Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 11 – 15, 2021

Python for Data Analytics

Data Preprocessing

Data Collection for Data Analytics

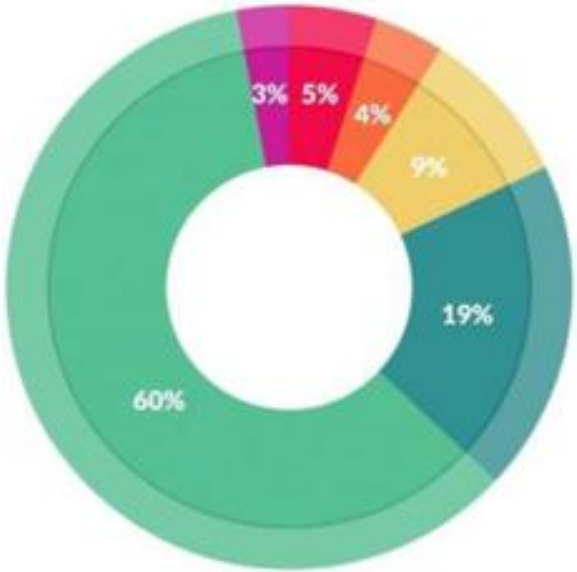
- You will typically get data in one of four ways:

1. Directly download a data file (or files) manually
2. Query data from a database
3. Query an API (usually web-based, these days)
4. Scrap data from a webpage

How to perform data preprocessing in Python?

Data Preprocessing

- The process of cleaning up the messy raw data for analysis
- Repetitive and tedious work
- Direct impact on analysis result and model performance
- Data collection and preprocessing occupy up to 80% of the analysis time



What data scientists spend the most time doing

- Building training set: 3%
- Cleaning and organizing data: 60%
- Collecting data set: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

Outline

- Handling Missing Values
- Handling Outliers
- Data Scaling
- Data Standardization
- Data Encoding
- Sampling for Imbalanced Data

Handling Missing Values

Handling Missing Values

- Contact the data source to correct the missing values
  - Especially for human or mechanical errors
- Drop the column or row that contains missing values
  - Easy, but the dataset size should be large
- Replace the missing value with another value
  - For numerical data: mean, median, mode (most frequent) or zero
  - For categorical data: mode
  - Use interpolation
  - Requires domain knowledge
- Leave the missing value as it is

Dataset for Handling Missing Values

- User behavior and payment data in a shared file system
- 200,000 entries with 22 columns
- Column information
  - rowid
  - iduser: User ID
  - mdutype: payment type
  - group: payment info (mdu: paid user, sdu: free user)
  - view/edit/share/search/cowork counts
  - add/del/move/rename counts
  - other user behaviors

```
df = pd.read_csv('testset.csv', index_col=0)
```

https://github.com/songhunhwa/songhunhwa.github.com/tree/master/tutorial/tutorial_03/testset.csv

Dataset Head & Tail

```
df.head()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns

```
df.tail()
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
199995	10100014533282	NaN	sdu	37.0	0.0	2.0	...	7.0	0.0	13064406.0	1922364.0	0.0	14986770.0
199996	10100037382422	a2p	mdu	6.0	0.0	0.0	...	0.0	0.0	15936676.0	0.0	0.0	15936676.0
199997	10100024157271	NaN	sdu	32.0	0.0	0.0	...	0.0	0.0	7305871.0	0.0	0.0	7305871.0
199998	10100022150627	NaN	sdu	18.0	0.0	0.0	...	0.0	0.0	53352144.0	0.0	0.0	53352144.0
199999	10100021804275	NaN	sdu	3.0	0.0	0.0	...	0.0	0.0	95232.0	0.0	0.0	95232.0

5 rows × 22 columns

Set Index

```
df.set_index('iduser', inplace=True)
```

	iduser	mdutype	group	viewCount	editCount	shareCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
0	10100018739106	NaN	sdu	12.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	3504812.0
1	10100037810674	NaN	sdu	23.0	0.0	0.0	...	0.0	0.0	17123098.0	0.0	0.0	17123098.0
2	10100036273719	NaN	sdu	4.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	2234363.0
3	10100027752244	NaN	sdu	6.0	0.0	1.0	...	2.0	0.0	602361.0	210114.0	0.0	812475.0
4	10100000624840	NaN	sdu	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 22 columns

iduser	mdutype	group	viewCount	editCount	shareCount	searchCount	...	saveCount	exportCount	viewTraffic	editTraffic	exportTraffic	tra
10100018739106	NaN	sdu	12.0	0.0	0.0	0.0	...	0.0	0.0	3504812.0	0.0	0.0	35048
10100037810674	NaN	sdu	23.0	0.0	0.0	1.0	...	0.0	0.0	17123098.0	0.0	0.0	171230
10100036273719	NaN	sdu	4.0	0.0	0.0	0.0	...	0.0	0.0	2234363.0	0.0	0.0	22343
10100027752244	NaN	sdu	6.0	0.0	1.0	0.0	...	2.0	0.0	602361.0	210114.0	0.0	8124
10100000624840	NaN	sdu	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	N

5 rows × 21 columns

Identifying Missing Values

- Check DataFrame information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
mdutype          9328 non-null object
group            200000 non-null object
viewCount        165369 non-null float64
editCount        165369 non-null float64
shareCount       165369 non-null float64
searchCount      165369 non-null float64
coworkCount      165369 non-null float64
add              63166 non-null float64
del              63166 non-null float64
move             63166 non-null float64
rename           63166 non-null float64
adddir           63166 non-null float64
movedir          63166 non-null float64
visdays          184306 non-null float64
openCount        149090 non-null float64
saveCount        149090 non-null float64
exportCount      149090 non-null float64
viewTraffic      149090 non-null float64
editTraffic      149090 non-null float64
exportTraffic    149090 non-null float64
traffic          149090 non-null float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

- Check per-column missing values

```
df.isnull().sum()
```

```
mdutype          190672
group                 0
viewCount         34631
editCount         34631
shareCount        34631
searchCount       34631
coworkCount       34631
add              136834
del              136834
move             136834
rename           136834
adddir           136834
movedir          136834
visdays           15694
openCount         50910
saveCount         50910
exportCount       50910
viewTraffic       50910
editTraffic       50910
exportTraffic     50910
traffic           50910
dtype: int64
```

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

```
mdutype          95.3360
group             0.0000
viewCount        17.3155
editCount        17.3155
shareCount       17.3155
searchCount      17.3155
coworkCount      17.3155
add              68.4170
del              68.4170
move             68.4170
rename           68.4170
adddir           68.4170
movedir          68.4170
visdays           7.8470
openCount        25.4550
saveCount        25.4550
exportCount      25.4550
viewTraffic      25.4550
editTraffic      25.4550
exportTraffic    25.4550
traffic          25.4550
dtype: float64
```

Dropping Missing Values (1)

- Remove rows having missing values

```
df_droprows = df.dropna()
df_droprows.isnull().sum()
```

```
mdutype          0
group            0
viewCount        0
editCount        0
shareCount       0
searchCount      0
coworkCount      0
add              0
del              0
move             0
rename           0
adddir           0
movedir          0
visdays          0
openCount        0
saveCount        0
exportCount      0
viewTraffic      0
editTraffic      0
exportTraffic    0
traffic          0
dtype: int64
```

```
df_droprows.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2717 entries, 10100022538111 to 10100003355450
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   mdutype         2717 non-null   object
 1   group           2717 non-null   object
 2   viewCount       2717 non-null   float64
 3   editCount       2717 non-null   float64
 4   shareCount      2717 non-null   float64
 5   searchCount     2717 non-null   float64
 6   coworkCount     2717 non-null   float64
 7   add             2717 non-null   float64
 8   del             2717 non-null   float64
 9   move            2717 non-null   float64
 10  rename          2717 non-null   float64
 11  adddir          2717 non-null   float64
 12  movedir         2717 non-null   float64
 13  visdays         2717 non-null   float64
 14  openCount       2717 non-null   float64
 15  saveCount       2717 non-null   float64
 16  exportCount     2717 non-null   float64
 17  viewTraffic     2717 non-null   float64
 18  editTraffic     2717 non-null   float64
 19  exportTraffic   2717 non-null   float64
 20  traffic         2717 non-null   float64
dtypes: float64(19), object(2)
memory usage: 467.0+ KB
```

Dropping Missing Values (2)

- Remove the whole column containing missing values

```
df_dropcols = df.dropna(axis=1)
df_dropcols.isnull().sum()
```

```
group    0
dtype: int64
```

```
df_dropcols.head()
```

	group
iduser	
10100018739106	sdu
10100037810674	sdu
10100036273719	sdu
10100027752244	sdu
10100000624840	sdu

```
df_dropcols.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 1 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   group   200000 non-null  object
dtypes: object(1)
memory usage: 3.1+ MB
```

Dropping Missing Values (3)

- Remove only missing rows related to the column 'viewCount'

```
df_dropView = df.dropna(subset=['viewCount'], axis=0)
df_dropView.info()
```

- *subset*: labels along the other axis to consider

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 165369 entries, 10100018739106 to 101000218
04275
Data columns (total 21 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   mdutype        7338 non-null    object
 1   group          165369 non-null  object
 2   viewCount      165369 non-null  float64
 3   editCount      165369 non-null  float64
 4   shareCount     165369 non-null  float64
 5   searchCount    165369 non-null  float64
 6   coworkCount    165369 non-null  float64
 7   add            61545 non-null   float64
 8   del            61545 non-null   float64
 9   move           61545 non-null   float64
 10  rename         61545 non-null   float64
 11  adddir         61545 non-null   float64
 12  movedir        61545 non-null   float64
 13  visdays        161772 non-null  float64
 14  openCount      149090 non-null  float64
 15  saveCount      149090 non-null  float64
 16  exportCount    149090 non-null  float64
 17  viewTraffic    149090 non-null  float64
 18  editTraffic    149090 non-null  float64
 19  exportTraffic  149090 non-null  float64
 20  traffic        149090 non-null  float64
dtypes: float64(19), object(2)
memory usage: 27.8+ MB
```

```
df_dropView.isnull().sum()/len(df_dropView)*100
```

```
mdutype          95.562651
group             0.000000
viewCount         0.000000
editCount         0.000000
shareCount        0.000000
searchCount       0.000000
coworkCount       0.000000
add              62.783230
del              62.783230
move             62.783230
rename           62.783230
adddir           62.783230
movedir          62.783230
visdays           2.175136
openCount         9.844046
saveCount         9.844046
exportCount       9.844046
viewTraffic       9.844046
editTraffic       9.844046
exportTraffic     9.844046
traffic           9.844046
dtype: float64
```

Dropping Missing Values (4)

- Remove columns whose percentage of non-null values is above the threshold (e.g., 80%)
  - *thresh*: require that many non-NA values

```
df_dropThre = df.dropna(thresh=0.8*len(df), axis=1)
df_dropThre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 101000218
04275
Data columns (total 7 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   group        200000 non-null  object
 1   viewCount    165369 non-null  float64
 2   editCount    165369 non-null  float64
 3   shareCount   165369 non-null  float64
 4   searchCount  165369 non-null  float64
 5   coworkCount  165369 non-null  float64
 6   visdays      184306 non-null  float64
dtypes: float64(6), object(1)
memory usage: 12.2+ MB
```

```
df_dropThre.isnull().sum()/len(df_dropThre)*100
```

```
group           0.0000
viewCount      17.3155
editCount      17.3155
shareCount     17.3155
searchCount    17.3155
coworkCount    17.3155
visdays         7.8470
dtype: float64
```

Replacing Missing Values (1)

- *df*.`select_dtypes`([*include*], [*exclude*])
  - Return a subset of the DataFrame's columns based on the column dtypes
  - *include*, *exclude*: A selection of dtypes or strings to be included/excluded (np.number: all numeric types, np.datetime64: datetimes, object: strings etc.)

```
numeric = df.select_dtypes(include=np.number)
numeric_cols = numeric.columns
numeric_cols
```

```
Index(['viewCount', 'editCount', 'shareCount', 'searchCount', 'coworkCount',
       'add', 'del', 'move', 'rename', 'adddir', 'movedir', 'visdays',
       'openCount', 'saveCount', 'exportCount', 'viewTraffic', 'editTraffic',
       'exportTraffic', 'traffic'],
      dtype='object')
```

Replacing Missing Values (2)

- Replace with the average value

```
df[numeric_cols] = df[numeric_cols].fillna(df.mean())
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   mdutype        9328 non-null    object
 1   group          200000 non-null  object
 2   viewCount      200000 non-null  float64
 3   editCount      200000 non-null  float64
 4   shareCount     200000 non-null  float64
 5   searchCount    200000 non-null  float64
 6   coworkCount    200000 non-null  float64
 7   add            200000 non-null  float64
 8   del            200000 non-null  float64
 9   move           200000 non-null  float64
 10  rename         200000 non-null  float64
 11  adddir         200000 non-null  float64
 12  movedir        200000 non-null  float64
 13  visdays        200000 non-null  float64
 14  openCount      200000 non-null  float64
 15  saveCount      200000 non-null  float64
 16  exportCount    200000 non-null  float64
 17  viewTraffic    200000 non-null  float64
 18  editTraffic    200000 non-null  float64
 19  exportTraffic  200000 non-null  float64
 20  traffic        200000 non-null  float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

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

```
mdutype          95.336
group             0.000
viewCount         0.000
editCount         0.000
shareCount        0.000
searchCount       0.000
coworkCount       0.000
add               0.000
del               0.000
move              0.000
rename            0.000
adddir            0.000
movedir           0.000
visdays           0.000
openCount         0.000
saveCount         0.000
exportCount       0.000
viewTraffic       0.000
editTraffic       0.000
exportTraffic     0.000
traffic           0.000
dtype: float64
```

Replacing Missing Values (3)

- Use linear interpolation

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   mdutype         9328 non-null    object
 1   group           200000 non-null  object
 2   viewCount       200000 non-null  float64
 3   editCount       200000 non-null  float64
 4   shareCount      200000 non-null  float64
 5   searchCount     200000 non-null  float64
 6   coworkCount     200000 non-null  float64
 7   add             199999 non-null  float64
 8   del             199999 non-null  float64
 9   move            199999 non-null  float64
 10  rename          199999 non-null  float64
 11  adddir          199999 non-null  float64
 12  movedir         199999 non-null  float64
 13  visdays         200000 non-null  float64
 14  openCount       200000 non-null  float64
 15  saveCount       200000 non-null  float64
 16  exportCount     200000 non-null  float64
 17  viewTraffic     200000 non-null  float64
 18  editTraffic     200000 non-null  float64
 19  exportTraffic   200000 non-null  float64
 20  traffic         200000 non-null  float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

before

```
df.iloc[84:89][['viewCount', 'editCount']]
```

iduser	viewCount	editCount
10100039309679	40.0	10.0
10100022148600	44.0	0.0
10100011509371	NaN	NaN
10100001192660	12.0	1.0
10100028428084	138.0	0.0

after

```
df.iloc[84:89][['viewCount', 'editCount']]
```

iduser	viewCount	editCount
10100039309679	40.0	10.0
10100022148600	44.0	0.0
10100011509371	28.0	0.5
10100001192660	12.0	1.0
10100028428084	138.0	0.0

Replacing Missing Values (4)

- Replace the categorical data with the most frequent value

```
df.mdutype.fillna(df.mdutype.mode()[0], inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200000 entries, 10100018739106 to 10100021804275
Data columns (total 21 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   mdutype        200000 non-null  object
 1   group          200000 non-null  object
 2   viewCount      200000 non-null  float64
 3   editCount      200000 non-null  float64
 4   shareCount     200000 non-null  float64
 5   searchCount    200000 non-null  float64
 6   coworkCount    200000 non-null  float64
 7   add            200000 non-null  float64
 8   del            200000 non-null  float64
 9   move           200000 non-null  float64
 10  rename         200000 non-null  float64
 11  adddir         200000 non-null  float64
 12  movedir        200000 non-null  float64
 13  visdays        200000 non-null  float64
 14  openCount      200000 non-null  float64
 15  saveCount      200000 non-null  float64
 16  exportCount    200000 non-null  float64
 17  viewTraffic    200000 non-null  float64
 18  editTraffic    200000 non-null  float64
 19  exportTraffic  200000 non-null  float64
 20  traffic        200000 non-null  float64
dtypes: float64(19), object(2)
memory usage: 33.6+ MB
```

before

```
df.mdutype.value_counts()
```

```
a2p    7094
mul    1650
p2a     584
Name: mdutype, dtype: int64
```

after

```
df.mdutype.value_counts()
```

```
a2p    197766
mul      1650
p2a       584
Name: mdutype, dtype: int64
```

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

```
mdutype          0.0
group            0.0
viewCount        0.0
editCount        0.0
shareCount       0.0
searchCount      0.0
coworkCount      0.0
add              0.0
del              0.0
move             0.0
rename           0.0
adddir           0.0
movedir          0.0
visdays          0.0
openCount        0.0
saveCount        0.0
exportCount      0.0
viewTraffic      0.0
editTraffic      0.0
exportTraffic    0.0
traffic          0.0
dtype: float64
```

Handling Outliers

Visualizing Outliers (1)

- Preparing dataset

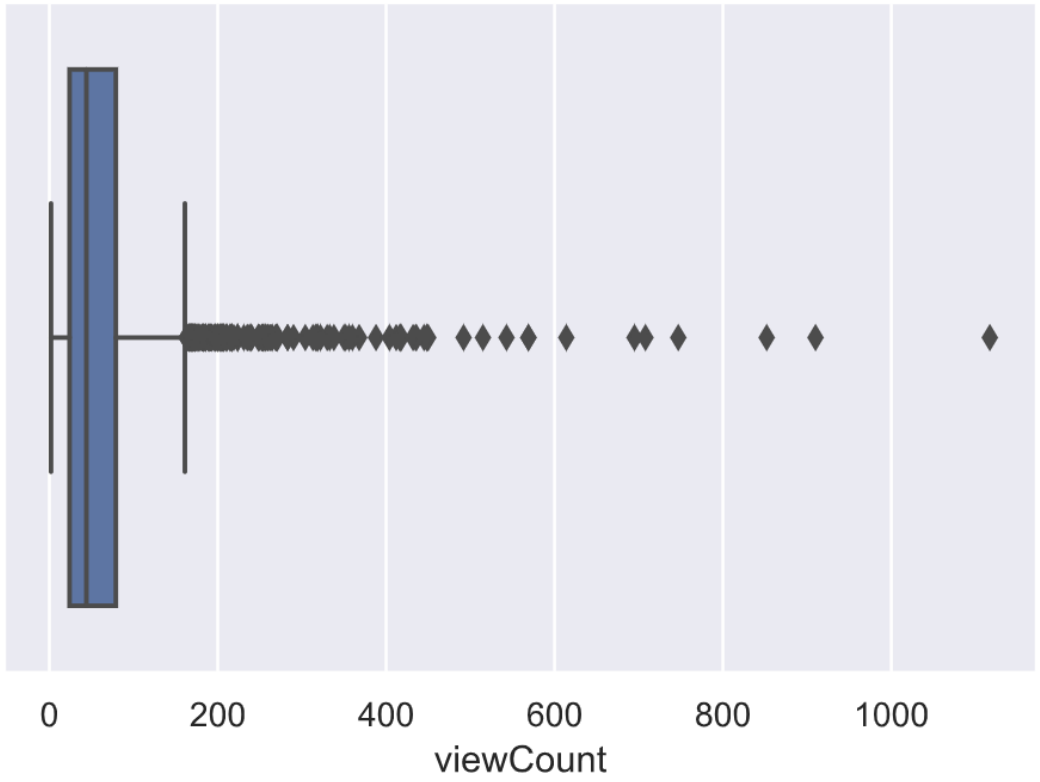
```
df = pd.read_csv('testset.csv', index_col=0)
df.set_index('iduser', inplace=True)
df = df.dropna(how='any')
df = df.select_dtypes(include=np.number)
df.head()
```

iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100022538111	35.0	68.0	1.0	0.0	0.0	...	0.0	934912.0	92672.0	0.0	1027584.0
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0

Visualizing Outliers (2)

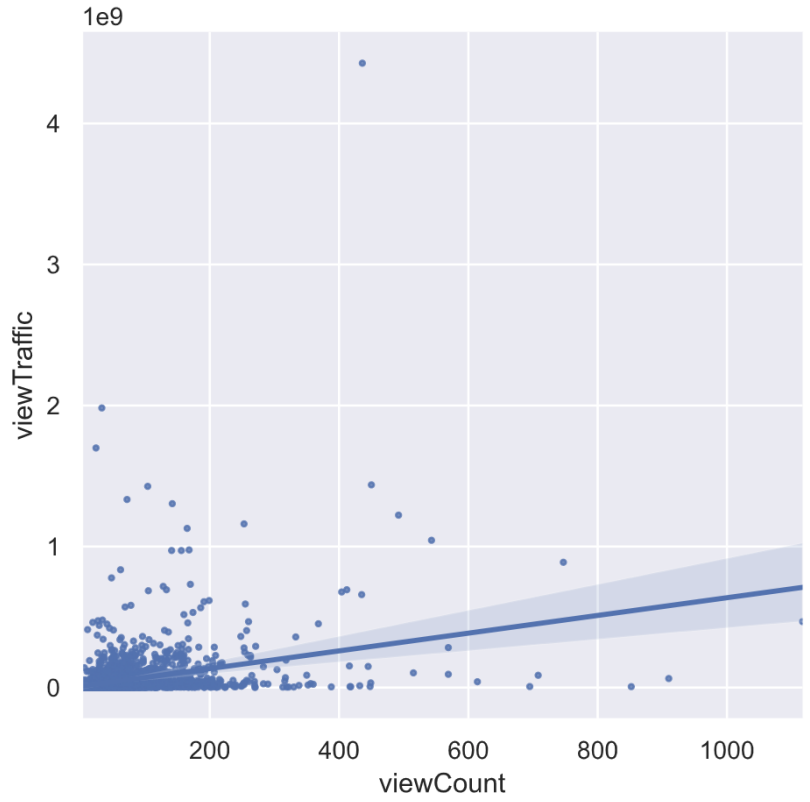
```
sns.set(style='darkgrid')
sns.boxplot(x=df.viewCount)
```

```
<AxesSubplot:xlabel='viewCount'>
```




```
sns.lmplot(x='viewCount', y='viewTraffic',
           data=df, scatter_kws={"s": 5})
```

```
<seaborn.axisgrid.FacetGrid at 0x226858376a0>
```



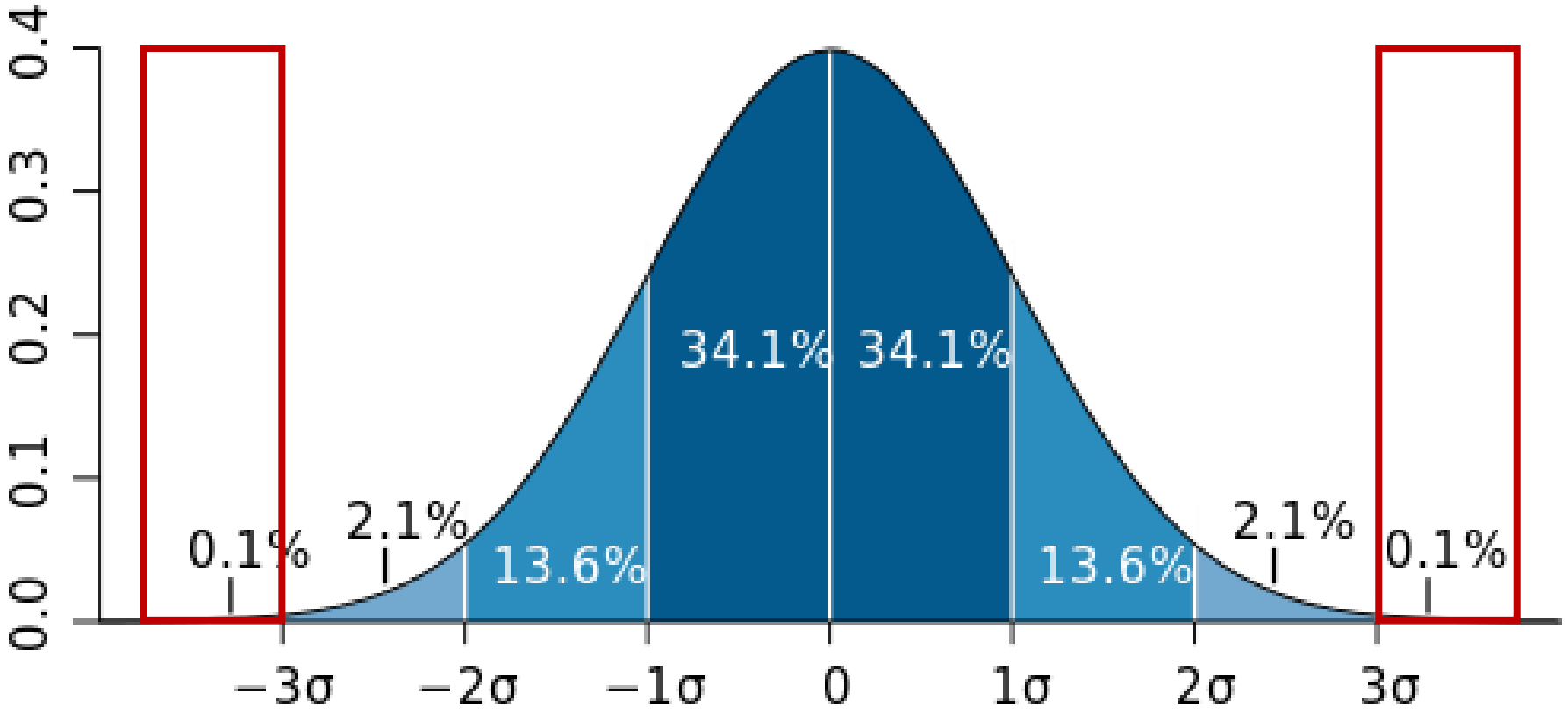
Handling Outliers

- Clipping using Normal Distribution

- Clipping using IQR Score

Clipping using Normal Distribution (1)

- Remove outliers outside of the $\boldsymbol{\mu \pm n\sigma}$ (e.g., n = 3)



Clipping using Normal Distribution (2)

- Clipping data

```
for c in df.columns:
    df = df[np.abs(df[c] - df[c].mean()) <= 3 * df[c].std()]
df.head()
```

$= |x - \mu| \leq 3\sigma$

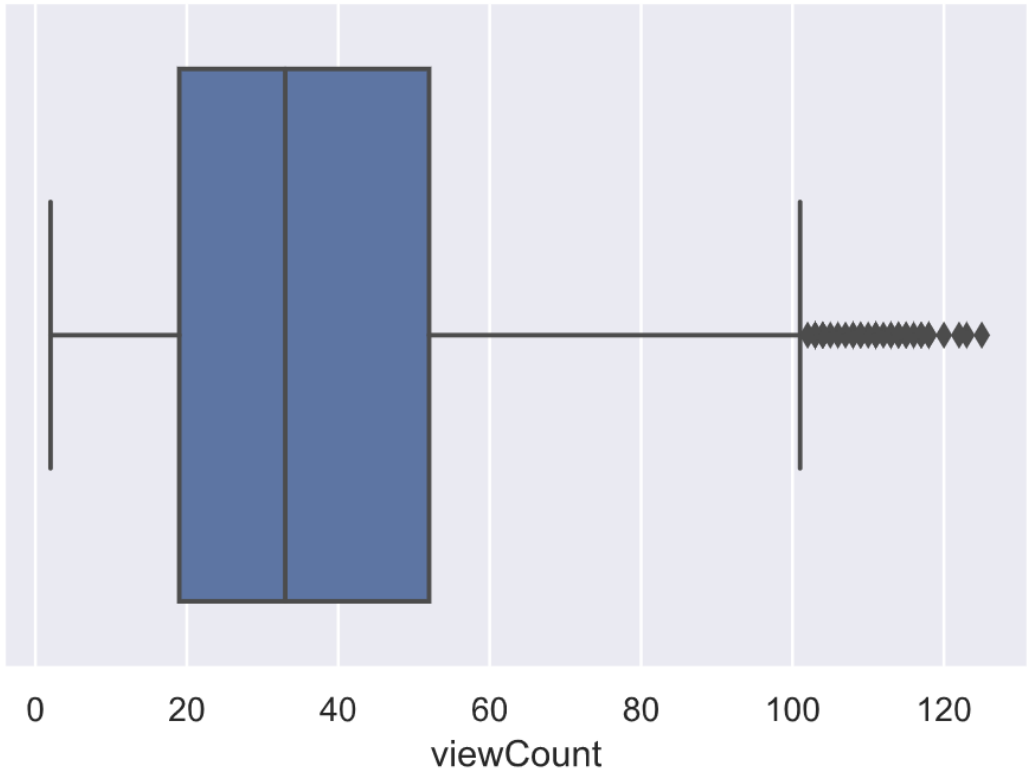
iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0
10100017371337	95.0	19.0	0.0	12.0	0.0	...	0.0	25970473.0	8772492.0	0.0	34742965.0
10100013627062	75.0	15.0	0.0	3.0	0.0	...	0.0	1289983.0	271360.0	0.0	1561343.0
10100012989173	49.0	0.0	2.0	13.0	0.0	...	0.0	2071600.0	51129.0	0.0	2122729.0

Clipping using Normal Distribution (3)

- After clipping

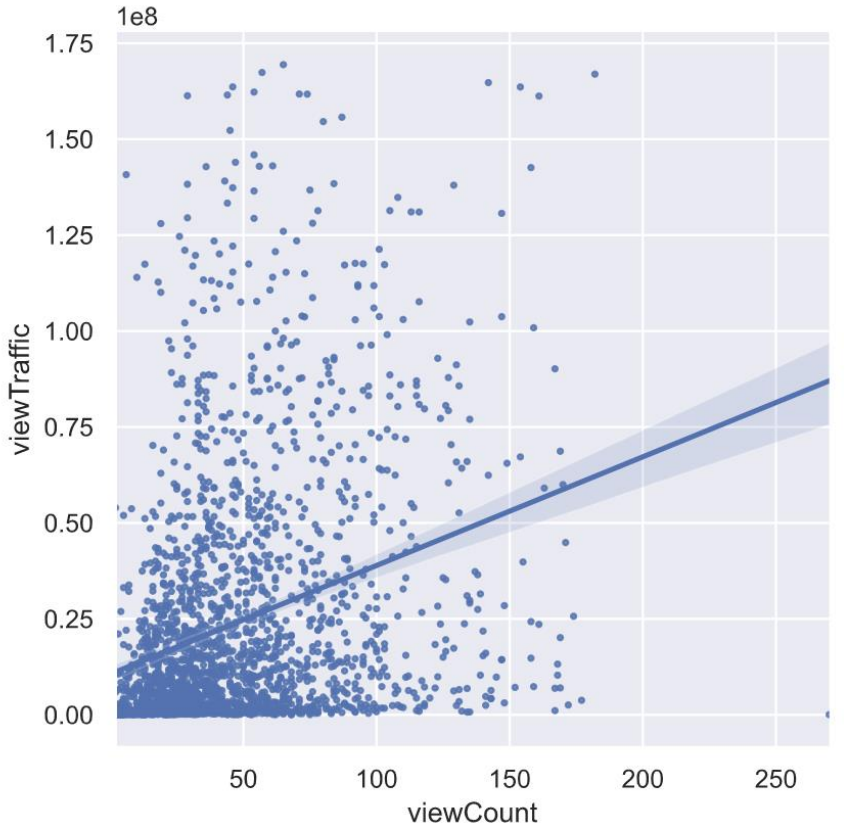
```
sns.boxplot(x=df.viewCount)
```

```
<AxesSubplot:xlabel='viewCount'>
```




```
sns.lmplot(x='viewCount', y='viewTraffic',
           data=df, scatter_kws={"s": 5})
```

```
<seaborn.axisgrid.FacetGrid at 0x22685944f10>
```



Z-Score (Standard Score)

- The number of standard deviations by which the value of a raw score is above or below the mean value

$$z = \frac{x - \mu}{\sigma}$$

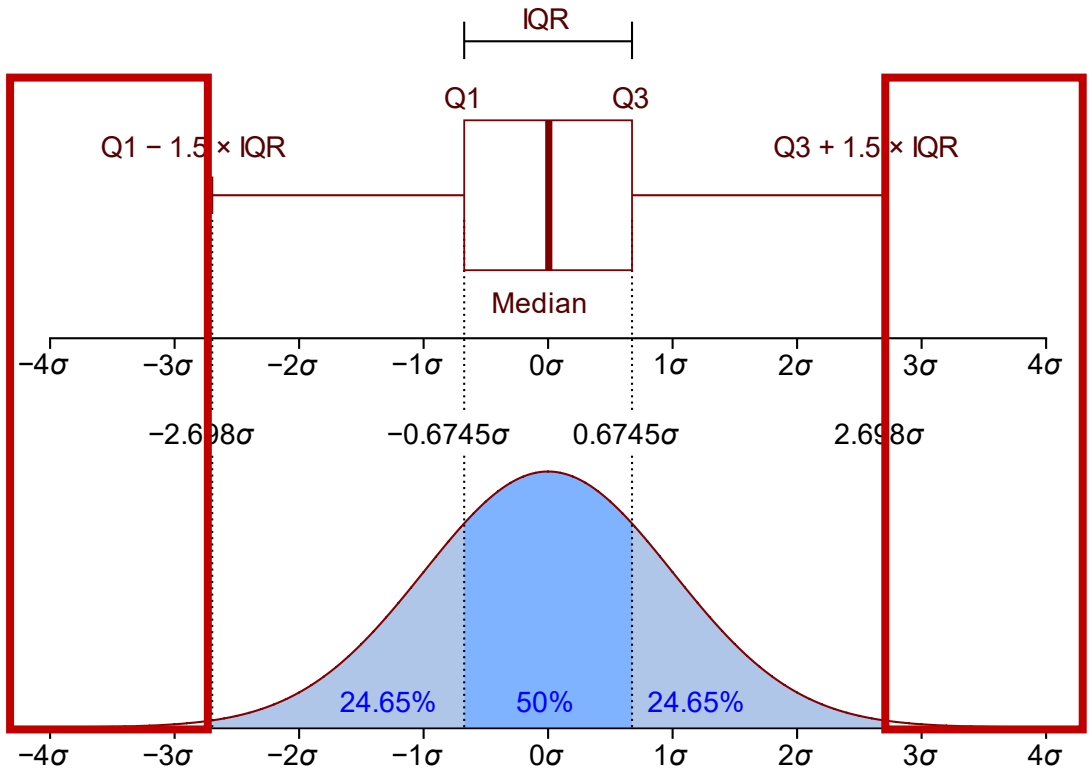
  - Re-scale data to normal distribution
  - In most cases, the data with $|z| > 3$ are considered outliers -- clipping those data is same as clipping data such as $|x - \mu| > 3\sigma$

```
from scipy import stats

for c in df.columns:
    df = df[np.fabs(stats.zscore(df[c])) <= 3]
df.head()
```

iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100039309679	40.0	10.0	2.0	3.0	0.0	...	1.0	2719076.0	88398.0	0.0	2807474.0
10100037687198	44.0	1.0	0.0	0.0	0.0	...	0.0	28866560.0	6246400.0	0.0	35112960.0

Clipping using IQR Score (1)

- IQR (InterQuartile Range)
  - Q_1 = 25%, Q_3 = 75%, IQR = Q_3 - Q_1
  - Remove data points outside of [Q_1 - 1.5IRQ, Q_3 + 1.5IQR]



Clipping using IQR Score (2)

- *np*.percentile(*a*, *q*, [*axis*], ...)
  - Compute the q-th percentile of the data along the specified axis
  - *a*: input array
  - *q*: percentile or sequence of percentiles to compute in [0, 100]
  - *axis*: axis or axes along which the percentiles are computed
- *df*.quantile(*q*, [*axis*], ...)
  - Return values at the given quantile over requested axis

```
a = np.random.randint(0, 20, 10)
a = np.sort(a)
```

```
array([ 4,  5,  5,  9, 13, 13, 14, 14, 17, 18])
```

```
np.percentile(a, 50)
```

```
13.0
```

```
np.percentile(a, [25, 75])
```

```
array([ 6., 14.])
```

```
a[np.where(a > 14)]
```

```
array([17, 18])
```

Clipping using IQR Score (3)

- Clipping data

```
for c in df.columns:
    Q1 = df[c].quantile(0.25)
    Q3 = df[c].quantile(0.75)
    IQR = Q3 - Q1
    df = df[(df[c] >= (Q1 - 1.5 * IQR)) & (df[c] <= (Q3 + 1.5 * IQR))]
df.head()
```

iduser	viewCount	editCount	shareCount	searchCount	coworkCount	...	exportCount	viewTraffic	editTraffic	exportTraffic	traffic
10100034231482	29.0	0.0	0.0	0.0	0.0	...	0.0	6401717.0	0.0	0.0	6401717.0
10100011294549	74.0	6.0	0.0	0.0	0.0	...	0.0	8749387.0	435164.0	0.0	9184551.0
10100020127806	26.0	0.0	0.0	0.0	0.0	...	0.0	10857632.0	0.0	0.0	10857632.0
10100030084140	28.0	6.0	0.0	1.0	0.0	...	0.0	37385701.0	252178.0	0.0	37637879.0
10100027809274	14.0	0.0	0.0	0.0	0.0	...	0.0	4257850.0	304324.0	0.0	4562174.0

Data Scaling

SK-Learn

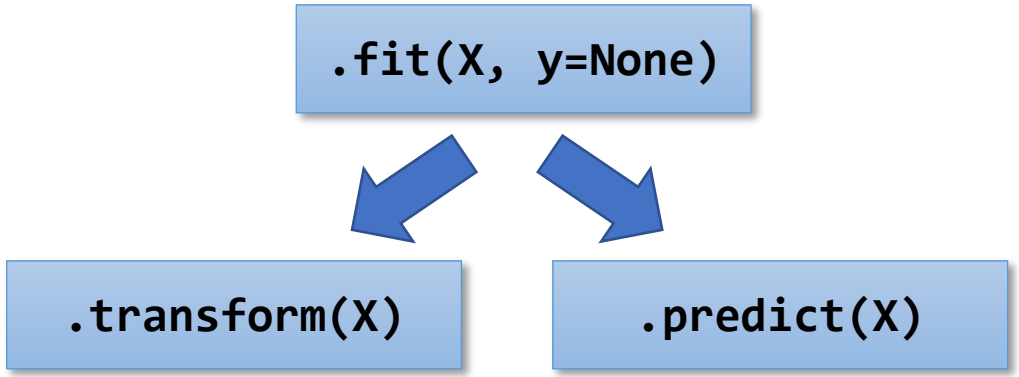
- SciKit (SciPy Toolkit)-learn, Sklearn, or SK-Learn
- Open source machine learning library for Python
- Built on top of SciPy
  - Designed to interoperate with Python numerical and scientific library
- Dependency
  - NumPy, SciPy, Matplotlib
- Open source (https://scikit-learn.org)
  - Initially developed by David Cournapeau as a "Google Summer of Code" project in 2007
  - Still under active development (v0.24 as of January 2021)

SK-Learn Modules

- Classification: identify to which category an object belongs to
  - Regression: predict continuous-valued attribute (linear, logistic, etc.)
  - SVM, Decision tree, Neural nets, Nearest neighbors, ...
- Clustering: grouping of similar objects
  - K-means, Hierarchical clustering, etc.
- Model selection: validate and choosing parameters and model
  - Cross validation, metrics, etc.
- Preprocessing: feature extraction & normalization
- Dimensionality reduction: reducing number of variables
  - PCA, Feature selection, etc.
- Datasets

Structure of SK-Learn

- `.fit()`
  - Build a model using the (training) data **X**
  - Requires the **y** values for classification and prediction



- `.predict()`
  - Predict the **y** values for the test data **X** based on the model
  - Perform classification, regression, clustering, etc.
- `.transform()`
  - Transform the input data **X** based on the model
  - Perform preprocessing, dimensionality reduction, feature extraction, feature selection, etc.

Example: StandardScaler()

- StandardScaler transform data using the following equation:

$$\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$$

- `scaler.fit()`
  - Build a model using the given data
  - Compute mean and variance
- `scaler.transform()`
  - Transform X to X_ using the model

```
import numpy as np
import sklearn.preprocessing as prep
```

```
X = np.arange(5, dtype='float').reshape(5, 1)
X
```

```
array([[0.],
       [1.],
       [2.],
       [3.],
       [4.]])
```

```
scaler = prep.StandardScaler()
scaler.fit(X)
scaler.mean_, scaler.var_
```

```
(array([2.]), array([2.]))
```

```
X_ = scaler.transform(X)
X_
```

```
array([[-1.41421356],
       [-0.70710678],
       [ 0.        ],
       [ 0.70710678],
       [ 1.41421356]])
```

Example: LinearRegression()

- Linear regression

- `regr.fit()`
  - Build a model for linear regression
  - $y = -0.42+3.58x_1-0.69x_2-1.22x_3$

- `regr.predict()`
  - Predict the y value for the test data X using the model

```
import numpy as np
from sklearn import linear_model

X = np.random.random((5,3))
y = np.random.random((5,1))
X, y
```

```
array([[0.34523274, 0.03465153, 0.49879222],
       [0.34154268, 0.558655  , 0.05047529],
       [0.55781272, 0.93527388, 0.59078667],
       [0.7568254 , 0.57266255, 0.90788885],
       [0.42153745, 0.09800326, 0.69636864]]) array([[0.02940408],
                                                     [0.41239473],
                                                     [0.1845568 ],
                                                     [0.78603924],
                                                     [0.33245886]])
```

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
regr.coef_, regr.intercept_
```

```
(array([[ 3.58372982, -0.68867795, -1.21704883]]), array
([-0.41676285]))
```

```
test = np.random.random((2,3))
print('test\n', test)
regr.predict(test)
```

```
test
 [[0.1394362  0.8556813  0.437153  ]
 [0.56918605 0.05113164 0.42308297]]

array([[-1.03838658],
       [ 1.07292029]])
```

Why Data Scaling?

- Features in a dataset can have very different scales

- Unscaled data can degrade the predictive performance of many machine learning algorithms
  - Many estimators assume that each feature takes values close to zero and all features vary on comparable scales
  - Metric-based and gradient-based estimators often assume approximately standardized data (normal distribution)
  - (cf.) Decision tree-based estimators are robust to arbitrary scaling of the data

- Unscaled data can slow down or even prevent the convergence of many gradient-based estimators

Data Scaling

- Standard scaling: → $\tilde{x}_i$ ~ Normal distribution ($\mu = 0, \sigma = 1$) $\tilde{x}_i = \frac{x_i - mean(x)}{std(x)}$
- Min-Max Scaling: → $\tilde{x}_i$ in [0, 1] $\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$
- Max-Abs Scaling: → $|\tilde{x}_i| \leq 1$ $\tilde{x}_i = \frac{x_i}{\max(|x|)}$
- Robust Scaling: → Based on median and IQR $\tilde{x}_i = \frac{x_i - median(x)}{Q3(x) - Q1(x)}$

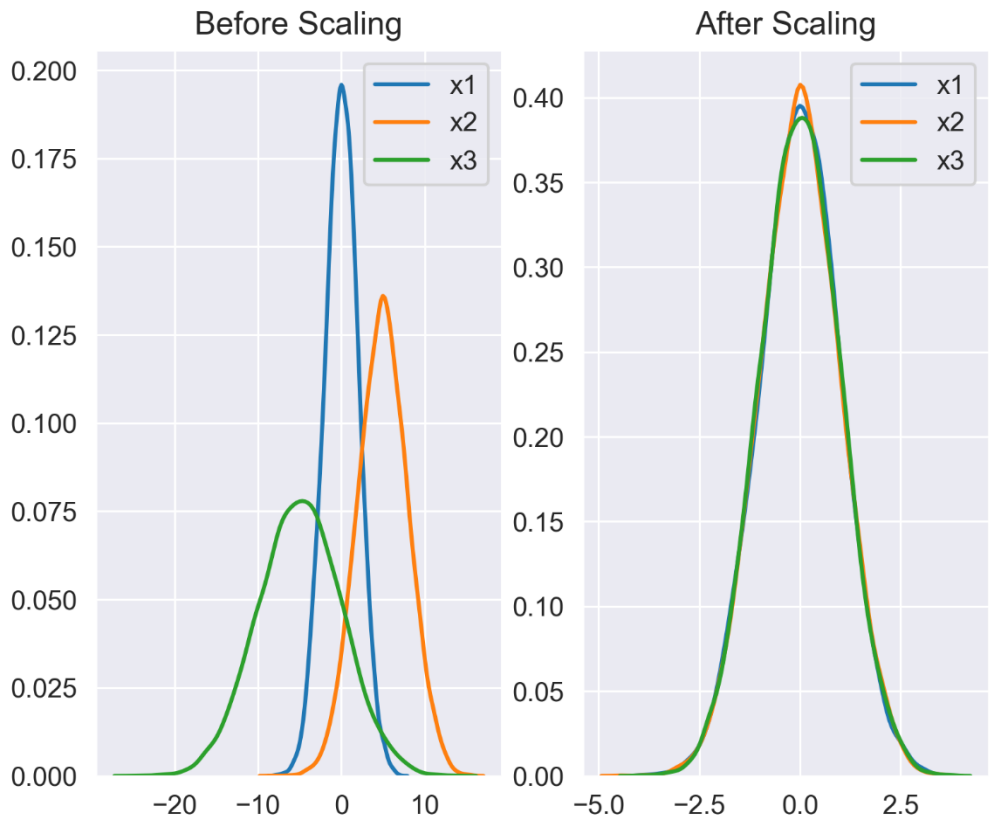
Data Scaling supported by SK-Learn

Scaling	Function	Class
Standard	`scale(x)`	`StandardScaler`
Min-Max	`minmax_scale()`	`MinMaxScaler`
Max-Abs	`maxabs_scale()`	`MaxAbsScaler`
Robust	`robust_scale()`	`RobustScaler`

Standard Scaling

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing

df = pd.DataFrame({
        'x1': np.random.normal(0, 2, 10000),
        'x2': np.random.normal(5, 3, 10000),
        'x3': np.random.normal(-5, 5, 10000)
})
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

sns.set_style('darkgrid')
_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

$$\widetilde{x_i} = \frac{x_i - mean(x)}{std(x)}$$

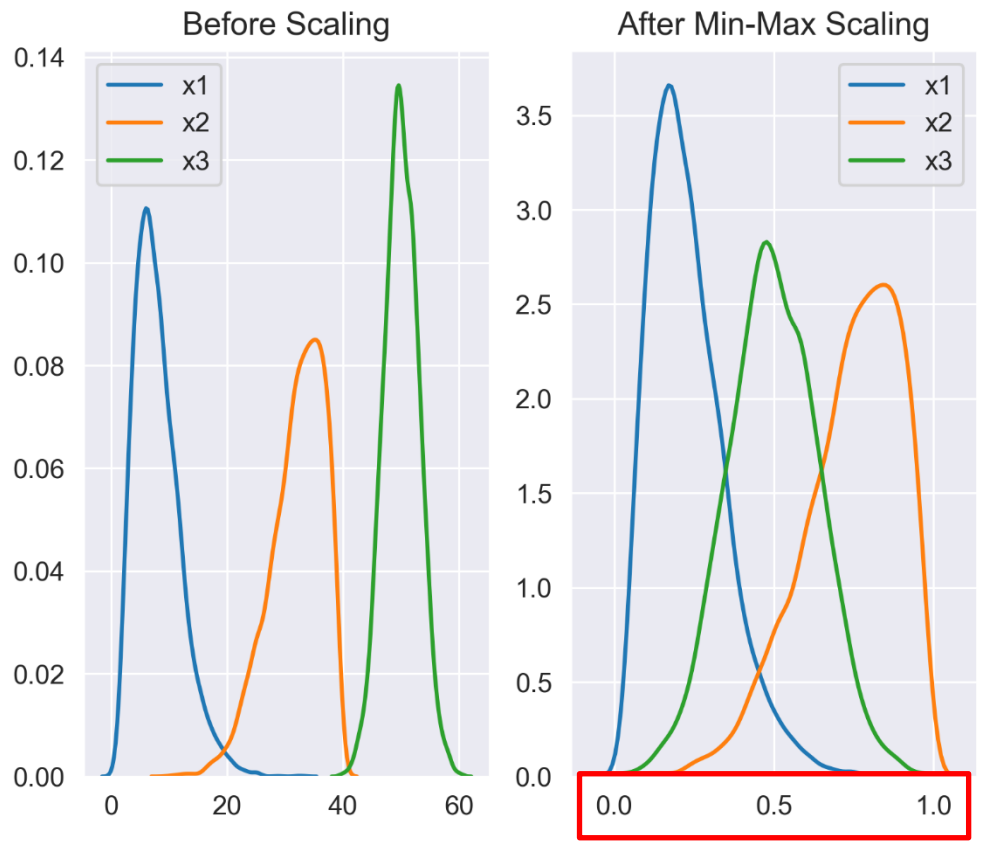


Min-Max Scaling

- All values are mapped in the range [0, 1]
- Very sensitive to the presence of outliers

$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

```
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),     # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,    # negative skew
        'x3': np.random.normal(50, 3, 10000)     # no skew
})

scaler = preprocessing.MinMaxScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Min-Max Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```

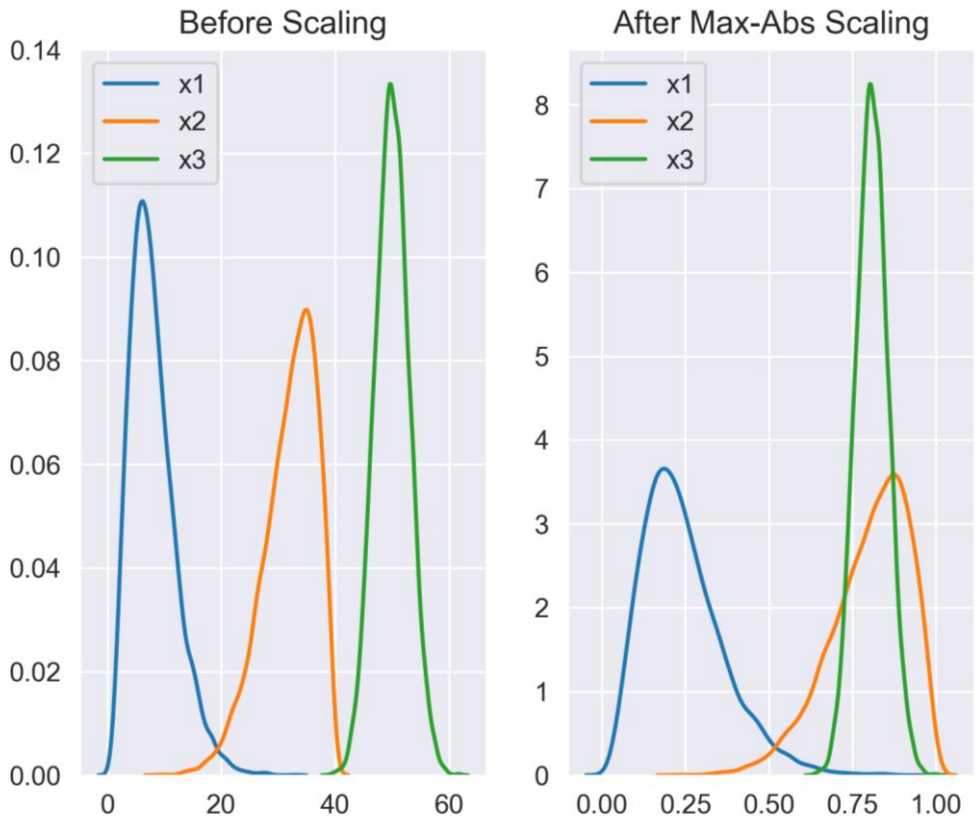


Max-Abs Scaling

- Doesn't change the shape of the distribution
- Also suffers from the presence of large outliers

$$\tilde{x}_i = \frac{x_i}{\max(|x|)}$$

```
df = pd.DataFrame({
        'x1': np.random.chisquare(8, 10000),    # positive skew
        'x2': np.random.beta(8, 2, 10000)*40,   # negative skew
        'x3': np.random.normal(50, 3, 10000)    # no skew
})

scaler = preprocessing.MaxAbsScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2', 'x3'])

_, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)

ax2.set_title('After Max-Abs Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
```



Robust Scaling (1)

- Based on percentiles
- Not influenced by a few number of very large marginal outliers

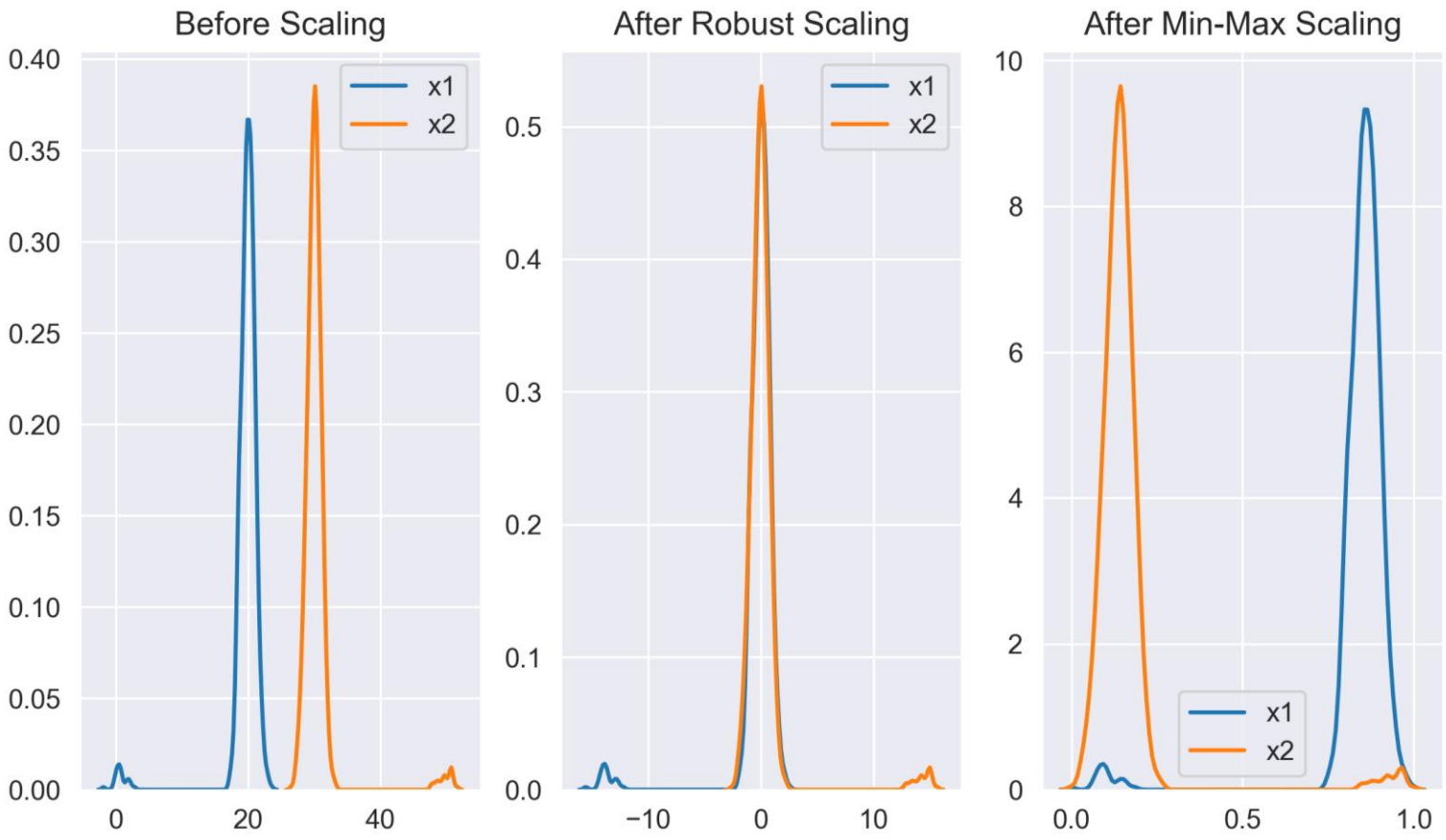
$$\widetilde{x_i} = \frac{x_i - median(x)}{Q3(x) - Q1(x)}$$

```
df = pd.DataFrame({
        # distribution with lower outliers
        'x1': np.hstack((np.random.normal(20,1,1000),
                         np.random.normal(1,1,25))),
        # distribution with upper outliers
        'x2': np.hstack((np.random.normal(30,1,1000),
                         np.random.normal(50,1,25)))
})

robust_scaler = preprocessing.RobustScaler()
robust_df = robust_scaler.fit_transform(df)
robust_df = pd.DataFrame(robust_df, columns=['x1', 'x2'])

minmax_scaler = preprocessing.MinMaxScaler()
minmax_df = minmax_scaler.fit_transform(df)
minmax_df = pd.DataFrame(minmax_df, columns=['x1', 'x2'])
```

```
_, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(9,5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df['x1'], ax=ax2)
sns.kdeplot(robust_df['x2'], ax=ax2)

ax3.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df['x1'], ax=ax3)
sns.kdeplot(minmax_df['x2'], ax=ax3)
```

Robust Scaling (2)

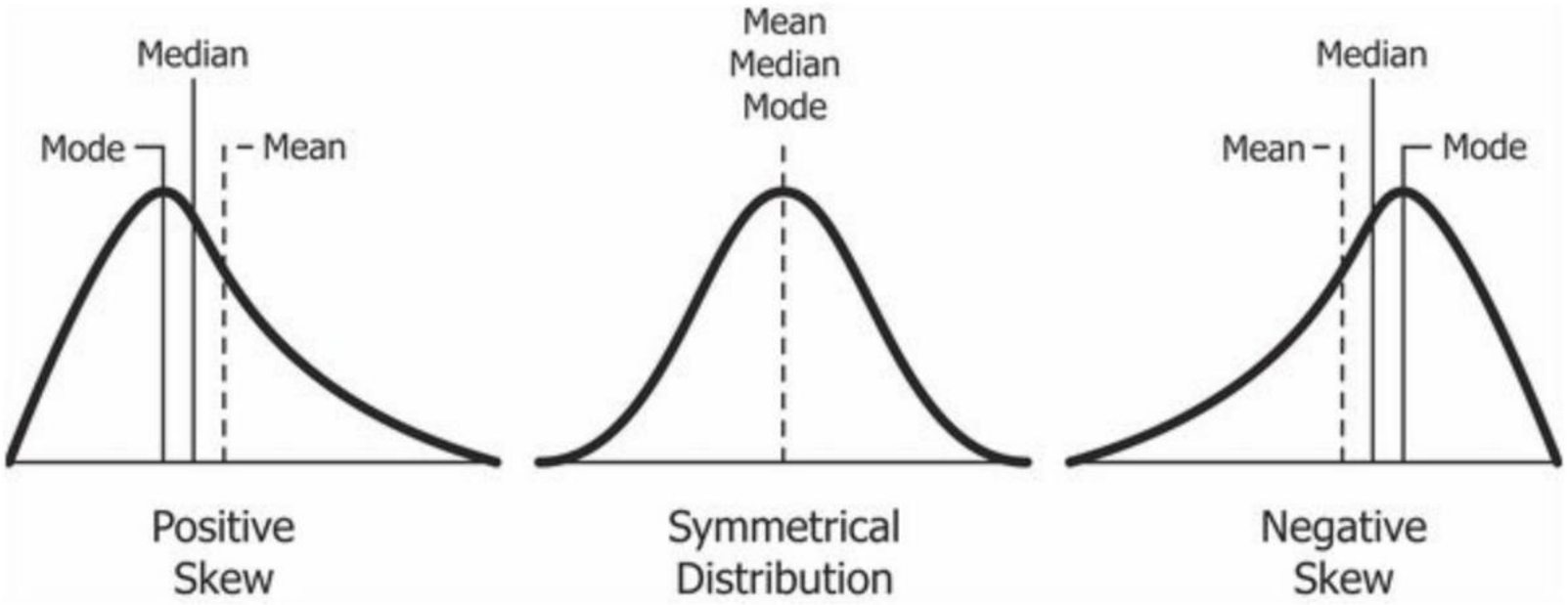
- Min-Max Scaling
  - Significantly affected by outliers
- Robust Scaling
  - Inliers are in [-2, 2]
  - Outliers still exist at the end of each distribution



Data Standardization

Data Skewness

- A measure of asymmetry of a distribution
  - Symmetrical (skewness = 0, e.g., normal distribution): mean == median == mode
  - Positive skew (skewness > 0): tail at right, mode > median > mean
  - Negative skew (skewness < 0): tail at left, mean < median < mode



Measuring Data Skewness

- *df*.skew([*axis*], [*skipna*], ...)
  - Return unbiased skew over requested axis
  - *axis*: axis for the function to be applied on
  - *skipna*: if True, exclude null values when computing the result (default True)

- Meaning of skewness value
  - -0.5 <= skewness <= 0.5: fairly symmetrical
  - -1 < skewness < -0.5 or 0.5 < skewness < 1: moderately skewed
  - skewness < -1 or skewness > 1: highly skewed

Handling Data Skewness

- Linear model performs better when the dataset follows normal distribution
- Dealing with positive skewness
  - Square root transformation (x to $x^{1/2}$)
  - Cube root transformation (x to $x^{1/3}$)
  - Log transformation (x to $\log_2 x, \log_e x, \ln x, ...$)
- Dealing with negative skewness
  - Square transformation (x^2)
  - Cube transformation (x^3)
  - Reflect the values and apply the methods used to reduce the positive skewness

The Boston Housing Dataset

- Dataset for housing values in areas of Boston in 70's
- 506 rows, 14 columns (13 attributes + housing value)
- Available in the SK-Learn datasets

CRIM: 범죄율
ZN: 25,000ft^2 초과 거주지역 비율
INDUS: 비소매상업지역 면접 비율
CHAS: 찰스강 경계에 위치한 경우 1
NOX: 일산화질소 농도
AGE: 1940년 이전 건축된 주택 비율
RM: 주택당 방 수
RAD: 방사형 고속도로까지의 거리
LSTAT: 인구 중 하위 계층 비율
DIS: 직업 센터의 거리
B: 인구 중 흑인 비율
TAX: 재산세율
PTRATIO: 학생/교사 비율
MEDV: 주택 가격의 median (단위: $1,000)

```
from sklearn import datasets
import pandas as pd
boston = datasets.load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

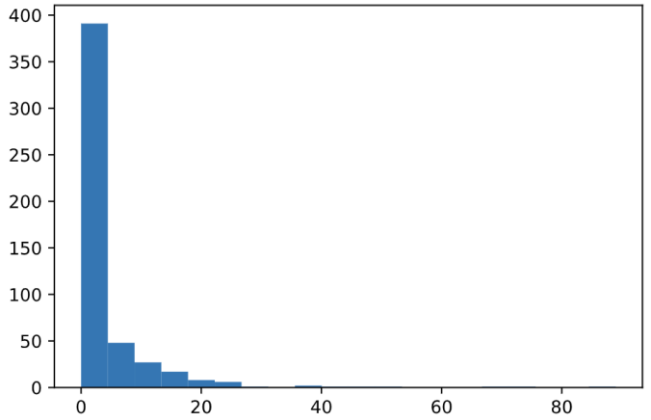
Skewness in Boston Housing Dataset

```
df.skew()
```

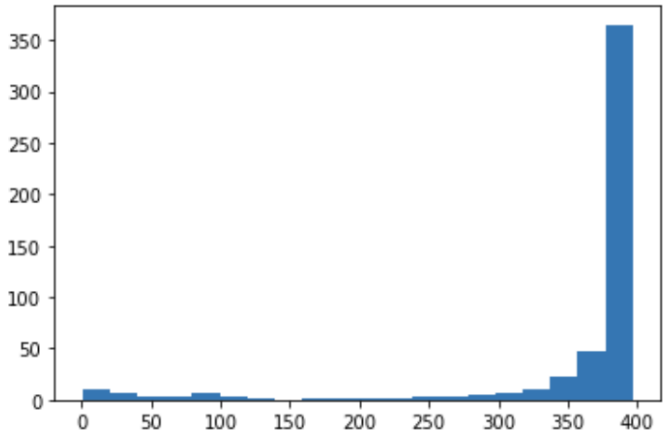
CRIM	5.223149
ZN	2.225666
INDUS	0.295022
CHAS	3.405904
NOX	0.729308
RM	0.403612
AGE	-0.598963
DIS	1.011781
RAD	1.004815
TAX	0.669956
PTRATIO	-0.802325
B	-2.890374
LSTAT	0.906460

dtype: float64

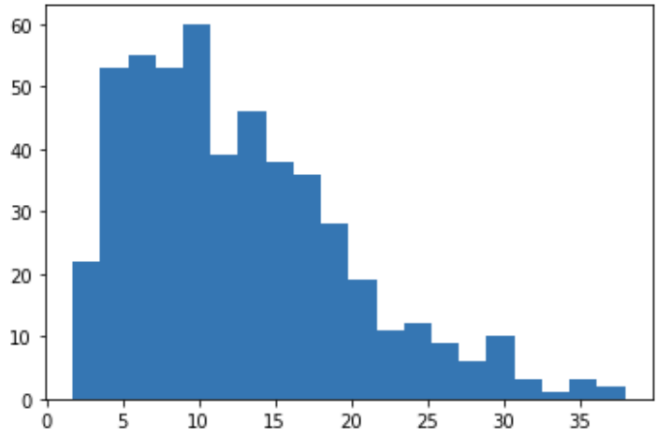
```
plt.hist(df.CRIM, bins=20)
```



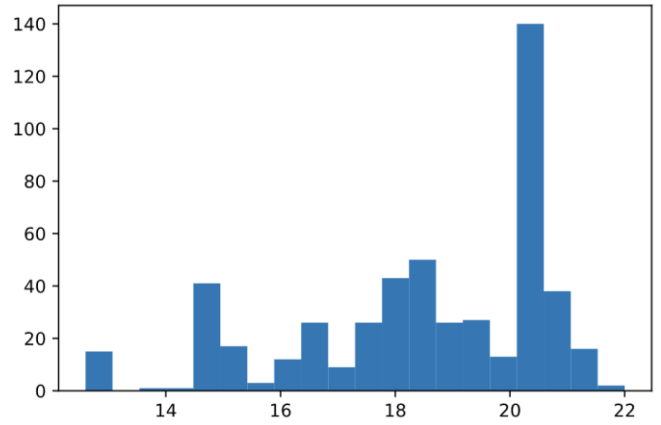

```
plt.hist(df.B, bins=20)
```




```
plt.hist(df.LSTAT, bins=20)
```




```
plt.hist(df.PTRATIO, bins=20)
```



Transforming Data (1)

- *sklearn.preprocessing.*`scale(`*X, ...*`)`
  - Standardize a dataset along any axis
  - Center to the zero mean and component wise scale to unit variance
  - *X*: the data to center and scale

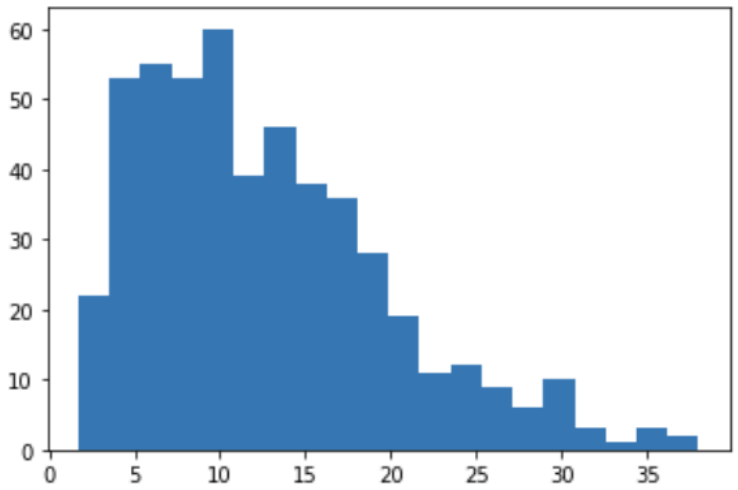
```
from sklearn import preprocessing

df['LSTAT_log'] = preprocessing.scale(np.log(df['LSTAT']+1))
df['LSTAT_sqrt'] = preprocessing.scale(np.sqrt(df['LSTAT']+1))
df[['LSTAT', 'LSTAT_log', 'LSTAT_sqrt']].skew()
```

```
LSTAT         0.906460
LSTAT_log    -0.187195
LSTAT_sqrt    0.359606
dtype: float64
```

Transforming Data (2)

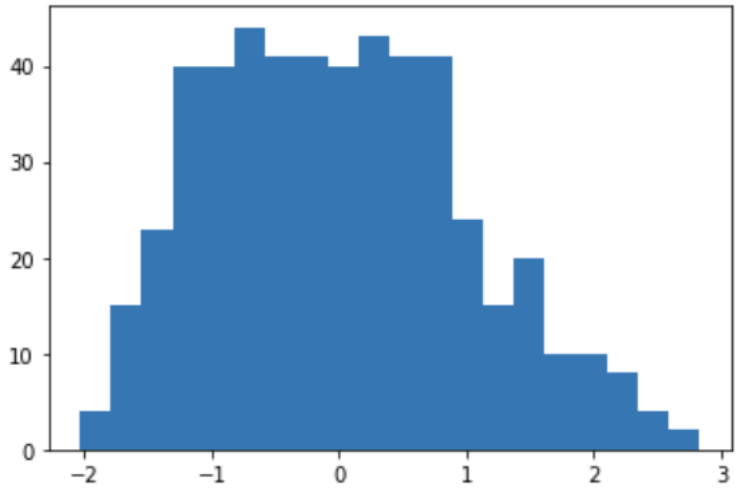
Original data

```
import matplotlib.pyplot as plt

plt.hist(df['LSTAT'], bins=20)
plt.show()
```



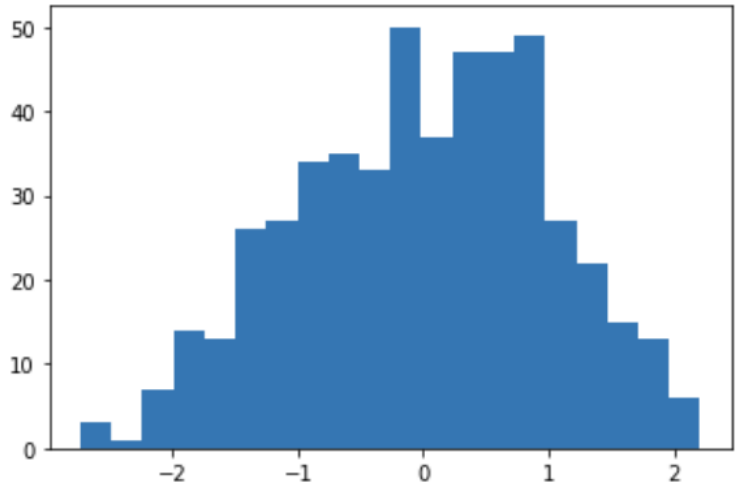
Square root transformation

```
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_sqrt'], bins=20)
plt.show()
```



Log transformation

```
import matplotlib.pyplot as plt

plt.hist(df['LSTAT_log'], bins=20)
plt.show()
```



Data Encoding

Approaches

- Pitfall: Label Encoding
  - Substitute each label with a corresponding number
  - Simplest
  - Can be "misinterpreted" by the algorithms (order and distance between labels?)

Label	Encoding
'dog'	0
'cat'	1
'rabbit'	2

- One-Hot Encoding
  - Most common, correct way to deal with non-ordinal categorical data
  - Represent a label as a vector of 0's and 1's

Label	Encoding
'dog'	(1, 0, 0)
'cat'	(0, 1, 0)
'rabbit'	(0, 0, 1)

SK-Learn LabelEncoder()

- *sklearn.preprocessing.*`LabelEncoder()`
  - Encode target labels with value between 0 and n_classes-1
- Methods
  - `fit`(X, [y]): fit label encoder
  - `transform`(X): transform labels to normalized encoding
  - `inverse_transform`(y): transform labels back to original encoding

```
from sklearn.preprocessing import LabelEncoder
X = ['A', 'B', 'A', 'A', 'B', 'C', 'C', 'A', 'C', 'B']
le = LabelEncoder()
le.fit_transform(X)
```

```
array([0, 1, 0, 0, 1, 2, 2, 0, 2, 1], dtype=int64)
```

SK-Learn OneHotEncoder()

- *sklearn.preprocessing*.`OneHotEncoder`([*sparse*], ...)
  - Encode categorial features using a one-hot numeric array
  - *sparse*: if True, return sparse matrix else return an array (default: True)

- Attributes
  - *categories_*: the categories of each feature determined during fitting

- Methods
  - `fit`(X, [y]): fit OneHotEncoder to X
  - `transform`(X): transform X using one-hot encoding
  - `inverse_transform`(X): convert the data back to the original representation

OneHotEncoder(): 1D Data

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
X = np.array(['a', 'b', 'a', 'c']).reshape(-1, 1)
ohe.fit(X)
X_encode = ohe.transform(X)
type(X_encode)
```

The result of OneHotEncoder() is a SciPy's sparse matrix →

```
scipy.sparse.csr.csr_matrix
```

```
X_encode.toarray()
```

`toarray()` converts sparse matrix to dense matrix

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.]])
```

```
ohe.inverse_transform([[0., 1., 0.]])
```

```
array([['b']], dtype='<U1')
```

OneHotEncoder(): 2D Data

```
X = np.array([[0, 0, 4],
              [1, 1, 0],
              [0, 2, 1],
              [1, 0, 2]])

ohe.fit(X)
X_encode = ohe.transform(X)
X_encode.toarray()
```

```
array([[1., 0., 1., 0., 0., 0., 0., 0., 1.],
       [0., 1., 0., 1., 0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 1., 0., 1., 0., 0.],
       [0., 1., 1., 0., 0., 0., 0., 1., 0.]])
```

```
ohe.inverse_transform([[1., 0., 0., 0., 1., 0., 0., 0., 1.]])
```

```
array([[0, 2, 4]], dtype=int32)
```

Using OneHotEncoder() in Pandas

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
df_encode = pd.DataFrame(ohe.fit_transform(df[['Animal']]).toarray())
df_ohe = df.join(df_encode)
df_ohe
```

	Animal	0	1	2
0	cat	1.0	0.0	0.0
1	dog	0.0	1.0	0.0
2	rabbit	0.0	0.0	1.0
3	cat	1.0	0.0	0.0
4	dog	0.0	1.0	0.0

	Animal
0	cat
1	dog
2	rabbit
3	cat
4	dog

Pandas One-Hot Encoding

- *pd*.get_dummies(*data*, [*columns*], ...)
  - Convert categorical variable into dummy/indicator variables
  - *data*: data of which to get dummy indicators
  - *columns*: column names in the DataFrame to be encoded

```
df_encode = pd.get_dummies(df, columns=['Animal'])
df_ohe = df.join(df_encode)
df_ohe
```

	Animal
0	cat
1	dog
2	rabbit
3	cat
4	dog

	Animal	Animal_cat	Animal_dog	Animal_rabbit
0	cat	1	0	0
1	dog	0	1	0
2	rabbit	0	0	1
3	cat	1	0	0
4	dog	0	1	0

SK-Learn Binarizer()

- *sklearn.preprocessing*.`Binarizer`([*threshold*], ...)
  - Binarizer data (set feature values to 0 or 1) according to a threshold
  - *threshold*: feature values below or equal to this are replaced by 0, above it by 1 (default: 0.0)

- Methods
  - `fit`(X, [y]): do nothing
  - `transform`(X): binarize each element of X

Binarizer() Example

```
from sklearn.preprocessing import Binarizer
X = np.array([[1., -1.,  2.],
              [2., -3.,  1.],
              [0.,  1., -1.]])
bin = Binarizer()
bin.transform(X)
```

```
array([[1., 0., 1.],
       [1., 0., 1.],
       [0., 1., 0.]])
```

```
bin = Binarizer(threshold = 1)
bin.transform(X)
```

```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 0.]])
```

Sampling for Imbalanced Data

Imbalanced Data

- A problem with classification where the classes are not represented equally
  - The model will be mostly tuned for the majority class
- Example:
  - A dataset with Class A : Class B = 9 : 1
  - The percentage of correct answers in the test dataset will also be 9 : 1
  - Even if a model classifies everything to Class A, it will have a 90% of accuracy
- Solutions: Balance data using sampling
  - Oversampling: increase the amount of minority class
  - Undersampling: use only part of majority class

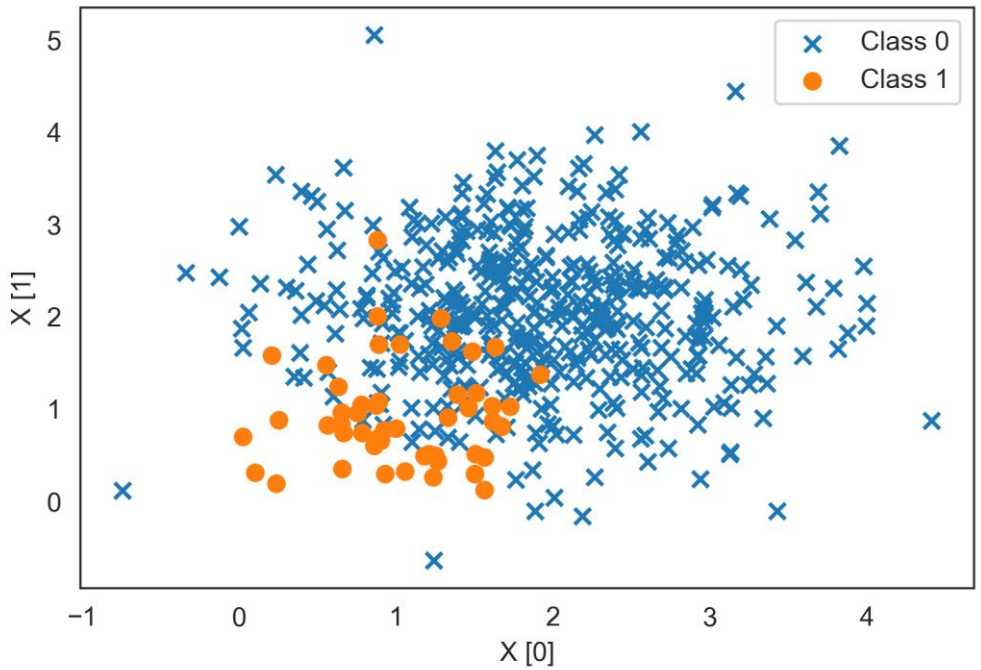
imbalanced-learn module

- A python package offering a number of re-sampling techniques
- Commonly used for datasets showing strong between-class imbalance
- Part of scikit-learn-contrib projects
- https://github.com/scikit-learn-contrib/imbalanced-learn
- Installation
  - pip install -U imbalanced-learn
  - conda install -c conda-forge imbalanced-learn

```
>>> import imblearn.under_sampling
>>> import imblearn.over_sampling
```

Creating Imbalanced Data

```
def plot(X, y):
    plt.scatter(X[y==0, 0], X[y==0, 1], marker='x', label='Class 0')
    plt.scatter(X[y==1, 0], X[y==1, 1], marker='o', label='Class 1')
    plt.xlabel('X [0]')
    plt.ylabel('X [1]')
    plt.legend()

n0 = 450
n1 = 50

a = np.random.randn(n0, 2)*0.8 + 2  # N(2, 0.8)
b = np.random.randn(n1, 2)*0.5 + 1  # N(1, 0.5)

X = np.vstack([a, b])
y = np.hstack([np.zeros(n0), np.ones(n1)])

plot(X, y)
```

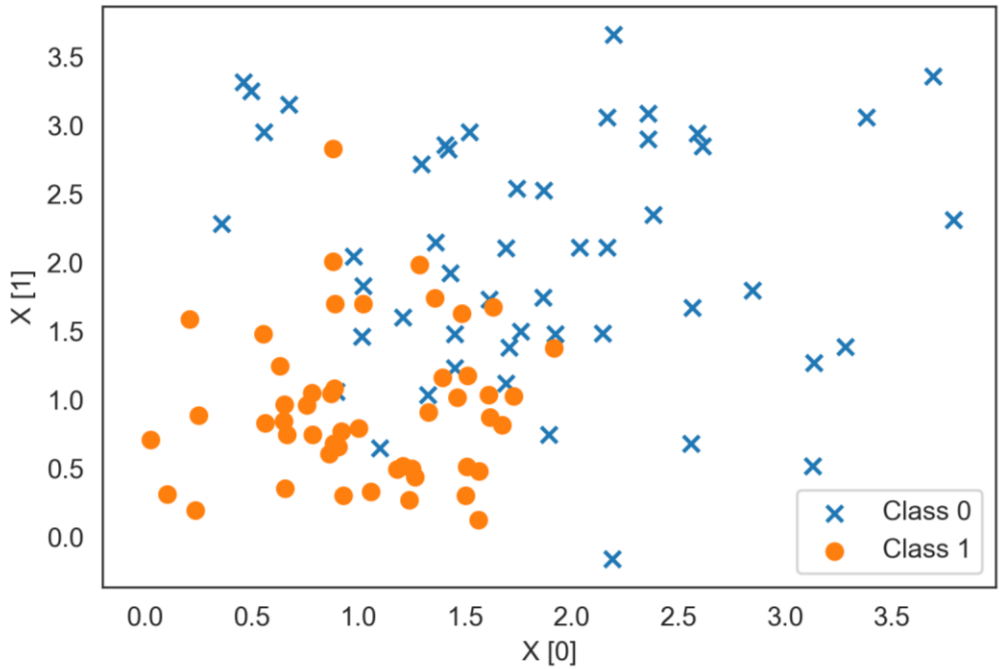


Undersampling: RandomUnderSampler()

- Under-sample the majority class by randomly picking samples

```
from imblearn.under_sampling import RandomUnderSampler

X_samp, y_samp = RandomUnderSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(100, 2) (100,)
```

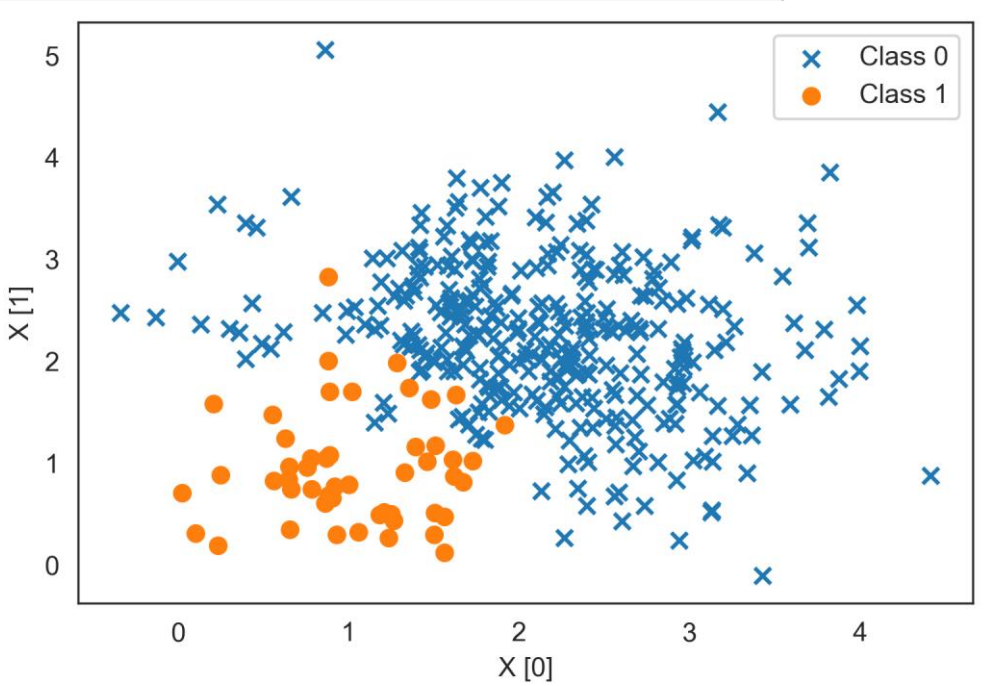


Undersampling: EditedNearestNeighbours()

- Keep a sample if all or majority of the NN's belong to the same class

```
from imblearn.under_sampling import EditedNearestNeighbours

X_samp, y_samp = EditedNearestNeighbours(kind_sel='all', n_neighbors=10,
                    random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(388, 2) (388,)
```



- *n_neighbors*: size of the neighbourhood to consider to compute the nearest neighbors
- *kind_sel*: 'all' (all have to agree to keep),
  'mode' (majority vote to keep)

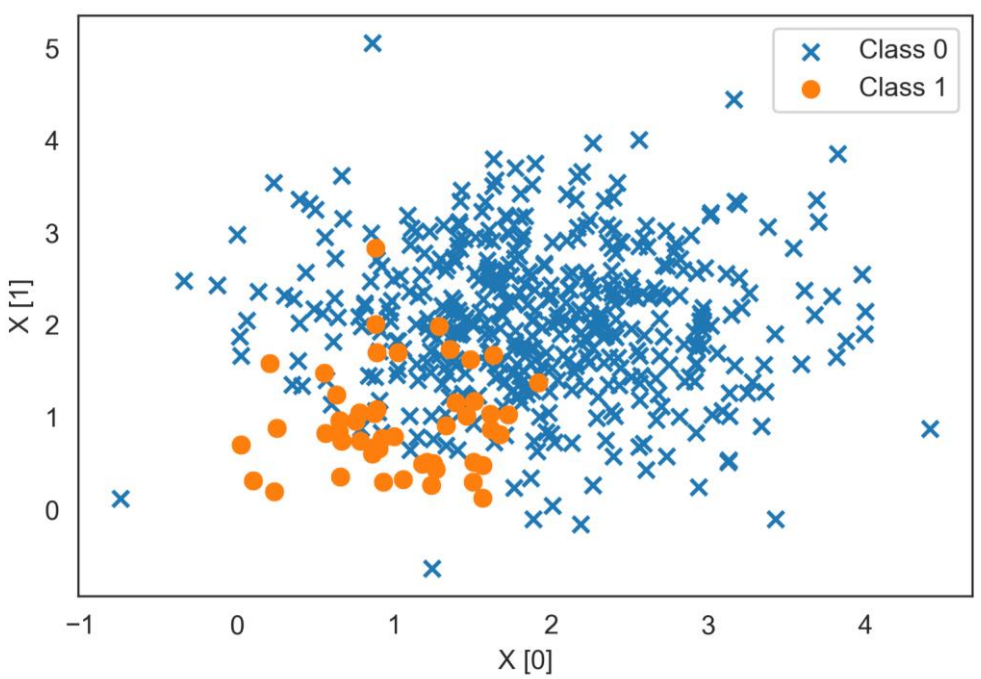
Oversampling: RandomOverSampler()

- Over-sample the minority class by picking samples at random

```
from imblearn.over_sampling import RandomOverSampler

X_samp, y_samp = RandomOverSampler(random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```

- Graph looks same, but the count has increased to 450

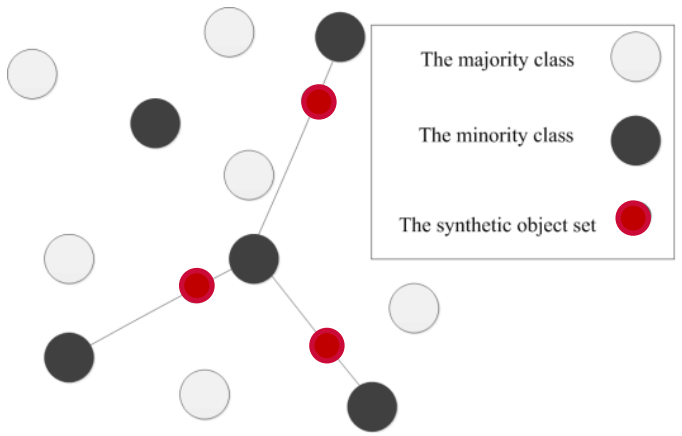


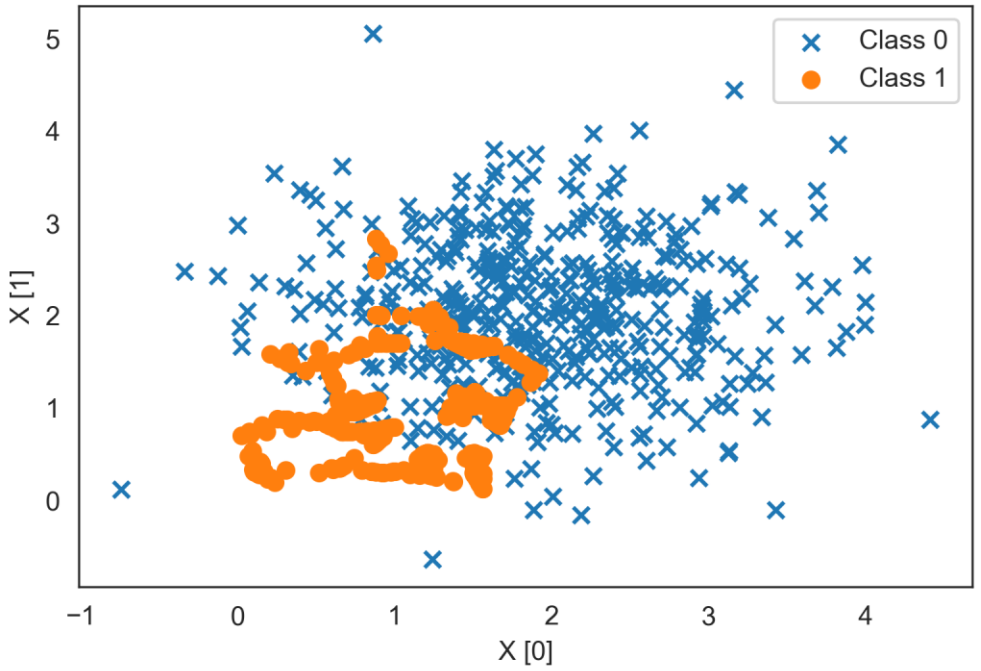
Oversampling: SMOTE()

- A sample is created at a randomly selected point between a minority sample and its neighbor which is randomly selected among *k* neighbors

```
from imblearn.over_sampling import SMOTE

X_samp, y_samp = SMOTE(k_neighbors=3).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(900, 2) (900,)
```



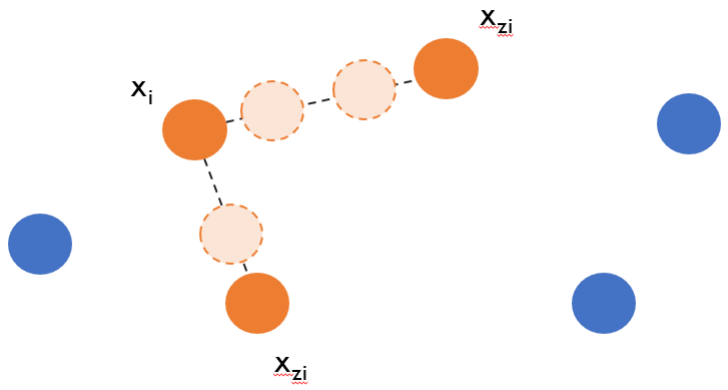


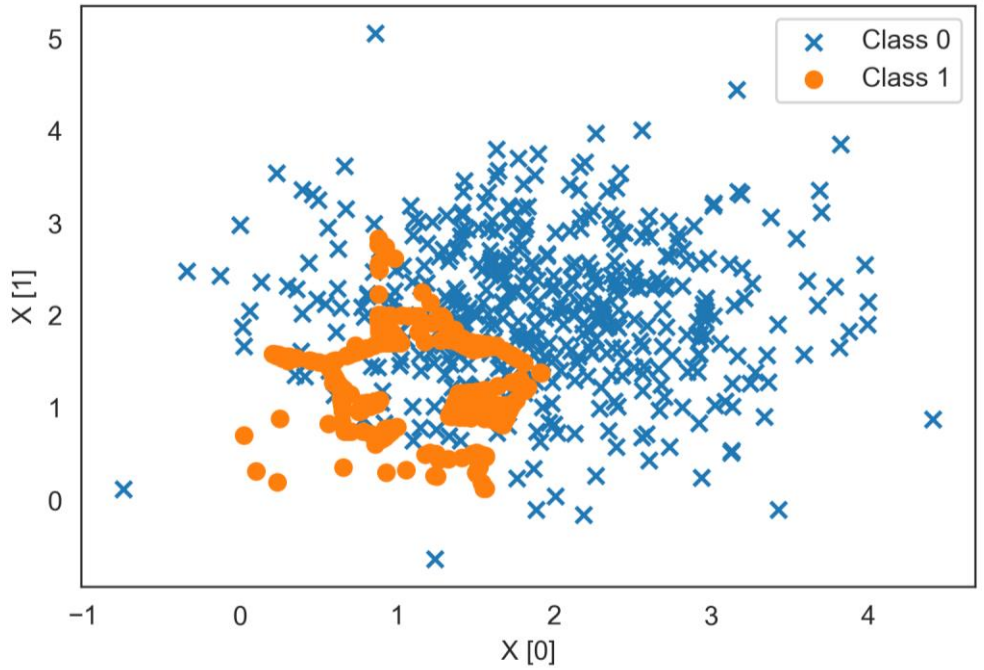
Oversampling: ADASYN()

- For a minority sample dominated by majority class samples, more synthetic minority class samples are generated

```
from imblearn.over_sampling import ADASYN

X_samp, y_samp = ADASYN(n_neighbors=3, random_state=0).fit_sample(X, y)
print(X_samp.shape, y_samp.shape)
plot(X_samp, y_samp)
```

```
(908, 2) (908,)
```





Thank You!