Member-only story

Data preprocessing with Python Pandas

Part 1 — Missing Data

Angelica Lo Duca · Follow

Published in Towards Data Science · 6 min read · Nov 12, 2020

Photo by Photo Mix from Pixabay

This tutorial explains how to preprocess data using the pandas library. Preprocessing is the process of doing a pre-analysis of data, in order to transform them into a standard and normalized format.

Preprocessing involves the following aspects:

- missing values
- data standardization
- data normalization
- data binning

In this tutorial we deal only with missing values.

You can download the source code of this tutorial as a Jupyter notebook from my Github Data Science Repository.

Import data

In this tutorial we will use the dataset related to Hepatitis, which can be downloaded from this link.

Firstly, import data using the pandas library and convert them into a dataframe. Through the `head(10)` method we print only the first 10 rows of the dataset

```
import pandas as pd
df = pd.read_csv('hepatitis.csv')
df.head(10)
```

	age	sex	steroid	antivirals	fatigue	malaise	anorexia	liver_big	liver_firm	spleen_palpable	spiders	ascites	varices	bilirubin	alk_phosphate	sgot	a
0	30	male	False	False	False	False	False	False	False	False	False	False	False	1.0	85.0	18.0	
1	50	female	False	False	True	False	False	False	False	False	False	False	False	0.9	135.0	42.0	
2	78	female	True	False	True	False	False	True	False	False	False	False	False	0.7	96.0	32.0	
3	31	female	NaN	True	False	False	False	True	False	False	False	False	False	0.7	46.0	52.0	
4	34	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	200.0	
5	34	female	True	False	False	False	False	True	False	False	False	False	False	0.9	95.0	28.0	
6	51	female	False	False	True	False	True	True	False	True	True	False	False	NaN	NaN	NaN	
7	23	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	NaN	
8	39	female	True	False	True	False	False	True	True	False	False	False	False	0.7	NaN	48.0	
9	30	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	120.0	

Identify missing values

We note that the dataset presents some problems. For example, the column email is not available for all the rows. In some cases it presents the `NaN` value, which means that the value is missing.

In order to check whether our dataset contains missing values, we can use the function `isna()`, which returns if an cell of the dataset if `NaN` or not. Then we can count how many missing values there are for each column.

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

which gives the following output:

```
age                 0
sex                 0
steroid             1
antivirals          0
fatigue             1
malaise             1
anorexia            1
liver_big          10
liver_firm         11
spleen_palpable     5
spiders             5
ascites             5
varices             5
bilirubin           6
alk_phosphate      29
sgot                4
albumin            16
protime            67
histology           0
class               0
dtype: int64
```

Now we can count the percentage of missing values for each column, simply by dividing the previous result by the length of the dataset (`len(df)`) and

multiplying per 100.

```
df.isna().sum()/len(df)*100
```

which gives the following output:

```
age                 0.000000
sex                 0.000000
steroid             0.645161
antivirals          0.000000
fatigue             0.645161
malaise             0.645161
anorexia            0.645161
liver_big           6.451613
liver_firm          7.096774
spleen_palpable     3.225806
spiders             3.225806
ascites             3.225806
varices             3.225806
bilirubin           3.870968
alk_phosphate      18.709677
sgot                2.580645
albumin            10.322581
protime            43.225806
histology           0.000000
class               0.000000
dtype: float64
```

When dealing with missing values, different alternatives can be applied:

- check the source, for example by contacting the data source to correct the missing values
- drop missing values
- replace the missing value with a value
- leave the missing value as it is.

Drop missing values

Dropping missing values can be one of the following alternatives:

- remove rows having missing values
- remove the whole column containing missing values We can use the `dropna()` by specifying the `axis` to be considered. If we set `axis = 0` we drop the entire row, if we set `axis = 1` we drop the whole column. If we apply the function `df.dropna(axis=0)` 80 rows of the dataset remain. If we apply the function `df.dropna(axis=1)`, only the columns age, sex, antivirals, histology and class remain. However, removed values are not applied to the original dataframe, but only to the result.

We can use the argument `inplace=True` in order to store changes in the original dataframe `df` (`df.dropna(axis=1,inplace=True)`).

```
df.dropna(axis=1)
```

	age	sex	antivirals	histology	class
0	30	male	False	False	live
1	50	female	False	False	live
2	78	female	False	False	live
3	31	female	True	False	live
4	34	female	False	False	live
...	...	...	...	...	...
150	46	female	False	True	die
151	44	female	False	True	live
152	61	female	False	True	live
153	53	male	False	True	live

As an alternative, we can specify only the column on which the dropping operation must be applied. In the following example, only missing rows related to the column `liver_big` are considered. This can be achieved through the `subset` parameter, which permits to specify the subset of columns where to apply the dropping operation.

```
df.dropna(subset=['liver_big'],axis=0,inplace=True)
```

Now we can check whether there are still missing values for the column `indirizzo`.

```
df.isna().sum()/len(df)*100
```

Another alternative involves the dropping of columns where a certain percentage of not-null values is available. This can be achieved through the `thresh` parameter. In the following example we keep only columns where there are at least the 80% of not null values.

```
df.dropna(thresh=0.8*len(df),axis=1,inplace=True)
```

Replace missing values

A good strategy when dealing with missing values involves their replacement with another value. Usually, the following strategies are adopted:

- for numerical values replace the missing value with the average value of the column
- for categorial values replace the missing value with the most frequent value of the column
- use other functions

In order to replace missing values, three functions can be used: `fillna()`, `replace()` and `interpolate()`. The `fillna()` function replaces all the NaN values with the value passed as argument. For example, for numerical values, all the NaN values in the numeric columns could be replaced with the average value. In order to list the type of a column, we can use the attribute `dtypes` as follows:

```
df.dtypes
```

which gives the following output:

```
age                 int64
sex                object
steroid            object
antivirals           bool
fatigue            object
malaise            object
anorexia           object
liver_big          object
liver_firm         object
spleen_palpable    object
spiders            object
ascites            object
varices            object
bilirubin         float64
alk_phosphate     float64
sgot              float64
albumin           float64
histology            bool
class              object
dtype: object
```

Numeric columns

Firstly, we select numeric columns.

```
import numpy as np
numeric = df.select_dtypes(include=np.number)
numeric_columns = numeric.columns
```

Then, we fill the NaN values of numeric columns with the average value, given by the `df.mean()` function.

```
df[numeric_columns] = df[numeric_columns].fillna(df.mean())
```

Now, we can check whether the NaN values in numeric columns have been removed.

```
df.isna().sum()/len(df)*100
```

which gives the following output:

```
age                0.000000
sex                0.000000
steroid            0.689655
```

```
antivirals          0.000000
fatigue             0.000000
malaise             0.000000
anorexia            0.000000
liver_big           0.000000
liver_firm          0.689655
spleen_palpable     0.689655
spiders             0.689655
ascites             0.689655
varices             0.689655
bilirubin           0.000000
alk_phosphate       0.000000
sgot                0.000000
albumin             0.000000
histology           0.000000
class               0.000000
dtype: float64
```

Categorial columns

We note that in `dtypes` the categorial columns are described as objects. Thus we can select the `object` columns. We would like to consider only boolean columns. However the `object` type includes also the column `class`, which is a string. We select all the object columns, and then we remove from them the column `class`. Then we can convert the type of the result to `bool`.

```
boolean_columns =
df.select_dtypes(include=np.object).columns.tolist()
```

```
boolean_columns.remove('class')
df[boolean_columns] = df[boolean_columns].astype('bool')
```

Now we can replace all the missing values for booleans with the most frequent value. We can use the `mode()` function to calculate the most frequent value. We use the `fillna()` function to replace missing values, but we could use also the `replace(old_value,new_value)` function.

```
df[boolean_columns].fillna(df.mode())
```

Now our dataset does not contain any missing value.

```
df.isna().sum()/len(df)*100
```

which gives the following output:

```
age                0.0
sex                0.0
steroid            0.0
antivirals         0.0
```

```
fatigue            0.0
malaise            0.0
anorexia           0.0
liver_big          0.0
liver_firm         0.0
spleen_palpable    0.0
spiders            0.0
ascites            0.0
varices            0.0
bilirubin          0.0
alk_phosphate      0.0
sgot               0.0
albumin            0.0
histology          0.0
class              0.0
dtype: float64
```

Interpolation

Another solution to replace missing values involves the usage of other functions, such as linear interpolation. In this case, for example, we could replace a missing value over a column, with the interpolation between the previous and the next ones. This can be achieved through the use of the `interpolate()` function.

Since we have already managed all the missing values, we reload the dataset.

```
df = pd.read_csv('hepatitis.csv')
df.isna().sum()/len(df)*100
```

We select only numeric columns.

```
numeric = df.select_dtypes(include=np.number)
numeric_columns = numeric.columns
df.head(10)
```

	age	sex	steroid	antivirals	fatigue	malaise	anorexia	liver_big	liver_firm	spleen_palpable	spiders	ascites	varices	bilirubin	alk_phosphate	sgot	a
0	30	male	False	False	False	False	False	False	False	False	False	False	False	1.0	85.0	18.0	
1	50	female	False	False	True	False	False	False	False	False	False	False	False	0.9	135.0	42.0	
2	78	female	True	False	True	False	False	True	False	False	False	False	False	0.7	96.0	32.0	
3	31	female	NaN	True	False	False	False	True	False	False	False	False	False	0.7	46.0	52.0	
4	34	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	200.0	
5	34	female	True	False	False	False	False	True	False	False	False	False	False	0.9	95.0	28.0	
6	51	female	False	False	True	False	True	True	False	True	True	False	False	NaN	NaN	NaN	
7	23	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	NaN	
8	39	female	True	False	True	False	False	True	True	False	False	False	False	0.7	NaN	48.0	
9	30	female	True	False	False	False	False	True	False	False	False	False	False	1.0	NaN	120.0	

Now we can apply the `interpolate()` function to numeric columns, by setting also the limit direction to `forward`. This means that the linear interpolation is applied starting from the first row until the last one.

```
df[numeric_columns] = df[numeric_columns].interpolate(method
='linear', limit_direction ='forward')
```

For example, in line 6 the column `bilirubin`, which was NaN before the interpolation, now assumes the value 0.95, which is the interpolation between 0.90 (line 4) and 1.00 (line 6).

```
df.head(10)
```

	age	sex	steroid	antivirals	fatigue	malaise	anorexia	liver_big	liver_firm	spleen_palpable	spiders	ascites	varices	bilirubin	alk_phosphate	sg
0	30	male	False	False	False	False	False	False	False	False	False	False	False	1.00	85.0	18.0000
1	50	female	False	False	True	False	False	False	False	False	False	False	False	0.90	135.0	42.0000
2	78	female	True	False	True	False	False	True	False	False	False	False	False	0.70	96.0	32.0000
3	31	female	NaN	True	False	False	False	True	False	False	False	False	False	0.70	46.0	52.0000
4	34	female	True	False	False	False	False	True	False	False	False	False	False	1.00	70.5	200.0000
5	34	female	True	False	False	False	False	True	False	False	False	False	False	0.90	95.0	28.0000
6	51	female	False	False	True	False	True	True	False	True	True	False	False	0.95	91.6	34.6666
7	23	female	True	False	False	False	False	True	False	False	False	False	False	1.00	88.2	41.3333
8	39	female	True	False	True	False	False	True	True	False	False	False	False	0.70	84.8	48.0000
9	30	female	True	False	False	False	False	True	False	False	False	False	False	1.00	81.4	120.0000

Summary

In this tutorial we have seen one of the aspects of data preprocessing, which is dealing with missing data. Missing data can alter the data analysis process, thus they must be managed.

Three strategies can be used to deal with missing data:

1. drop missing data: this can be done when the dataset has a small number of missing data
2. replace missing data with other values, such as the mean or the most frequent value
3. leave missing data as they are.

If you would like to learn about the other aspects of data preprocessing, such as data standardization and data normalization, stay tuned...

If you wanted to be updated on my research and other activities, you can follow me on Twitter, Youtube and and Github.

Data Preprocessing | Data Analysis | Data Science | Python | Pandas

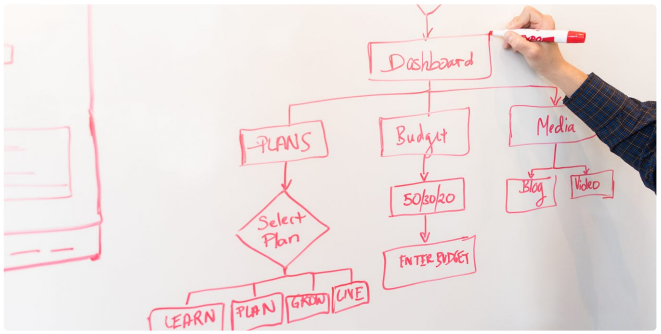
Written by Angelica Lo Duca

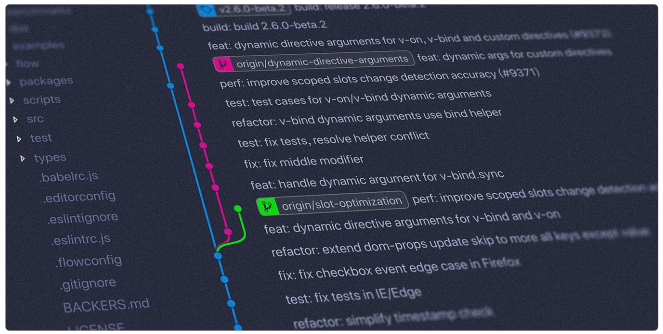
Follow

3.7K Followers · Writer for Towards Data Science

Researcher | +50k monthly views | I write on Data Science, Python, Tutorials, and, occasionally, Web Applications | Book Author of Comet for Data Science

More from Angelica Lo Duca and Towards Data Science





Angelica Lo Duca in Towards Data Science

How to Use ChatGPT to Generate Diagrams

A quick tutorial on how to write proper prompts to make ChatGPT generate...

· 4 min read · May 29

188 3

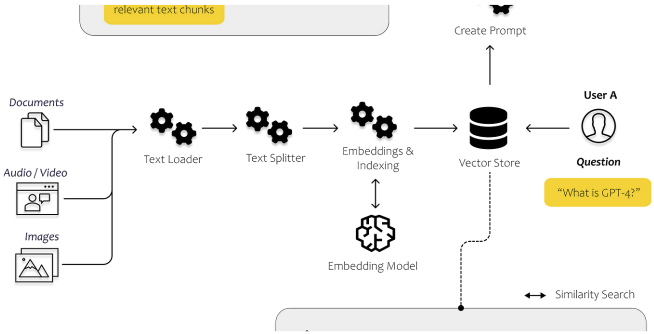
Miriam Santos in Towards Data Science

Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

1.7K 19

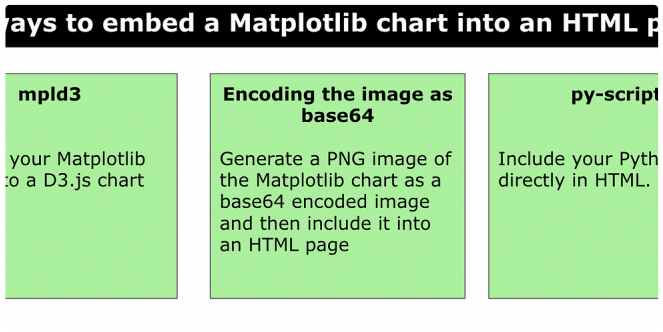


Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt...

· 26 min read · Jun 22



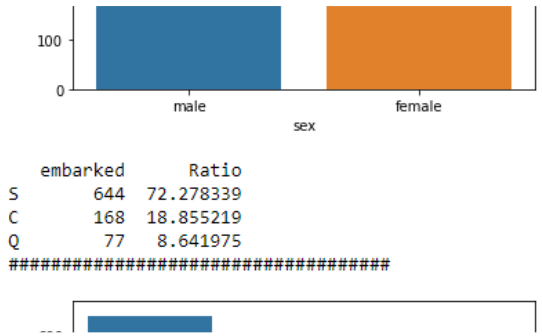
Angelica Lo Duca in Towards Data Science

3 Ways to Embed a Matplotlib Chart into an HTML Page

A tutorial on how to import a Matplotlib chart into an HTML file

· 5 min read · Jun 22

Recommended from Medium



Muhammed Resit Cicekd... in Python in Plain Engli...

Python Data Analyze—Advanced Functional Exploratory Data...

In the first step,(...

Dr. Soumen Atta, Ph.D. in Level Up Coding

Mastering Data Visualization with Pandas: A Step-by-Step Tutorial

Data visualization is a critical aspect of data analysis that enables us to explore and...

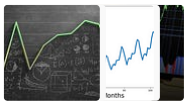
✦ · 3 min read · Feb 18

6

✦ · 7 min read · Mar 14

62

Lists



Predictive Modeling w/ Python

18 stories · 125 saves

Coding & Development

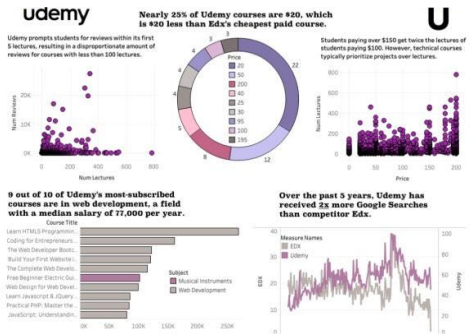
11 stories · 51 saves

New_Reading_List

174 stories · 21 saves

Practical Guides to Machine Learning

10 stories · 134 saves



Zach Qui... in Pipeline: Your Data Engineering Res...

Susan Maina in Towards Data Science

Creating The Dashboard That Got Me A Data Analyst Job Offer

A walkthrough of the Udemy dashboard that got me a job offer from one of the biggest...

✦ · 9 min read · Dec 5, 2022

1.2K 20

Pivot tables in Pandas and Handling Multi-Index Data with...

Learn how to pivot a Pandas DataFrame and get meaningful insights

✦ · 11 min read · Feb 10

426 2

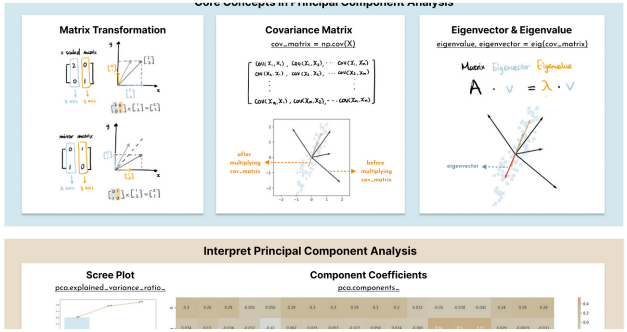
Youssef Hosni in Level Up Coding

13 SQL Statements for 90% of Your Data Science Tasks

Structured Query Language (SQL) is a programming language designed for...

✦ · 15 min read · Feb 27

3K 33



Destin Gong in Towards Data Science

A Visual Learner's Guide to Explain, Implement and Interpret Principal...

Linear Algebra for Machine Learning — Covariance Matrix, Eigenvector and Principa...

✦ · 11 min read · Jan 25

81 1

See more recommendations