Python For Data Science | Pandas

Data Cleaning Using Pandas

2 Likes 4.1k views 6 min read Updated on Sep 21, 2022
Prerna-Singh Tech Content Lead 97 Blogs written Save

Data preparation involves data collection and data cleaning. When working with multiple sources of data, there are instances where the collected data could be incorrect, mislabeled, or even duplicated. This would lead to unreliable machine learning models and wrong outcomes. Hence, it is important to clean your data and get it into a usable form beforehand. In this article, we cover the concept of data cleaning using Pandas.

As a data scientist, most of your time is going to be spent preparing your data for analysis. In fact, according to Forbes, data preparation is the 'most time-consuming, least enjoyable data science task'. Naturally, one would want to increase productivity in this phase to move on to the more interesting parts – getting insights from data. Pandas is a very popular Python library mainly used for data pre-processing purposes such as data cleaning, manipulation, and transformation. It provides a quick and efficient way to manage and analyze your data. In this blog on ***data cleaning using Pandas***, we will cover the following sections:

What is Data Cleaning?

Data cleaning is the process of dealing with messy, disordered data and eliminating incorrect, missing, duplicated values in your dataset. It improves the quality and accuracy of the data being fed to the algorithms that will solve your data science problem.

Now, let's get to the fun part, shall we?

Data Cleaning Using Pandas

We are going to perform data cleaning using pandas. The data used in this blog can be found here. This dataset describes the Airbnb listing activity in New York City for the year 2019. It contains information about hosts, geographical availability, and other metrics required to make predictions and draw conclusions. Let's start with preparing this data for it.

Firstly, let's import the Pandas library:

Copy code

```
import pandas as pd
```

Now, let's load the dataset:

Displaying the first 5 rows of the dataset:

Copy code

```
df.head()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80

Copy code

```
df.info()
```

Use **info()** to get information about the dataset:

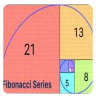
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  object
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

We can see all the 16 columns listed above along with their data types. You can also see the memory usage displayed at the end as 6+ MB.

Let's start with our data cleaning process now –

Finding duplicated values in a DataFrame

- **duplicated()**: This function displays the boolean values in a columnar format. False means no values are duplicated:



Fibonacci Number

4 min read

KNN Algorithm in Machine Learning

8 min read

View all blogs >

Copy code

```
df.duplicated()
```

```
id
2539        False
2595        False
3647        False
3831        False
5022        False
            ...
36484665    False
36485057    False
36485431    False
36485609    False
36487245    False
Length: 48895, dtype: bool
```

Each element of the *'id'* column of the dataset is displayed, showing whether the value is duplicated or not.

But as you can see, there are 48895 elements here and we can't check against each one individually. So, we will use the **any()** function to find out if there are any duplicated values at all:

Copy code

```
df.duplicated().any()
```

```
False
```

So, there are no duplicate values. But if there were, we could've used the following syntax to remove those:

Syntax –

DataFrame.drop_duplicates(*subset=None*, *keep='first'*, *inplace=False*, *ignore_index=False*)

Finding missing elements in a DataFrame

There are four ways to find the null values, if present, in the dataset.

- **isnull()**: This function displays the dataset with boolean values. False means the value is not null:

Copy code

```
df.isnull()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
48890	False	False	False	False	False	False	False	False	False	False
48891	False	False	False	False	False	False	False	False	False	False

- **isna()**: This function also displays the dataset with boolean values. False means the value is not N/A:

Copy code

```
df.isna()
```

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
48890	False	False	False	False	False	False	False	False	False	False
48891	False	False	False	False	False	False	False	False	False	False

- **isna().any()**: This function provides the boolean values too but in a columnar format:

Copy code

```
df.isna().any()
```

```
name                    True
host_id                False
host_name               True
neighbourhood_group    False
neighbourhood          False
latitude               False
longitude              False
room_type              False
price                  False
minimum_nights         False
number_of_reviews      False
last_review             True
reviews_per_month       True
availability_365       False
dtype: bool
```

We can see there are 4 columns with null values present: *'name'*, *'host_name'*, *'last_review'*, and *'reviews_per_month'*.

- **isna().sum()**: This function gives the column-wise sum of the null values present in the dataset.

Copy code

```
df.isna().sum()
```

```
name                      16
host_id                    0
host_name                 21
neighbourhood_group        0
neighbourhood              0
latitude                   0
longitude                  0
room_type                  0
price                      0
minimum_nights             0
number_of_reviews          0
last_review            10052
reviews_per_month      10052
availability_365           0
dtype: int64
```

We can see the number of null values against each of the 4 columns.

Filling the missing values in a DataFrame

- **fillna()**: This function will replace the null values in a DataFrame with the specified values.

Copy code

```
Syntax –
DataFrame.fillna(value, method, axis, inplace, limit, downcast)
```

The *value* parameter can be a dictionary that takes the column names as key.

Let's fill in the values for the *'name', 'host_name'*, and *'last_review'* columns:

Copy code

```
df.fillna({'name':'Not Stated','host_name':'Not Stated','last_review':0},
inplace=True)
```

By default, the method does not make changes to the object directly. Instead, it returns a modified copy of our object. This is avoided by setting the *inplace* parameter.

Do you want to check if the null values got filled? Let's do it using the sum() function for missing values again:

Copy code

```
df.isna().sum()
```

```
id                                0
name                              0
host_id                           0
host_name                         0
neighbourhood_group               0
neighbourhood                     0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimum_nights                    0
number_of_reviews                 0
last_review                       0
reviews_per_month             10052
calculated_host_listings_count    0
availability_365                  0
dtype: int64
```

Can you see? We have successfully removed the null values for the 3 columns!

Now, what shall we do about the *'reviews_per_month'* column?

Dropping columns in a DataFrame

- **drop()**: This function will remove the columns from the DataFrame.

Copy code

```
Syntax –
DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None,
inplace=False, errors='raise')
```

Let's drop the *'reviews_per_month'* column:

Copy code

```
df.drop(['reviews_per_month'], axis = 1, inplace=True)
```

Let's check whether we've dropped it:

Copy code

```
#Display all column names
df.columns
```

```
Index(['id', 'name', 'host_id', 'host_name', 'neighbourhood_group',
       'neighbourhood', 'latitude', 'longitude', 'room_type', 'price',
       'minimum_nights', 'number_of_reviews', 'last_review',
       'calculated_host_listings_count', 'availability_365'],
      dtype='object')
```

We cannot see the *'reviews_per_month'* column here as it has been successfully removed.

Changing the index of a DataFrame

When dealing with data, it is helpful in most cases to use a uniquely valued identifying field of the data as its index.

In our dataset, we can assume that the *'id'* field would serve this purpose. So, let's first check if all the values in this field are unique or not:

Copy code

```
df['id'].is_unique
```

```
True
```

Now that we know that all values in the *'id'* column are unique, let's set this column as the index using **set_index()** function:

Copy code

```
df = df.set_index('id', inplace=True)
df.head()
```

id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149
2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225
3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150
3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89
5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80

You can now access each record directly by using **iloc[]** as shown:

Copy code

```
df.iloc[4]
#Displaying the 5th record from the dataset
```

```
name                              Entire Apt: Spacious Studio/Loft by central park
host_id                                                                       7192
host_name                                                                    Laura
neighbourhood_group                                                      Manhattan
neighbourhood                                                          East Harlem
latitude                                                                  40.79851
longitude                                                                -73.94399
room_type                                                          Entire home/apt
price                                                                           80
minimum_nights                                                                  10
number_of_reviews                                                                9
last_review                                                             2018-11-19
calculated_host_listings_count                                                   1
availability_365                                                                 0
Name: 5022, dtype: object
```

Renaming Columns of a DataFrame

In many cases, you might require renaming the columns for better interpretation.

You can do this by using a dictionary, where the key is the current column name, and the value is the new column name:

Copy code

```
new_col = {'name':'listing_name', 'number_of_reviews':'reviews'}
```

```
df.rename(columns=new_col, inplace=True)
df.head()
```

id	listing_name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149
2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225
3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150
3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89
5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80

Converting the Data Type of Columns

While checking the DataFrame info() above, we saw that the *'last_review'* column was of object type. Let's recall it here:

Copy code

```
df['last_review'].dtype.type
```

```
numpy.object_
```

Since the column contains dates, we are going to convert its data type to datetime as shown:

Copy code

```
df['last_review'] = pd.to_datetime(df['last_review'], format='%Y-%m-%d')
df['last_review'].dtype.type
```

```
numpy.datetime64
```

Converting the Data Type to Reduce Memory Usage

You can reduce memory usage by changing the data types of columns.

Let's do it for the *'host_id'* column. We will convert it from int64 to int32 as shown:

Copy code

```
df['host_id'] = df['host_id'].astype('int32')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48895 entries, 2539 to 36487245
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   listing_name                    48895 non-null  object 
 1   host_id                         48895 non-null  int32  
 2   host_name                       48895 non-null  object 
 3   neighbourhood_group             48895 non-null  object 
 4   neighbourhood                   48895 non-null  object 
 5   latitude                        48895 non-null  float64
 6   longitude                       48895 non-null  float64
 7   room_type                       48895 non-null  object 
 8   price                           48895 non-null  int64  
 9   minimum_nights                  48895 non-null  int64  
 10  reviews                         48895 non-null  int64  
 11  last_review                     48895 non-null  object 
 12  calculated_host_listings_count  48895 non-null  int64  
 13  availability_365                48895 non-null  int64  
dtypes: float64(2), int32(1), int64(5), object(6)
memory usage: 5.4+ MB
```

So, we have reduced the memory usage from 6+ MB to 5.4+ MB.

Data Cleaning in Pandas – Try it yourself

Click the google colab icon below to run the demo in colab.

Open in Colab

In [ ]:
```
import pandas as pd
```

In [ ]:
```
#Importing the dataset by reading the csv file
df = pd.read_csv('AB_NYC_2019.csv')
df.head()
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	nei
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	

data-cleaning-pandas-demo.ipynb hosted with ❤ by GitHub view raw

Endnotes

Pandas is a very powerful data processing tool for the Python programming language. It provides a rich set of functions to process various types of file formats from multiple data sources. The Pandas library is specifically useful for data scientists working with data cleaning and analysis. If you seek to learn the basics and various functions of Pandas, you can explore related articles here.

————————————————————————————-

Prerna-Singh
Tech Content Lead 97 Blogs

Prerna is a Tech enthusiast and former Research analyst. She is currently exploring Machine Learning & Data Science with previous experience in Blockchain & Big Data Analytics.