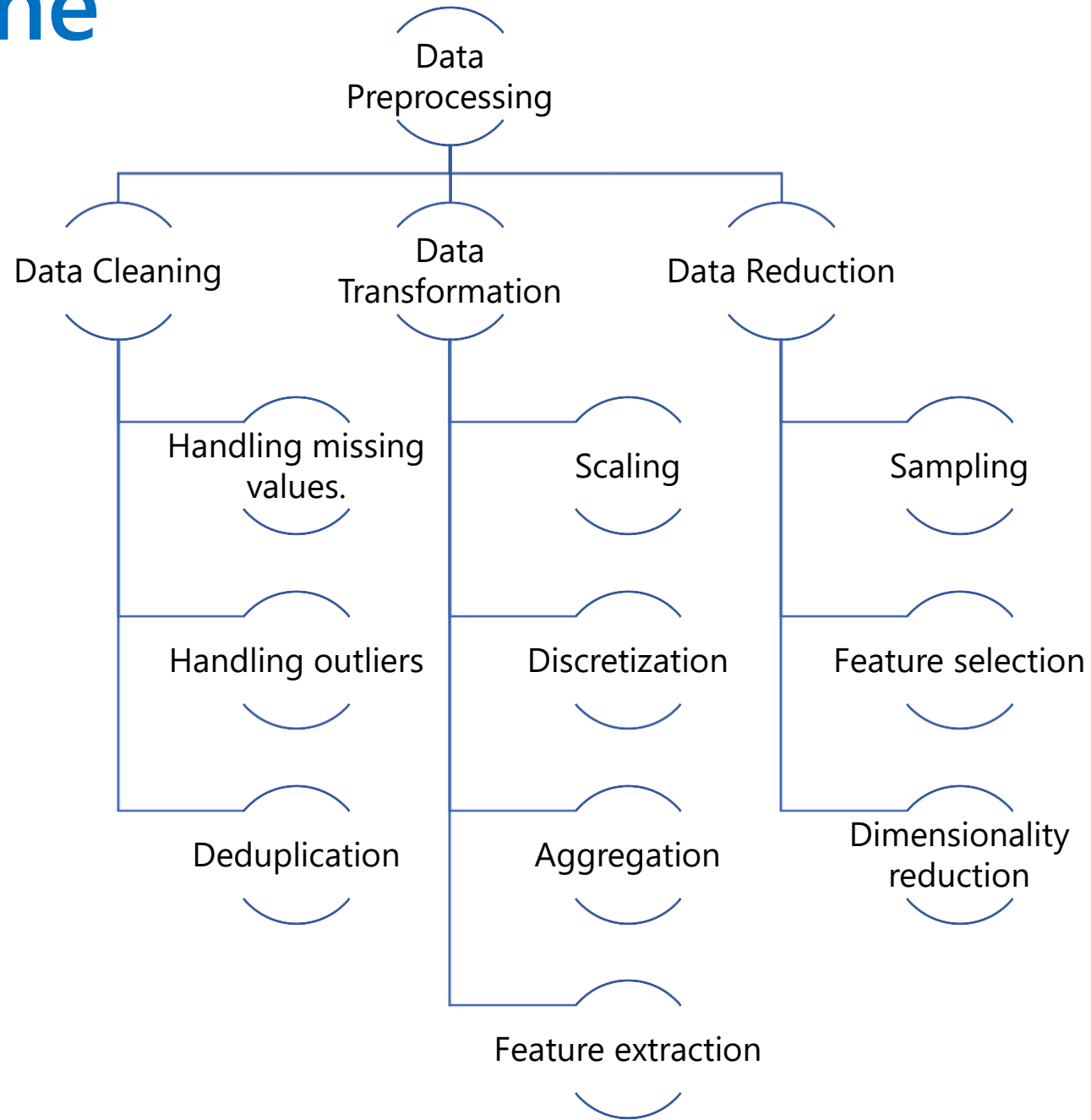


# Data Preprocessing (DP)

---



# Outline



# Issues of Data Quality

- Quality issues due to **invalid data**
  - Caused by errors in the process to generate the features.
  - Fix: correct or regenerate them.
- Quality issues due to **valid data**
  - Exist because of domain-specific reasons.
  - Fix: correct in some cases or do not correct unless required by the trained models, e.g., models cannot be training with missing values or outliers.

# Issues of Data Quality

- **Measurement & Data Collection Issues w.r.t. Quality**

- **Precision:** the closeness of measurements to one another, represented by the standard deviation of the measurements, e.g. repeated measure of body temperature
- **Bias:** a systematic variation of measurements from the intended quantity measurement, only known when external reference available, e.g. bias in weight measure instrument
- **Noise:** modification of original values, e.g. distortion of a person's voice when talking on a poor phone, salary="-10".
- **Outliers:** considerably different from most values in the dataset or unusual with respect to the typical values.
- **Irregular values:** feature values do not match what we expect, e.g., features with the same value for every instance, (0, 1, m, f, M, and F) to for Male/Female.
- **Missing values** (Null values): Not measured or Not available, e.g. people decline to give their age and weight, and annual income is not applicable to children.

# Issues of Data Quality

- **Main Quality Indicators**

- **Accuracy:** data recorded with sufficient precision and little bias
- **Correctness:** data recorded without error and spurious objects
- **Completeness:** any parts of data records missing
- **Consistency:** compliance with established rules and constraints
- **Redundancy:** unnecessary duplicates

# Why Is Data Dirty?

- Incomplete data may come from
  - “Not applicable” data value when collected
  - Different considerations between the time when the data was collected and when it is analyzed.
  - Human/hardware/software problems
- Noisy data (incorrect values) may come from
  - Faulty data collection instruments
  - Human or computer error at data entry
  - Errors in data transmission
- Inconsistent data may come from
  - Different data sources, e.g., e.g., one rating “1,2,3”, another rating “A, B, C”
- Duplicate records

# Issues of Data Quality

- **Why Quality is Important?**

- No quality data, no quality results; “Garbage in, garbage out!”
- Total data quality control requires a cultural change
- For most ML projects, *tackling the quality issue at the data source* cannot be always expected; **workaround?**
  - By cleaning the data as much as possible
  - By developing and using more tolerate ML solutions
- Data quality is relevant to the intended purpose of the ML project, e.g. Does spelling errors in student names really matter when the increase/decrease of student numbers in subject areas over the years are of interest only?

# Handling Data Quality Issues

- **Missing Values:**

- Remove features that are missing in excess of 60% of their values.
- Replace missing values with an indicator (flag).
- Impute with mean, median or mode features that  $< 30\%$  of their values missing.
- build a ML model that estimates a replacement for a missing value based on the other features.

- **Outliers**

- Clamp values above an upper threshold and below a lower threshold to these threshold values.



# Data preprocessing (DP)

- Real world data is seldom in a form suitable for ML algorithms.
  - noisy, has redundant and irrelevant data, or too large with high dimensions, etc.
  - "garbage in, garbage out"
- DP objectives:
  - Transform data into suitable forms, i.e., feature vectors
  - Select/extract relevant features
  - Increase efficiency of ML algorithm
  - Allow for better performance
- DP is highly application-specific, thus needs domain knowledge
- Learners trained with different preprocessed training sets and the best performing set is used for the final model
- DP tasks include:
  - Data cleaning, transformation and reduction





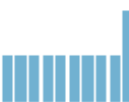




# Handling missing values





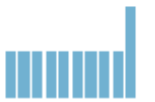


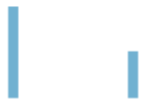

**Missing Values:** some instances without values for one or more features.





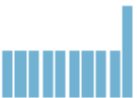

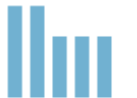

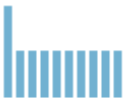
- Remove features/instances that are excessively missing their values.
- Replace missing values with an indicator (flag).
- Impute with mean, median or mode of features
- Build ML model that estimates replacement based on the other features.










Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick		17		4				
Bruce	37	14	63		1	veggie		NA
Steve	83		77	7	1	chicken		n/a
Clint	27	9	118	9		shrimp	3	None
Wanda	19	7	52	2	2	shrimp		empty
Natasha	26	4	162	5	3			-
Carol		3	127	11	1	veggie	1	""
Mandy	44	2	68	8	1	chicken		null





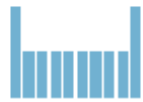




Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick	46	17		4				
Bruce	37	14	63		1	veggie		NA
Steve	83	mean fill	77	7	1	chicken		n/a
Clint	27		118	9		shrimp	3	None
Wanda	19		52	2	2	shrimp		empty
Natasha	26		162	5	3			-
Carol	46	3	127	11	1	veggie	1	*****
Mandy	44	2	68	8	1	chicken		null

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
								
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick	48	17		4				
Bruce	37	14	63		1	veggie		NA
Steve	83	median fill		77	7	chicken		n/a
Clint	27			9		shrimp	3	None
Wanda	19			2	2	shrimp		empty
Natasha	26	4	162	5	3			_
Carol	48	3	127	11	1	veggie	1	""""
Mandy	44	2	68	8	1	chicken		null




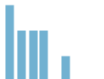






Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
								
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick	46	17		4				
Bruce	37	14	63		1	veggie		NA
Steve	83	12	77	7	1	chicken		n/a
Clint	27	9	118	9		shrimp	3	None
Wanda	19	7	interpolation		2	shrimp		empty
Natasha	26	4			3			_
Carol	46	3			1	veggie	1	""""
Mandy	44	2	68	8	1	chicken		null











Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
								
Tony	48	27	250	1	5	shrimp		Pepper
Donald	67	25	86	not missing at random		beef		Jane
Henry	69	21	95			chicken	62	Janet
Janet	62	21	110			beef		Henry
Nick	46	17	250	4				
Bruce	37	14	63		1	veggie		NA
Steve	83	12	77	7	1	chicken		n/a
Clint	27	9	118	9		shrimp	3	None
Wanda	19	7	52	2	2	shrimp		empty
Natasha	26	4	162	5	3			_
Carol	46	3	127	11	1	veggie	1	""""
Mandy	44	2	68	8	1	chicken		null

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
								
Tony	48	27	78	1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick	34	17	60	imputation				
Bruce	37	14	63				veggie	NA
Steve	83	16	77	7	1	chicken		n/a
Clint	27	9	118	9		shrimp	3	None
Wanda	19	7	52	2	2	shrimp		empty
Natasha	26	4	162	5	3			-
Carol	27	3	127	11	1	veggie	1	*****
Mandy	44	2	68	8	1	chicken		null

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
								
Tony	48	27	250	1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick	46	17	250	4				
Bruce	missing rank order			12	1	veggie		NA
Steve				7	1	chicken		n/a
Clint				9		shrimp	3	None
Wanda				2	2	shrimp		empty
Natasha	26	4	162	5	3			-
Carol	46	3	127	11	1	veggie	1	*****
Mandy	44	2	68	8	1	chicken		null














Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing
									
Tony	48	27	250	1	5	shrimp		Pepper	false
Donald	67	25	86	10	2	beef		Jane	false
Henry	69	21	95	6	1	chicken	62	Janet	false
Janet	62	21	110	3	1	beef		Henry	false
Nick	46	17	250	4	no information				false
Bruce	37	14	63	-99					true
Steve	83	12	77	7	1	chicken		n/a	false
Clint	27	9	118	9		shrimp	3	None	false
Wanda	19	7	52	2	2	shrimp		empty	false
Natasha	26	4	162	5	3			_	false
Carol	46	3	127	11	1	veggie	1	*****	false
Mandy	44	2	68	8	1	chicken		null	false

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing
									
Tony	48	27	250	1	5	shrimp		Pepper	false
Donald	67	25	86	10	2	beef		Jane	false
Henry	69	21	95	6	1	chicken	62	Janet	false
Janet	62	21	110	3	1	beef		Henry	false
Nick	46	17	250	4	0	zero replacement			false
Bruce	37	14	63	-99	1				true
Steve	83	12	77	7	1				false
Clint	27	9	118	9	0				false
Wanda	19	7	52	2	2	shrimp		empty	false
Natasha	26	4	162	5	3			-	false
Carol	46	3	127	11	1	veggie	1	""""	false
Mandy	44	2	68	8	1	chicken		null	false


Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing	attending_party_IsMissing
										
Tony	48	27	250	1	5	shrimp		Pepper	false	false
Donald	67	25	86	10	2	beef		Jane	false	false
Henry	69	21	95	6	1	chicken	62	Janet	false	false
Janet	62	21	110	3	1	beef		Henry	false	false
Nick	46	17	250	4	0	zero replacement with missing flag				true
Bruce	37	14	63	-99	1					false
Steve	83	12	77	7	1					false
Clint	27	9	118	9	0					true
Wanda	19	7	52	2	2	shrimp		empty	false	false
Natasha	26	4	162	5	3			_	false	false
Carol	46	3	127	11	1	veggie	1	""""	false	false
Mandy	44	2	68	8	1	chicken		null	false	false

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing	attending_party_IsMissing	emergency_contact (2)
											
Tony	48	27	250	1	5	shrimp		Pepper	false	false	present
Donald	67	25	86	10	2	beef		Jane	false	false	present
Henry	69	21	95	6	1	chicken	62	Janet	false	false	present
Janet	62	21	multiple representations for "missing"					Henry	false	false	present
Nick	46	17						no	false	true	absent
Bruce	37	14						NA	true	false	absent
Steve	83	12						n/a	false	false	absent
Clint	27	9						None	false	true	absent
Wanda	19	7						empty	false	false	absent
Natasha	26	4						_	false	false	absent
Carol	46	3	127	11	1	veggie	1	""""	false	false	absent
Mandy	44	2	68	8	1	chicken		null	false	false	absent

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing	attending_party_IsMissing
										
Tony	48	27	250	1	5	shrimp	62	Pepper	false	false
Donald	67	25	86	10	2	beef		Jane	false	false
Henry	69	21	95	6	1	chicken		Janet	false	false
Janet	62	21	drop mostly empty columns			beef		Henry	false	false
Nick	46	17						no	false	true
Bruce	37	14				veggie		NA	true	false
Steve	83	12				chicken		n/a	false	false
Clint	27	9	118	9	0	shrimp		None	false	true
Wanda	19	7	52	2	2	shrimp		empty	false	false
Natasha	26	4	162	5	3			_	false	false
Carol	46	3	127	11	1	veggie	1	""""	false	false
Mandy	44	2	68	8	1	chicken		null	false	false

drop rows missing  
critical values

Column 0	age	years_seniority	income	parking_space	attending_party	entree	emergency_contact	parking_space_IsMissing	attending_party_IsMissing	emergency_con (2)
Henry	65	21	95	0	5	shrimp	Pepper	false	false	present
Janet	62	21	110	3	2	beef	Jane	false	false	present
Nick	46	17	250	4	1	chicken	Janet	false	false	present
Bruce	37	14	63	-99	1	veggie	NA	true	false	absent
Steve	83	12	77	7	1	chicken	n/a	false	false	absent
Clint	27	9	118	9	0	shrimp	None	false	true	absent
Wanda	19	7	52	2	2	shrimp	empty	false	false	absent
Natasha	26	4	162	5	3		-	false	false	absent
Carol	46	3	127	11	1	veggie	""	false	false	absent
Mandy	44	2	68	8	1	chicken	null	false	false	absent

Restaurant Data Analysis: Customer Profiles and Preferences											
Column 0	age	years_seniority	income	parking_space	attending_party	entree	emergency_contact	parking_space_IsMissing	attending_party_IsMissing	emergency_contact_IsMissing (2)	
											
Tony	48	27	250	1	5	shrimp	Pepper	false	false	present	
Donald	67	25	86	10	2	beef	Jane	false	false	present	
Henry	69	21	95	6	1	chicken	Janet	false	false	present	
Janet	62	21	110	3	1	beef	Henry	false	false	present	
Bruce	37	14	63	-99	1	veggie	NA	true	false	absent	
Steve	83	12	77	7	1	chicken	n/a	false	false	absent	
Clint	27	9	118	9	0	shrimp	None	false	true	absent	
Wanda	19	7	52	2	2	shrimp	empty	false	false	absent	
Carol	46	3	127	11	1	veggie	""""	false	false	absent	
Mandy	44	2	68	8	1	chicken	null	false	false	absent	

# Handling missing values

**Missing Values:** some instances without values for one or more features.

**Remove features/instances that are excessively missing their values.**

Replace missing values with an indicator (flag), or specific values (domain knowledge)

Impute with mean, median or mode of features

Build ML model that estimates replacement based on the other features.

```
# Creating the dataframe
df = pd.DataFrame({"A": [12, None, 1, None, 1],
                  "B": [None, 2, 2, 3, 2],
                  "C": [10, 3, 6, 5, 6],
                  "D": [14, None, 6, None, 6],
                  "E": [20, None, 8, 3, 8],
                  "F": [10, 3, 6, 5, 6],
                  })
```

```
# Print the dataframe
df
```

	A	B	C	D	E	F
0	12.0	NaN	10	14.0	20.0	10
1	NaN	2.0	3	NaN	NaN	3
2	1.0	2.0	6	6.0	8.0	6
3	NaN	3.0	5	NaN	3.0	5
4	1.0	2.0	6	6.0	8.0	6

```
#remove rows with any NaNs
dfCopy = df.copy()
dfCopy = dfCopy.dropna()
dfCopy
```

	A	B	C	D	E	F
2	1.0	2.0	6	6.0	8.0	6
4	1.0	2.0	6	6.0	8.0	6

```
#remove columns with any NaNs
dfCopy = df.copy()
dfCopy = dfCopy.dropna(axis=1)
dfCopy
```

	C	F
0	10	10
1	3	3
2	6	6
3	5	5
4	6	6



# Handling missing values

**Missing Values:** some instances without values for one or more features.

Remove features/instances that are excessively missing their values.

**Replace missing values with an indicator (flag), or specific values (domain knowledge)**

Impute with mean, median or mode of features

Build ML model that estimates replacement based on the other features.

```
# Creating the dataframe
df = pd.DataFrame({"A": [12, None, 1, None, 1],
                  "B": [None, 2, 2, 3, 2],
                  "C": [10, 3, 6, 5, 6],
                  "D": [14, None, 6, None, 6],
                  "E": [20, None, 8, 3, 8],
                  "F": [10, 3, 6, 5, 6],
                  })
```

```
# Print the dataframe
df
```

	A	B	C	D	E	F
0	12.0	NaN	10	14.0	20.0	10
1	NaN	2.0	3	NaN	NaN	3
2	1.0	2.0	6	6.0	8.0	6
3	NaN	3.0	5	NaN	3.0	5
4	1.0	2.0	6	6.0	8.0	6

```
dfCopy = df.copy()
dfCopy.fillna(5.0, inplace=True)
dfCopy
```

	A	B	C	D	E	F
0	12.0	5.0	10	14.0	20.0	10
1	5.0	2.0	3	5.0	5.0	3
2	1.0	2.0	6	6.0	8.0	6
3	5.0	3.0	5	5.0	3.0	5
4	1.0	2.0	6	6.0	8.0	6

```
#replace the NaNs with different values for columns
dfCopy = df.copy()
dfCopy['A'].fillna(5, inplace=True)
dfCopy['B'].fillna(7, inplace=True)
dfCopy['C'].fillna(3, inplace=True)
dfCopy['D'].fillna(4, inplace=True)
dfCopy['E'].fillna(6, inplace=True)
dfCopy
```

	A	B	C	D	E	F
0	12.0	7.0	10	14.0	20.0	10
1	5.0	2.0	3	4.0	6.0	3
2	1.0	2.0	6	6.0	8.0	6
3	5.0	3.0	5	4.0	3.0	5
4	1.0	2.0	6	6.0	8.0	6

# Handling missing values

**Missing Values:** some instances without values for one or more features.

Remove features/instances that are excessively missing their values.

Replace missing values with an indicator (flag).

## Impute with mean, median or mode of features

Build ML model that estimates replacement based on the other features

```
# Creating the dataframe
df = pd.DataFrame({"A": [12, None, 1, None, 1],
                  "B": [None, 2, 2, 3, 2],
                  "C": [10, 3, 6, 5, 6],
                  "D": [14, None, 6, None, 6],
                  "E": [20, None, 8, 3, 8],
                  "F": [10, 3, 6, 5, 6],
                  })

# Print the dataframe
df
```

	A	B	C	D	E	F
0	12.0	NaN	10	14.0	20.0	10
1	NaN	2.0	3	NaN	NaN	3
2	1.0	2.0	6	6.0	8.0	6
3	NaN	3.0	5	NaN	3.0	5
4	1.0	2.0	6	6.0	8.0	6

```
df.mean() #It i
A      4.666667
B      2.250000
C      6.000000
D      8.666667
E      9.750000
F      6.000000
dtype: float64
```

```
dfCopy = df.copy()
dfCopy['A'].fillna(df['A'].mean(),inplace=True)
dfCopy['B'].fillna(df['B'].mean(),inplace=True)
dfCopy['C'].fillna(df['C'].mean(),inplace=True)
dfCopy['D'].fillna(df['D'].mean(),inplace=True)
dfCopy['E'].fillna(df['E'].mean(),inplace=True)
dfCopy
```

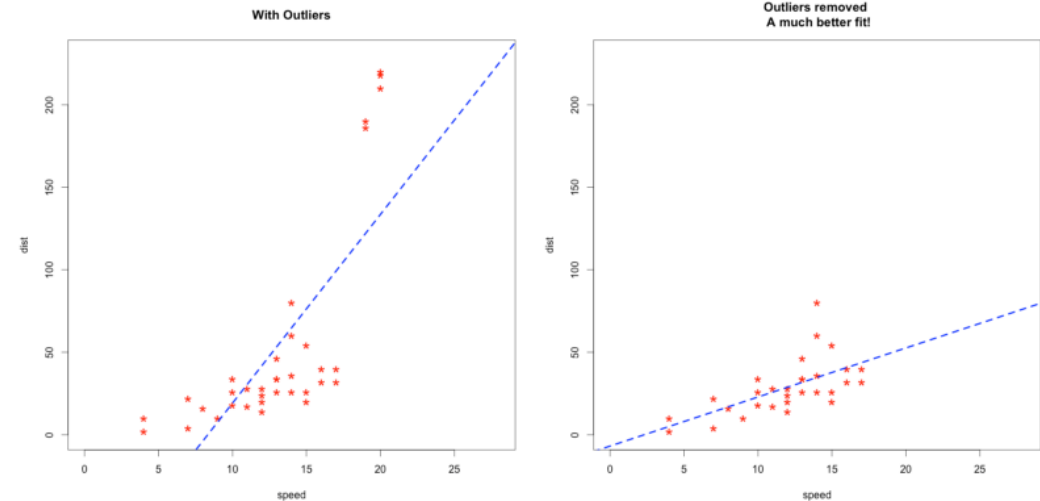
	A	B	C	D	E	F
0	12.000000	2.25	10	14.000000	20.00	10
1	4.666667	2.00	3	8.666667	9.75	3
2	1.000000	2.00	6	6.000000	8.00	6
3	4.666667	3.00	5	8.666667	3.00	5
4	1.000000	2.00	6	6.000000	8.00	6

# Handling outliers

<https://www.r-bloggers.com/outlier-detection-and-treatment-with-r/>

- **Outliers:** points that are considerably different from the other instances.

- Some ML techniques, e.g., logistic regression, are sensitive to outliers.
- Remove/clamp/not is application-dependent, as they can:
  - contain useful information
  - be noise, e.g., due to measurement error
- In large data, outliers are expected and if in small number, they are usually not a real problem
- Detected using:
  - (Visually) box, scatter, bar or histogram plots
  - (Mathematically) IQR, mean/standard deviations, etc.
  - (Algorithmically) clustering



# Data deduplication

- **Duplicates:** instances with the exact same feature values.
- ML algorithms produce different results if some data is duplicated (repetition gives them more influence/bias on the result). E.g.,
  - The same person submits a form more than once.
  - Retweets are Tweets with the exact same content as the original Tweet except for metadata such as the timestamp and the user who retweeted it
- Detected using:
  - **Simple comparison of the instances**
  - Clustering, since similarity metrics are used

```
# Creating the dataframe
df = pd.DataFrame({"A": [12, None, 1, None, 1],
                   "B": [None, 2, 2, 3, 2],
                   "C": [10, 3, 6, 5, 6],
                   "D": [14, None, 6, None, 6],
                   "E": [20, None, 8, 3, 8],
                   "F": [10, 3, 6, 5, 6],
                   })

# Print the dataframe
df
```

	A	B	C	D	E	F
0	12.0	NaN	10	14.0	20.0	10
1	NaN	2.0	3	NaN	NaN	3
2	1.0	2.0	6	6.0	8.0	6
3	NaN	3.0	5	NaN	3.0	5
4	1.0	2.0	6	6.0	8.0	6

```
df.duplicated() #True means the entry is duplicated.
```

```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

```
dfCopy = df[~df.duplicated()]
dfCopy.head()
```

	A	B	C	D	E	F
0	12.000000	2.25	10.0	14.000000	20.00	10.0
1	4.666667	2.00	3.0	8.666667	9.75	3.0
2	1.000000	2.00	6.0	6.000000	8.00	6.0
3	4.666667	3.00	5.0	8.666667	3.00	5.0

# Normalization – Range Normalization

- Continuous features that cover very different ranges can cause difficulty for some ML algorithms
  - e.g., Ages cover [16, 96], whereas Salaries range are [10,000, 100,000]—features with large values (scales) dominate

Person	Age	Salary
1	20	25,000
2	65	25,500
3	25	25,700
4	22	26,900
5	55	25,400

- Person 1 is closer to Person 3 than Person 2, however, some distance functions will say that Person 1 and Person 2 are more similar.
- Example:  $distance(x, y) = \sum_{i=1}^n |x_i - y_i|$

$$|x\_Age - y\_Age| + |x\_Salary - y\_Salary|$$

- $distance(1, 2) = 45 + 500 = 545$
- $distance(1, 3) = 5 + 700 = 705$

# Normalization – Range Normalization

- **Range Normalization:**

- **Linear scaling** of the original values of the continuous feature into a given range  $[low, high]$ .

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \times (high - low) + low$$

where  $x'_i$  is the normalized feature value,  $x_i$  is the original value,  $\min(x)$  is the minimum value of feature  $x$ ,  $\max(x)$  is the maximum value of feature  $x$ , and  $low$  and  $high$  are the minimum and maximum values of the desired range.

- Typical ranges are  $[0, 1]$  and  $[-1, 1]$
- When ranges are  $[0, 1]$

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

- Sensitive to outliers.

```
iris = sns.load_dataset('iris') #Load iris dataset as a dataframe
```

```
iris.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

```
#Linear Scaling; new attributes' range: 0 to 1
# copy the data
data = iris.copy()

# Normalize the attributes, ignore the class Label (last attribute)
for i in range(len(data.columns)-1):
    column = data.columns[i]
    data[column] = (data[column] - data[column].min()) / (data[column].max() - data[column].min())

#view first 10 rows
data.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	0.222222	0.625000	0.067797	0.041667	setosa
1	0.166667	0.416667	0.067797	0.041667	setosa
2	0.111111	0.500000	0.050847	0.041667	setosa
3	0.083333	0.458333	0.084746	0.041667	setosa
4	0.194444	0.666667	0.067797	0.041667	setosa
5	0.305556	0.791667	0.118644	0.125000	setosa
6	0.083333	0.583333	0.067797	0.083333	setosa
7	0.194444	0.583333	0.084746	0.041667	setosa
8	0.027778	0.375000	0.067797	0.041667	setosa
9	0.166667	0.458333	0.084746	0.000000	setosa

# Normalization

- **Standardization:**

- Standardize data into standard scores.
  - How many standard deviations a feature value is from the mean for that feature.

$$x'_i = \frac{x_i - \mu}{\sigma}$$

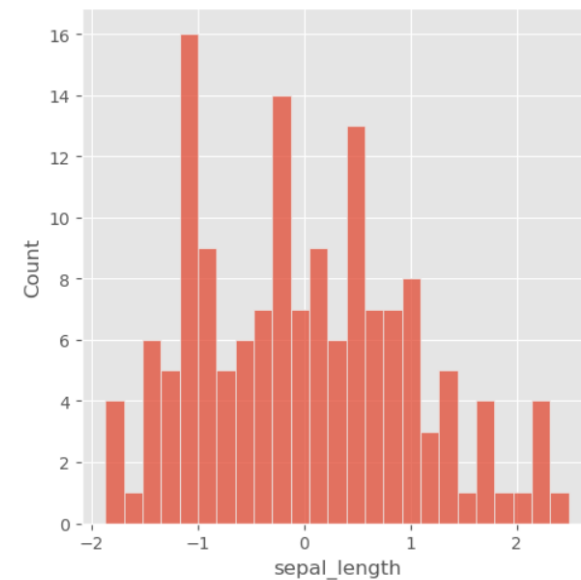
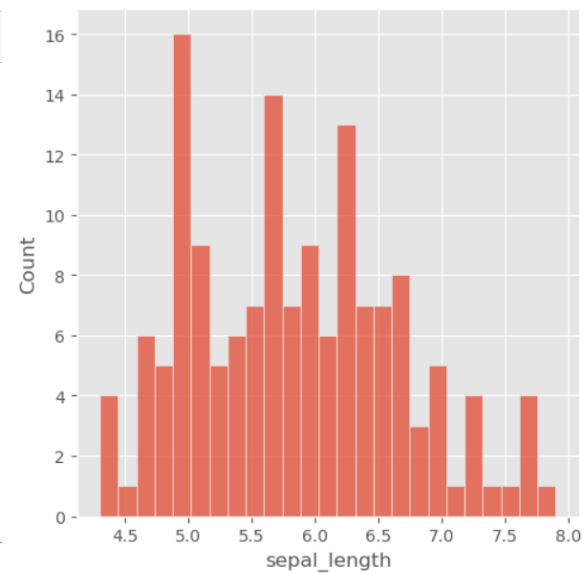
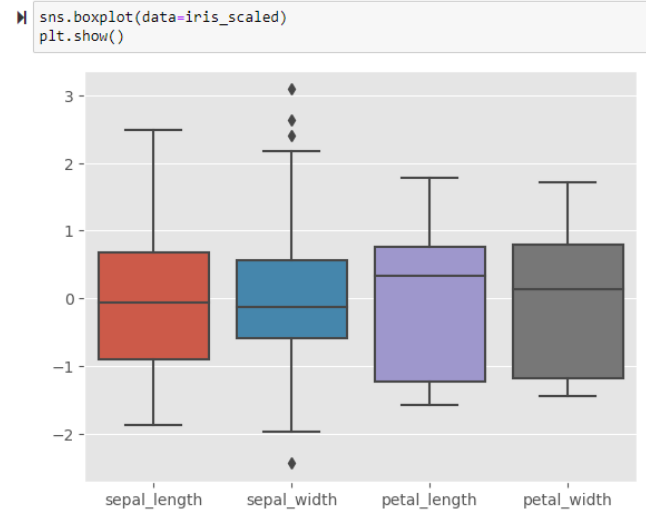
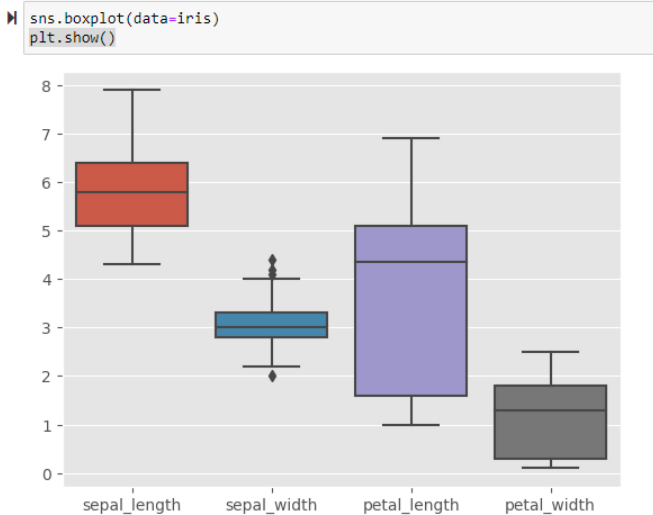
where  $x'_i$  is the normalized feature value,  $x_i$  is the original value,  $\mu$  is the mean for feature  $x$ , and  $\sigma$  is the standard deviation for  $x$ .

- Squashes the values to have a mean of 0 and a standard deviation of 1.
- Results in the majority of values in range  $[-1, 1]$ .
- Handles outliers but columns do not have the same scale.
- Assumes data is normally distributed
  - If this does not hold, then it may introduce some distortions

	HEIGHT			SPONSORSHIP EARNINGS		
	Values	Range	Standard	Values	Range	Standard
	192	0.500	-0.073	561	0.315	-0.649
	197	0.679	0.533	1,312	0.776	0.762
	192	0.500	-0.073	1,359	0.804	0.850
	182	0.143	-1.283	1,678	1.000	1.449
	206	1.000	1.622	314	0.164	-1.114
	192	0.500	-0.073	427	0.233	-0.901
	190	0.429	-0.315	1,179	0.694	0.512
	178	0.000	-1.767	1,078	0.632	0.322
	196	0.643	0.412	47	0.000	-1.615
	201	0.821	1.017	1111	0.652	0.384
<b>Max</b>	206			1,678		
<b>Min</b>	178			47		
<b>Mean</b>	193			907		
<b>Std. Dev.</b>	8.26			532.18		



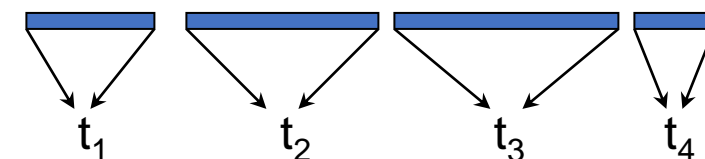
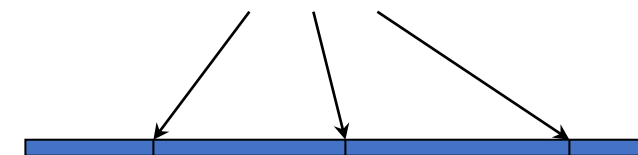
```
iris_scaled = iris.copy()
X = iris_scaled.values[:,0:-1] #columns, except the class label, of the matrix of the dataframe
X = X.astype(float) #convert X's dtype from object to float
X_scaled = preprocessing.StandardScaler().fit_transform(X)
iris_scaled.iloc[:,0:-1] = X_scaled
iris_scaled.head()
```



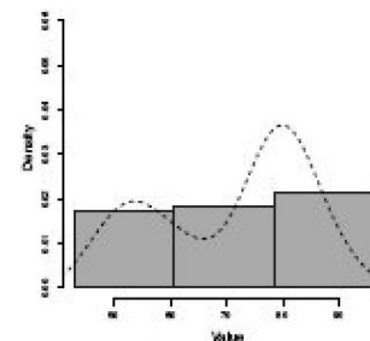
# Discretization (Binning)

- Converting a continuous feature into a categorical feature.
  - Bins: ranges correspond to the categorical values of the new feature.
  - equal-width vs. equal-frequency binning
  - Number of bins? Difficult
    - Trade-off: very low; loss of information re. the distribution of values in the original continuous feature. as the number of bins grows, we can end up with empty bins

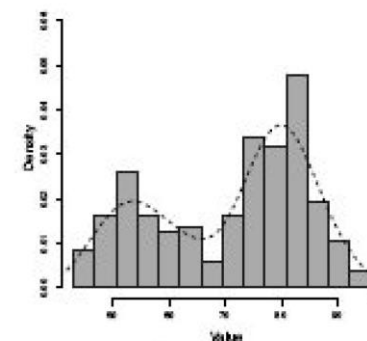
① Determine the number & locations of the split points



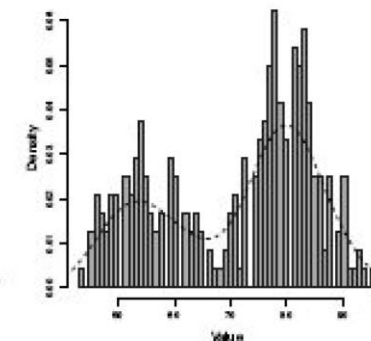
② Mapping values within each sub-range to a category label



(a) 3 bins



(b) 14 bins



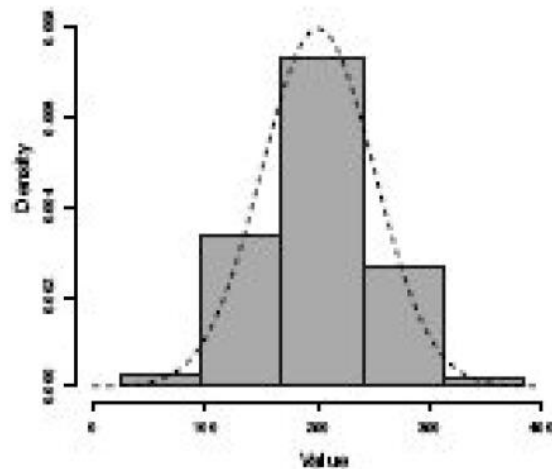
(c) 60 bins

# Discretization (Binning)

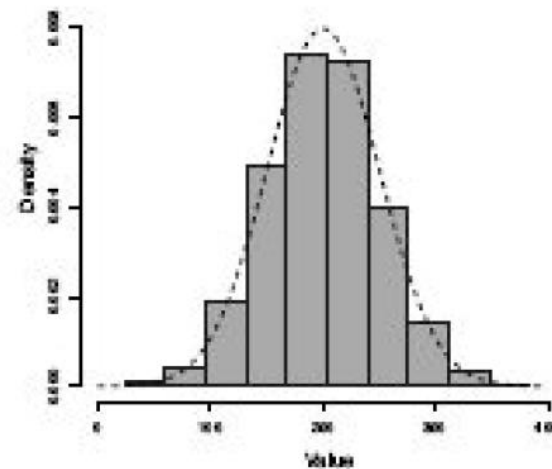
- Equal-width binning:

**Good** if the data follows a uniform distribution

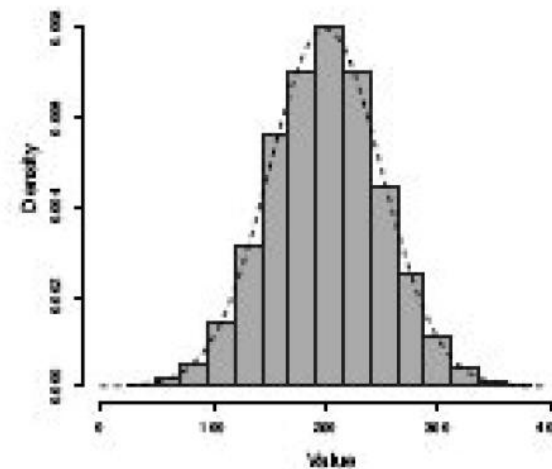
However, for normal distribution, bins at the tail will end up with few instances



(a) 5 equal-width bins



(b) 10 equal-width bins



(c) 15 equal-width bins

# Discretization (Binning)

**cut in pandas implements the equal-width binning**

```
pd.cut(iris['sepal_length'], bins=4) #Four bins for the sepal_length attribute
```

```
0    (4.296, 5.2]
1    (4.296, 5.2]
2    (4.296, 5.2]
3    (4.296, 5.2]
4    (4.296, 5.2]
```

```
...
145   (6.1, 7.0]
146   (6.1, 7.0]
147   (6.1, 7.0]
148   (6.1, 7.0]
149   (5.2, 6.1]
```

Name: sepal\_length, Length: 150, dtype: category

Categories (4, interval[float64, right]): [(4.296, 5.2] < (5.2, 6.1] < (6.1, 7.0] < (7.0, 7.9]]

```
pd.cut(iris['sepal_length'], bins=4, labels=False) #Four bins for the sepal_length attribute
```

```
0    0
1    0
2    0
3    0
4    0
```

```
..
145   2
146   2
147   2
148   2
149   1
```

Name: sepal\_length, Length: 150, dtype: int64

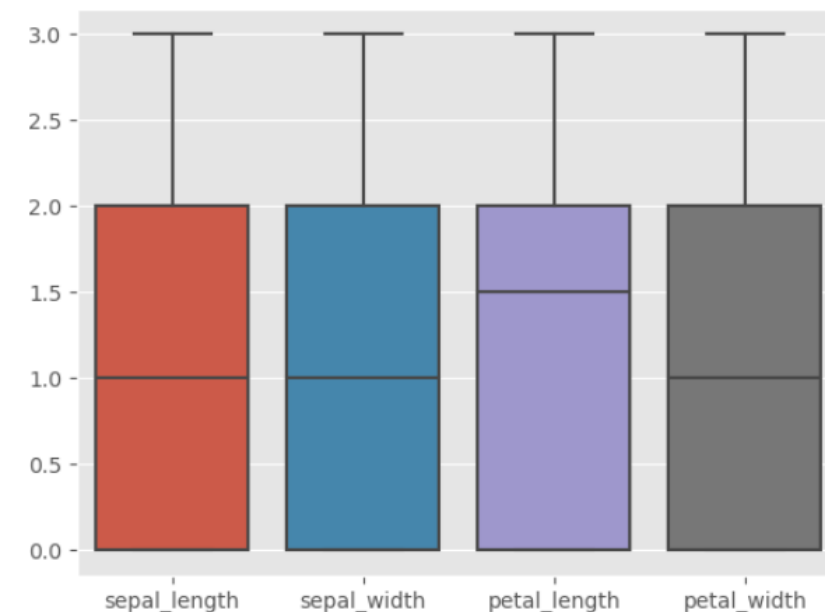
```
irisCopy = iris.copy()
for c in iris.columns[:-1]:
    irisCopy[c] = pd.cut(iris[c], bins=4, labels=False) #four bins

print(irisCopy)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	0	2	0	0	setosa
1	0	1	0	0	setosa
2	0	1	0	0	setosa
3	0	1	0	0	setosa
4	0	2	0	0	setosa
..	...	...	...	...	...
145	2	1	2	3	virginica
146	2	0	2	2	virginica
147	2	1	2	3	virginica
148	2	2	2	3	virginica
149	1	1	2	2	virginica

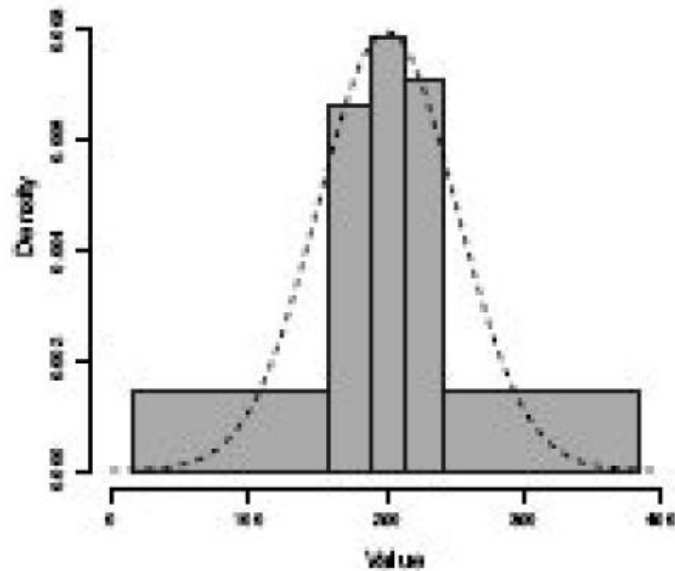
[150 rows x 5 columns]

```
sns.boxplot(data=irisCopy)
plt.show()
```

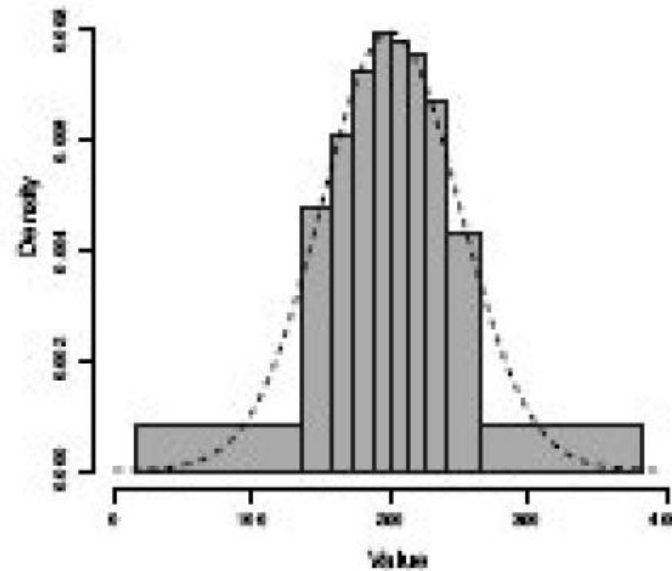


# Discretization (Binning)

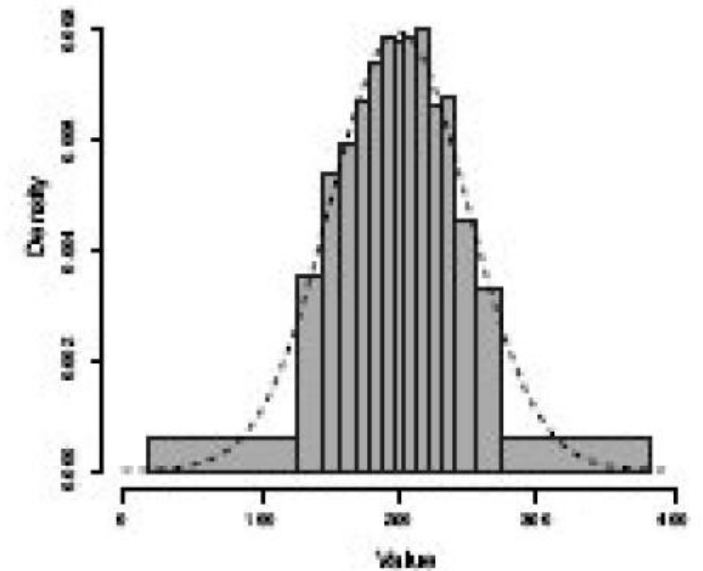
- Equal-frequency binning



(d) 5 equal-frequency bins



(e) 10 equal-frequency bins



(f) 15 equal-frequency bins

# Discretization (Binning)

gcut in pandas implements the equal frequency binning

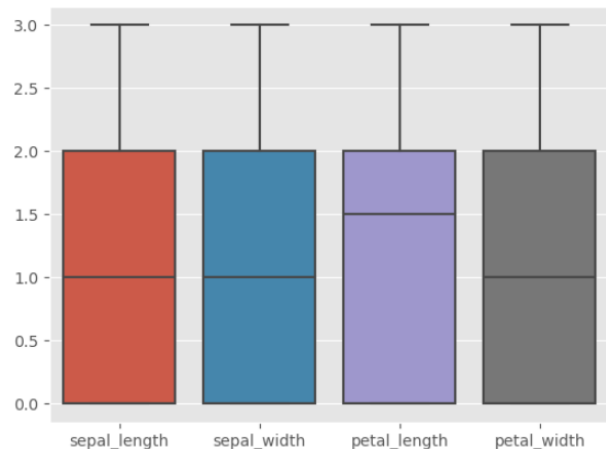
```
irisCopy = iris.copy()
irisCopy['sepal_length'] = pd.qcut(iris['sepal_length'], q=4, labels=False) #Four bins for the sepal_length attribute
irisCopy.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	0	3.5	1.4	0.2	setosa
1	0	3.0	1.4	0.2	setosa
2	0	3.2	1.3	0.2	setosa
3	0	3.1	1.5	0.2	setosa
4	0	3.6	1.4	0.2	setosa

```
irisCopy = iris.copy()
for c in iris.columns[:-1]:
    irisCopy[c] = pd.qcut(iris[c], q=4, labels=False) #define the quantiles of the ranges

irisCopy.head(10)
```

```
sns.boxplot(data=irisCopy)
plt.show()
```



	sepal_length	sepal_width	petal_length	petal_width	species
0	0	3	0	0	setosa
1	0	1	0	0	setosa
2	0	2	0	0	setosa
3	0	2	0	0	setosa
4	0	3	0	0	setosa
5	1	3	1	1	setosa
6	0	3	0	0	setosa
7	0	3	0	0	setosa
8	0	1	0	0	setosa
9	0	2	0	0	setosa

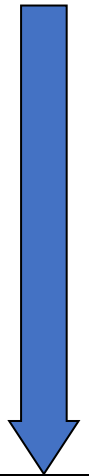
# Data Aggregation

- Summarises low level data details to higher level data abstraction.
  - reduce the time of learning.
  - discover more stable patterns.
  - reduce noise and diminish distortion
- Ex.:
  - By applying aggregate functions (e.g. count, sum, average)

TID	Date	Item	Store	Price	Clubcard#	.....
.....	.....	.....	.....	.....	.....	.....
32144	06/06/2006	milk	Buckingham	1.99	1111	.....
11122	04/04/2006	watch	Buckingham	25.99	1011	.....
11122	04/04/2006	battery	Buckingham	3.99	1011	.....
11123	04/04/2006	beer	Buckingham	9.99	1022	.....
22244	04/04/2006	beer	MK	6.99	1022	.....
22244	04/04/2006	nappies	MK	10.89	1022	.....
23311	05/04/2006	beer	MK	6.99	1011	.....
.....	.....	.....	.....	.....	.....	.....



Date	Store	AveragePrice	.....
.....	.....	.....	.....
06/06/2006	Buckingham	1.99	.....
04/04/2006	Buckingham	13.32	.....
04/04/2006	MK	8.94	.....
05/04/2006	MK	6.99	.....
.....	.....	.....	.....



Number of Items	TotalPrice	Clubcard#	.....
.....	.....	.....	.....
1	1.99	1111	.....
3	36.97	1011	.....
2	27.87	1022	.....
.....	.....	.....	.....

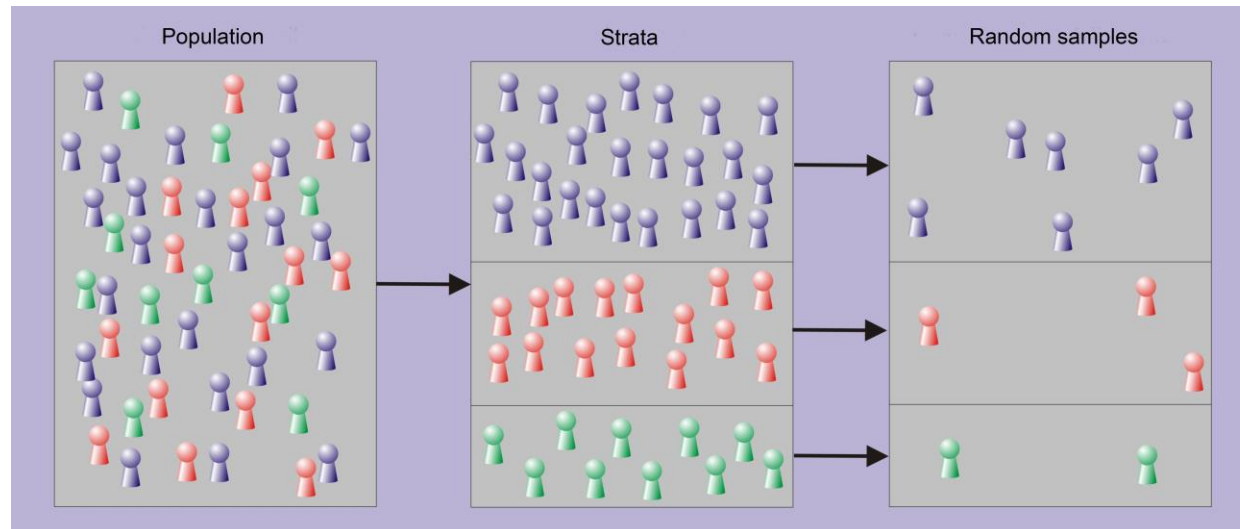
# Feature extraction

- Create new features from functions of the original features
- As with ML schemes, there are many different feature extraction techniques and there are no simple recipes.
  - Extract new features from the existing ones, e.g. extracting color, texture and shape from image of pixel values
  - Mapping data to a new space, e.g. wavelet transformation from time domain to frequency domain
- Deep learners automatically extract features
  - e.g., CNNs can extract the relevant areas in an image by themselves without any image pre-processing.



# Sampling

- Analyzing the whole data set is often too expensive.
- Data sampling reduces the number of instances to a smaller (representative) subset.
  - Top % sampling: not recommended – can introduce bias
  - Random sampling with/out replacement – maintains distributions, but could omit or underrepresent instances of features with small proportion of instances
  - Stratified sampling -- same relative frequencies are maintained
  - Over/down sampling – different relative frequencies



# Sampling

```
# Creating the dataframe
index = ['Person 1', 'Person 2', 'Person 3', 'Person 4', 'Person 5', 'Person 6', 'Person 7', 'Person 8', 'Person 9', 'Person 10']
df = pd.DataFrame({"Age": [20, 65, 25, 22, 55, 70, 40, 60, 80, 77],
                  "Salary": [25000, 25500, 25700, 26900, 25400, 55000, 39700, 35900, 32400, 60000],
                  "Class": ["No", "Yes", "Yes", "Yes", "No", "No", "Yes", "Yes", "Yes", "Yes"]},
                  index=index)
```

```
# Print the dataframe
df
```

	Age	Salary	Class
Person 1	20	25000	No
Person 2	65	25500	Yes
Person 3	25	25700	Yes
Person 4	22	26900	Yes
Person 5	55	25400	No
Person 6	70	55000	No
Person 7	40	39700	Yes
Person 8	60	35900	Yes
Person 9	80	32400	Yes
Person 10	77	60000	Yes

```
df_sample = df.sample(n=4)
df_sample
```

	Age	Salary	Class
Person 6	70	55000	No
Person 4	22	26900	Yes
Person 10	77	60000	Yes
Person 7	40	39700	Yes

```
df_sample = df.sample(n=4)
df_sample
```

	Age	Salary	Class
Person 3	25	25700	Yes
Person 2	65	25500	Yes
Person 9	80	32400	Yes
Person 8	60	35900	Yes

```
#Sample 2 rows from each class
```

```
df_sample = df.groupby('Class', group_keys=False).apply(lambda x: x.sample(2))
df_sample
```

	Age	Salary	Class
Person 6	70	55000	No
Person 5	55	25400	No

```
#Sample 60% for each class
```

```
df_sample = df.groupby('Class', group_keys=False).apply(lambda x: x.sample(frac=.6))
df_sample
```

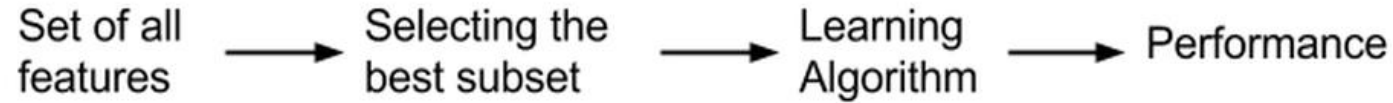
	Age	Salary	Class
Person 1	20	25000	No
Person 5	55	25400	No
Person 4	22	26900	Yes
Person 9	80	32400	Yes
Person 8	60	35900	Yes
Person 2	65	25500	Yes

# Feature selection

- Some features are *redundant* or *irrelevant*.
- Many features and few samples (or data points), e.g., texts and DNA data.
- Use subset(s) of useful/relevant features for the subsequent ML tasks.
  - simplify ML models; make them easier to interpret
  - shorter training times
  - avoid the curse of dimensionality
  - enhance performance
- Approaches:
  - Manually: use common sense and domain knowledge.
  - Brute-force: try all possible feature subsets as input to ML algorithm. Expensive!
  - Filter
  - Wrapper
  - Embedded

# Feature selection

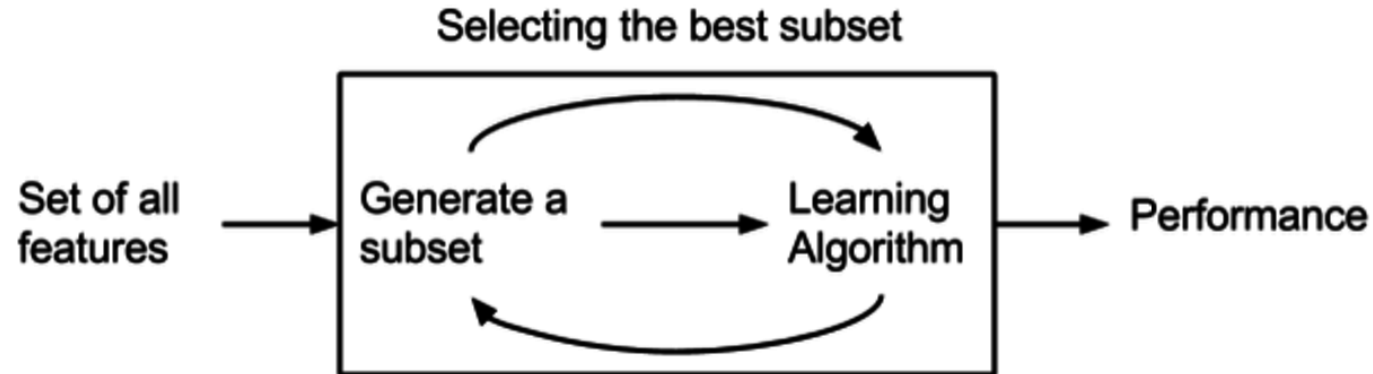
- **Filter:** select variables regardless of the model.



- Ex.: based on correlations with the target variable to predict.
- Effective in time and robust.
- Univariate: select redundant variables when they do not consider the relationships between variables.
  - Multi-variate: more elaborate methods try to minimize this problem by removing variables highly correlated to each other.
- many filters provide a feature ranking rather than an explicit best feature subset
- Underlying methods:
  - information gain
  - chi-square test
  - fisher score
  - correlation coefficient
  - variance threshold
  - ...

# Feature selection

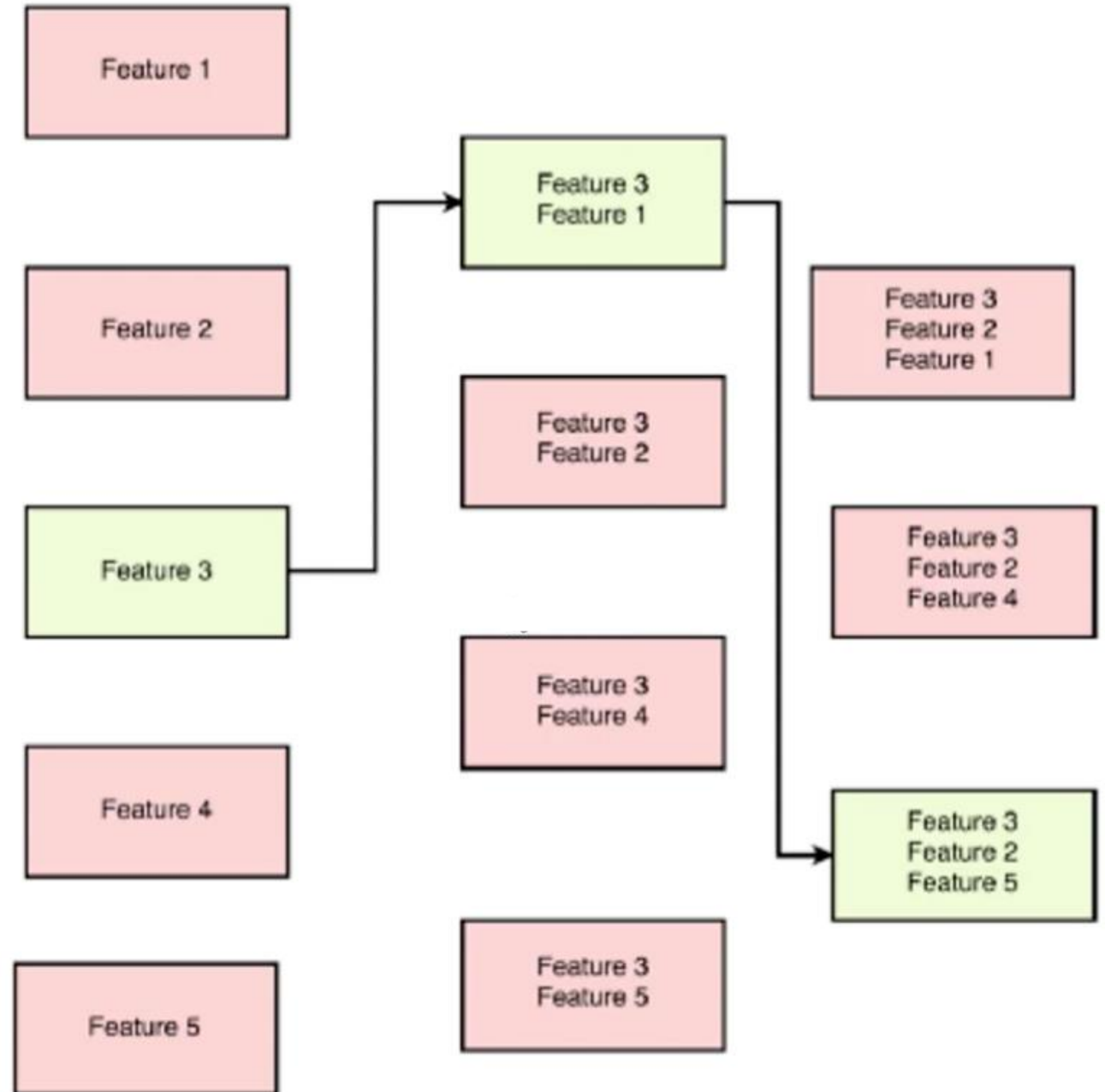
- **Wrapper:** iteratively select or eliminate a set of features using the model performance.



- detect the possible interactions between variables.
  - find the optimal feature subset for the desired ML algorithm
  - significant computation time when the number of variables is large
- Underlying methods:
    - forward selection
    - backward elimination
    - recursive feature elimination
    - genetic algorithms

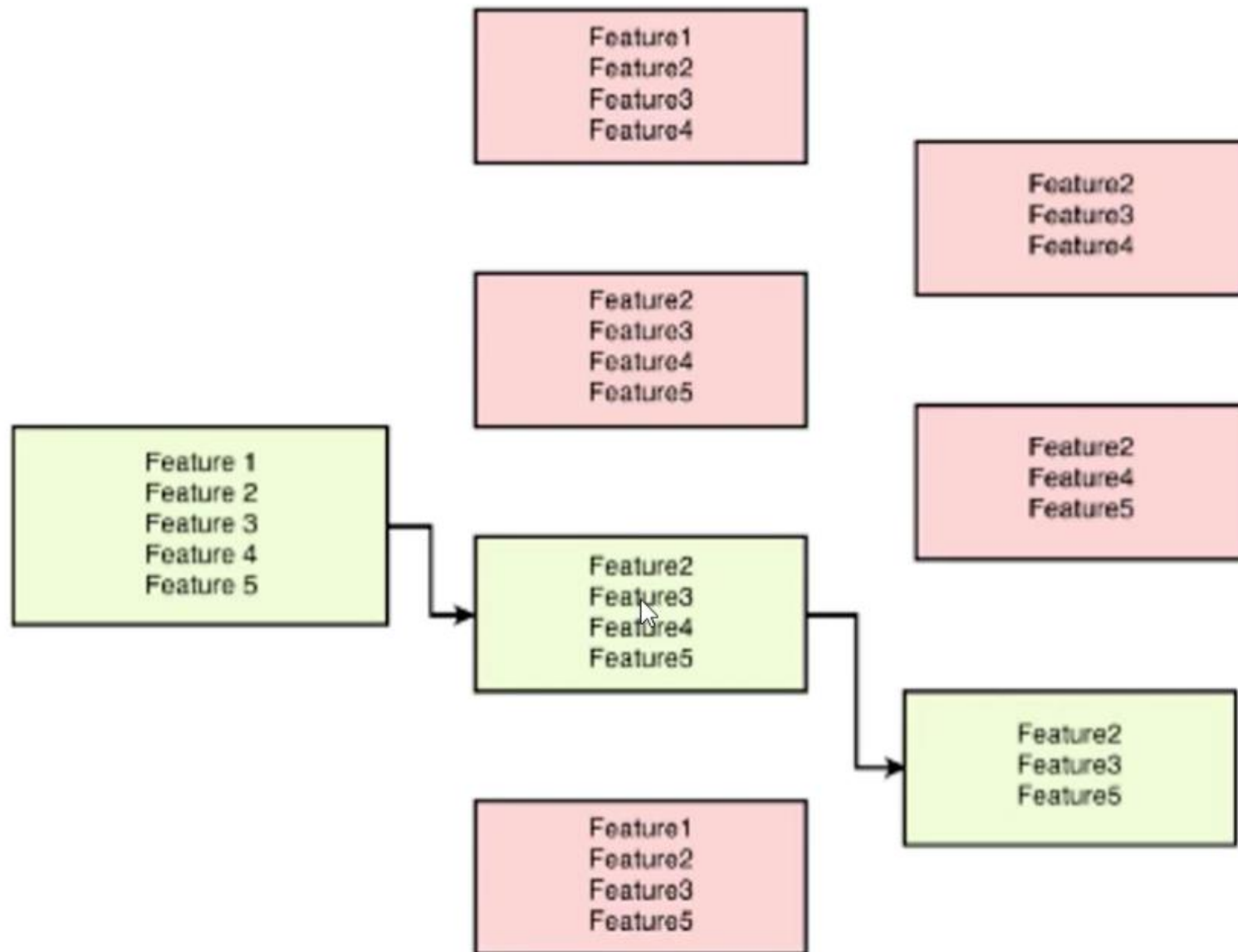
# Feature selection

- Wrapper:
  - Forward selection:



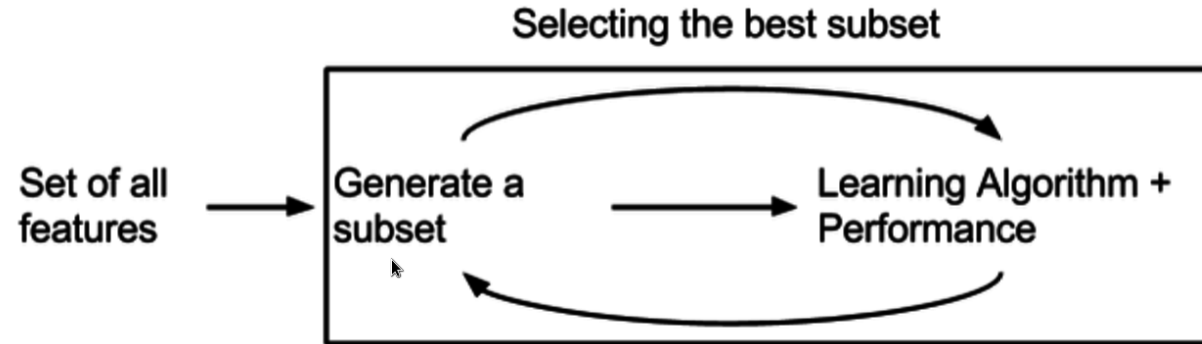
# Feature selection

- Wrapper:
  - Backward elimination



# Feature selection

- **Embedded:** perform feature selection as part of the model construction process.

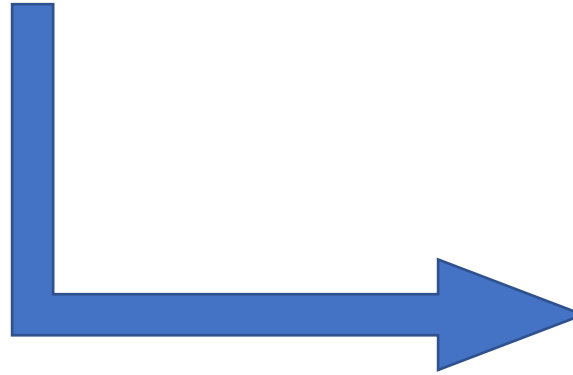
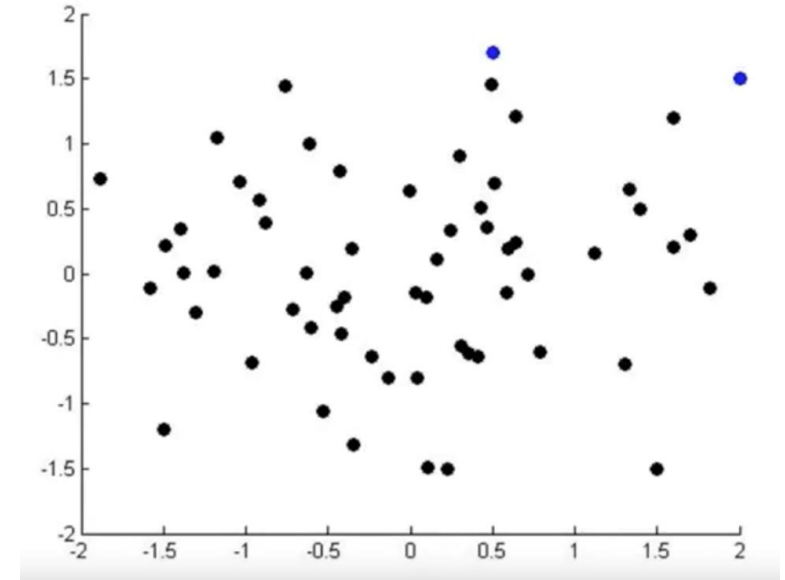


- take into consideration the in
  - fast like filter methods.
  - more accurate than filter methods.
  - better model performance
- 
- Ex.:
    - Decision trees
    - L1 (LASSO)-regularization
    - ...



# Dimensinality Reduction

Country	GDP (trillions of US\$)	Per capita GDP (thousands of intl. \$)	Human Develop- ment Index	Life expectancy	Poverty Index (Gini as percentage)	Mean household income (thousands of US\$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	...



Country	$z_1$	$z_2$
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

# Dimensionality Reduction

- Observable features reduced into a smaller set of indirectly observable features, a.k.a., latent variables or hidden variables.
- Transform data from a high-dimensional space into a low-dimensional space and retains properties of the original data
  - raw data are often sparse as a consequence of the curse of dimensionality
  - data compression → better efficiency for the learning algorithms.
  - Enable data visualization
- Underlying methods:
  - Principal component analysis (PCA)
  - Linear discriminant analysis (LDA)
  - Autoencoder
  - ...

# Summary

