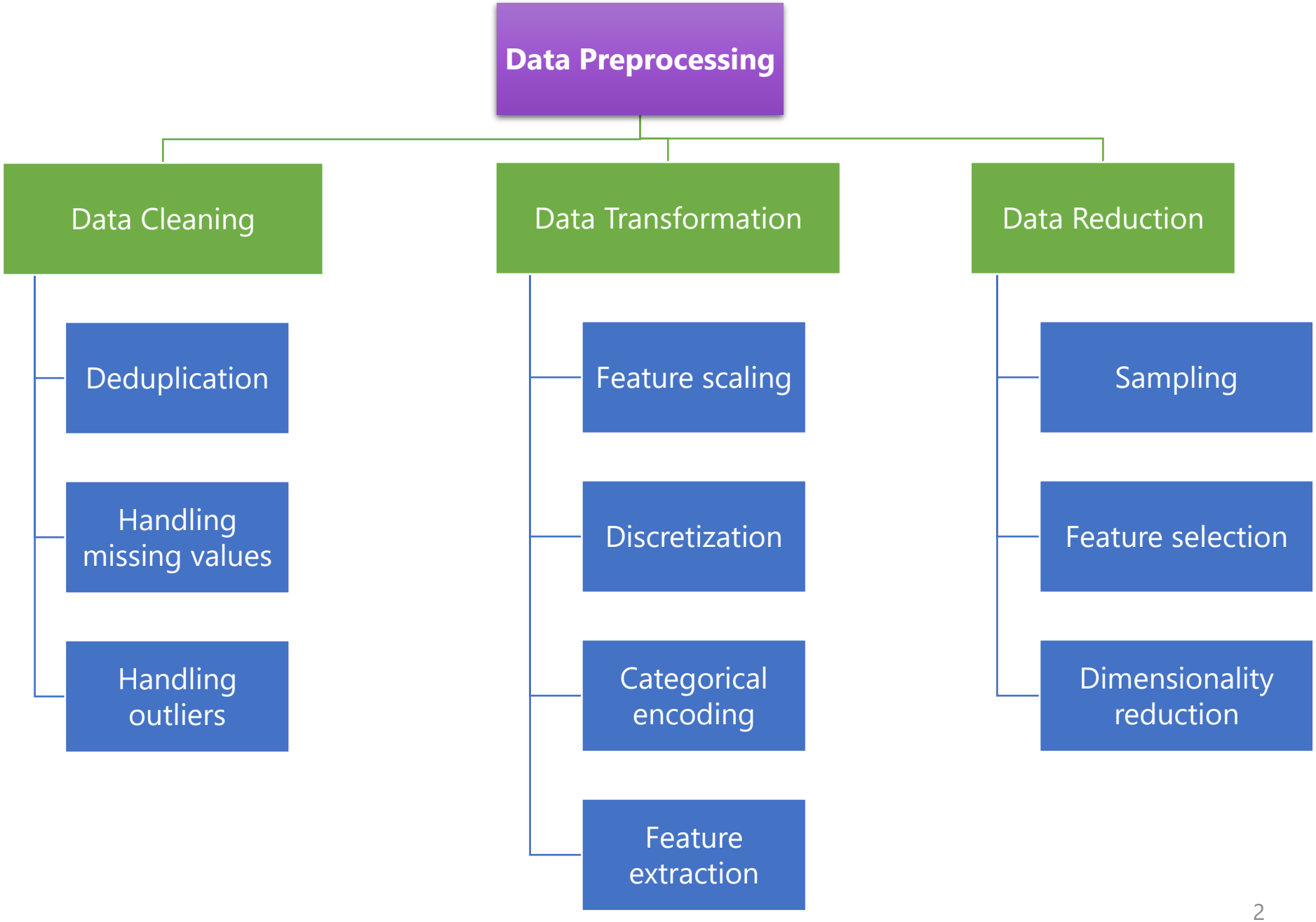
Data Preprocessing (DP)

Outline


Data Preprocessing
Data Cleaning
Deduplication
Handling missing values
Handling outliers
Data Transformation
Feature scaling
Discretization
Categorical encoding
Feature extraction
Data Reduction
Sampling
Feature selection
Dimensionality reduction

Data Quality Issues

Data Quality Issues

- **Missing values**: (Null values) Not measured or Not available, e.g. people decline to disclose their age and weight, and annual income is not applicable to children.

- **Outliers:** Data points that differ significantly from most observations in the dataset
  - E.g., A few extremely high-income values in a household income dataset

- **Irregular values:** Inconsistent or unexpected representations of the same concept, often due to heterogeneous data sources, e.g., Gender recorded as 0/1, M/F, and Male/Female across datasets, one rating "1,2,3", another rating "A, B, C"

- **Bias:** Systematic error in data collection, measurement, or sampling causing the data to consistently deviate from the true population
  - It might favor certain groups or outcomes, leading to inaccurate, misleading, or unfair results
  - E.g., A job-satisfaction survey **collects responses mainly from urban employees**, underrepresenting rural workers and skewing results, leading to biased or unfair conclusions

- **Noise:** Random errors or fluctuations in data that distort values without a consistent pattern
  - Due to errors in data entry / transmission / computation
  - E.g., distortion of a person's voice when talking on a poor phone, random responses due to unclear survey questions

- **Low precision:** measurements vary widely or contain errors making the data less reliable
  - Can be caused by measurement instruments, data collection processes, or human error
  - E.g., A thermometer repeatedly reports temperatures with ±1.5 ° C fluctuation due to calibration issues

How to deal with Data Quality issues?

Issue	Handling Techniques
Missing Values	Remove rows/features with excessive missingness; impute values; add missing-value indicator features
Outliers	Detect outliers using simple statistics (IQR, z-score) or visual inspection; cap extreme values or treat them as missing; keep them if they represent meaningful rare cases.
Irregular Values	Standardize formats and value mappings; normalize during data integration
Bias	Use representative or stratified sampling Apply re-weighting or fairness-aware methods
Noise	Apply data cleaning and validation rules; use smoothing or filtering techniques; adopt robust ML models
Precision	Calibrate measurement instruments; improve data collection procedures; aggregate or smooth repeated measurements

Why Data Quality is Important?

- High-quality data is **accurate**, **correct**, **complete**, **consistent**, and **free of unnecessary duplication**
- Poor data quality leads to poor results — "Garbage in, garbage out"
- For most ML projects, *fixing data quality issues at the source is not always feasible*. **Practical workarounds include:**
  - Cleaning the data as much as possible
  - Designing and using robust ML models that tolerate noise, missing values, and imperfections
- Data quality should always be evaluated relative to the project's objective

e.g. Spelling errors in student names are irrelevant when predicting GPA

Data Preprocessing (DP)

- Real-world data is rarely in a form directly suitable for ML algorithms
  - It is often noisy, incomplete, redundant, irrelevant, or high-dimensional
- DP objectives:
  - Transform raw data into suitable representations (**feature vectors**)
  - Select or extract relevant features
  - Improve computational efficiency of ML algorithms
  - Enable better model performance and reliability
- DP is highly application-specific, thus needs domain knowledge
- DP main tasks:
  - Data cleaning, Transformation and Reduction

Data Deduplication

03.dp**01-deduplication**

Data Deduplication

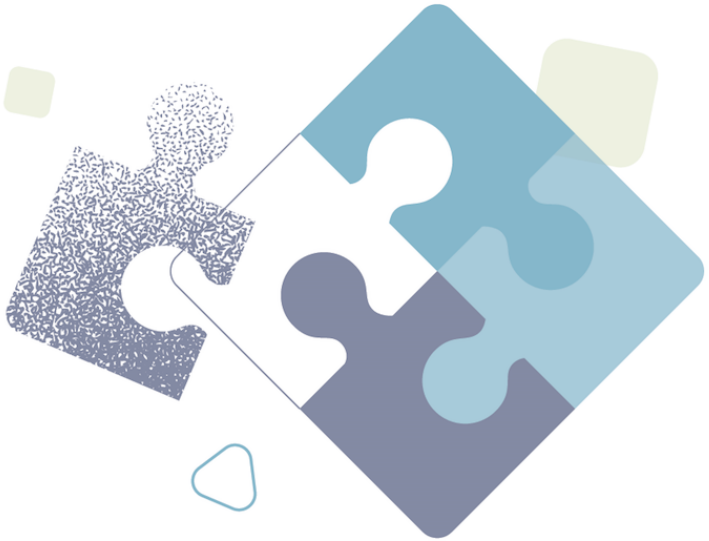
- **Duplicates:** instances with identical feature values, representing the same real-world entity or event

- Duplicate data can **bias ML models**, as repeated instances receive more influence and may distort learning outcomes. E.g.,
  - The same person submits an online form multiple times
  - Retweets share the same content as the original tweet, differing only in metadata (e.g., user, timestamp)

- Duplicate detection techniques:
  - **Exact matching**: direct comparison of records
  - **Similarity-based methods**: clustering or distance metrics to identify near-duplicates

```
# Deduplicate based on all columns except 'order_id'
df_deduped = df.drop_duplicates(subset=df.columns.difference(['order_id']))
```

Handling Missing Values



03.dp\02-handling-missing-values

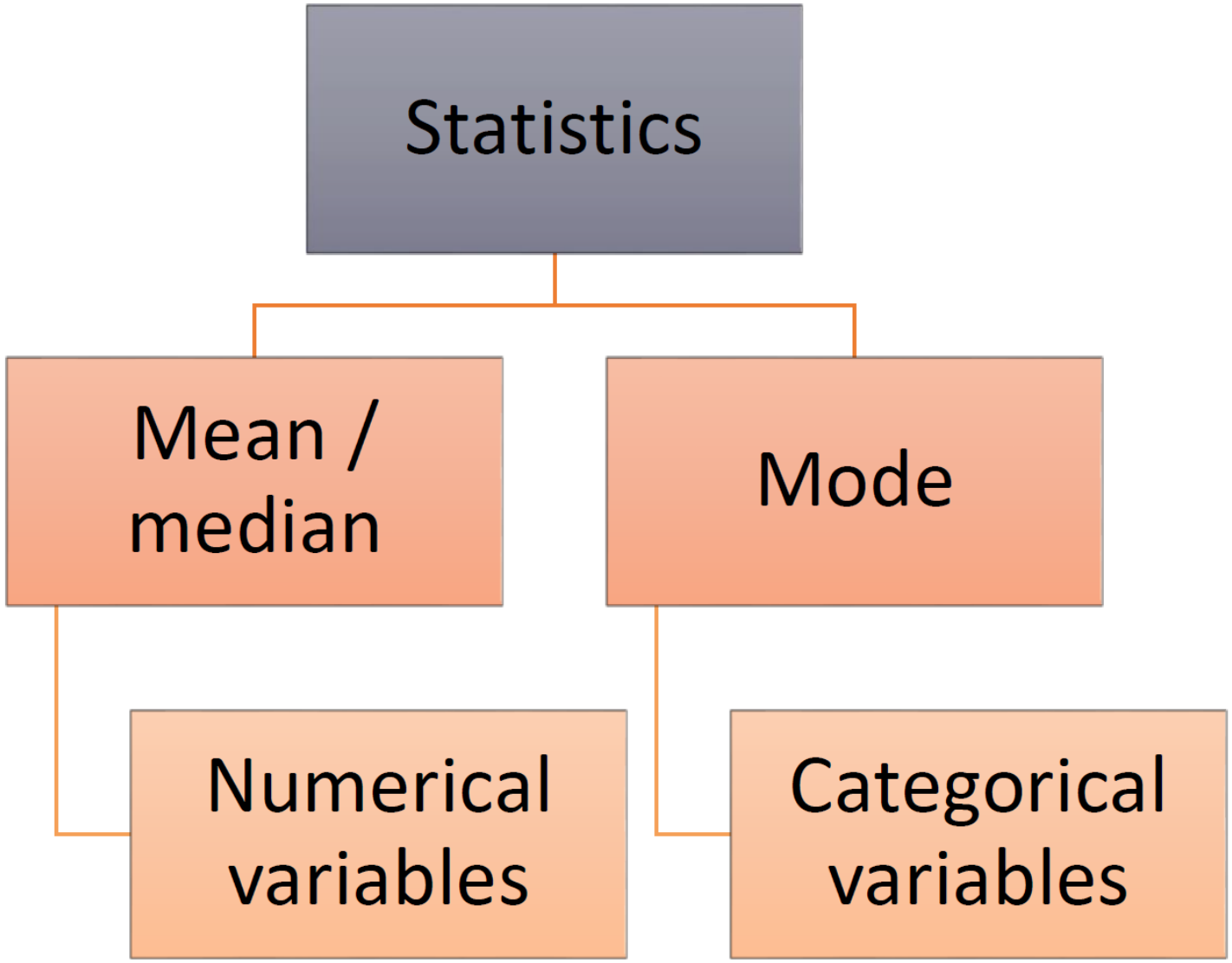
Handling missing values

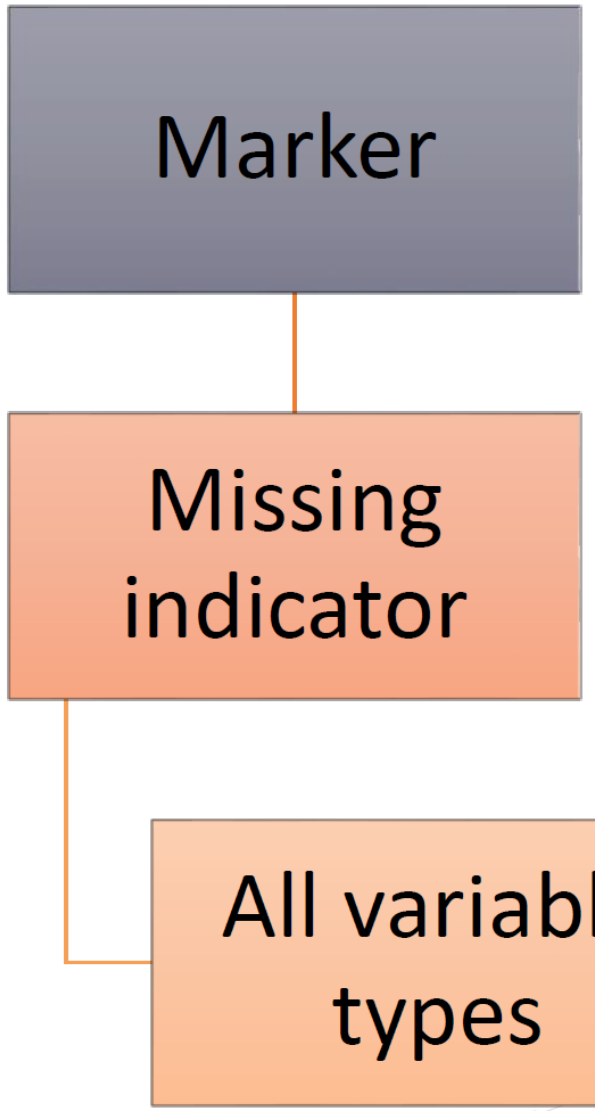
- **Remove** features or instances with excessive missingness (e.g., more than 60%)
- **Add a missing-value indicator** (flag) to explicitly mark missing data
- **Impute values** using simple statistics: Mean or median for numerical features & Mode for categorical features
- **Model-based imputation**: train an ML model to predict missing values from 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

https://gallery.azure.ai/Experiment/Methods-for-handling-missing-values-1

Handling missing values





Mean / Median imputation

- For numerical features, use **Mean** or **Median** imputation
  - If the data is **approximately normally distributed**, mean and median are similar
  - If the data is **skewed**, the **median** is preferred
- For categorical features, use **mode** imputation
- Assumes data is missing at random (MAR)
  - Missingness **depends on other features**, not on the missing value (e.g., income missing more often for younger people)

Price
100
90
50
40
20
100
60
120
200

Mean = 86.66

Median = 90

Price
100
90
50
40
20
100
86.66
60
120
86.66
200

Limitations of Mean / Median Imputation

- Distortion of the original variable distribution
  - Distortion of the original variance
  - Distortion of the covariance with the remaining features of the dataset
- The higher the percentage of missing values, the higher the distortions
- Hence limit its usage to when:
  - Data is missing completely at random
  - No more than 5% missing data per variable

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		beef		Jane
Henry	69	21	95	6				
Janet	62	21	110	3				
Nick	46	17	250	4				
Bruce	37	14	63					
Steve	83	12	77	7				
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

Values are **Missing Not at Random (MNAR)** but missing because those with very high incomes preferred not to state them.
In this case, we can make a reasonable guess for what "high" means and fill in the blanks

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	37	14	63	12				
Steve	83	12	77	7				
Clint	27	9	118	9				
Wanda	19	7	52	2				
Natasha	26	4	162	5	3			_
Carol	46	3	127	11	1	veggie	1	""""
Mandy	44	2	68	8	1	chicken		null

Missing Rank Order: replace missing values with a missing rank. Our knowledge of how parking spaces are numbered let us make a guess here.

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						Henry
Nick	46	17						
Bruce	37	14						NA
Steve	83	12						n/a
Clint	27	9					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

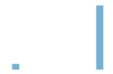
Interpolate: replace missing value with the average of the previous and next values

e.g.,

```
df['years_seniority'].interpolate(method='linear'))
```

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact	parking_space_IsMissing
Tony	48	27	250	1					false
Donald	67	25	86	10					false
Henry	69	21	95	6					false
Janet	62	21	110	3					false
Nick	46	17	250	4					false
Bruce	37	14	63	-99	1	veggie		NA	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

Replace missing values with a **Dummy Value** and create a **Missing Indicator variable** to flag missing data

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					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

Replace missing values with **0** and create a **Missing Indicator variable**

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						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

multiple representations for "missing"

For categorical data, missing values can be **replaced with a string** communicating that they are missing

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				beef		Henry	false	false
Nick	46	17						r		
Bruce	37	14				veggie				
Steve	83	12				chicken		r		
Clint	27	9	118	9	0	shrimp	3	N		
Wanda	19	7	52	2	2	shrimp		e		
Natasha	26	4	162	5	3			_		
Carol	46	3	127	11	1	veggie	1	""""	false	false
Mandy	44	2	68	8	1	chicken		null	false	false

drop mostly empty columns

Drop mostly empty columns that are missing too many values to be useful.

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)
					5	shrimp	Pepper	false	false	present
					2	beef	Jane	false	false	present
					1	chicken	Janet	false	false	present
Janet	62	21	110	3	1	beef	Henry	false	false	present
Nick	46	17	250	4	0		no	false	true	absent
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

Remove features/instances

Remove features/instances that are excessively missing their values

```
# 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

Impute with mean, median or mode

Impute with mean, median or mode of 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

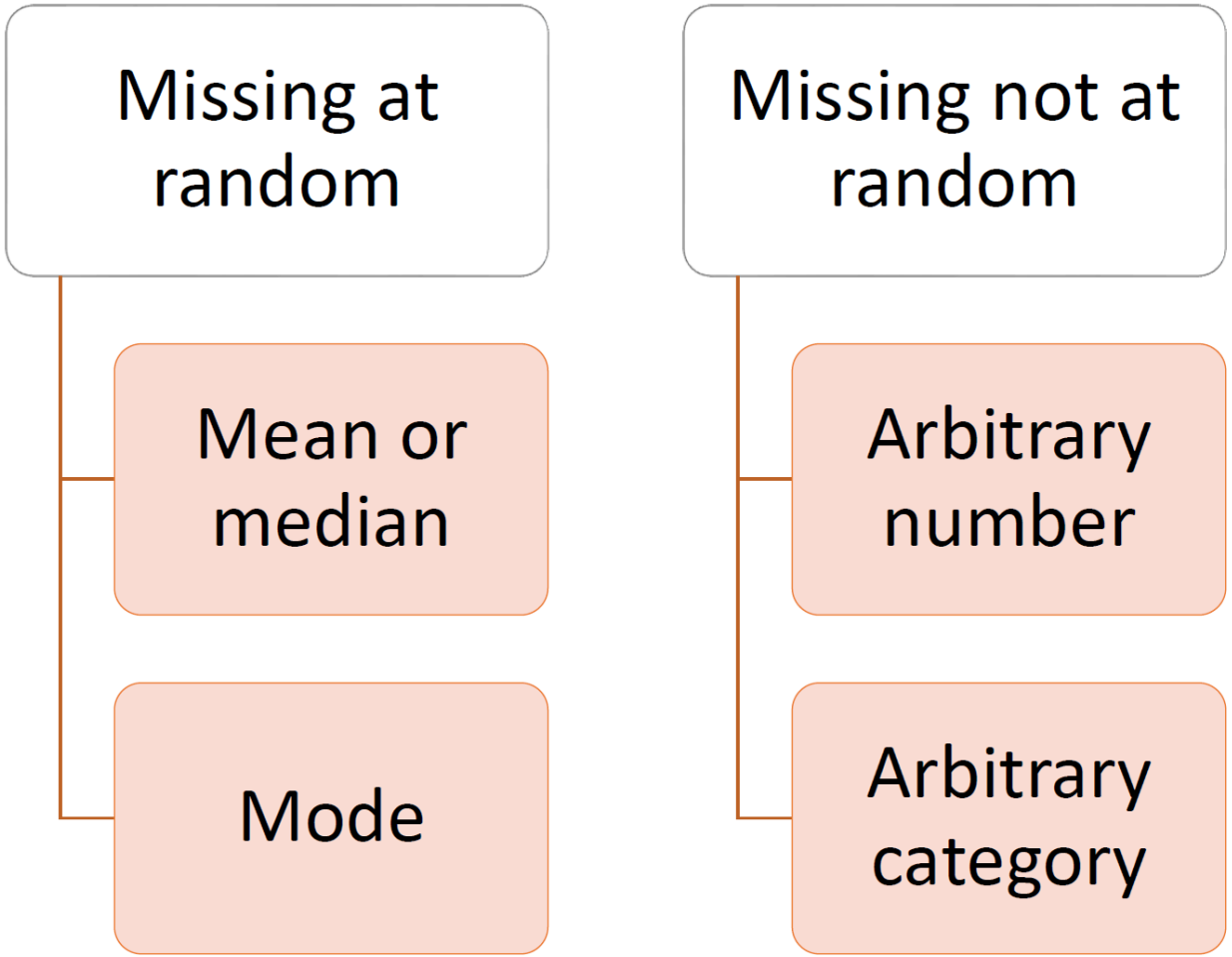
```
df.mean()  #It i
```

```
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
```

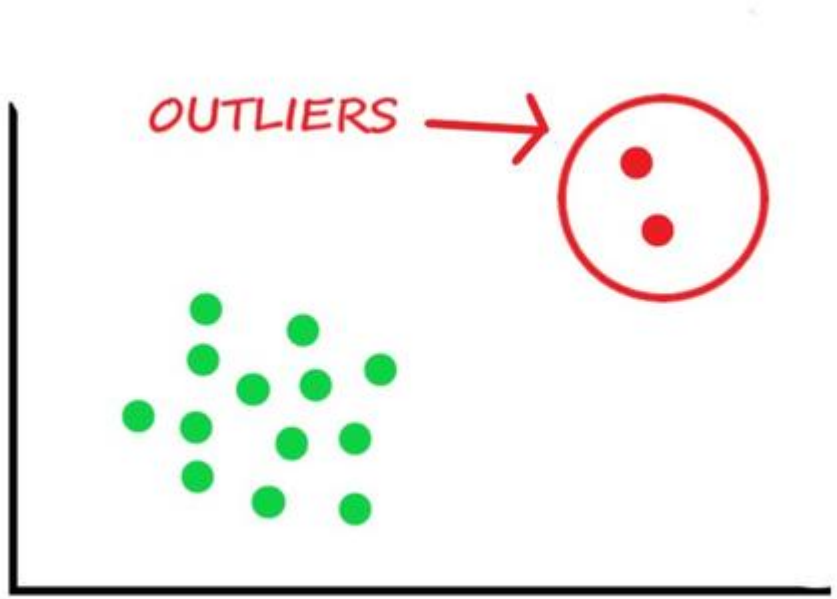
	A	B	C	D	E	F
0	12.000000	2.25	10	14.000000	20.00	10
1	4.666667	2.00	3	8.666667	9.75	3
2	1.000000	2.00	6	6.000000	8.00	6
3	4.666667	3.00	5	8.666667	3.00	5
4	1.000000	2.00	6	6.000000	8.00	6

Summary



Adding a **Missing Indicator** variable to flag missing data is a good imputation strategy

Handling Outliers

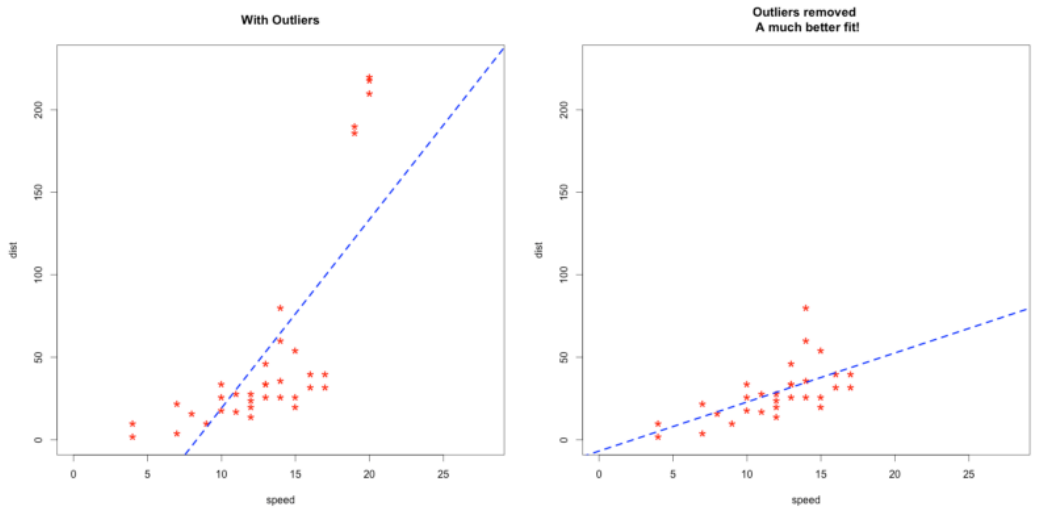


03.dp\3-handling-outliers

Handling Outliers

- Some ML techniques, e.g., logistic regression, are sensitive to outliers
- How to handle outliers is application-dependent, since they may:
  - Represent meaningful rare cases
  - Be noise caused by measurement or data entry errors
- In large datasets, a small number of outliers is expected and often not problematic
- Outlier detection methods:
  - **Visual**: boxplots, scatter plots, histograms
  - **Statistical**: IQR, mean ± standard deviation
  - **Algorithmic**: clustering-based approaches

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



Ways to handle outliers

- **Trimming (Removal):** Remove outliers from the dataset
  - ✅ Fast and simple but ⚠️ may remove big chunk of data and lose information
- **Capping** (Winsorization): Replace extreme values with predefined threshold values instead of removing them (see next slide)
- **Treat outliers as missing values** and perform missing data imputation
  - ✅ No data is removed ⚠️ May distort the original distribution
- **Discretization**: Group values into bins. Place extreme values into lower or upper bins
  - ✅ Reduces outlier impact & simplifies feature representation

Ways to handle outliers - Capping

- **Capping (Winsorization):** Replace extreme values with predefined threshold values instead of removing them
  - **For skewed data (e.g., person's height) use IQR-based method:**

  Lower threshold = **Q1 – 1.5 × IQR**

  Upper threshold = **Q3 + 1.5 × IQR**
  - **For approximately normal data(e.g., income, house prices) use z-score (mean ± 3σ):**

    Lower threshold = **μ – 3σ**

    Upper threshold = **μ + 3σ**
  - Values below the lower threshold are set to the lower threshold
  - Values above the upper threshold are set to the upper threshold
  - ✅ No data is removed
  - ⚠️ May distort the original distribution

Feature Scaling

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

03.dp\04-feature-scaling

Feature Magnitude matters

- Continuous features that cover very different ranges (i.e., Magnitude) can cause difficulty for some ML algorithms that are sensitive to feature magnitude
  - 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: $\boldsymbol{distance(x, y) = \sum_{i=1}^{n} |x_i - y_i|}$

  |x_Age - y_Age| + |x_Salary - y_Salary|

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

Feature scaling

- Feature scaling is the process of transforming feature values measured on different scales to **a common scale**
  - It is typically the final step in the data preprocessing pipeline, applied just before training the ML model
  - Feature scaling does not change the shape of the data distribution
- Why Feature Scaling is important:
  - Prevents features with large numeric ranges from dominating the model
  - Makes features comparable
  - Improves performance of distance-based algorithms (e.g., KNN, k-means, PCA)
  - Speeds up convergence of gradient-based optimization(e.g., neural networks, Support Vector Machine (SVMs))

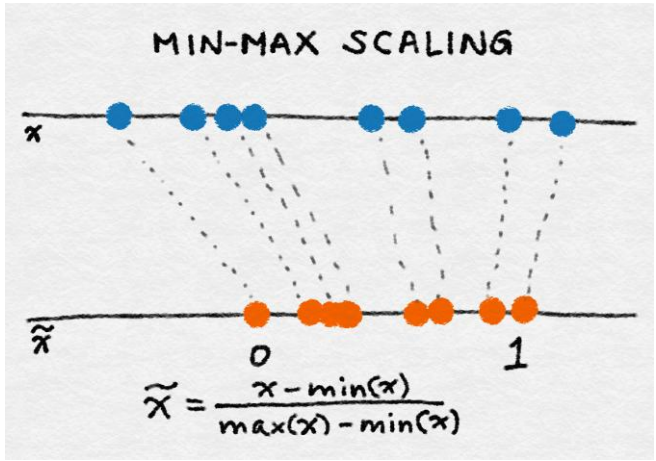
Min-Max Scaling

- **Scales the variable between 0 and 1**

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

x_i' is the scaled value

- Aka Normalization
- Sensitive to outliers



Price
100
90
50
40
20
100
50
60
120
40
200

Max = 200
Min = 20
Range = 200 – 20 = 180

$$\frac{\text{Obs. - Min}}{\text{Range}}$$

Price
0.44
0.39
0.17
0.11
0.00
0.44
0.17
0.22
0.56
0.11
1.00

Image source

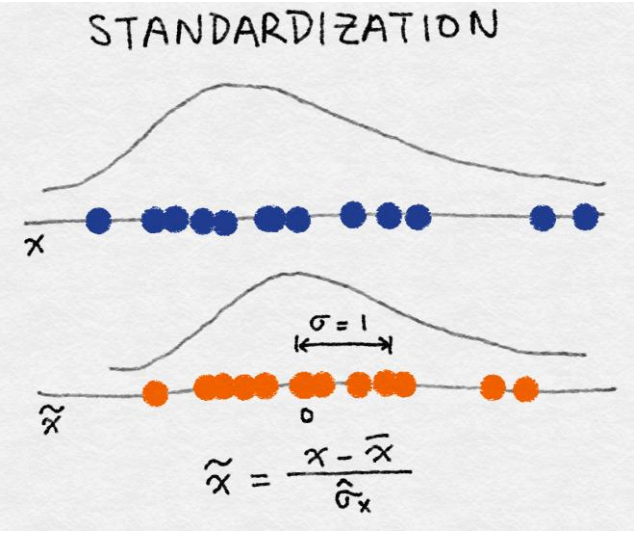
Standardization

- Standardization centers a feature at **zero mean** and scales it to **unit variance**

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

x_i' is the standardized feature value, x_i = original value, μ = mean for feature x, and σ = standard deviation

  - The standardized value indicates how many standard deviations a data point is from the mean
  - Transforms data to have mean = 0 and standard deviation = 1
  - Most values typically fall in the range [−1, 1]
  - Preserves outliers
  - Works best when the data is approximately normally distributed



Price
100
90
50
40
20
100
50
60
120
40
200

Mean = 79
Standard dev = 51

Obs. - Mean

Standard dev

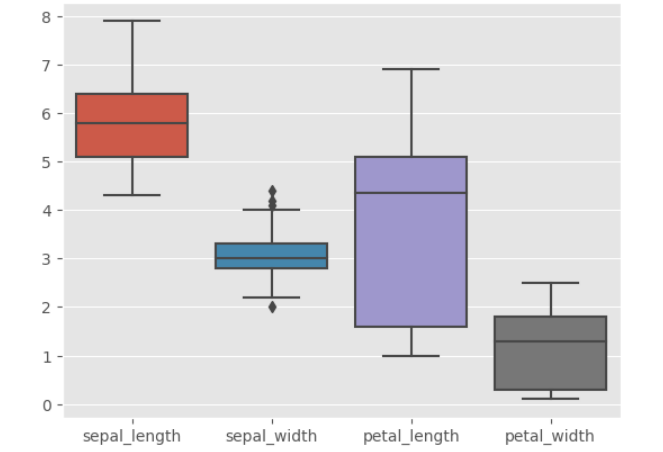
Price
0.41
0.22
-0.57
-0.76
-1.16
0.41
-0.57
-0.37
0.80
-0.76
2.37

Image source

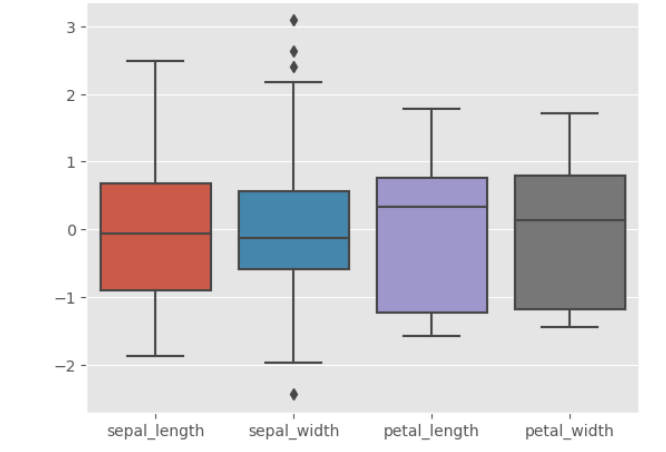
Feature scaling does not change the shape of the distribution

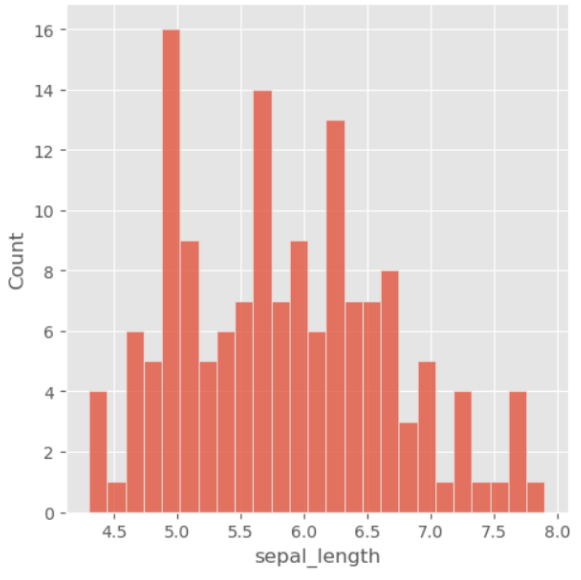
```
iris_scaled = iris.copy()
X = iris_scaled.values[:,0:-1]  #columns, except the class label, of the matrix of the dataframe
X = X.astype(float) #convert X's dtype from object to float
X_scaled = preprocessing.StandardScaler().fit_transform(X)
iris_scaled.iloc[:,0:-1] = X_scaled
iris_scaled.head()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

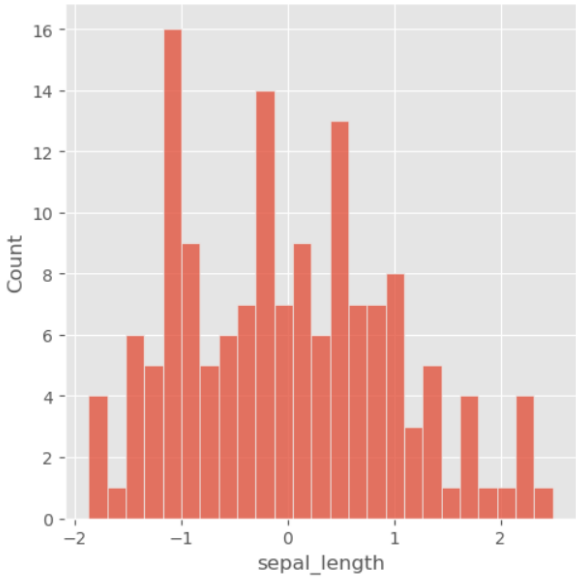
```
sns.boxplot(data=iris)
plt.show()
```



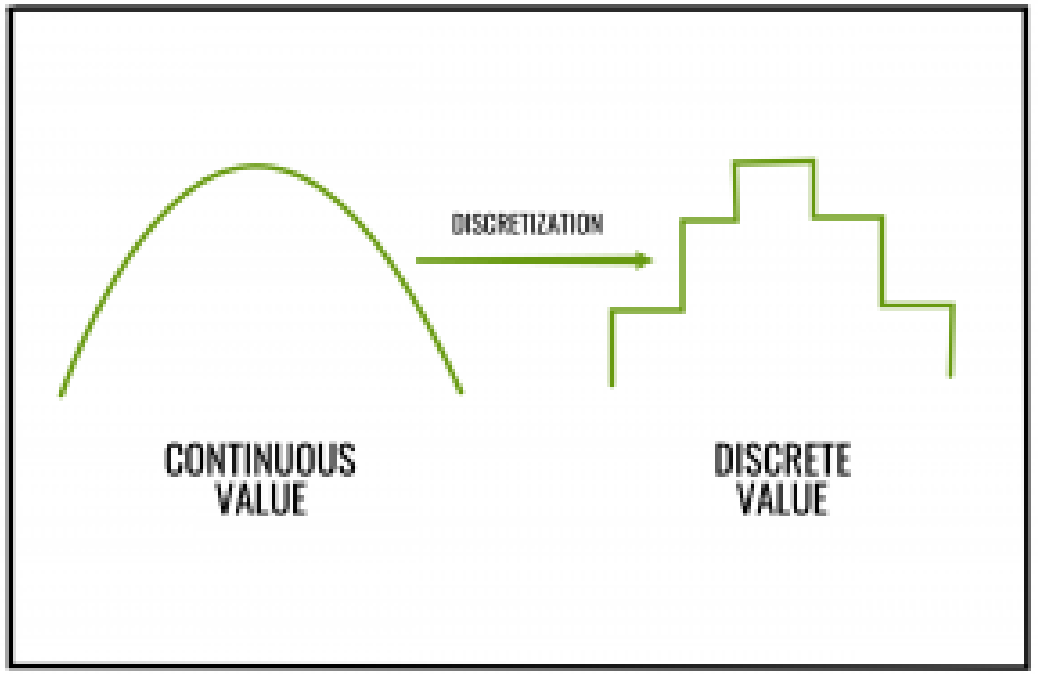

```
sns.boxplot(data=iris_scaled)
plt.show()
```





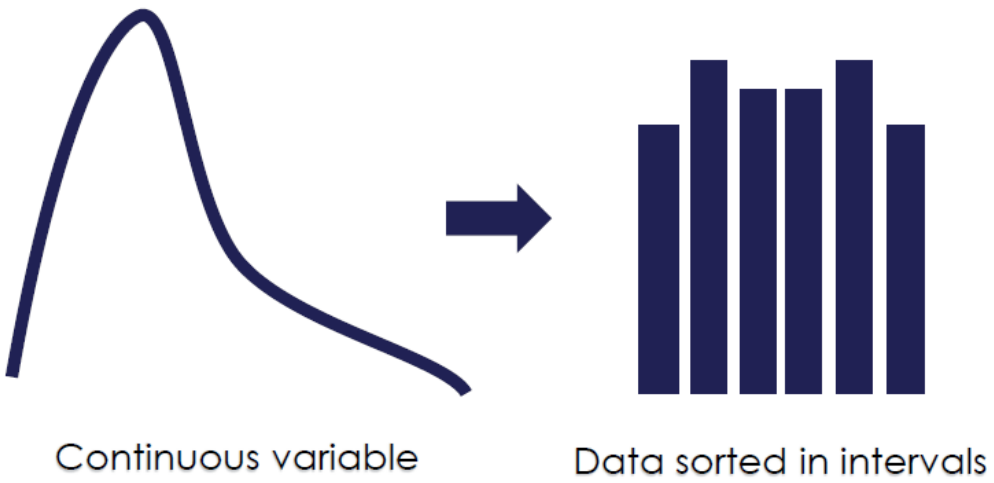


Discretization



03.dp**05-discretization**

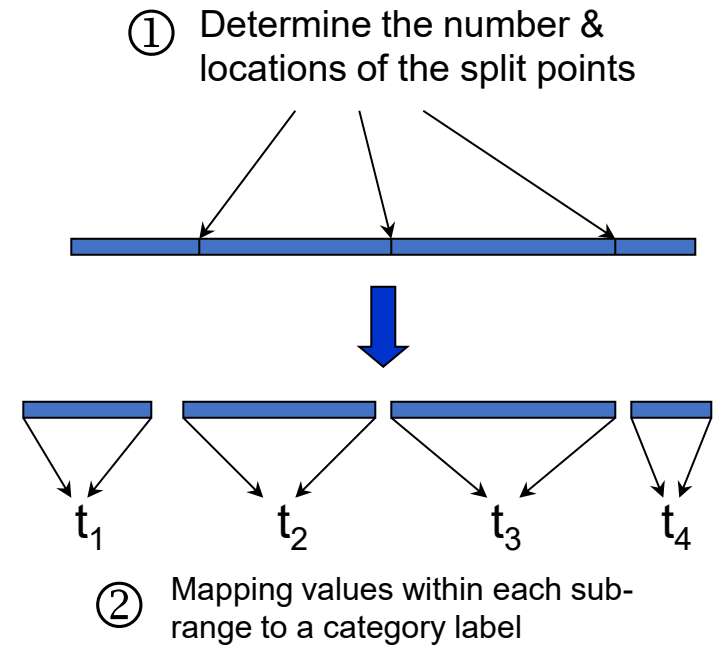
Discretization (Binning)

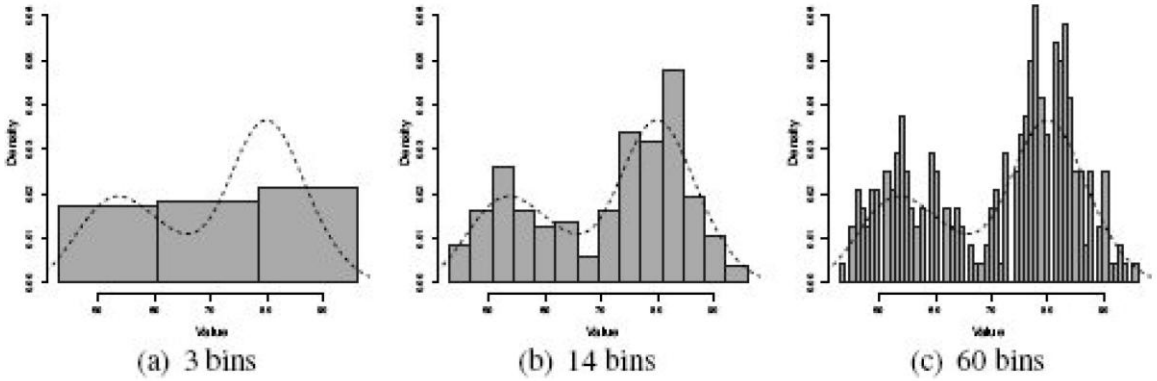


- Discretization (aka Binning) is the process of transforming continuous variables into discrete variables by creating a set of contiguous intervals (aka bins) that span the range of the variable's values
  - Improve performance (e.g., decision trees and Naïve Bayes, work better with discrete attributes)
  - Reduce training time
  - Mitigate the effect of outliers: outliers are placed into the lower or upper intervals
  - Create simpler features (for us humans)

Discretization + encoding

- After discretization, we commonly use the intervals as categories
  - E.g., range of grades is mapped to a letter grade 90 to 100 mapped to A, 85 to 90 mapped to B+, etc.
  - Number of bins? Difficult to determine as there is a trade-off:
    - very low causes loss of information regarding the distribution of values in the original continuous feature
    - as the number of bins grows, we can end up with empty bins





Limitations of discretization

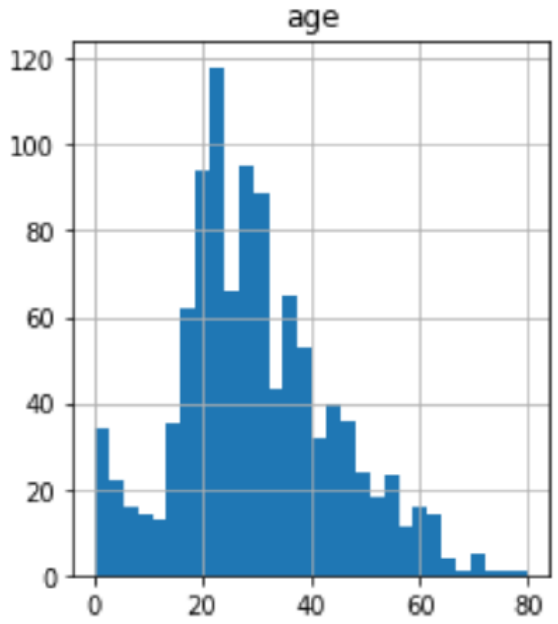
- Discretization can also lead to a loss of information
  - For example, by combining values that are strongly associated with different classes (target values) into the same bin
- The aim of a discretization algorithm is to find the minimal number of intervals without a significant loss of information
  - In practice, many discretization methods require the user to input the number of intervals into which the values will be sorted
  - The job of the algorithm is then to find the cut-points for those intervals

Equal-width Discretization

- Equal width discretization divides the scope of possible values into **N bins** of the same width
- The interval width is determined by:

$$\texttt{width = (max value -min value) / N}$$

  - where N is the number of bins or intervals



Min age = 0
Max age = 73
Intervals = 10

width = (73 - 0) / 10 = 7.3

Intervals:
0-7; 7-14; 14-21 … 70-77

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
```

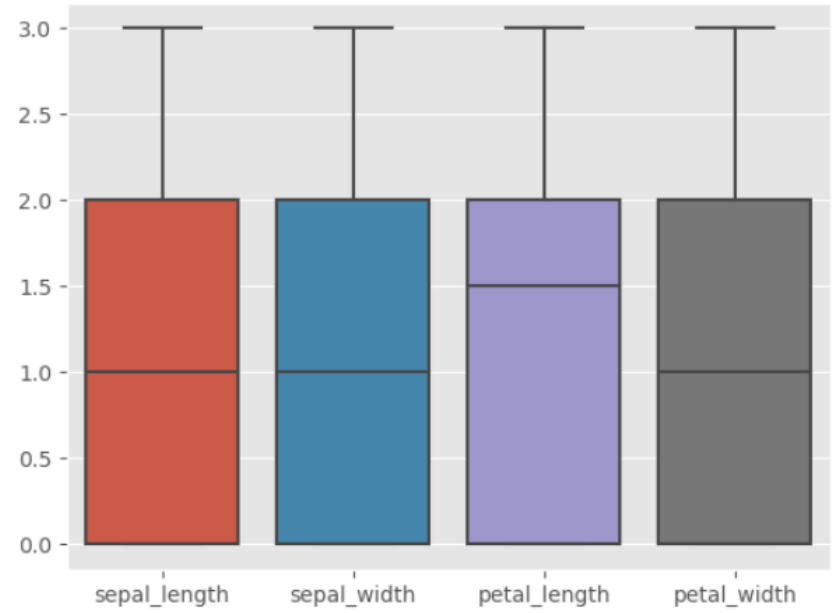
```
irisCopy = iris.copy()
for c in iris.columns[:-1]:
    irisCopy[c] = pd.cut(iris[c], bins=4,labels=False) #four bins

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()
```



Equal-frequency Discretization

- Equal frequency discretization divides the data into intervals (**N bins)** of equal frequency
  - Let's say we have a dataset containing the age of individuals. We want to discretize the ages into three categories: 'young', 'middle-aged', and 'old'
  - We sort the ages in ascending order: [10, 25, 30, 35, 38, 45, 50, 55, 60, 65, 70, 75]
  - We divide the sorted ages into three intervals of equal frequency (each interval contains approximately the same number of observations)
  - We want to discretize them into three categories. We have 12 ages, so each category should ideally contain around 4 ages.
  - After equal-frequency discretization, the categories might look like this:
    - 'young': [10, 25, 30, 35]
    - 'middle-aged': [38, 45, 50, 55]
    - 'old': [60, 65, 70, 75]

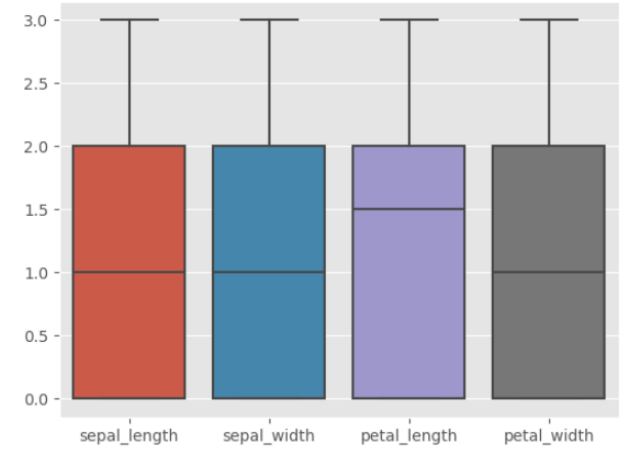
Equal-frequency Discretization

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

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



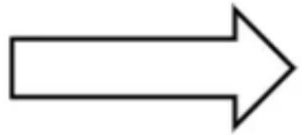

```
irisCopy = iris.copy()
for c in iris.columns[:-1]:
    irisCopy[c] = pd.qcut(iris[c], q=4,labels=False) #define the quantiles of the ranges

irisCopy.head(10)
```

	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

Categorical Variable Encoding

Grades
A
B
C
D
Fail



Grades	Encoded
A	4
B	3
C	2
D	1
Fail	0

03.dp\06-categorical-encoding

Categorical encoding

- Categorical encoding refers to replacing the string values of a categorical variable with a numerical representation
  - To produce variables that can be used to train ML models
- Many approaches available:
  - One hot encoding (used for Linear models)
  - Ordinal / Label encoding (used for Tree based models)
  - Count / frequency encoding

One hot encoding

- One-hot encoding represent categorical variables as **binary vectors**
  - Each category is represented by a binary variable (0 or 1), where 1 indicates the presence of the category and 0 indicates the absence
  - Suitable for tree-based ML algorithms

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

- One hot encoding into k -1 variables
  - More generally, a categorical variable should be encoded by creating **k-1** binary variables, where k is the number of distinct categories
  - We can use 1 less dimension and still represent the whole information
  - Suitable for linear models

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow
1	0
1	0
0	1
0	0
0	1

One hot encoding - Limitations

- Expands the feature space without adding extra information
  - Not suitable **high cardinality** categorical variables
- Many dummy variables may be identical, introducing redundant information

Ordinal Encoding

- Ordinal / Label / Integer encoding:

  replacing the categories by digits from 1 to n (or 0 to n-1), where n is the number of distinct categories of the variable

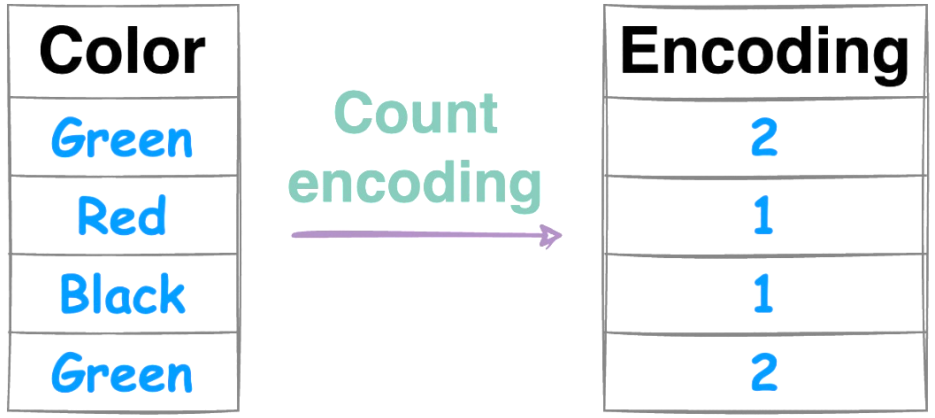
  + Does not expand the feature space
  + Can work ok with tree-based ML algorithms
  - Not suitable for linear models

Color
Green
Red
Blue

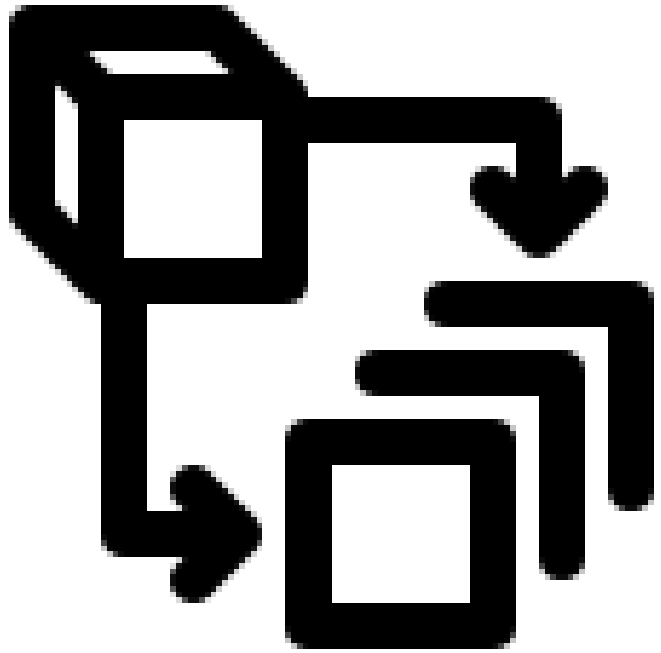
Color
1
2
3

Count / frequency encoding

- Count / frequency encoding: replacing categorical labels with the count or percentage of observations that belong to each category in the dataset
  - + Does not expand the feature space:
    - + Suitable for handling categorical variables with high cardinality
  - + Can work ok with tree-based ML algorithms
  - - Not suitable for linear models
  - - If 2 different categories appear in the same number of observations, they will be replaced by the same number
    - - May result in losing valuable information



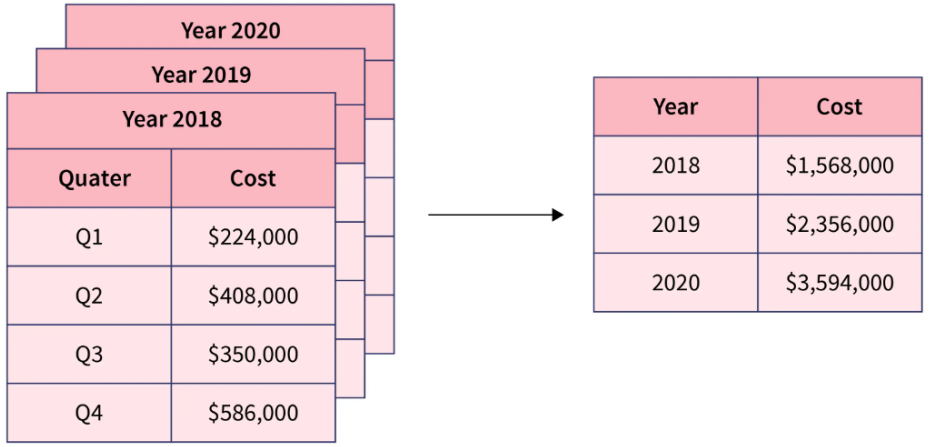
Feature extraction



03.dp**07-feature-extraction**

Feature Creation

- Creation of new features by combining existing ones using math functions, such as:
  - Data aggregation: summarise low level data details to higher level data abstraction by applying aggregate functions (e.g. count, sum, average)
  - Ratio: Debt to income ratio
  - Subtraction: Income - Expenses

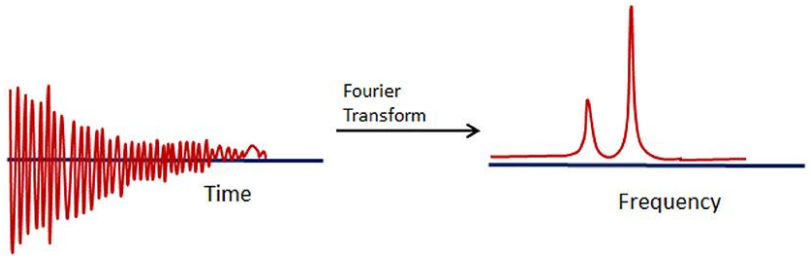


Advantages

- reduce the time of learning
- discover more stable patterns
- reduce noise and diminish distortion

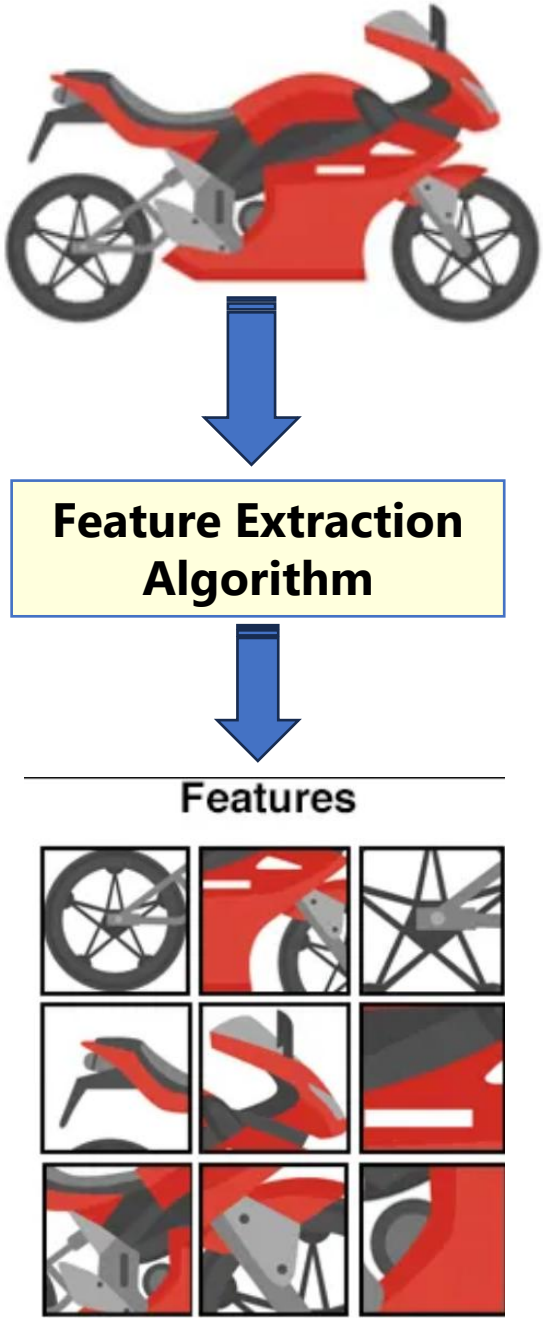
Feature extraction

- Create new features from functions of the original features
- 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

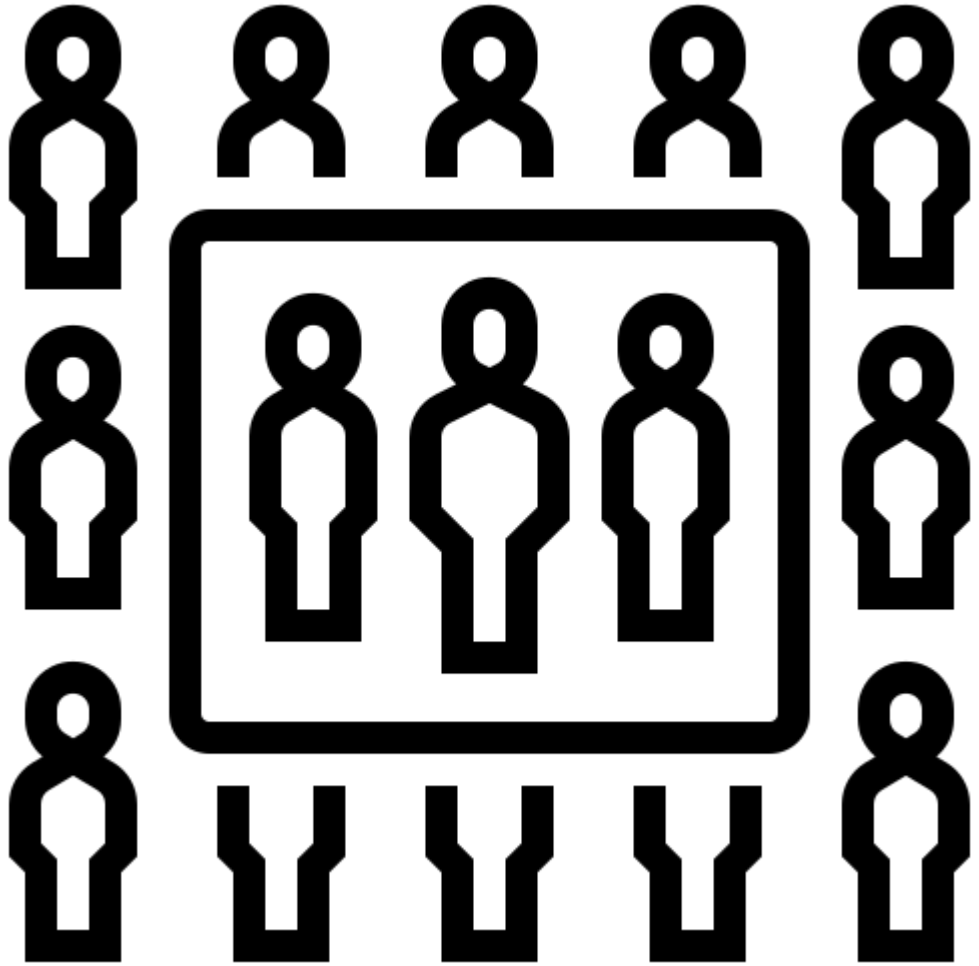


Mapping data to a new space, e.g. wavelet transformation from time domain to frequency domain

- Deep learners automatically extract features
  - e.g., CNNs can extract the relevant areas in an image by themselves without any image pre-processing.



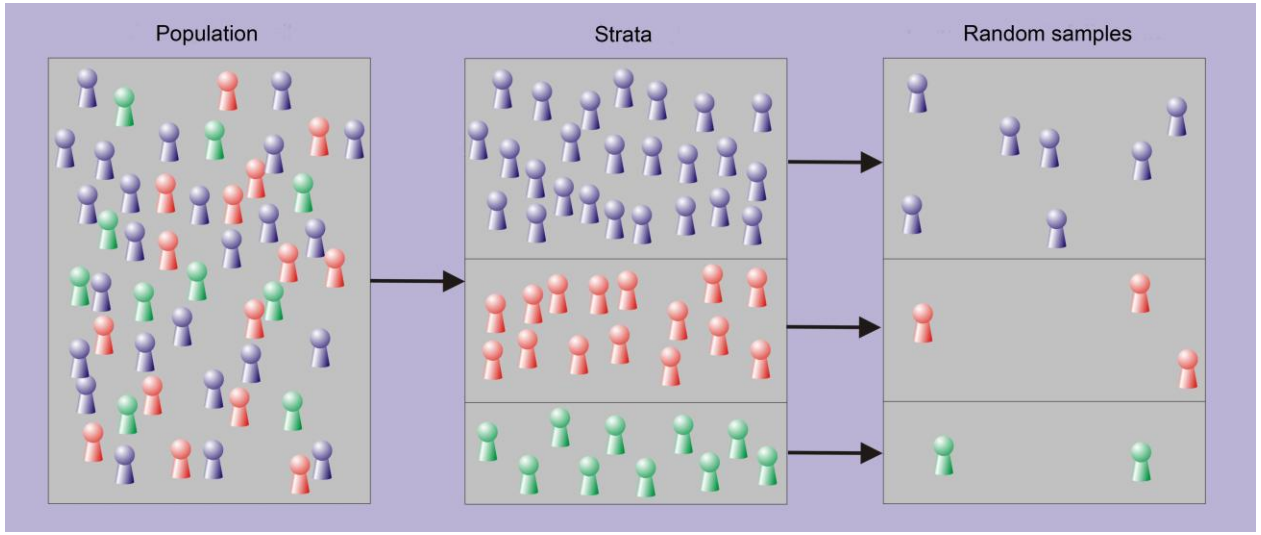
Sampling



03.dp\08-sampling

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



https://www.pinterest.com/pin/357754764120670109/

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

Address Imbalanced Data using Random over/under sampling

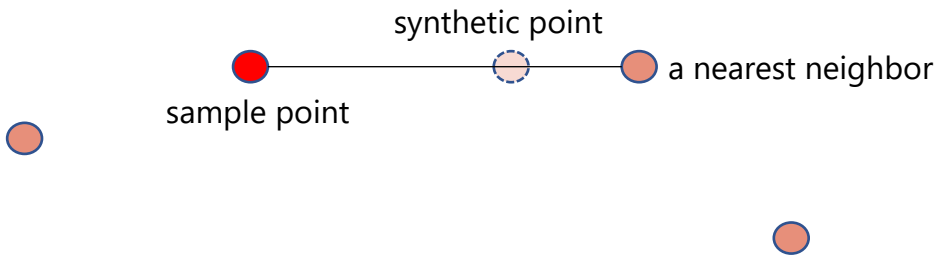
- Random *oversampling*: randomly duplicate data points from the minority class
  - Overfitting and fixed boundaries
- Random undersampling: randomly delete data points from the majority class
  - Causes Loss of information

Synthetic Minority Over-sampling Technique (SMOTE)

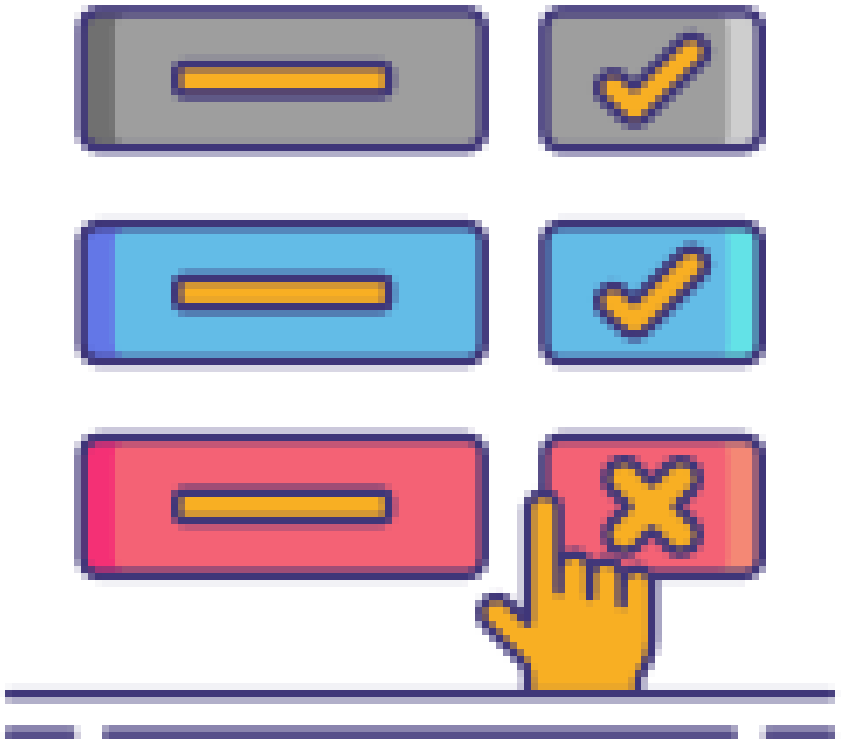
- Main steps:

1. Take the difference between a sample point and one of its nearest neighbors.
2. Multiply the difference by a random number between 0 and 1 and add it to the feature vector.

This causes the selection of a random point along the line segment between two specific features.



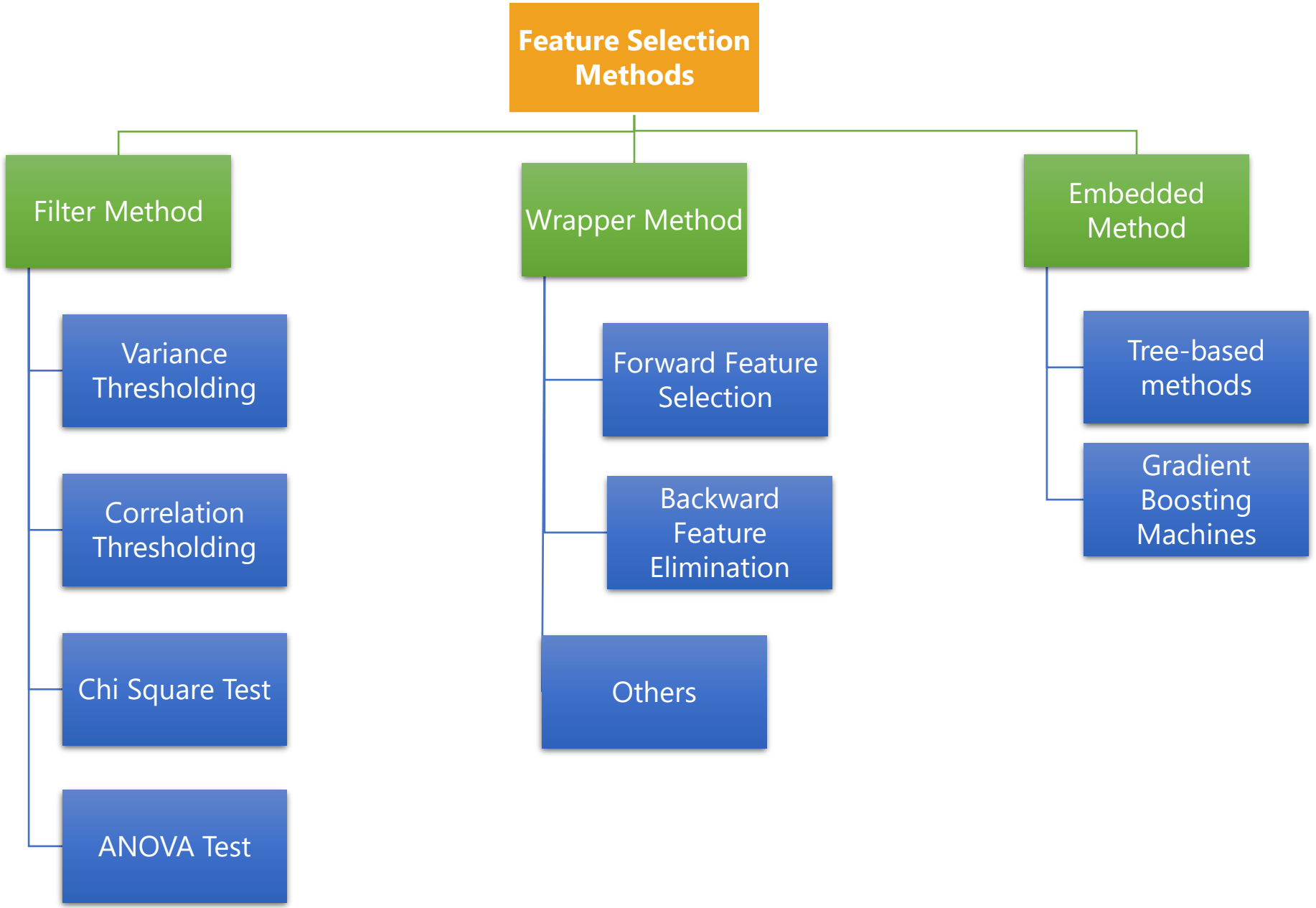
Feature Selection

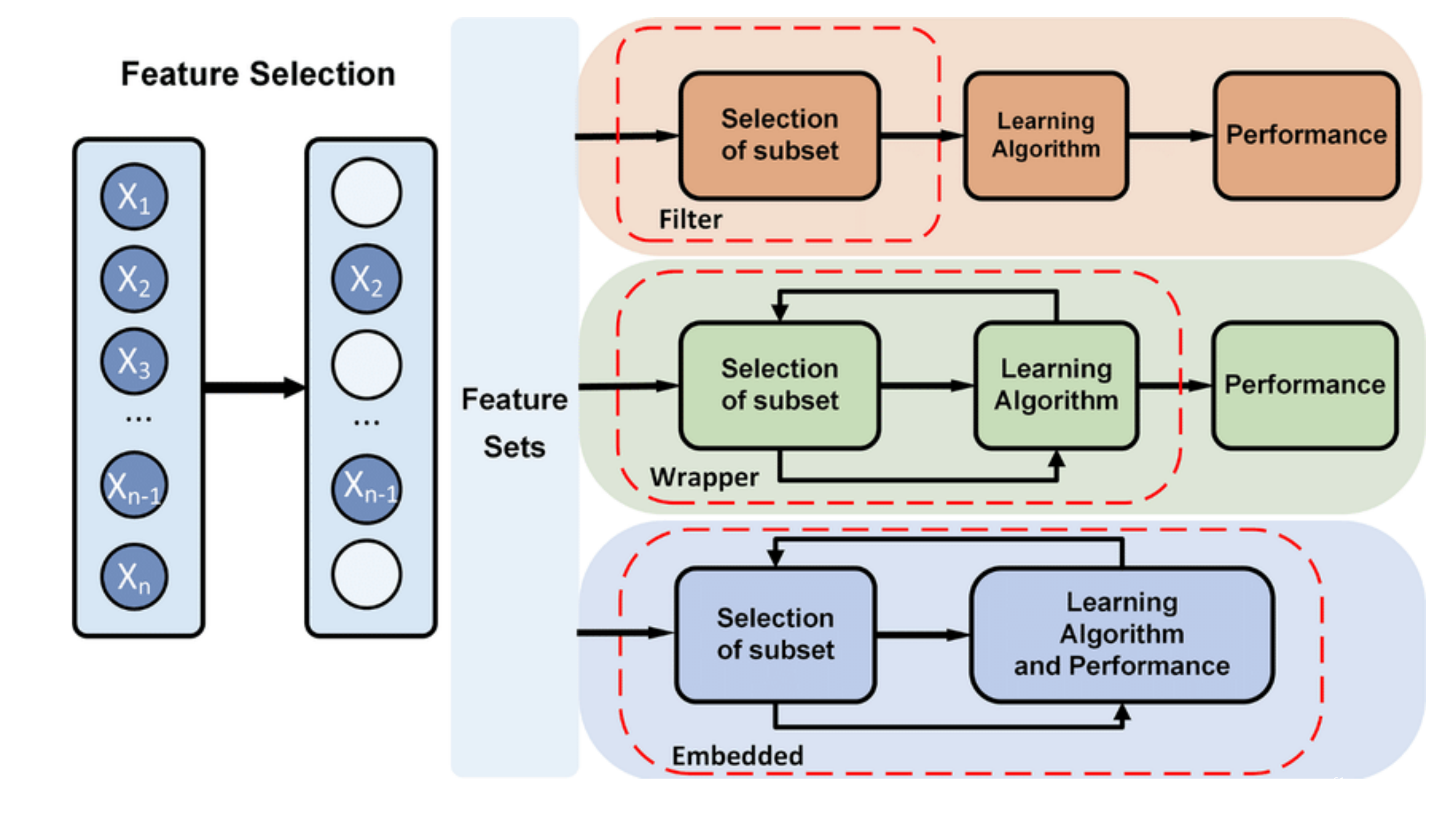


03.dp**09-feature-selection**

Feature Selection

- Feature selection is the process of selecting a subset of features to train ML models:
  - Some features may be *redundant* or *irrelevant*
- Why use a subset of useful/relevant features?
  - Simplify ML models => make them easier to interpret and maintain
  - Shorter training times
  - Avoid the curse of dimensionality
  - Enhance performance
- Alternative to **Dimensionality Reduction Techniques** e.g., Principal Component Analysis (PCA), are used to **transform** the original feature space into a lower-dimensional space while preserving the most important information


Feature Selection Methods
Filter Method
Wrapper Method
Embedded Method
Variance Thresholding
Correlation Thresholding
Chi Square Test
ANOVA Test
Forward Feature Selection
Backward Feature Elimination
Others
Tree-based methods
Gradient Boosting Machines


Feature Selection
X1
X2
X3
...
Xn-1
Xn
X2
...
Xn-1
Feature Sets
Selection of subset
Learning Algorithm
Performance
Filter
Selection of subset
Learning Algorithm
Performance
Wrapper
Selection of subset
Learning Algorithm and Performance
Embedded

Feature Selection Methods

- **Manual**: use domain knowledge and common sense
- **Filter**: select features using statistics (e.g., variance and correlation), before training:
  - Remove features with **low variance**, as they have weak discriminatory power

  E.g., feature "Has Email Address" having 1 for 99% of users

  - Select features with the **highest correlation coefficients** with the target variable

  E.g., In house price prediction, keep features like size or number of rooms if they correlate well with price
- **Wrapper**: Train a model on different feature subsets, and choose the best-performing one
- **Embedded**: feature selection happens during model training, e.g., decision trees, gradient boosting

Wrapper Method - Forward Feature Selection

- Start with an empty Feature Set (FS)
- Add one feature at a time that improves model performance the most
- Repeat by testing the current set plus one remaining feature
- Stop when no significant improvement is achieved (or a target performance is reached)

👎 May miss important feature combinations

👎 Computationally expensive for large feature sets

Forward Feature Selection

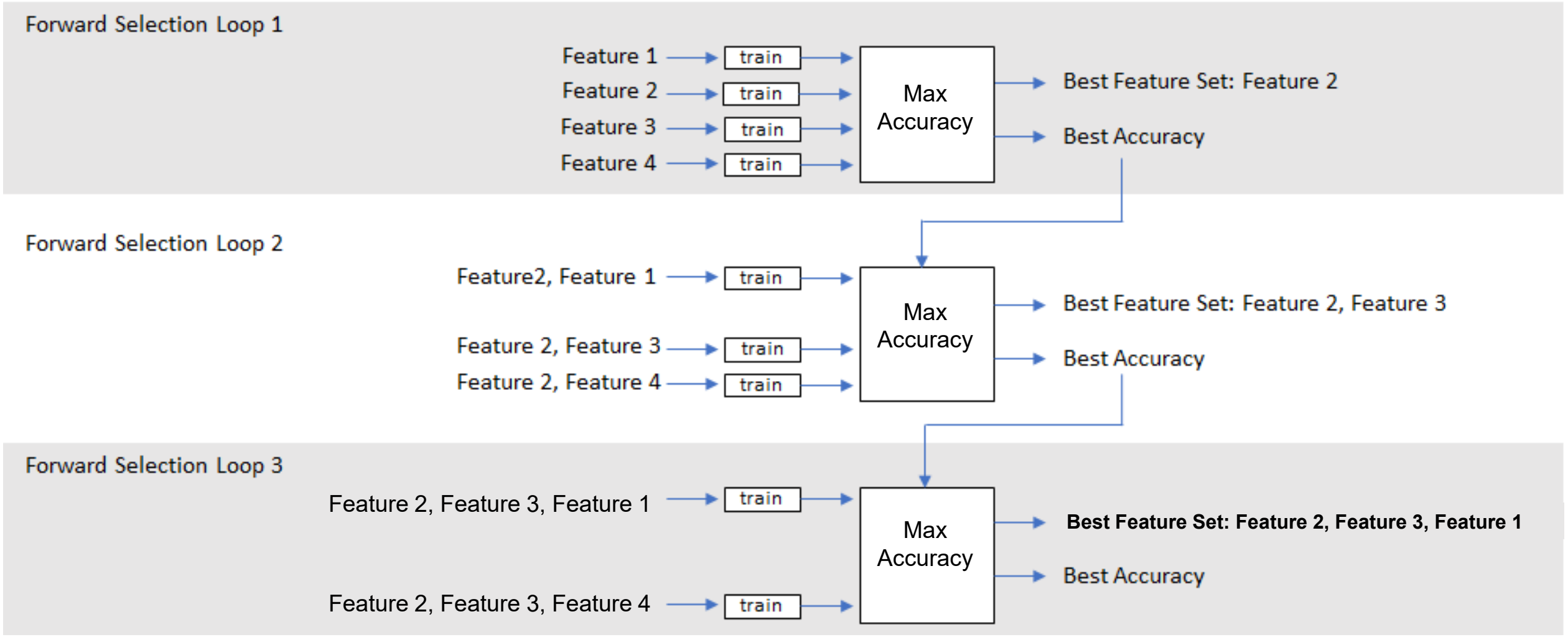


Image source

Wrapper Method - Backward Feature Elimination

- Start with all features
- Remove one feature at a time that least affects model performance
- Repeat until removing features significantly degrades performance (or a target performance is reached)
- 👍 It better captures **feature interactions** because removing an important feature leads to a noticeable drop in model accuracy
  - Features that work better together than alone
  - E.g., Transaction amount and transaction time may be weak alone, but together strongly indicate fraudulent behavior
  - E.g., Credit Card Balance and Credit Limit – Individually, each feature is weak

Together (credit utilization = balance / limit) they strongly predict default risk

Backward Feature Elimination

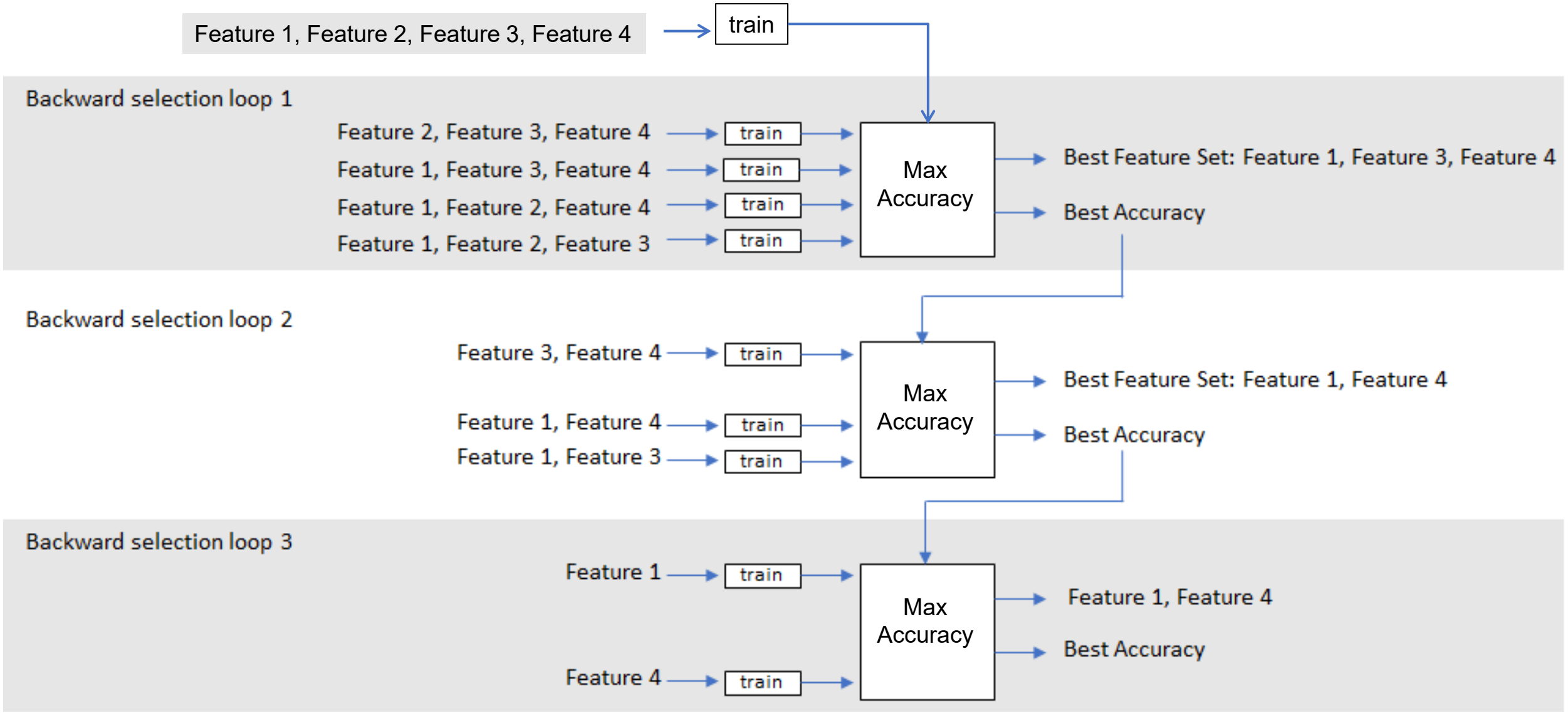


Image source

Dimensionality Reduction

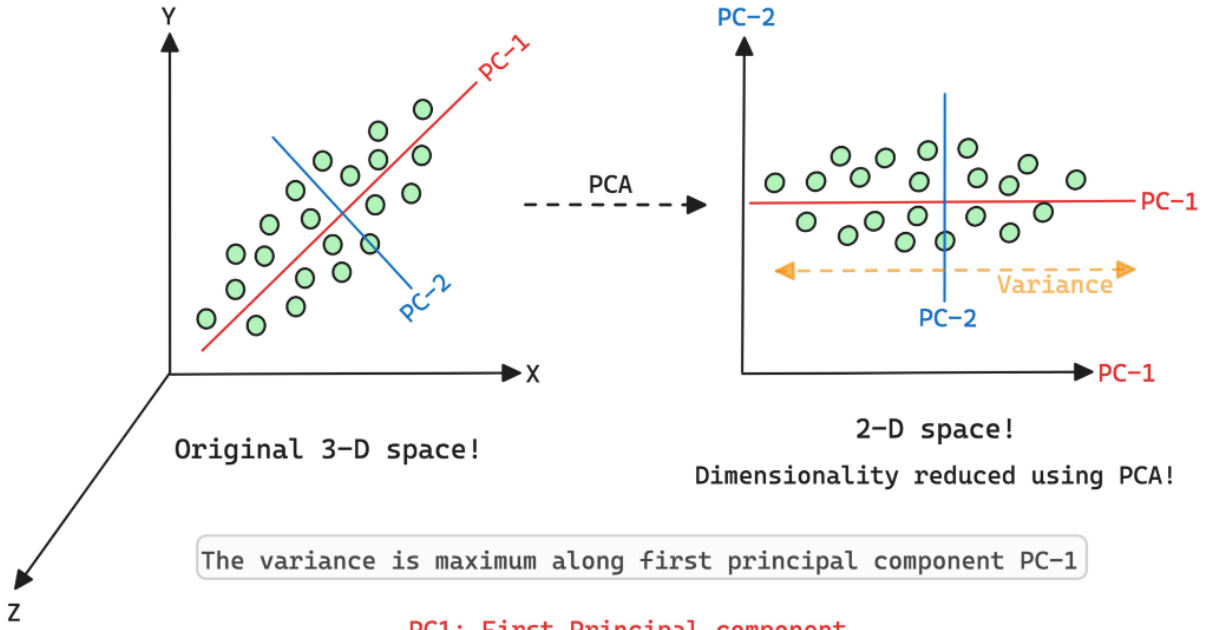


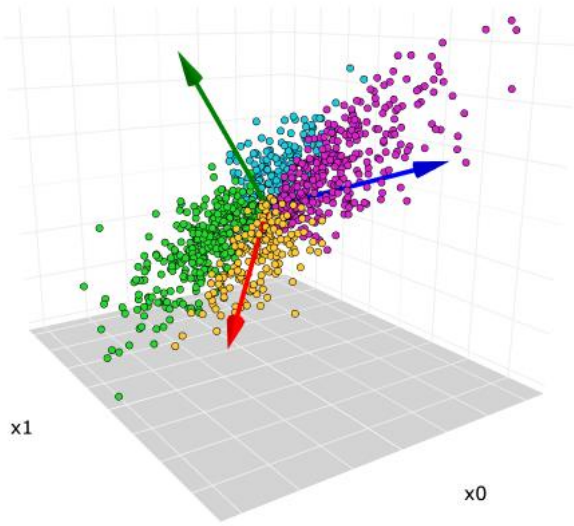
Image source

03.dp\10-pca

Dimensionality Reduction

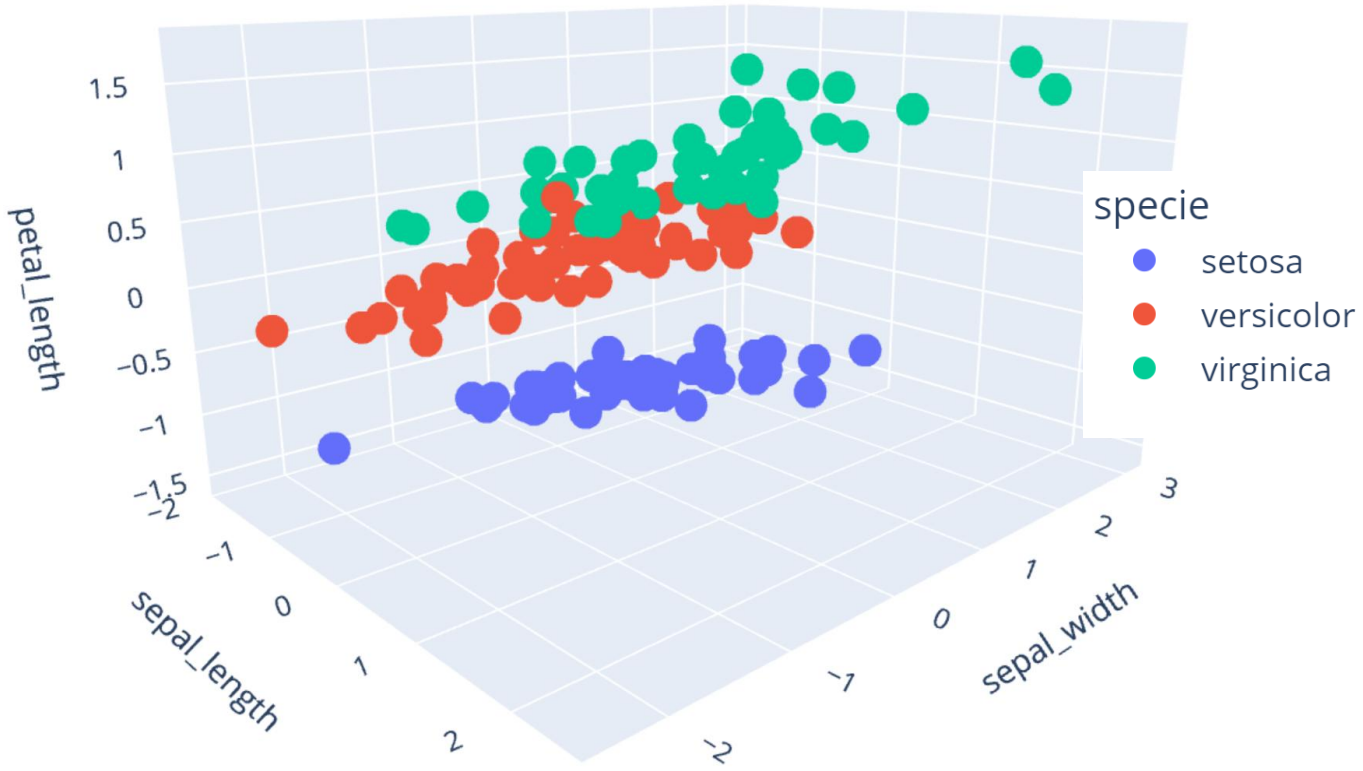
- **Dimensionality reduction** is used to reduce the number of features (dimensions) in a dataset while **preserving** the essential information **(i.e., retaining as much variability as possible)**
  - Analogous to summarizing the most important points of 1000-pages book in just 2 or 3 pages
  - It can help improve computational efficiency, and enable data visualization

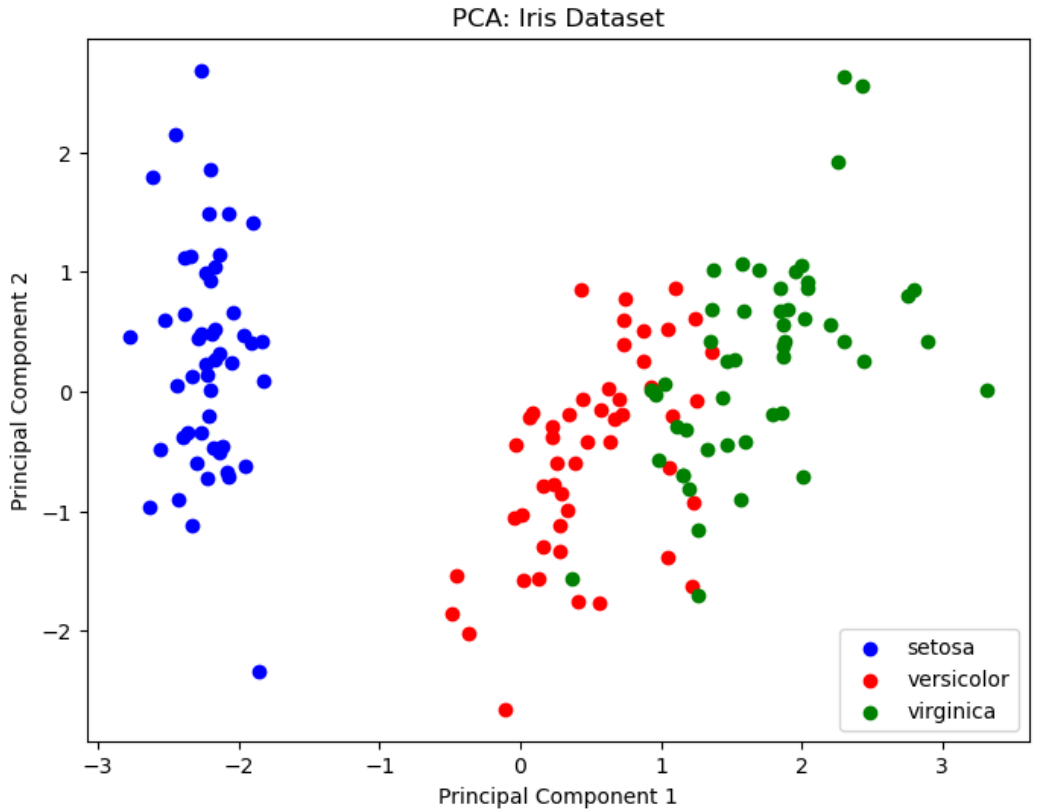
- One common method for dimensionality reduction is Principal Component Analysis (PCA)
  - PCA identifies the directions (principal components) along which the data varies the most (i.e., eigenvectors of covariance matrix)
  - Think of these "directions" as the main axes along which your data points are spread out the most
  - PCA is like a photographer for your data to capture the big picture without being overwhelmed by all the details
    - It helps you find the best angles to capture the most important aspects of your data and discarding less relevant details



More info in this article

PCA Example

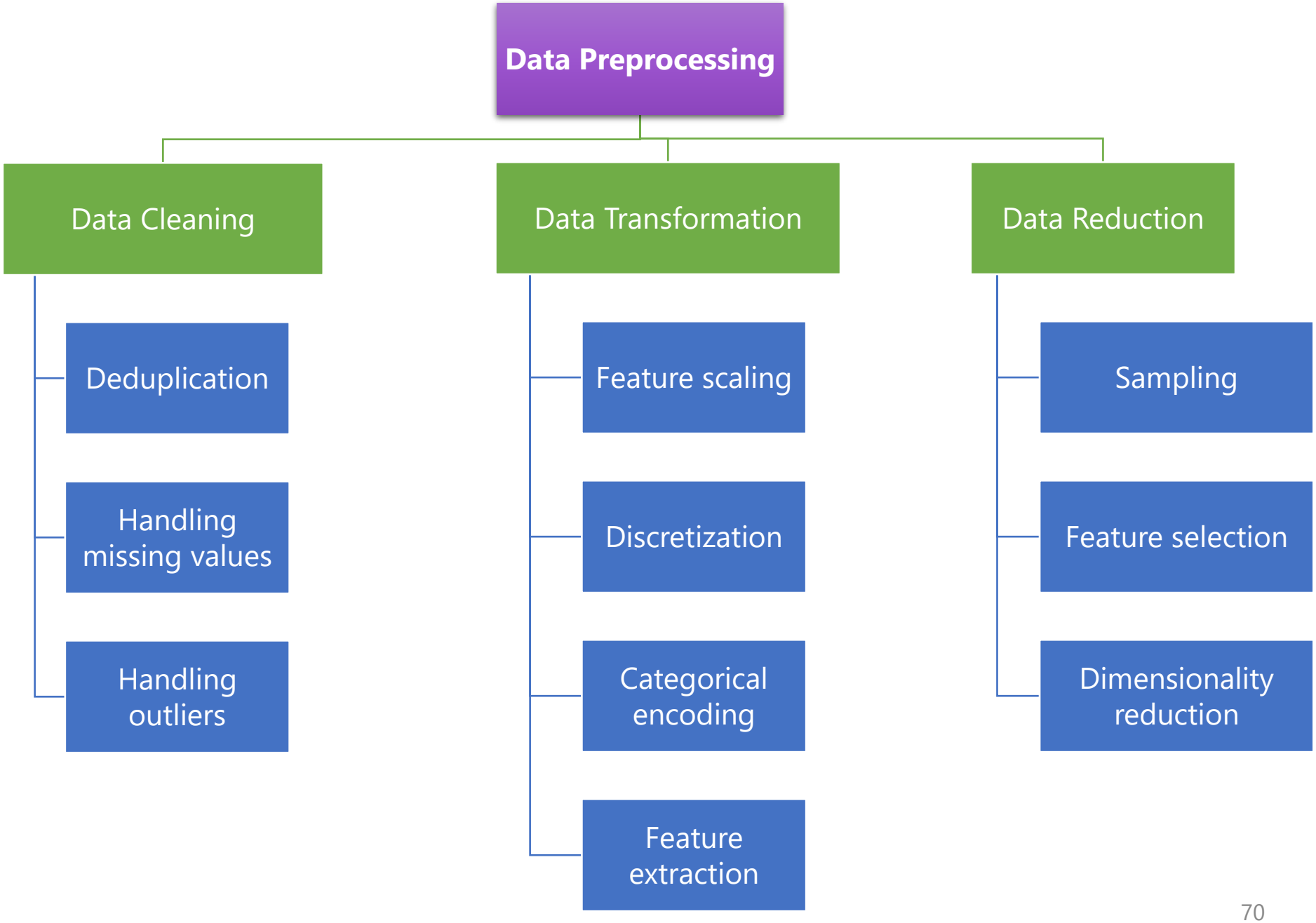




- (**Left**) The original data & (**Right**) The same data but reduced to 2-D with PCA

`03.dp\10-pca\pca.ipynb`

Summary


Data Preprocessing
Data Cleaning
Deduplication
Handling missing values
Handling outliers
Data Transformation
Feature scaling
Discretization
Categorical encoding
Feature extraction
Data Reduction
Sampling
Feature selection
Dimensionality reduction