Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Lecture 06:

Feature analysis, engineering and encoding

Markus Hohle

University California, Berkeley

Data Science for Scientific Computing

MSSE 277A, 3 Units

Feature analysis, engineering and encoding

Outline

What is a feature?

Encodings

Normalization

Visualization

Feature analysis, engineering and encoding

Outline

What is a feature?

Encodings

Normalization

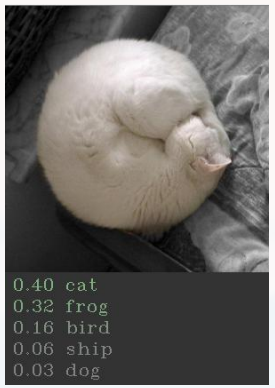
Visualization

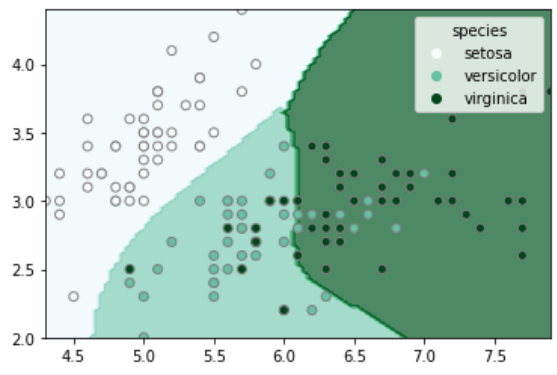
Feature analysis, engineering and encoding:

What is a feature?

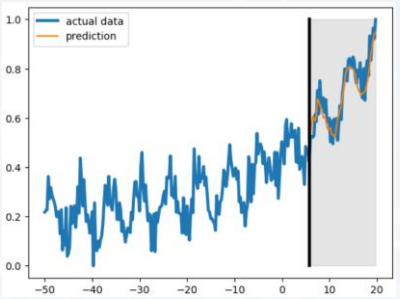
data analysis tasks

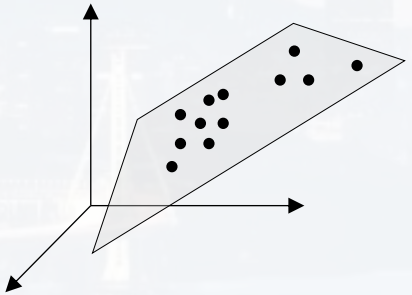
- classification





- regression

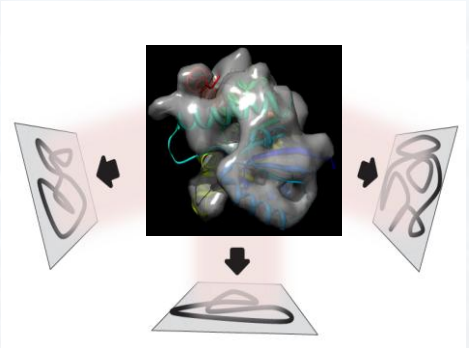




numerical operations

based on a mathematical criterium,
the *objective function*

- generation



Feature analysis, engineering and encoding:

What is a feature?

data analysis tasks

- classification

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

- regression

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	toxicity_score
0	341.704	2.65585	3.09407	2	9.11147	80.9281
1	335.951	3.22262	2.89039	7	8.92848	83.4911
2	235.203	2.44115	2.48203	1	6.49731	61.8406
3	246.505	2.76656	2.71547	7	7.45089	57.0538
4	437.939	3.4801	3.59569	3	10.9156	131.326

- generation

“create a molecule with the properties XYZ, such that it is non-toxic”

carrier or linker proteins (drug development)

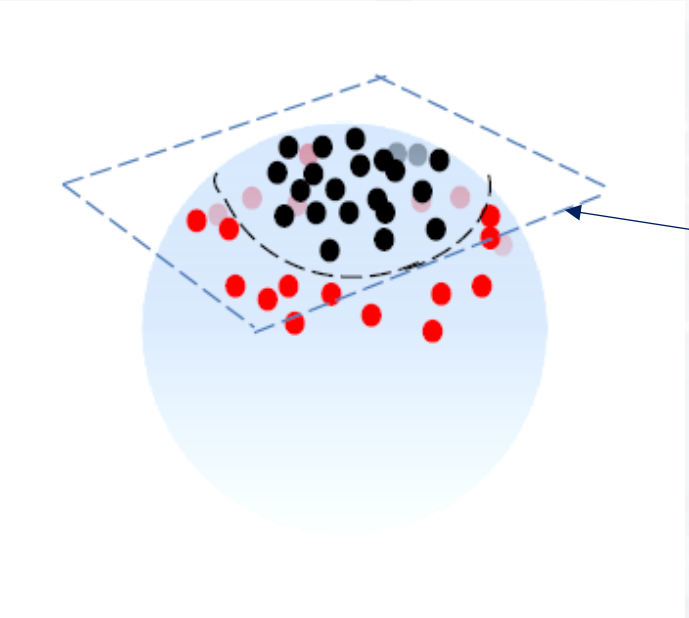
Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

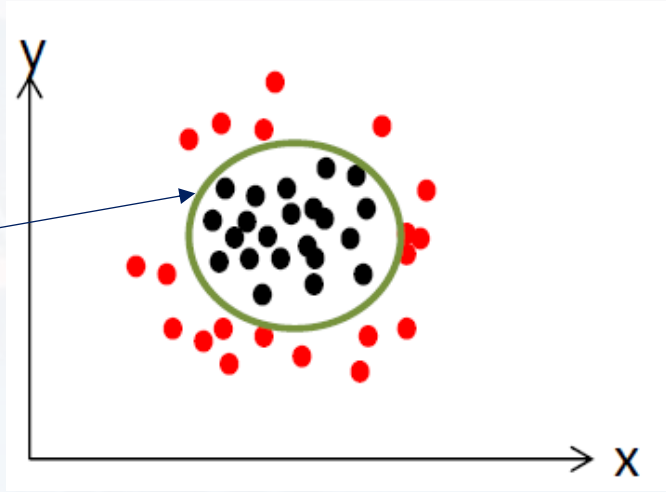
Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic



high dimensional feature space
(visualization via PCA, UMAP, t-SNE etc see later)

- toxic molecules
- non-toxic molecules

classification boundary from ML-algorithm



lower dimensional projection
(visualization via PCA, UMAP, t-SNE etc see later)

Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
data = pd.read_csv('Cystfibr.txt', delimiter = '\t')
```

source

data - DataFrame

Index	age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
0	7	0	109	13.1	68	32	258	183	137	95
1	7	1	112	12.9	65	19	449	245	134	85
2	8	0	124	14.1	64	22	441	268	147	100
3	8	1	125	16.2	67	41	234	146	124	85

feature vector

features can be **numerical**: **units are arbitrary** (cm, inches etc), i.e. the **value itself is arbitrary** (depending on unit system → **normalization**!)

Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
data = pd.read_csv('Cystfibr.txt', delimiter = '\t')
```

source

data - DataFrame

Index	age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
0	7	0	109	13.1	68	32	258	183	137	95
1	7	1	112	12.9	65	19	449	245	134	85
2	8	0	124	14.1	64	22	441	268	147	100
3	8	1	125	16.2	67	41	234	146	124	85

feature vector

features can be **categorical: have to be converted to numerical values → encoding**!)

Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

or:

```
data        = pd.ExcelFile('AD_data.xlsx')
sheet_names = data.sheet_names

AD          = data.parse(sheet_name = sheet_names[1])
```

Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

AD - DataFrame

Index	state	sex	age range(0: <75; 1: >=75)	education	economic	lifestyle	heridity	marriage	health	BIM	p
0	0	0	0	2	1	2	2	3	5	24.19	0.553777
1	0	0	1	1	4	1	1	3	5	18.05	0.375283
2	0	0	0	3	2	2	3	3	5	21.98	0.541784
3	0	0	0	1	1	4	2	3	5	26.18	0.527598

features can be actual **numerical**, but binned to **categorical**

Note: That is no good practice! Binning leads to loss of information!

Feature analysis, engineering and encoding:

What is a feature?

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

AD - DataFrame

Index	state	sex	age range(0: <75; 1: >=75)	education	economic	lifestyle	heridity	marriage	health	BIM	p
0	0	0	0	2	1	2	2	3	5	24.19	0.553777
1	0	0	1	1	4	1	1	3	5	18.05	0.375283
2	0	0	0	3	2	2	3	3	5	21.98	0.541784
3	0	0	0	1	1	4	2	3	5	26.18	0.527598

most features are **categorical**

Number of categories/states for a specific feature is called **cardinality.**

Feature analysis, engineering and encoding:

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

Number of categories for a specific feature is called **cardinality.**

Let's check out cardinality:

```
Categoricals = AD.columns[:-2]

for c in Categoricals:

    card = len(set(AD[c]))
    print("Cardinality of " + c + ': '+ str(card))
```

np.max won't be save: We don't know how categories have been enumerated

Feature analysis, engineering and encoding:

This is a **feature vector**

- tells us where the datapoint is located in **feature space**
- can be **high dimensional**!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

Number of categories for a specific feature is called **cardinality.**

Let's check out cardinality:

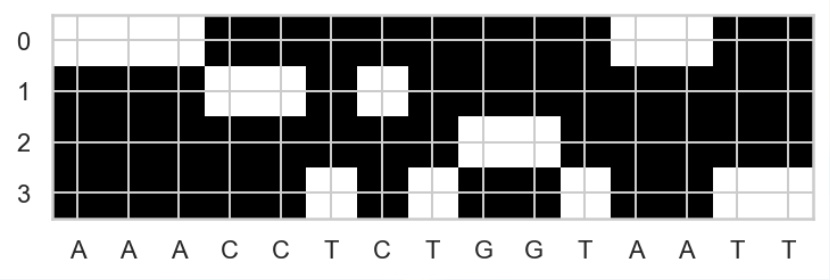
```
Categoricals = AD.columns[:-2]

for c in Categoricals:

    card = len(set(AD[c]))
    print("Cardinality of " + c + ': '+ str(ca
```

```
Cardinality of state: 2
Cardinality of sex: 2
Cardinality of age range(0: <75; 1: >=75): 2
Cardinality of education: 3
Cardinality of economic: 5
Cardinality of lifestyle: 5
Cardinality of heridity: 3
Cardinality of marriage: 3
Cardinality of health: 5
```

Feature analysis, engineering and encoding:

What is a feature?

cardinality ***c*** of features:



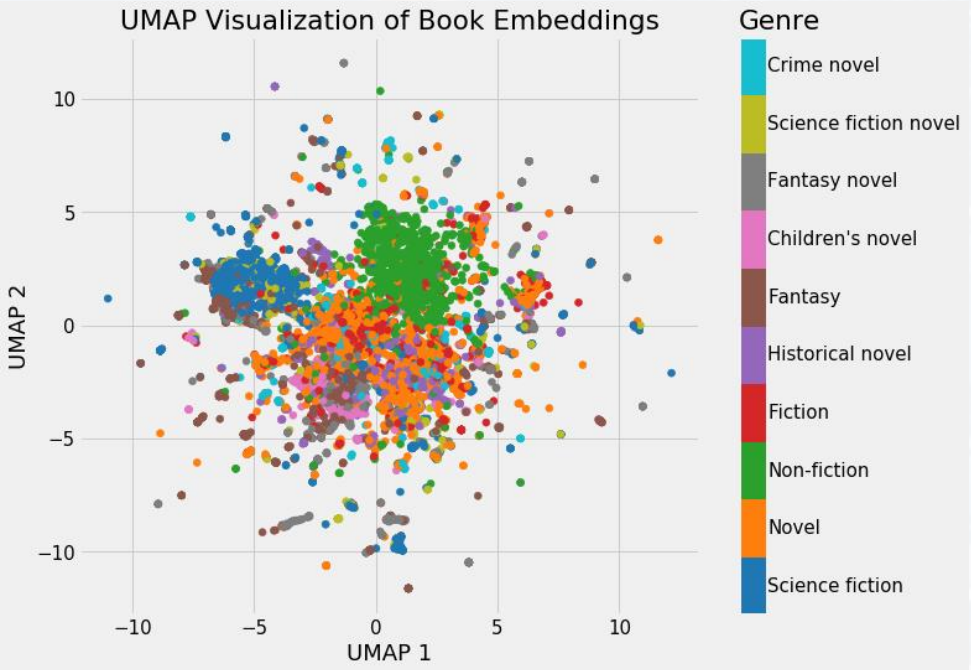
four nucleotides

c = 4

```
;Cytochrome c1, heme protein, mitochondrial

>cow
MAAAAATLRGAMVGPRGAGLPGARARGLLCGARPG
LSALGMLAAGGAGLAVALHSAVSASDLELHPPSYPWS
CSSCHSMDYVAYRHLVGVCYTEDEAKALAEEVEVQDG
```

twenty amino acids

c = 20



source

10^4 words

c = 10^4

Feature analysis, engineering and encoding:

What is a feature?

summary:

- **features** are entries (= variables) of the **feature vector**
- the **feature vector** tells us where the datapoint is located in **feature space**
- the **feature space** can be **high dimensional**
- **features** can be numerical; values change depending on unit system → **normalization**.
- **features** can be categorical → proper **encoding**
- features need to be **equally weighted** so that any algorithm/model identifies patterns in the data and makes predictions **based on those patterns**, not based on the choice of feature encoding!

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Non-Toxic
3	246.505	2.76656	2.71547	7	7.45089	Non-Toxic
4	437.939	3.4801	3.59569	3	10.9156	Non-Toxic

Feature analysis, engineering and encoding

Outline

What is a feature?

Encodings

Normalization

Visualization

Feature analysis, engineering and encoding:

features can be categorical → proper **encoding**

Index	state	sex	age range(0: <75; 1: >=75)	education	economic	lifestyle	heridity	marriage	health	BIM	p
0	0	0	0	2	1	2	2	3	5	24.19	0.553777
1	0	0	1	1	4	1	1	3	5	18.05	0.375283
2	0	0	0	3	2	2	3	3	5	21.98	0.541784
3	0	0	0	1	1	4	2	3	5	26.18	0.527598

```
Cardinality of state: 2
Cardinality of sex: 2
Cardinality of age range(0: <75; 1: >=75): 2
Cardinality of education: 3
Cardinality of economic: 5
Cardinality of lifestyle: 5
Cardinality of heridity: 3
Cardinality of marriage: 3
Cardinality of health: 5
```

Feature analysis, engineering and encoding:

features can be categorical → proper **encoding**

Index	state	sex	age range(0: <75; 1: >=75)	education	economic	lifestyle	heridity	marriage	health	BIM	p
0	0	0	0	2	1	2	2	3	5	24.19	0.553777
1	0	0	1	1	4	1	1	3	5	18.05	0.375283
2	0	0	0	3	2	2	3	3	5	21.98	0.541784
3	0	0	0	1	1	4	2	3	5	26.18	0.527598

```
Cardinality of state: 2
Cardinality of sex: 2
Cardinality of age range(0: <75; 1: >=75): 2
Cardinality of education: 3
Cardinality of economic: 5
Cardinality of lifestyle: 5
Cardinality of heridity: 3
Cardinality of marriage: 3
Cardinality of health: 5
```

- encoding the different categories of the feature “heridity” just 1, 2, 3 …etc **is arbitrary**!
- Why not encoding like -2, -1, 0, 1, 2?
- choice **will affect location** in feature space and therefore **influence result** for any categorization/ regression/generation task
- **What is a good encoding strategy?**

Feature analysis, engineering and encoding:

problem: Features need to be **equally weighted** so that any algorithm/model identifies patterns in the data and makes predictions **based on those patterns**, not based on the choice of feature encoding!

categorical features can be:

ordinal = there is an **inherent order** and features can be **ranked**

example: - health status
poor, fair, good, excellent etc

- grades
A+, A, A-, B+, …

```
Cardinality of economic: 5
Cardinality of lifestyle: 5
Cardinality of heridity: 3
Cardinality of marriage: 3
Cardinality of health: 5
```

nominal = **no inherent order**

example: - ACGT

- name of a person

Feature analysis, engineering and encoding:

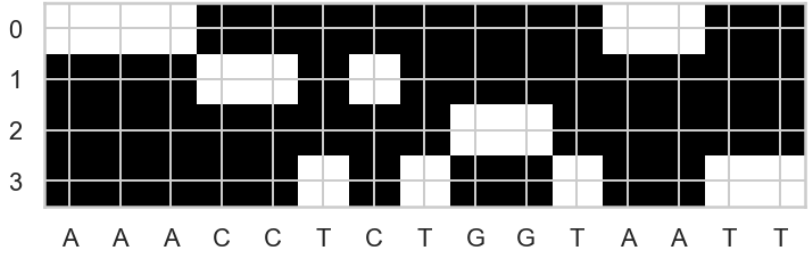
The most common encoding strategies:

- one-hot encoding
- dummy encoding
- ordinal encoding
- binary encoding
- count encoding

Feature analysis, engineering and encoding:

- **fast** and **simple**
- if categories are not related and **purely nominal**
- suitable if **low cardinality**
- each category is a **binary vector**

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding




```
Cat = "ACGT"
Enc = np.identity(len(Cat))

Encoding = {c: e for c, e in zip(Cat, Enc)}
```

Encoding - Dictionary (4 elements)

Key	Type	Size	Value
A	Array of float64	(4,)	[1. 0. 0. 0.]
C	Array of float64	(4,)	[0. 1. 0. 0.]
G	Array of float64	(4,)	[0. 0. 1. 0.]
T	Array of float64	(4,)	[0. 0. 0. 1.]

Feature analysis, engineering and encoding:

- **fast** and **simple**
- if categories are not related and **purely nominal**
- suitable if **low cardinality**
- each category is a **binary vector**

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding

```
Cat = "ACGT"
Enc = np.identity(len(Cat))

Encoding = {c: e for c, e in zip(Cat, Enc)}


Sequence = 'CCTGTAATC'

Encoded  = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

```
[[0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]]
```

Feature analysis, engineering and encoding:

- **similar to one-hot encoding**
- if categories are not related and **purely nominal**
- suitable if **low cardinality**
- each category is a **binary value**


```
AD_dummy = pd.get_dummies(AD, columns = Categoricals, dtype = int)
```

Index	BIM	p	state_0	state_1	sex_0	sex_1	(0: <75; 1:	(0: <75; 1:	education_1	education_2	education_3	economic_1	economic_2	economic_3	economic_4	economic_5
0	24.19	0.553777	1	0	1	0	1	0	0	1	0	1	0	0	0	0
1	18.05	0.375283	1	0	1	0	0	1	1	0	0	0	0	0	1	0
2	21.98	0.541784	1	0	1	0	1	0	0	0	1	0	1	0	0	0
3	26.18	0.527598	1	0	1	0	1	0	1	0	0	1	0	0	0	0

lifestyle_1	lifestyle_2	lifestyle_3	lifestyle_4	lifestyle_5	heridity_1	heridity_2	heridity_3	marriage_1	marriage_2	marriage_3	health_1	health_2	health_3	health_4	health_5
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1

Feature analysis, engineering and encoding:

- **similar to one-hot encoding**
- if categories are not related and **purely nominal**
- suitable if **low cardinality**
- each category is a **binary value**


```
AD_dummy = pd.get_dummies(AD, columns = Categoricals, dtype = int)
```

Index	BIM	p	state_0	state_1	sex_0	sex_1	(0: <75; 1:	(0: <75; 1	education_1	education_2	education_3	economic_1	economic_2	economic_3	economic_4	economic_5
0	24.19	0.553777	1	0	1	0	1	0	0	1	0	1	0	0	0	0
1	18.05	0.375283	1	0	1	0	0	1	1	0	0	0	0	0	1	0
2	21.98	0.541784	1	0	1	0	1	0	0	0	1	0	1	0	0	0
3	26.18	0.527598	1	0	1	0	1	0	1	0	0	1	0	0	0	0

information is redundant: - if i.e. `sex_0 = 1`, we know that `sex_1 = 0`
- 100% correlation, many ML methods will perform poorly

idea: **- drop one column**

```
AD_dummy = pd.get_dummies(AD, columns = Categoricals, dtype = int, drop_first = True)
```

Feature analysis, engineering and encoding:

These features are purely nominal

- one-hot or dummy encoding applicable

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding

Index	BIM	p	state_1	sex_1	age range(0: <75; 1: >=75)_1	education_2	education_3	economic_2	economic_3	economic_4	economic_5
0	24.19	0.553777	0	0	0	1	0	0	0	0	0
1	18.05	0.375283	0	0	1	0	0	0	0	1	0
2	21.98	0.541784	0	0	0	0	1	1	0	0	0
3	26.18	0.527598	0	0	0	0	0	0	0	0	0

lifestyle_2	lifestyle_3	lifestyle_4	lifestyle_5	heridity_2	heridity_3	marriage_2	marriage_3	health_2	health_3	health_4	health_5
1	0	0	0	1	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1
0	0	1	0	1	0	0	1	0	0	0	1

Feature analysis, engineering and encoding:

These features **might be ordinal**!

- one-hot or dummy encoding **might not be applicable!**

Index	BIM	p	state_1	sex_1	age range(0: <75; 1: >=75)_1	education_2	education_3	economic_2	economic_3	economic_4	economic_5
0	24.19	0.553777	0	0	0	1	0	0	0	0	0
1	18.05	0.375283	0	0	1	0	0	0	0	1	0
2	21.98	0.541784	0	0	0	0	1	1	0	0	0
3	26.18	0.527598	0	0	0	0	0	0	0	0	0

lifestyle_2	lifestyle_3	lifestyle_4	lifestyle_5	heridity_2	heridity_3	marriage_2	marriage_3	health_2	health_3	health_4	health_5
1	0	0	0	1	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1
0	0	1	0	1	0	0	1	0	0	0	1

Feature analysis, engineering and encoding:

- if categories are **ordinal**
- assigning numerical value in specific order


```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

widowed or divorced, and 3 representing married with a spouse. The education levels were divided into high (university degree or other professional qualification), middle (high school or junior high school), and low (practical qualification related to work). Economic status was divided into five categories based on the Townsend Deprivation Index (which combines information on social class, employment, cars, housing, etc). Higher scores indicate better marital status, higher education levels and better economic status, respectively.

Feature analysis, engineering and encoding:

- if categories are **ordinal**
- assigning numerical value in specific order


```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

widowed or divorced, and 3 representing married with a spouse. The education levels were divided into high (university degree or other professional qualification), middle (high school or junior high school), and low (practical qualification related to work). Economic status was divided into five categories based on the Townsend Deprivation Index (which combines information on social class, employment, cars, housing, etc). Higher scores indicate better marital status, higher education levels and better economic status, respectively.

Feature analysis, engineering and encoding:

- if categories are **ordinal**
- assigning numerical value in specific order


```
AD = pd.read_excel('AD_data.xlsx', sheet_name = 'Summary Form')
```

source

Index	state	sex	e(0: <75;	education	economic
0	0	0	0	2	1
1	0	0	1	1	4
2	0	0	0	3	2
3	0	0	0	1	1
4	0	0	0	3	2
5	0	0	0	2	4

Feature analysis, engineering and encoding:

- uses **binary system**
- suitable if **high cardinality**
- mixture between **one-hot** and **ordinal** encoding
- **memory efficient** alternative to one-hot or dummy
- **reduces dimensionality**
- often used for language models

one hot:

```
Cat      = "abcdefghijklmnopqrstuvwxyz .!?"
Enc      = np.identity(len(Cat))

Encoding = {c: e for c, e in zip(Cat, Enc)}

Sequence = 'this is a sentence.'
Encoded  = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

Feature analysis, engineering and encoding:


```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0.]]
```

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding

- one-hot uses memory very inefficiently if high cardinality

- dimension of encoded vector = cardinality

Feature analysis, engineering and encoding:

binary:


```
Cat             = "abcdefghijklmnopqrstuvwxyz .!?"

bytes_object    = Cat.encode('utf-8')

Encoding        = {c: f"{b:08b}" for c, b in zip(Cat, bytes_object)}

Sequence        = 'this is a sentence.'
Encoded         = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

turning keys into 8-bit bytes object

Feature analysis, engineering and encoding:

binary:


```
Cat             = "abcdefghijklmnopqrstuvwxyz .!?"

bytes_object    = Cat.encode('utf-8')

Encoding        = {c: f"{b:08b}" for c, b in zip(Cat, bytes_object)}

Sequence        = 'this is a sentence.'
Encoded         = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

Key	Type	Size	Value
a	str	8	01100001
b	str	8	01100010
c	str	8	01100011
d	str	8	01100100
e	str	8	01100101
f	str	8	01100110
g	str	8	01100111

Feature analysis, engineering and encoding:

binary:


```
Cat             = "abcdefghijklmnopqrstuvwxyz .!?"

bytes_object    = Cat.encode('utf-8')

Encoding        = {c: f"{b:08b}" for c, b in zip(Cat, bytes_object)}

Sequence        = 'this is a sentence.'
Encoded         = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

```
['01110100' '01101000' '01101001' '01110011' '00100000' '01101001'
 '01110011' '00100000' '01100001' '00100000' '01110011' '01100101'
 '01101110' '01110100' '01100101' '01101110' '01100011' '01100101'
 '00101110']
```

Feature analysis, engineering and encoding:

one-hot

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 1. 0. 0.]]
```

- one-hot encoding
- dummy encoding
- ordinal encoding
- **binary encoding**
- count encoding

binary

```
['01110100' '01101000' '01101001' '01110011' '00100000' '01101001'
 '01110011' '00100000' '01100001' '00100000' '01110011' '01100101'
 '01101110' '01110100' '01100101' '01101110' '01100011' '01100101'
 '00101110']
```

Feature analysis, engineering and encoding:

- assigns values based on **how often categories** appear in the dataset
- aka **frequency encoding**
- **memory efficient**
- **reduces dimensionality**
- if categories are not related and **purely nominal**
- **for large datasets: rel. frequencies are approximations of probabilities**
- problem: different values for different data sets → **hard to compare for small datasets, data leakage**

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding

```
Sequence = 'this is a sentence.'

Encoding = {s: Sequence.count(s) for s in set(Sequence)}
```

Key	Type	Size	Value
	int	1	3
n	int	1	2
t	int	1	2
.	int	1	1
h	int	1	1
s	int	1	3
c	int	1	1
a	int	1	1
e	int	1	3
i	int	1	2

Feature analysis, engineering and encoding:


```
Sequence = 'this is a sentence.'

Encoding = {s: Sequence.count(s) for s in set(Sequence)}

Encoded  = [Encoding[s] for s in Sequence]

print(np.array(Encoded))
```

```
[2 1 2 3 3 2 3 3 1 3 3 3 2 2 3 2 1 3 1]
```

Feature analysis, engineering and encoding:

one-hot encoding
dummy encoding
ordinal encoding
binary encoding
count encoding

one-hot

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 1. 0. 0.]]
```

binary

```
['01110100' '01101000' '01101001' '01110011' '00100000' '01101001'
 '01110011' '00100000' '01100001' '00100000' '01110011' '01100101'
 '01101110' '01110100' '01100101' '01101110' '01100011' '01100101'
 '00101110']
```

count/frequency

```
[2 1 2 3 3 2 3 3 1 3 3 3 2 2 3 2 1 3 1]
```

Feature analysis, engineering and encoding:

summary

encoding	pros	cons	applicable if
one-hot	simple, variable is binary vector	inefficient for high cardinality	categories are nominal & low cardinality
dummy	simple, variable is binary number	inefficient for high cardinality	categories are nominal & low cardinality
ordinal	efficient for high cardinality	not applicable for nominal categories	categories are ordinal, i. e. have an inherent order
binary	very efficient for high cardinality, reduces dimensionality	less interpretable for humans	recommended if high cardinality
count	simple, easy to interpret	encodes the variable with values from itself (data leakage)	recommended if high cardinality

note, there are more encoding strategies such as target or hash encoding, embeddings etc

Feature analysis, engineering and encoding

Outline

What is a feature?

Encodings

Normalization

Visualization

Feature analysis, engineering and encoding: Normalization

Imagine a competition like **Decathlon** with various disciplines

Decathlon: each athlete has to perform in a competition of **ten** combined track and field events.

100 Meters Running:	measure time
Long Jump:	measure distance
Shot Put:	measure distance
High Jump:	measure distance
400 Meters Running:	measure time
110 Meters Hurdles:	measure time
Discus Throw:	measure distance
Pole Vault:	measure distance
Javelin Throw:	measure distance
1500 Meters Running:	measure time

feature vector

How do you decide who's the best athlete?

- time: the **lower** the value the better
- distance: the **larger** the value the better
- comparing values: being 1sec faster in 100m running are eons, but are no big deal in 1500m running

Feature analysis, engineering and encoding: Normalization

Same problem: how do we compare different quantities measured in arbitrary units?

```
data = pd.read_csv('Cystfibr.txt', delimiter = '\t')
```

source

data - DataFrame

Index	age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
0	7	0	109	13.1	68	32	258	183	137	95
1	7	1	112	12.9	65	19	449	245	134	85
2	8	0	124	14.1	64	22	441	268	147	100
3	8	1	125	16.2	67	41	234	146	124	85

feature vector

features can be **numerical**: **units are arbitrary** (cm, inches etc), i.e. the **value itself is arbitrary** (depending on unit system → **normalization**!)

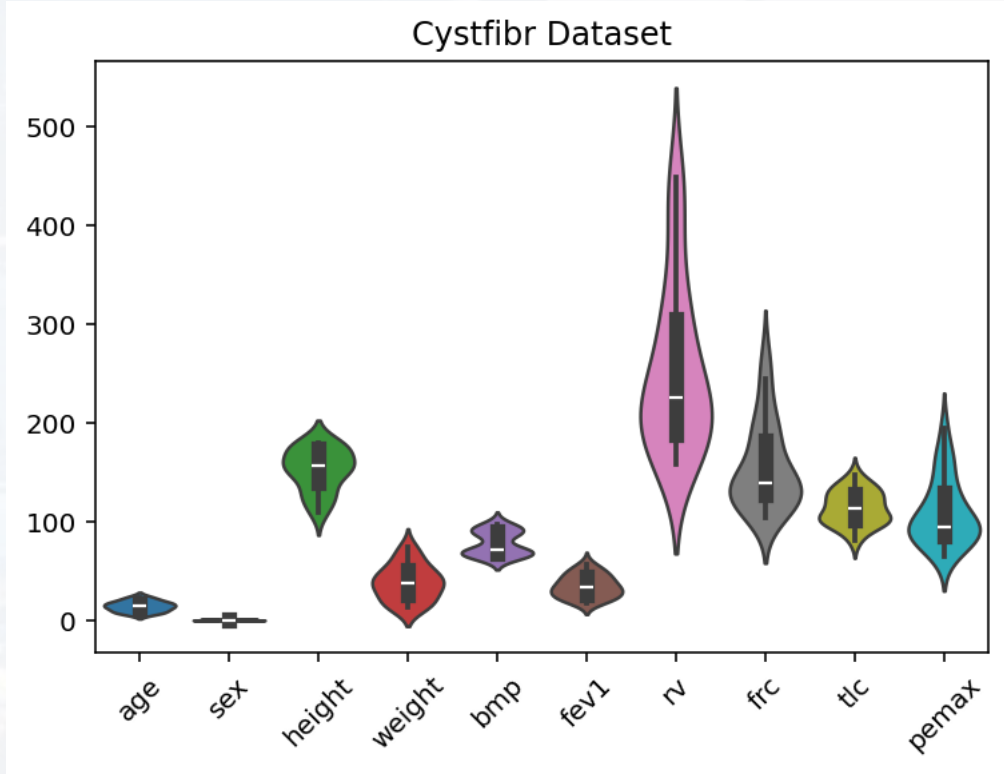
Feature analysis, engineering and encoding: Normalization

two main normalization methods:

min/max: mapping all values to a certain interval (usually 0/1 or -1/+1)

standard: mapping values to standard normal ($\mu = 0, \sigma^2 = 1$)
less robust if outliers: squeezes too many datapoints to $\sigma^2 = 1$

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('Cystfibr.txt',\
       delimiter = '\t')

sns.violinplot(data = data)
plt.title("Cystfibr Dataset")
plt.xticks(rotation = 45)
plt.show()
```



Feature analysis, engineering and encoding: Normalization

two main normalization methods:

scaling data to standard normal

scaling data to min/max range

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler

data = pd.read_csv('Cystfibr.txt', delimiter = '\t')
```

initializing scaler

```
SMinMax = MinMaxScaler(feature_range = (0, 1))
SStand  = StandardScaler()
```

actual scaling: fitting data and transforming

```
Scaled  = SMinMax.fit_transform(data)
```

```
sns.violinplot(data = pd.DataFrame(Scaled, columns = data.columns))
plt.title("Cystfibr Dataset - min/max scaled")
plt.xticks(rotation = 45)
plt.show()
```

scaler returns array, converting back to dataframe

Feature analysis, engineering and encoding: Normalization

two main normalization methods:

scaling data to standard normal

scaling data to min/max range

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler

data = pd.read_csv('Cystfibr.txt', delimiter = '\t')
```

initializing scaler

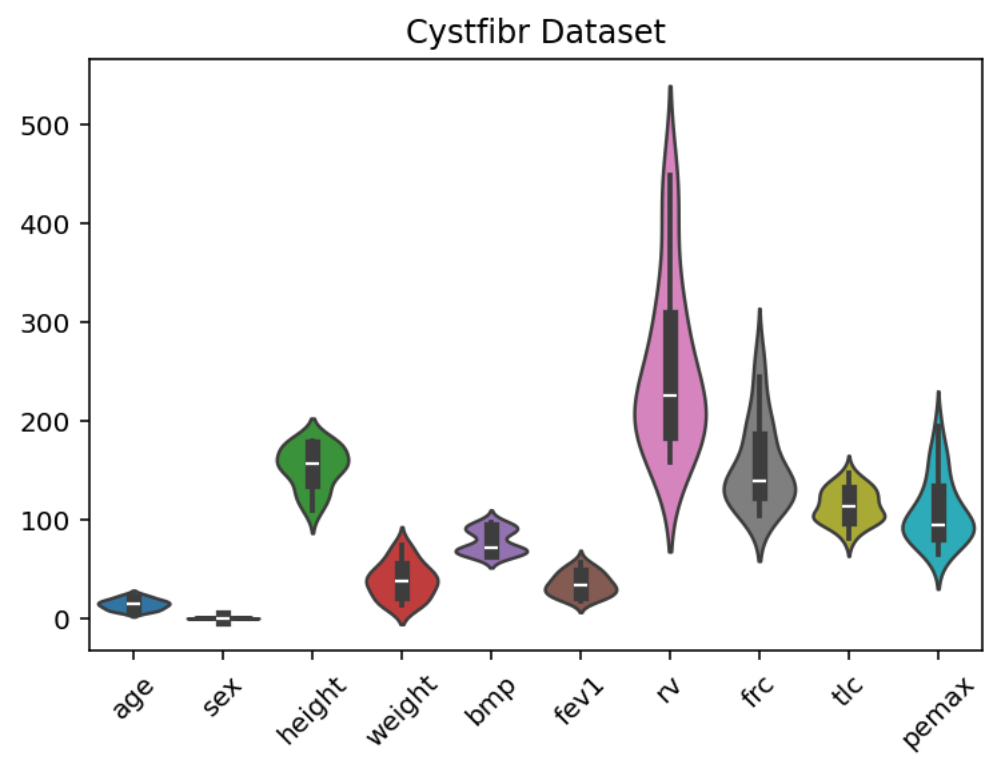
```
SMinMax = MinMaxScaler(feature_range = (0, 1))
SStand  = StandardScaler()
```

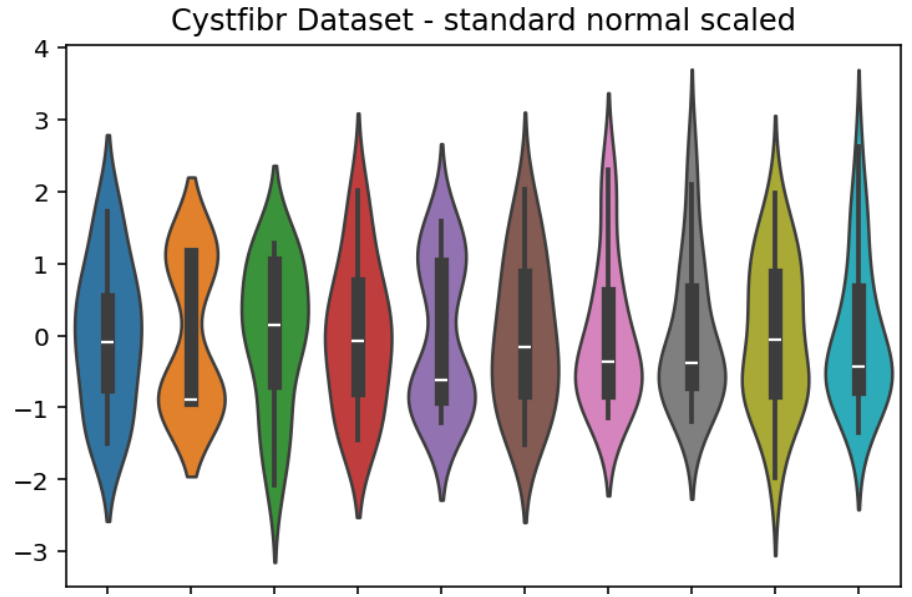
actual scaling: fitting data and transforming

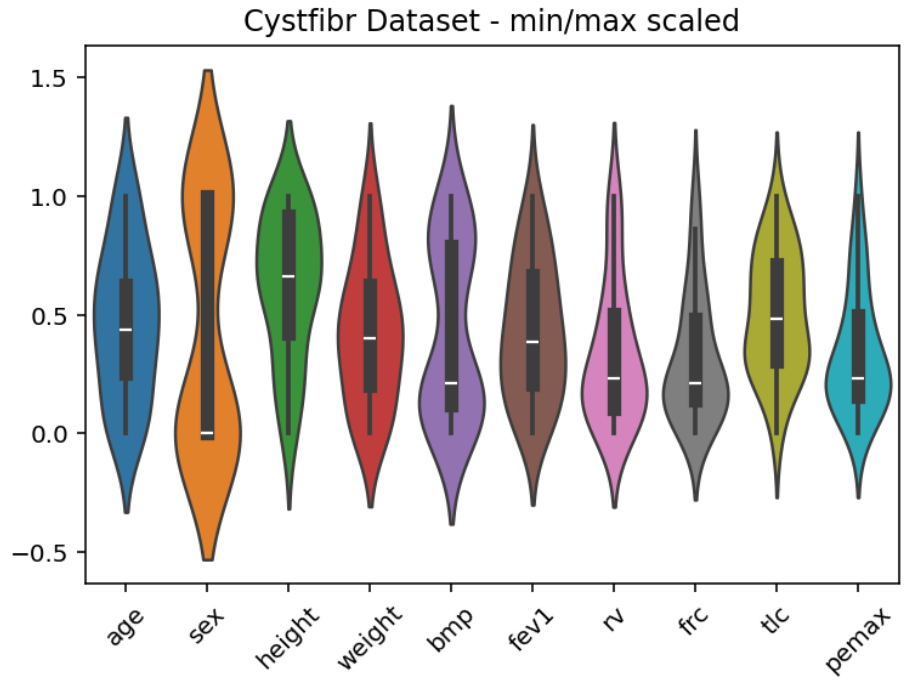
```
Scaled  = SStand.fit_transform(data)

sns.violinplot(data = pd.DataFrame(Scaled, columns = data.columns))
plt.title("Cystfibr Dataset - standard normal scaled")
plt.xticks(rotation = 45)
plt.show()
```

Feature analysis, engineering and encoding: Normalization

two main normalization methods:







Feature analysis, engineering and encoding: Normalization

`fit_transform`
`transform`
`fit`

`fit:` fits the data to the desired scaling

`transform:` performs the actual scaling, applying fitted values

`fit_transform:` performs `fit` and `transform`

Why are there these three options?

later, for supervised learning we need a **training dataset** in order to train a model
→ run `fit_transform`

once the model has been trained, we want to evaluate its performance using a **test dataset**
→ run `transform` only, **not** `fit` in order to test the model unbiased

Feature analysis, engineering and encoding:

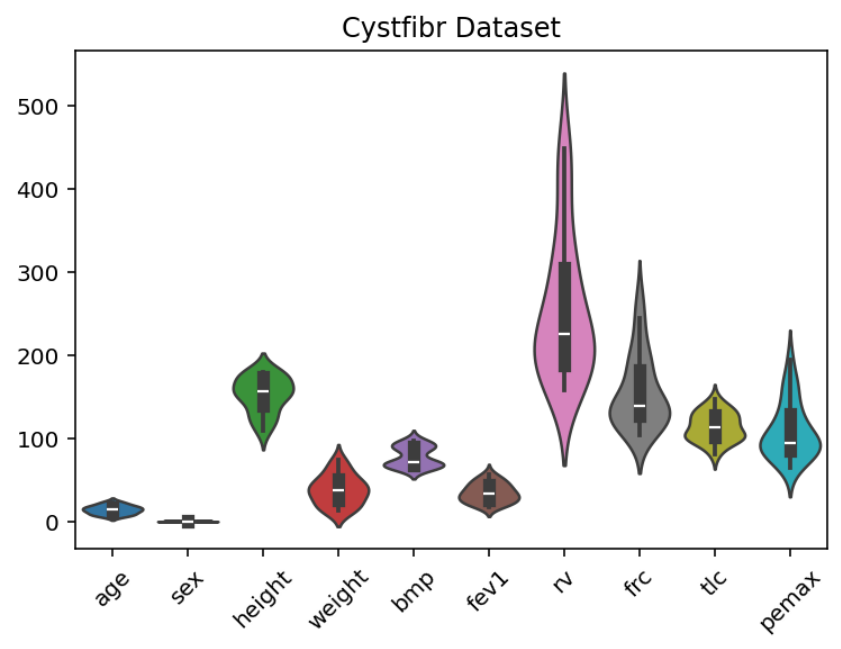
Normalization

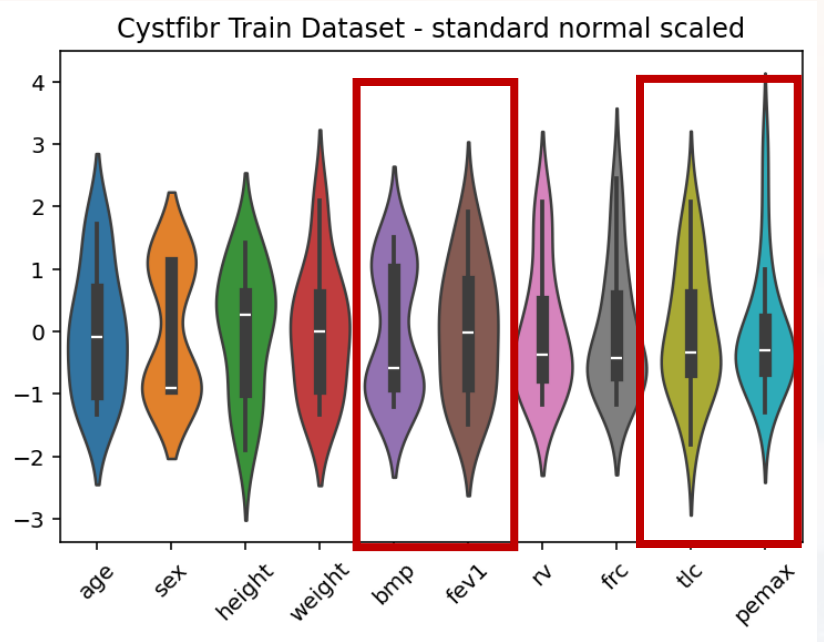
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler

data            = pd.read_csv('Cystfibr.txt', delimiter = '\t')
SStand          = StandardScaler()

Train_80        = data.sample(frac = 0.8)
Test_20         = data.drop(Train_80.index)

Scaled_Train_80 = SStand.fit_transform(Train_80)
Scaled_Test_20  = SStand.transform(Test_20)

sns.violinplot(data = pd.DataFrame(Scaled_Train_80, columns = data.columns))
plt.title("Cystfibr Train Dataset - standard normal scaled")
plt.xticks(rotation = 45)
plt.show()

sns.violinplot(data = pd.DataFrame(Scaled_Test_20, columns = data.columns))
plt.title("Cystfibr Test Dataset - standard normal scaled")
plt.xticks(rotation = 45)
plt.show()
```

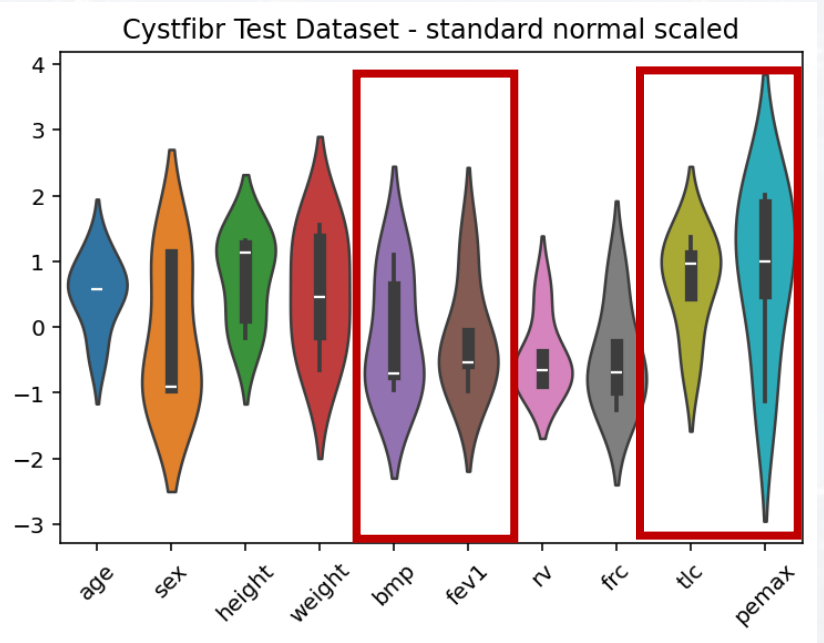
common split training/ test data is 80%/20% **(see random sampling)**

Feature analysis, engineering and encoding:

Normalization







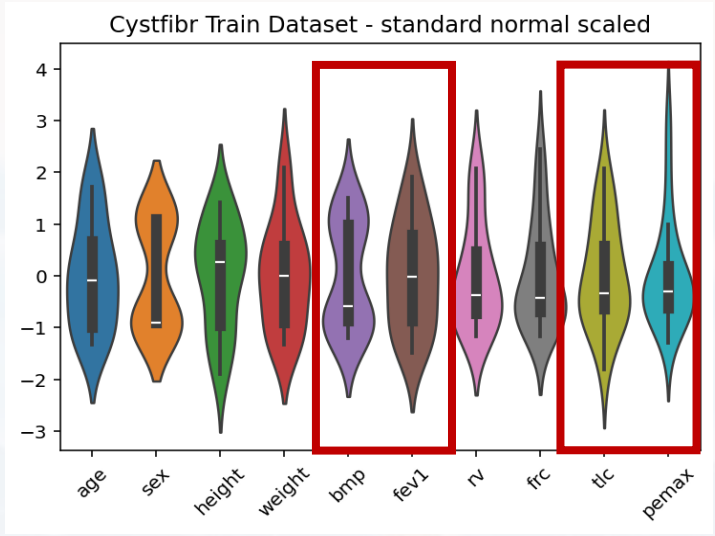
subsamples might (and will!!) indeed be different

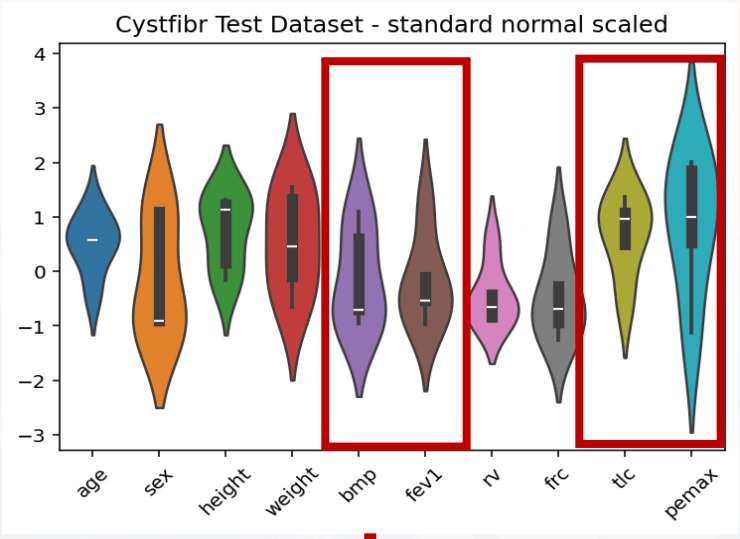
→ running `transform` only on the test set and evaluating the model shows **how the model deals with data variance**

→ running `fit_transform` on the test set too is **data leakage** and **“cheating”**

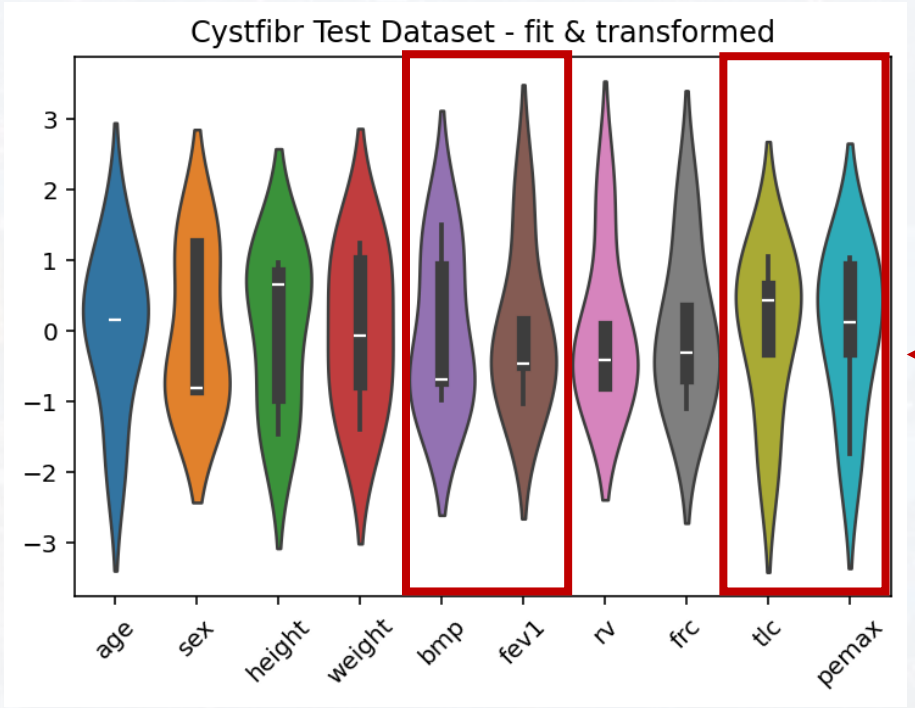
Feature analysis, engineering and encoding:

Normalization



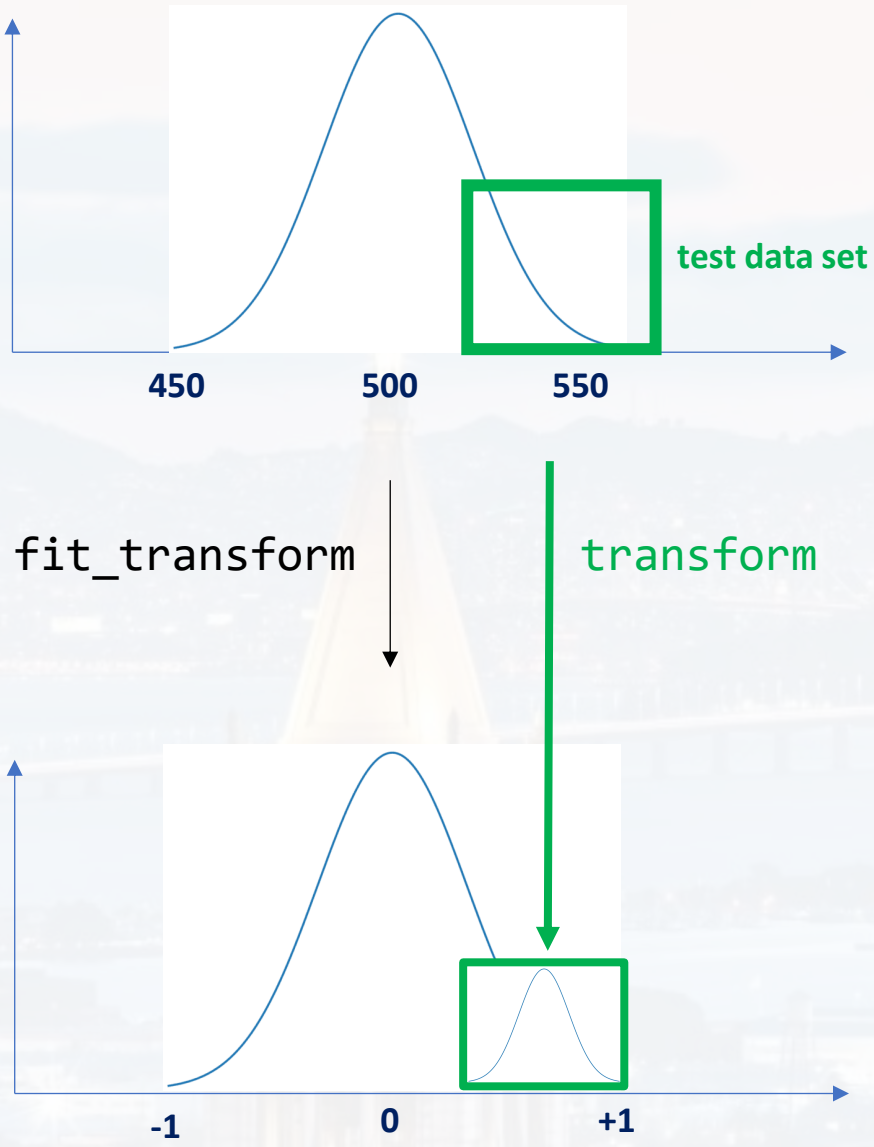


running `fit_transform` on the test set too would transform the test data to ***its*** particular parameters



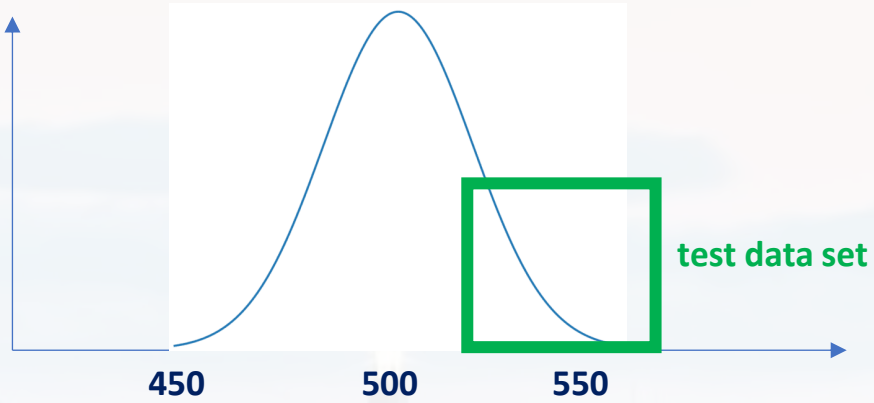
Feature analysis, engineering and encoding:

Normalization



Feature analysis, engineering and encoding:

Normalization

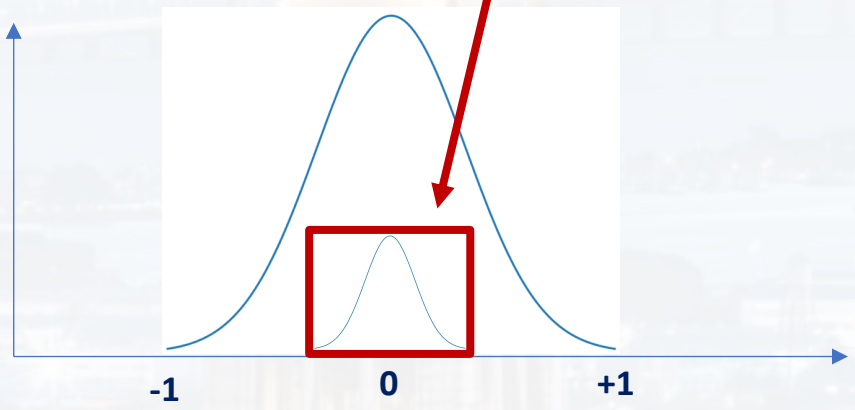


data set (just one feature)

`fit_transform`

`fit_transform`

Wrong!



Feature analysis, engineering and encoding: Normalization

Summary:

- we need to scale features because their **numerical value is arbitrary** (depends on unit system)

- this would confuse the model (**larger numerical values dominate the optimization process**, e.g. MSE etc)

- two common scaling methods:
  `StandardScaler`
  `MinMaxScaler`

- `fit_transform` for **training data**

- `transform` for **test data**

Feature analysis, engineering and encoding: Normalization

Workflow:

1) Encode categorical features! Pick an encoding strategy based on the following criteria:

   - Are the feature values/states **nominal** **(= no inherent order such as gene names, cities etc)**?
   - Are the feature values/states **ordinal** **(= there is an inherent order)** and the feature values can be ranked (health status, tax bracket etc)?
   - **cardinality** of the feature **(= how many states/values can a feature have)**
   - **memory usage**

2) Scale/normalize features using common scaling methods:

```
StandardScaler
MinMaxScaler
```

see also **`Encoding.ipynb`**

Feature analysis, engineering and encoding

Outline

What is a feature?

Encodings

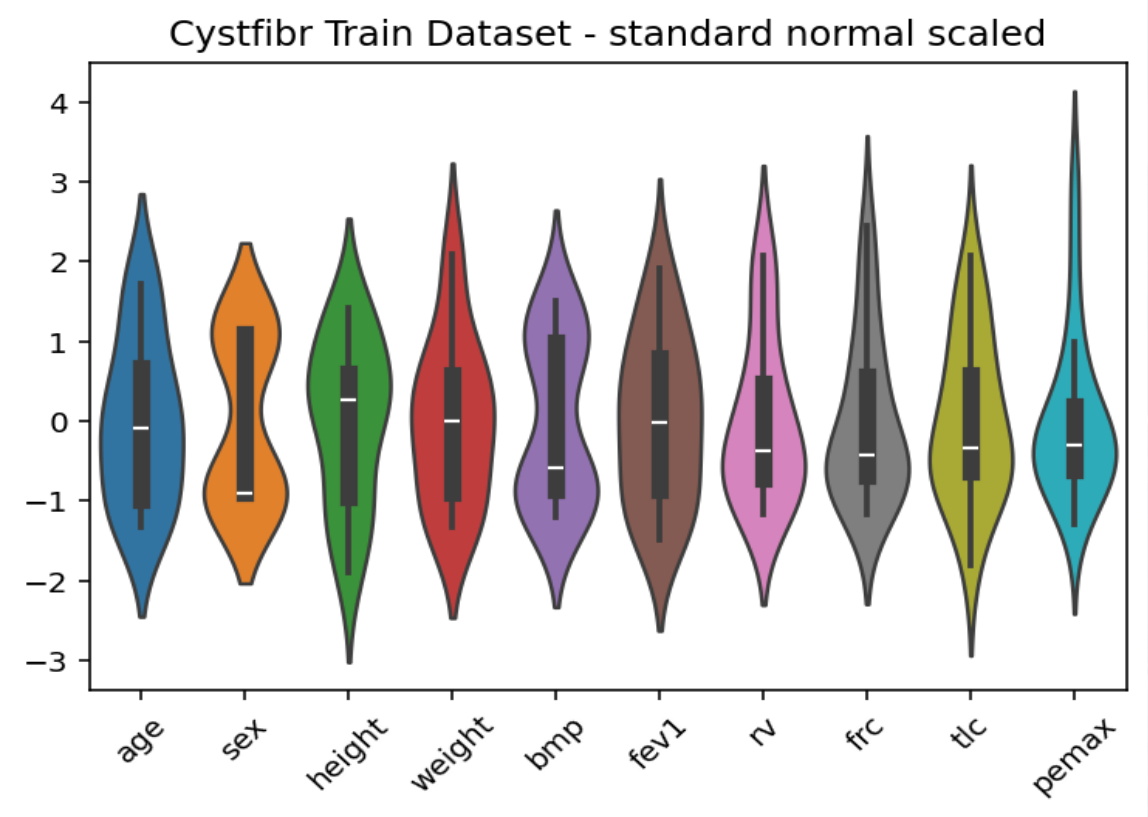
Normalization

Visualization

Feature analysis, engineering and encoding:

We learned already about the **violine plot**:

- shows the distribution of the variables
- quick look in order to get an idea about data range, distribution and outliers



Feature analysis, engineering and encoding:

Another great visualization tool is the **pair plot**:

- shows the distribution of the variables
- quick look in order to get an idea about data range, distribution and outliers

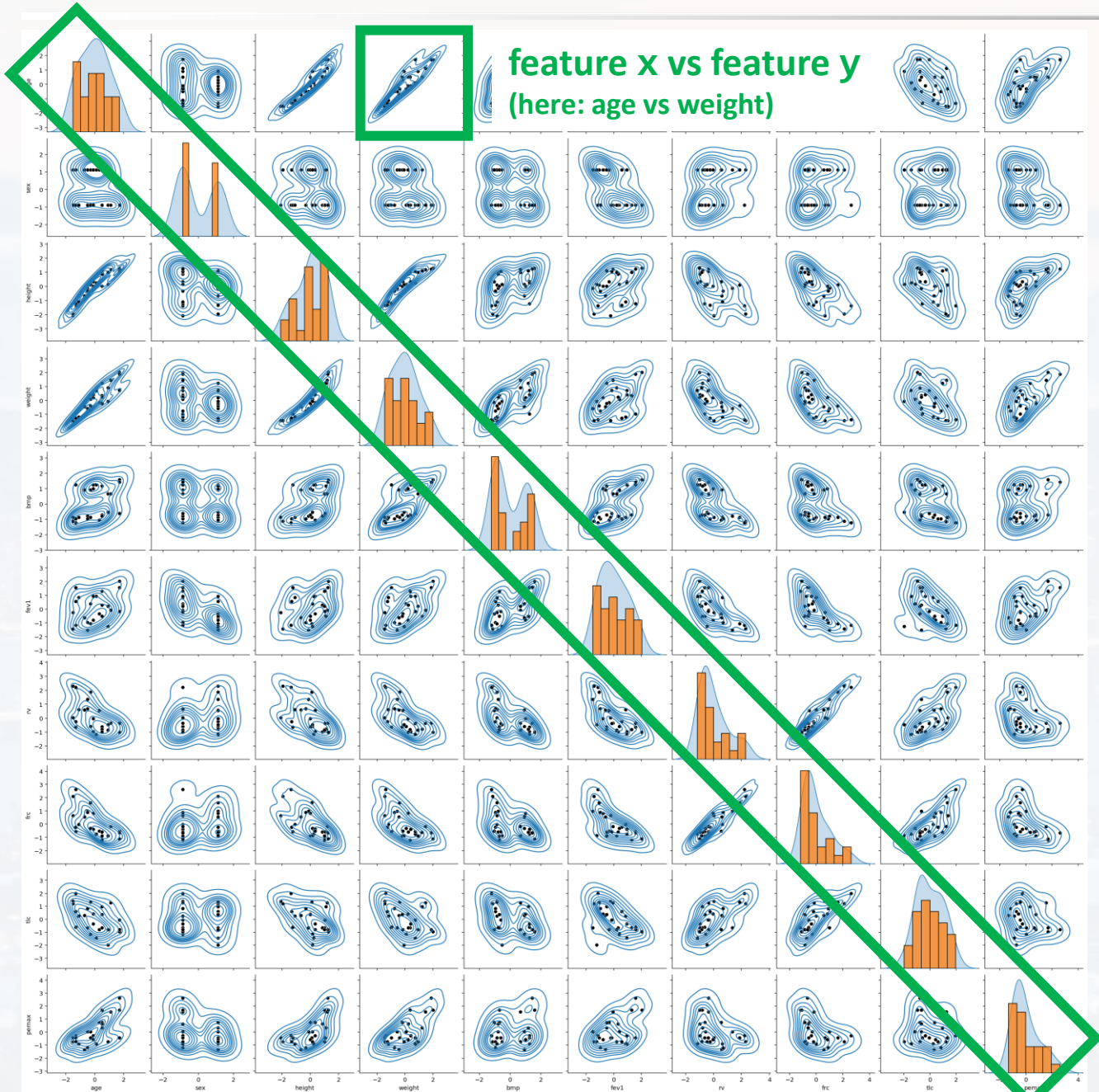
- **shows correlations** (see later)

```
data            = pd.read_csv('Cystfibr.txt', delimiter = '\t')

SStand          = StandardScaler()
Scaled          = SStand.fit_transform(data)

Pair = sns.pairplot(pd.DataFrame(Scaled, columns = data.columns), kind = 'kde')

Pair.map_lower(sns.scatterplot, c = 'k')
Pair.map_upper(sns.scatterplot, c = 'k')
Pair.map_diag(sns.histplot, stat = 'density')
```

Feature analysis, engineering and encoding:

Visualization

pair plot:



age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
7	0	109	13.1	68	32	258	183	137	95
7	1	112	12.9	65	19	449	245	134	85
8	0	124	14.1	64	22	441	268	147	100

Feature analysis, engineering and encoding

Thank you very much for your attention!