Member-only story

Understanding GroupBy in Polars DataFrame by Examples

Learn how to use the groupby() method in Polars to aggregate data and apply functions

Wei-Meng Lee · Follow
Published in Towards Data Science · 8 min read · Aug 5, 2022

Photo by Hannah Busing on Unsplash

So far in my previous few articles on Polars, I have introduced you to the basics of the Polars DataFrame, why it is a better dataframe library than Pandas, and how its lazy evaluation feature allows it to optimize your queries. As you explore Polars, you will encounter one very common question — how do I perform a *groupby* on the Polars DataFrame? Most Pandas users are familiar with the `groupby()` function, where you can group data according to categories and then apply functions to the categories. How do you do that in Polars?

In this article, I will walk you through how you can use the `groupby()` function on the Polars DataFrame. In particular, I will illustrate them using a few different examples. Ready? Let's go!

Example 1

For the first example, I am going to create the Polars DataFrame by hand:

```
import polars as pl

scores = {'Zone': ['North', 'North', 'South', 'South',
                   'East', 'East', 'West', 'West'],
          'School': ['Rushmore', 'Rushmore','Bayside','Rydell',
                     'Shermer','Shermer','Ridgemont','Hogwarts'],
          'Name': ['Jonny', 'Mary', 'Joe', 'Jakob',
                   'Jimmy', 'Erik', 'Lam', 'Yip'],
          'Math': [78, 39, 76, 56, 67, 89, 100, 55],
          'Science': [70, 45, 68, 90, 45, 66, 89, 32]}

df = pl.DataFrame(scores, columns =
                  ['Zone', 'School', 'Name',
                   'Science', 'Math'])
df
```

As you can see from the output, the dataframe has a shape of eight rows and five columns:

shape: (8, 5)

Zone	School	Name	Science	Math
str	str	str	i64	i64
"North"	"Rushmore"	"Jonny"	78	70
"North"	"Rushmore"	"Mary"	39	45
"South"	"Bayside"	"Joe"	76	68
"South"	"Rydell"	"Jakob"	56	90
"East"	"Shermer"	"Jimmy"	67	45
"East"	"Shermer"	"Erik"	89	66
"West"	"Ridgemont"	"Lam"	100	89
"West"	"Hogwarts"	"Yip"	55	32

Image by author

Grouping by zone

Let's now see all the schools in each zone by using the `groupby()` and the `agg()` methods:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        'School'
    )
```

```
)
q.collect()
```

> *You use the* `lazy()` *method to ensure that Polars uses lazy evaluation for all the queries that follow.*

You will see the following output:

shape: (4, 2)

Zone	School
str	list[str]
"East"	["Shermer", "Shermer"]
"North"	["Rushmore", "Rushmore"]
"South"	["Bayside", "Rydell"]
"West"	["Ridgemont", "Hogwarts"]

Image by author

Note that the schools for each zone are now stored as a list under the `School` column. To see all the schools in a particular zone, you can use the `filter()` method:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        'School'
    )
    .filter(
        pl.col('Zone')=='East'
    )
)
q.collect()
```

shape: (1, 2)

Zone	School
str	list[str]
"East"	["Shermer", "Shermer"]

Image by author

Viewing the maximum score for each zone

If you want to know the highest **Science** scores for each zone, use the `max()` method on the **Science** column:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        pl.col('Science').max().alias('Science(Max)')
    )
```

```
)
q.collect()
```

> *The `alias()` method renames the output column.*

Zone	Science(Max)
str	i64
"West"	100
"North"	78
"East"	89
"South"	76

Image by author

You can also directly use the `pl.max()` method on the **Science** column like this:

```
# pl.col('Science').max().alias('Science(Max)')
pl.max('Science').alias('Science(Max)')
```

If you want to also count the number of schools in each zone, you can add in another expression to the `agg()` method:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        [
            pl.col('Science').count().alias('Number of Schools'),
            pl.col('Science').max().alias('Science(Max)')
        ]
    )
)
q.collect()
```

The result is as follows:

shape: (4, 3)

Zone	Number of Schools	Science(Max)
str	u32	i64
"East"	2	89
"North"	2	78
"South"	2	76
"West"	2	100

Image by author

You can add different expressions to get the results of the aggregated groups:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        [
            pl.col('Science').count().alias('Number of Schools'),
            pl.col('Science').max().alias('Science(Max)'),
            pl.col('Science').min().alias('Science(Min)'),
            pl.col('Science').mean().alias('Science(Mean)'),
            pl.col('Math').max().alias('Math(Max)'),
            pl.col('Math').min().alias('Math(Min)'),
            pl.col('Math').mean().alias('Math(Mean)'),
        ]
    )
)
q.collect()
```

shape: (4, 8)

Zone	Number of Schools	Science(Max)	Science(Min)	Science(Mean)	Math(Max)	Math(Min)	Math(Mean)
str	u32	i64	i64	f64	i64	i64	f64
"North"	2	78	39	58.5	70	45	57.5
"East"	2	89	67	78.0	66	45	55.5
"South"	2	76	56	66.0	90	68	79.0
"West"	2	100	55	77.5	89	32	60.5

Image by author

Modifying the sort order

The order of the result returned by the `groupby()` method is randomized, so each time you run the above code snippet you will might see the **Zone** in different order. You can use the `sort()` method to give it an order:

```
q = (
    df
    .lazy()
    .groupby(by='Zone')
    .agg(
        [
            pl.max('Science').alias('Science(Max)')
        ]
    )
    .sort(by='Zone')
)
q.collect()
```

shape: (4, 2)

Zone	Science(Max)
str	i64
"East"	89
"North"	78
"South"	76
"West"	100

Image by author

Observe that the **Zone** is sorted alphabetically, and not based on the *cardinal directions* (i.e. North, South, East, and West). So how do you sort the **Zone** based on cardinal directions? The following code snippet shows how this can be done:

```
df_sortorder = pl.DataFrame({
   'Zone' : ['North','South','East','West'],
   'Zone_order' : [0,1,2,3]
}).lazy()

q = (
    df
    .lazy()
    .join(df_sortorder, on='Zone', how='left')
    .groupby(by=['Zone','Zone_order'])
    .agg(
        [
            pl.max('Science').alias('Science(Max)')
        ]
    )
    .sort('Zone_order')
    .select(
        pl.exclude('Zone_order')
    )
)
q.collect()
```

In the above code snippet, I:

- Created another dataframe (`df_sortorder`) specifying the order that I want the zones to be sorted
- Joined the original dataframe with `df_sortorder` so that my dataframe now have a new column called `Zone_order`
- Sorted the group based on the new `Zone_order` column
- Excluded the `Zone_order` column in the final result

The result is as shown below:

shape: (4, 2)

Zone	Science(Max)
str	i64
"North"	78
"South"	76
"East"	89
"West"	100

Image by author

Example 2

The next example uses the dataset "**vehicle-make-model-data**." Download the **csv_data.csv** file from this page.

> ***License*** *— MIT License*
>
> ***Description*** *— Accurate motor vehicle make & model data since year 2001. This data set includes Car, Motorcycle, Truck, and UTV manufactures and their corresponding models. The data is database agnostic and is user-friendly as* same*set of data is ported to mysql, json, and csv format. Json and csv data sets are flattened while mysql data set being normalized to 3 tables. Currently there are 19,722 models and increasing.*

First, load the CSV file into a Polars dataframe:

```
q = (
    pl.scan_csv('csv_data.csv')
)
q.collect()
```

You should see the following:

shape: (19772, 3)

year	make	model
i64	str	str
2001	"ACURA"	"CL"
2001	"ACURA"	"EL"
2001	"ACURA"	"INTEGRA"
2001	"ACURA"	"MDX"
2001	"ACURA"	"NSX"
2001	"ACURA"	"RL"
2001	"ACURA"	"TL"
2001	"AM GENERAL"	"HUMMER"
2001	"AMERICAN IRONH...	"CLASSIC"
2001	"AMERICAN IRONH...	"LEGEND"
2001	"AMERICAN IRONH...	"OUTLAW"
2001	"AMERICAN IRONH...	"RANGER"
...	...	...
2015	"VOLKSWAGEN"	"TOUAREG"
2015	"VOLVO"	"S60"
2015	"VOLVO"	"S80"
2015	"VOLVO"	"V60"
2015	"VOLVO"	"V60 CROSS COUN...
2015	"VOLVO"	"XC60"
2015	"VOLVO"	"XC70"
2015	"YAMAHA"	"SR400"
2016	"KIA"	"SORENTO"
2016	"MAZDA"	"6"
2016	"MAZDA"	"CX-5"
2016	"VOLVO"	"XC90"

Image by author

The dataframe contains a list of manufacturers (**make**) and their **model**s from the year 2001 to 2016.

Grouping by year and make

Let's first group the dataframe by **year** and **make**, and then count the number of **model**s per **make** for each **year**:

```
q = (
    pl.scan_csv('csv_data.csv')
    .groupby(by=['year','make'])
    .agg(
        pl.col(['make']).count().alias('count')
    )
    .sort(by=['year','make'])
)

q.collect()
```

shape: (1467, 3)

year	make	count
i64	str	u32
2001	"ACURA"	7
2001	"AM GENERAL"	1
2001	"AMERICAN IRONH...	7
2001	"APRILIA"	16
2001	"ARCTIC CAT"	51
2001	"ASTON MARTIN"	2
2001	"ATK"	7
2001	"AUDI"	13
2001	"AVANTI"	1
2001	"BENTLEY"	3
2001	"BIG DOG"	8
2001	"BIMOTA"	1
...	...	...
2015	"SMART"	1
2015	"SUBARU"	8
2015	"SUZUKI"	1
2015	"TESLA"	2
2015	"TOYOTA"	16
2015	"VICTORY"	1
2015	"VOLKSWAGEN"	13
2015	"VOLVO"	6
2015	"YAMAHA"	1
2016	"KIA"	1
2016	"MAZDA"	2
2016	"VOLVO"	1

Image by author

The above can be simplied by using the `pl.count()` method:

```
q = (
    pl.scan_csv('csv_data.csv')
    .groupby(by=['year','make'])
    .agg(
        pl.count()
    )
    .sort(by=['year','make'])
)

q.collect()
```

If you want to see all the models for each car make, you can specify the **model** column:

```
q = (
    pl.scan_csv('csv_data.csv')
    .groupby(by=['year','make'])
    .agg(
        'model'
    )
    .sort(by=['year','make'])
)

q.collect()
```

This will group all the **model** names as lists for each **year** and **make**:

shape: (1467, 3)

year	make	model
i64	str	list[str]
2001	"ACURA"	["CL", "EL", ... "TL"]
2001	"AM GENERAL"	["HUMMER"]
2001	"AMERICAN IRONH...	["CLASSIC", "LEGEND", ... "THUNDER"]
2001	"APRILIA"	["ATLANTIC 500", "ETV 1000 CAPONORD", ... "SL1000 FALCO"]
2001	"ARCTIC CAT"	["250 2X4", "250 4X4", ... "ZRT 800 LE"]
2001	"ASTON MARTIN"	["DB7", "VANQUISH"]
2001	"ATK"	["125 CC", "250 ENDURO", ... "605 ENDURO"]
2001	"AUDI"	["A3", "A4", ... "TT QUATTRO"]
2001	"AVANTI"	["II"]
2001	"BENTLEY"	["ARNAGE", "AZURE", "CONTINENTAL"]
2001	"BIG DOG"	["BOXER", "BULLDOG", ... "WOLF"]
2001	"BIMOTA"	["SB8R"]
...	...	...
2015	"SMART"	["FORTWO"]
2015	"SUBARU"	["BRZ", "FORESTER", ... "XV CROSSTREK"]
2015	"SUZUKI"	["TU250X"]
2015	"TESLA"	["S", "X"]
2015	"TOYOTA"	["4RUNNER", "AVALON", ... "YARIS"]
2015	"VICTORY"	["GUNNER"]
2015	"VOLKSWAGEN"	["AMAROK", "BEETLE", ... "TOUAREG"]
2015	"VOLVO"	["S60", "S80", ... "XC70"]
2015	"YAMAHA"	["SR400"]
2016	"KIA"	["SORENTO"]
2016	"MAZDA"	["6", "CX-5"]
2016	"VOLVO"	["XC90"]

Image by author

Showing the first and last model for each car make

If you want to show the first and last **model** for each car **make**, use the `first()` and `last()` methods, respectively:

```
q = (
    pl.scan_csv('csv_data.csv')
    .groupby(by=['year','make'])
    .agg(
        [
            'model',
            pl.first('model').alias('first'),
```

```
            pl.last('model').alias('last'),
        ]
    )
    .sort(by=['year','make'])
)

q.collect()
```

The first and last models will now be shown in their own separate columns:

shape: (1467, 5)

year	make	model	first	last
i64	str	list[str]	str	str
2001	"ACURA"	["CL", "EL", ... "TL"]	"CL"	"TL"
2001	"AM GENERAL"	["HUMMER"]	"HUMMER"	"HUMMER"
2001	"AMERICAN IRONH...	["CLASSIC", "LEGEND", ... "THUNDER"]	"CLASSIC"	"THUNDER"
2001	"APRILIA"	["ATLANTIC 500", "ETV 1000 CAPONORD", ... "SL1000 FALCO"]	"ATLANTIC 500"	"SL1000 FALCO"
2001	"ARCTIC CAT"	["250 2X4", "250 4X4", ... "ZRT 800 LE"]	"250 2X4"	"ZRT 800 LE"
2001	"ASTON MARTIN"	["DB7", "VANQUISH"]	"DB7"	"VANQUISH"
2001	"ATK"	["125 CC", "250 ENDURO", ... "605 ENDURO"]	"125 CC"	"605 ENDURO"
2001	"AUDI"	["A3", "A4", ... "TT QUATTRO"]	"A3"	"TT QUATTRO"
2001	"AVANTI"	["II"]	"II"	"II"
2001	"BENTLEY"	["ARNAGE", "AZURE", "CONTINENTAL"]	"ARNAGE"	"CONTINENTAL"
2001	"BIG DOG"	["BOXER", "BULLDOG", ... "WOLF"]	"BOXER"	"WOLF"
2001	"BIMOTA"	["SB8R"]	"SB8R"	"SB8R"
...	...	...	...	...
2015	"SMART"	["FORTWO"]	"FORTWO"	"FORTWO"
2015	"SUBARU"	["BRZ", "FORESTER", ... "XV CROSSTREK"]	"BRZ"	"XV CROSSTREK"
2015	"SUZUKI"	["TU250X"]	"TU250X"	"TU250X"
2015	"TESLA"	["S", "X"]	"S"	"X"
2015	"TOYOTA"	["4RUNNER", "AVALON", ... "YARIS"]	"4RUNNER"	"YARIS"
2015	"VICTORY"	["GUNNER"]	"GUNNER"	"GUNNER"
2015	"VOLKSWAGEN"	["AMAROK", "BEETLE", ... "TOUAREG"]	"AMAROK"	"TOUAREG"
2015	"VOLVO"	["S60", "S80", ... "XC70"]	"S60"	"XC70"
2015	"YAMAHA"	["SR400"]	"SR400"	"SR400"
2016	"KIA"	["SORENTO"]	"SORENTO"	"SORENTO"
2016	"MAZDA"	["6", "CX-5"]	"6"	"CX-5"
2016	"VOLVO"	["XC90"]	"XC90"	"XC90"

Image by author

Example 3

For this example, we shall use this dataset.

> *License: CC0: Public Domain*
>
> ***Description** — This dataset contains 1338 rows of insured data, where the Insurance charges are given against the following attributes of the insured: Age, Sex, BMI, Number of Children, Smoker and Region. The attributes are a mix of numeric and categorical variables.*

First, load the CSV file as a Polars DataFrame:

```
q = (
    pl.scan_csv('insurance.csv')
)
q.collect()
```

The dataframe contains the various profiles of customers:

shape: (1338, 7)

age	sex	bmi	children	smoker	region	charges
i64	str	f64	i64	str	str	f64
19	"female"	27.9	0	"yes"	"southwest"	16884.924
18	"male"	33.77	1	"no"	"southeast"	1725.5523
28	"male"	33.0	3	"no"	"southeast"	4449.462
33	"male"	22.705	0	"no"	"northwest"	21984.4706
32	"male"	28.88	0	"no"	"northwest"	3866.8552
31	"female"	25.74	0	"no"	"southeast"	3756.6216
46	"female"	33.44	1	"no"	"southeast"	8240.5896
37	"female"	27.74	3	"no"	"northwest"	7281.5056
37	"male"	29.83	2	"no"	"northeast"	6406.4107
60	"female"	25.84	0	"no"	"northwest"	28923.1369
25	"male"	26.22	0	"no"	"northeast"	2721.3208
62	"female"	26.29	0	"yes"	"southeast"	27808.7251
…	…	…	…	…	…	…
42	"female"	32.87	0	"no"	"northeast"	7050.0213
51	"male"	30.03	1	"no"	"southeast"	9377.9047
23	"female"	24.225	2	"no"	"northeast"	22395.7442
52	"male"	38.6	2	"no"	"southwest"	10325.206
57	"female"	25.74	2	"no"	"southeast"	12629.1656
23	"female"	33.4	0	"no"	"southwest"	10795.9373
52	"female"	44.7	3	"no"	"southwest"	11411.685
50	"male"	30.97	3	"no"	"northwest"	10600.5483
18	"female"	31.92	0	"no"	"northeast"	2205.9808
18	"female"	36.85	0	"no"	"southeast"	1629.8335
21	"female"	25.8	0	"no"	"southwest"	2007.945
61	"female"	29.07	0	"yes"	"northwest"	29141.3603

Image by author

Counting the numbers of males and females

To count the number of males and females in each region, use the `sum()` method when aggregating:

```
q = (
    pl.scan_csv('insurance.csv')
    .groupby(by='region')
    .agg(
        [
            (pl.col('sex') == 'male').sum().alias('male'),
            (pl.col('sex') == 'female').sum().alias('female'),
        ]
    )
    .sort(by='region')
)
q.collect()
```

The number of males and females for each region will now be displayed in 2 columns:

shape: (4, 3)

region	male	female
str	u32	u32
"northeast"	163	161
"northwest"	161	164
"southeast"	189	175
"southwest"	163	162

Image by author

Calculating the mean charges for each gender

To calculate the mean charges for each gender in each region, you have to:

- select the `charges` column first
- filter for each gender
- calculate the mean charges
- rename the column

Here is the code snippet to perform the steps described above:

```
q = (
    pl.scan_csv('insurance.csv')
    .groupby(by='region')
    .agg(
        [
            (pl.col('charges')
             .filter(pl.col('sex')== 'male'))
            .mean()
            .alias('male_mean_charges'),

            (pl.col('charges')
             .filter(pl.col('sex')== 'female'))
            .mean()
            .alias('female_mean_charges'),
        ]
    )
    .sort(by='region')
)
q.collect()
```

Here is the result of the above code snippet:

shape: (4, 3)

region	male_mean_charges	female_mean_charges
str	f64	f64
"northeast"	13854.005374	12953.203151
"northwest"	12354.119575	12479.870397
"southeast"	15879.617173	13499.669243
"southwest"	13412.883576	11274.411264

Image by author

To verify that our above result is correct, we can write the following code snippet to manually retrieve all the rows containing males and from the

northeast region, and then computing their mean charges:

```
q = (
    pl.scan_csv('insurance.csv')
    .filter(
        (pl.col('region')=='northeast') & (pl.col('sex') == 'male')
    )
    .select(
        pl.col('charges')
    )
    .mean()
)
q.collect()
```

The result below shows the mean charges for all the males in the northeast region:

shape: (1, 1)

charges
f64
13854.005374

Image by author

Computing the proportion of smokers in each region

To calculate the proportion of smokers in each region, you need to:

- sum up all the smokers
- count the total number of rows in the smoker column

Here is the code snippet to perform the steps described above:

```
q = (
    pl.scan_csv('insurance.csv')
    .groupby(by='region')
    .agg(
        [
            ((pl.col('smoker')=='yes').sum() /
             (pl.col('smoker')).count() * 100).alias('Smoker %')
        ]
    )
    .sort(by='region')
)
q.collect()
```

The result below shows the percentage of smokers in each region:

shape: (4, 2)

region	Smoker %
str	f64
"northeast"	20.679012
"northwest"	17.846154
"southeast"	25.0
"southwest"	17.846154

Image by author

> *I will be running a workshop on Polars in the upcoming ML Conference (22–24 Nov 2022) in Singapore. If you want a jumpstart on the Polars DataFrame, register for my workshop at https://mlconference.ai/machine-learning-advanced-development/using-polars-for-data-analytics-workshop/.*

Summary

I hope the code samples above provide you with enough fodders to see the use of the `groupby()` method. As you would have noticed from the code, the `groupby()` method is often used in conjunction with the `agg()` method. One thing you have to remember when using Polars is that you should avoid using the `apply()` function as it will affect performance of your query.

For your reference, here are my previous articles on the Polars DataFrame:

Understanding Lazy Evaluation in Polars

Understand what is eager and lazy execution and how you can use lazy execution to optimize your queries

towardsdatascience.com

Visualizing Polars DataFrames using Plotly Express

Learn how to plot your Polars DataFrames

towardsdatascience.com

Manipulating Values in Polars DataFrames

Learn how to use the various methods in Polars to manipulate your dataframes

towardsdatascience.com

Polars Groupby Pandas Data Science Programming

More from the list: "Reading list"

Curated by @reenum

Suhith Illesinghe
Agile is dead
· 6d ago

Allie Pasc... in UX Collecti...
"Winning" by design: Deceptive UX patterns...
6d ago

Oliver Bennet
Mastering Bash: Essential Commands for Everyda...
Oct 23

The I've

View list

Written by Wei-Meng Lee

2.8K Followers · Writer for Towards Data Science

Follow

ACLP Certified Trainer | Blockchain, Smart Contract, Data Analytics, Machine Learning, Deep Learning, and all things tech (http://calendar.learn2develop.net).

More from Wei-Meng Lee and Towards Data Science

Wei-Meng Lee in AI Advances

Accelerating Hugging Face Pre-trained Models on Apple Silicon...

A Guide to Training and Serving Models Efficiently on M1, M2, and M3 Chips

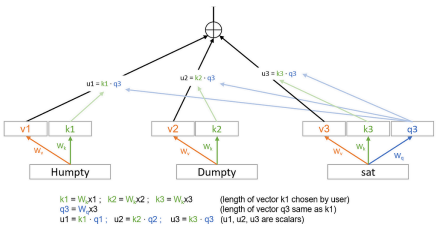
Oct 13 286 3

Shaw Talebi in Towards Data Science

5 AI Projects You Can Build This Weekend (with Python)

From beginner-friendly to advanced

Oct 9 3.5K 60

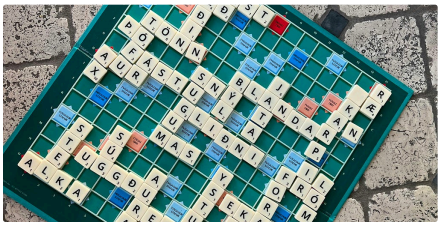


Rohit Patel in Towards Data Science

Understanding LLMs from Scratch Using Middle School Math

In this article, we talk about how LLMs work, from scratch—assuming only that you know...

Oct 19 1.8K 19



Wei-Meng Lee in AI Advances

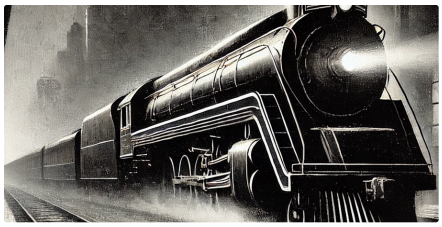
Understanding Tokenization in NLP

Breaking Down Word-Level, Subword-Level, Padding, and Attention Masks for Effective...

4d ago 321 2

See all from Wei-Meng Lee | See all from Towards Data Science

Recommended from Medium

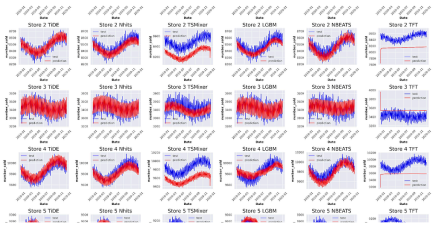


Tim Zhao

A Better Way to Use the Pandas DataFrame: Treat Each Row as a...

Pandas is a powerful and flexible data manipulation library that is a staple in the...

Aug 13



Esther Cifuentes

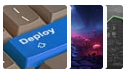
Comparing Time Series Algorithms

Evaluating Leading Time Series Algorithm with Darts.

Oct 16 83

Lists

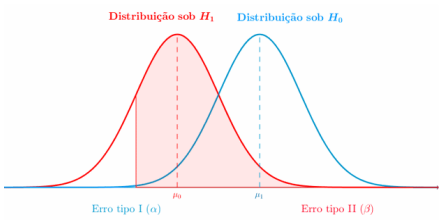
General Coding Knowledge
20 stories · 1688 saves



Predictive Modeling w/ Python
20 stories · 1626 saves

Coding & Development
11 stories · 878 saves

Practical Guides to Machine Learning
10 stories · 1982 saves

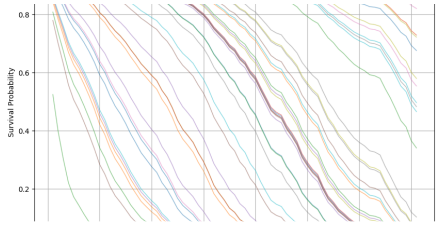


Crystal X

Use Python in Excel to explore 20 popular hypothesis tests

Hypothesis testing is an integral part of statistical analysis. Statisticians use...

6d ago 15 1



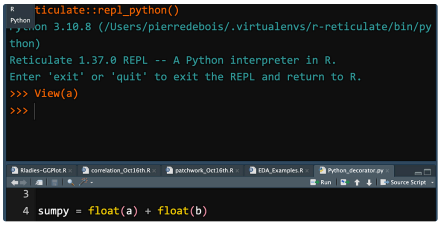
Arngrímur Einarsson

Let's Talk About Churn—Survival Analysis in Data Science

One of the crucial task a data scientist has to deal with is to predict when an event will...

Oct 22 5

Sean Ma in Python in Plain English



Pierre DeBois

Read CSV Files 10x to 40x Faster Using pyarrow and polars

Learning faster Python fast with Sean

Jun 13 21

A Simple R/Python Switch: A RStudio Update

The latest version of RStudio includes a new R/Python Switch as well as Quarto 1.5

Oct 17 22

See more recommendations