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

- **Precision:** Precision refers to the accuracy of the measurements or values in the dataset.
  - Low precision implies that the data has a high level of variability or error, making it less reliable for analysis
  - Can be caused by measurement instruments, data collection processes, or human error
  - E.g., Due to calibration issues or environmental factors, the thermometer consistently records temperatures with an additional +/- 1.5 degrees Celsius of error
- **Bias:** systematic deviation of data from the true values. It might favor certain groups or outcomes, leading to inaccurate or unfair results
  - E.g., In a survey on job satisfaction, respondents are predominantly from urban areas, leading to a bias in the dataset towards urban perspectives. This bias might skew the results, as the experiences and opinions of individuals from rural or remote areas are not adequately represented
- **Noise:** random variations or errors in the data causing modification of original values 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
- **Outliers:** considerably different from most values in the dataset or unusual with respect to the typical values.
  - E.g., In a dataset of household income, there are a few entries representing extremely high incomes that are outliers
- **Irregular values:** unusual or unexpected values that don't conform to the expected patterns in the dataset, e.g., different values used (0, 1, m, f, M, and F) to refer to Male/Female. May come from different data sources, e.g., one rating "1,2,3", another rating "A, B, C"
- **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.

Data Main Quality Indicators

- **Accuracy:** data recorded with sufficient precision and little bias
- **Correctness:** data recorded without error and spurious objects
- **Completeness:** no parts of data records are missing
- **Consistency:** compliance with established rules and constraints
- **No redundancy:** absence of unnecessary duplicates

Why Data Quality is Important?

- No quality data leads 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 building a model to predict student GPA?

How to deal with Data Quality issues?

- Quality issues due to **invalid data**
  - Caused by errors in the features generation process
  - Solution: correct and regenerate them to ensure their accuracy and reliability

- Quality issues due to **valid data**
  - Exist because of domain-specific factors inherent to the dataset
  - Solution:
    - Correct in some cases
    - Do not correct unless required by the trained models, e.g., models cannot be training with missing values or outliers

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.

- 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

- DP tasks include:
  - Data cleaning, transformation and reduction

Data Deduplication

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

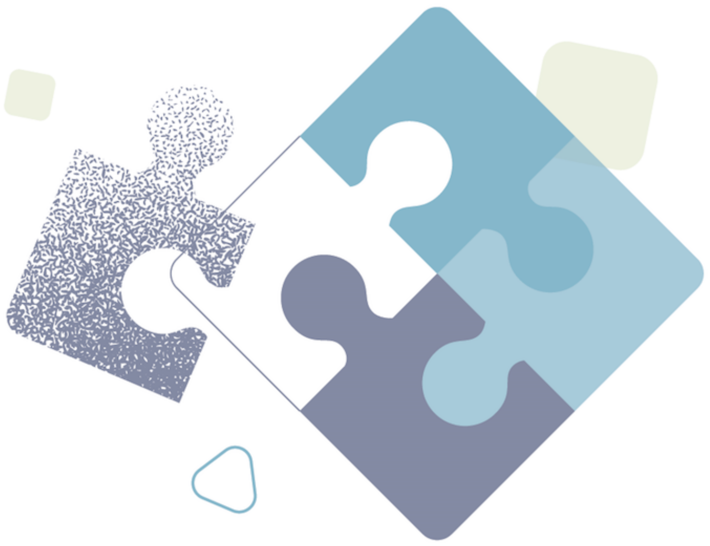
```
df.duplicated()  #True means the entry is duplicated.
```

```
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

Handling Missing Values



Handling missing values

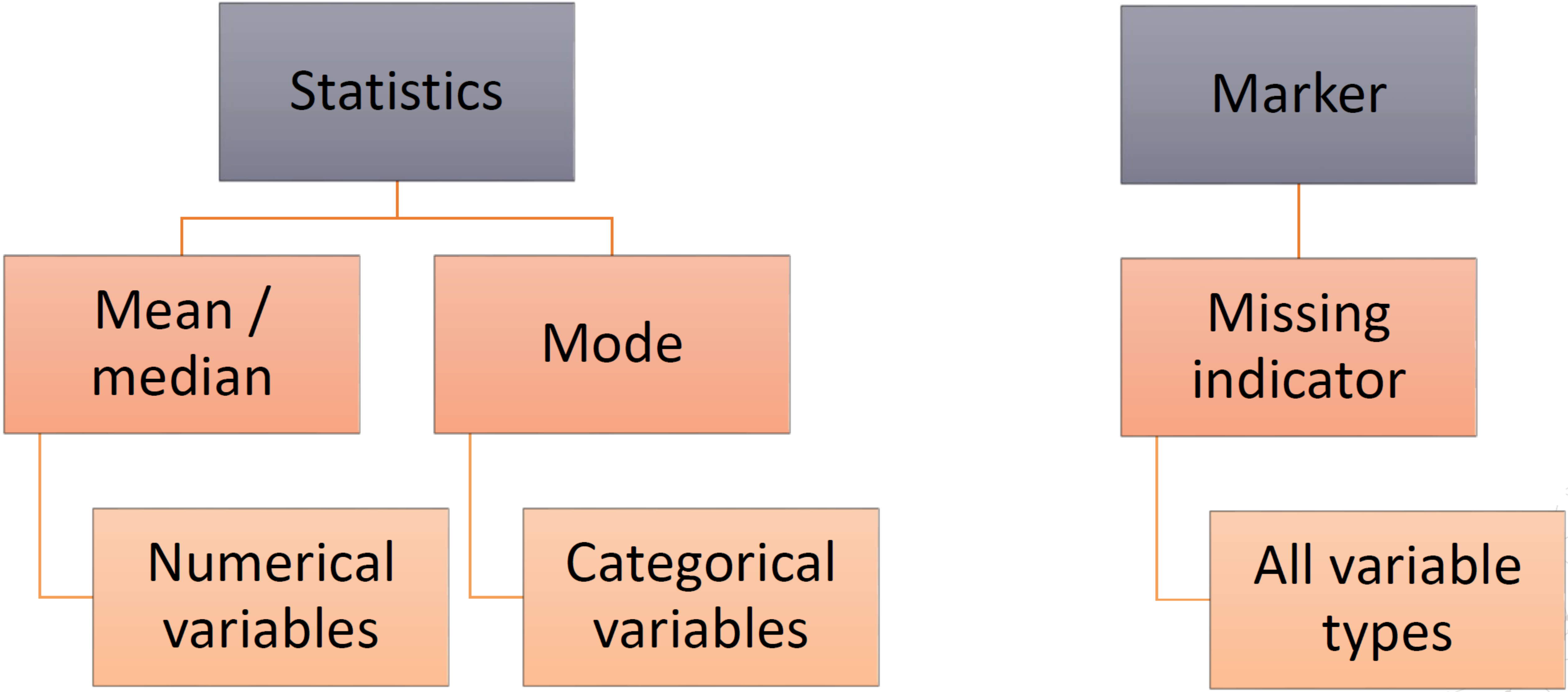
Missing Values: some instances without values for one or more features

- Remove features/instances that are excessively missing their values (e.g., in excess of 60%)
- Replace missing values with an indicator (flag)
- Impute with statistical estimates of the missing values e.g., mean, median for numerical features or mode for categorical 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

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

Handling missing values



Mean / Median imputation

- Mean / median imputation consists of replacing all occurrences of missing values (NA) within a variable by the mean or median
  - If the variable is normally distributed the mean and median are approximately the same
  - If the variable is skewed, the median is a better representation
- Suitable numerical variables
  - Use **mode** imputation for categorical variables
- Assumes data is missing at random

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 variables 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

Arbitrary value imputation

- Arbitrary value imputation consists of replacing all occurrences of missing values (NA) within a variable with an arbitrary value
  - A value that is different from most values in the distribution (Find the variable value range then pick a value outside that range)
  - Typically used arbitrary values are 0, 999, -999 or -1 for numerical variables
  - "Missing" often used categorical variables
  - Often used in combination with a **Missing Indicator** variable to flag missing data
- Used when data is not missing at random

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)**, that is, that they are 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	$\boxed{12}$				
Steve	83	12	77	7				
Clint	27	9	118	9				
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

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	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								
Bruce	37	14				veggie				
Steve	83	12				chicken				
Clint	27	9	118	9	0	shrimp	3			
Wanda	19	7	52	2	2	shrimp				
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
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	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

Handling missing values

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

Handling missing values

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

```python
# 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

```python
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

```python
#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

Impute with mean, median or mode of features

```python
# 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

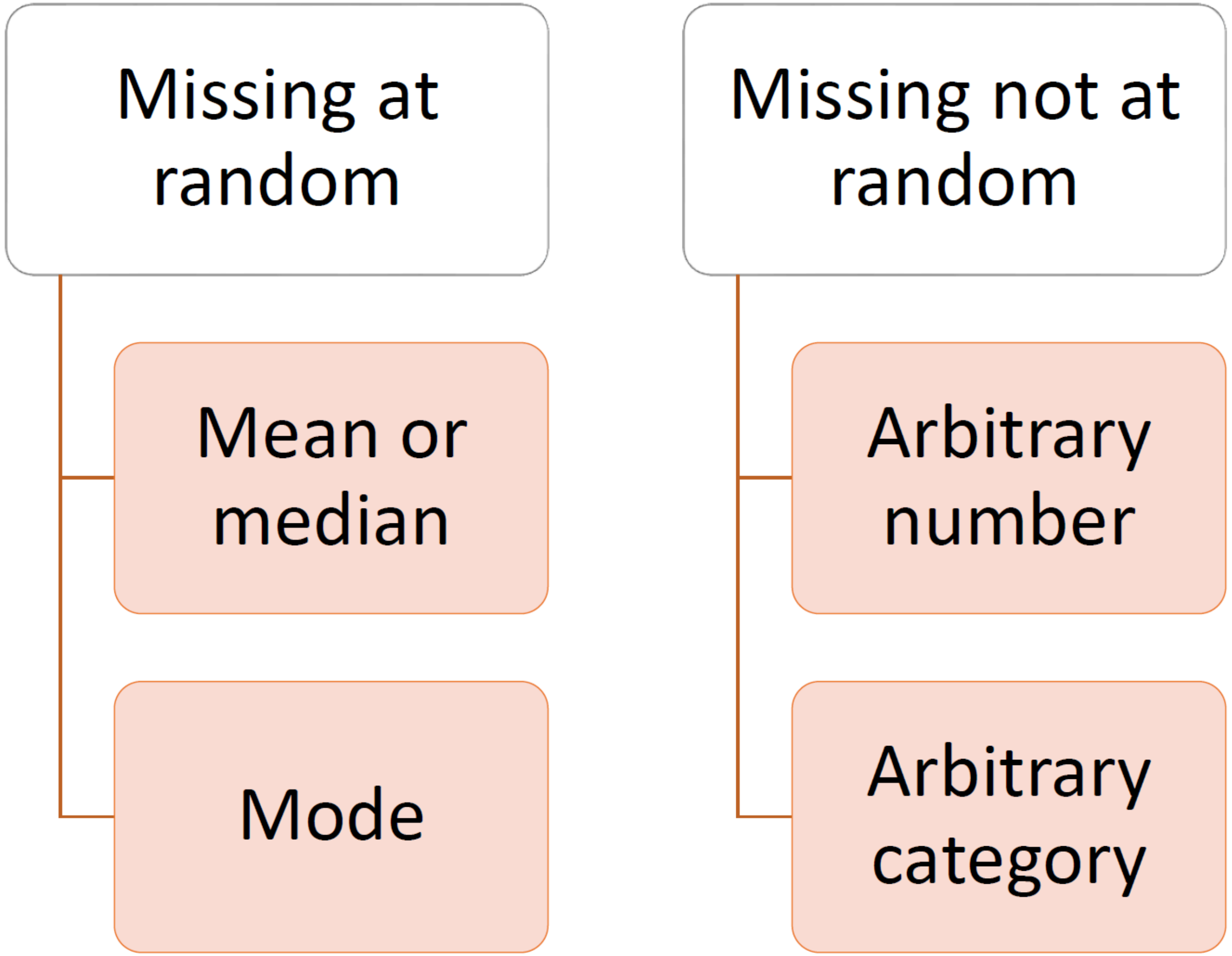
```python
df.mean()  #It i
```

```
A    4.666667
B    2.250000
C    6.000000
D    8.666667
E    9.750000
F    6.000000
dtype: float64
```

```python
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
```

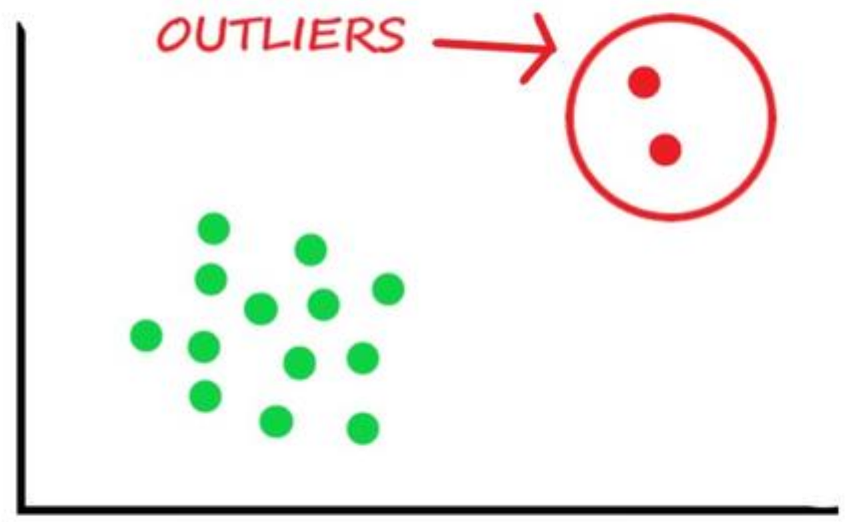
	A	B	C	D	E	F
0	12.000000	2.25	10	14.000000	20.00	10
1	4.666667	2.00	3	8.666667	9.75	3
2	1.000000	2.00	6	6.000000	8.00	6
3	4.666667	3.00	5	8.666667	3.00	5
4	1.000000	2.00	6	6.000000	8.00	6

Summary



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

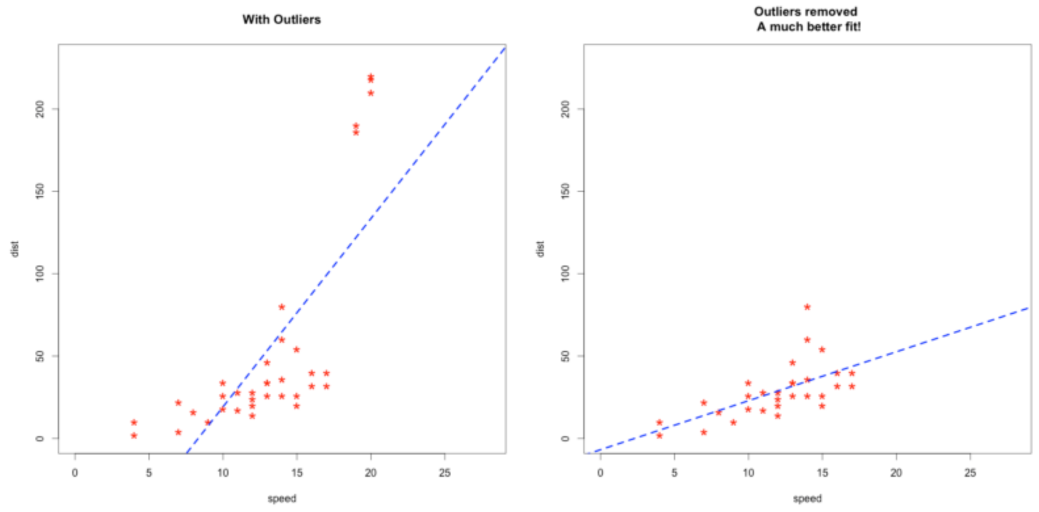
Handling Outliers



Handling Outliers

- **Outliers:** points that are considerably different from the other instances.
  - Some ML techniques, e.g., logistic regression, are sensitive to outliers.
  - Clamp/Cap/Do nothing 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) boxplot, scatter chart, bar chart or histogram plots
    - (Mathematically) IQR, mean/standard deviations, etc.
    - (Algorithmically) clustering

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



Ways to handle outliers

- **Trimming** (aka clamping): Removing outliers from the data set
  - Fast and simple but May remove big chunk of data
- **Capping** (aka winsorization), involves:
  - Set Thresholds (e.g., Q1 - 1.5 * IQR for the lower threshold, and Q3 + 1.5 * IQR for the upper threshold for skewed distribution OR Values outside mean $\pm 3 \times$ standard deviations for normal distributions)
  - Replace Outliers: any data point below the lower threshold is replaced with the lower threshold value, and any data point above the upper threshold is replaced with the upper threshold value
- **Treat outliers as missing data** and perform missing data imputation
  - These last 2 techniques: no data is removed but may distort the variable distribution
- **Discretization**: Put outliers into lower / upper bins

Feature Scaling

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

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 adjusting values measured on different scales to **a common scale**
  - Generally, the last step in the data pre-processing pipeline, performed just before training the ML model
  - Feature scaling does not change the shape of the distribution
- The goal of feature scaling is to:
  - prevent features with larger values from dominating the model
  - make the features more comparable
  - return better estimates if the models are based on distance calculations (like KNNs, k-means, and PCA)
  - speed up the convergence of gradient descent in neural networks and 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

Obs. - Min

Range

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

Standardization

- Centres the variable at zero and sets the variance to 1
  - 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 standardized feature value, x_i is the original value, μ is the mean for feature x, and σ 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]
  - Preserves 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

Price	Price
100	0.41
90	0.22
50	-0.57
40	-0.76
20	-1.16
100	0.41
50	-0.57
60	-0.37
120	0.80
40	-0.76
200	2.37

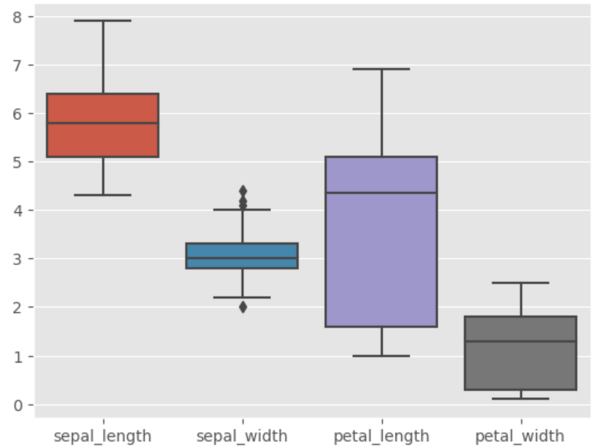
Mean = 79
Standard dev = 51

$$\frac{\text{Obs. - Mean}}{\text{Standard dev}}$$

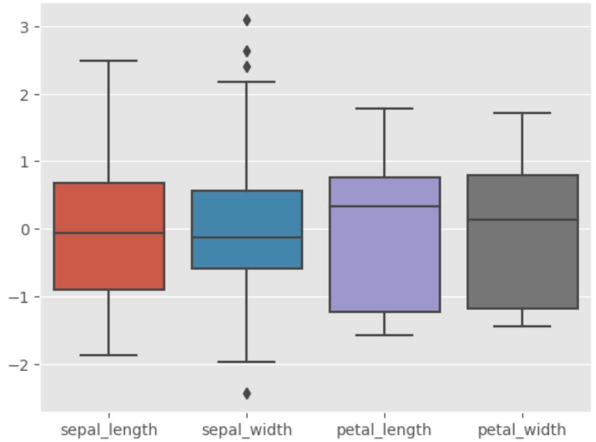
Feature scaling does not change the shape of the distribution

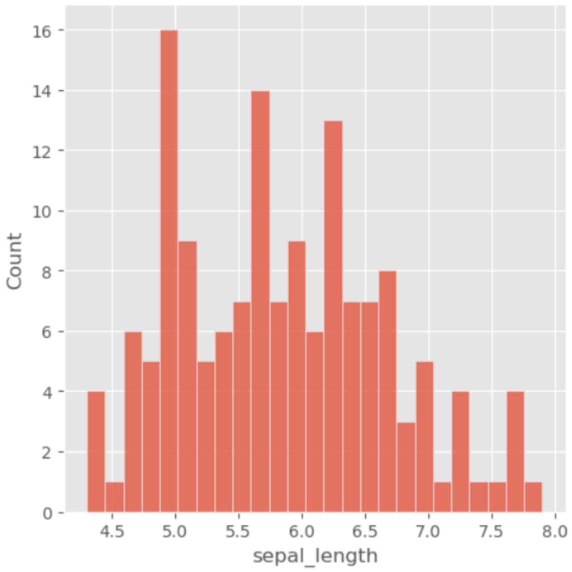
```python
iris_scaled = iris.copy()
X = iris_scaled.values[:,0:-1]  #columns, except the class label, of the matrix of the dataframe
X = X.astype(float) #convert X's dtype from object to float
X_scaled = preprocessing.StandardScaler().fit_transform(X)
iris_scaled.iloc[:,0:-1] = X_scaled
iris_scaled.head()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

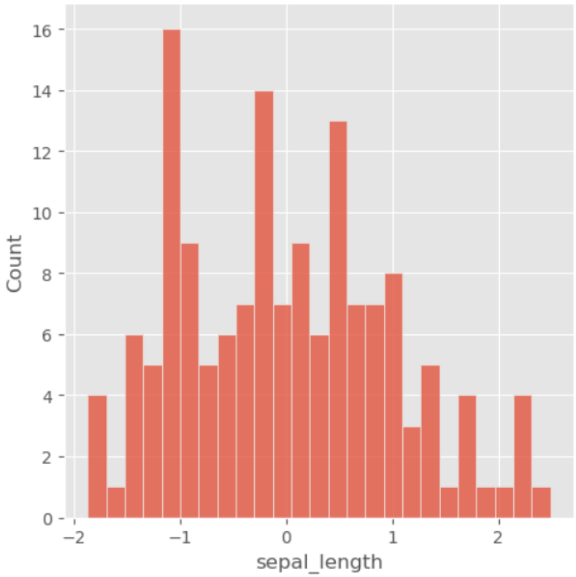
```python
sns.boxplot(data=iris)
plt.show()
```



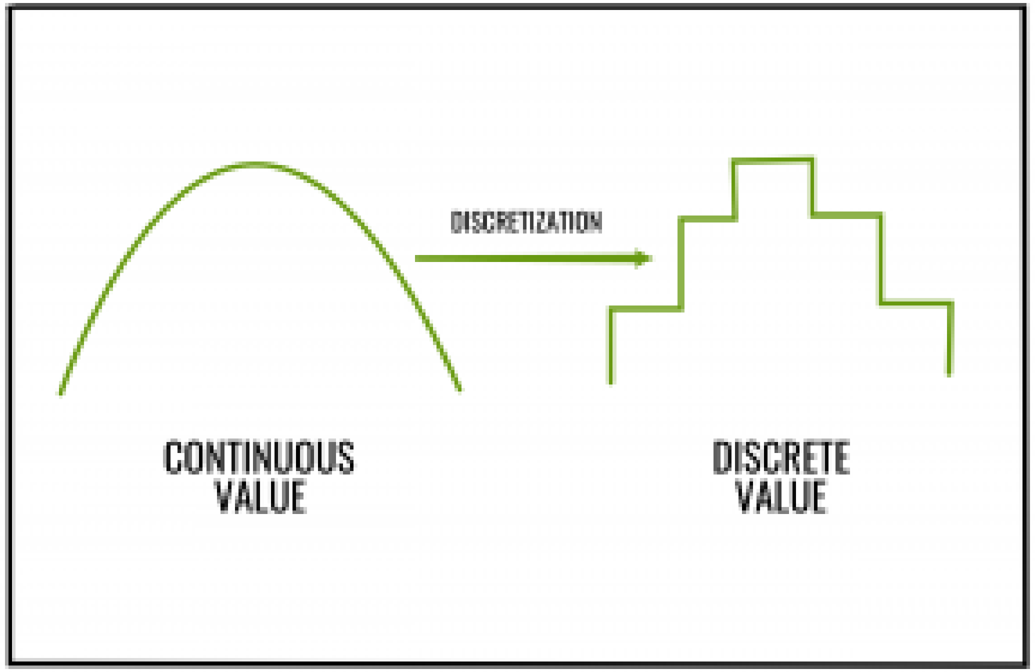

```python
sns.boxplot(data=iris_scaled)
plt.show()
```



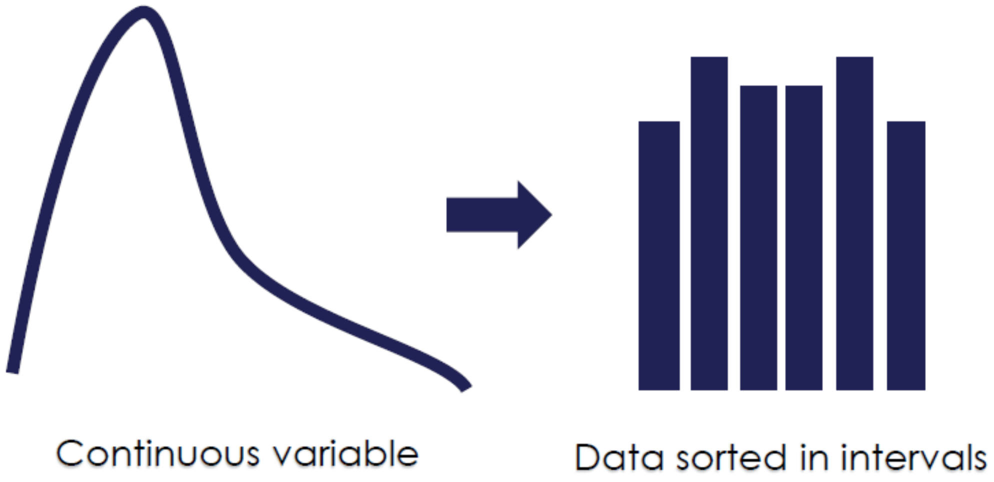




Discretization



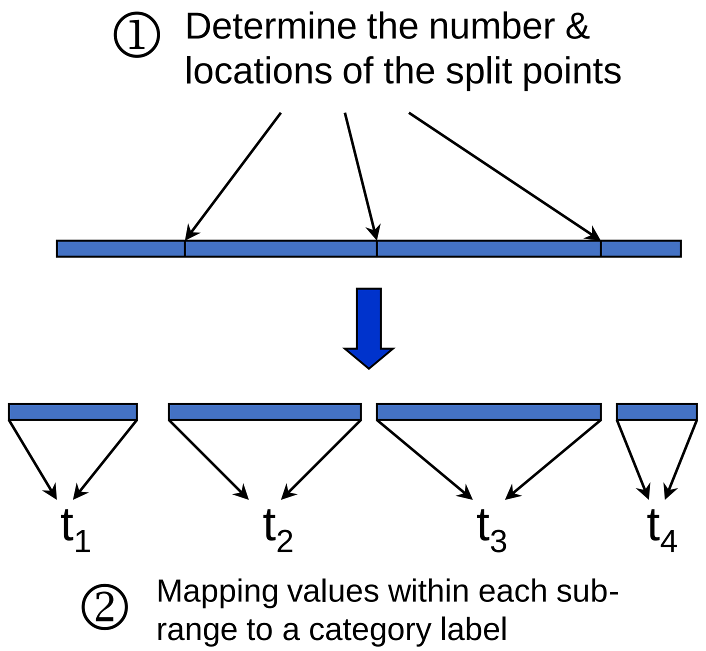
Discretization (Binning)

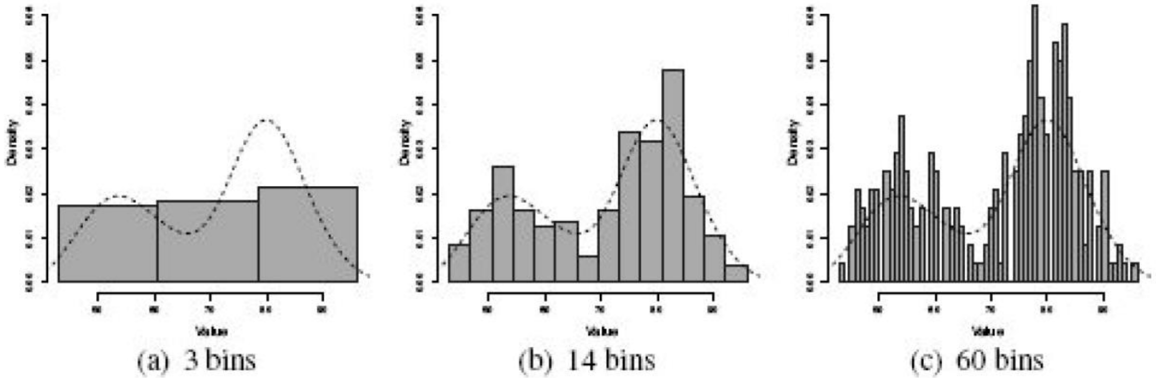


- Discretization (aka Binning) is the process of transforming continuous variables into discrete variables by creating a set of contiguous intervals (aka bins) that span the range of the variable's values
  - Improve performance (e.g., decision trees and Naïve Bayes, work better with discrete attributes)
  - 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

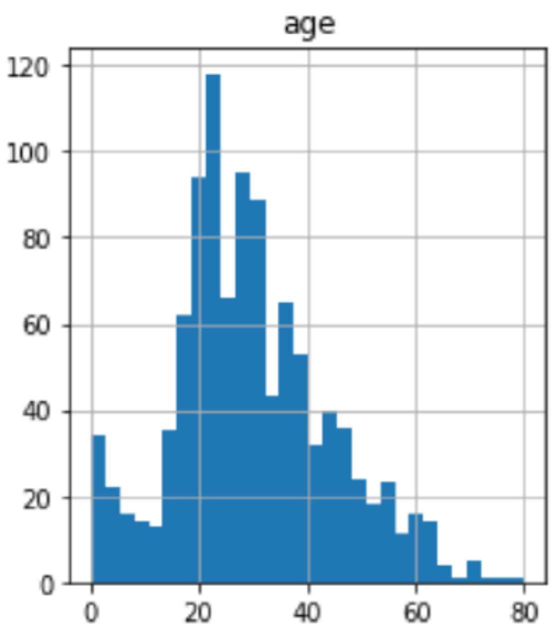
- Discretization can also lead to a loss of information
  - For example, by combining values that are strongly associated with different classes (target values) into the same bin
- The aim of a discretization algorithm is to find the minimal number of intervals without a significant loss of information
  - 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:

```
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
```

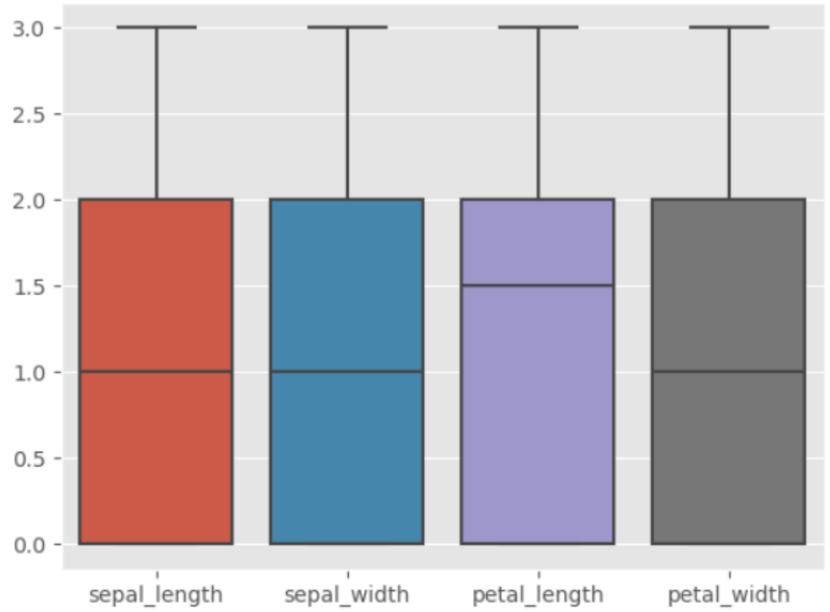
```
irisCopy = iris.copy()
for c in iris.columns[:-1]:
    irisCopy[c] = pd.cut(iris[c], bins=4,labels=False) #four bins

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]

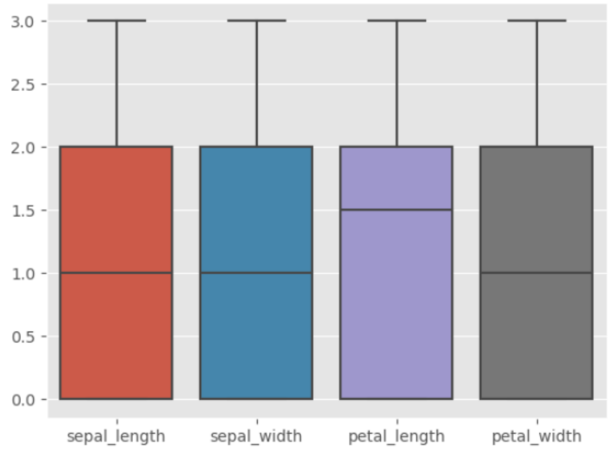
Equal-frequency Discretization

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

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
- Many dummy variables may be identical, introducing redundant information

Ordinal Encoding

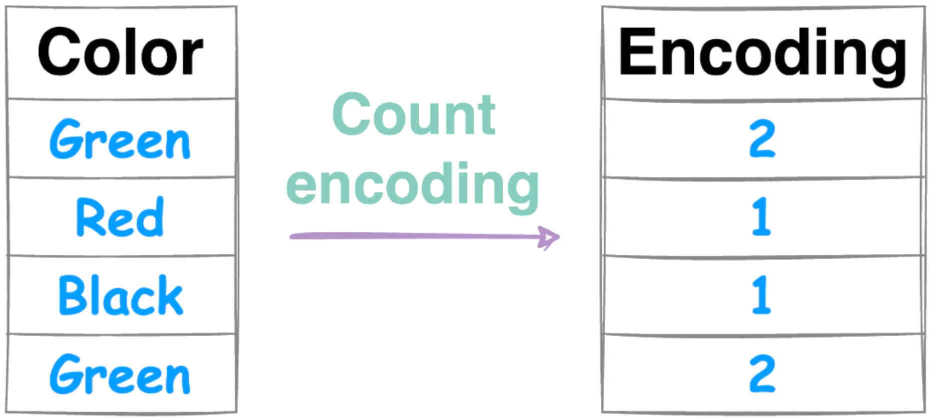
- Ordinal / Label / Integer encoding: replacing the categories by digits from 1 to n (or 0 to n-1), where n is the number of distinct categories of the variable
  - + Does not expand the feature space
  - + Can work ok with tree-based ML algorithms
  - - Not suitable for linear models

Color
Green
Red
Blue

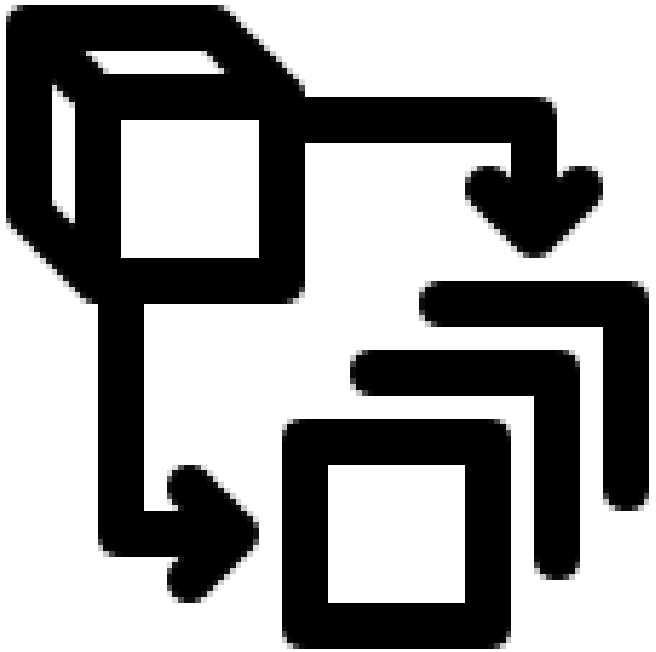
Color
1
2
3

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
  - \+ 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

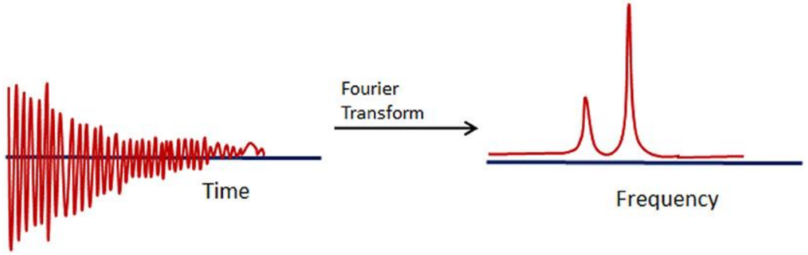


Feature extraction



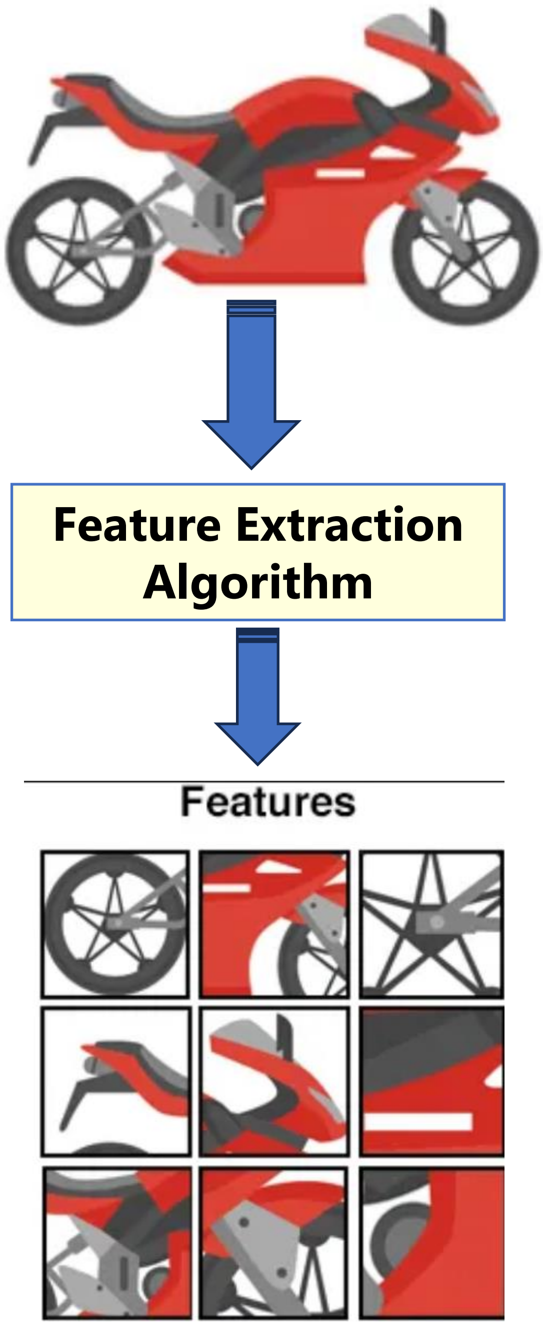
Feature extraction

- 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.



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

Year 2020

Year 2019

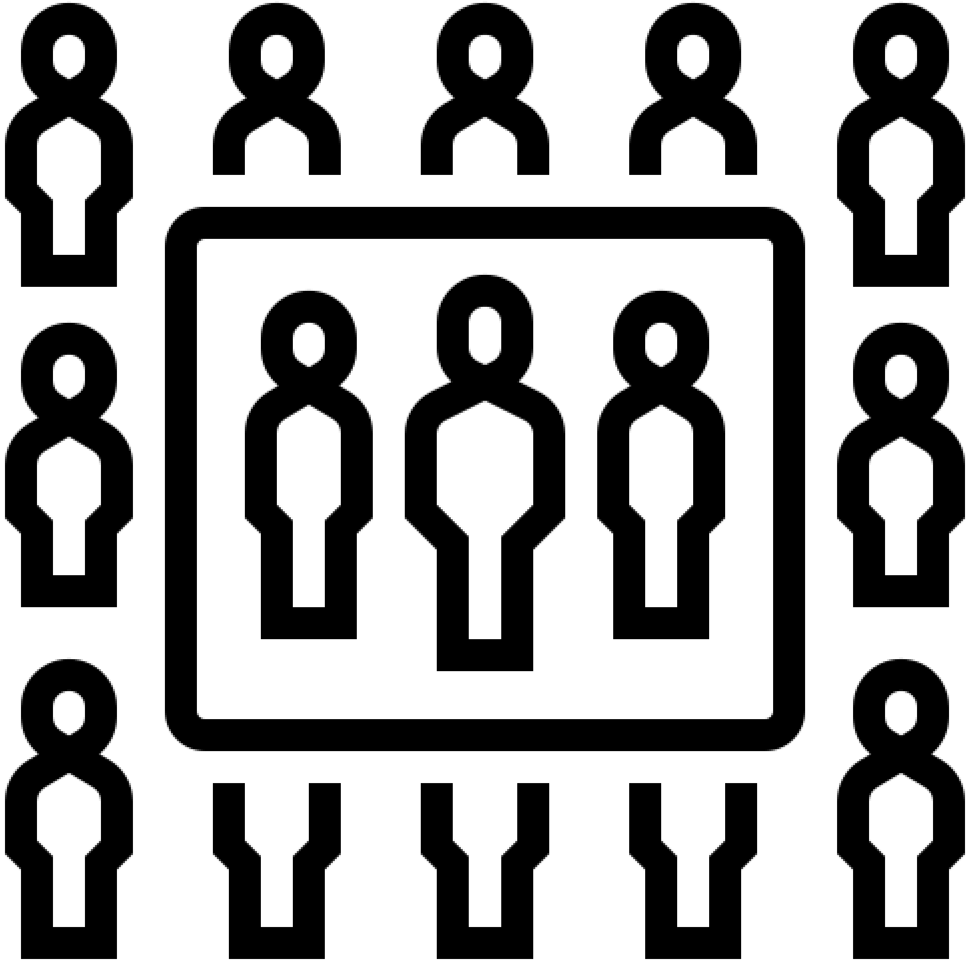
Year 2018	
Quater	Cost
Q1	$224,000
Q2	$408,000
Q3	$350,000
Q4	$586,000

Year	Cost
2018	$1,568,000
2019	$2,356,000
2020	$3,594,000

Advantages

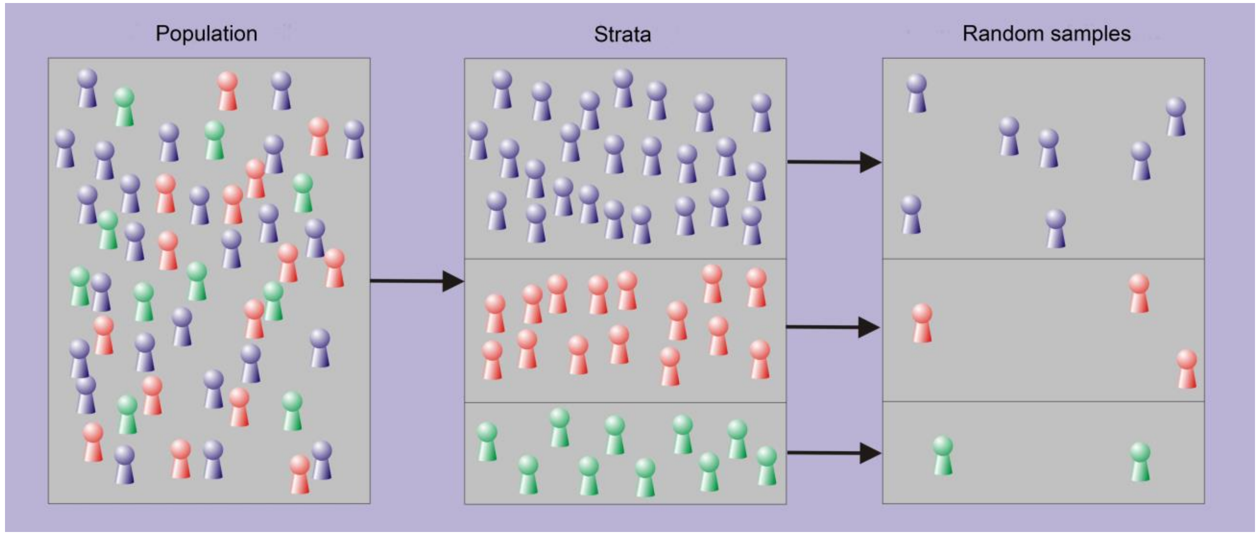
- reduce the time of learning
- discover more stable patterns
- reduce noise and diminish distortion

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

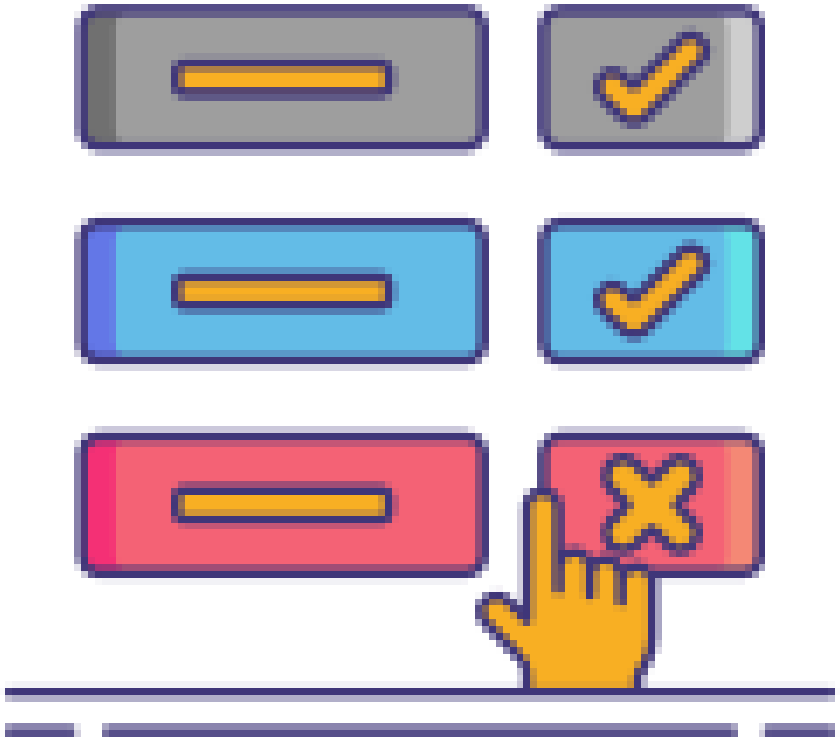
```
#Sample 2 rows from each class
df_sample = df.groupby('Class', group_keys=False).apply(lambda x: x.sample(2))
df_sample
```

	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
```

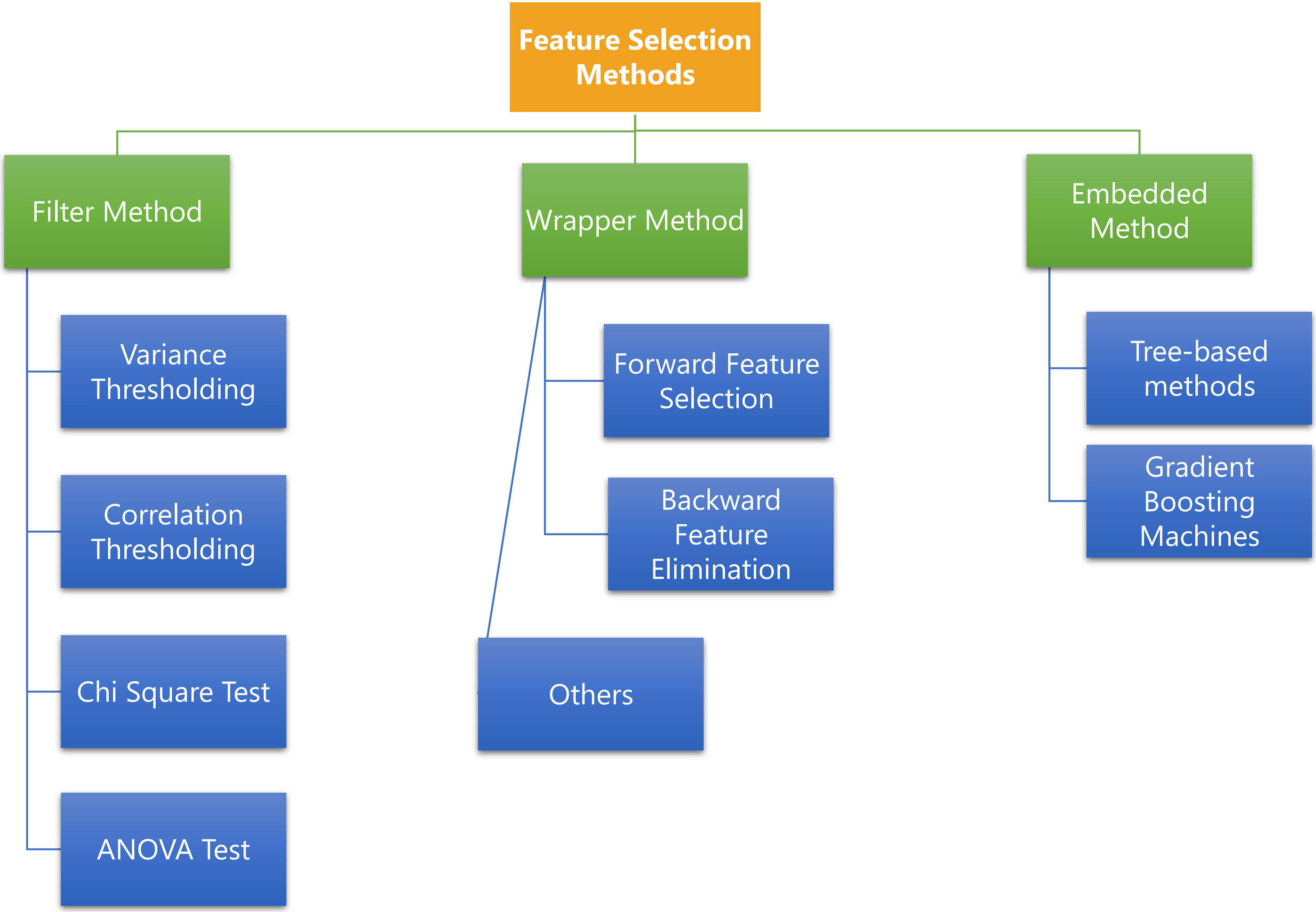
	Age	Salary	Class
Person 1	20	25000	No
Person 5	55	25400	No
Person 4	22	26900	Yes
Person 9	80	32400	Yes
Person 8	60	35900	Yes
Person 2	65	25500	Yes

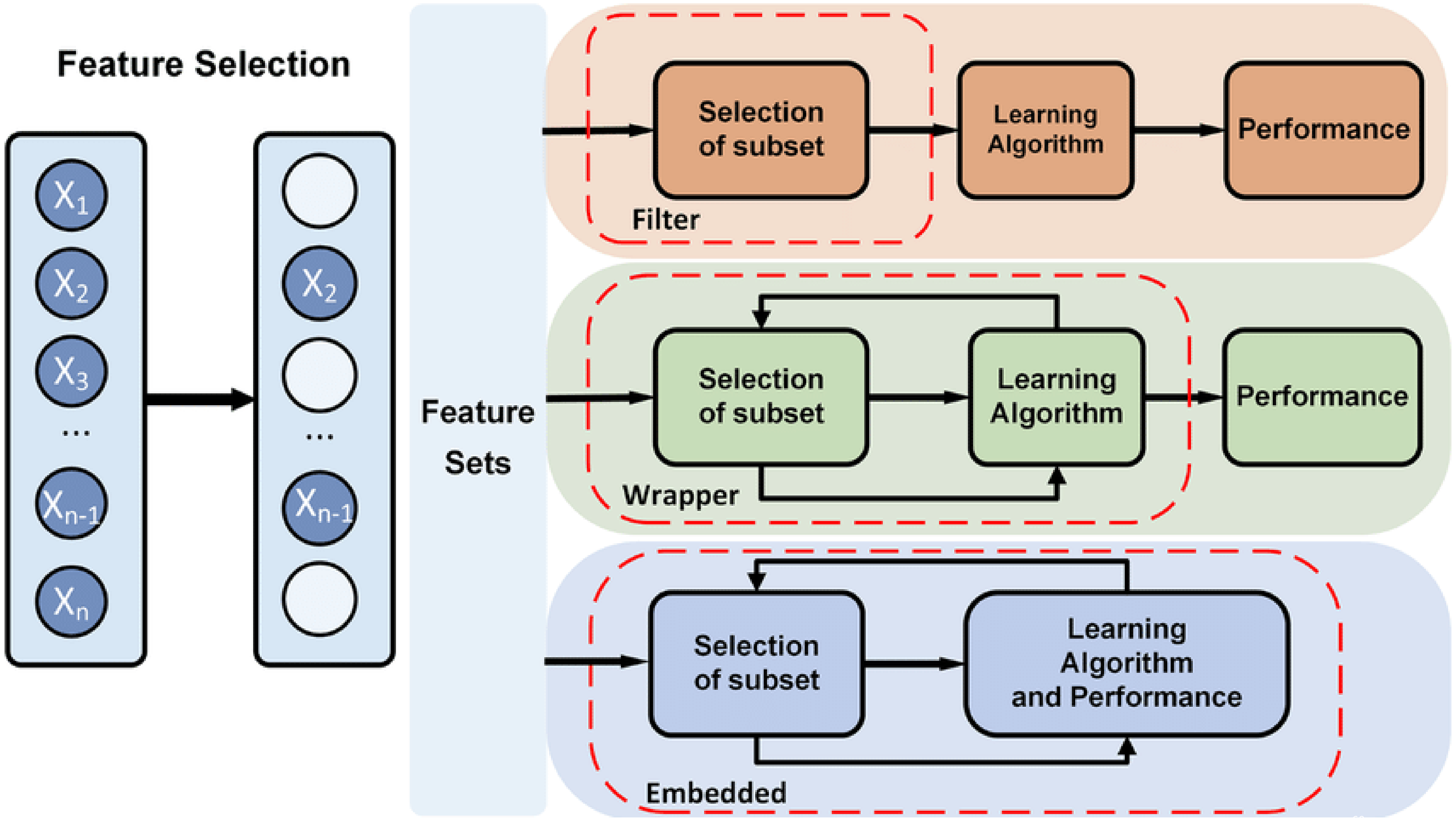
Feature Selection



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
X_1
X_2
X_3
...
X_{n-1}
X_n
X_2
...
X_{n-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

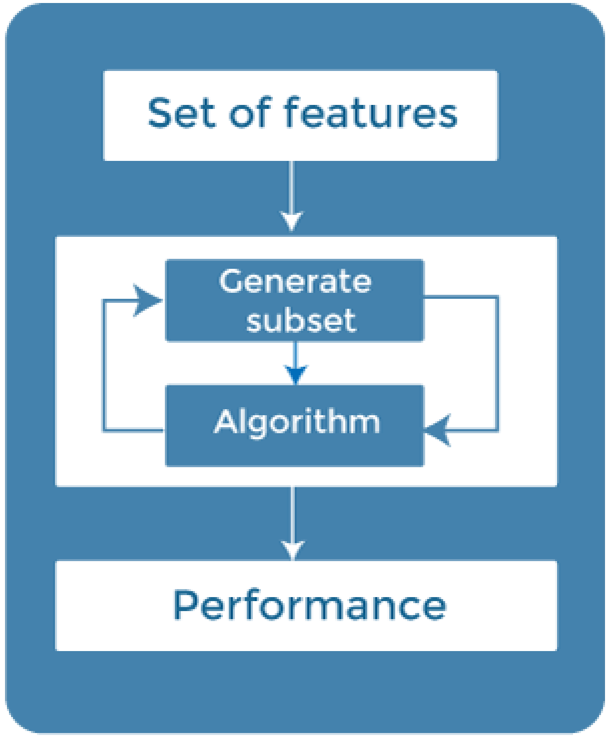
- **Manually:** use common sense and domain knowledge
- **Filter Methods**: Select features based on their statistical properties such as correlation
  - They are independent of the ML algorithm and assess the relevance of features before model training
- **Wrapper Methods**: Unlike filter methods, wrapper methods select features by directly measuring the performance of a specific ML algorithm
  - They involve iterative model training using different feature subsets and evaluating the performance of the model based on metrics like accuracy
- **Embedded Methods**: Embedded methods are feature selection techniques that are integrated into the model training process
  - E.g., Tree-based methods and Gradient Boosting Machines inherently perform feature selection by penalizing less important features

Filter Methods

- Filter methods rely on statistical measures to filter/rank features independently of the ML model being used
  - ✓ Offer computational efficiency and simplicity
  - May overlook complex relationships and interaction between features: A feature may not be useful on its own but may be an important influencer when combined with other features. Filter methods may miss such features
- **Variance Threshold**: Remove features with low variance, as they may provide less discriminatory power
  - E.g., In a dataset with a binary outcome variable (0 or 1), if a feature has very little variation (e.g., 99% of the samples have the same value of 1), it may not contribute much to the predictive power of the model
- **Correlation-based Feature Selection**: Select features with the highest correlation coefficients with the target variable
  - Features with higher correlation coefficients are more likely to contribute significantly to predicting the target variable
  - E.g., In a housing price prediction model, you can calculate the correlation between each numerical feature (e.g., size, number of bedrooms, distance to the city center) and the target variable (house price)

Wrapper Methods

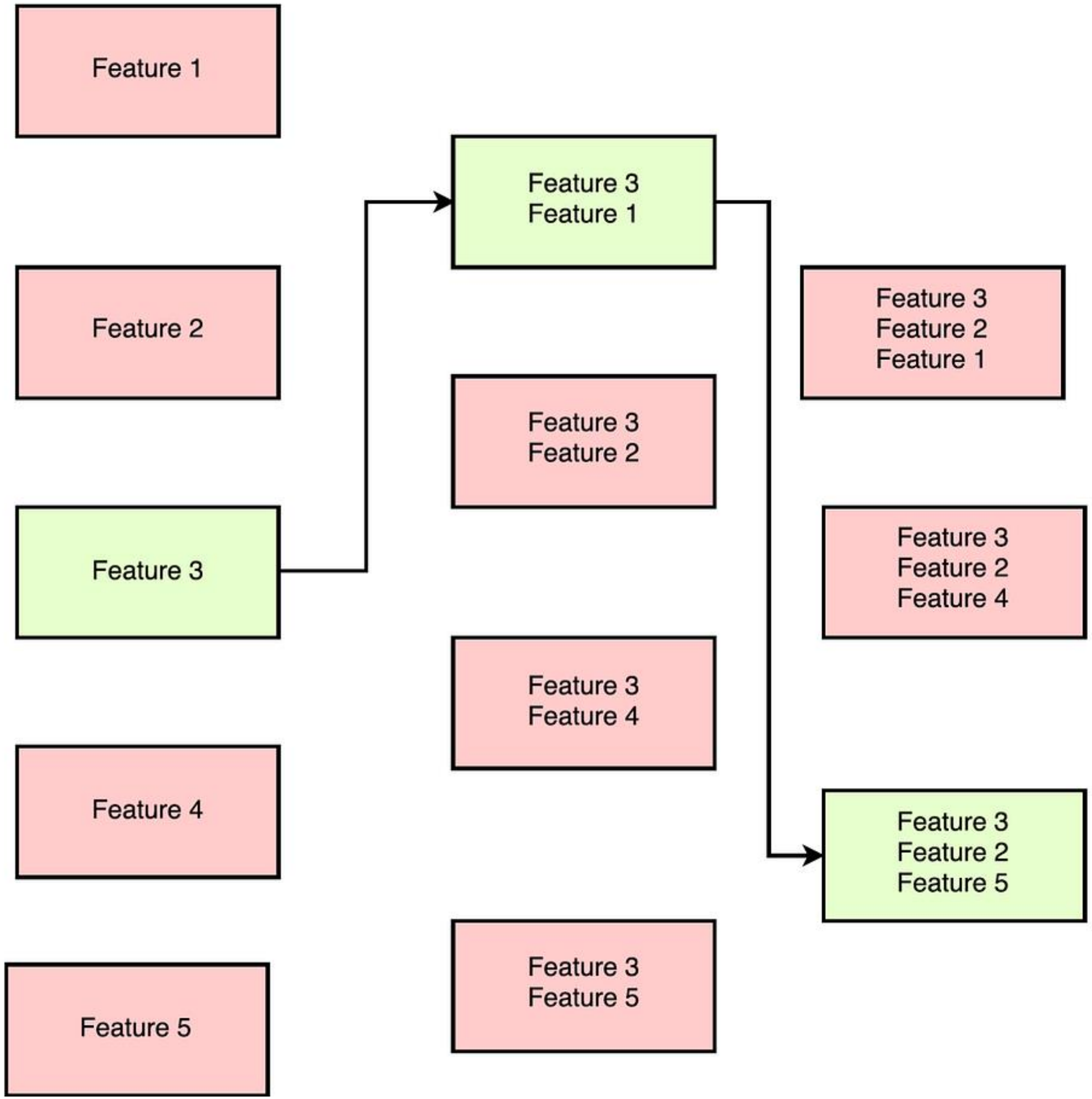
- **Wrapper methods:** iteratively select or eliminate a set of features using the model performance
  - Finds the optimal features subset for the desired ML algorithm
  - Detects the possible interactions between variables
  - Significant computation time when the number of variables is large: Wrappers are terribly slow for large datasets
- Underlying methods:
  - Forward Feature Selection
  - Backward Feature Elimination
  - Others



Forward Selection (FS)

- Forward Search – $O(n^2 \cdot \text{learning/testing time})$ – Greedy:
  - It is an iterative method in which we start with having no feature in the model (empty set)
  - In each iteration, we *keep adding the feature* which best improves our model till an addition of a new variable does not improve the performance of the model
  - Steps:
    1. *Score each feature by itself* and add the best feature to the initially empty set Feature Set (FS)
    2. Try each subset consisting of the *current FS plus one remaining feature* and add the best feature to FS
    3. *Continue until stop getting* significant improvement
- Forward search could miss important higher order combinations.

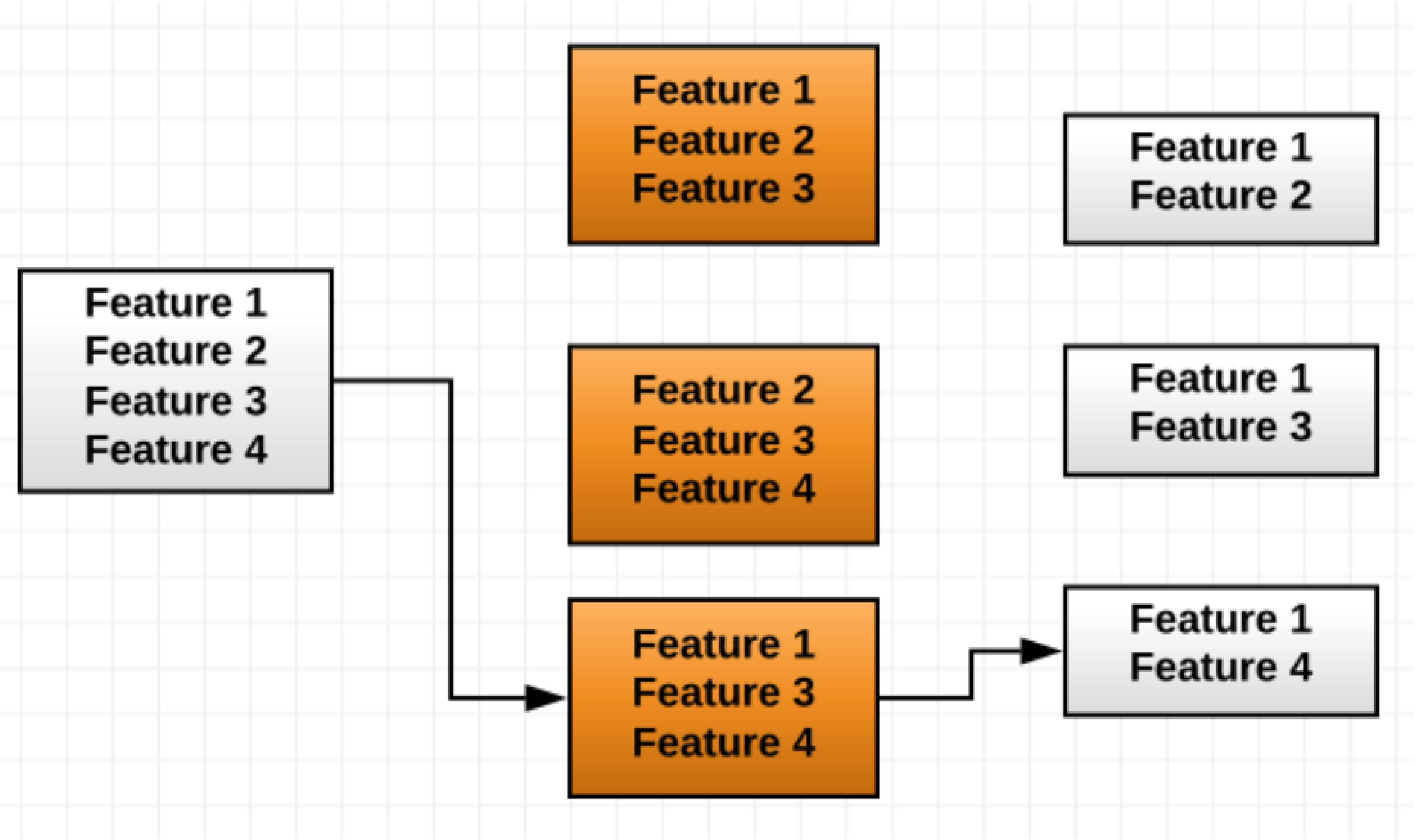
Forward Feature Selection



Backward Elimination

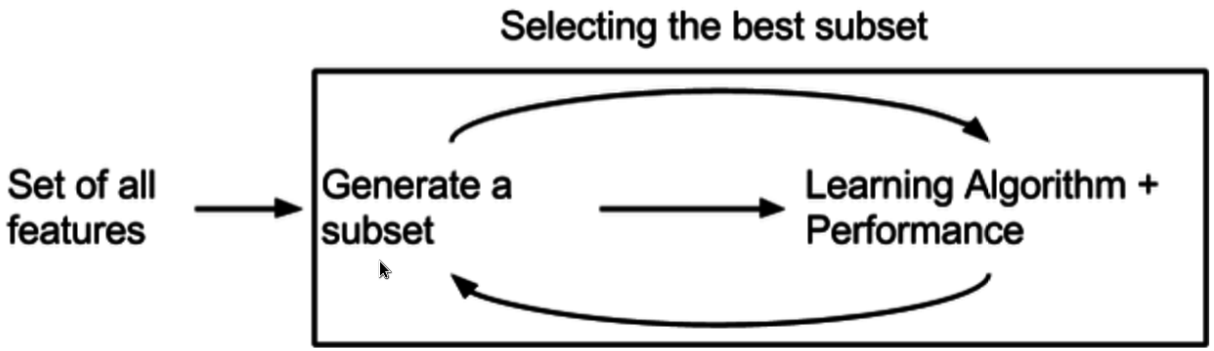
- Backward Search – $O(n^2 \cdot \text{learning/testing time})$ Greedy
  - In backward elimination, we start with all the features and remove the *least significant feature* at each iteration which improves the performance of the model. We repeat until no improvement is observed on removal of features
  - Steps:
    1. Score the initial complete FS
    2. Try each subset consisting of the current FS minus one feature in FS and drop the feature from FS causing least decrease in accuracy
    3. Continue until begin to get significant decreases in accuracy
- Backward search is more robust to the higher order problem, since accuracy will drop if we drop an important feature

Backward Feature Elimination



Embedded Feature Selection

- Perform feature selection as part of the model construction process



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

Dimensionality Reduction

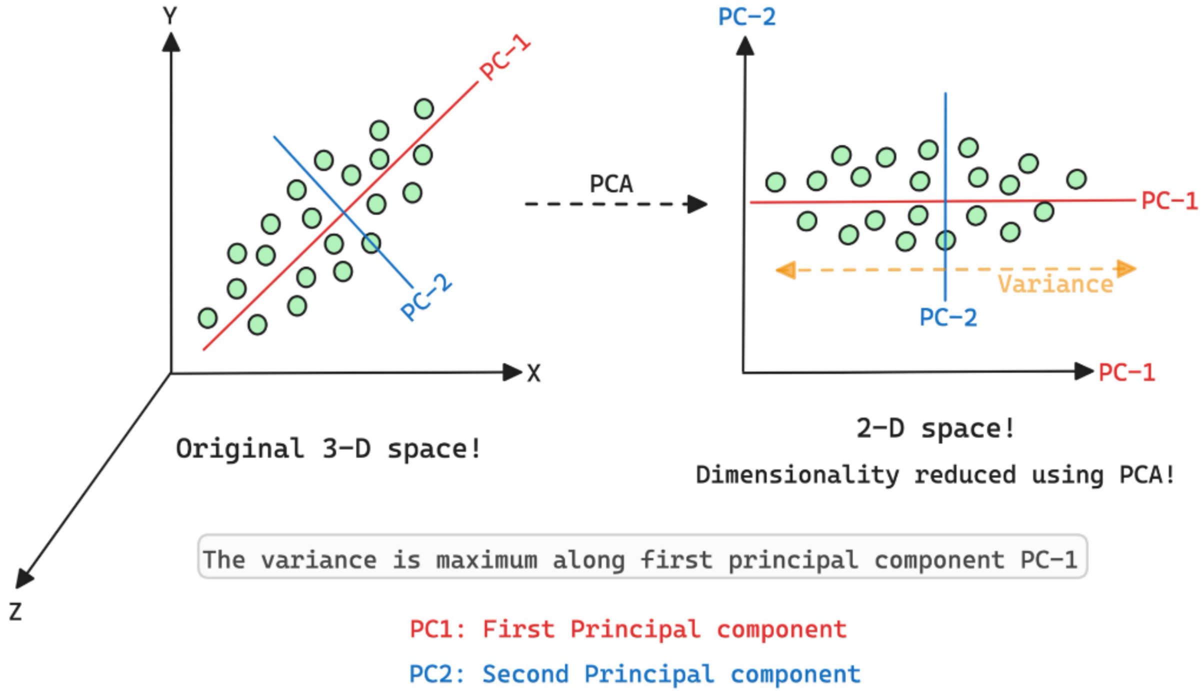
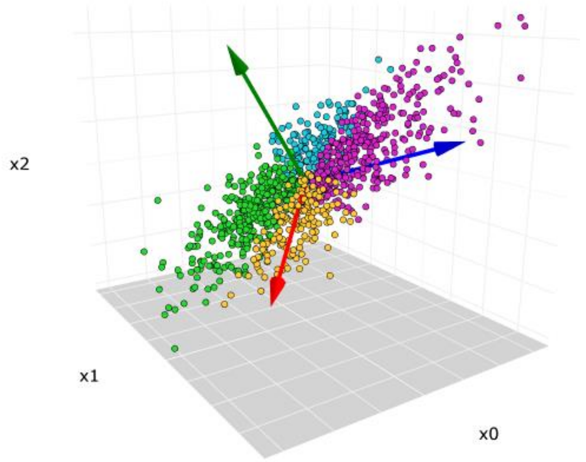


Image [source]

Dimensionality Reduction

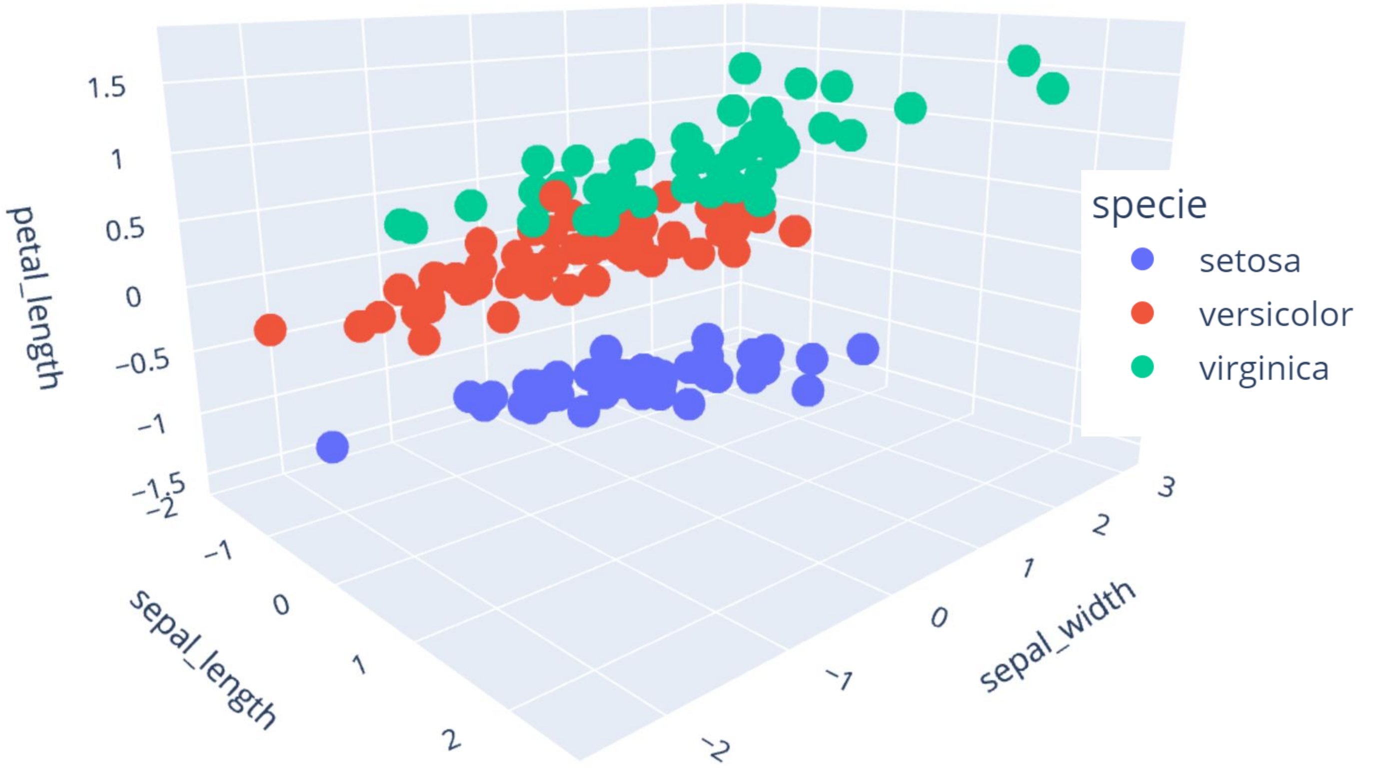
- **Dimensionality reduction** is used to reduce the number of features (dimensions) in a dataset while **preserving** the essential information **(i.e., retaining as much variability as possible)**
  - Analogous to summarizing the most important points of 1000-pages book in just 2 or 3 pages
  - It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

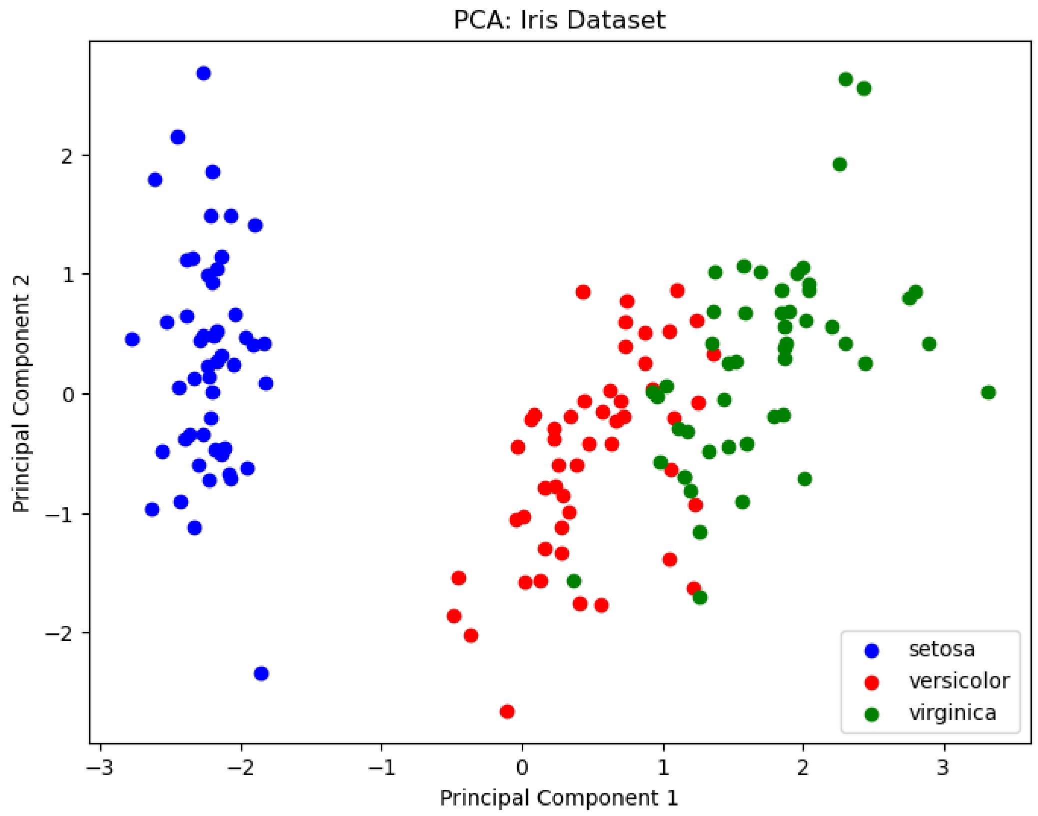
- One common method for dimensionality reduction is Principal Component Analysis (PCA)
  - PCA identifies the directions (principal components) along which the data varies the most (i.e., eigenvectors of covariance matrix)
  - Think of these "directions" as the main axes along which your data points 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

 `04.dp\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