SLU04 - Basic Stats with Pandas

July 6th and 7th, 2019

1. Introduction

Motivation

- How to make sense of thousands or millions of data points in your data set?
- How to summarize the data?
- How to deal with outliers?

Overview

Objective: learn how to use pandas to obtain simple statistics of datasets.

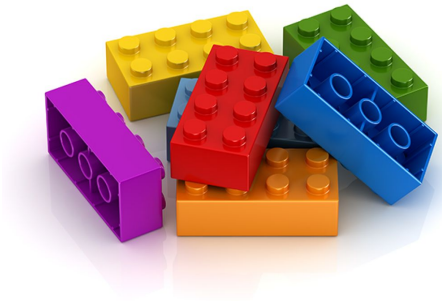
We will cover:

- Minimum, maximum, argmin, argmax
- Mode
- Mean & median
- Standard deviation
- Skewness & Kurtosis
- Quantiles
- Outliers & how to deal with them

2. Topic Explanation

Descriptive Statistics ⟹ Inspecting the distribution of the data ⟹ Outlier detection

Dataset: legos



```
In [2]: lego = pd.read_csv('data/sets.csv')
```

```
In [3]: lego.head()  # let's have a look at the first lines of the dataframe
```

Out[3]:

	set_num	name	year	theme_id	num_parts
0	00-1	Weetabix Castle	1970	414	471
1	0011-2	Town Mini-Figures	1978	84	12
2	0011-3	Castle 2 for 1 Bonus Offer	1987	199	2
3	0012-1	Space Mini-Figures	1979	143	12
4	0013-1	Space Mini-Figures	1979	143	12

set_num	Unique set ID.
name	The name of the set.
year	Year the set was published.
theme_id	Unique ID for the theme used for the set (from `themes.csv`).
num_parts	The number of parts included in the set.

Maximum & minimum

- *What are the maximum and minimum values for the number of parts?*
- *Which sets do these values correspond to?*

```
In [6]: lego.num_parts.max()
Out[6]: 5922

In [7]: lego.num_parts.idxmax()
Out[7]: 170
```

`idxmax()` gives the ***first index*** of the maximum value

```
In [10]: lego.num_parts.idxmin()
Out[10]: 1683

In [11]: lego.num_parts.min()
Out[11]: -1
```

Maximum & minimum

Basic Statistics can help us finding data problems.

- ***Are there more than one set with -1 parts?***

```
In [12]: lego.loc[lego.num_parts == lego.num_parts.min()]
```

Out[12]:

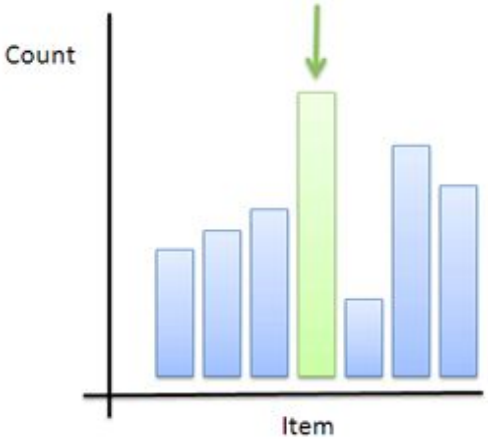
	set_num	name	year	theme_id	num_parts
1683	240-1	Wooden Storage Box Large, Empty	1967	383	-1
6545	66392-1	Duplo Cars Super Pack 3 in 1 (5816, 5817, 5818)	2012	506	-1
11645	Vancouver-1	LEGO Store Grand Opening Exclusive Set, Oakrid...	2012	408	-1

Mode

- ***What is the year that had more sets published?***

```
In [14]: lego.year.mode()

Out[14]: 0    2014
         dtype: int64
```



The **mode** is the most frequently appearing value in a population or sample.

Mean

- ***What is the average number of parts in the sets of legos?***

```
In [16]: lego.num_parts.mean()
Out[16]: 162.3043701799486
```

The **mean** is the sum all the all of the observations and dividing by the number of observations.

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Median

First, arrange the observations in an ascending order.

If the number of observations (n) is odd: the median is the value at position

$$\left(\frac{n+1}{2}\right)$$

If the number of observations (n) is even:

1. Find the value at position $\left(\frac{n}{2}\right)$
2. Find the value at position $\left(\frac{n+1}{2}\right)$
3. Find the average of the two values to get the median.

Median

In [17]:
```
pd.Series([1,
           2,
           3, # <--- ok, this one is easy
           4,
           5]
          ).median()
```

Out[17]: 3.0

In [18]:
```
pd.Series([1,
           2, # <--- both 2 and 3 are in "the middle", as there are only 4
           3, # <--- so the median will split the difference!
           4,
          ]
          ).median()
```

Out[18]: 2.5

Mean vs Median

- *What is the average number of parts in the sets of legos?*

```
In [16]: lego.num_parts.mean()
Out[16]: 162.3043701799486
```

```
In [19]: lego.num_parts.median()
Out[19]: 45.0
```

Mean vs Median

In [22]:
```
s = pd.read_csv('data/student_income.csv', header=None)[0]
```

In [23]:
```
print('The mean income of students is   %0.1f' % s.mean())
print('The median income of students is %0.1f' % s.median())
```

```
The mean income of students is   600.0
The median income of students is 625.0
```

In [24]:
```
rich_mc_money = pd.Series([10000])  # <--- Wooooow!
s = s.append(rich_mc_money)
```

In [25]:
```
print('The mean income of students is   %0.1f' % s.mean())
print('The median income of students is %0.1f' % s.median())
```

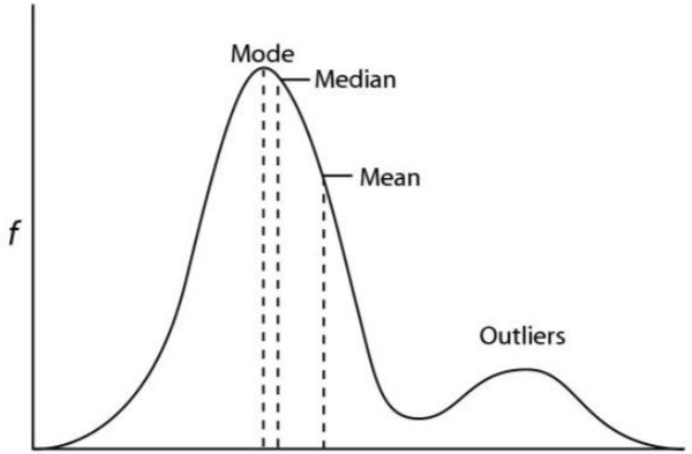
```
The mean income of students is   903.2
The median income of students is 630.0
```


BREAKING NEWS
AVERAGE STUDENT BUDGET UP 300$
10:58
"THIS IS THE RESULT OF OUR GOOD GOVERNMENT", SAYS USELESS POLITICIAN

Mean vs Median

- The **median** may be a better indicator of the most typical value if a set of scores has an **outlier**.

- With **skewed distributions** the median is not as strongly influenced by the skewed values when compared with the mean.

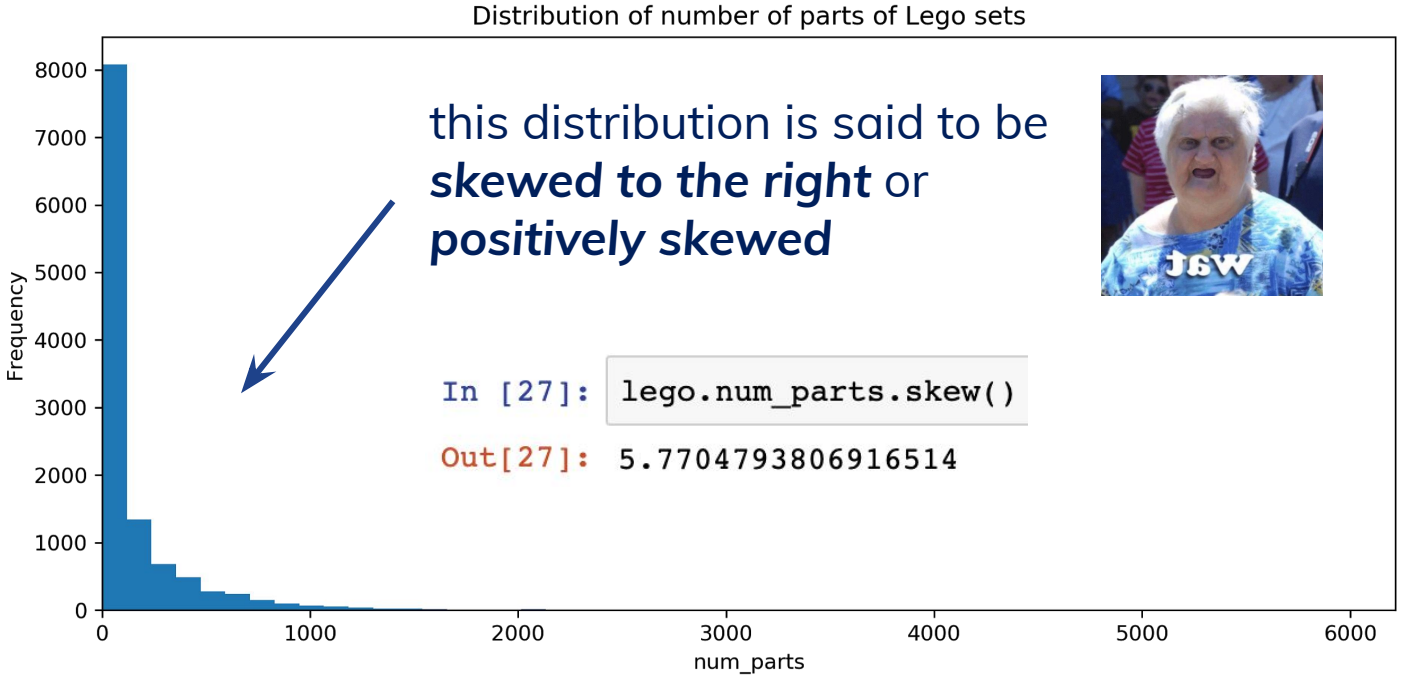


Descriptive Statistics

Inspecting the distribution of the data

Outlier detection

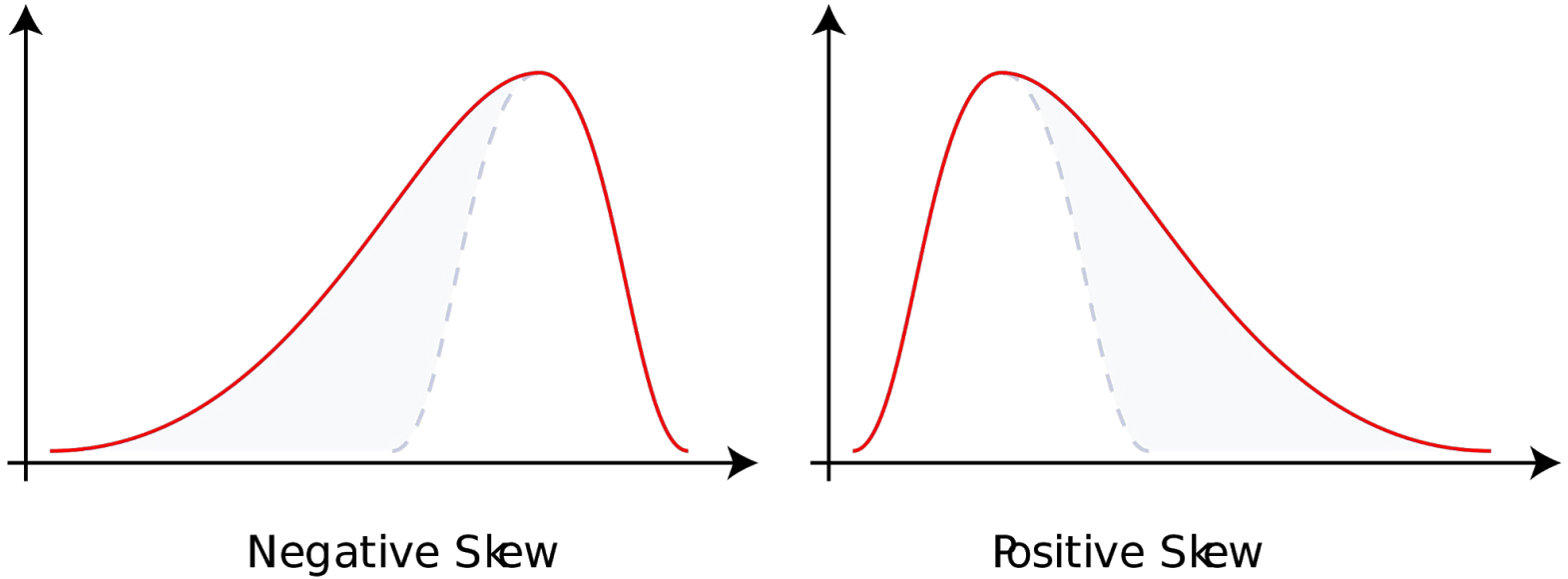
Skewness



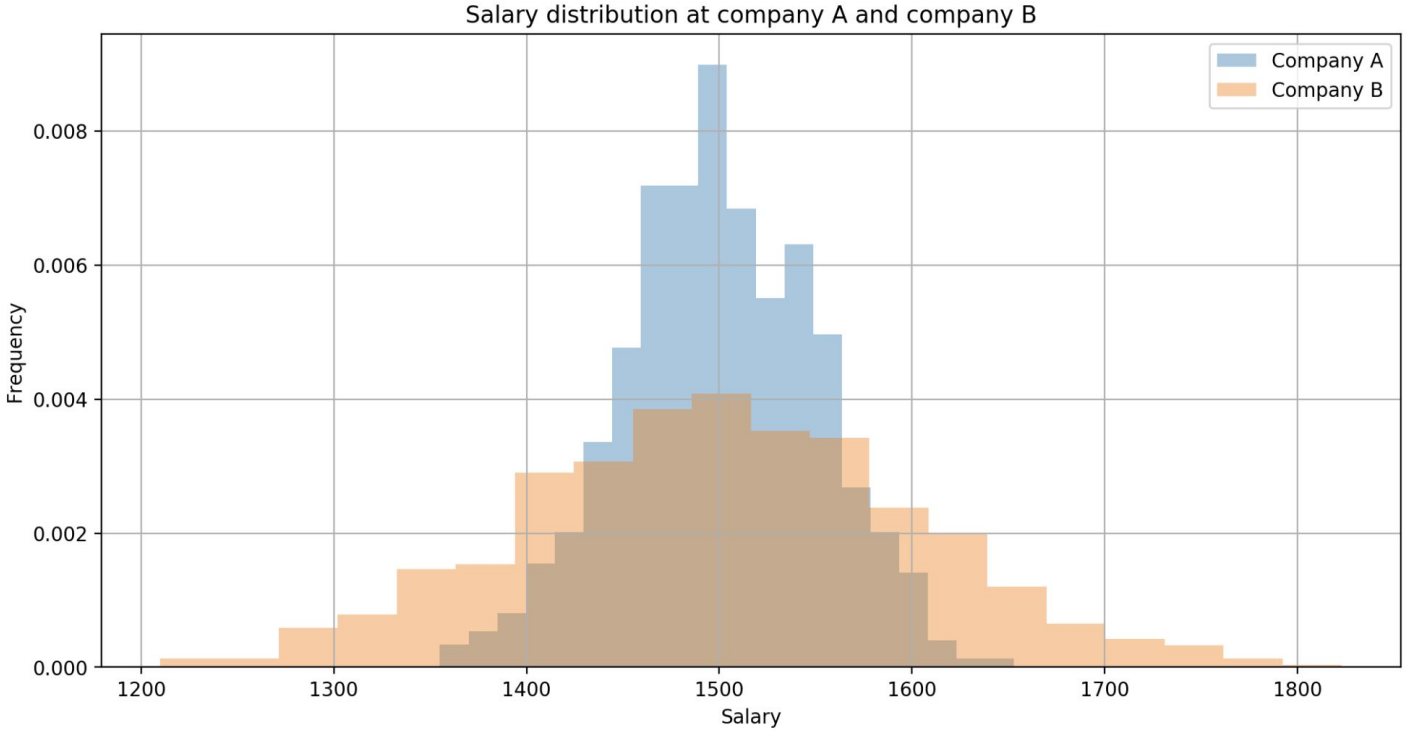
this distribution is said to be ***skewed to the right*** or ***positively skewed***


```
In [27]: lego.num_parts.skew()
Out[27]: 5.7704793806916514
```

Skewness



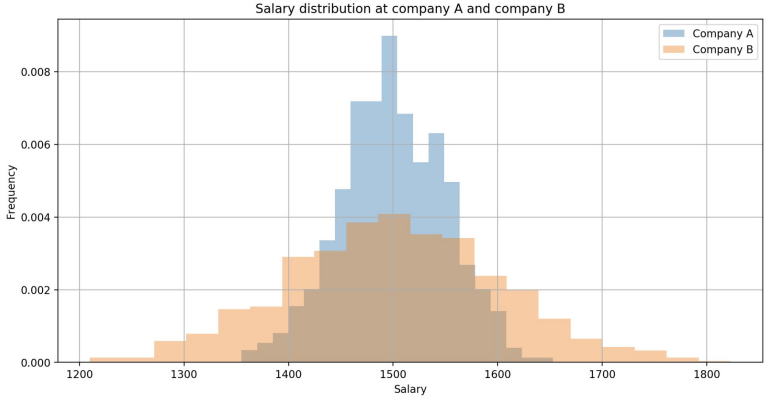
Standard deviation



Standard deviation

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2}$$

mean

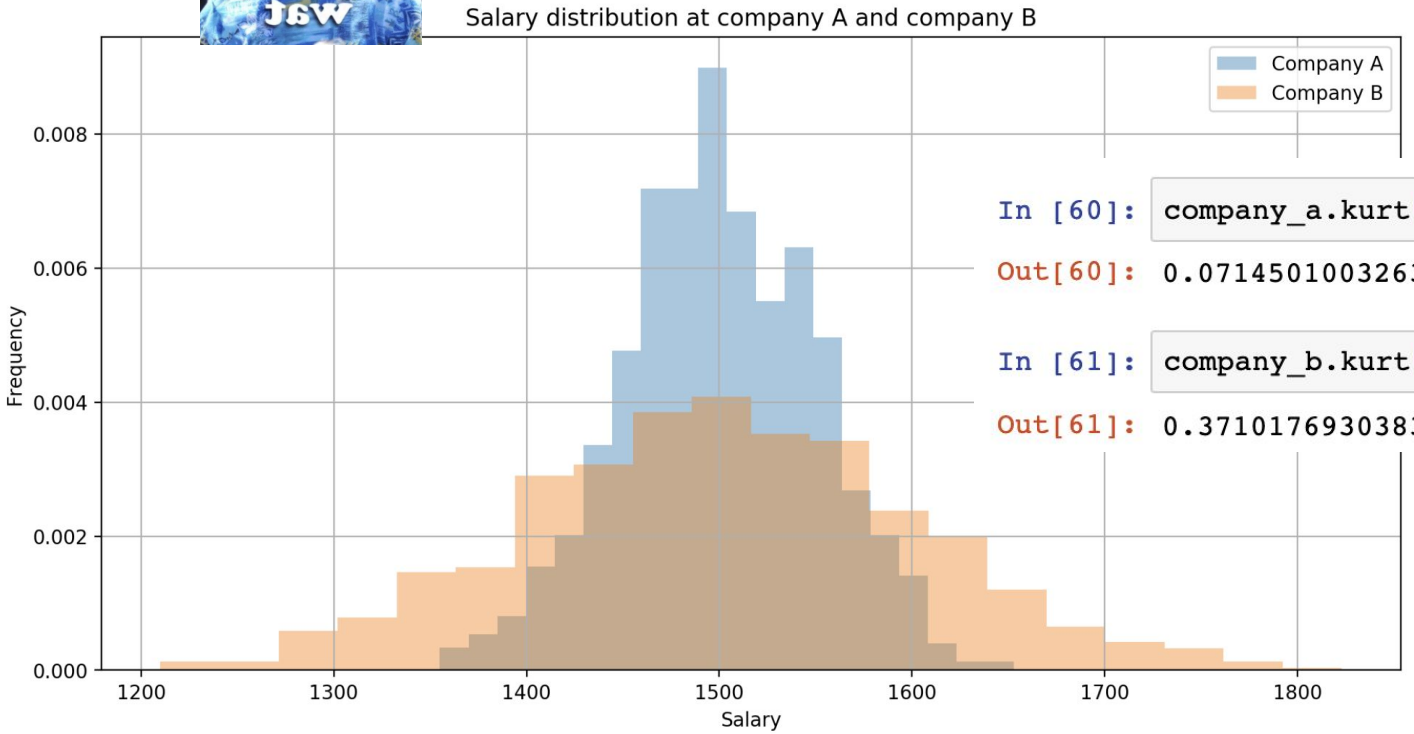



```
In [30]: print('Company A has a mean of %0.1f and a standard deviation of %0.1f' % (
             company_a.mean(), company_a.std()))

         print('Company B has a mean of %0.1f and a standard deviation of %0.1f' % (
             company_b.mean(), company_b.std()))
```

```
Company A has a mean of 1499.9 and a standard deviation of 49.7
Company B has a mean of 1500.3 and a standard deviation of 101.8
```

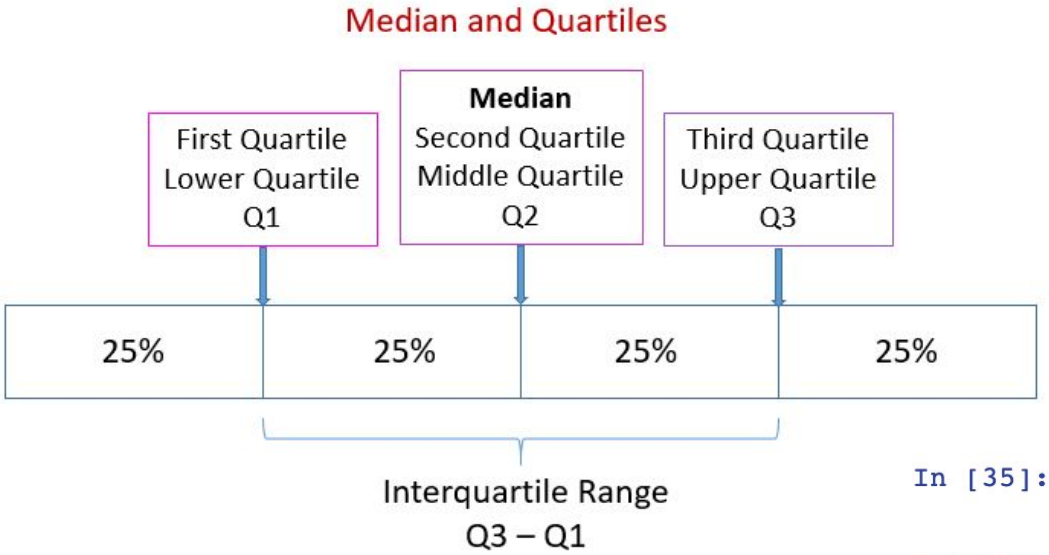
Kurtosis




```
In [60]: company_a.kurt()
Out[60]: 0.0714501003263166

In [61]: company_b.kurt()
Out[61]: 0.3710176930383273
```

Quantiles




```
In [35]: quartiles = [.25, .5, .75]
         lego.num_parts.quantile(q=quartiles)

Out[35]: 0.25     10.0
         0.50     45.0
         0.75    172.0
         Name: num_parts, dtype: float64
```

Descriptive Statistics

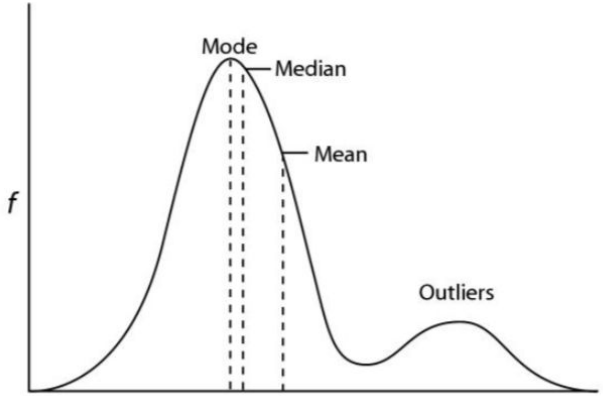
Inspecting the distribution of the data

Outlier detection

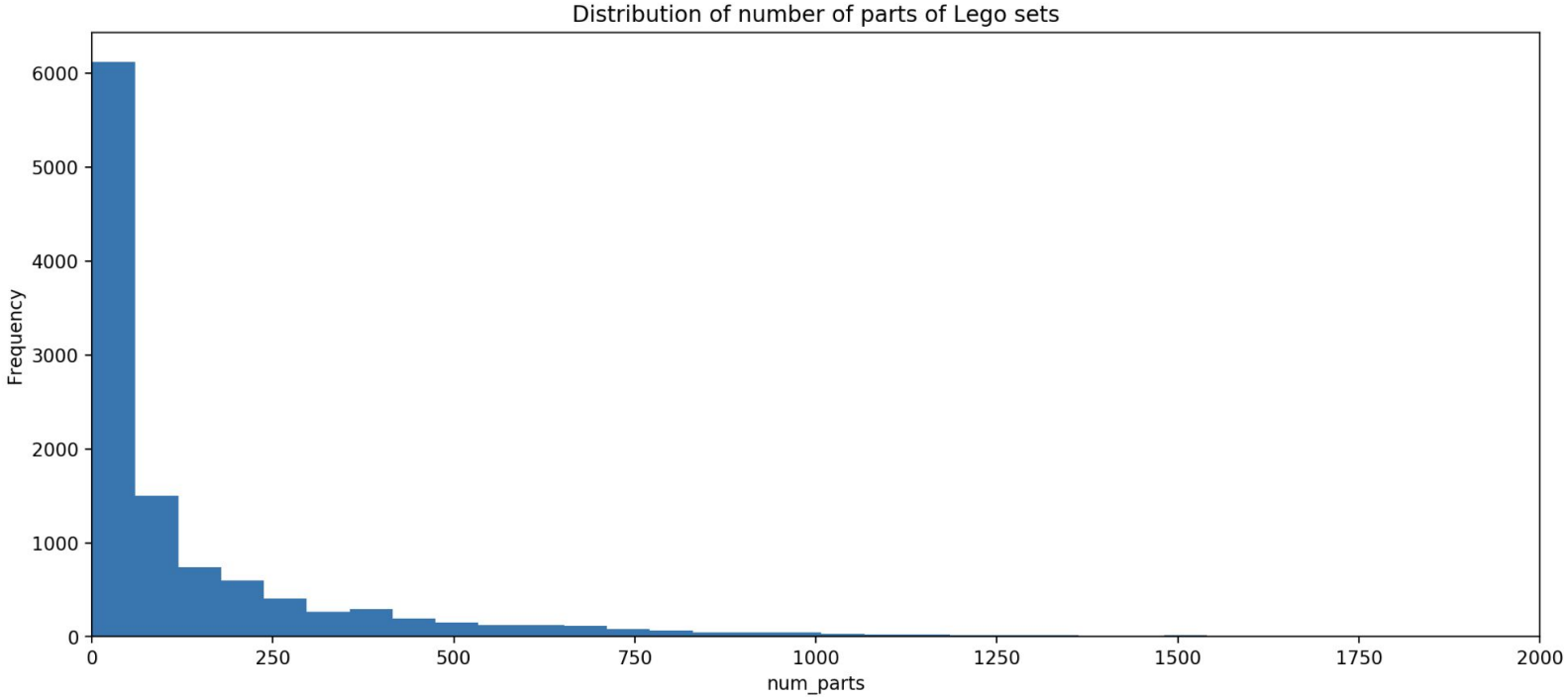
Dealing with outliers

A few strategies to deal with **outliers** are the following:

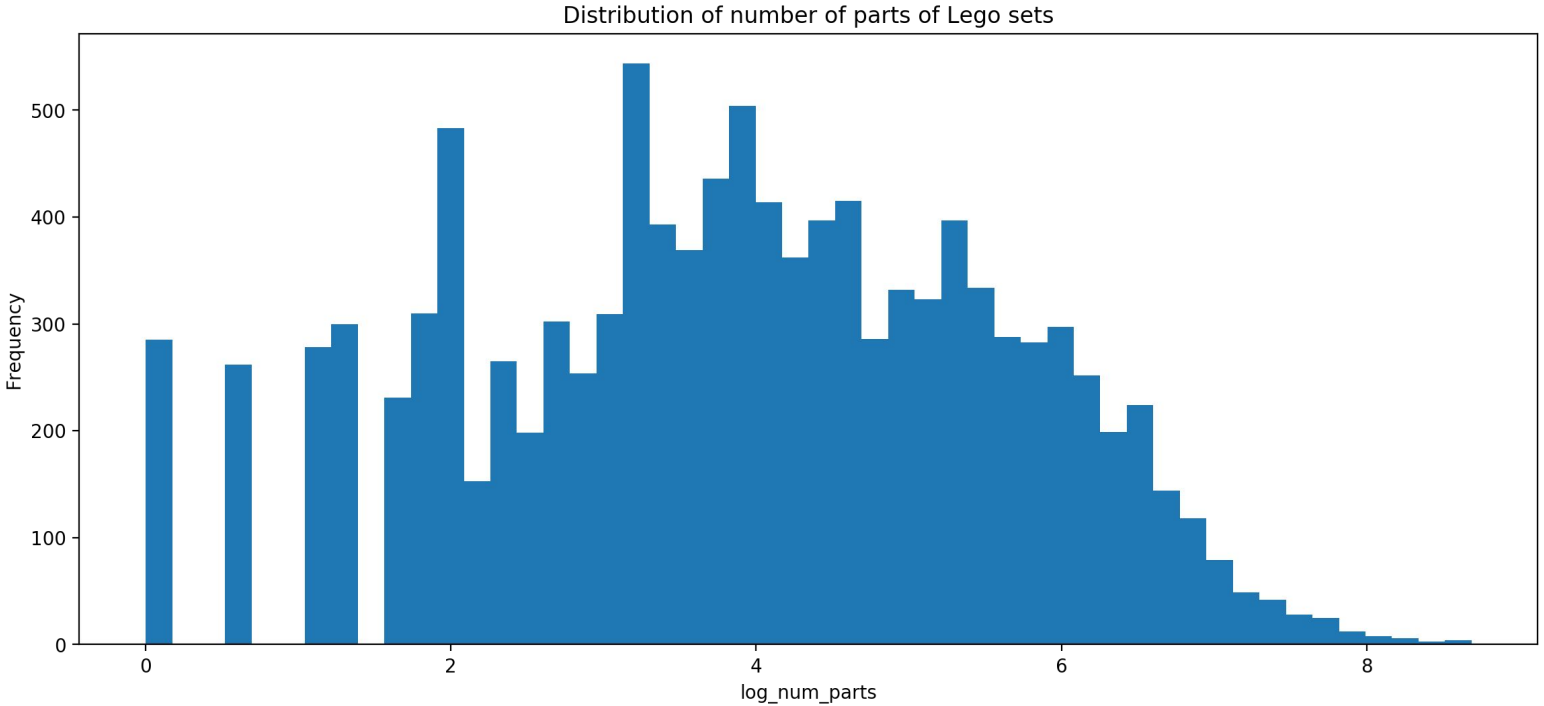
- do nothing;
- drop instances with outliers;
- drop column with outliers;
- impute values;
- binning;
- transforming data: e.g. **log transformation**.



log transformation



log transformation



3. Recap

Recap

```
In [37]: lego.num_parts.describe()

Out[37]: count    11670.000000
         mean       162.304370
         std        330.224172
         min          0.000000
         25%         10.000000
         50%         45.000000
         75%        172.000000
         max       5922.000000
         Name: num_parts, dtype: float64
```