Published in Towards Data Science

Wei-Meng Lee Follow
Aug 8, 2022 · 9 min read · · Listen

Data Cleansing in Polars

Learn how to perform data cleansing in your Polars DataFrames

Photo by Hans-Jurgen Mager on Unsplash

In this article, I will continue our exploration of the Polars DataFrame library. This time, I am going to focus on something that data scientists spend their most time on — *data cleansing*. Data Cleansing is the process of detecting and correcting corrupt values in your dataset. In real life, this seemingly simple process takes a lot of time as most of the data that you encounter is likely to contain missing values, incorrect values, or irrelevant values.

Loading the DataFrame

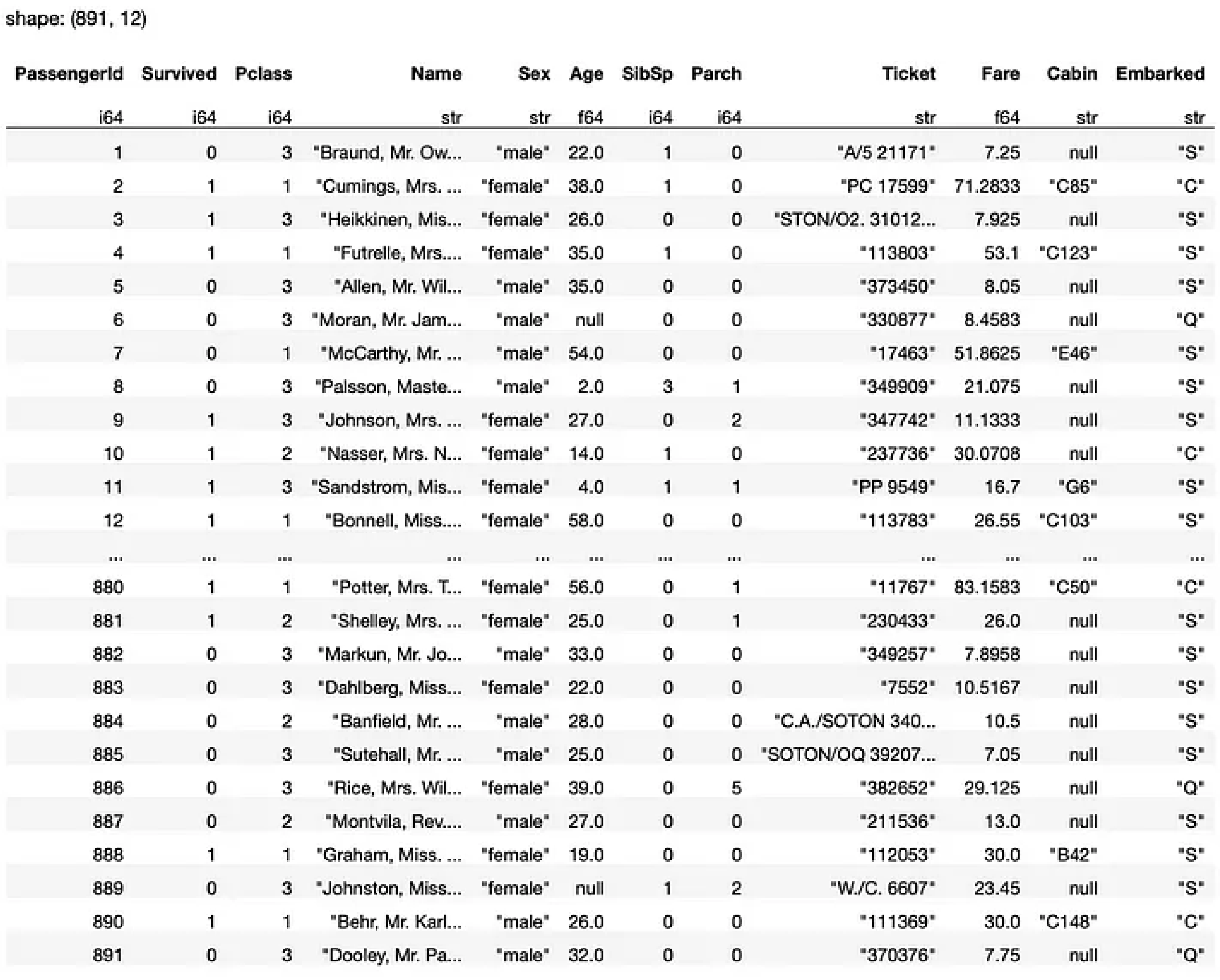
For this article, I will be using the Titanic dataset.

> **Source of Data:** The data source for this article is from https://www.kaggle.com/datasets/tedllh/titanic-train.
>
> **Licensing —** Database Contents License (DbCL) v1.0 https://opendatacommons.org/licenses/dbcl/1-0/

First, load the CSV into a Polars DataFrame:

```
import polars as pl
q = (
    pl.scan_csv('titanic_train.csv')
)
df = q.collect()
df
```

The dataframe contains 891 rows with 12 columns:



```
shape: (891, 12)

PassengerId  Survived  Pclass  Name                 Sex       Age   SibSp  Parch  Ticket                Fare     Cabin   Embarked
i64          i64       i64     str                  str       f64   i64    i64    str                   f64      str     str
1            0         3       "Braund, Mr. Ow...   "male"    22.0  1      0      "A/5 21171"           7.25     null    "S"
2            1         1       "Cumings, Mrs. ...   "female"  38.0  1      0      "PC 17599"            71.2833  "C85"   "C"
3            1         3       "Heikkinen, Mis...   "female"  26.0  0      0      "STON/O2. 31012...    7.925    null    "S"
4            1         1       "Futrelle, Mrs....   "female"  35.0  1      0      "113803"              53.1     "C123"  "S"
5            0         3       "Allen, Mr. Wil...   "male"    35.0  0      0      "373450"              8.05     null    "S"
6            0         3       "Moran, Mr. Jam...   "male"    null  0      0      "330877"              8.4583   null    "Q"
7            0         1       "McCarthy, Mr. ...   "male"    54.0  0      0      "17463"               51.8625  "E46"   "S"
8            0         3       "Palsson, Maste...   "male"    2.0   3      1      "349909"              21.075   null    "S"
9            1         3       "Johnson, Mrs. ...   "female"  27.0  0      2      "347742"              11.1333  null    "S"
10           1         2       "Nasser, Mrs. N...   "female"  14.0  1      0      "237736"              30.0708  null    "C"
11           1         3       "Sandstrom, Mis...   "female"  4.0   1      1      "PP 9549"             16.7     "G6"    "S"
12           1         1       "Bonnell, Miss....   "female"  58.0  0      0      "113783"              26.55    "C103"  "S"
...          ...       ...     ...                  ...       ...   ...    ...    ...                   ...      ...     ...
880          1         1       "Potter, Mrs. T...   "female"  56.0  0      1      "11767"               83.1583  "C50"   "C"
881          1         2       "Shelley, Mrs. ...   "female"  25.0  0      1      "230433"              26.0     null    "S"
882          0         3       "Markun, Mr. Jo...   "male"    33.0  0      0      "349257"              7.8958   null    "S"
883          0         3       "Dahlberg, Miss...   "female"  22.0  0      0      "7552"                10.5167  null    "S"
884          0         2       "Banfield, Mr. ...   "male"    28.0  0      0      "C.A./SOTON 340...    10.5     null    "S"
885          0         3       "Sutehall, Mr. ...   "male"    25.0  0      0      "SOTON/OQ 39207...    7.05     null    "S"
886          0         3       "Rice, Mrs. Wil...   "female"  39.0  0      5      "382652"              29.125   null    "Q"
887          0         2       "Montvila, Rev....   "male"    27.0  0      0      "211536"              13.0     null    "S"
888          1         1       "Graham, Miss. ...   "female"  19.0  0      0      "112053"              30.0     "B42"   "S"
889          0         3       "Johnston, Miss...   "female"  null  1      2      "W./C. 6607"          23.45    null    "S"
890          1         1       "Behr, Mr. Karl...   "male"    26.0  0      0      "111369"              30.0     "C148"  "C"
891          0         3       "Dooley, Mr. Pa...   "male"    32.0  0      0      "370376"              7.75     null    "Q"
```

Image by author

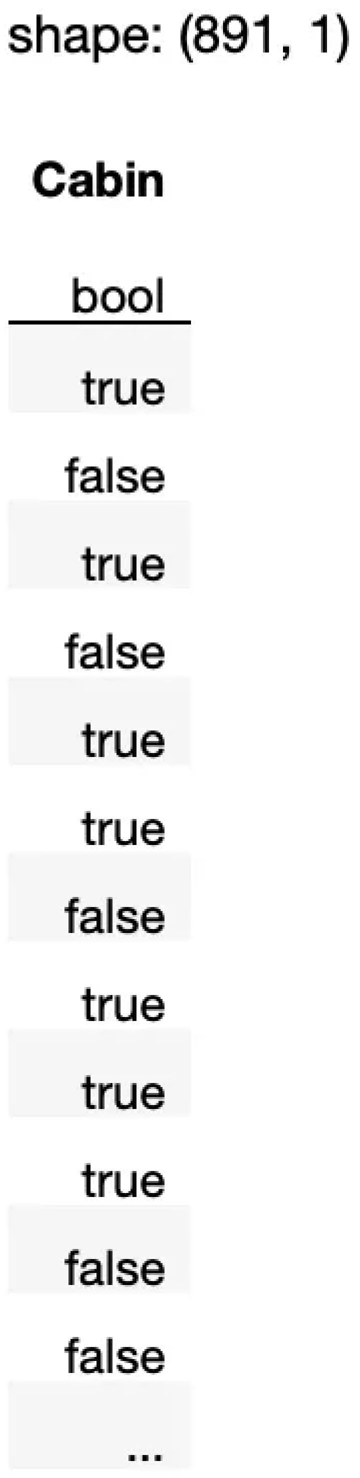
> *All missing values in the CSV file will be loaded as `null` in the Polars DataFrame.*

Looking for Null Values

To check for null values in a specific column, use the `select()` method to select the column and then call the `is_null()` method:

```
df.select(
    pl.col('Cabin').is_null()
)
```

The `is_null()` method returns the result as a DataFrame of boolean values:



```
shape: (891, 1)
Cabin
bool
true
false
true
false
true
true
false
true
true
true
false
false
...
```

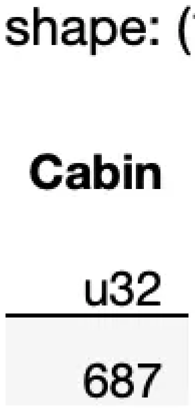
Image by author

Counting the Number of Null Values

It would be more useful to know how many rows in the column has null values, rather than seeing a DataFrame full of boolean values. Hence, you can use the `sum()` method:

```
df.select(
    pl.col('Cabin').is_null().sum()
)
```

As you can see from the result below, the **Cabin** column has 687 null values:



```
shape: (1, 1)
Cabin
u32
687
```

Image by author

> *Difference between* `sum()` *vs* `count()` *— the* `sum()` *method only sums those values that are* `true` *; whereas the* `count()` *method sums all values, including* `false` *.*

If you want to know if *any* of the values in the column contains null, use the `any()` method:

```
df.select(
    pl.col('Cabin').is_null().any()   # returns true
)
```

Likewise, to see if *all* the values in the columns are null, use the `all()` method:

```
df.select(
    pl.col('Cabin').is_null().all()  # returns false
)
```

Counting the Number of Null Values for Each Column in the DataFrame

Often, you want to have a quick glance of which columns in a DataFrame contains null values, instead of checking each column one by one. As such, it would be useful to iterate through the columns and print the results. You can fetch all the columns in a DataFrame by calling the `get_columns()` method and then iterating through it:

```
for col in df.get_columns():
    print(f'{col.name} - {col.is_null().sum()}')
```

The above code snippet prints out the following output:

```
PassengerId - 0
Survived - 0
Pclass - 0
Name - 0
Sex - 0
Age - 177
SibSp - 0
Parch - 0
Ticket - 0
Fare - 0
Cabin - 687
Embarked - 2
```

You can see that the following columns contain null values:

- **Age**
- **Cabin**
- **Embarked**

Replacing Null Values

Once you have determined which columns contain null values, the next logical step would be to:

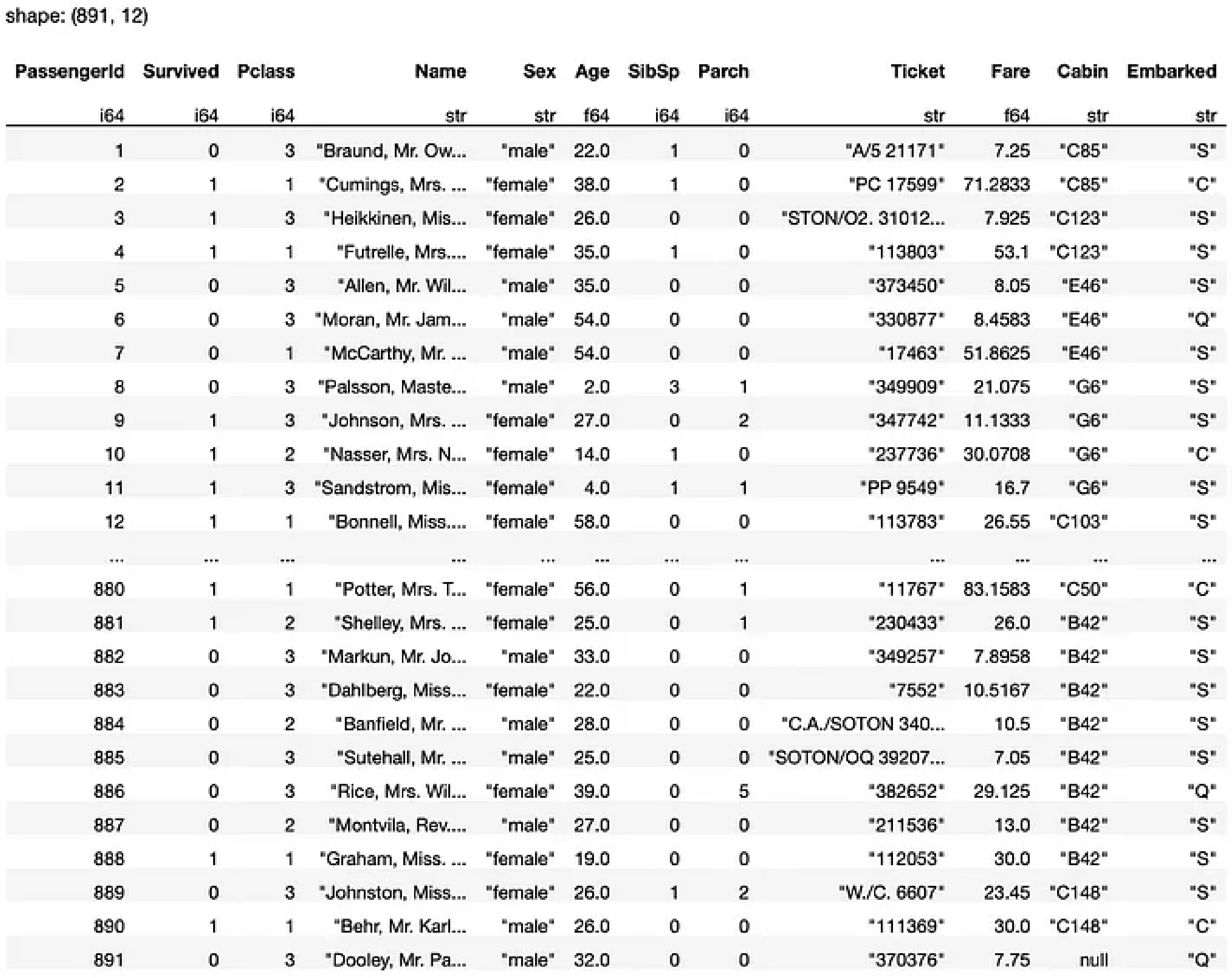
- Fill in the null values with some other values
- Remove rows that contain null values

Filling in the entire dataframe

You can fill the null values in the entire dataframe using the `fill_null()` method:

```
df.fill_null(strategy='backward')
```

In the above statement, I used the *backward-fill* strategy, where all the null values are filled with the *next* non-null value:



shape: (891, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
1	0	3	"Braund, Mr. Ow...	"male"	22.0	1	0	"A/5 21171"	7.25	"C85"	"S"
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C85"	"C"
3	1	3	"Heikkinen, Mis...	"female"	26.0	0	0	"STON/O2. 31012...	7.925	"C123"	"S"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"C123"	"S"
5	0	3	"Allen, Mr. Wil...	"male"	35.0	0	0	"373450"	8.05	"E46"	"S"
6	0	3	"Moran, Mr. Jam...	"male"	54.0	0	0	"330877"	8.4583	"E46"	"Q"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"E46"	"S"
8	0	3	"Palsson, Maste...	"male"	2.0	3	1	"349909"	21.075	"G6"	"S"
9	1	3	"Johnson, Mrs. ...	"female"	27.0	0	2	"347742"	11.1333	"G6"	"S"
10	1	2	"Nasser, Mrs. N...	"female"	14.0	1	0	"237736"	30.0708	"G6"	"C"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"G6"	"S"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"C103"	"S"
...	...	...	...	...	...	...	...	...	...	...	...
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C50"	"C"
881	1	2	"Shelley, Mrs. ...	"female"	25.0	0	1	"230433"	26.0	"B42"	"S"
882	0	3	"Markun, Mr. Jo...	"male"	33.0	0	0	"349257"	7.8958	"B42"	"S"
883	0	3	"Dahlberg, Miss...	"female"	22.0	0	0	"7552"	10.5167	"B42"	"S"
884	0	2	"Banfield, Mr. ...	"male"	28.0	0	0	"C.A./SOTON 340...	10.5	"B42"	"S"
885	0	3	"Sutehall, Mr. ...	"male"	25.0	0	0	"SOTON/OQ 39207...	7.05	"B42"	"S"
886	0	3	"Rice, Mrs. Wil...	"female"	39.0	0	5	"382652"	29.125	"B42"	"Q"
887	0	2	"Montvila, Rev....	"male"	27.0	0	0	"211536"	13.0	"B42"	"S"
888	1	1	"Graham, Miss. ...	"female"	19.0	0	0	"112053"	30.0	"B42"	"S"
889	0	3	"Johnston, Miss...	"female"	26.0	1	2	"W./C. 6607"	23.45	"C148"	"S"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	0	"111369"	30.0	"C148"	"C"
891	0	3	"Dooley, Mr. Pa...	"male"	32.0	0	0	"370376"	7.75	null	"Q"

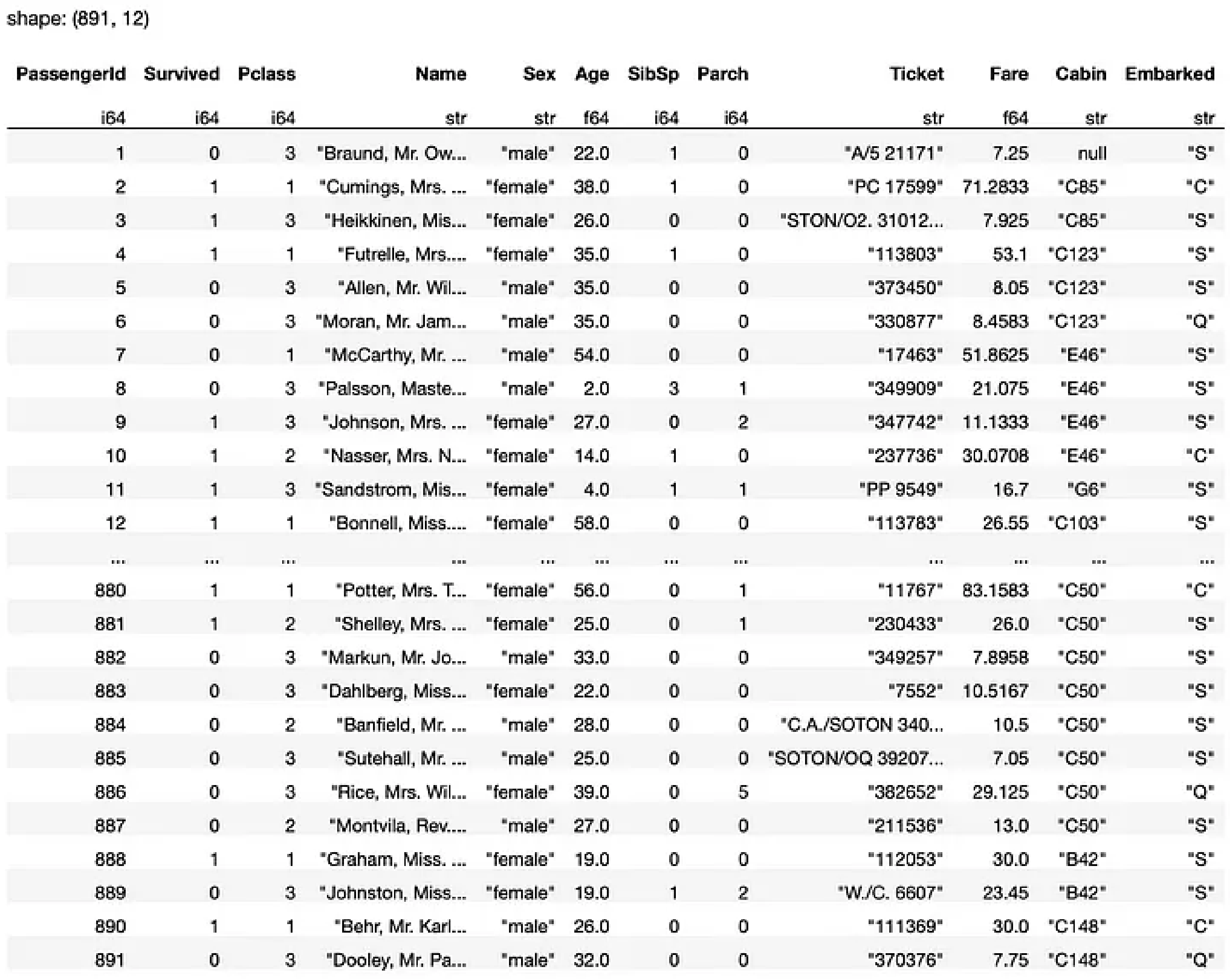
Image by author

> *Notice that the last row contains a null value for the* ***Cabin*** *column. This is because it does not have a next row for it to reference a value to fill.*

You can also use the *forward-fill* strategy, where all null values are filled with the *previous* non-null values:

```
df.fill_null(strategy='forward')
```

The output for the above statement is as shown below:



shape: (891, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
1	0	3	"Braund, Mr. Ow...	"male"	22.0	1	0	"A/5 21171"	7.25	null	"S"
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C85"	"C"
3	1	3	"Heikkinen, Mis...	"female"	26.0	0	0	"STON/O2. 31012...	7.925	"C85"	"S"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"C123"	"S"
5	0	3	"Allen, Mr. Wil...	"male"	35.0	0	0	"373450"	8.05	"C123"	"S"
6	0	3	"Moran, Mr. Jam...	"male"	35.0	0	0	"330877"	8.4583	"C123"	"Q"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"E46"	"S"
8	0	3	"Palsson, Maste...	"male"	2.0	3	1	"349909"	21.075	"E46"	"S"
9	1	3	"Johnson, Mrs. ...	"female"	27.0	0	2	"347742"	11.1333	"E46"	"S"
10	1	2	"Nasser, Mrs. N...	"female"	14.0	1	0	"237736"	30.0708	"E46"	"C"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"G6"	"S"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"C103"	"S"
...	...	...	...	...	...	...	...	...	...	...	...
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C50"	"C"
881	1	2	"Shelley, Mrs. ...	"female"	25.0	0	1	"230433"	26.0	"C50"	"S"
882	0	3	"Markun, Mr. Jo...	"male"	33.0	0	0	"349257"	7.8958	"C50"	"S"
883	0	3	"Dahlberg, Miss...	"female"	22.0	0	0	"7552"	10.5167	"C50"	"S"
884	0	2	"Banfield, Mr. ...	"male"	28.0	0	0	"C.A./SOTON 340...	10.5	"C50"	"S"
885	0	3	"Sutehall, Mr. ...	"male"	25.0	0	0	"SOTON/OQ 39207...	7.05	"C50"	"S"
886	0	3	"Rice, Mrs. Wil...	"female"	39.0	0	5	"382652"	29.125	"C50"	"Q"
887	0	2	"Montvila, Rev....	"male"	27.0	0	0	"211536"	13.0	"C50"	"S"
888	1	1	"Graham, Miss. ...	"female"	19.0	0	0	"112053"	30.0	"B42"	"S"
889	0	3	"Johnston, Miss...	"female"	19.0	1	2	"W./C. 6607"	23.45	"B42"	"S"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	0	"111369"	30.0	"C148"	"C"
891	0	3	"Dooley, Mr. Pa...	"male"	32.0	0	0	"370376"	7.75	"C148"	"Q"

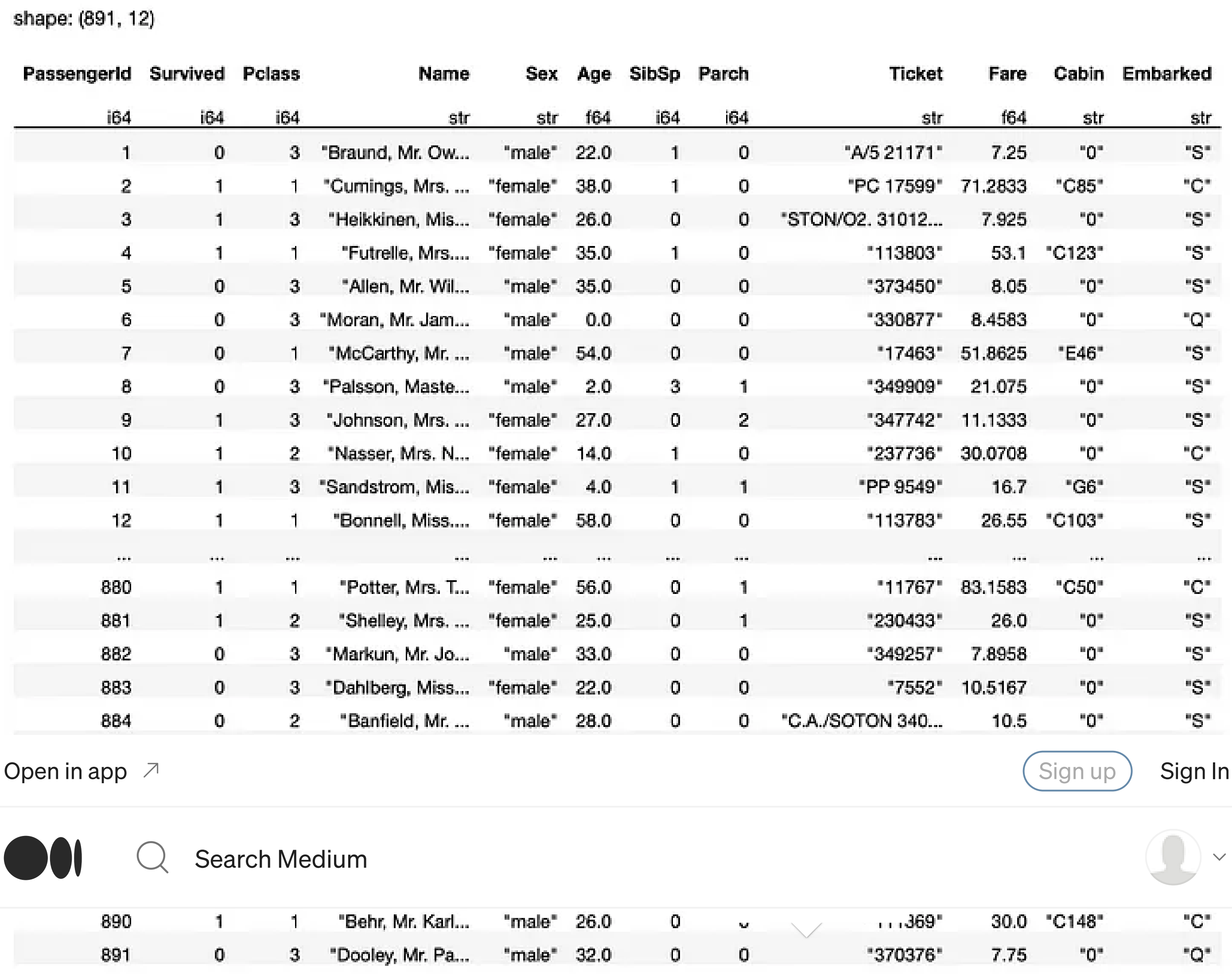
Image by author

> *Now you will realize that the Cabin value for the first row is* `null` *as it does not have a previous row to reference the value to fill.*

You can also fill the null values with a fixed value, such as 0:

```
df.fill_null(0)
```

The output for the above statement is as shown below:



shape: (891, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
1	0	3	"Braund, Mr. Ow...	"male"	22.0	1	0	"A/5 21171"	7.25	"0"	"S"
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C85"	"C"
3	1	3	"Heikkinen, Mis...	"female"	26.0	0	0	"STON/O2. 31012...	7.925	"0"	"S"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"C123"	"S"
5	0	3	"Allen, Mr. Wil...	"male"	35.0	0	0	"373450"	8.05	"0"	"S"
6	0	3	"Moran, Mr. Jam...	"male"	0.0	0	0	"330877"	8.4583	"0"	"Q"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"E46"	"S"
8	0	3	"Palsson, Maste...	"male"	2.0	3	1	"349909"	21.075	"0"	"S"
9	1	3	"Johnson, Mrs. ...	"female"	27.0	0	2	"347742"	11.1333	"0"	"S"
10	1	2	"Nasser, Mrs. N...	"female"	14.0	1	0	"237736"	30.0708	"0"	"C"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"G6"	"S"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"C103"	"S"
...	...	...	...	...	...	...	...	...	...	...	...
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C50"	"C"
881	1	2	"Shelley, Mrs. ...	"female"	25.0	0	1	"230433"	26.0	"0"	"S"
882	0	3	"Markun, Mr. Jo...	"male"	33.0	0	0	"349257"	7.8958	"0"	"S"
883	0	3	"Dahlberg, Miss...	"female"	22.0	0	0	"7552"	10.5167	"0"	"S"
884	0	2	"Banfield, Mr. ...	"male"	28.0	0	0	"C.A./SOTON 340...	10.5	"0"	"S"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	[illegible]	[illegible]369"	30.0	"C148"	"C"
891	0	3	"Dooley, Mr. Pa...	"male"	32.0	0	0	"370376"	7.75	"0"	"Q"

Image by author

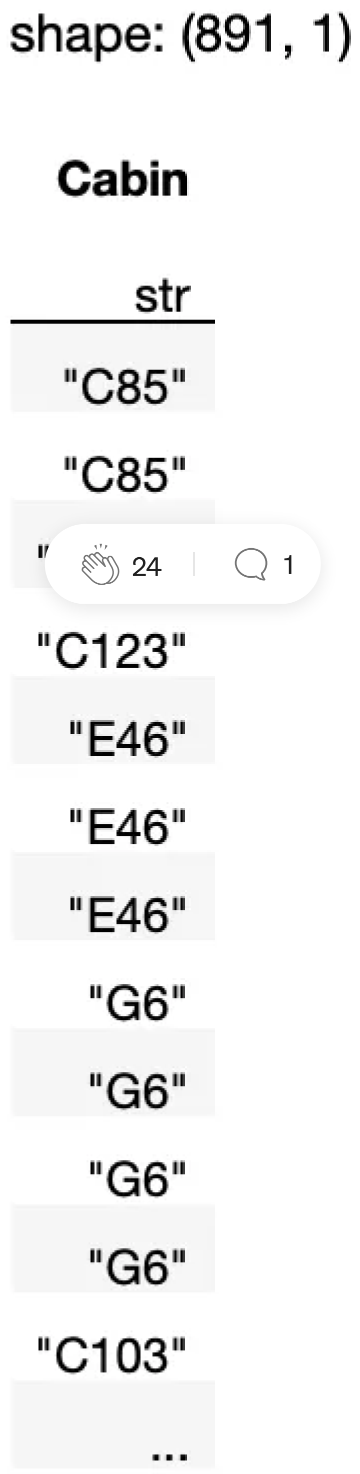
Filling in a specific column

More often than not, you would adopt different fill values for different columns, depending on the data type of the column. For example, you can fill the **Cabin** column using the *backward-fill* strategy:

```
df.select(
    pl.col('Cabin').fill_null(fill_value='backward')
)
```

> *Take note that when you call the `fill_null()` method on an expression, the parameter to set for the strategy is `fill_value` and not `strategy`.*

Notice that the above statement returns a dataframe containing of just the **Cabin** column:



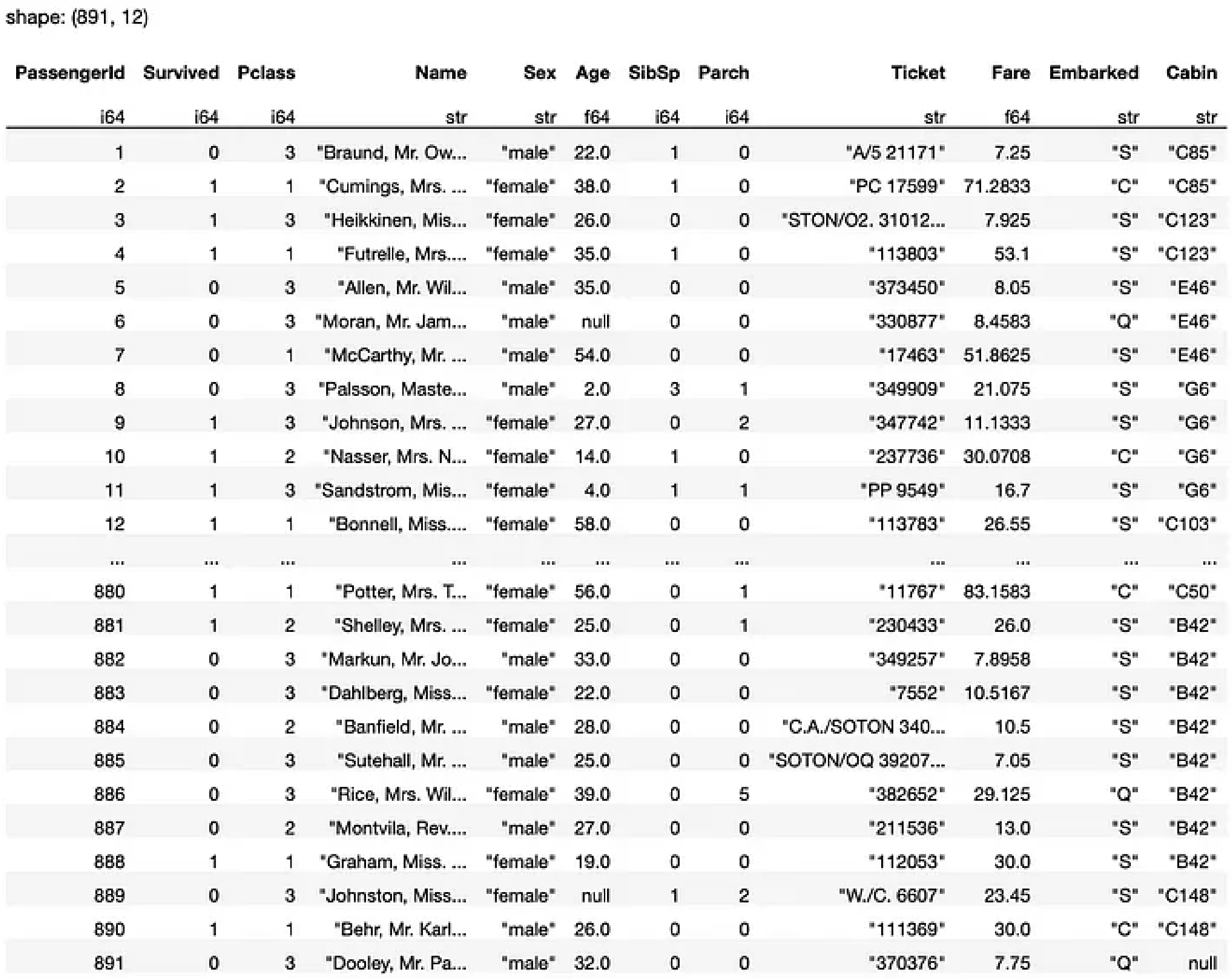
```
shape: (891, 1)

 Cabin
   str
 "C85"
 "C85"
"[illegible]
"C123"
 "E46"
 "E46"
 "E46"
  "G6"
  "G6"
  "G6"
  "G6"
"C103"
   ...
```

Image by author

If you want to include the other columns in the dataframe, add the following statement in bold:

```
df.select(
    [
        pl.exclude('Cabin'),      # select all columns except Cabin
        pl.col('Cabin').fill_null(fill_value='backward')
    ]
)
```

The entire dataframe is now returned:



shape: (891, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Cabin
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
1	0	3	"Braund, Mr. Ow...	"male"	22.0	1	0	"A/5 21171"	7.25	"S"	"C85"
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C"	"C85"
3	1	3	"Heikkinen, Mis...	"female"	26.0	0	0	"STON/O2. 31012...	7.925	"S"	"C123"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"S"	"C123"
5	0	3	"Allen, Mr. Wil...	"male"	35.0	0	0	"373450"	8.05	"S"	"E46"
6	0	3	"Moran, Mr. Jam...	"male"	null	0	0	"330877"	8.4583	"Q"	"E46"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"S"	"E46"
8	0	3	"Palsson, Maste...	"male"	2.0	3	1	"349909"	21.075	"S"	"G6"
9	1	3	"Johnson, Mrs. ...	"female"	27.0	0	2	"347742"	11.1333	"S"	"G6"
10	1	2	"Nasser, Mrs. N...	"female"	14.0	1	0	"237736"	30.0708	"C"	"G6"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"S"	"G6"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"S"	"C103"
...	...	...	...	...	...	...	...	...	...	...	...
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C"	"C50"
881	1	2	"Shelley, Mrs. ...	"female"	25.0	0	1	"230433"	26.0	"S"	"B42"
882	0	3	"Markun, Mr. Jo...	"male"	33.0	0	0	"349257"	7.8958	"S"	"B42"
883	0	3	"Dahlberg, Miss...	"female"	22.0	0	0	"7552"	10.5167	"S"	"B42"
884	0	2	"Banfield, Mr. ...	"male"	28.0	0	0	"C.A./SOTON 340...	10.5	"S"	"B42"
885	0	3	"Sutehall, Mr. ...	"male"	25.0	0	0	"SOTON/OQ 39207...	7.05	"S"	"B42"
886	0	3	"Rice, Mrs. Wil...	"female"	39.0	0	5	"382652"	29.125	"Q"	"B42"
887	0	2	"Montvila, Rev....	"male"	27.0	0	0	"211536"	13.0	"S"	"B42"
888	1	1	"Graham, Miss. ...	"female"	19.0	0	0	"112053"	30.0	"S"	"B42"
889	0	3	"Johnston, Miss...	"female"	null	1	2	"W./C. 6607"	23.45	"S"	"C148"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	0	"111369"	30.0	"C"	"C148"
891	0	3	"Dooley, Mr. Pa...	"male"	32.0	0	0	"370376"	7.75	"Q"	null

Image by author

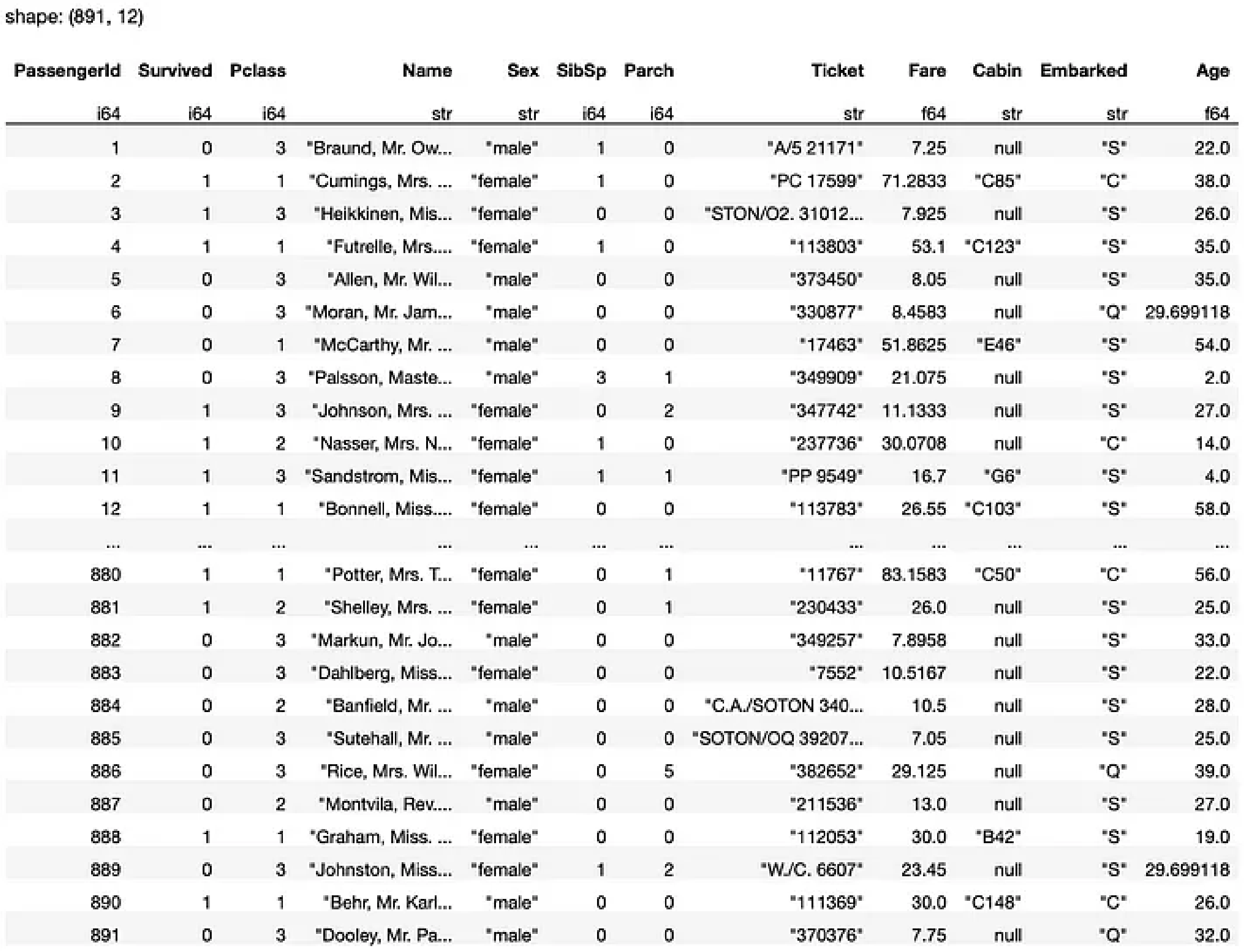
You can also replace the null values in a specific columns with a fixed value:

```
df.select(
    [
        pl.exclude('Age'),
        pl.col('Age').fill_null(fill_value=0)
    ]
)
```

Or replace them with the mean of the column:

```
df.select(
    [
        pl.exclude('Age'),
        pl.col('Age').fill_null(fill_value=pl.col('Age').mean())
    ]
)
```

The above code snippet returns the following result:



```
shape: (891, 12)

PassengerId Survived Pclass                Name      Sex SibSp Parch              Ticket    Fare   Cabin Embarked       Age
        i64      i64    i64                 str      str   i64   i64                 str     f64     str      str       f64
          1        0      3  "Braund, Mr. Ow...   "male"     1     0        "A/5 21171"     7.25    null      "S"      22.0
          2        1      1  "Cumings, Mrs. ... "female"     1     0        "PC 17599"   71.2833   "C85"      "C"      38.0
          3        1      3  "Heikkinen, Mis... "female"     0     0  "STON/O2. 31012...    7.925    null      "S"      26.0
          4        1      1   "Futrelle, Mrs.... "female"    1     0          "113803"     53.1  "C123"      "S"      35.0
          5        0      3    "Allen, Mr. Wil...   "male"   0     0          "373450"     8.05    null      "S"      35.0
          6        0      3  "Moran, Mr. Jam...   "male"     0     0          "330877"   8.4583    null      "Q" 29.699118
          7        0      1   "McCarthy, Mr. ...   "male"    0     0           "17463"  51.8625   "E46"      "S"      54.0
          8        0      3  "Palsson, Maste...   "male"     3     1          "349909"   21.075    null      "S"       2.0
          9        1      3   "Johnson, Mrs. ... "female"    0     2          "347742"  11.1333    null      "S"      27.0
         10        1      2   "Nasser, Mrs. N... "female"    1     0          "237736"  30.0708    null      "C"      14.0
         11        1      3 "Sandstrom, Mis... "female"      1     1         "PP 9549"     16.7    "G6"      "S"       4.0
         12        1      1    "Bonnell, Miss.... "female"   0     0          "113783"    26.55  "C103"      "S"      58.0
        ...      ...    ...                 ...      ...   ...   ...                 ...      ...     ...      ...       ...
        880        1      1     "Potter, Mrs. T... "female"  0     1           "11767"  83.1583   "C50"      "C"      56.0
        881        1      2    "Shelley, Mrs. ... "female"   0     1          "230433"     26.0    null      "S"      25.0
        882        0      3   "Markun, Mr. Jo...   "male"    0     0          "349257"   7.8958    null      "S"      33.0
        883        0      3  "Dahlberg, Miss... "female"     0     0            "7552"  10.5167    null      "S"      22.0
        884        0      2    "Banfield, Mr. ...   "male"   0     0  "C.A./SOTON 340...     10.5    null      "S"      28.0
        885        0      3    "Sutehall, Mr. ...   "male"   0     0 "SOTON/OQ 39207...     7.05    null      "S"      25.0
        886        0      3    "Rice, Mrs. Wil... "female"   0     5          "382652"   29.125    null      "Q"      39.0
        887        0      2    "Montvila, Rev....   "male"   0     0          "211536"     13.0    null      "S"      27.0
        888        1      1  "Graham, Miss. ... "female"     0     0          "112053"     30.0   "B42"      "S"      19.0
        889        0      3  "Johnston, Miss... "female"     1     2        "W./C. 6607"    23.45    null      "S" 29.699118
        890        1      1    "Behr, Mr. Karl...   "male"   0     0          "111369"     30.0  "C148"      "C"      26.0
        891        0      3   "Dooley, Mr. Pa...   "male"    0     0          "370376"     7.75    null      "Q"      32.0
```

Image by author

What if you want to replace a null value with the most frequently occurring value in the column? For example, for the **Embarked** column you want to replace the null values with the port that most passengers embarked on. In this case you can use the `mode()` method:

```
df.select(
    [
        pl.exclude('Embarked'),
        pl.col('Embarked').fill_null(
            fill_value=pl.col('Embarked').mode())
    ]
)
```

The `mode()` method returns the most frequently occurring value in a column.

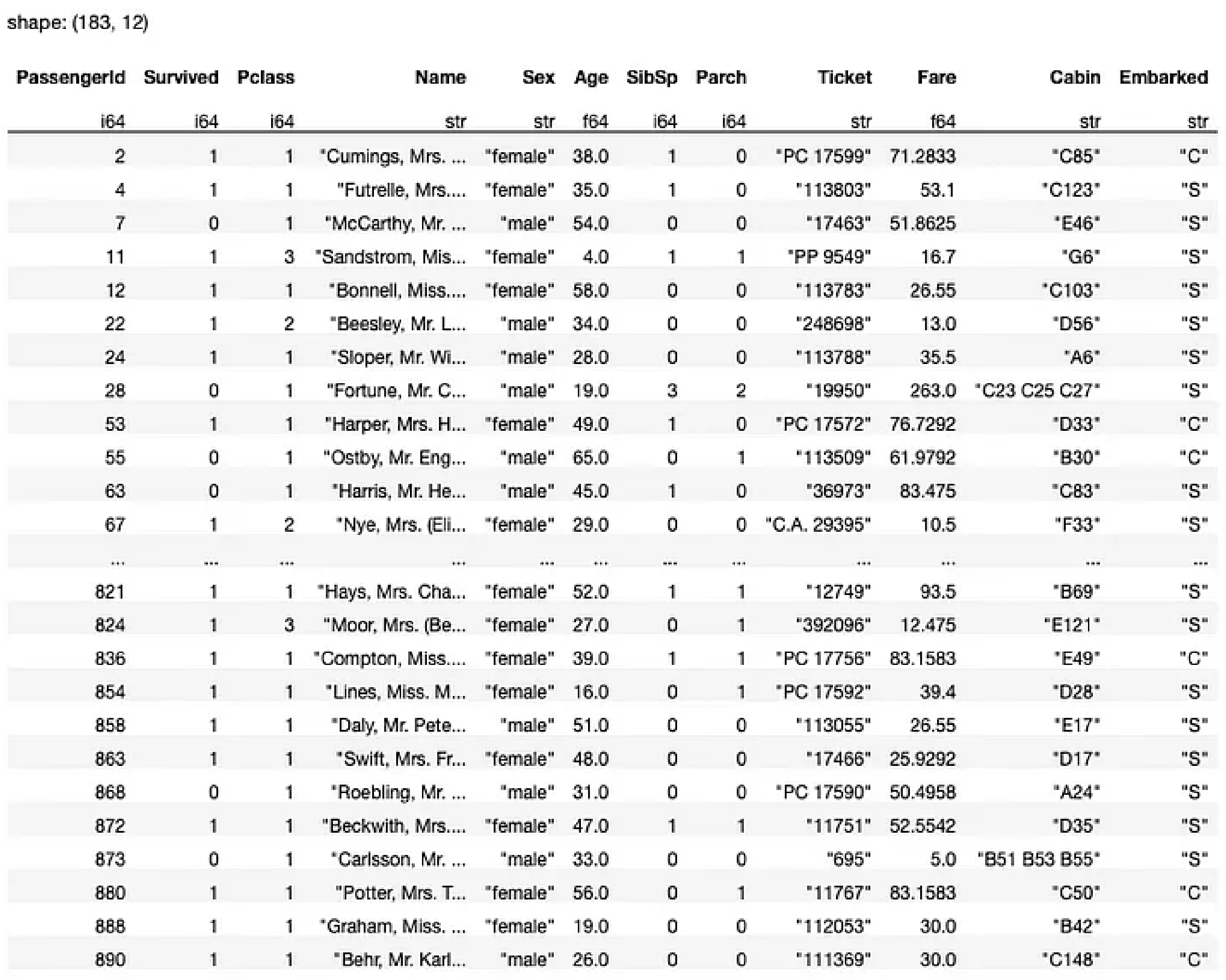
> *Note that `mode()` does not work on floating-point columns.*

Dropping rows and columns

Sometimes it just makes sense to drop rows when there are null values in your dataframe, especially when the number of rows with null values are small relative to the total number of rows you have. To drop all rows in the entire dataframe with null values, use the `drop_nulls()` method:

```
df.drop_nulls()
```

Note that after doing this, the result is left with 183 rows:



shape: (183, 12)

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C85"	"C"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"C123"	"S"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"E46"	"S"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"G6"	"S"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"C103"	"S"
22	1	2	"Beesley, Mr. L...	"male"	34.0	0	0	"248698"	13.0	"D56"	"S"
24	1	1	"Sloper, Mr. Wi...	"male"	28.0	0	0	"113788"	35.5	"A6"	"S"
28	0	1	"Fortune, Mr. C...	"male"	19.0	3	2	"19950"	263.0	"C23 C25 C27"	"S"
53	1	1	"Harper, Mrs. H...	"female"	49.0	1	0	"PC 17572"	76.7292	"D33"	"C"
55	0	1	"Ostby, Mr. Eng...	"male"	65.0	0	1	"113509"	61.9792	"B30"	"C"
63	0	1	"Harris, Mr. He...	"male"	45.0	1	0	"36973"	83.475	"C83"	"S"
67	1	2	"Nye, Mrs. (Eli...	"female"	29.0	0	0	"C.A. 29395"	10.5	"F33"	"S"
...	...	...	...	...	...	...	...	...	...	...	...
821	1	1	"Hays, Mrs. Cha...	"female"	52.0	1	1	"12749"	93.5	"B69"	"S"
824	1	3	"Moor, Mrs. (Be...	"female"	27.0	0	1	"392096"	12.475	"E121"	"S"
836	1	1	"Compton, Miss....	"female"	39.0	1	1	"PC 17756"	83.1583	"E49"	"C"
854	1	1	"Lines, Miss. M...	"female"	16.0	0	1	"PC 17592"	39.4	"D28"	"S"
858	1	1	"Daly, Mr. Pete...	"male"	51.0	0	0	"113055"	26.55	"E17"	"S"
863	1	1	"Swift, Mrs. Fr...	"female"	48.0	0	0	"17466"	25.9292	"D17"	"S"
868	0	1	"Roebling, Mr. ...	"male"	31.0	0	0	"PC 17590"	50.4958	"A24"	"S"
872	1	1	"Beckwith, Mrs....	"female"	47.0	1	1	"11751"	52.5542	"D35"	"S"
873	0	1	"Carlsson, Mr. ...	"male"	33.0	0	0	"695"	5.0	"B51 B53 B55"	"S"
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C50"	"C"
888	1	1	"Graham, Miss. ...	"female"	19.0	0	0	"112053"	30.0	"B42"	"S"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	0	"111369"	30.0	"C148"	"C"

Image by author

In our dataset, it is not really a good idea to do this as the **Cabin** column has the most number of null values. Hence you should really drop the **Cabin** column instead, like this:

```
df.select(
     pl.exclude('Cabin')
)
```

You can also use the `drop()` method to drop one or more columns:

```
df.drop(['Cabin'])            # drop the Cabin column
df.drop(['Ticket','Fare'])    # drop the Ticket and Fare columns
```

Notice that the `drop()` method does not modify the original dataframe — it simply returns the dataframe with the specified column(s) dropped. If you want to modify the original dataframe, use the `drop_in_place()` method:

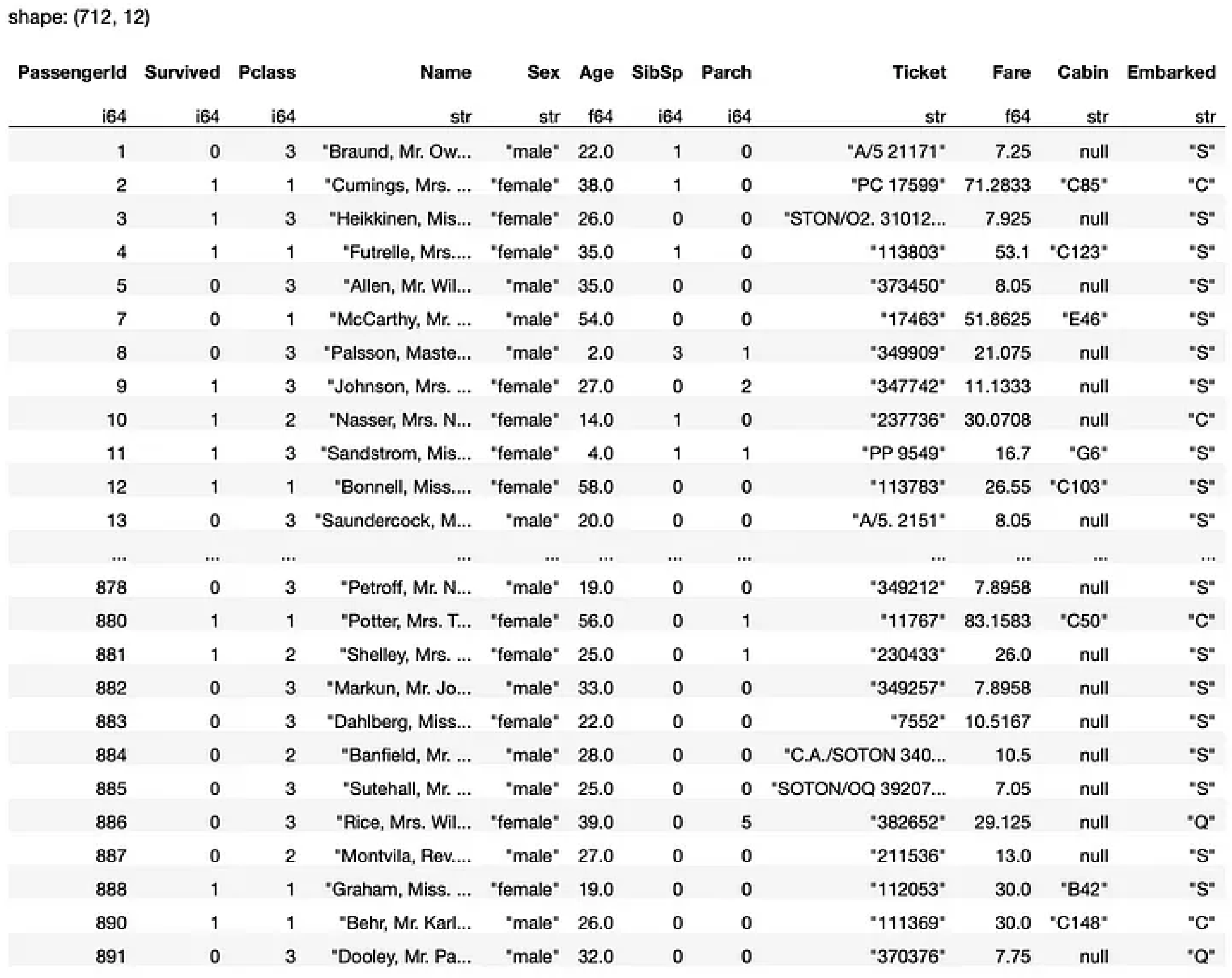
```
df.drop_in_place('Ticket')
```

> *Note that the `drop_in_place()` method can only drop a single column. It returns the dropped column as a Polars Series.*

For the `drop_nulls()` method, you can also drop rows based on specific columns, using the `subset` parameter:

```
df.drop_nulls(subset=['Age','Embarked'])
```

In this case, only rows with null values in the **Age** and **Embarked** columns will be dropped:



shape: (712, 12)

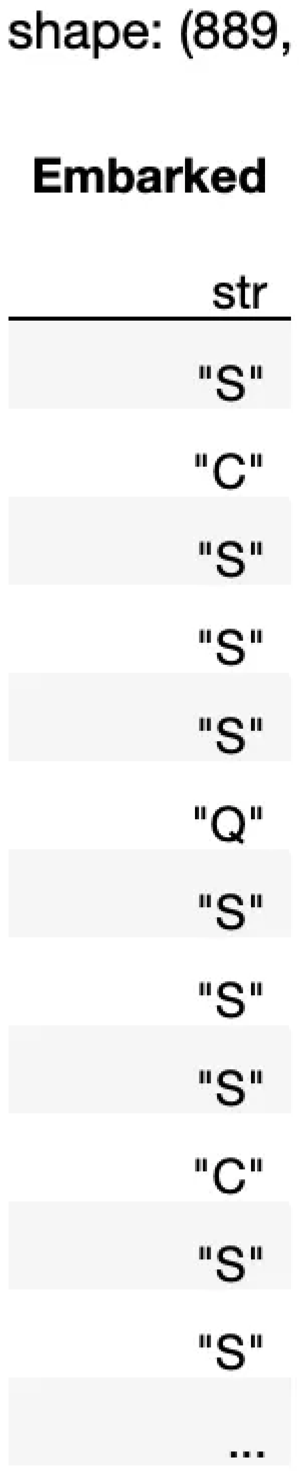
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
i64	i64	i64	str	str	f64	i64	i64	str	f64	str	str
1	0	3	"Braund, Mr. Ow...	"male"	22.0	1	0	"A/5 21171"	7.25	null	"S"
2	1	1	"Cumings, Mrs. ...	"female"	38.0	1	0	"PC 17599"	71.2833	"C85"	"C"
3	1	3	"Heikkinen, Mis...	"female"	26.0	0	0	"STON/O2. 31012...	7.925	null	"S"
4	1	1	"Futrelle, Mrs....	"female"	35.0	1	0	"113803"	53.1	"C123"	"S"
5	0	3	"Allen, Mr. Wil...	"male"	35.0	0	0	"373450"	8.05	null	"S"
7	0	1	"McCarthy, Mr. ...	"male"	54.0	0	0	"17463"	51.8625	"E46"	"S"
8	0	3	"Palsson, Maste...	"male"	2.0	3	1	"349909"	21.075	null	"S"
9	1	3	"Johnson, Mrs. ...	"female"	27.0	0	2	"347742"	11.1333	null	"S"
10	1	2	"Nasser, Mrs. N...	"female"	14.0	1	0	"237736"	30.0708	null	"C"
11	1	3	"Sandstrom, Mis...	"female"	4.0	1	1	"PP 9549"	16.7	"G6"	"S"
12	1	1	"Bonnell, Miss....	"female"	58.0	0	0	"113783"	26.55	"C103"	"S"
13	0	3	"Saundercock, M...	"male"	20.0	0	0	"A/5. 2151"	8.05	null	"S"
...	...	...	...	...	...	...	...	...	...	...	...
878	0	3	"Petroff, Mr. N...	"male"	19.0	0	0	"349212"	7.8958	null	"S"
880	1	1	"Potter, Mrs. T...	"female"	56.0	0	1	"11767"	83.1583	"C50"	"C"
881	1	2	"Shelley, Mrs. ...	"female"	25.0	0	1	"230433"	26.0	null	"S"
882	0	3	"Markun, Mr. Jo...	"male"	33.0	0	0	"349257"	7.8958	null	"S"
883	0	3	"Dahlberg, Miss...	"female"	22.0	0	0	"7552"	10.5167	null	"S"
884	0	2	"Banfield, Mr. ...	"male"	28.0	0	0	"C.A./SOTON 340...	10.5	null	"S"
885	0	3	"Sutehall, Mr. ...	"male"	25.0	0	0	"SOTON/OQ 39207...	7.05	null	"S"
886	0	3	"Rice, Mrs. Wil...	"female"	39.0	0	5	"382652"	29.125	null	"Q"
887	0	2	"Montvila, Rev....	"male"	27.0	0	0	"211536"	13.0	null	"S"
888	1	1	"Graham, Miss. ...	"female"	19.0	0	0	"112053"	30.0	"B42"	"S"
890	1	1	"Behr, Mr. Karl...	"male"	26.0	0	0	"111369"	30.0	"C148"	"C"
891	0	3	"Dooley, Mr. Pa...	"male"	32.0	0	0	"370376"	7.75	null	"Q"

Image by author

You can also call the `drop_nulls()` method directly on specific columns:

```
df.select(
    pl.col(['Embarked']).drop_nulls()
)
```

In this case, the result will be a dataframe containing the **Embarked** column with the null values removed:



shape: (889, 1)

Embarked
str
"S"
"C"
"S"
"S"
"S"
"Q"
"S"
"S"
"S"
"C"
"S"
"S"
...

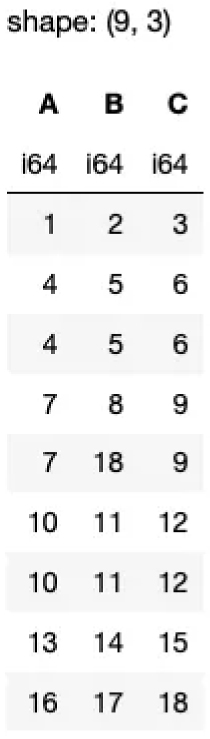
Image by author

Removing Duplicate Values

The final data cleansing technique that I will discuss in this article is that of removing duplicates. For this example, I will manually create a Polars DataFrame:

```
import polars as pl
```

```
df = pl.DataFrame(
    {
        "A": [1,4,4,7,7,10,10,13,16],
        "B": [2,5,5,8,18,11,11,14,17],
        "C": [3,6,6,9,9,12,12,15,18]
    }
)
df
```



shape: (9, 3)

A	B	C
i64	i64	i64
1	2	3
4	5	6
4	5	6
7	8	9
7	18	9
10	11	12
10	11	12
13	14	15
16	17	18

Image by author

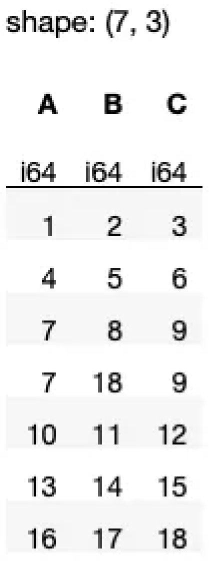
Observe that there are a few duplicated rows, e.g. 4,5,6 as well as 10,11,12. Additionally, there are rows where values for columns A and C are duplicates, e.g. 7,9.

Using the unique() method

First, let's remove the duplicates using the `unique()` method:

```
df.unique()
```

> *You can also use the `distinct()` method. However, it has since been deprecated in version 0.13.13 of Polars and hence it is not recommended to use it.*


shape: (7, 3)

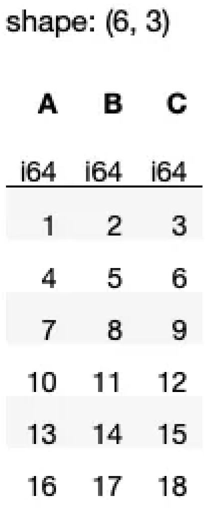
A	B	C
i64	i64	i64
1	2	3
4	5	6
7	8	9
7	18	9
10	11	12
13	14	15
16	17	18

Image by author

Observe from the output that if you don't supply any argument to the `distinct()` method, all the duplicating rows will be removed and only one row will be kept.

You can also remove duplicates based on specific columns using the `subset` parameter:

```
df.unique(subset=['A','C'], keep='first')
```


shape: (6, 3)

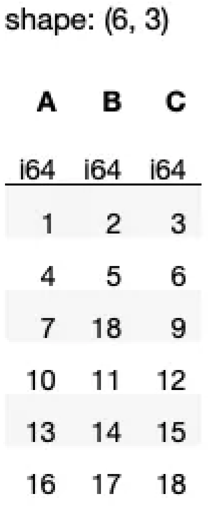
A	B	C
i64	i64	i64
1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18

Image by author

In the above result, observe that the row 7,8,9 is kept while the next row 7,18,9 is removed. This is because for these two rows, they have duplicate values for columns A and C. The `keep='first'` argument (default argument value) keeps the first duplicate row and removes the rest.

If you want to keep the last duplicate row, set `keep` to `'last'`:

```
df.unique(subset=['A','C'], keep='last')
```


shape: (6, 3)

A	B	C
i64	i64	i64
1	2	3
4	5	6
7	18	9
10	11	12
13	14	15
16	17	18

Image by author

Observe that the row 7,18,9 will now be kept.

Removing all duplicate rows

What about removing all duplicate rows in Polars? Unlike in Pandas where you can set the `keep` parameter in the `drop_duplicates()` method to `False` to remove *all* duplicate rows:

```
# Assuming df is a Pandas DataFrame
df.drop_duplicates(keep=False)
```

The `keep` parameter in the `unique()` method in Polars does not accept the `False` value. So if you want to remove duplicate values, you have to do things a little differently.

First, you can use the `is_duplicated()` method to get a Series result indicating which rows in the dataframe are duplicates:

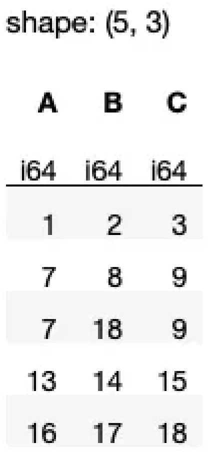
```
df.is_duplicated()
```

You will get a result as follows:

```
shape: (9,)
Series: '' [bool]
[
        false
        true
        true
        false
        false
        true
        true
        false
        false
]
```

To fetch all those non-duplicate rows (essentially dropping all duplicate rows), you can use the *square bracket indexing* method:

```
df[~df.is_duplicated()]
```


shape: (5, 3)

A	B	C
i64	i64	i64
1	2	3
7	8	9
7	18	9
13	14	15
16	17	18

Image by author

But since *square bracket indexing* method is not recommended in Polars, you should use a more Polar-friendly method. You can do this using the `filter()` and `pl.lit()` methods:

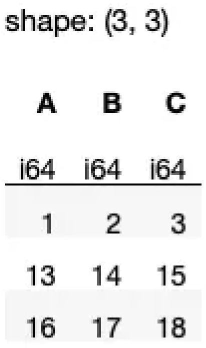
```
df.filter(
    pl.lit(~df.is_duplicated())
)
```

The result will be the same as the previous result.

Finally, if you want to remove all duplicates based on specific columns, you can simply use the `pl.col()` method and pass in the specific column names:

```
df.filter(
    ~pl.col(['A','C']).is_duplicated()
)
```

The above statement generates the following output:



```
shape: (3, 3)

  A    B    C
 i64  i64  i64
-------------
   1    2    3
  13   14   15
  16   17   18
```

Image by author

Join Medium with my referral link - Wei-Meng Lee

Read every story from Wei-Meng Lee (and thousands of other writers on Medium). Your membership fee directly supports…

weimenglee.medium.com

> *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/.*

ML CONFERENCE
THE ONLINE CONFERENCE
WORKSHOP
Workshop: Using Polars For Data Analytics
Wei-Meng Lee, Developer Learning Solutions

Summary

In this article, I have showed you how to:

- Check for null values in your dataframe
- Count the number of null values
- Replace null values using different strategies — fixed value, forward-fill, and backward fill
- Drop rows that contains null values
- Drop entire columns
- Drop duplicate rows

My only wish is for the `keep` parameter in the `unique()` method in Polars to accept the `False` value so that I can simply remove duplicate rows easily without resorting to using the `filter()` method. If you know of a more efficient method, let me know in the comments!

Data Cleansing Polars Dataframe Duplicates Nulls