The Lady Is A Tramp	lorenz hart/richard rodgers	0.99

The above query returns three columns after ordering the UnitPrice and Compose in descending order and the name in ascending order (`ASC` is a default keyword).

Grouping in SQL

One of the most powerful functions of Pandas is the `groupby` . You can use it to transform a table into virtually any shape you want. Its very close cousin in SQL - `GROUP BY` clause can be used to achieve the same functionality. For example, the below query counts the number of songs in each genre:



GenreId	genre_count
1	1297
2	130
3	374
4	332
5	12
6	81
7	579

The difference between the GROUP BY in SQL and `groupby` in Pandas is that SQL does not allow selecting columns that weren't given in the GROUP BY clause. For example, adding an extra free column in the above query generates an error:

However, you can choose as many columns in the SELECT statement as you like as long as you are using some type of aggregate function on them:

GenreId	AlbumId	genre_count	avg_duration	avg_price
1	1	10	4.00069166666666665	0.99
1	2	1	5.709366666666667	0.99
1	3	3	4.767155555555556	0.9899999999999999
1	4	8	5.11095625	0.9900000000000001
1	5	15	4.901898888888889	0.9900000000000001
1	6	13	4.4242628205128 21	0.9900000000000001
1	7	12	4.513006944444444	0.9900000000000001
1	10	14	4.67584880952381	0.9900000000000001
1	30	14	5.345142857142857	0.9900000000000001
1	31	9	4.566535185185185	0.99

The above query includes almost all the topics we have learned up to this point. We are grouping by both album ID and genre ID, and for each group, we calculate the average duration and price of a song. We are also making efficient use of aliasing.

We can make the query even more powerful by ordering by the average duration and genre count:

GenreId	AlbumId	genre_count	avg_duration	avg_price
19	227	5	52.615260000000006	1.9899999999999998
20	253	24	48.759572222222225	1.9900000000000002
20	227	2	45.771458333333335	1.99
21	229	22	45.60556136363636	1.99
19	231	16	44.05144166666666	1.9899999999999993
18	227	12	43.763733333333334	1.9899999999999995
21	231	8	43.750514583333334	1.99
18	226	1	43.704166666666666	1.99
19	229	4	43.6093375	1.99
19	228	3	43.44713888888889	1.99

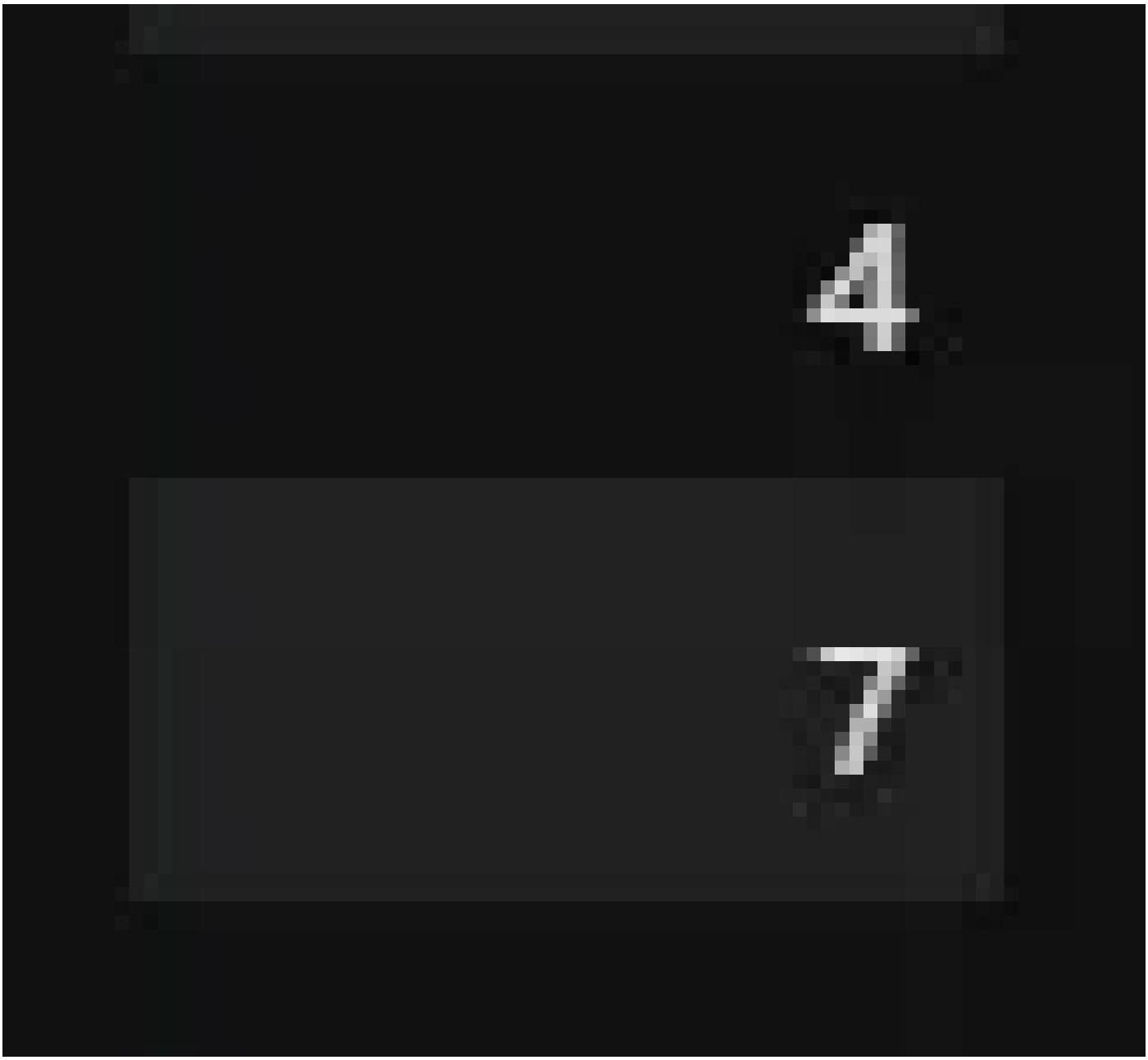
Pay attention to how we use the alias names of the aggregate functions in the ORDER BY clause. Once you alias a column or the result of the aggregate function, you can refer to them only by their alias for the rest of the query.

Using conditionals with HAVING

By default, SQL does not allow conditional filtering using aggregate functions in the WHERE clause. For example, we want to select only the genres where the number of songs is greater than 100. Let's try this with the WHERE clause:

The correct way of filtering rows based on the results of aggregate functions is using the HAVING clause:


Genreld
1
2
3



HAVING clause is usually used with GROUP BY. Whenever you want to filter rows using aggregate functions, the HAVING clause is the way to go!

> *All images were generated by myself.*

Summary

By now, you should have realized how powerful SQL can be and how much more readable it becomes compared to Pandas. Even though we learned a ton, we have barely scratched the surface.

For practice problems, I recommend Data Lemur or LeetCode if you are feeling adventurous.

Loved this article and, let's face it, its bizarre writing style? Imagine having access to dozens more just like it, all written by a brilliant, charming, witty author (that's me, by the way :).

For only 4.99$ membership, you will get access to not just my stories, but a treasure trove of knowledge from the best and brightest minds on Medium. And if you use my referral link, you will earn my supernova of gratitude and a virtual high-five for supporting my work.

Join Medium with my referral link — Bex T.

Get exclusive access to all my ⚡ premium ⚡ content and all over Medium without limits. Support my work by buying me a...

ibexorigin.medium.com

Data Science | Sql | Programming | Pandas | Python

Follow

Written by Bex T.

19.6K Followers · Writer for Towards Data Science

BEXGBoost | DataCamp Instructor | 🥇 Top 10 AI/ML Writer on Medium | Kaggle Master |
https://www.linkedin.com/in/bextuychiev/

More from Bex T. and Towards Data Science

Bex T. in Towards AI

Forget PIP, Conda, requirements.txt! Use Poetry Instead And Thank Me Later

Pain-free dependency management is finally here

✨ · 8 min read · Jun 24

1.3K 16

Ken Jee in Towards Data Science

How to Use ChatGPT to Learn Data Science Faster, Even If You Are Already Advanced

Discussing the relevance of data science in the AI-driven future and providing a step-by-step guide on how to learn data science...

✨ · 10 min read · Jul 1

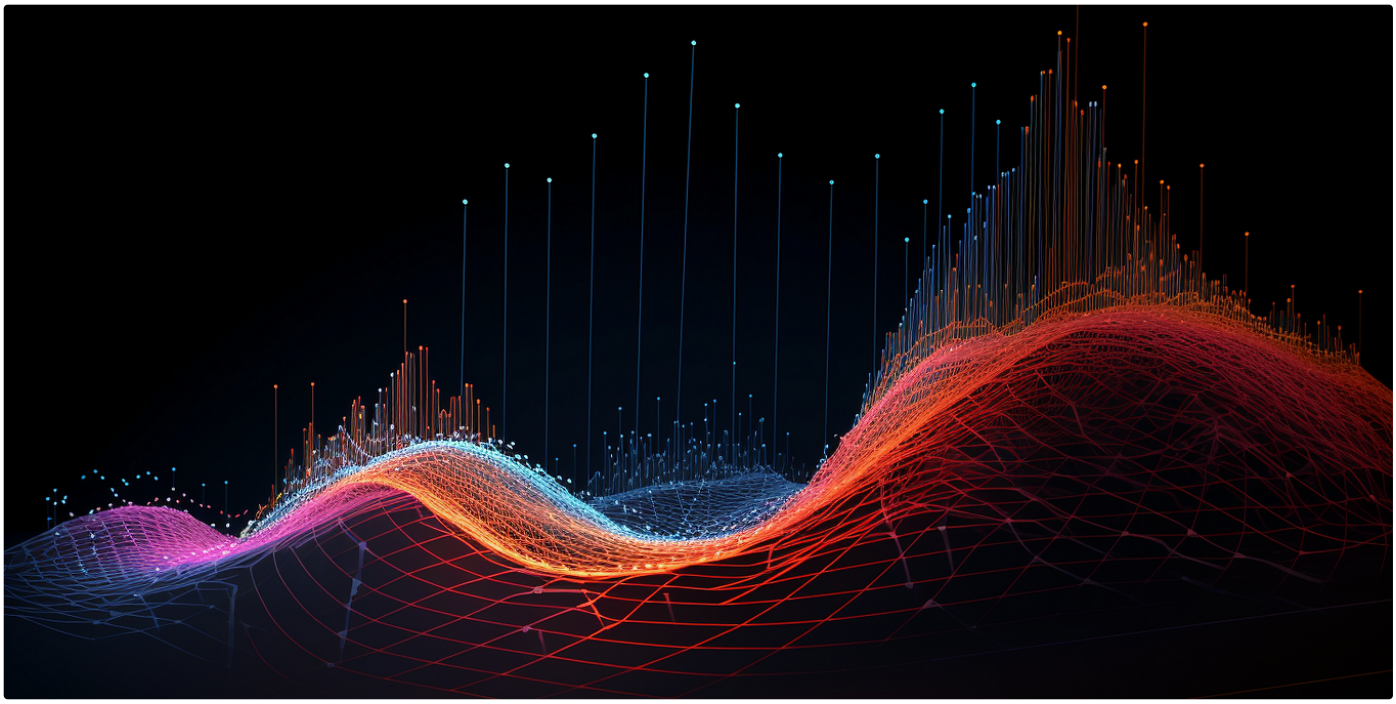
451 4

Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

· 12 min read · Apr 25



Bex T. in Towards AI

10 Advanced Matplotlib Concepts You Must Know To Create Killer Visuals

Become Leonardo da Matplotlib

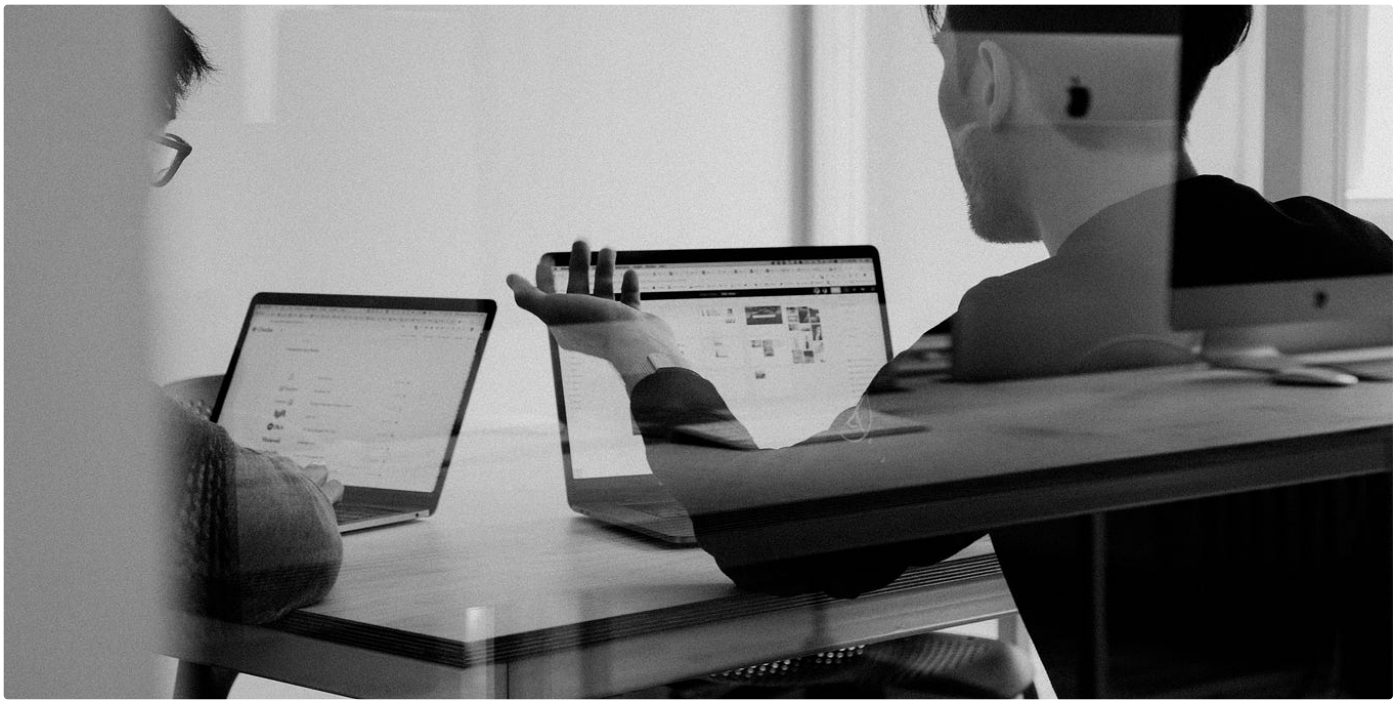
✦ · 9 min read · Jun 30

663 5

See all from Bex T.

See all from Towards Data Science

Recommended from Medium



Khushal Kumar

My Data Analyst Interview at a Finance Company: 6 Questions and Answers.

Interview Experience for a Data Analyst Role at a Finance Sector Company

3 min read · Apr 8

89 3

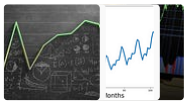
Miriam Santos in Towards Data Science

A Data Scientist's Essential Guide to Exploratory Data Analysis

Best Practices, Techniques, and Tools to Fully Understand Your Data

11 min read · May 30

Lists



Predictive Modeling w/ Python

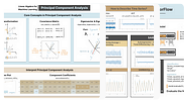
18 stories · 144 saves

New_Reading_List

174 stories · 29 saves

Coding & Development

11 stories · 58 saves



Practical Guides to Machine Learning

10 stories · 159 saves

Benedict Neo in bitgrit Data Science Publication

11 Machine Learning Interview Questions

On Machine Learning concepts, ML System Design, and Python

8 min read · Mar 2

137 2

Jari Roomer in Better Humans

How I Eliminated Procrastination From My Life (Using Neuroscience)

Keep this part of the brain in optimal condition if you want to stop procrastinating.

· 6 min read · Jun 22

11.4K 136

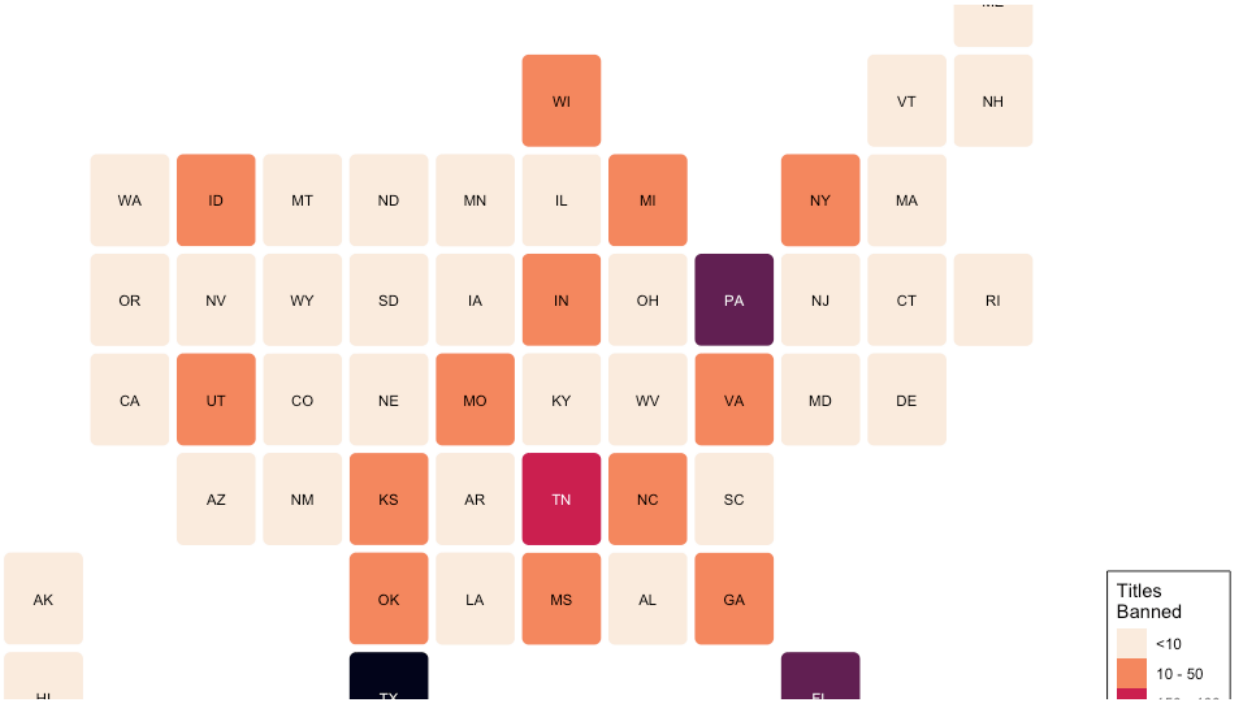
ZenRows

Web Scraping in Python: Avoid Detection Like a Ninja

Scraping should be about extracting content from HTML. It sounds simple but has many obstacles. The first one is to obtain the said HTML...

13 min read · Apr 5

299 4



Statecraft by Arman Madani

We Analyzed 1,626 Banned Books...Here's What We Found

What do banned books have in common?

5 min read · Jul 3

2.2K 47

See more recommendations