Data Journalism

Data Cleaning

Part 2

Angelica Lo Duca
angelica.loduca@iit.cnr.it

Data Cleansing involves the following aspects:

- missing values
- data formatting
- data normalization
- **data standardization**
- **data binning**
- **remove duplicates**

Data Standardization

Standardization transforms data to have a mean of zero and a standard deviation of 1.

Techniques for standardization

- z-score
- z-map

z-score

The new value is calculated as the difference between the current value and the average value, divided by the standard deviation.

We can use the **`zscore()`** function of the **`scipy.stats`** library.

Example

```
from scipy.stats import zscore
df['Value'] = zscore(df['Value'])
```

Value
1
3
4

→

Value
-1.34
0.26
1.07

MEAN: 2.66 STD: 1.25

z-map

The new value is calculated as the difference between the current value and the average value of a comparison array, divided by the standard deviation of a comparison array.

We can use the **`zmap()`** function of the **`scipy.stats`** library.

Example

```
from scipy.stats import zmap
df['Value'] = zmap(df['Value'], df['Count'])
```

Value	Count
1	3
3	4
4	5

Value	Count
-3.67	3
-1.22	4
0	5

Data Binning

Data binning (or bucketing) groups data in bins (or buckets), in the sense that it replaces values contained into a small interval with a single representative value for that interval.

Binning

Binning can be applied to convert numeric values to categorical or to sample (quantize) numeric values.

Binning is a technique for data smoothing. Data smoothing is employed to remove noise from data. Three techniques for data smoothing:

- binning
- regression
- outlier analysis

Techniques for binning

- convert numeric to categorical
  - binning by distance
  - binning by frequency
- reduce numeric values
  - sampling

Binning by distance - cut()

- Define the bin edges
- Convert numeric into categorical variables
- Define the number of bins and the associated labels

Size
1000
5
500
100
250
400

bins = 4

Label	Ranges
small	0-50
medium	51-100
large	101-500
very large	> 500

Size
very large
small
large
medium
large
large

Example

```
import numpy as np

bins = [ 0, 50, 100, 500, 1000 ]

labels = ['small', 'medium', 'large','very large']

df['Size'] = pd.cut(df['Size'] , bins=bins, labels=labels,
include_lowest=True)
```

Example 2 - Linear Space among ranges

```
min_value = df[‘Size’].min()

max_value = df[‘Size’].max()

n_bins = 4

bins = np.linspace(min_value,max_value,n_bins+1)

array([   5.   ,  336.66666667,  668.33333333, 1000.  ])

labels = ['small', 'medium', 'large','very large']

df['Size'] = pd.cut(df['Size'] , bins=bins, labels=labels, include_lowest=True)
```

Example 2 (cont.)

Size
1000
5
500
100
250
400

bins = 4

Label	Ranges
small	0 - 5
medium	5 - 336.67
large	336.67-668.33
very large	668.33 - 1000

Size
very large
small
medium
small
small
medium

Binning by frequency - qcut()

- Quantile-based discretization function
- Calculate the size of each bin so that each bin contains (almost) the same number of observations, but the bin range will vary.

Example

Size
1000
5
500
100
250
400
10
30

bins = 4
2 observations for each bin

Label
small
medium
large
very large

Size
very large
small
very large
medium
large
large
small
medium

Example (cont.)

```
labels = ['small', 'medium', 'large','very large']

n_bins = 4

df['Size'] = pd.qcut(df['Size'], q=n_bins,precision=1,
labels=labels)
```

We can set the `precision` parameter to define the number of decimal points.

Sampling

It permits to reduce the number of samples, by grouping similar values or contiguous values. There are three approaches to perform sampling:

- by bin means: each value in a bin is replaced by the mean value of the bin.
- by bin median: each bin value is replaced by its bin median value.
- by bin boundary: each bin value is replaced by the closest boundary value, i.e. maximum or minimum value of the bin.

binned_statistics()

- We exploit the `binned_statistic()` function of the `scipy.stats` package can be used.
- This function receives two arrays as input, x_data and y_data, as well as the statistics to be used (e.g. median or mean) and the number of bins to be created.
- The function returns the values of the bins as well as the edges of each bin.

Example

Size
1000
5
500
100
250
400
10
30

bins = 4

Intervals
5 - 253.75
273.75 - 502.5
502.5 - 751.25
751.25 - 1000

Size
875,625
129.375
378.125
129.375
129.375
378.125
129.375
129.376

Example (cont.)

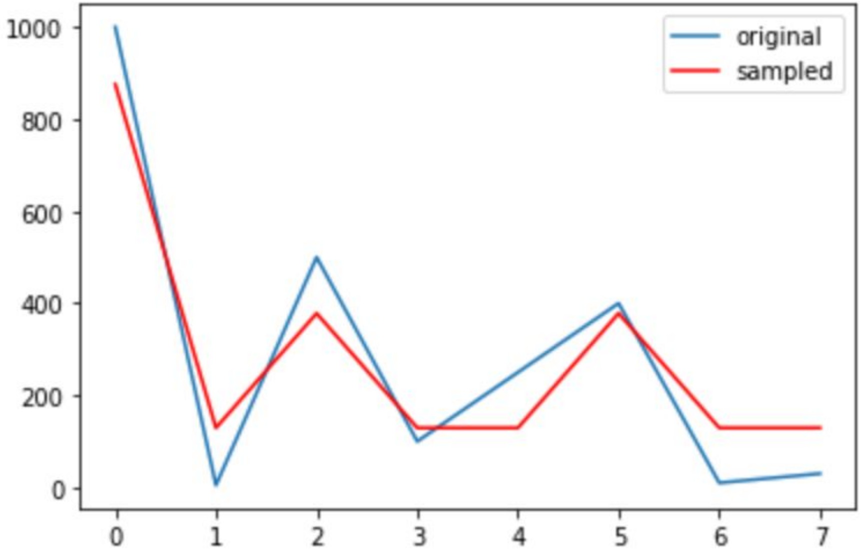
```
from scipy.stats import binned_statistic

x_data = np.arange(0, len(df))

y_data = df['Size']

x_bins,bin_edges, misc = binned_statistic(y_data,x_data,
statistic="median", bins=4)

bin_intervals = pd.IntervalIndex.from_arrays(bin_edges[:-1],
bin_edges[1:],closed='both')

IntervalIndex([[5.0, 253.75], [253.75, 502.5], [502.5,
751.25], [751.25, 1000.0]])
```

Example (cont.)

```
def set_to_median(x, bin_intervals):
    for interval in bin_intervals:
        if x in interval:
            return interval.mid
```

Example (cont.)

```
df['sampled_size'']= df['Size'].apply(lambda x:
set_to_median(x, bin_intervals))
```



Natural breaks in data

We can use the package `jenkspy`, which contains a single function, called `jenks_breaks()`, which calculates the natural breaks of an array, exploiting the Fisher-Jenks algorithm.

We can install the package by running `pip3 install jenkspy`.

Example

```
import jenkspy

breaks = jenkspy.jenks_breaks(df['Size'], nb_class=3)

df['size_break'] = pd.cut(df['Size'] , bins=breaks,
labels=labels, include_lowest=True)
```

Remove Duplicates

Remove all rows that appear at least twice.

The concept of duplicate

	Name	Surname	Value
1	Mark	Grenn	3
2	Mark	Grenn	3
3	Mark	Grenn	4

Rows 1 and 2 are duplicates

Rows 1, 2 and 3 are duplicates in column Name and Surname

Drop duplicates on the basis of all columns

keep just one row for each duplicate

Name	Surname	Value
Mark	Grenn	3
Mark	Grenn	4

Do not maintain any row for the duplicate

Name	Surname	Value
Mark	Grenn	4

Drop duplicates on the basis of the Name and Surname Columns

Keep just one value for column

Name	Surname	Value
Mark	Grenn	3

Do not maintain any row for the duplicate

Name	Surname	Value

drop_duplicates()

```
df1 = df.drop_duplicates()

df2 = df.drop_duplicates(keep=False)

df3 = df.drop_duplicates(subset=["Name", "Surname"])

df4 = df.drop_duplicates(subset=["Name", "Surname"],
keep=False)
```