Data Wrangling

부산대학교 정보·의생명공학대학

정보컴퓨터공학부

Data wrangling (or munging)

- Good data scientists spend most of their time (80%) cleaning and formatting data.

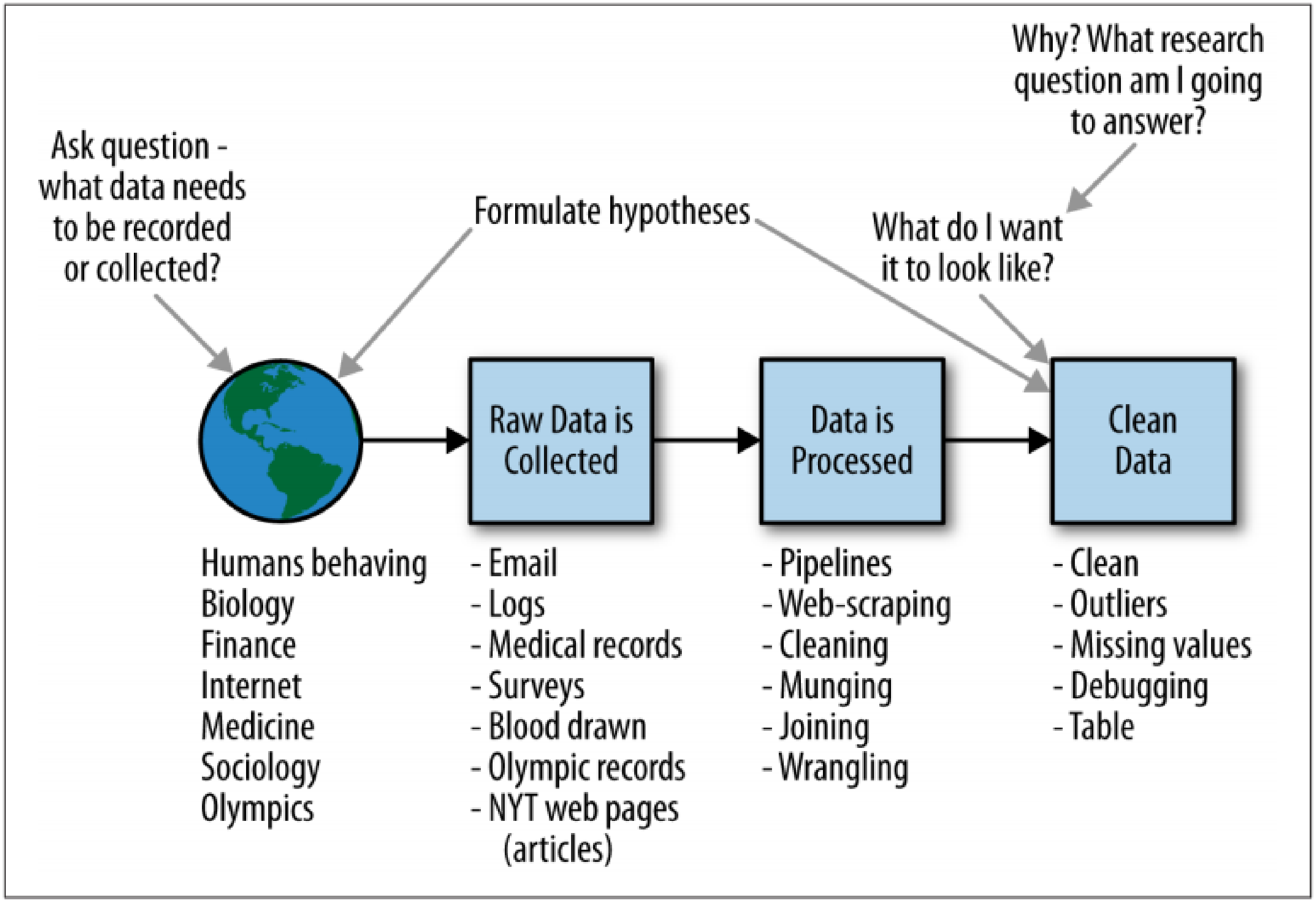
  The rest spend most of their time complaining there is no data available.

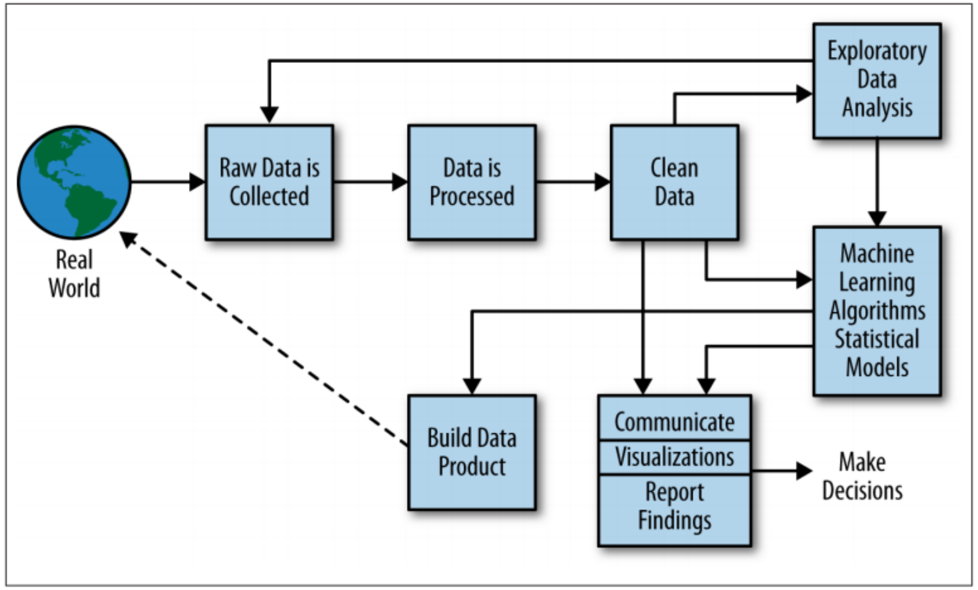
  Data munging or *data wrangling* is the art of acquiring data and preparing it for analysis.

  -- Steven Skiena

- Data wrangling (or data munging) is the process of **transforming** and mapping data from one "raw" data form into another format with the intent of making it more appropriate for further analysis.
- Involves cleaning and unifying messy and complex data sets for easy access and analysis
- Data Cleaning: Removal of faulty values or empty cells. A part of data-preprocessing
- Typical data wrangling techniques involve combining different datasets, reshaping.

When do we perform the data wrangling?





DATA CLEANING

Data Cleaning

- Deals with missing and incorrect data
  - NaN: Not a number
  - negative values in height measure
- Possible actions
  - discarding the sample (row)
  - discarding the variable (column)
  - replacing missing or incorrect values (data imputation)
    - e.g., with proper values using mean, median, mode, or knn
  - leaving them as they are

Mean (Download Speed) = 130

Mobile ID	Mobile Package	Download Speed	Data Limit Usage
1	Fast+	157	80%
2	Lite	99	70%
3	Fast+	167	10%
4	Fast+	N/A	80%
5	Lite	76	70%
6	Fast+	155	10%
7	Fast+	N/A	95%
8	Lite	76	77%
9	Fast+	180	95%

Mobile ID	Mobile Package	Download Speed	Data Limit Usage
1	Fast+	157	80%
2	Lite	99	70%
3	Fast+	167	10%
4	Fast+	130	80%
5	Lite	76	70%
6	Fast+	155	10%
7	Fast+	130	95%
8	Lite	76	77%
9	Fast+	180	95%

Delete

Mobile ID	Mobile Package	Download Speed	Data Limit Usage
1	Fast+	N/A	80%
2	Lite	N/A	70%
3	Fast+	167	10%
4	Fast+	N/A	80%
5	Lite	76	70%
6	Fast+	N/A	10%
7	Fast+	N/A	95%
8	Lite	76	77%
9	Fast+	180	77%

Mobile ID	Mobile Package	Data Limit Usage
1	Fast+	80%
2	Lite	70%
3	Fast+	10%
4	Fast+	80%
5	Lite	70%
6	Fast+	10%
7	Fast+	95%
8	Lite	77%
9	Fast+	77%

Mobile ID	Mobile Package	Download Speed	Data Limit Usage	
1	Fast+	157	80%	
2	Lite	99	70%	
3	Fast+	167	10%	
4	Fast+	N/A	80%	Delete
5	Lite	76	70%	
6	Fast+	155	10%	
7	N/A	N/A	95%	Delete
8	Lite	76	77%	
9	Fast+	180	N/A	Delete

Mobile ID	Mobile Package	Download Speed	Data Limit Usage
1	Fast+	157	80%
2	Lite	99	70%
3	Fast+	167	10%
5	Lite	76	70%
6	Fast+	155	10%
8	Lite	76	77%

Data Cleaning Example

❖ Checking missing values: isnull()

```
[1]  import numpy as np
     import pandas as pd

     # sample data
     np.random.seed(48)
     df = pd.DataFrame(10*np.random.rand(4,2).round(2))
     df.columns = ["A","B"]
     print(df)
```

```
     A    B
0  0.2  8.9
1  2.8  3.0
2  7.9  3.2
3  8.6  4.5
```

```
[2]  # adding nan
     df.A[1] = np.nan
     print(df)
```

```
     A    B
0  0.2  8.9
1  NaN  3.0
2  7.9  3.2
3  8.6  4.5
```

```
[3]  # checking missing values
     df.isnull()
```

	A	B
0	False	False
1	True	False
2	False	False
3	False	False

```
[4]  df.isnull().sum()
```

```
A    1
B    0
dtype: int64
```

Data Cleaning Example

❖ Discarding samples or variables: dropna()

```
print(df)

     A    B
0  0.2  8.9
1  NaN  3.0
2  7.9  3.2
3  8.6  4.5
```

```
[5]  df2 = df.dropna()
     print(df2)

     A    B
0  0.2  8.9
2  7.9  3.2
3  8.6  4.5
```

```
[6]  print(df.dropna(axis=1))

     B
0  8.9
1  3.0
2  3.2
3  4.5
```

```
[7]  print(df.dropna(how="all"))

     A    B
0  0.2  8.9
1  NaN  3.0
2  7.9  3.2
3  8.6  4.5
```

```
[9]  df.B[1] = np.nan
     print(df)

     A    B
0  0.2  8.9
1  NaN  NaN
2  7.9  3.2
3  8.6  4.5
```

```
[10]  print(df.dropna(how="all"))

     A    B
0  0.2  8.9
2  7.9  3.2
3  8.6  4.5
```

Data Cleaning Example

❖ Discarding samples or variables: dropna()

```
[16] df = pd.DataFrame([[np.nan, np.nan, np.nan],
                         [1,np.nan,np.nan],
                         [2,3,np.nan],
                         [4,5,6],[np.nan,7,8]])
     print(df)
```

```
     0    1    2
0  NaN  NaN  NaN
1  1.0  NaN  NaN
2  2.0  3.0  NaN
3  4.0  5.0  6.0
4  NaN  7.0  8.0
```

```
[17] print(df.dropna(thresh=3, axis=1))
```

```
     0    1
0  NaN  NaN
1  1.0  NaN
2  2.0  3.0
3  4.0  5.0
4  NaN  7.0
```

Data Cleaning Example

- Replacing missing values or data imputation
  - Mean, Median, Mode
  - Last Observation Carried Forward (LOCF)
  - Next Observation Carried Backward (NOCB)
  - Linear Interpolation
  - k-NN
  - Arbitrary Value Imputation

Mobile ID	Date	Download Speed	Data Limit Usage
1	1-Jan	157	80%
2	2-Jan	99	81%
3	3-Jan	167	83%
4	4-Jan	90	84%
5	5-Jan	N/A	86%
6	6-Jan	155	87%
7	7-Jan	N/A	89%
8	8-Jan	N/A	90%
9	9-Jan	180	92%

Mobile ID	Date	Download Speed	Data Limit Usage
1	1-Jan	157	80%
2	2-Jan	99	81%
3	3-Jan	167	83%
4	4-Jan	90	84%
5	5-Jan	90	86%
6	6-Jan	155	87%
7	7-Jan	155	89%
8	8-Jan	155	90%
9	9-Jan	180	92%

Data Cleaning Example

❖ Replacing missing values: fillna(value)

```
[12]  import numpy as np
      import pandas as pd

      # sample data
      np.random.seed(11)
      df = pd.DataFrame(10*np.random.rand(10,3).round(2))
      df.columns = ["A","B","C"]
      df.head()

      # insert nan
      idx = np.random.rand(10,3)<0.1
      df[idx]= np.nan
      df.head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	NaN	1.1
4	NaN	8.6	NaN

```
[13]  df.fillna(0).head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	0.0	1.1
4	0.0	8.6	0.0

```
[14]  df.fillna(-1).head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	-1.0	1.1
4	-1.0	8.6	-1.0

Data Cleaning Example

- Replacing missing values
  - Last Observation Carried Forward (LOCF)
  - Next Observation Carried Backward (NOCB)

```
[15] df.head(6)
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	NaN	1.1
4	NaN	8.6	NaN
5	6.3	0.2	1.2

```
[16] df.fillna(method="ffill").head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	4.9	1.1
4	8.5	8.6	1.1

```
[17] df.fillna(method="bfill").head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	8.6	1.1
4	6.3	8.6	1.2

```
[18] df.fillna(method="bfill", axis = 1).head()
```

	A	B	C
0	1.8	0.2	4.6
1	7.2	4.2	4.9
2	0.1	4.9	9.4
3	8.5	1.1	1.1
4	8.6	8.6	NaN

Data Cleaning Example

❖ Replacing missing values

- mean, median, mode

```
[32] print(df)
```

```
     A    B    C
0  1.8  0.2  4.6
1  7.2  4.2  4.9
2  0.1  4.9  9.4
3  8.5  NaN  1.1
4  NaN  8.6  NaN
5  6.3  0.2  1.2
6  3.2  1.6  7.6
7  8.2  3.4  3.2
8  1.1  NaN  NaN
9  6.0  0.6  4.8
```

```
[33] print(df.mean(axis=0).round(2))
     print(df.mean(axis=1).round(2))
```

```
A    4.71
B    2.96
C    4.60
dtype: float64
0    2.20
1    5.43
2    4.80
3    4.80
4    8.60
5    2.57
6    4.13
7    4.93
8    1.10
9    3.80
dtype: float64
```

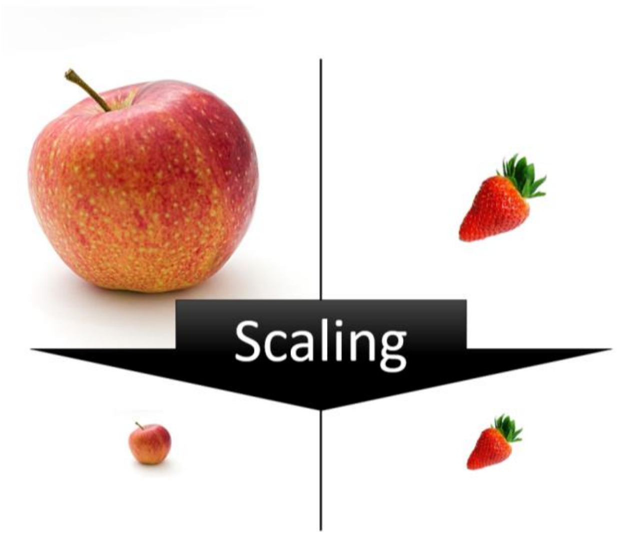
```
[44] df.fillna(df.mean().round(2)).head()
     # df.fillna(df.mean(axis=0).round(2), axis=0).head()
```

	A	B	C
0	1.80	0.20	4.6
1	7.20	4.20	4.9
2	0.10	4.90	9.4
3	8.50	2.96	1.1
4	4.71	8.60	4.6

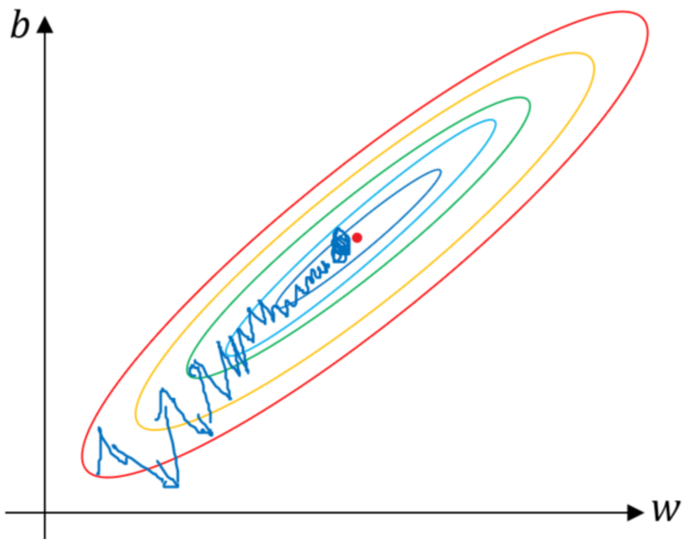
DATA SCALING

Data Scaling

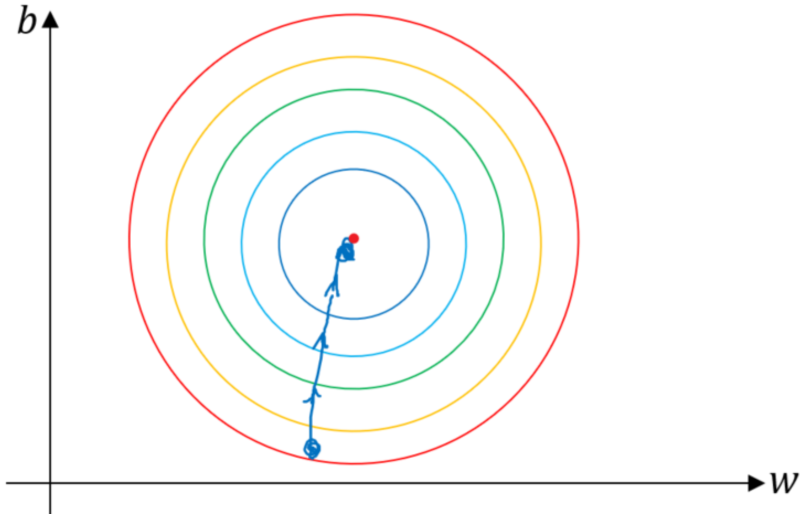
- Also known as feature scaling
- Making sure features are on a similar scale
  - Each feature contributes approximately proportionately to the final distance



Unnormalized



Normalized



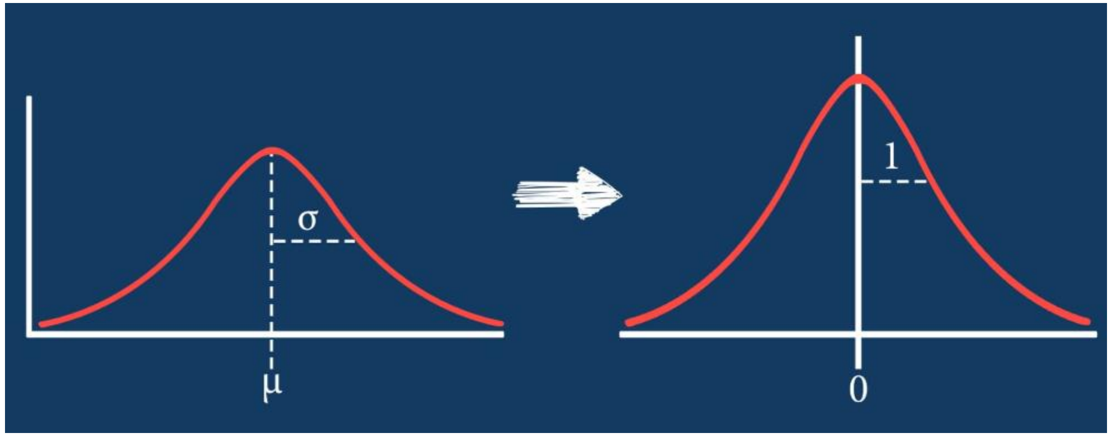
Data Scaling

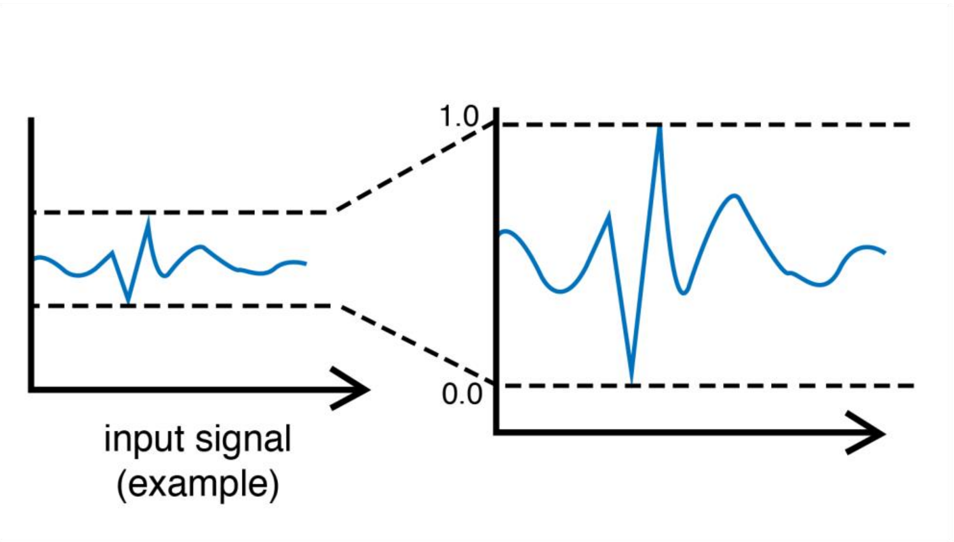
❖ Standardization

- $x_{new} = \dfrac{x - \mu}{\sigma}$

❖ Normalization, min-max scaling

- $x_{new} = \dfrac{x - \min(x)}{\mathrm{man}(x) - \min(x)}$





Data Scaling Example

❖ Standardization

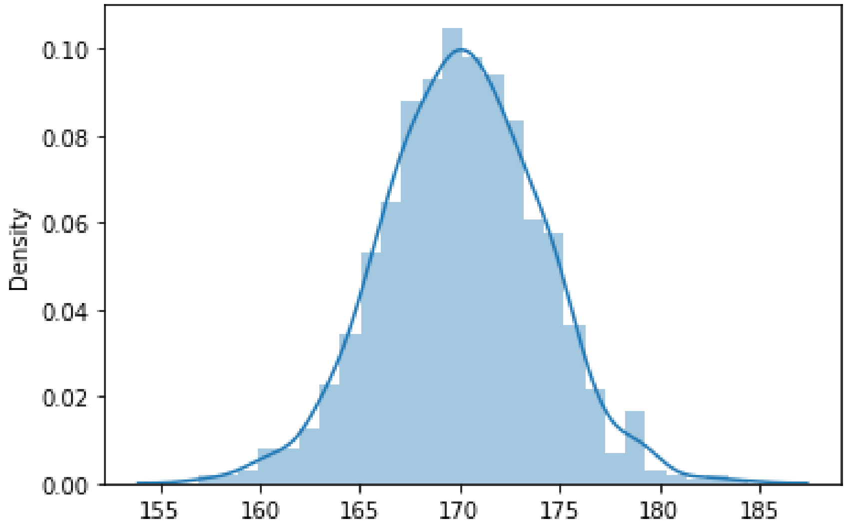
```
[22] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[23] np.random.seed(5)
     n_samples = 1000
     height = 4*np.random.randn(n_samples).round(2) + 170
     weight = 5*np.random.randn(n_samples).round(2) + 65
     df_raw = pd.DataFrame({"height": height, "weight": weight})
     df_raw[:5]
```

	height	weight
0	171.76	67.7
1	168.68	67.0
2	179.72	68.6
3	169.00	64.9
4	170.44	65.1

```
[24] # copying data
     df = df_raw.copy()
     sns.distplot(df.height.values)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributio
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f04bde34990>
```



Data Scaling Example

❖ Standardization

```
[25] from sklearn.preprocessing import StandardScaler
     scale = StandardScaler()
     df[['h_sc','w_sc']] = scale.fit_transform(df[['height','weight']])
     df[:5]
```

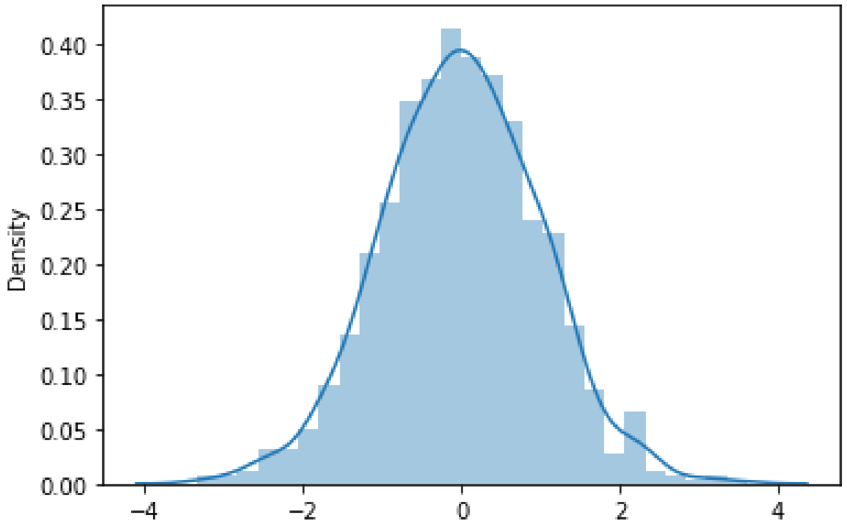
	height	weight	h_sc	w_sc
0	171.76	67.7	0.411016	0.556316
1	168.68	67.0	-0.366907	0.415903
2	179.72	68.6	2.421491	0.736847
3	169.00	64.9	-0.286084	-0.005336
4	170.44	65.1	0.077620	0.034782

```
[27] # inverse transform
     scale.inverse_transform(df[['h_sc','w_sc']])[:5]
```

```
array([[171.76,  67.7 ],
       [168.68,  67.  ],
       [179.72,  68.6 ],
       [169.  ,  64.9 ],
       [170.44,  65.1 ]])
```

```
[26] sns.distplot(df.h_sc.values)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.p
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba8007d0>
```

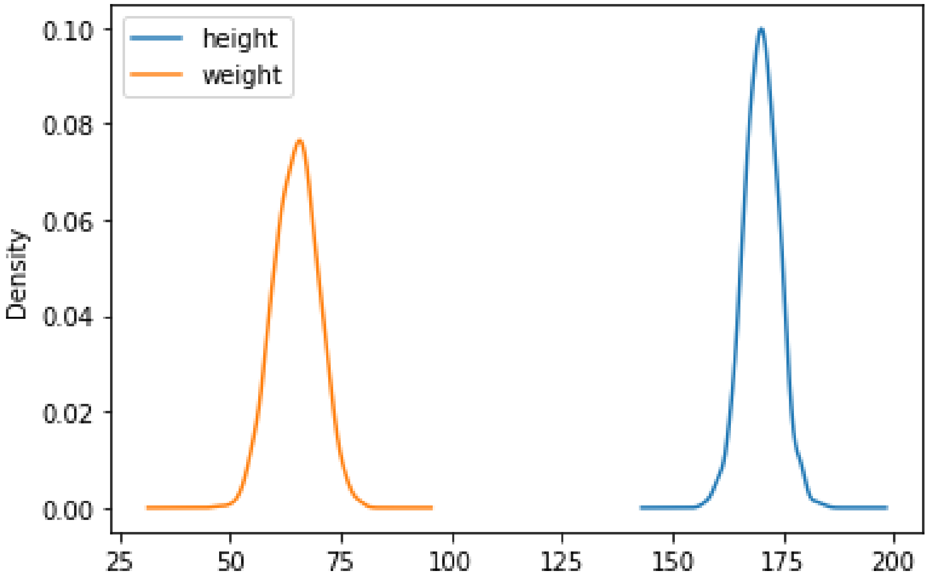


Data Scaling Example

❖ Standardization

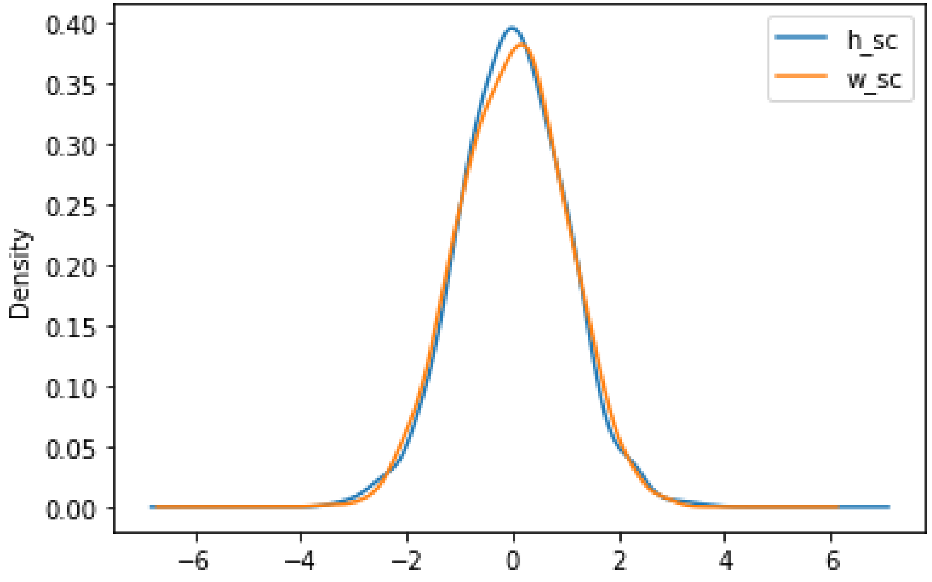
```
[28] df[['height','weight']].plot.kde()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba44f




```
[29] df[['h_sc','w_sc']].plot.kde()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba3e5b



Data Scaling Example

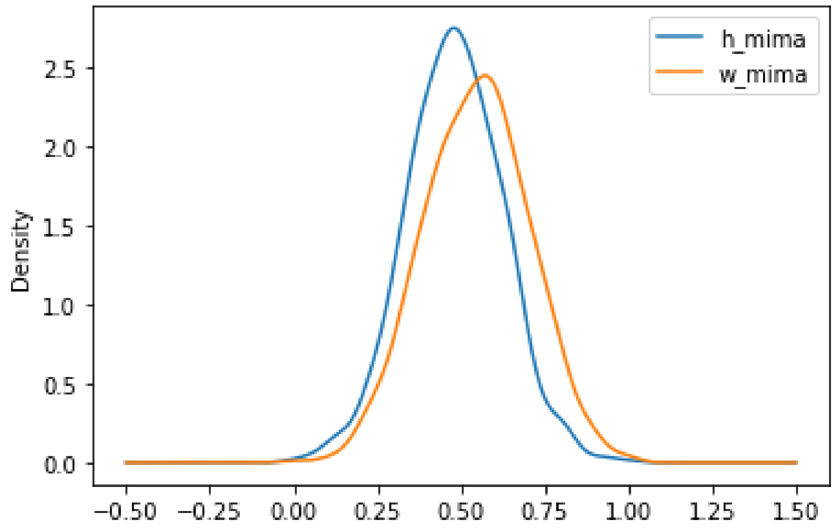
❖ Min-max scaling (normalization)

```
[30] from sklearn.preprocessing import MinMaxScaler
     minmax = MinMaxScaler()
     df[['h_mima','w_mima']]=minmax.fit_transform(df[['height','weight']])
     df[:5]
```

	height	weight	h_sc	w_sc	h_mima	w_mima
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937
3	169.00	64.9	-0.286084	-0.005336	0.439768	0.545313
4	170.44	65.1	0.077620	0.034782	0.492017	0.551562

```
[31] df[["h_mima","w_mima"]].plot.kde()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba360710>
```



Data Scaling Example

❖ Effects of outlier

```
[32] height_1 = height.copy()
     height_1[0] = 200
     df["height_1"] = height_1
     df[:3]
```

	height	weight	h_sc	w_sc	h_mima	w_mima	height_1
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72

```
[33] df['h_1_sc']=scale.fit_transform(df[['height_1']])
     df['h_1_mima']=minmax.fit_transform(df[['height_1']])
     df[:3]
```

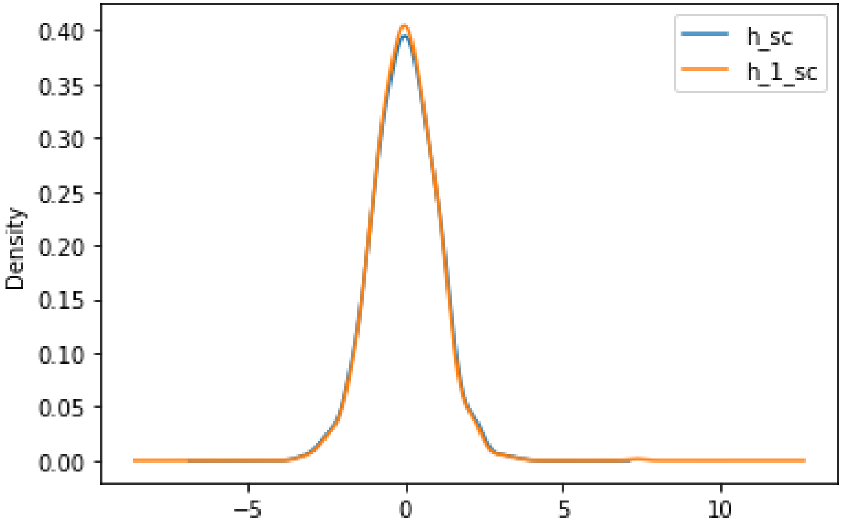
	height	weight	h_sc	w_sc	h_mima	w_mima	height_1	h_1_sc	h_1_mima
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00	7.331585	1.000000
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68	-0.363868	0.273655
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72	2.348705	0.529685

Data Scaling Example

❖ Effects of outlier

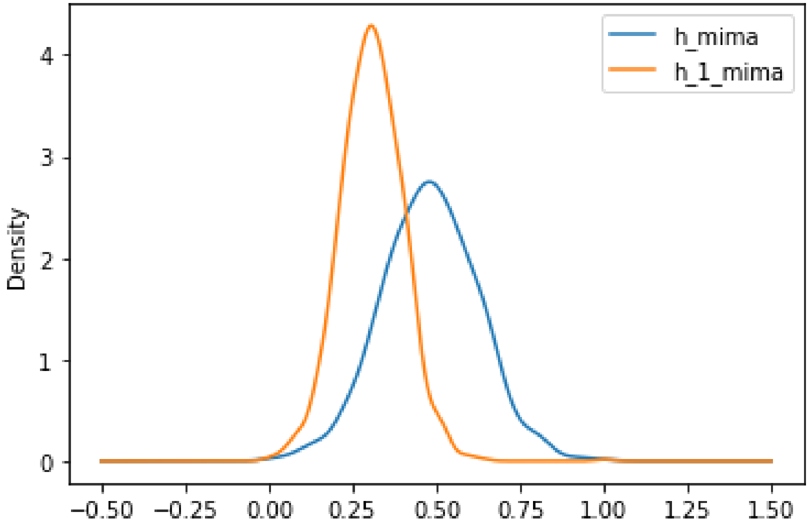
```
[34] df[['h_sc','h_1_sc']].plot.kde()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba2eb990>




```
[35] df[['h_mima','h_1_mima']].plot.kde()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f04ba271510>

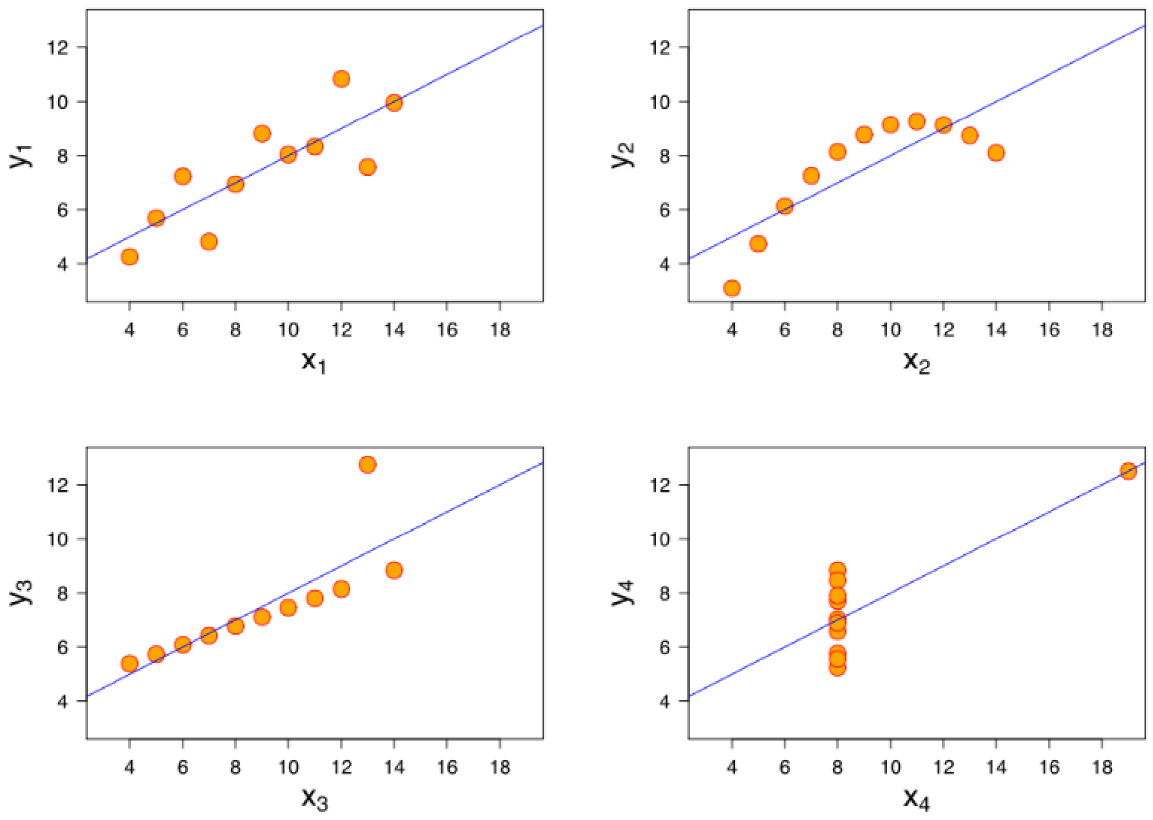


Dealing with Outliers

❖ An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense this definition leaves it up to the analyst to decide what will be considered abnormal.

❖ Two purposes w.r.t. outlier analysis

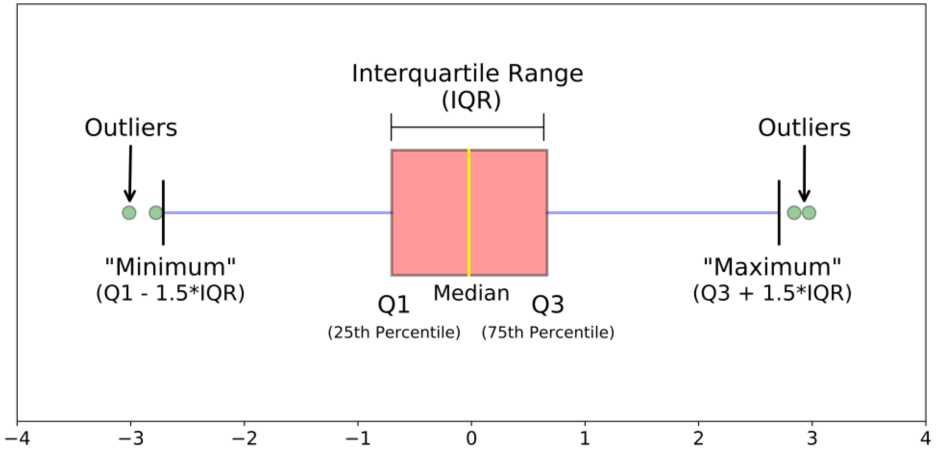
- improve the quality of the dataset and the accuracy of the model/analysis
- Outlier detection

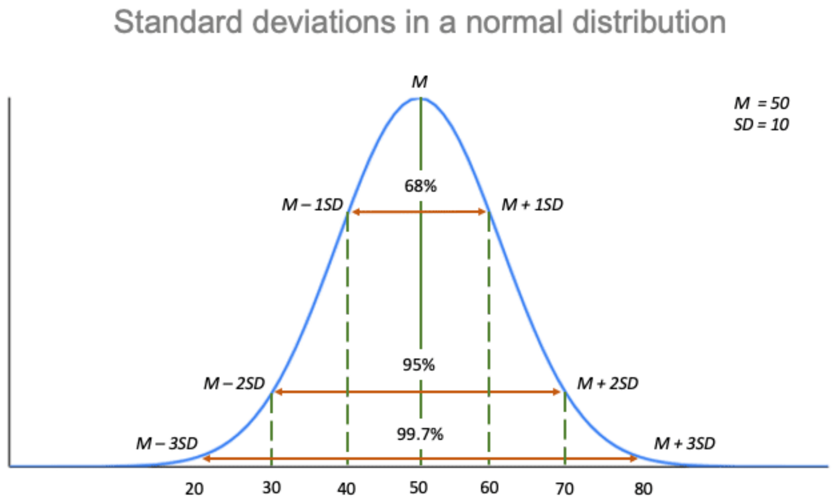


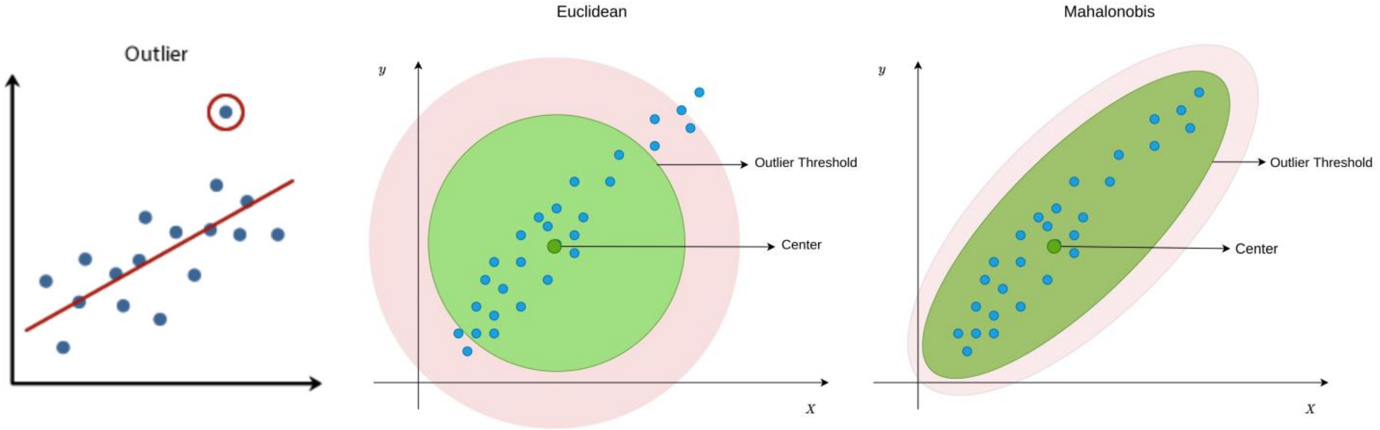
How to find outliers

❖ EDA

- Boxplot: < 1.5 IRQ
- Standardization: < 3 SD
- Scatter plot (multivariate)



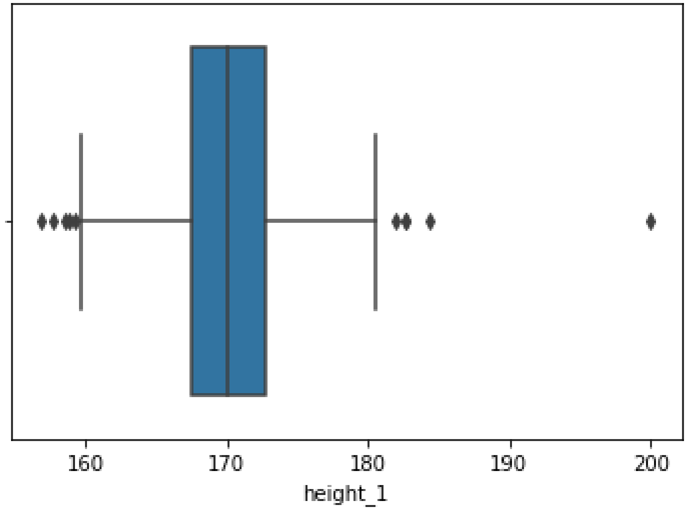




$$D^2 = (x-m)^T \cdot C^{-1} \cdot (x-m)$$

Outlier Detection and Removal Example

```
[36] f = sns.boxplot(x = df.height_1)
```




```
[37] df.shape
```

```
(1000, 9)
```

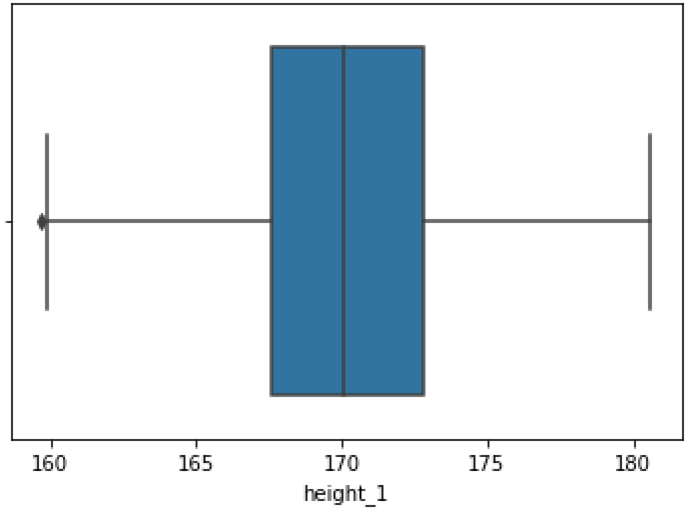
```
[38] Q1 = df.height_1.quantile(0.25)
     Q3 = df.height_1.quantile(0.75)
     IRQ = Q3 - Q1
     lower = Q1 - 1.5*IRQ
     upper = Q3 + 1.5*IRQ
     print(lower, upper)
```

```
159.58499999999998 180.74500000000006
```

```
[39] # filter using query
     filtered = df.query('@lower <= height_1 <= @upper')
     filtered.shape
```

```
(990, 9)
```

```
[40] f = sns.boxplot(x = filtered.height_1)
```

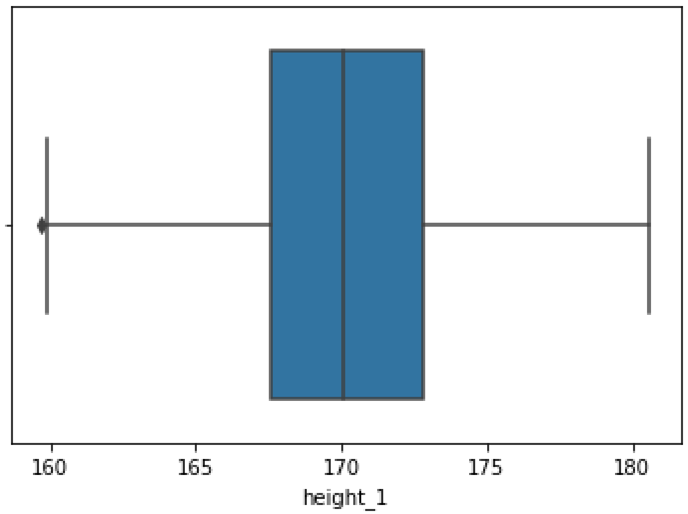


Outlier Detection and Removal Example

```
[41] # filter using loc
     df2 = df.copy()
     df2.loc[df2.height_1 > upper] = np.nan
     df2.loc[df2.height_1 < lower] = np.nan
     df2 = df2.dropna()
     df2.shape
```

(990, 9)

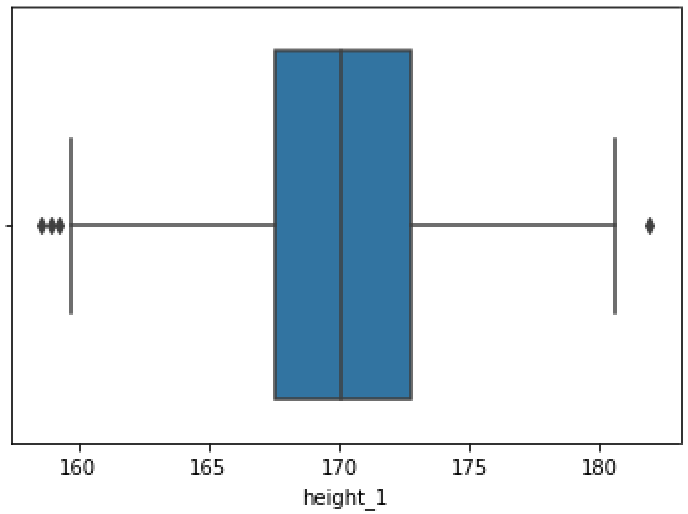
```
[42] f = sns.boxplot(x = df2.height_1)
```




```
[43] # filter using SD
     df3 = df.copy()
     df3 = df3.query('-3 <= h_1_sc <= 3')
     df3.shape
```

(994, 9)

```
[44] f = sns.boxplot(x = df3.height_1)
```



Data Encoding

- Dividing a continuous variable (interval, ratio) into categories (nominal)
  - E.g., scores to grades
- Transforming categorical variables into numerical or a set of binary variables
  - Label encoding
  - One-hot encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Data Encoding Example

```
[45] df.loc[df.h_sc > 1, 'group'] = 'tall'
     df.loc[(df.h_sc <= 1) & df.h_sc >=1, 'group'] = 'normal'
     df.loc[df.h_sc < -1, 'group'] = 'small'
     df[:5]
```

	height	weight	h_sc	w_sc	h_mima	w_mima	height_1	h_1_sc	h_1_mima	group
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00	7.331585	1.000000	normal
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68	-0.363868	0.273655	normal
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72	2.348705	0.529685	tall
3	169.00	64.9	-0.286084	-0.005336	0.439768	0.545313	169.00	-0.285243	0.281076	normal
4	170.44	65.1	0.077620	0.034782	0.492017	0.551562	170.44	0.068571	0.314471	normal

```
[46] df.group.value_counts()
```

```
normal    692
tall      156
small     152
Name: group, dtype: int64
```

Data Encoding Example

```
[47] # label encoding
     from sklearn.preprocessing import LabelEncoder
     encoder = LabelEncoder()
     df['en'] = encoder.fit_transform(df.group)
     encoder.classes_
```

```
array(['normal', 'small', 'tall'], dtype=object)
```

```
[48] df.head()
```

	height	weight	h_sc	w_sc	h_mima	w_mima	height_1	h_1_sc	h_1_mima	group	en
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00	7.331585	1.000000	normal	0
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68	-0.363868	0.273655	normal	0
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72	2.348705	0.529685	tall	2
3	169.00	64.9	-0.286084	-0.005336	0.439768	0.545313	169.00	-0.285243	0.281076	normal	0
4	170.44	65.1	0.077620	0.034782	0.492017	0.551562	170.44	0.068571	0.314471	normal	0

Data Encoding Example

```
[49] # one-hot encoding
     df_group = pd.get_dummies(df.group)
     df_group.head()
```

	normal	small	tall
0	1	0	0
1	1	0	0
2	0	0	1
3	1	0	0
4	1	0	0

```
[50] df_new = pd.concat((df, df_group), axis = 1)
     df_new.head()
```

	height	weight	h_sc	w_sc	h_mima	w_mima	height_1	h_1_sc	h_1_mima	group	en	normal	small	tall
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00	7.331585	1.000000	normal	0	1	0	0
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68	-0.363868	0.273655	normal	0	1	0	0
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72	2.348705	0.529685	tall	2	0	0	1
3	169.00	64.9	-0.286084	-0.005336	0.439768	0.545313	169.00	-0.285243	0.281076	normal	0	1	0	0
4	170.44	65.1	0.077620	0.034782	0.492017	0.551562	170.44	0.068571	0.314471	normal	0	1	0	0

Data Encoding Example

```
[50] df_new = pd.concat((df, df_group), axis = 1)
     df_new.head()
```

	height	weight	h_sc	w_sc	h_mima	w_mima	height_1	h_1_sc	h_1_mima	group	en	normal	small	tall
0	171.76	67.7	0.411016	0.556316	0.539913	0.632812	200.00	7.331585	1.000000	normal	0	1	0	0
1	168.68	67.0	-0.366907	0.415903	0.428157	0.610937	168.68	-0.363868	0.273655	normal	0	1	0	0
2	179.72	68.6	2.421491	0.736847	0.828737	0.660937	179.72	2.348705	0.529685	tall	2	0	0	1
3	169.00	64.9	-0.286084	-0.005336	0.439768	0.545313	169.00	-0.285243	0.281076	normal	0	1	0	0
4	170.44	65.1	0.077620	0.034782	0.492017	0.551562	170.44	0.068571	0.314471	normal	0	1	0	0

```
[51] X = df_new[['h_sc','w_sc','normal','small','tall']]
     X.head()
```

	h_sc	w_sc	normal	small	tall
0	0.411016	0.556316	1	0	0
1	-0.366907	0.415903	1	0	0
2	2.421491	0.736847	0	0	1
3	-0.286084	-0.005336	1	0	0
4	0.077620	0.034782	1	0	0