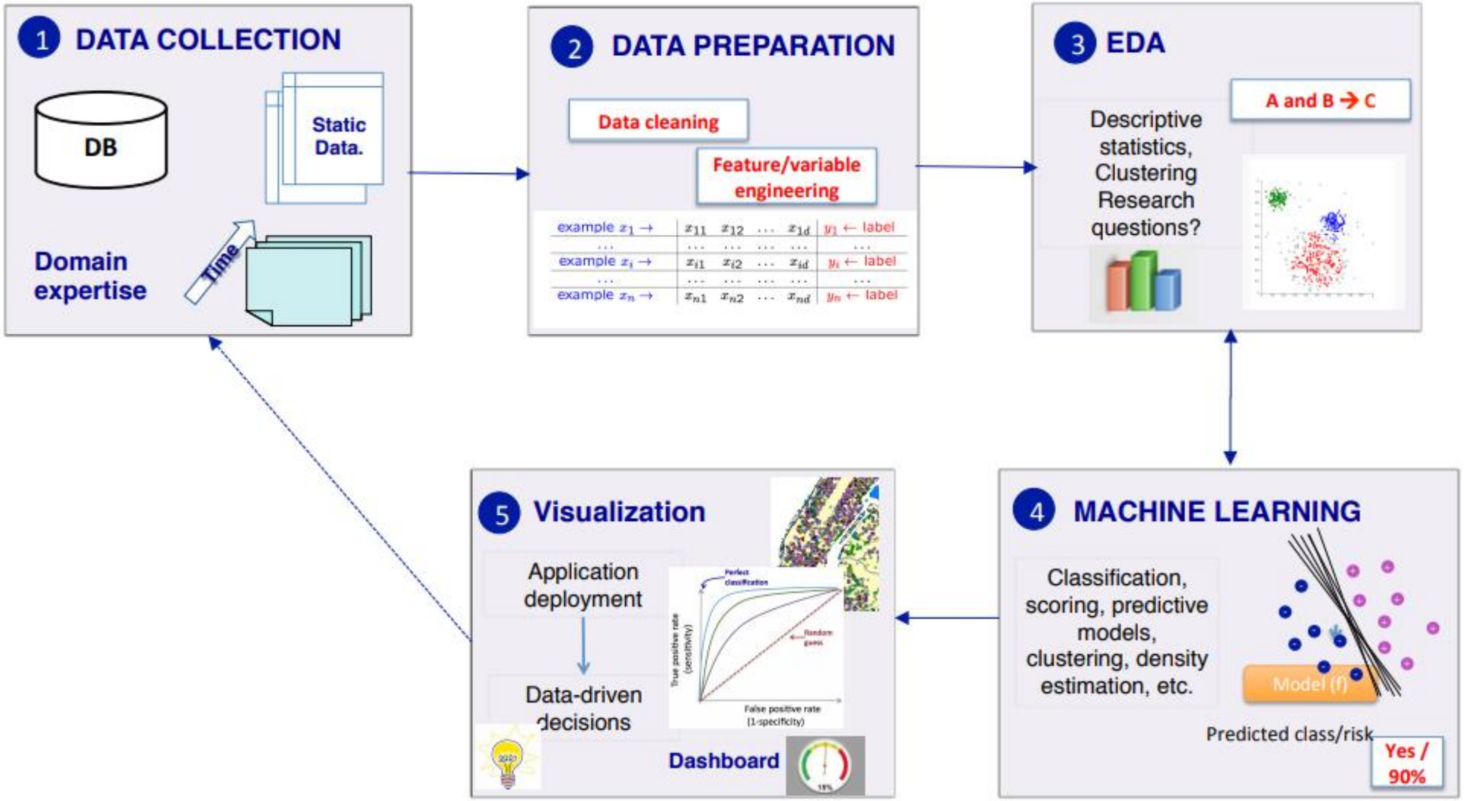
Introduction to Machine Learning. Lec.2 Data preprocessing

Aidos Sarsembayev, IITU, 2018

Outline

- What is data preprocessing?
- Why do we need it?
- How should we do it?

The cycle of ML algorithm’s development



1 DATA COLLECTION

DB

Static Data.

Domain expertise

2 DATA PREPARATION

Data cleaning

Feature/variable engineering

3 EDA

A and B → C

Descriptive statistics, Clustering Research questions?

4 MACHINE LEARNING

Classification, scoring, predictive models, clustering, density estimation, etc.

Model (f)

Predicted class/risk

Yes / 90%

5 Visualization

Application deployment

Data-driven decisions

Dashboard

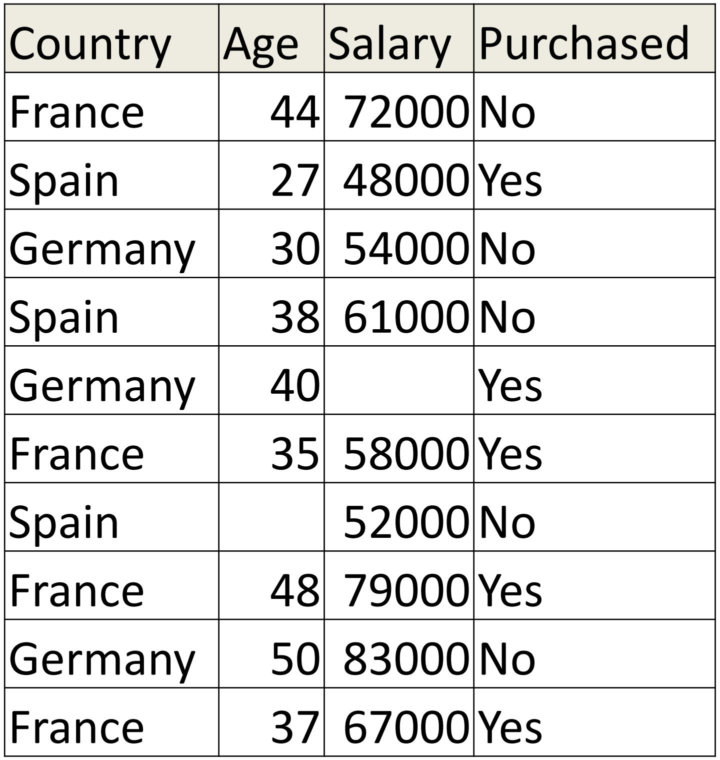
Time
example x_1 → x_{11} x_{12} … x_{1d} y_1 ← label
example x_i → x_{i1} x_{i2} … x_{id} y_i ← label
example x_n → x_{n1} x_{n2} … x_{nd} y_n ← label
Perfect classification
Random guess
True positive rate (sensitivity)
False positive rate (1-specificity)
19%

Data preparation

This stage consists of few steps. These are:

- Tackling the missing data problems
- Encoding the categorical data
- Splitting the dataset into train and test sets
- Scaling the features

Data preparation



Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40		Yes
France	35	58000	Yes
Spain		52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

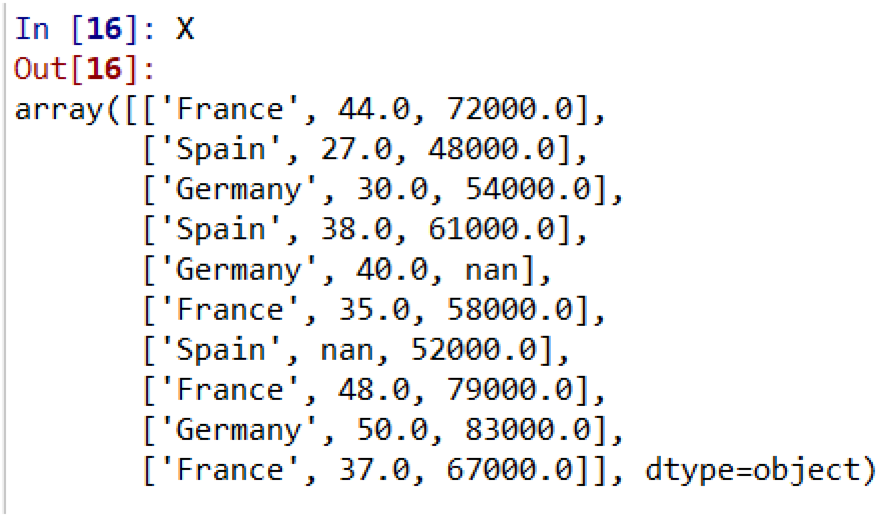
Data preparation

But first of all we make a **matrix of features** by splitting the dependent and independent variables

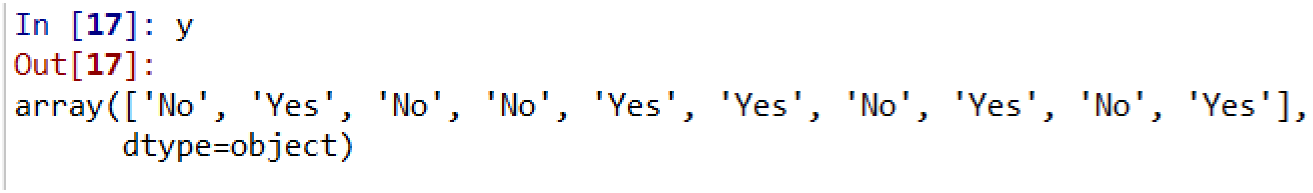
The dependent variable in this case is the **last column**. This will be our output – y

The independent variables in this case are the **all columns before the last one**. This will be our input – X

Data preparation



```
In [16]: X
Out[16]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```



```
In [17]: y
Out[17]:
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Missing data problems

From the dataset table you may notice that there are two cells with missing data. One at 'Salary' and the other at 'Age'

You may handle this by two possible solutions:

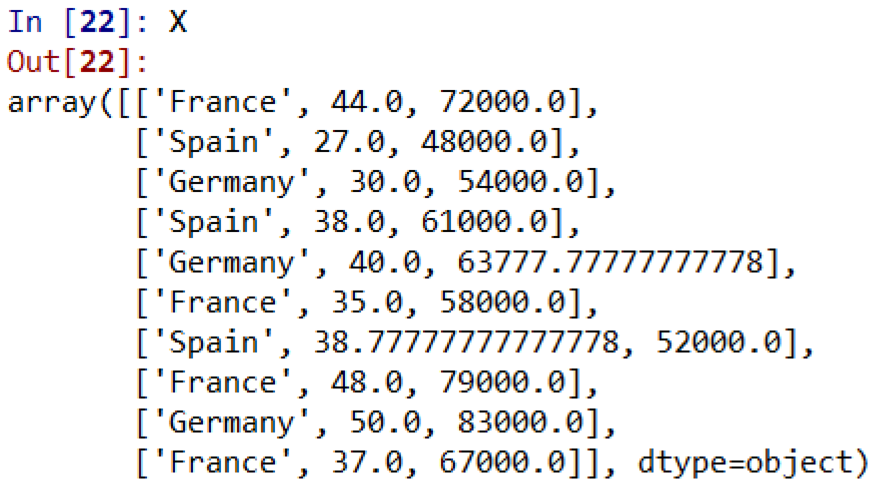
1) by deleting the lines w/ missing data. However, this is a dangerous way since these lines may contain crucial data

2) by putting the mean of the columns values

We will try the second option

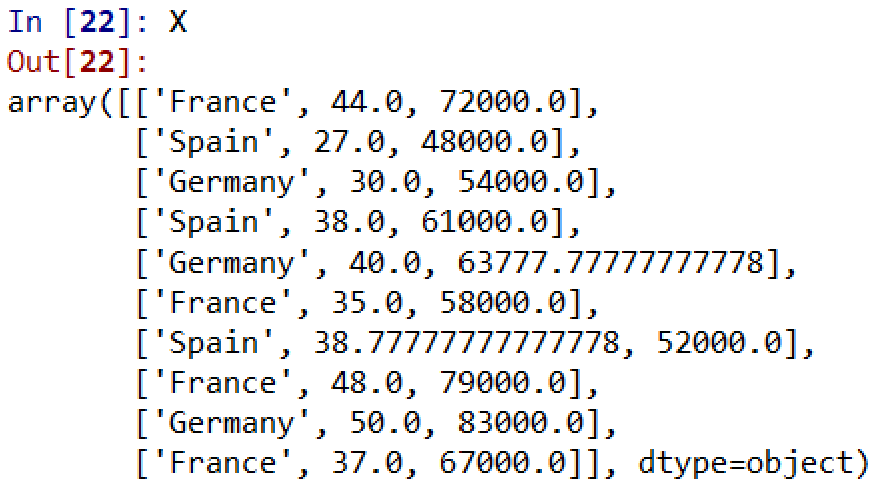
For this purpose we import the Imputer class from the sklearn library

Missing data problems



```
In [22]: X
Out[22]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

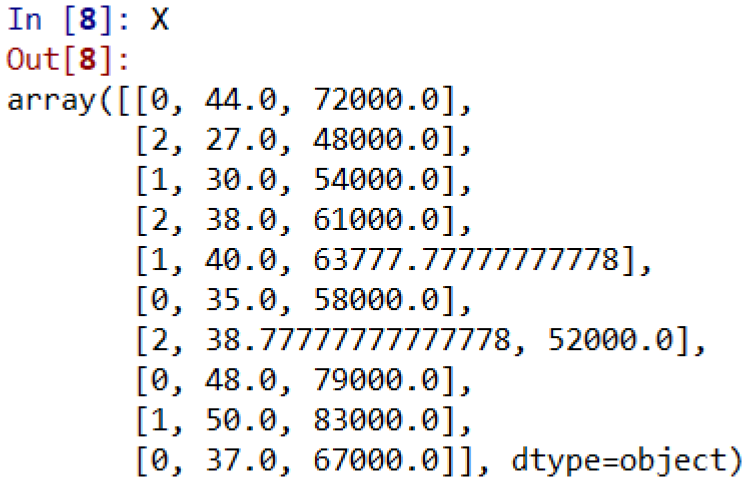
Encoding the categorical data



```
In [22]: X
Out[22]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

Categorical data is usually form of Strings and is impossible to interpret in mathematical formulas. We have to encode it in numerical data. For that we are using **Label encoding**

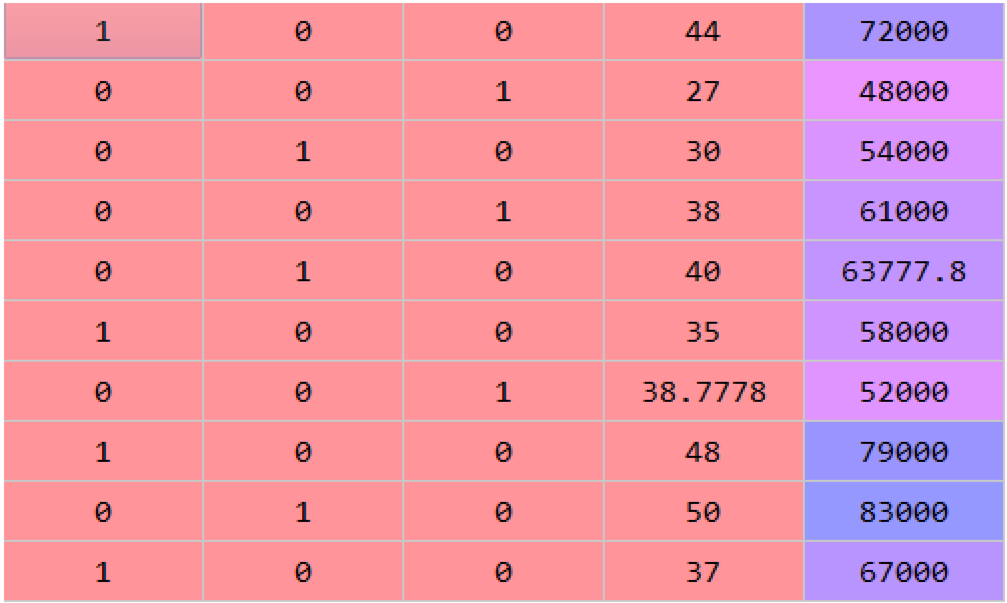
Encoding the categorical data



```
In [8]: X
Out[8]:
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 38.77777777777778, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

Now it has a numerical form. But if you take a look at the values you might see the possible problem which may appear once we process the data with an algorithm.

Encoding the categorical data

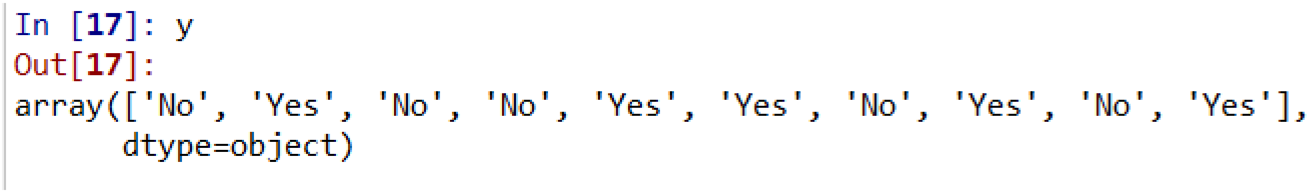


1	0	0	44	72000
0	0	1	27	48000
0	1	0	30	54000
0	0	1	38	61000
0	1	0	40	63777.8
1	0	0	35	58000
0	0	1	38.7778	52000
1	0	0	48	79000
0	1	0	50	83000
1	0	0	37	67000

Therefore, we do **OneHotEncoding**

$N_{cat\_data} = N_{columns}$

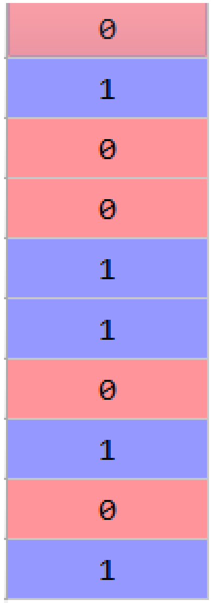
Encoding the categorical data



```
In [17]: y
Out[17]:
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Our output still looks like categorical data. We should apply encoding here as well. Which type of encoding should we use?

Encoding the categorical data



0
1
0
0
1
1
0
1
0
1

Do you think we need to go further by applying the OneHotEncoder?

Splitting the dataset into train and test sets

- Why do we need to split the data into training and test sets?
- Which proportions should we follow?

Splitting the dataset into train and test sets

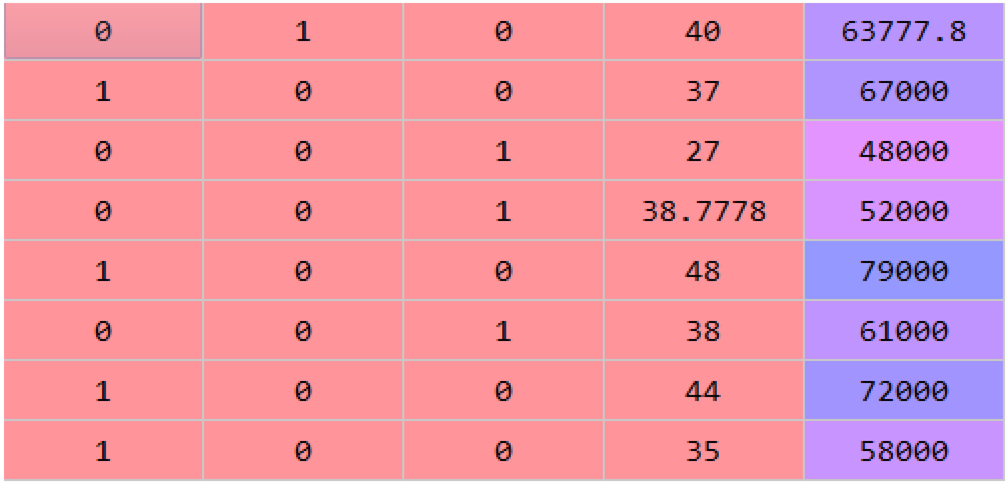
- X_train
- y_train

70, 75, 80 %%

- X_test
- y_test

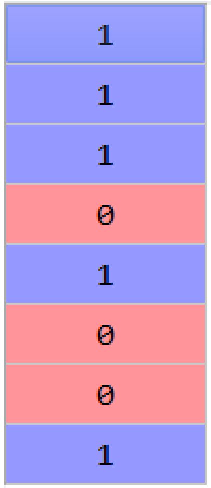
30, 25, 20 %%

Splitting the dataset into train and test sets



0	1	0	40	63777.8
1	0	0	37	67000
0	0	1	27	48000
0	0	1	38.7778	52000
1	0	0	48	79000
0	0	1	38	61000
1	0	0	44	72000
1	0	0	35	58000

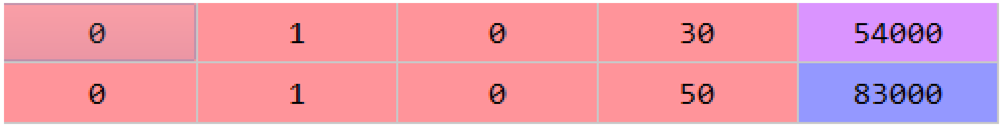
X_train



1
1
1
0
1
0
0
1

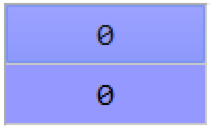
y_train

Splitting the dataset into train and test sets



0	1	0	30	54000
0	1	0	50	83000

X_test

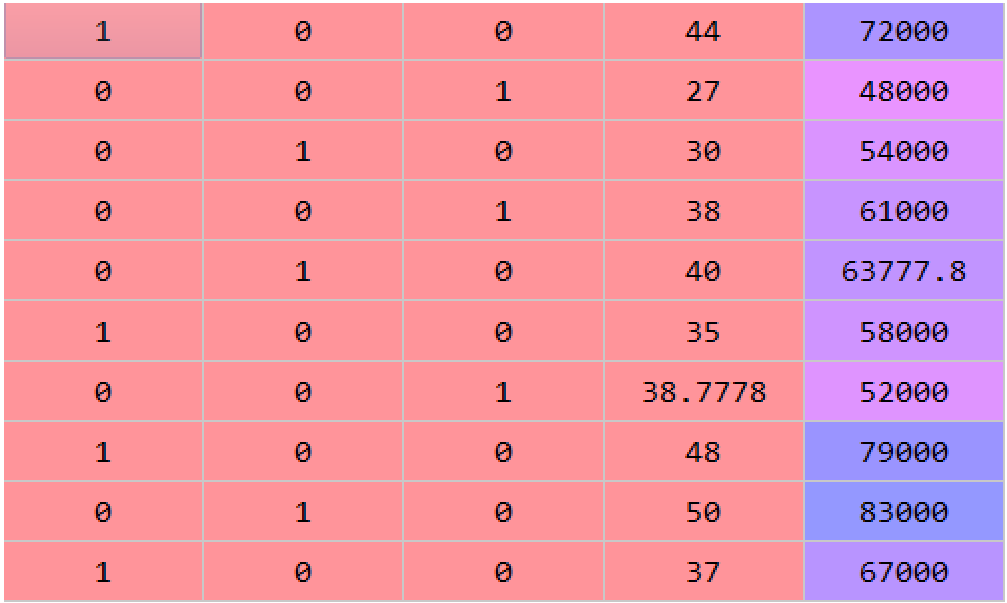


0
0

y_train

Scaling the features

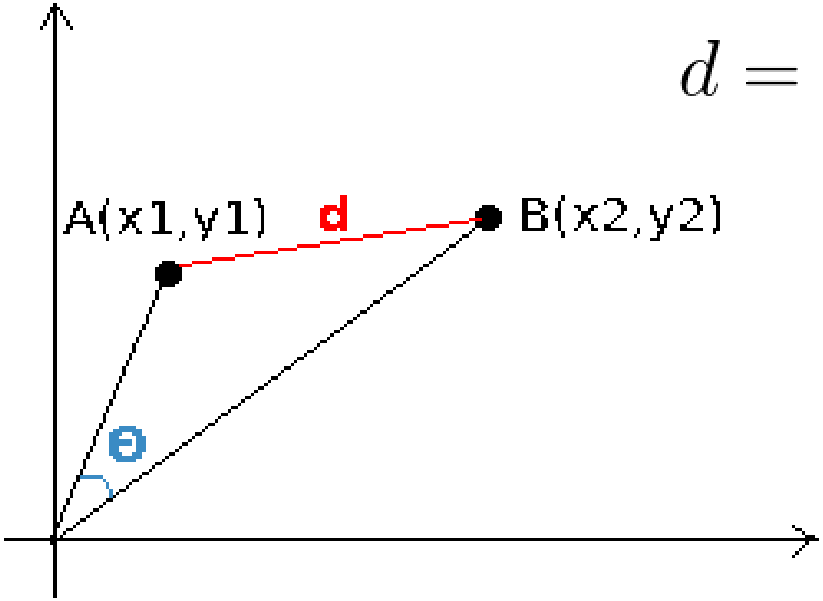
- Why should we scale the features?



1	0	0	44	72000
0	0	1	27	48000
0	1	0	30	54000
0	0	1	38	61000
0	1	0	40	63777.8
1	0	0	35	58000
0	0	1	38.7778	52000
1	0	0	48	79000
0	1	0	50	83000
1	0	0	37	67000

Scaling the features

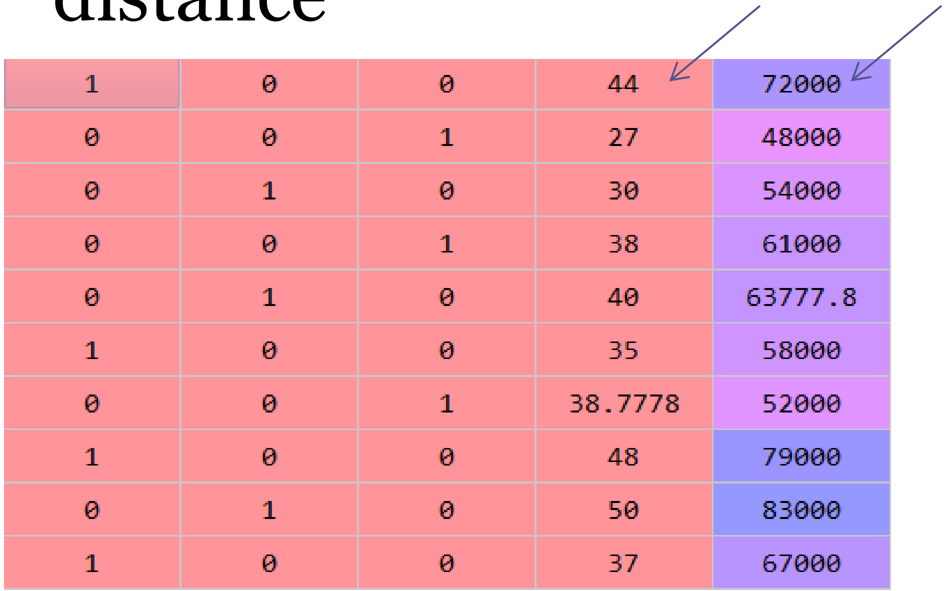
- A lot of ML algorithms are based on distance calculations (i.g. Euclidean distance)



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Scaling the features

- Say we want to find the correlations based on the distance



1	0	0	44	72000
0	0	1	27	48000
0	1	0	30	54000
0	0	1	38	61000
0	1	0	40	63777.8
1	0	0	35	58000
0	0	1	38.7778	52000
1	0	0	48	79000
0	1	0	50	83000
1	0	0	37	67000

x,y	x^2, y^2
72000	5184000000
44	1936

The distance here tends to be very huge

Scaling the features

This is why we apply either/or:

- Normalization
- Standardization

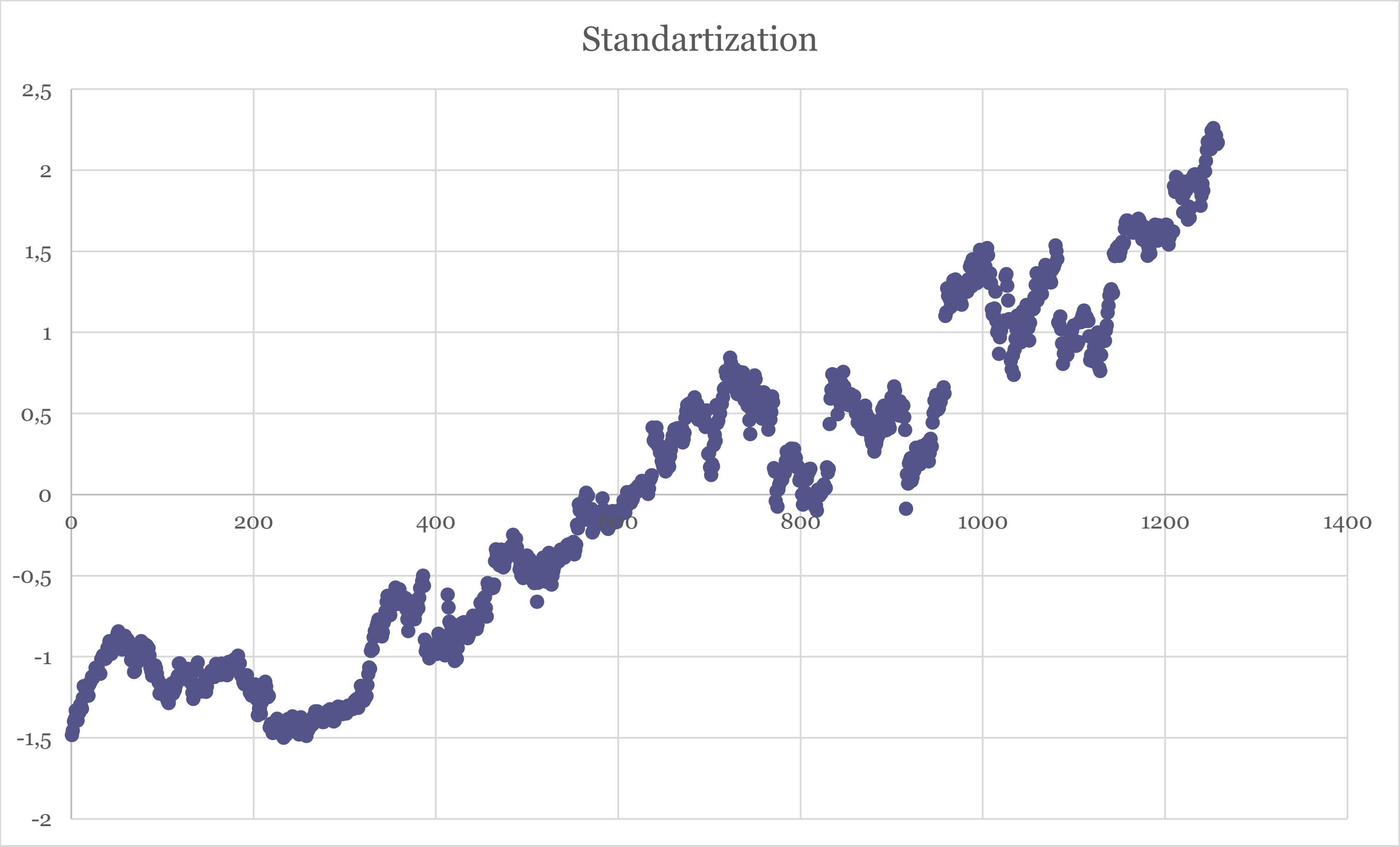
$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

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

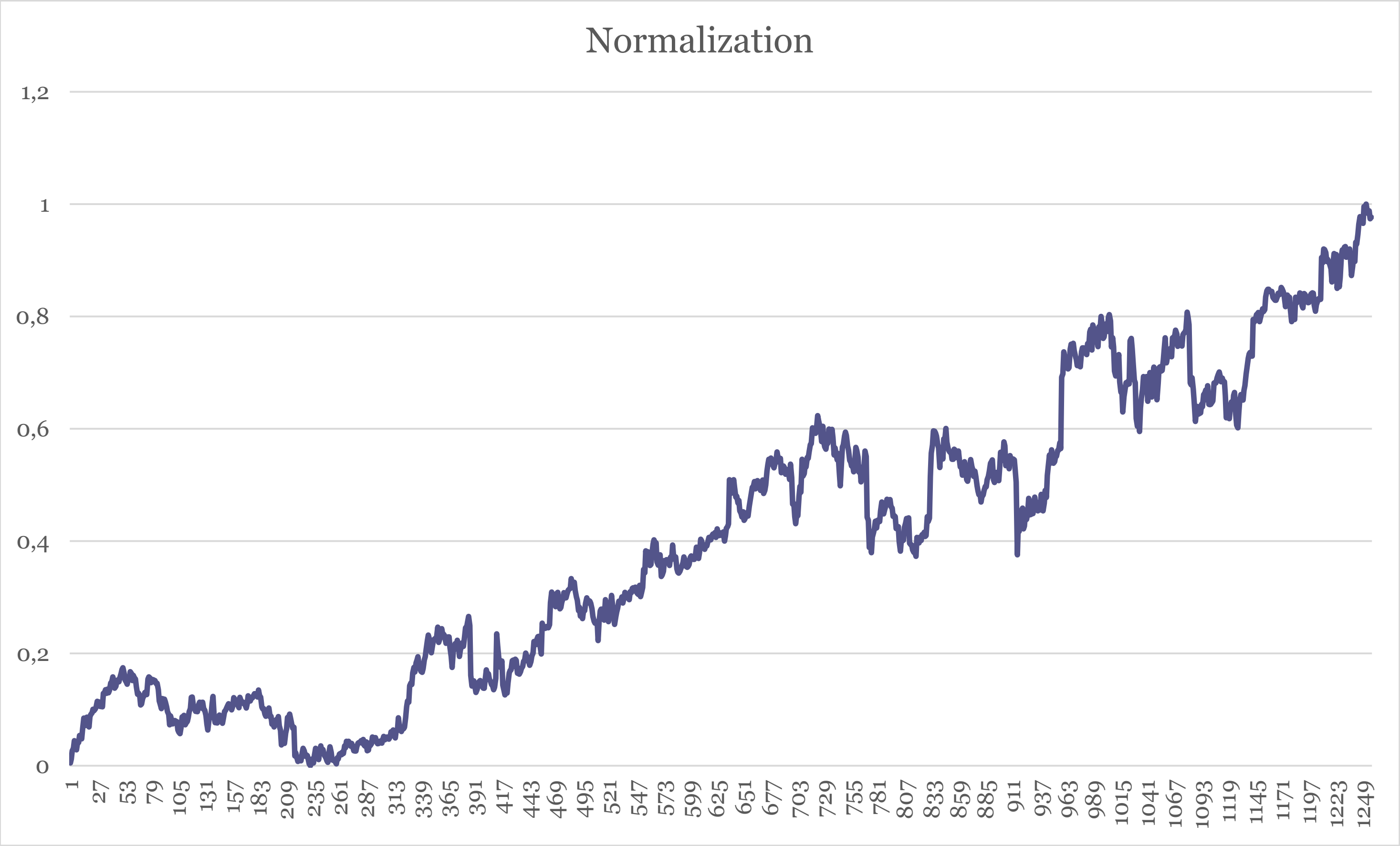
$\mu =$ Mean

$\sigma =$ Standard Deviation

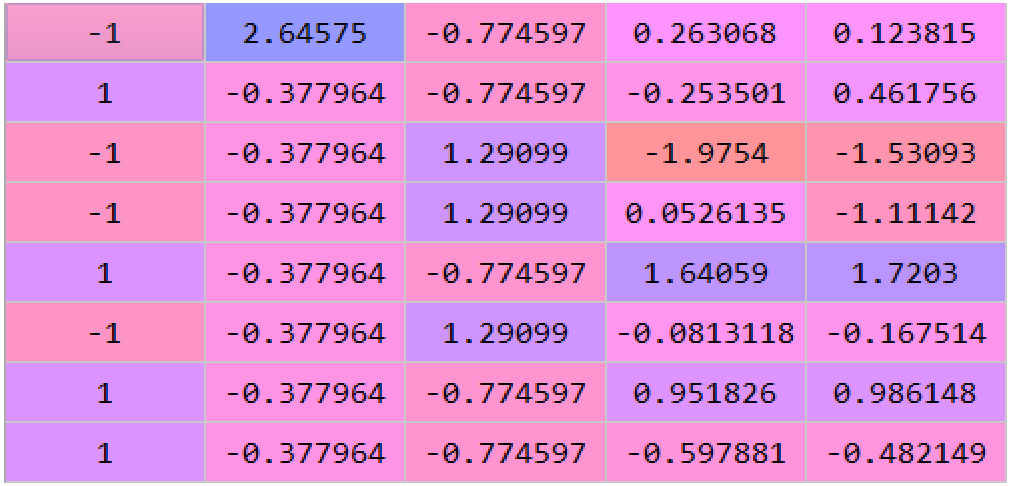
Scaling the features



Scaling the features

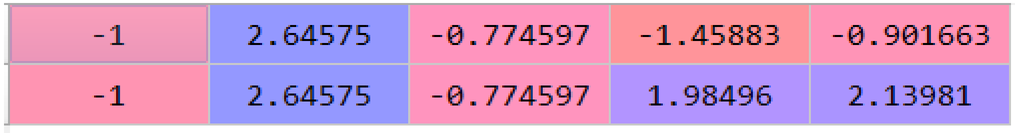


Scaling the features



-1	2.64575	-0.774597	0.263068	0.123815
1	-0.377964	-0.774597	-0.253501	0.461756
-1	-0.377964	1.29099	-1.9754	-1.53093
-1	-0.377964	1.29099	0.0526135	-1.11142
1	-0.377964	-0.774597	1.64059	1.7203
-1	-0.377964	1.29099	-0.0813118	-0.167514
1	-0.377964	-0.774597	0.951826	0.986148
1	-0.377964	-0.774597	-0.597881	-0.482149

X_train



-1	2.64575	-0.774597	-1.45883	-0.901663
-1	2.64575	-0.774597	1.98496	2.13981

X_test

Interesting to play with

- https://www.kaggle.com/uciml/pima-indians-diabetes-database