Introduction to Machine Learning for Life Sciences

Course Outline

Day 1 | Day 2 | Day 3 | Day 4

Data Processing & Feature Engineering

- Python refresher: NumPy, pandas, scikit-learn
- Data exploration & preprocessing
- Feature extraction & selection techniques

Supervised & Unsupervised Learning

- Linear & logistic regression, decision trees
- Ensemble methods: Random Forest, XGBoost
- Clustering: K-means, DBSCAN
- Hands-on model training in Google Colab

Deep Learning

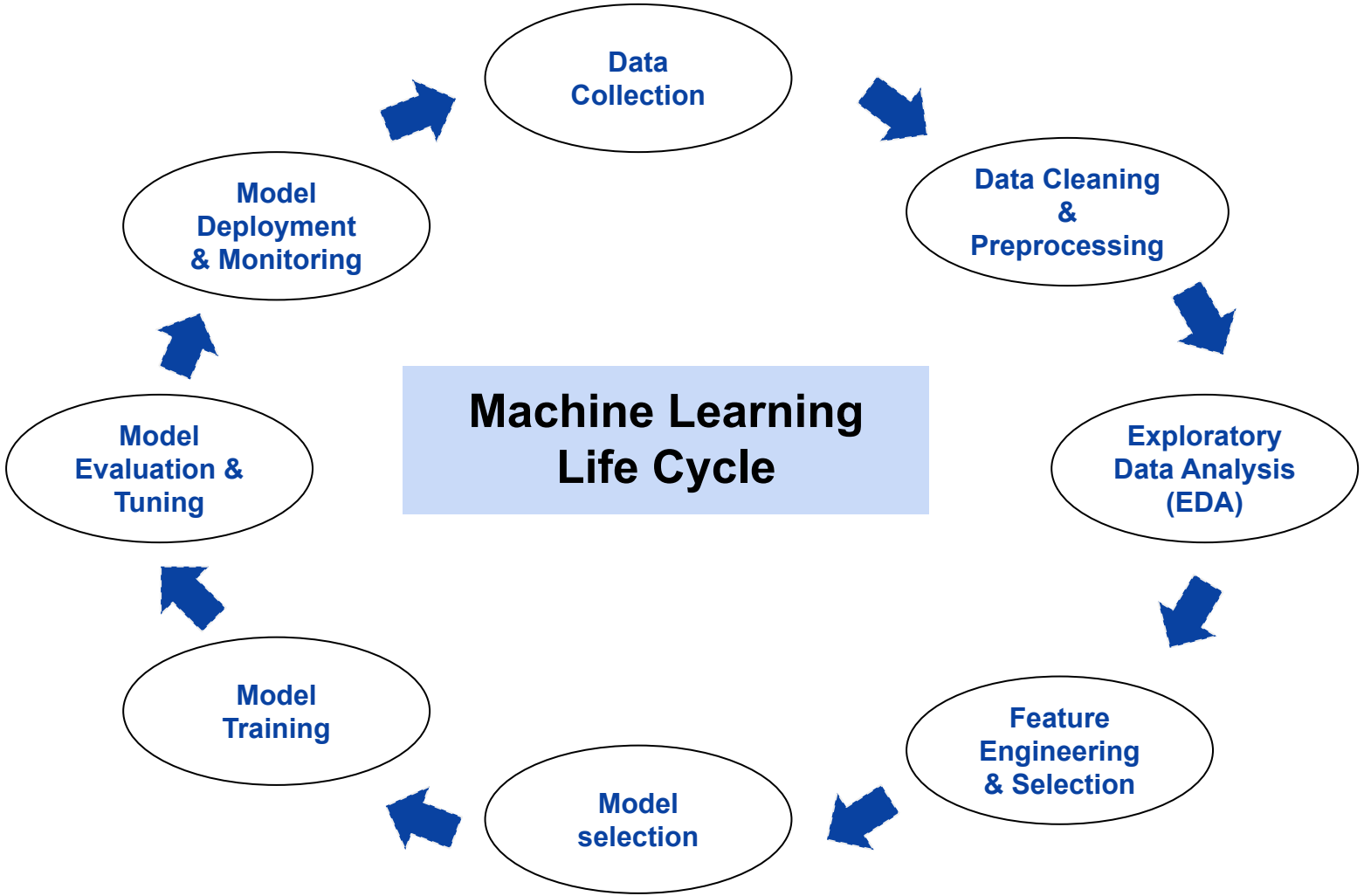
- Neural networks & CNNs explained
- PyTorch for image & tabular data classification
- Model interpretability with SHAP

Large Language Models (LLMs)

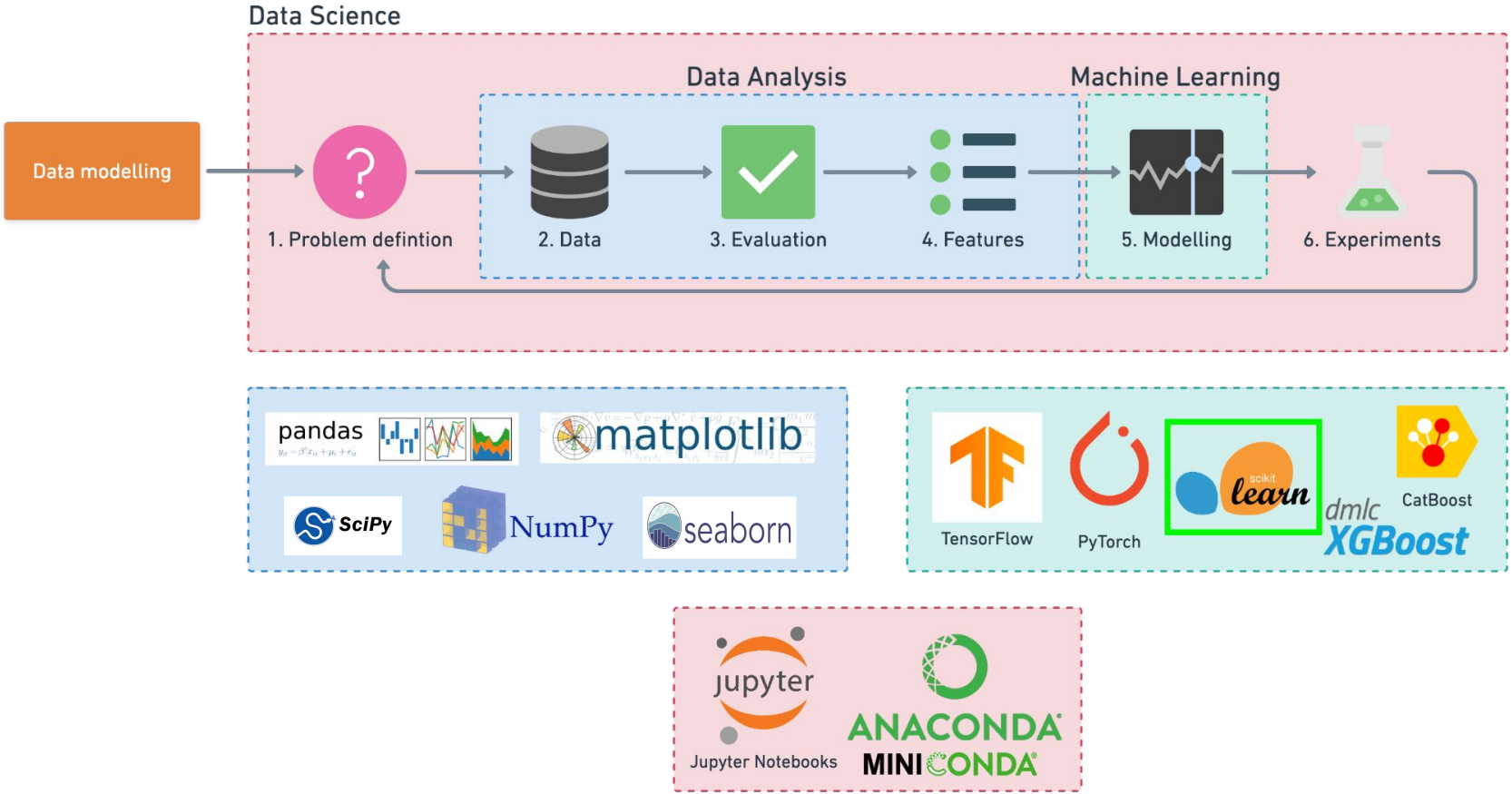
- Transformer architecture & attention mechanisms
- Practical demo with Hugging Face Transformers
- Fine-tuning LLMs

Day 1– *Introduction to Machine Learning for Life Sciences*

Data Processing &Feature Engineering

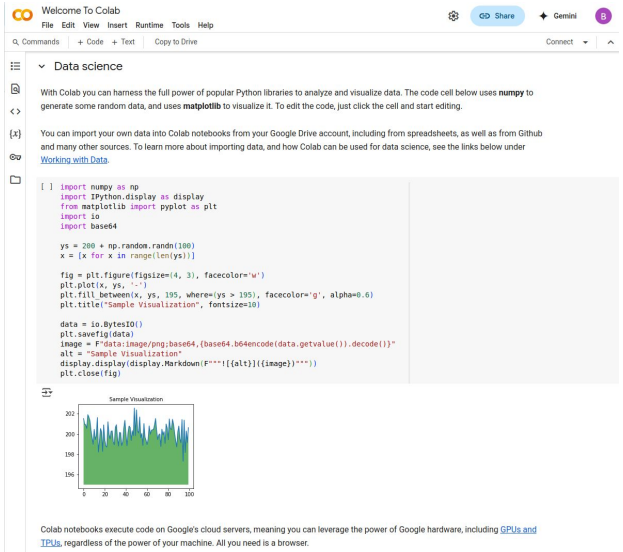

Machine Learning Life Cycle
Data Collection
Data Cleaning & Preprocessing
Exploratory Data Analysis (EDA)
Feature Engineering & Selection
Model selection
Model Training
Model Evaluation & Tuning
Model Deployment & Monitoring

Tools you can use



Jupyter Notebooks

- Interactive coding tool, mostly for Python
- Combine code + text + math + visuals
- Great for data science and analysis
- Runs locally



Google Colab

- Google's online version of Jupyter Notebook
- Runs in the cloud (no setup needed)
- Free GPU/TPU access
- Google Drive integration
- Real-time collaboration like Google Docs

https://github.com/bsc-life/ml4br-ml-course/

NumPy :

Numerical Python

1. What is Numpy ? why?
2. ndarray object
3. Creating, indexing & slicing
4. Reshaping
5. Iterating
6. Filtering

What is NumPy?

NumPy (Numerical Python) is a powerful library for numerical computing in Python.

Why Use NumPy over Lists?

- **Element Overhead:** Lists store extra metadata per element, increasing memory usage.
- **Datatype:** Mixed data types reduce efficiency.
- **Memory Fragmentation:** Non-contiguous storage can cause fragmentation.
- **Performance:** list are Slower for numerical tasks due to lack of optimization.
- **Functionality:** Lacks specialized tools for numerical operations.

Install NumPy using:

```
pip install numpy
```

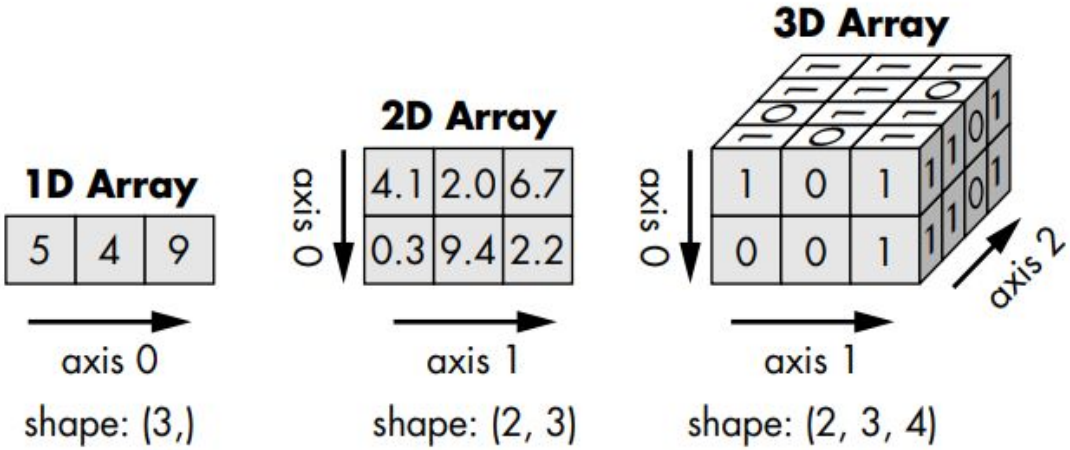
Importing NumPy :

```
import numpy as np
```

NumPy

NumPy Arrays

- NumPy's main object is the ndarray (N-dimensional array)
- Stores data in a grid-like structure (rows, columns, etc.)
- Can have any number of dimensions (1D, 2D, 3D, ...) with any length

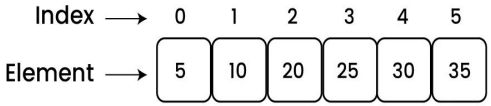


NumPy Arrays :Creating Array

We can create a NumPy ndarray object by using the array() function.

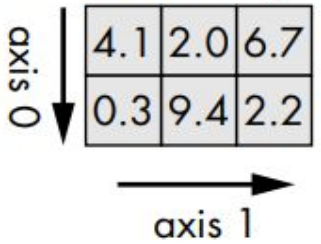
Creating 1D arrays from lists:

```
my_list= [5,10,20,25,30,35]
array= np.array(my_list)
```



Creating 2D arrays from lists:

```
my_list_of_list=[[4.1 , 2.0 , 6.7],[0.3 , 9.4 , 2.2]]
array2= np.array(my_list_of_list)
```



Remember : Numpy Array can contain only a single data type !!!

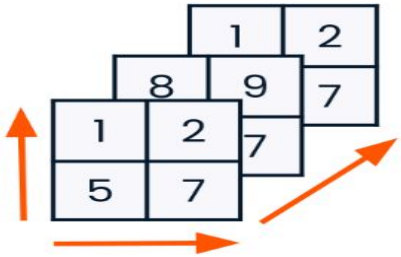
NumPy

NumPy Arrays :Creating Array

Creating 3D arrays from lists:

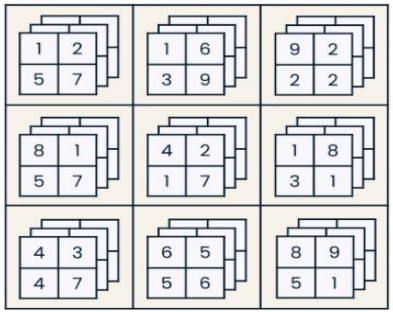
we can create a 3D array by creating a list of lists of lists.

```
my_list= [[[1,2],[5,7]],[[8,9],[6,7]],[[1,2],[3,7]]]
array= np.array(my_list)
```



Creating 4D arrays from lists:

we can create a 4D array by creating an array of 3D arrays..



How to check dimension ?

The ndim attribute returns an integer that tells us how many dimensions the array have.

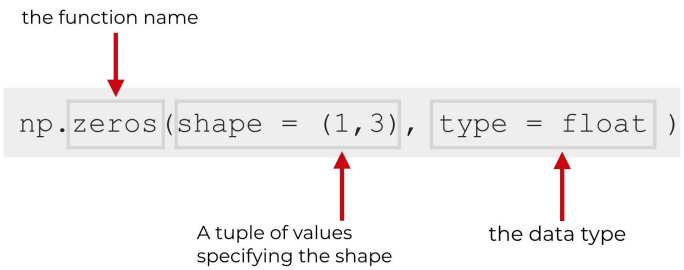
```
print(array2.ndim)
```

NumPy

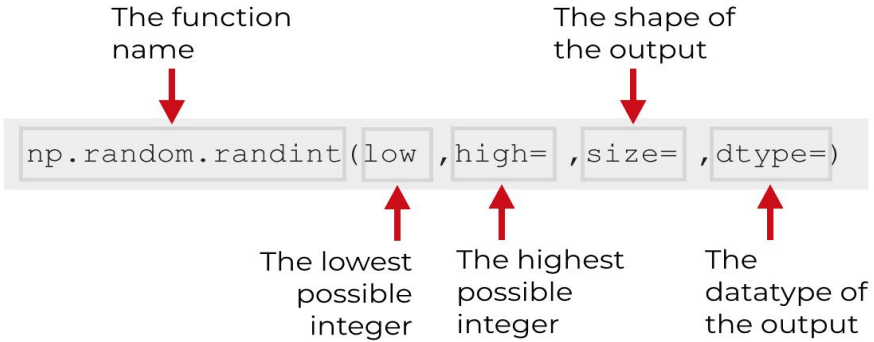
NumPy Arrays :Creating Array

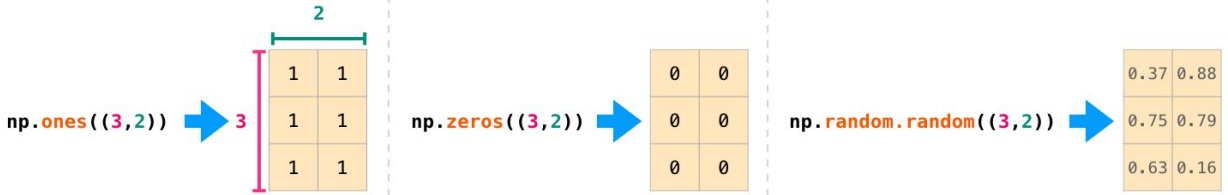
There are many Numpy functions to create array, including :

1. **np.zeros ( )**



2 .np.random.randint ( )

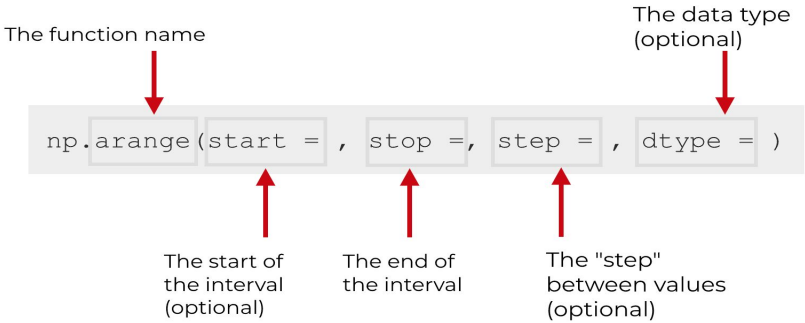


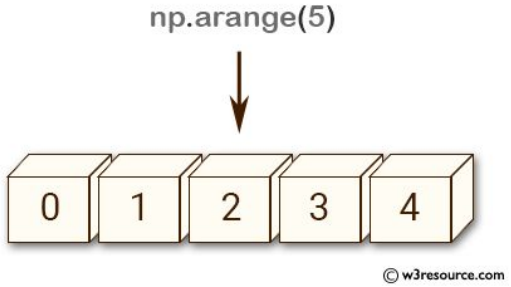


NumPy

NumPy Arrays :Creating Array

3. np.arange ( )

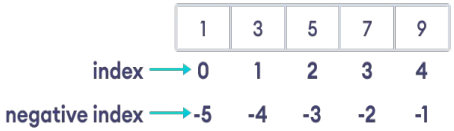


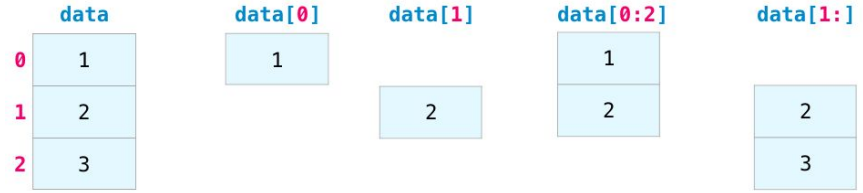


NumPy Arrays : Indexing

Indexing is an order-based method for accessing data.

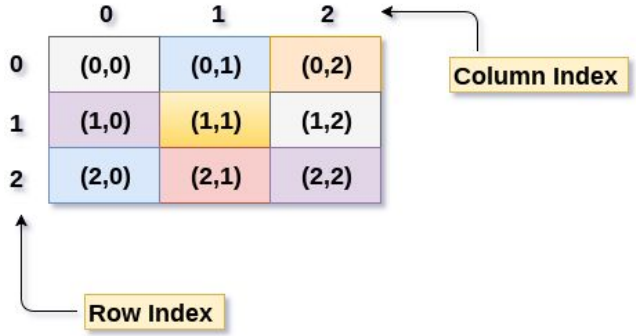
Indexing 1D:

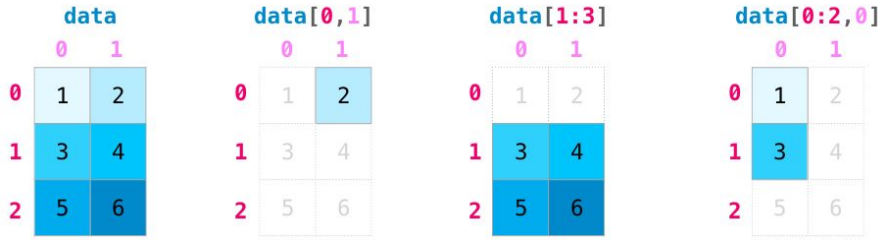




Indexing 2D:

use comma separated integers representing the dimension and the index of the element.

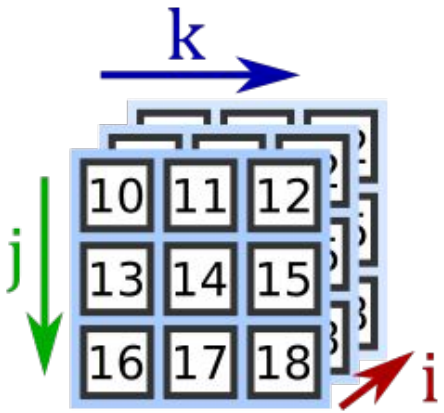




NumPy

NumPy Arrays : Indexing

Indexing 3D:



Negative Indexing:

Use negative indexing to access an array from the end.

Positive indexing

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Negative indexing

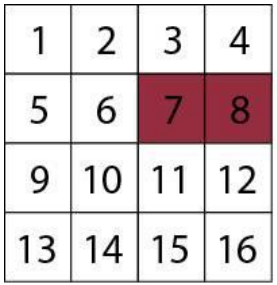
	-4	-3	-2	-1
-3	1	2	3	4
-2	5	6	7	8
-1	9	10	11	12

NumPy

NumPy Arrays : Slicing

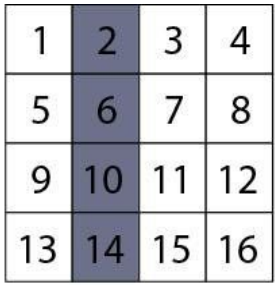
Extracts a subset of data based on given indices from one array and creates a new array with the sliced data.

We pass slice instead of index like this: [*start:end*]



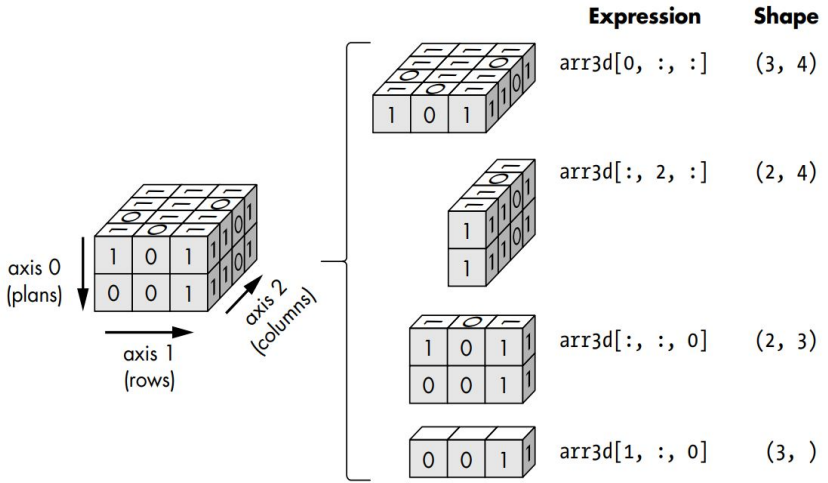
Row one, columns two to four

```
>>> arr[1, 2:4]
array([7, 8])
```



All rows in column one

```
>>> arr[:, 1]
array([2,  6, 10, 14])
```



NumPy

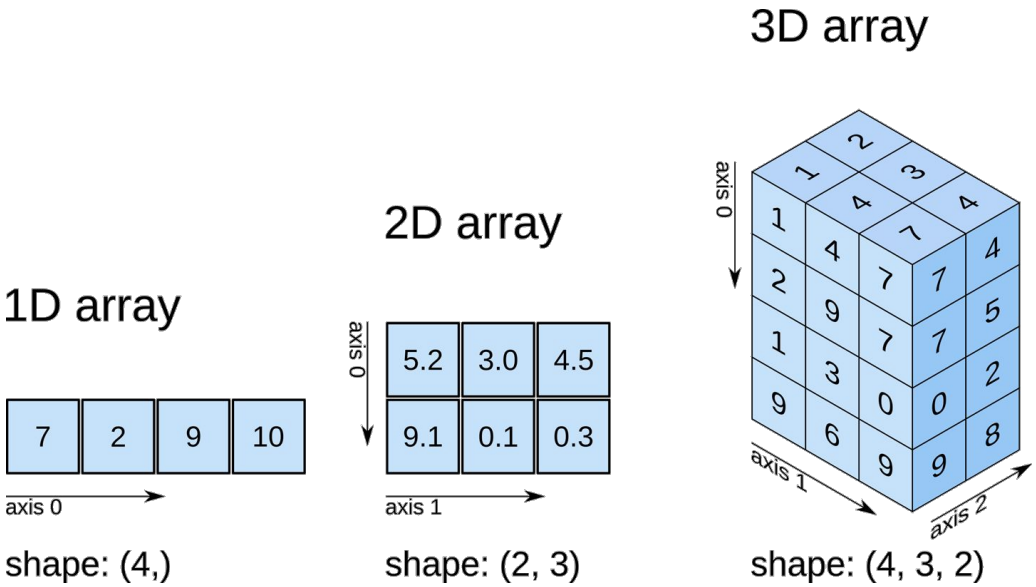
NumPy Arrays : Shape

Number of elements in each dimension

The function name

```
np.shape(input_array)
```

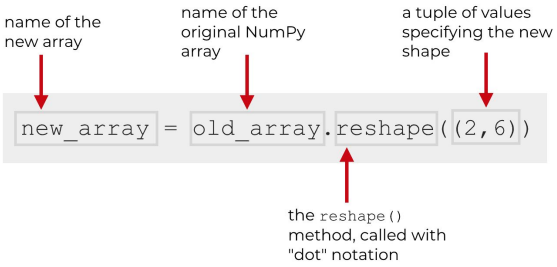
The array from which you want to retrieve the shape

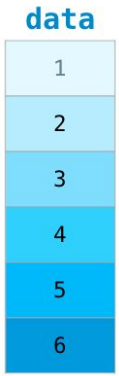


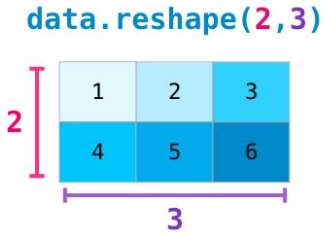
NumPy

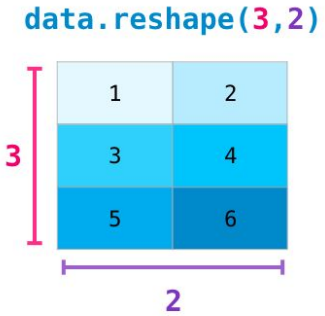
NumPy Arrays : Reshaping

Changing the shape of an array by adding or removing dimensions or change number of elements in each dimension.







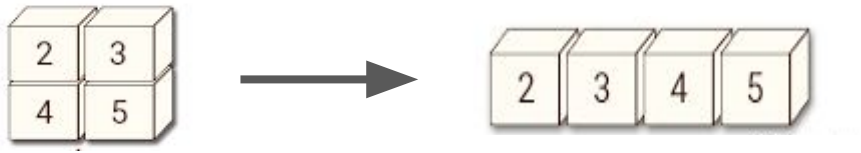


!!!! Only one unknown dimension is allowed & we pass -1 as the value for that dimension.

NumPy : Flattening Array

converting a multidimensional array into a 1D array. We can use:

1. **np.reshape (-1)**
2. **np.flatten ( )**



NumPy

Numpy : Array Iterating

Iterating means going through elements one by one.

1. **Using Basic For Loops**

1D array:

```
for x in arr:
  print(x)
```

2D array:

```
for x in arr:
  for y in x:
    print(y)
```

3D array:

```
for x in arr:
  for y in x:
    for z in y:
          print(z)
```

2. **Using np.nditer ( )**

We can define order of iteration (row-major order 'C' or column-major order 'F')

2. **Using np.enumerate ( )**

To access both the index and value during iteration.

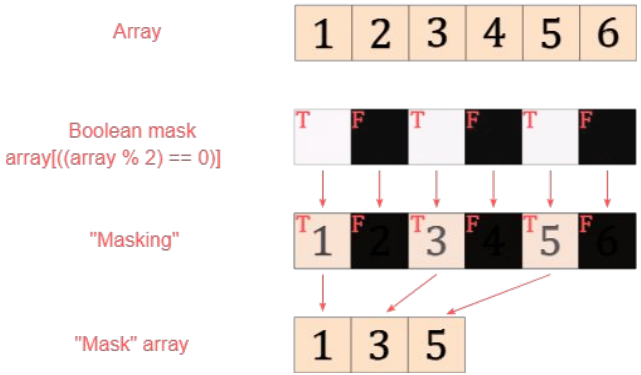
NumPy

Numpy : Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them

1. **Masks & indexing**

   Using condition for filtering the array & Create a new array with that filtering function



2. **np.where (condition)**

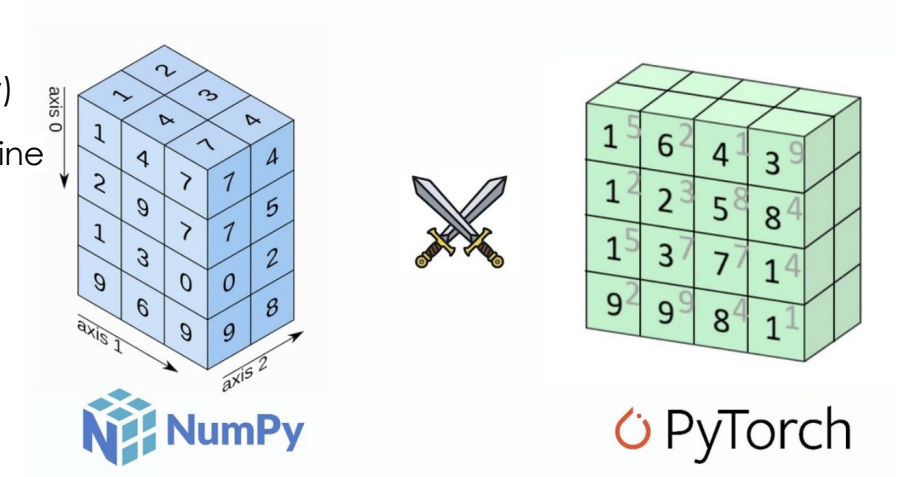
   search an array for a certain value, and return the indexes that get a match

NumPy

PyTorch vs NumPy

Numpy :

- Array Based (ndarray)
- used in typical machine learning algorithms
- Run on CPUs



Pytorch :

- n-dimensional array called Tensor
- Run on GPUs
- used in deep learning for **heavy matrix computation**.
- Automatic differentiation for building and training neural networks

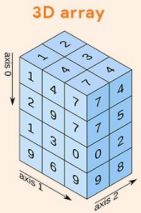
NumPy

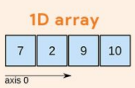
NumPy

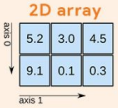
NumPy — import numpy as np

```
np.array(
[7, 2, 9, 10]
)
```

```
np.array(
[[5.2, 3.0, 4.5],
[9.1, 0.1, 0.3]]
)
```







shape: (4,) | shape: (2, 3) | shape: (4, 3, 2)

Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2X2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types
Standard double-precision floating point
Complex numbers represented by 128 floats
Boolean type storing `TRUE` and `FALSE` values
Python object type
Fixed-length string type
Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
 array([[-0.5,  0. ,  0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
 array([[ 2.5,  4. ,  6. ],
        [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
 array([[ 0.66666667,  1.        ,  1.        ],
        [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
 array([[  1.5,   4. ,   9. ],
        [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
 array([[ 7.,  7.],
        [ 7.,  7.]])
```

Subtraction
Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
 array([[False,  True,  True],
        [False, False, False]], dtype=bool)
>>> a < 2
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
 3
>>> b[1,2]
 6.0
```

Select the element at the 2nd index
Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

```
>>> a[0:2]
 array([1, 2])
>>> b[0:2,1]
 array([ 2.,  5.])
>>> b[:1]
 array([[1.5, 2., 3.]])
>>> c[1,...]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]
 array([3, 2, 1])
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to `b[0:1, :]`)
Same as `[1,:,:]`
Reversed array `a`

Boolean Indexing

```
>>> a[a<2]
 array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
 array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
 array([[ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5],
        [ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5]])
```

Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

Pandas :

Python Data Analysis

1. What is pandas? why?
2. Pandas series & DataFrame
3. Loading and viewing Data
4. Cleaning DataFrames
5. Correlation

Pandas

Pandas

"Panel Data", and "Python Data Analysis" is a Python library used for working with data sets.

Why Use Pandas?

1. Including functions for analyzing, cleaning, exploring, and manipulating data.
2. Allow to analyze big data and make conclusions based on statistical theories.
3. clean messy data sets, and make them readable and relevant.

Install

```
pip install pandas
```

Import

```
Import pandas as pd
```

Panda Series

The **Series** object represents one-dimensional data structures in Pandas.

A series consists of two components.

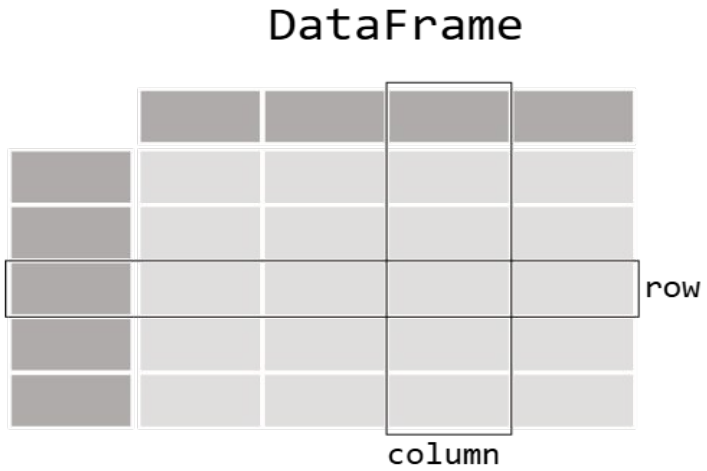
- One-dimensional data (Values)
- Index

pandas

DataFrames

Datasets in Pandas are multi-dimensional tables called **DataFrames**.

Each **Series** is a column in Data Frames.



Creating DataFrames

```
data = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=data)
```

pandas

Load Files Into a DataFrame

```
df = pd.read_csv('data.csv')     #To read .CSV files
```

```
df = pd.read_json('data.json')   #To read .json Files
```

Pandas by default will only return the first 5 rows, and the last 5 rows.

To print the entire DataFrame:

```
Pd.options.display.max_rows
```

```
df.to_string()
```

Viewing Data

1. **.head( ) :** returns the headers and a specified number of rows, starting from the top.
2. **.tail( ) :** returns the headers and a specified number of rows, starting from the bottom.
3. **.shape :** returns a tuple containing the shape of the DataFrame. (No. rows, No. columns)
4. **.describe( ):** summary statistics for numerical columns, like mean and median.
5. **.info ( ):** Information about the dataset
6. **.isnull.sum( ):** to count the number of nulls in each column.
7. **.isnull.sum.sum ( ):** to count the number of nulls in DataFrame

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Subsetting Column

Subsetting the column:

```
df ["my_column1"]
```

Subsetting **multiple** column:

```
df [["my_column1","my_column1"]]
```

Subsetting Rows

- **Index based :** *Row/s can be fetched by passing in a boolean series with one/multiple True*

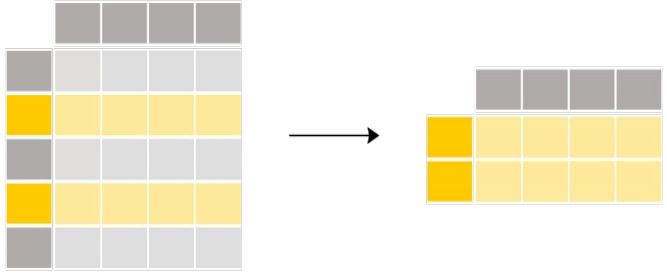
```
df[df.index== value]

df[df.index.isin(range(start_index,last_index))]
```

- **Conditional Expression:** *To select rows based on a conditional expression, use a condition inside the selection brackets* **[ ].**

```
df[df ["my_column1"] > condition ]
```

- **.isin( ) method:** *returns a True for each row the values are in the provided list.*

```
df[df ["my_column1"].isin([2,3])]
```



- **.notna( ) method :** *returns a True for each row the values are not a Null value.*

```
df[df ["my_column1"].notna()]
```

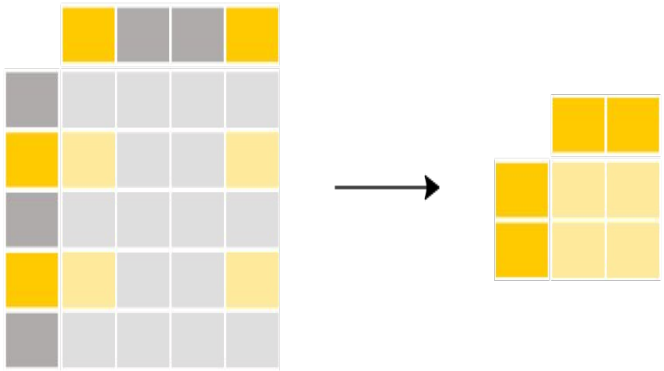
Subsetting Rows & Columns

- **.iloc[ ] method:** when using the positions in the table.

```
df.iloc[start_row:end_row,start_column:end_column]
```

- **.loc[ ] method:** Select specific rows and/or columns using loc when using the row and column names.

```
df.loc[condition for selecting rows, column_name]
```



Cleaning DataFrame

Data cleaning means fixing bad data in your data set.

Bad data could be:

- Missing values (Empty Cells)
- Wrong data
- Duplicates
- Column Renaming

Missing values

- **.dropna ( ) to remove Rows:** removing rows with Nan values.

```
new_df = df.dropna( )
```

dropna( ) returns a *new* DataFrame, and will not change the original

To change the original DataFrame, use the ***inplace = True*** argument:

```
df.dropna(inplace= True )
```

To remove the columns with one or more NaN values, use the ***axis = 1*** argument:

```
df.dropna(inplace= True,axis = 1 )
```

pandas

Missing Values

- **.fillna ( ) to replace values:** replace empty cells with a value.

1. Fill all nan values with a certain value

```
df.fillna(value,inplace=True )
```

2. Specific column with a certain value

```
df.fillna({“column_name”: value},inplace=True )
```

3. Mean, Median or Mode value of the specified column

```
x = df[“column_name”].mean()

df.fillna(x,inplace=True )
```

Duplicates

- **.duplicated method:** returns a True Boolean values for every row that is duplicated.

```
df.duplicated ()
```

- **.drop_duplicates () :** To remove duplicates.

```
df.drop_duplicated (inplace = True)
```

Wrong Values and Typos

It does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99" .

- **Replacing values:** set boundaries for values and replace it with a certain value if a cell falls out of boundary.

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

- **Removing Rows:** remove the rows that contains wrong data and not include in the analysis.

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Column Renaming

.rename() : you can use columns as an argument to rename specific columns.

```
df.rename(columns={“old_column1”: “new_renamed_column”},inplace =True)
```

Data Correlation

.corr( ) Method: Compute pairwise correlation of columns, excluding NA/null values.

```
df.corr()
```

Method of correlation:

- **pearson :** standard correlation coefficient

  The linear relationship between two variables

- **kendall :** Kendall Tau correlation coefficient

  Non-parametric test procedure when the data is not normally distributed and the two variables only have an ordinal scale level.

- **spearman :** Spearman rank correlation

  The non-parametric counterpart of Pearson's correlation.

Data Wrangling

with pandas Cheat Sheet
http://pandas.pydata.org

Pandas API Reference Pandas User Guide

Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

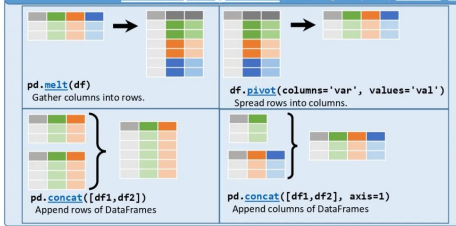
Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Reshaping Data – Change layout, sorting, reindexing, renaming



`pd.melt(df)`
Gather columns into rows.

`pd.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg', ascending=False)`
Order rows by values of a column (high to low).

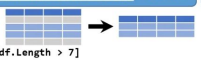
`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length', 'Height'])`
Drop columns from DataFrame

Subset Observations - rows



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)` Randomly select n rows.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

Subset Variables - columns

`df[['width', 'length', 'species']]`
Select multiple columns with specific names.

`df['width']` or `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]`
Select rows 10-20.

`df.iloc[:, [1, 2, 5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[:, 'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.loc[df['a'] > 10, ['a', 'c']]`
Select rows meeting logical condition, and only the specific columns .

`df.iat[1, 2]` Access single value by index

`df.at[4, 'A']` Access single value by label

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

Summarize Data

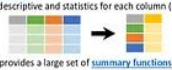
`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df.shape`
Tuple of # of rows, # of columns in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive and statistics for each column (or GroupBy).



pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

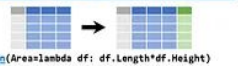
`sum()` Sum values of each object.
`count()` Count non-NA/null values of each object.
`median()` Median value of each object.
`quantile([0.25,0.75])` Quantiles of each object.
`apply(function)` Apply function to each object.
`min()` Minimum value in each object.
`max()` Maximum value in each object.
`mean()` Mean value of each object.
`var()` Variance of each object.
`std()` Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

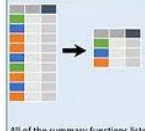
`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)` Element-wise max.
`min(axis=1)` Element-wise min.
`clip(lower=-10,upper=10)` Trim values at input thresholds
`abs()` Absolute value.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()` Size of each group.
`agg(function)` Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)` Copy with values shifted by 1.
`rank(method='dense')` Ranks with no gaps.
`rank(method='min')` Ranks. Ties get min rank.
`rank(pct=True)` Ranks rescaled to interval [0, 1].
`rank(method='first')` Ranks. Ties go to first value.
`shift(-1)` Copy with values lagged by 1.
`cumsum()` Cumulative sum.
`cummax()` Cumulative max.
`cummin()` Cumulative min.
`cumprod()` Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column

`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points

Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

bdf

x1	x3
A	T
B	F
D	T

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

zdf

x1	x2
B	2
C	3
D	4

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Matplotlib:

Matlab,Plot,and Library

1. What is Matplotlib? why?
2. Axes interface
3. Pyplot interface
4. Creating subplots

What is Matplotlib?

Matplotlib is a graph plotting library in python that serves as a visualization utility.

Matplotlib has two interfaces:

1. **Axes interface (object-based, explicit)**
   create a **Figure** and one or more **Axes** objects, then *explicitly* use methods on these objects to add data, configure limits, set labels etc.

2. **pyplot interface (function-based, implicit)**
   consists of functions in the **pyplot** module. Figure and Axes are manipulated through these functions and are only *implicitly* present in the background.

matplotlib

Axes Interface

Install :

```
pip install matplotlib
```

Import:

```
import matplotlib
```

Axes allow placement of plots at any location in the figure. A given figure can contain many axes.

API:

- ***Figure***: for figure-level method
- ***axes***: add data, limits, labels etc.
- ***subplots***: create Figure and Axes

```
fig= plt.figure()

ax1 =fig.add_subplot(1,1,1)

ax.plot()

plt.show()
```

matplotlib

.axes cheatsheet

- **plot(x, y) :** Generate y vs x graph
- **set_xlabel() :** Label for the X-axis
- **set_ylabel() :** Label for the Y-axis
- **set_title() :** Title of the plot
- **legend() :** Generate legend for the graph
- **hist() :** Generate histogram plot
- **scatter()**: Generate scatter plot

matplotlib

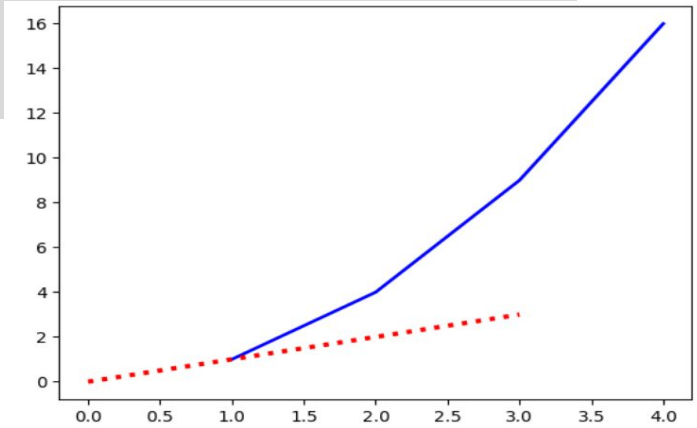
Pyplot Interface

```
import  matplotlib.pyplot as plt
```

plt.plot() method : create a line plot where the 1st argument is the **x** variable and the 2nd argument is the **y** variable.

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16],ls= "-",color="Blue",lw=2)
plt.plot(np.arange(4),ls= ":",color="red",lw=3)
plt.savefig("myfigure")
plt.show()
```

Linestyle or **ls :** string argument
Color or **c** **:** string argument
Linewidth or **lw :** float argument
marker : string argument to mark points



.plt cheatsheet

Add **title** to figure : plt.title()
Add **label** to **x_axis** : plt.xlabel()
Add **label** to **y_axis** : plt.ylabel()
Add **Text** to the figure : plt.text ()
Define **axis limits** : plt.axis() or plt.xlim() & plt.ylim()
Add **grid** : plt.grid(True)
Annotate figure : plt.annotate()
Axis scale :

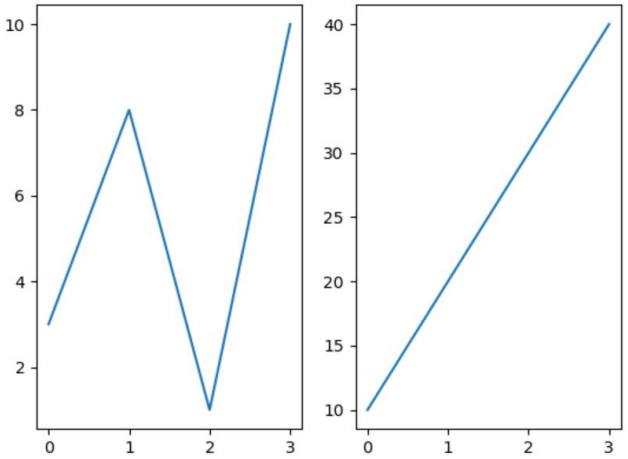
- *Logarithmic* plt.yscale("log")
- *Linear* plt.yscale("linear")
- *Symmetric log* plt.yscale("symlog")
- *Logit* plt.yscale("logit")

Add **legend**: plt.legend( )

matplotlib

Creating subplots

With the ***plt.subplot ( )*** function we can draw multiple plots in one figure.



3 arguments that describes the layout of the figure :

- ***1st*** argument as the number of ***rows***
- ***2nd*** argument as the number of ***columns***
- ***3rd*** argument as ***index*** of ***current plot***

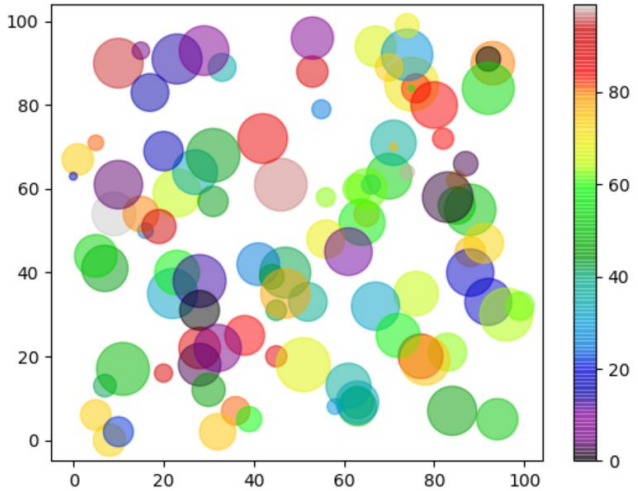
```
plt.subplots(1,2,1)

#the figure has 1 row, 2 columns, and this plot is the first plot.
```

matplotlib

plt.scatter

scatter() function to draw a scatter plot as one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

- multiple scatter plot in one figure by calling before plt.show()
- define the color for each point by setting ***c argument***
- define the size for each point by setting ***s argument***
- Define transparency of dots by setting ***alpha argument***
- Using a colormap by setting the ***cmp argument*** and include in the plot by calling ***plt.colorbar()***



matplotlib

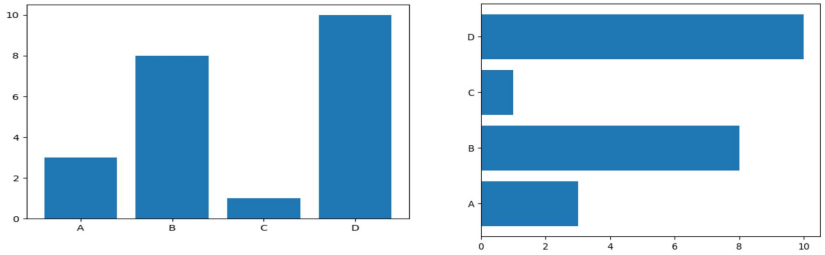
plt.bar

the ***bar()*** function to draw bar graphs for categorical data.

1. **x -axis** as the 1st arguments : **categories**
2. **Y-axis** as the second categories: **values**

use the ***barh()*** function for horizontal bars.

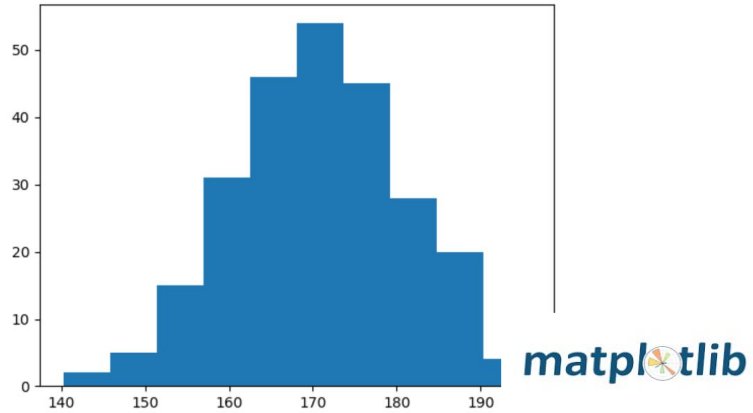
- **Color keyword** to define colors as strings or Hexadecimal color values
- For bar() : ***width*** to set width of the bars
- For barh(): ***height*** to set height of the bars



plt.hist

The ***hist()*** create a histogram graph showing *frequency* distributions.

It is a graph showing the number of observations within each given interval.



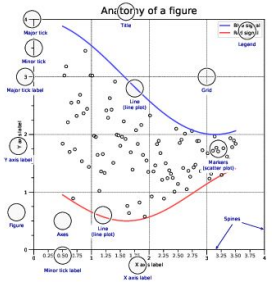
matplotlib

Cheat sheet — Version 3.7.4

Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows, cols, …)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols, …)
ax = G[0, :]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X], Y, [fmt], …)
X, Y, fmt, color, marker, linestyle

scatter(X, Y, …)
X, Y, [s]izes, [c]olors, marker, cmap

bar[h](x, height, …)
x, height, width, bottom, align, color

imshow(Z, …)
Z, cmap, interpolation, extent, origin

contour[f]([X], [Y], Z, …)
X, Y, Z, levels, colors, extent, origin

pcolormesh([X], [Y], Z, …)
X, Y, Z, vmin, vmax, cmap

quiver([X], [Y], U, V, …)
X, Y, U, V, C, units, angles

pie(X, …)
Z, explode, labels, colors, radius

text(x, y, text, …)
x, y, text, va, ha, size, weight, transform

fill[_between][x](…)
X, Y1, Y2, color, where
```

Advanced plots

```
step(X, Y, [fmt], …)
X, Y, fmt, color, marker, where

boxplot(X, …)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr, …)
X, Y, xerr, yerr, fmt

hist(X, bins, …)
X, bins, range, density, weights

violinplot(D, …)
D, positions, widths, vert

barbs([X], [Y], U, V, …)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions, …)
positions, orientation, lineoffsets

hexbin(X, Y, C, …)
X, Y, C, gridsize, bins
```

Scales

```
ax.set_[xy]scale(scale, …)
linear  any values
log     values > 0
symlog  any values
logit   0 < values < 1
```

Projections

```
subplot(…, projection=p)
p='polar'
p='3d'
p=ccrs.Orthographic()
import cartopy.crs as ccrs
```

Lines

```
linestyle or ls
"-" ":" "--" "-." (0,(0.01,2))
capstyle or dash_capstyle
"butt" "round" "projecting"
```

Markers

```
markevery
10  [0, -1]  (25, 5)  [0, 25, -1]
```

Colors

```
'Cn'
'x'
'name'
(R,G,B[,A])
'#RRGGBB[AA]'
'x,y'
```

Colormaps

```
plt.get_cmap(name)
```

Uniform: viridis, magma, plasma

Sequential: Greys, YlOrBr, Wistia

Diverging: Spectral, coolwarm, RdGy

Qualitative: tab10, tab20

Cyclic: twilight

Tick locators

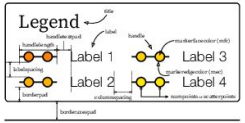
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
handles, labels, loc, title, frameon
```




```
ax.colorbar(…)
mappable, ax, cax, orientation

ax.annotate(…)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

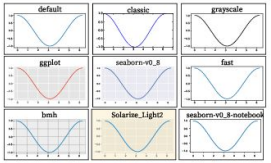
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s Save	ctrl + w Close plot
r Reset view	f Fullscreen 0/1
f View forward	b View back
p Pan view	o Zoom to rect
x X pan/zoom	y Y pan/zoom
g Minor grid 0/1	G Major grid 0/1
l X axis log/linear	L Y axis log/linear

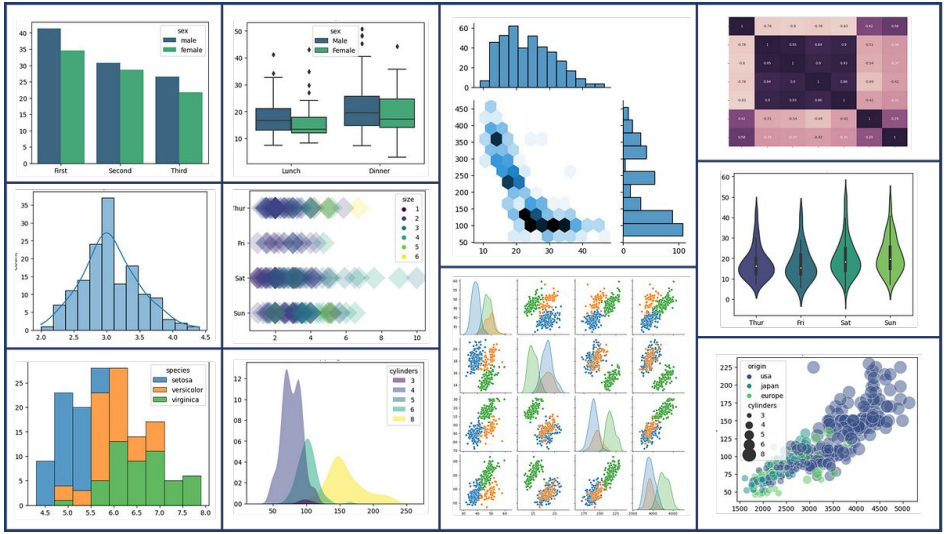
Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

Seaborn :

Statistical data visualization

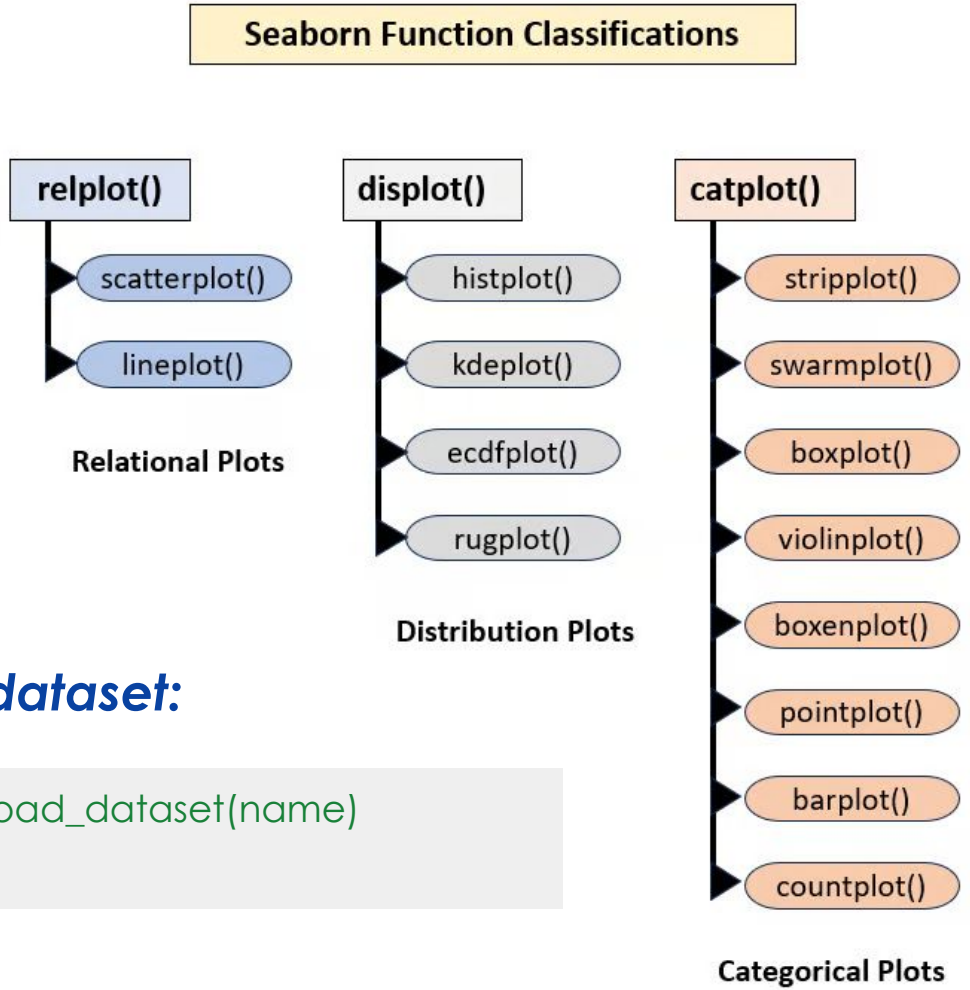
1. What is Seaborn
2. Seaborn vs. Matplotlib
3. Seaborn functions (relashional plots , Categorical Plot, Distribution plots)



Seaborn

Python data visualization library based on matplotlib.

- Easy to use
- Works well with **pandas**
- Built on top of **matplotlib**



Import :

```
Import seaborn as sns
Import pandas as pd
```

Load dataset:

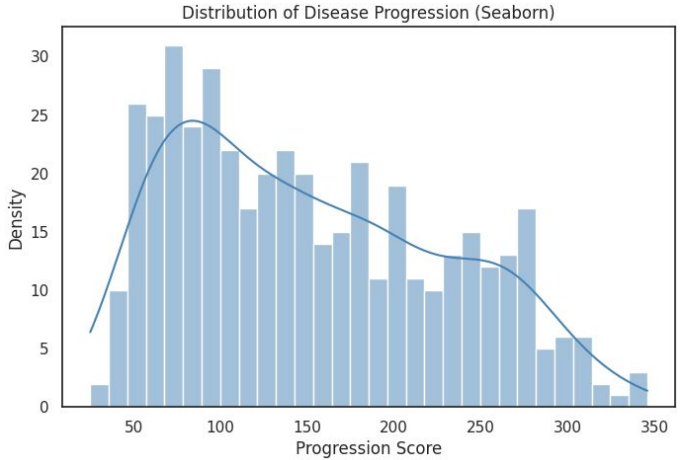
```
seaborn.load_dataset(name)
```

Seaborn vs matplotlib

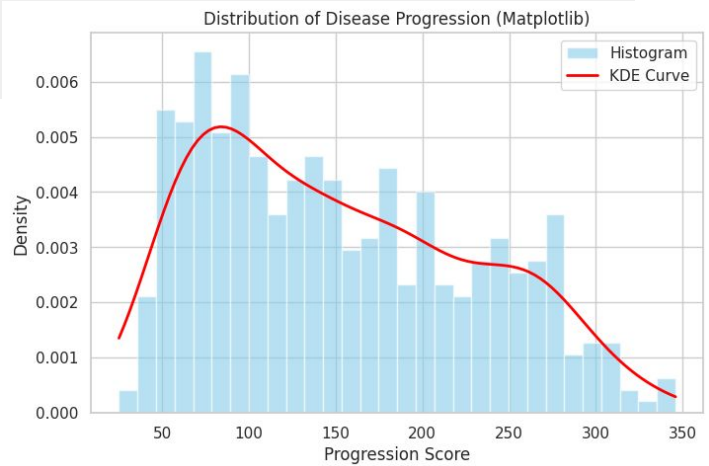
Aspect	Matplotlib	Seaborn
Type	Low-level plotting library	High-level interface for statistical graphics
Flexibility	Highly flexible, create almost any plot	Simplified interface for common statistical plots
Ease of Use	Steeper learning curve, more verbose code	Easier to use, minimal code
Data Handling	General data structures	Designed to work with Pandas DataFrames
Statistical Features	No built-in statistical functions	Includes built-in statistical functions
Default Styles	Limited default styles and color palettes	Offers a range of default styles and optimized color palettes
Customization	Requires manual customization for visual appeal	Visually appealing by default
Best Suited For	Highly customizable plots for presentations and publications	Statistical graphics and exploratory data analysis
Integration	Standalone library	Built on top of Matplotlib; can be used together

Seaborn vs matplotlib

```
import seaborn as sns
import pandas as pd
from sklearn.datasets import load_diabetes

# Load data
diabetes = load_diabetes()
df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
df['progression'] = diabetes.target

# Plot
sns.set(style="whitegrid")
plt.figure(figsize=(8, 5))
sns.histplot(df['progression'], kde=True, bins=30, color='steelblue')

plt.title("Distribution of Disease Progression (Seaborn)")
plt.xlabel("Progression Score")
plt.ylabel("Density")
plt.show()
```



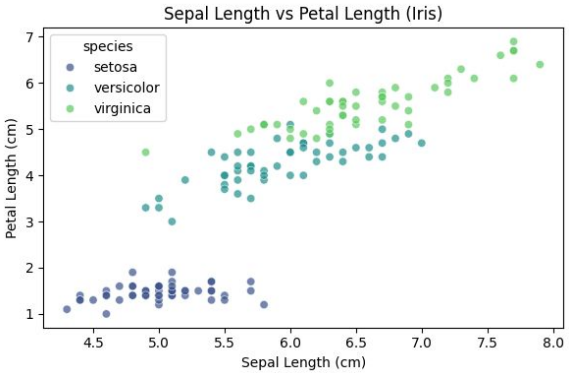

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
import numpy as np
from scipy.stats import gaussian_kde
diabetes = load_diabetes()                # Load data
y = diabetes.target
kde = gaussian_kde(y)                     # KDE estimation
x_vals = np.linspace(min(y), max(y), 100)
# Plot
plt.figure(figsize=(8, 5))
plt.hist(y, bins=30, density=True, alpha=0.6, color='skyblue',
label='Histogram')
plt.plot(x_vals, kde(x_vals), color='red', linewidth=2, label='KDE
Curve')
    plt.title("Distribution of Disease Progression (Matplotlib)")
plt.xlabel("Progression Score")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



Relational Plot (relplot)

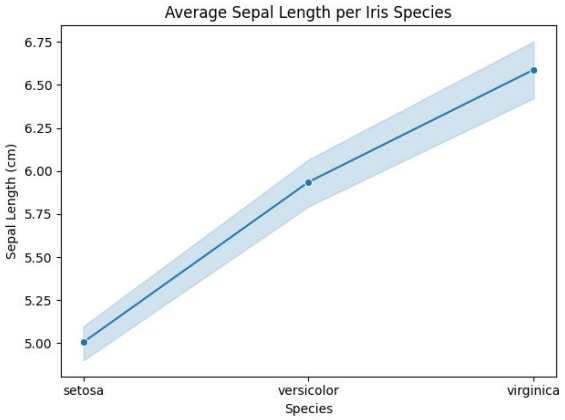
1. *Scatterplot*

Shows the relationship between two numeric variables.Helps you see correlation, clusters, outliers, or trends.



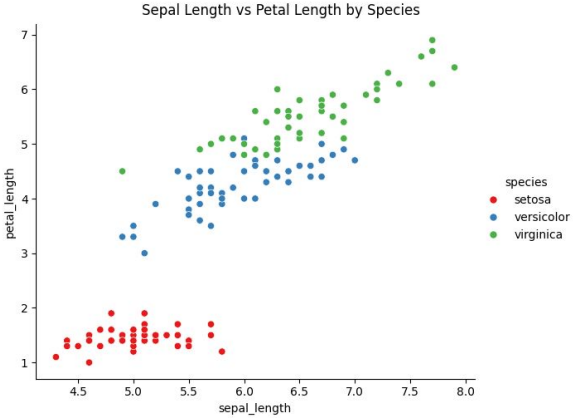
2.LinePlot

Displays a continuous trend over time or order.



3.replot

show the relationship between two continuous variables.

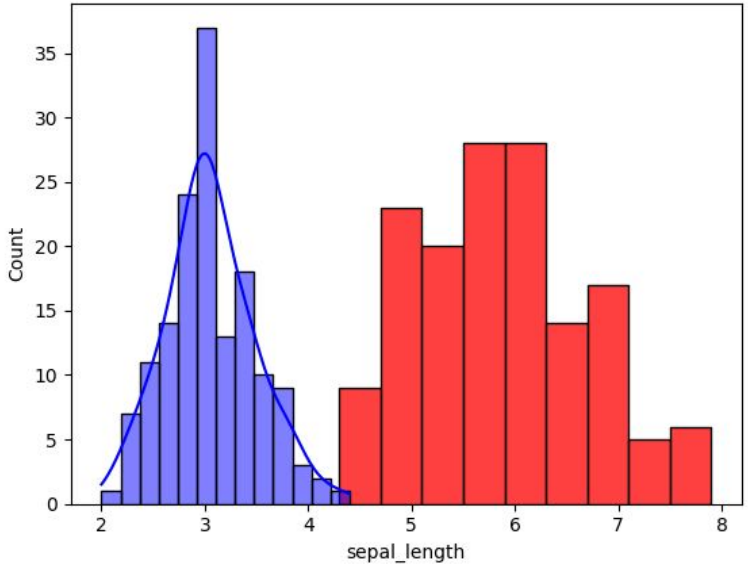


seaborn

Distribution Plot (displot)

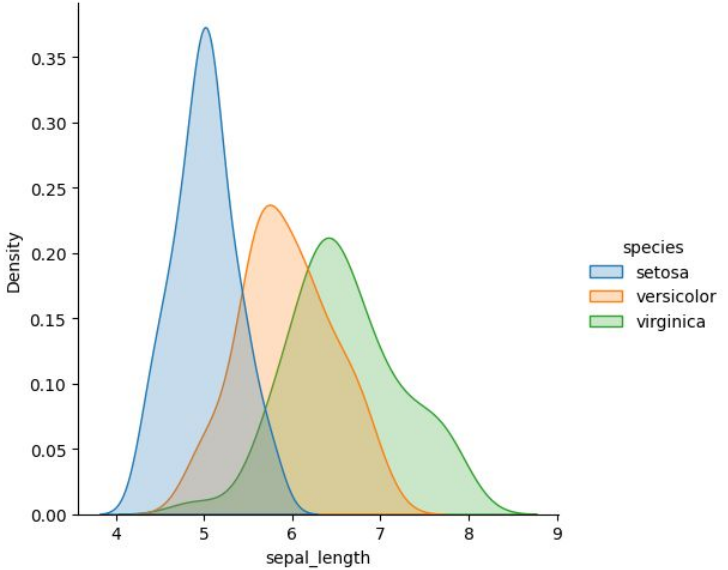
1.histplot

- Histograms visualize the distribution of a continuous variable.



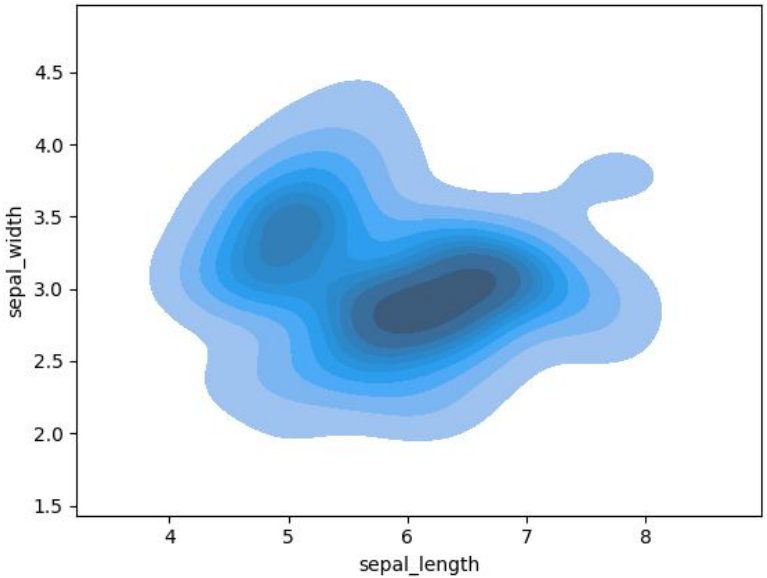
2.Displot

- Displays a continuous trend over time or order.



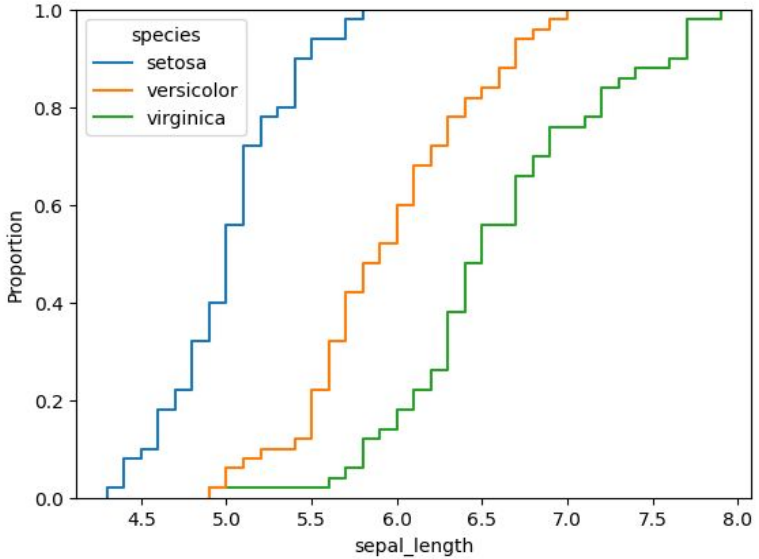
Distribution Plot (displot)

3. KDE Plot



4. ECDF Plot

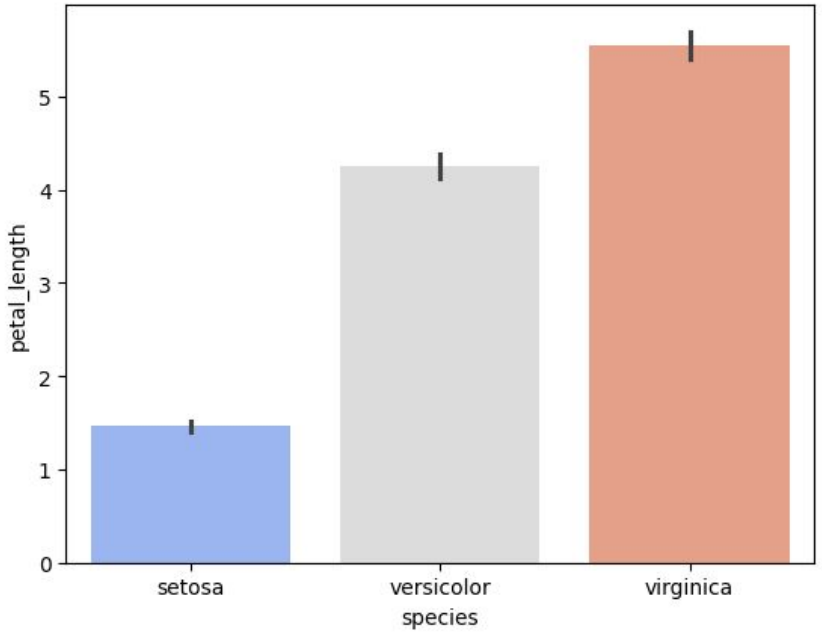
➢ Cumulative distribution



seaborn

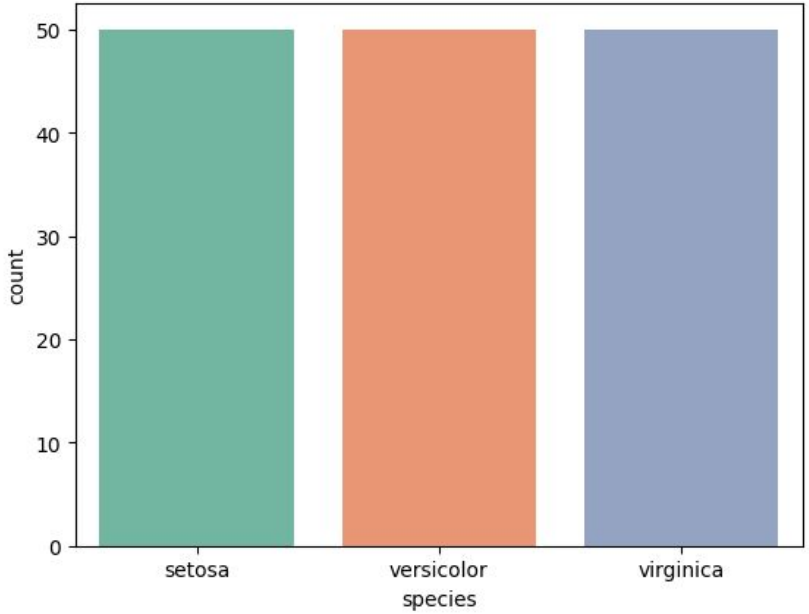
Categorical Plot (catplot)

1. Barplot



2. count plot

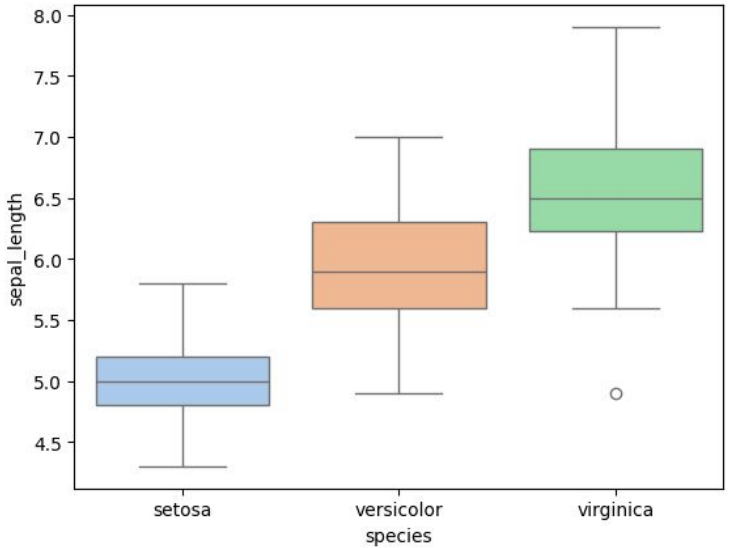
Count plots take in a categorical list and return bars that represent the number of list entries per category.



Categorical Plot (catplot)

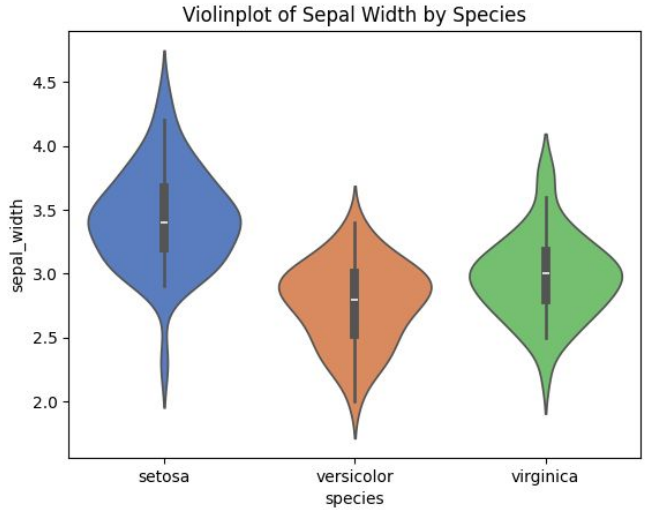
3. Box plot

commonly used to compare the distribution of one or more variables across different categories.



4. Violin plot

- combines aspects of both box plots and density plots.
- displays a density estimate of the data, smoothed by kernel density estimator, along with the interquartile range (IQR) and median in a box plot-like form.



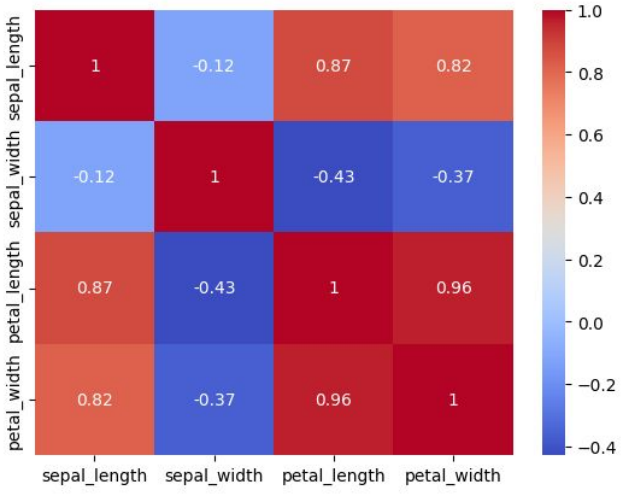
matrix plots

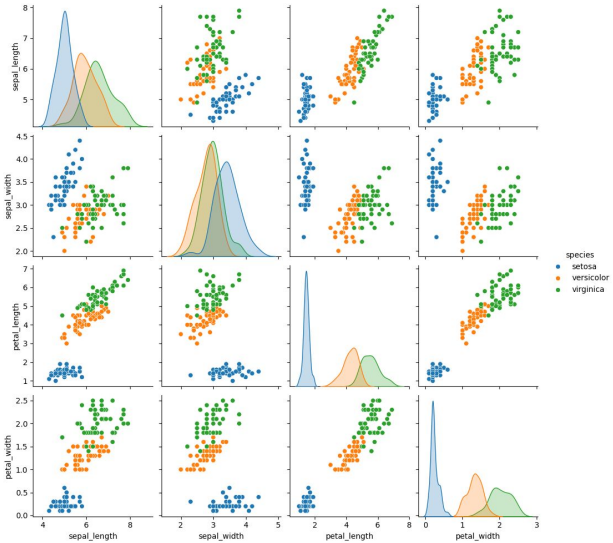
Correlation : *"How strongly are two variables connected?"*

- **Detect multicollinearity**
  Useful for identifying pairs of features with high correlation, which can negatively affect certain models (e.g., linear regression).

- **Feature relevance**
  Helps highlight variables that are strongly correlated with the target, assisting in feature selection.

- **Exploratory overview**
  Provides a compact visual summary of relationships among variables, guiding deeper analysis and model design.





COFFEE BREAK

Scikit Learn:

Sci*entific tool****kit*** *for Machine* ***Learn****ing*

What is Scikit-learn?

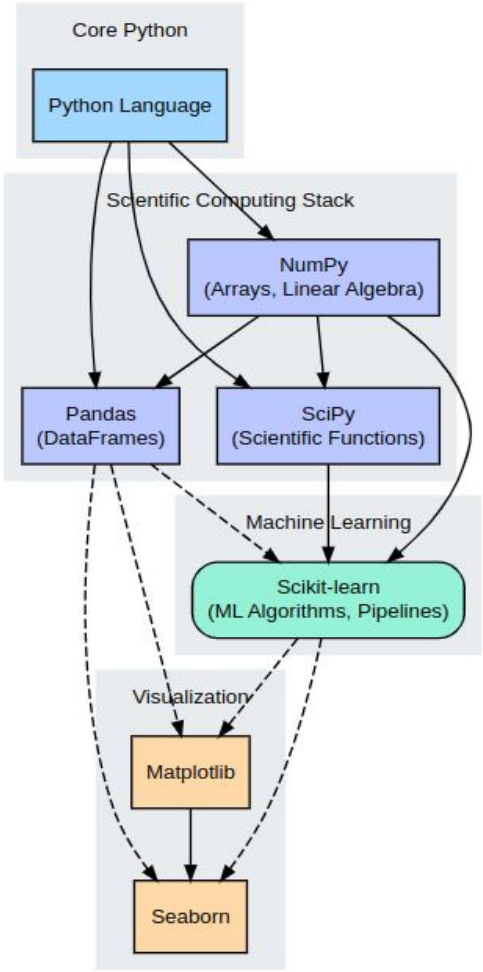
Overview of the Scikit-learn API

Loading Example Datasets and preprocessing

Hands-on Practical

Scikit-Learn

- Built on **NumPy**, **SciPy,** and **matplotlib**
- Open Source , Provides extensive documentation on its website https://scikit-learn.org



Scikit-Learn datasets

- Iris plants dataset
- Diabetes dataset

Installation

```
pip install  scikit-learn
```

```
lmport sklearn
```

Scikit-Learn

- **Simple and efficient tools for predictive data analysis**

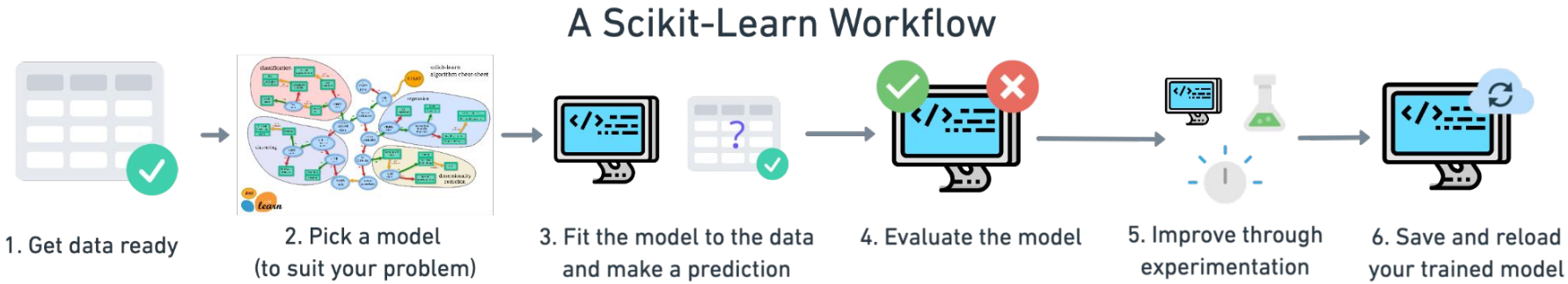
  data preprocessing

  supervised and unsupervised learning (clustering, regression & classification)

  model fitting

  model selection

  model evaluation



Scikit-Learn

- **Clean and Consistent API**

The Estimator: The Base Object: with a fit(X, y) method that learns from data and stores results in attributes.

The Transformer: Learns from data using fit(X, y) and modifies it with transform(X) or combined fit_transform(X, y).

The Predictor: Fits data with fit(X, y), predicts new results using predict(X), and evaluates performance via score(X, y)

Data Representation in Scikit-learn

Features Matrix (X)

The input data, often called features, samples, or instances, is typically represented as a two-dimensional array-like structure.

Shape: The expected shape of X is (n_samples, n_features).

Target Variable (y)

- **Shape:** The target variable is typically represented as a one-dimensional array (a vector). Its expected shape is (n_samples,).

Component	Convention	Typical Structure	Shape	Data Type (Common)
Features	X	2D Array/DataFrame	(n_samples, n_features)	float64
Target Variable	y	1D Array/Series	(n_samples,)	float64 (Regression) int or object (Classification)

Scikit-Learn

Essential step in any machine learning project to prepare data ready for modeling including:

Step 1: Data Cleaning

- Removing duplicates
- Correcting inconsistent formats
- Handling missing values

Step 2: Data splitting

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

Step 3: Data Transformation

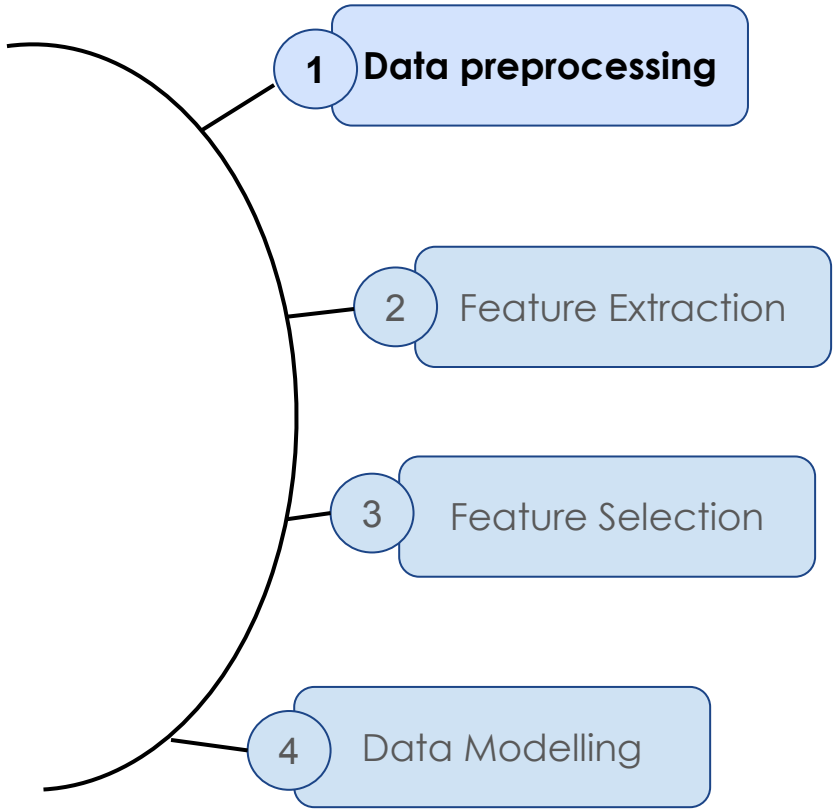
- Scaling and normalization
- Encoding categorical variables
- Feature engineering and extraction (Dimensionality reduction, kernel approximation , ect..)

Step 4 : Model Training & Evaluation

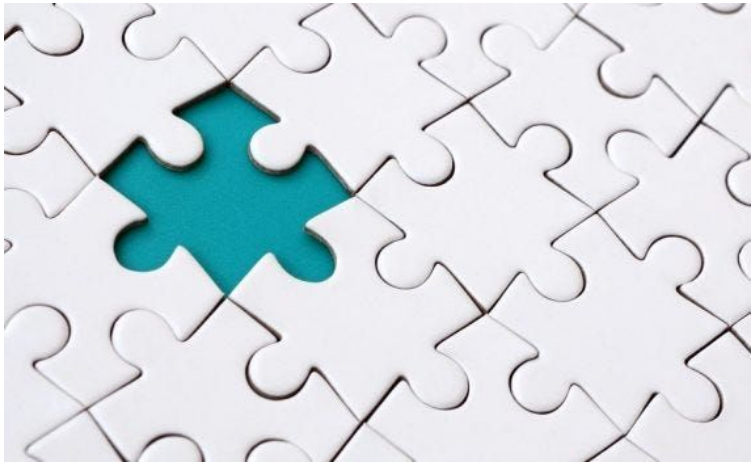
1. Data Preprocessing

Steps for

Machine Learning



Missing data



MCAR (Missing Completely At Random)

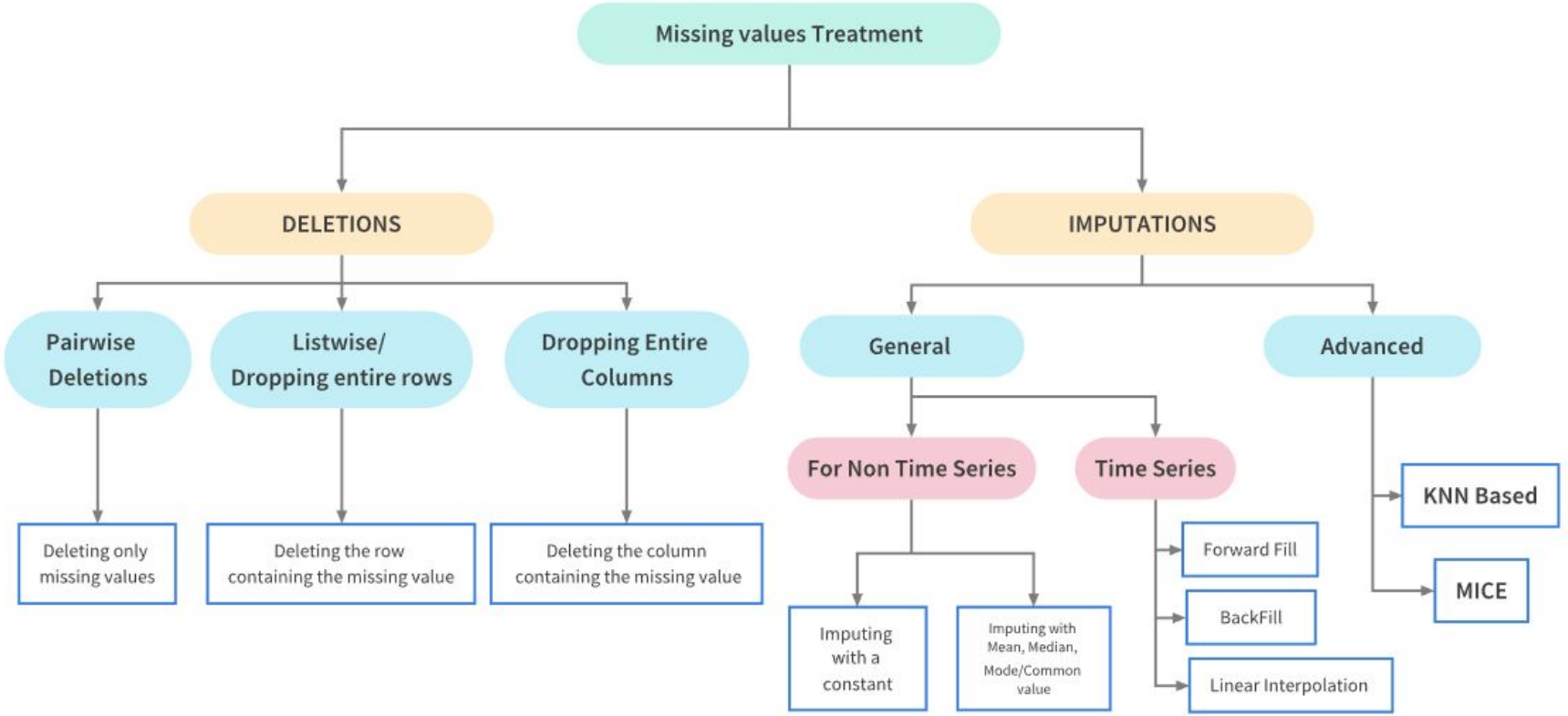
- The missingness has no relation to any variable.
- Safest to handle via deletion or simple imputation.

MAR (Missing At Random)

- The missingness is related to other observed data.
- Can be handled with imputation models using other variables.

MNAR (Missing Not At Random)

- The missingness depends on unobserved data (e.g., income missing more often for high earners).
- Most challenging. Requires domain expertise or advanced models.



Pandas dropna( ), Pandas dropna(axis=1)

Panda Fillna( ).mean fillna( ).median() fillna( ).mode( )

Panda Fillna (method="bfill") Fillna (method="ffill")

imputer = SimpleImputer(missing_values=np.nan, strategy= ['mean', 'median', 'most_frequent', 'constant'])

Panda interpolate (method='linear', 'polynomial' 'cubic', 'limit_direction='forward', 'backward', 'both, ' axis=0)

Missing data - Imputation

Iterative Imputer

- For datasets where imputing each feature independently would lose important correlations.
- Predicts missing values by modeling each feature as a function of others (uses regression).

```
imputer =sklearn.IterativeImputer(random_state=0)
```

KNN Imputer

- Fills missing values with the average of the k closest rows (neighbors) based on similarity.

```
imputer = KNNImputer(n_neighbors=2)
```

fit() or .fit_transform()

Data Transformation

Converting the data from one form to another to make it more suitable for analysis.

1. Scaling
    - Normalization
    - standardization

2. Encoding categorical Data

Normalization

The process of adjusting values measured on different scales to a common scale.

shifts and scales a distribution; it does not change its shape.

Common methods include:

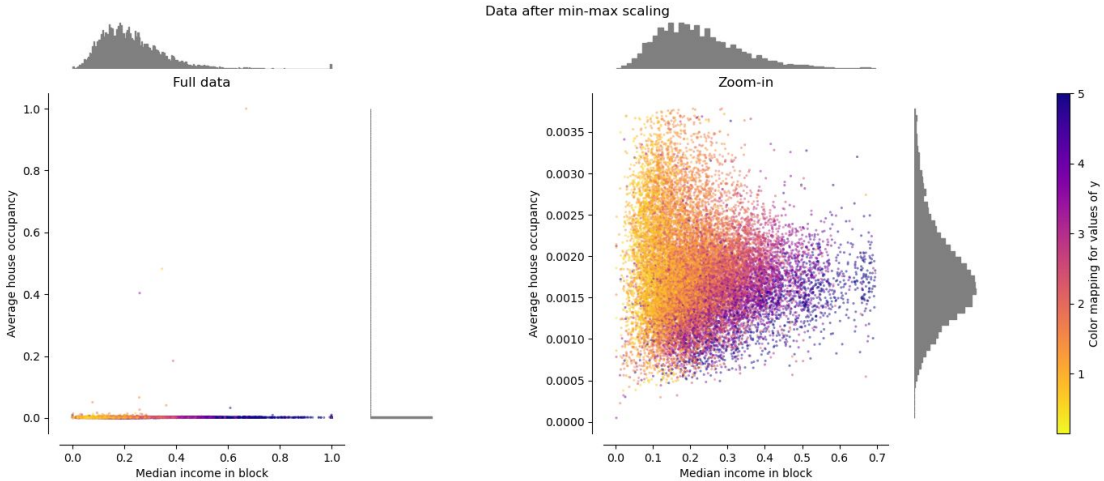
- Min-Max normalization
- Abs-Max normalization
- Long normalization
- Decimal scaling

Normalization (Min-Max Scaling):

- Scales the values to a specified range, typically **[0,1]**.
- preserves the original distribution.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['scaled_column']] = scaler.fit_transform(data[['numeric_column']])
```

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

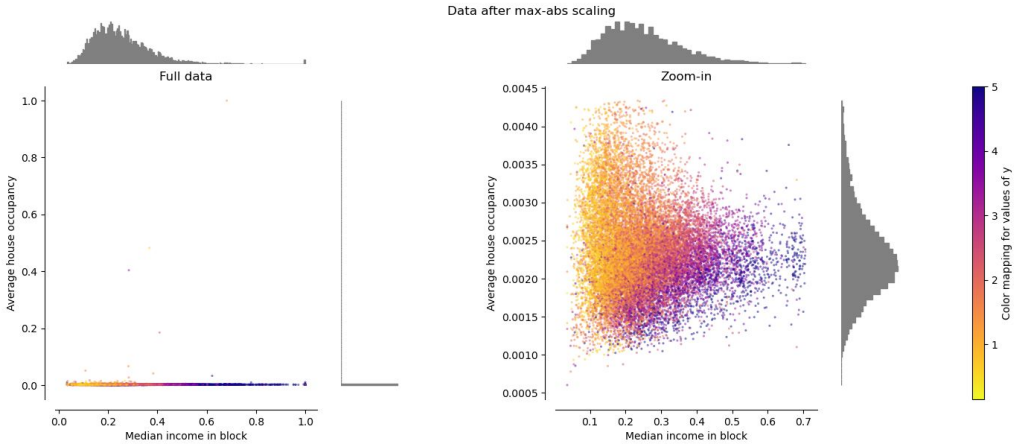


Normalization (MaxAbsScaler):

- scales in a way that the data lies within the **range [-1, 1]** .
- Suitable for data that is already centered at zero or sparse data.
- doesn't reduce the effect of outliers; linearly scales them down.

```
from sklearn.preprocessing import MinAbsScaler
scaler = MinAbsScaler()
data[['scaled_column']] = scaler.fit_transform(data[['numeric_column']])
```

$$xscaled = \frac{x}{max(x)}$$

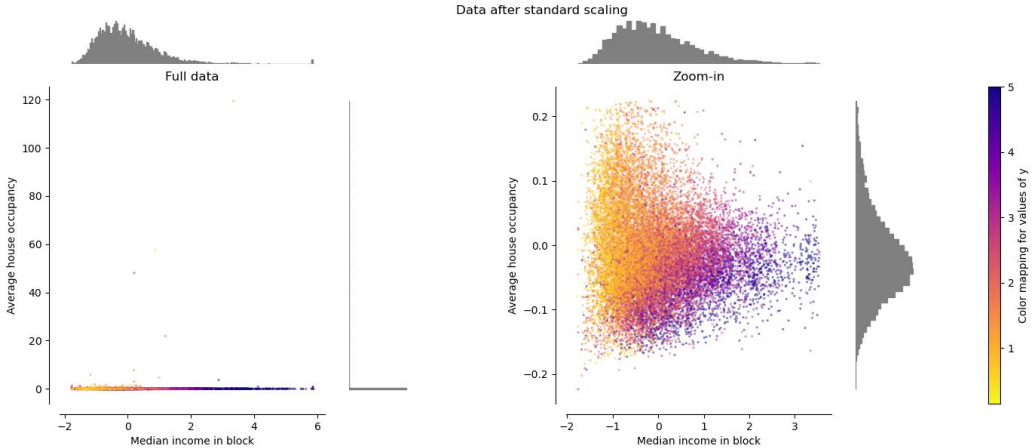


Standardization (StandardScaler):

- Removes the mean and scales the data to unit variance.
- Help to reduce outliers impact but have an influence when computing the empirical mean and standard deviation.
- cannot guarantee balanced feature scales in the presence of extreme outliers.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
 data[['standardized_column']]=scaler.fit_transform(data[['numeric_column']])
```

$$X_i' = \frac{X_i - \mu}{\sigma} = \frac{X_i - X_{\text{mean}}}{X_{std}}$$

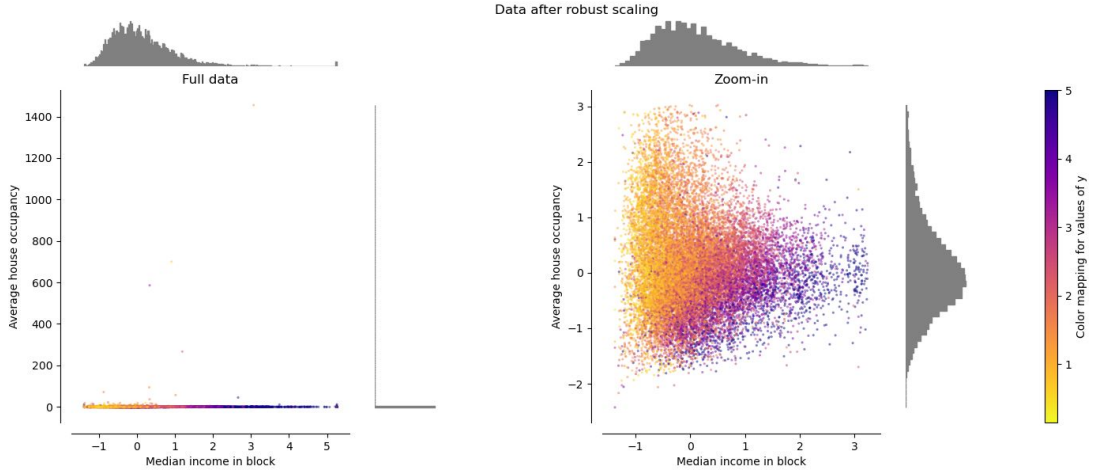


RobustScaler:

- The centering and scaling statistics are based on percentiles
- Not influenced by a small number of very large marginal outliers

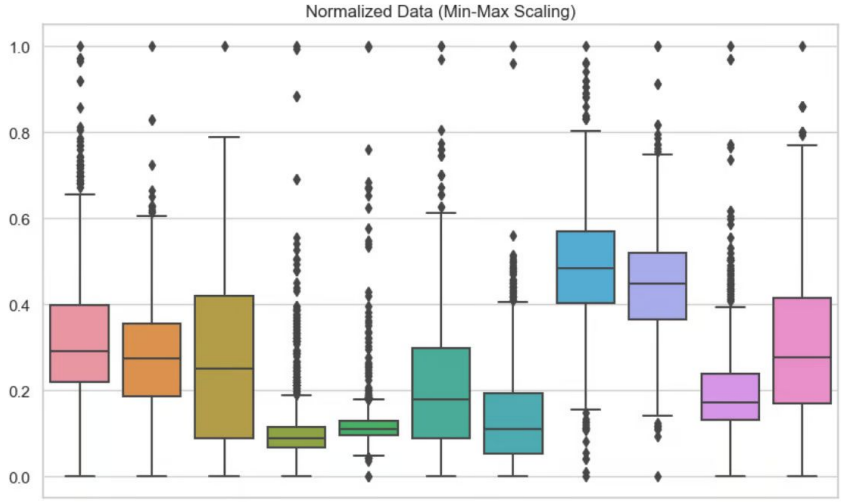
```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
 data[['standardized_column']]=scaler.fit_transform(data[['numeric_column']])
```

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

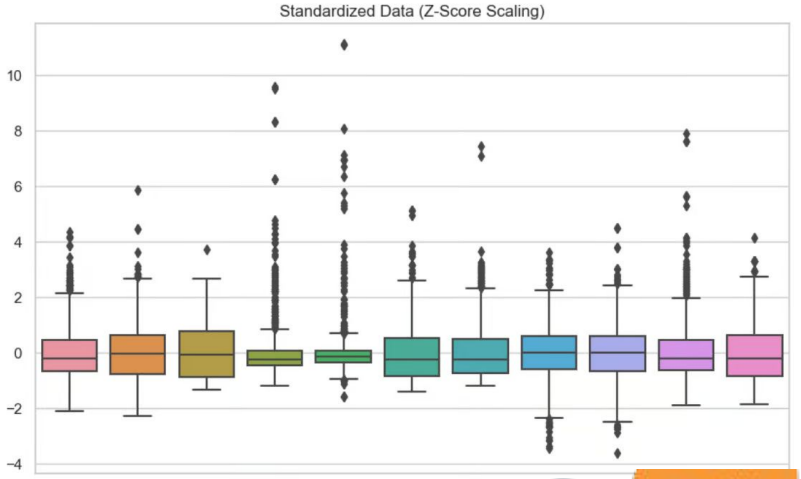


Normalization vs. Standardization

- Scale data to a range based on min & max values
- Can help adjust for outliers depending on the technique
- Applied in KNN and neural network that require data to be on a consistent scale



- Centers around 0 mean and scales by standard deviation 1.
- More consistent to fix outlier
- Best for algorithms that requires features to have summon scale such as PCA, SVM, and logistic regression



scikit learn

Encoding Categorical data

Categorical data need to be encoded into numerical values before they can be used in machine learning algorithms. We can use :

1. **Label Encoder**

Encode target labels with value between 0 and n_classes-1.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['encoded_column'] = le.fit_transform(data['categorical_column'])
```

Original Data:

Color	Size	Price
Blue	L	100
Green	M	150
Red	S	200
Green	XL	120
Red	M	180

Label Encoding

Label Encoded Data:

Color	Size	Price
0	0	100
1	1	150
2	2	200
1	3	120
2	1	180

!! Label Encoder can introduce an unintended ordinal relationship between categories if they don't have a natural order

Encoding Categorical data

2. One-Hot Encoder

Encode categorical features as a one-hot numeric array.

```
from sklearn.preprocessing import OnehotEncoder
 encoder= OnehotEncoder()
 data['encoded_column'] = encoder.fit_transform(data['categorical_column'])
```

id	color
1	red
2	blue
3	green
4	blue

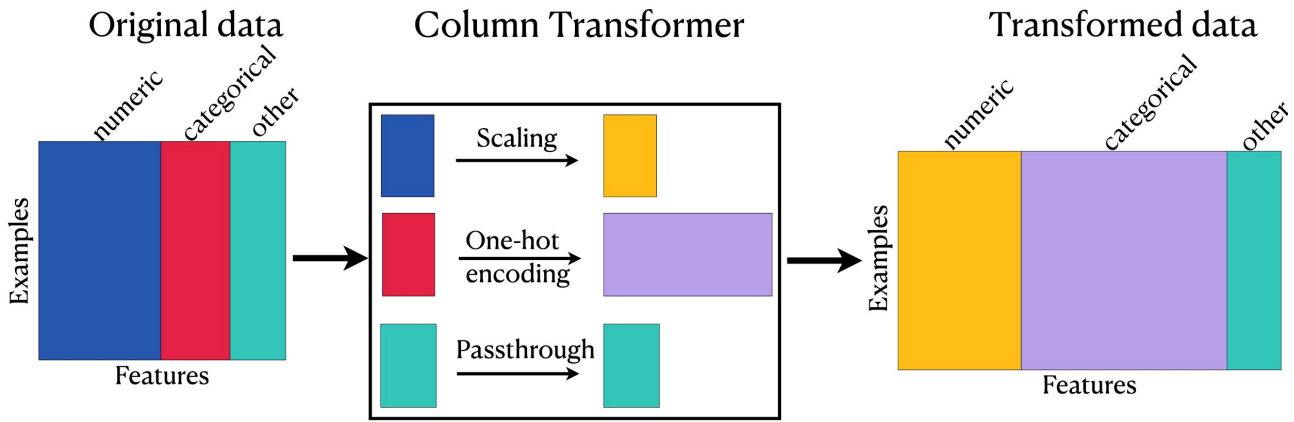
id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Encoding Categorical data

3. Column Transformer

Applies transformers only to a group of columns .

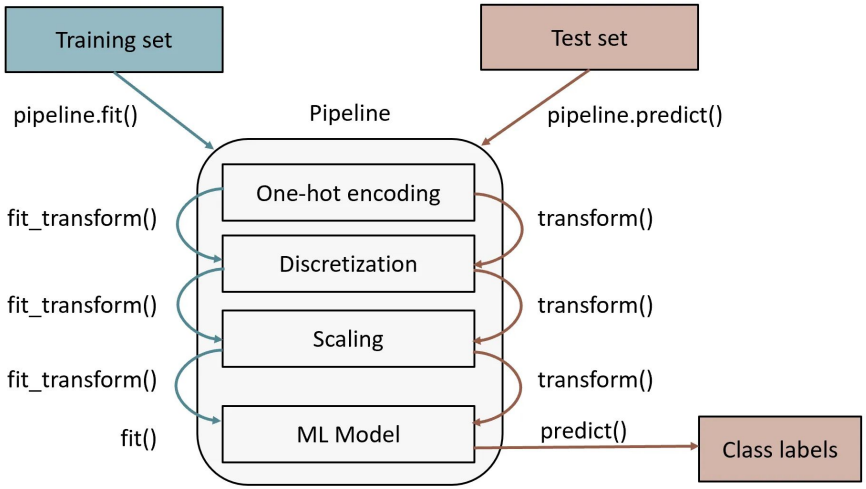
```
from sklearn.compose import ColumnTransformer
 preprocessor = ColumnTransformer( [ ("categorical", categorical_preprocessor, ["catg", "catag"]), ("numerical", numeric_preprocessor, ["num", "num"]), ] )
```



Scikit-learn Pipeline

- Combine all the preprocessing in one step

```
from sklearn.pipeline import pipeline , make_pipeline

pipe= Pipeline([('scaler', StandardScaler()), ('clf',
LogisticRegression())])

pipe = make_pipeline(StandardScaler(),
OneHotEncoder(), LogisticRegression())

pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```



Best Practices for Data Preprocessing

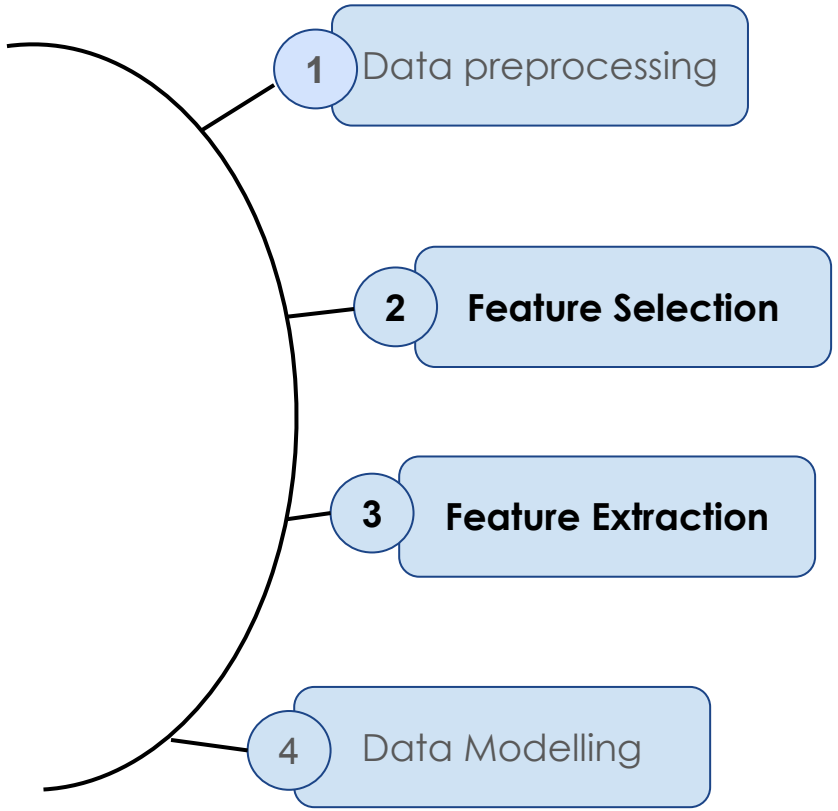
- **Understand the data**
  - **Key features**: Identify the most relevant variables for your task.
  - **Potential anomalies**: Look for outliers or inconsistencies that may distort the analysis.
  - **Relationships**: Explore correlations and dependencies between variables to inform transformations or feature engineering.
- **Automate repetitive steps**
  - Use tools like sklearn Pipeline() to build repeatable, structured workflows.
- **Document every preprocessing step**
  - Makes it easier to revisit your process later — for yourself or other team members.
- **Iterate and improve over time**
  - Let model feedback guide adjustments to your pipeline.
  - Example: Feature engineering might reveal valuable new inputs, or fine-tuning how outliers are handled can boost performance.

COFFEE BREAK

2. Feature Engineering

Steps for

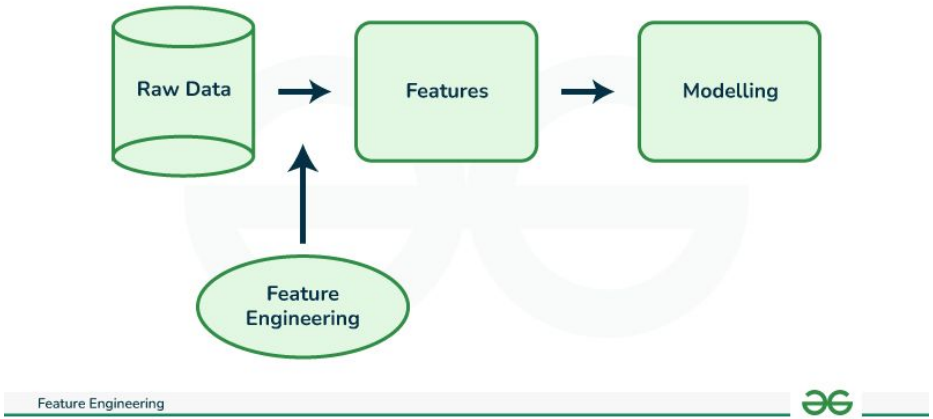
Machine Learning



Feature Engineering

The process of **transforming raw data into features** **that are suitable for machine learning models**

that better represent the underlying patterns to machine learning models, leading to **improved performance**.

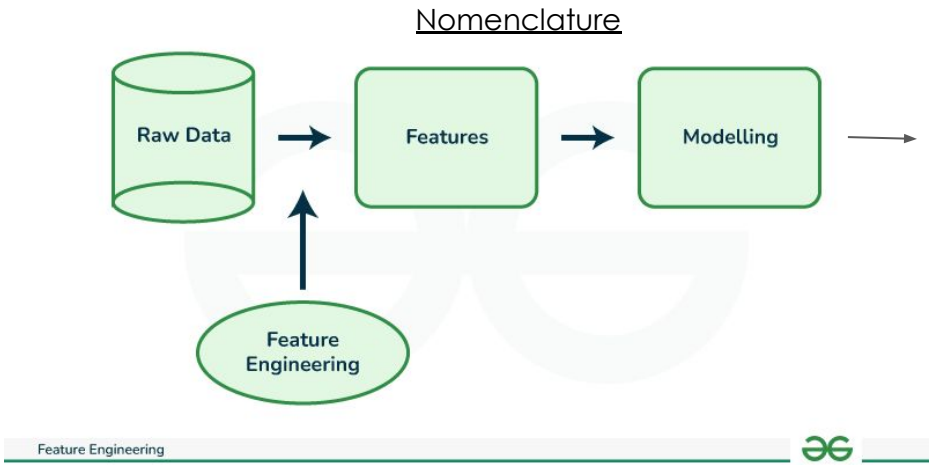


https://www.geeksforgeeks.org/what-is-feature-engineering/

Feature Engineering

The process of **transforming raw data into features** **that are suitable for machine learning models**

that better represent the underlying patterns to machine learning models, leading to **improved performance**.



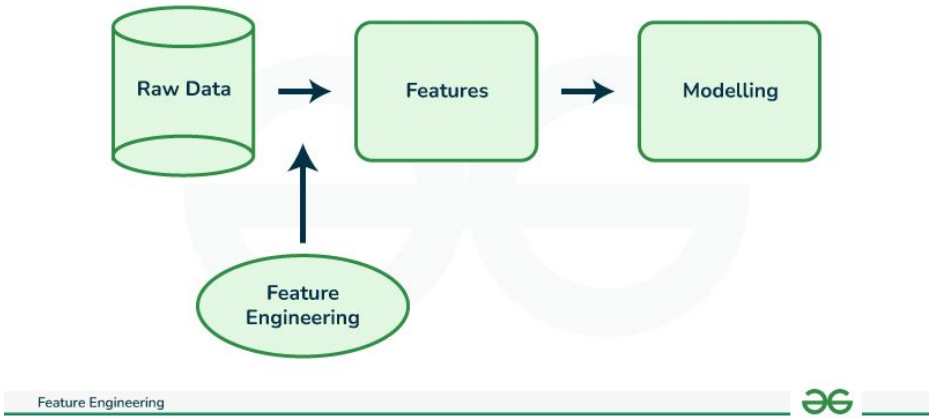
https://www.geeksforgeeks.org/what-is-feature-engineering/

Feature Engineering

Feature engineering is the process of **transforming** raw **data** into features that better represent the underlying patterns to machine learning models, leading to **improved performance**.

It includes:

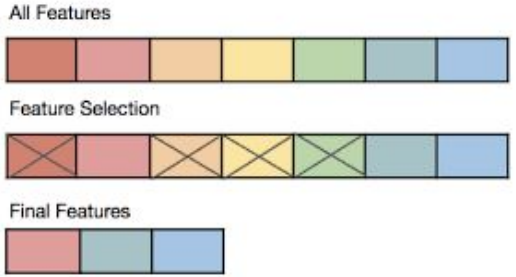
- Creating new features
- Transforming or encoding features
- Selecting the most useful features
- Extracting new representations of the data



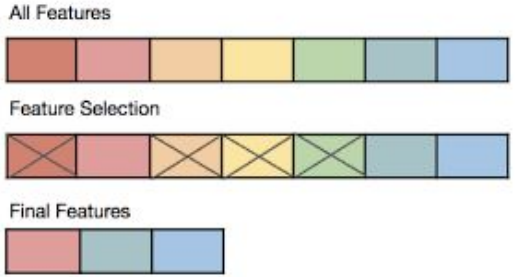
https://www.geeksforgeeks.org/what-is-feature-engineering/

Feature Selection

Reduce the number of features by keeping only the most informative ones.



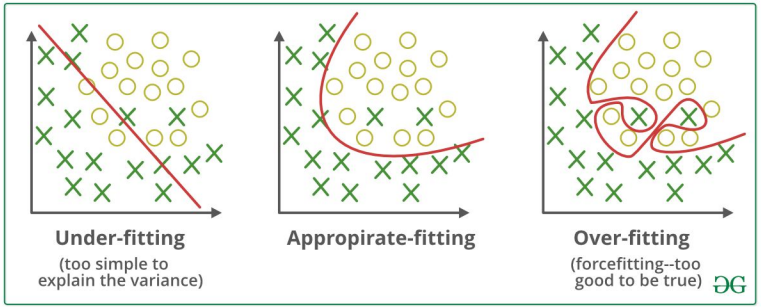
Feature Selection



Reduce the number of features by keeping only the most informative ones.

This helps:

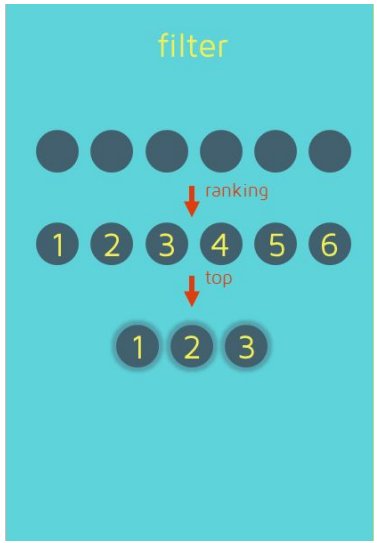
- **Improve** model **accuracy** (removing noisy or irrelevant features)
- **Reduce overfitting** (better generalization to new / unseen data)
- **Speed up training and inference** (specially critical for real-time systems or large-scale models)
- **Increase** model **interpretability** (simpler models tend to be easier to interpret)



Feature Selection

Feature Selection - Filter methods

Use **statistical tests** to rank or score each feature, independent of the model (**model-independent**).



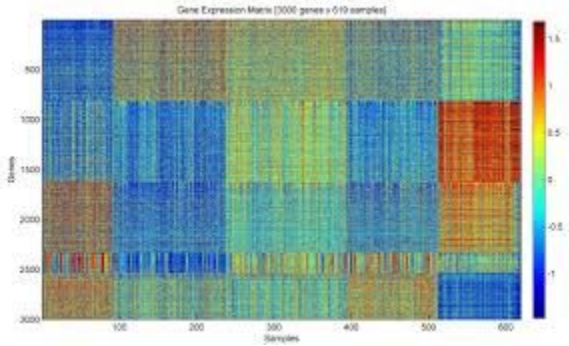
Feature Selection - Filter methods

Use **statistical tests** to rank or score each feature, independent of the model (**model-independent**).

Examples:

- **Variance Threshold** – remove features with little variability.
- **Chi-squared test** – for categorical targets.
- **Mutual Information** – measures mutual dependency between features and target.

E.g. Gene expression Matrix



Feature Selection - Filter methods

Use **statistical tests** to rank or score each feature, independent of the model (**model-independent**).

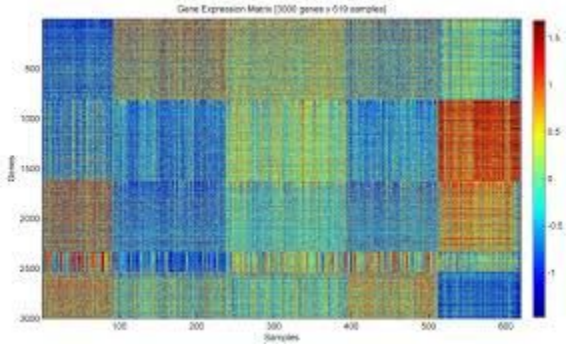
Examples:

- **Variance Threshold** – remove features with little variability.
- **Chi-squared test** – for categorical targets.
- **Mutual Information** – measures mutual dependency between features and target.

Use when:

- You want a quick, **model-agnostic** selection
- For high-dimensional data (e.g., genomics, text)

E.g. Gene expression Matrix



Feature Selection - Filter methods

Chi-squared test - statistical test to **assess** if **two categorical variables are independent**.

Output: χ^2 statistic (higer, more dependence) and p-value

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

- O = observed frequency
- E = expected frequency

Feature Selection - Filter methods

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

- O = observed frequency
- E = expected frequency

Chi-squared test - statistical test to **assess** if **two categorical variables are independent**.

Output: χ^2 statistic (higer, more dependence) and p-value

Feature selection: Assess the **independence** between **each feature** and the **target variable**

Features are sorted by their χ^2 or p-value and then **top-k or top-percentile** of features are selected.

E.g. Classification based on genomic mutation data (presence/absence of variants)

Feature Selection - Filter methods

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

- O = observed frequency
- E = expected frequency

Chi-squared test - statistical test to **assess** if **two categorical variables are independent**.

Output: χ^2 statistic (higer, more dependence) and p-value

Feature selection: Assess the **independence** between **each feature** and the **target variable**

Features are sorted by their χ^2 or p-value and then **top-k or top-percentile** of features are selected.

Use when:

- Your **features** **are** **categorical** or can be discretized (e.g. using binning)
- The **target** variable is **categorical** (**classification**)
- Simple, fast way to rank feature relevance

Limitations:

- Requires **non-negative**, discrete inputs (not ideal for continuous values)
- Not suitable for regression (target must be categorical)
- **Assumes independence between features** (evaluates one at a time) and doesn't capture interactions or complex relationships.

E.g. Classification based on genomic mutation data (presence/absence of variants)

Feature Selection - Filter methods

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

- O = observed frequency
- E = expected frequency

Chi-squared test - statistical test to **assess** if **two categorical variables are independent**.

Output: χ^2 statistic (higer, more dependence) and p-value

Feature selection: Assess the **independence** between **each feature** and the **target variable**

Features are sorted by their χ^2 or p-value and then **top-k or top-percentile** of features are selected.

Use when:

- Your **features** are **categorical** or can be discretized (e.g. using binning)
- The **target** variable is **categorical** (**classification**)
- Simple, fast way to rank feature relevance

Limitations:

- Requires **non-negative**, discrete inputs (not ideal for continuous values)
- Not suitable for regression (target must be categorical)
- **Assumes independence between features** (evaluates one at a time) and doesn't capture interactions or complex relationships.

E.g. Classification based on genomic mutation data (presence/absence of variants)

Feature Selection - Filter methods

Mutual Information - measures the **dependency** between two variables. It quantifies how much knowing one variable reduces uncertainty about the other. (0, Inf)

Feature Selection - Filter methods

Mutual Information - measures the **dependency** between two variables. It quantifies how much knowing one variable reduces uncertainty about the other. (0, Inf)

Feature selection: How much **information** does this **feature** give me about the **target**?

Compute MI between each feature and the target. Then, rank features based on the score. (Higher MI, greater dependency)

Feature Selection - Filter methods

Mutual Information - measures the **dependency** between two variables. It quantifies how much knowing one variable reduces uncertainty about the other. (0, Inf)

Feature selection: How much **information** does this **feature** give me about the **target**?

Compute MI between each feature and the target. Then, rank features based on the score. (Higher MI, greater dependency)

Use when:

- **Features can be continuous or discrete** (encode categorical as integers)
- Compatible with **classification** and **regression**
- Captures **nonlinear relationships** and it does not assume independence between features (non-parametric).

Limitations:

- Doesn't consider feature interactions (evaluates one at a time)
- Could yield redundant features

E.g. Regression/Classification based on continuous/discrete/categorical values with non-linear relationships to target

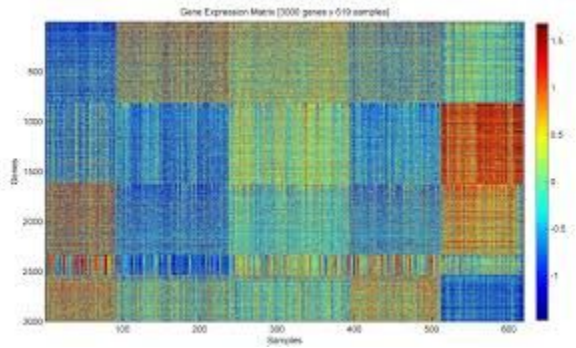
Feature Selection - Filter methods

Use **statistical tests** to rank or score each feature, independent of the model (**model-independent**).

Other:

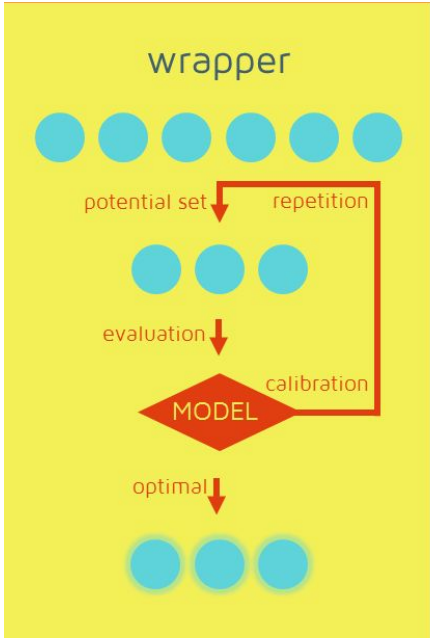
- **Biological priors**: sDEGs, mutated genes, …
- **Low count** filtering (removing genes with low read counts in transcriptomics)
- Remove **highly correlated features** (using a correlation threshold or mrmr)

E.g. Gene expression Matrix



Feature Selection - Wrapper methods

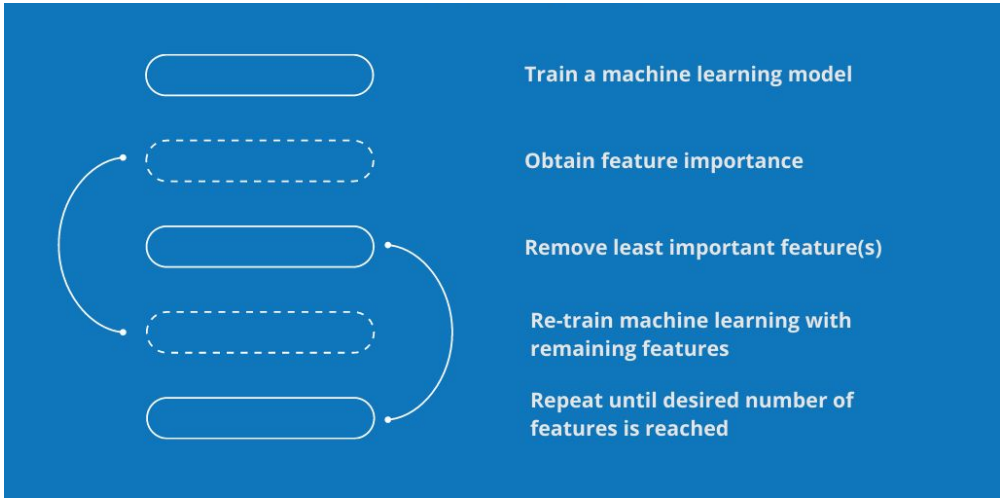
Wrapper methods train a model on different feature subsets and select the one that performs best (**model-based**).



Feature Selection – Wrapper methods

Wrapper methods train a model on different feature subsets and select the one that performs best (**model-based**).

Recursive Feature Elimination (RFE) – removes the least important feature(s) recursively



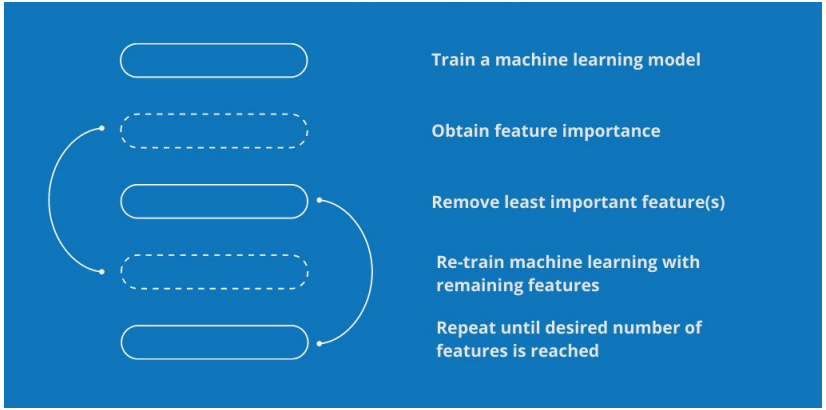
Feature Selection – Wrapper methods

Wrapper methods train a model on different feature subsets and select the one that performs best (**model-based**).

Recursive Feature Elimination (RFE) – removes the least important feature(s) recursively

Use when:

- You want **high accuracy**
- Okay with high computational cost
- Consider feature interactions
- Model-aware selection

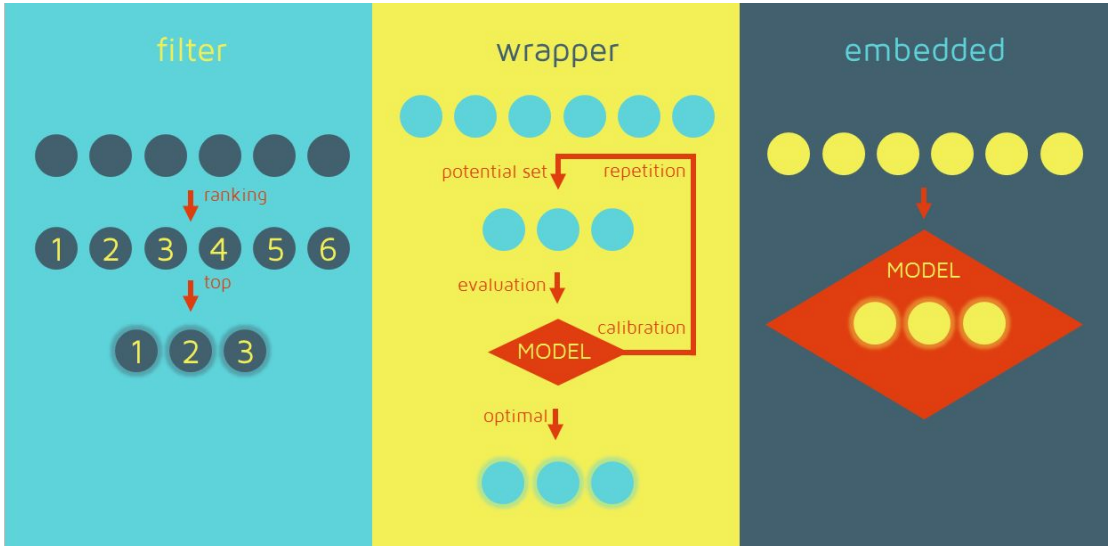


Limitations:

- **Computationally expensive**
- Slow or impractical with very large data sets (does not scale well)
- Risk of overfitting (specially on small datasets if not cross-validated)

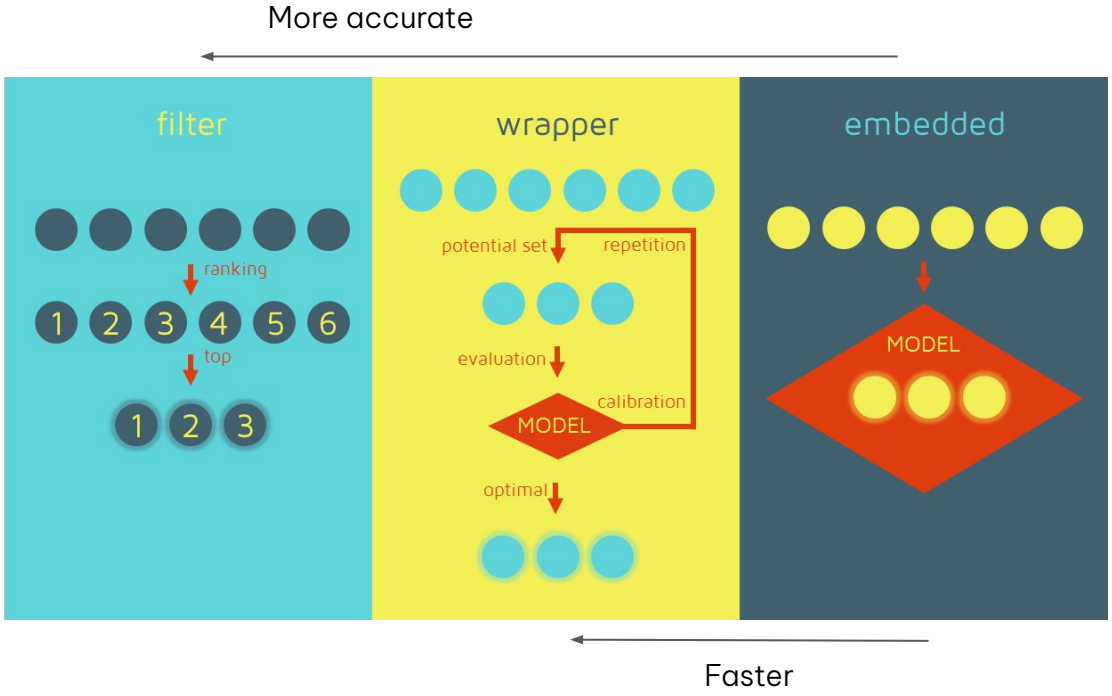
Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).



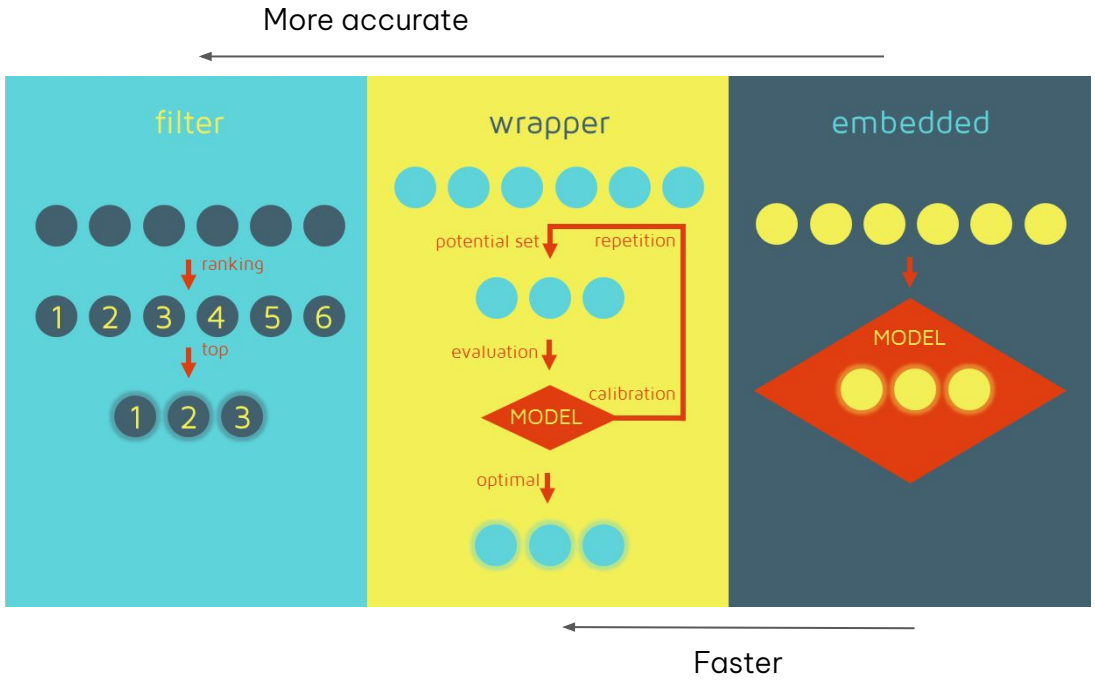
Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).



Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).



Internal mechanisms to determine:

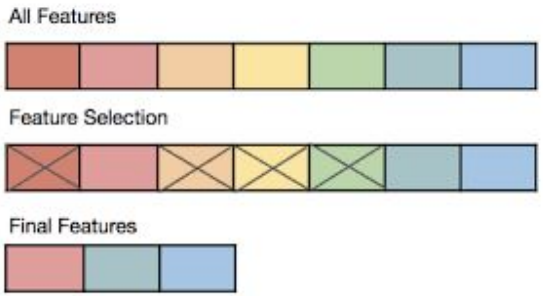
- Which features **to keep**
- Which features **to drop** by assigning low or zero importance

Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).

Examples:

Lasso Regression
(L1 Regularization)
L1 penalty: forces some coefficients to become zero.



Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).

Examples:

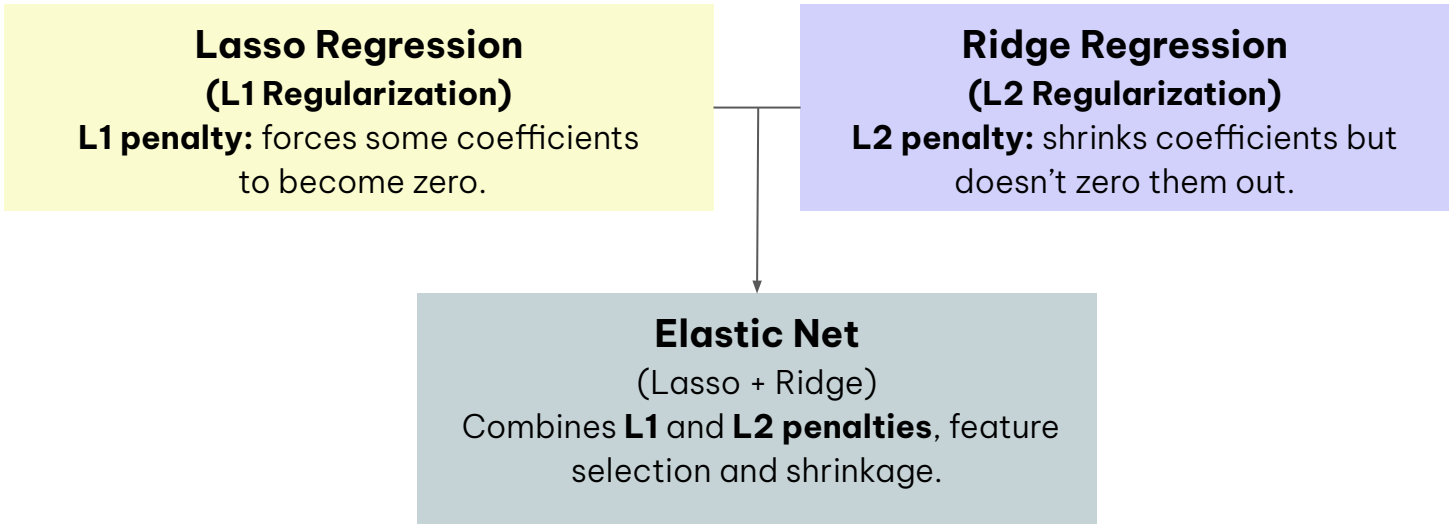
Lasso Regression
(L1 Regularization)
L1 penalty: forces some coefficients to become zero.

Ridge Regression
(L2 Regularization)
L2 penalty: shrinks coefficients but doesn't zero them out.

Feature Selection – Embedded methods

Embedded methods perform feature selection **during training** (**Built-in to Model**).

Examples:



Feature Selection – Embedded methods

	Lasso Regression	**Ridge Regression**	**Elastic Net**
Type of penalty	L1 (absolute value)	L2 (squared magnitude)	L1 + L2
Feature selection	Yes	No	Yes (via L1)
When to use	Many features and want to remove irrelevant ones	Many small effects, all features matter	Best for many correlated features, balance of shinkage & selection
Hyperparameters	Alpha (controls the strength of regularization)	Alpha	Alpha and L1_ratio (adjusts the balance between Lasso and Ridge)

*A higher alpha means more shrinkage.

Feature Selection – Embedded methods

Lasso Regression

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
```

Ridge Regression

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
```

Elastic Net

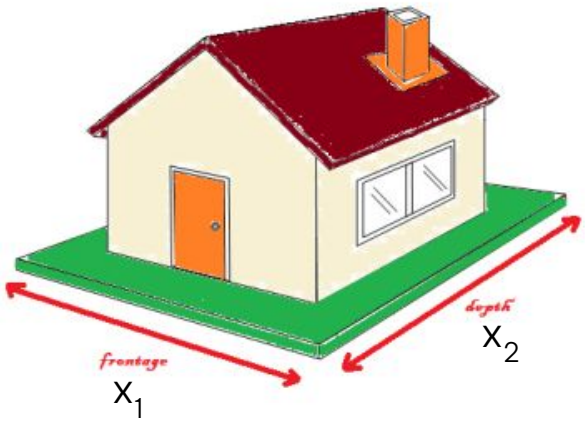
```
from sklearn.linear_model import ElasticNet

enet = ElasticNet(alpha=0.1, l1_ratio=0.5)
enet.fit(X_train, y_train)
```

50% L1, 50% L2

Feature Extraction

Transform the **feature space** by combining or projecting features into a new set of variables.

Instead of removing features, **you create new ones** that summarize or better represent the data.

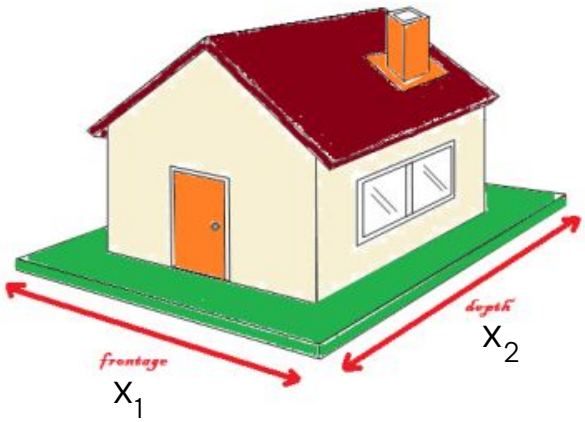


Feature Extraction

Transform the **feature space** by combining or projecting features into a new set of variables.

Instead of removing features, **you create new ones** that summarize or better represent the data.

Price $y = ax_1 + bx_2 + c$



Feature Extraction

Transform the **feature space** by combining or projecting features into a new set of variables.

Instead of removing features, **you create new ones** that summarize or better represent the data.

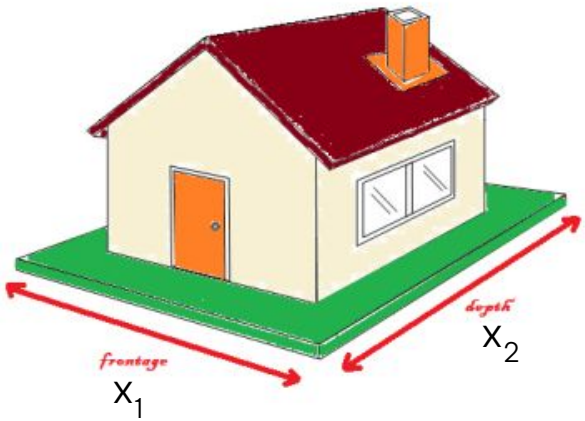
Price $y = ax_1 + bx_2 + c$

Area = frontage x depth

$x_3 = x_1 x_2$

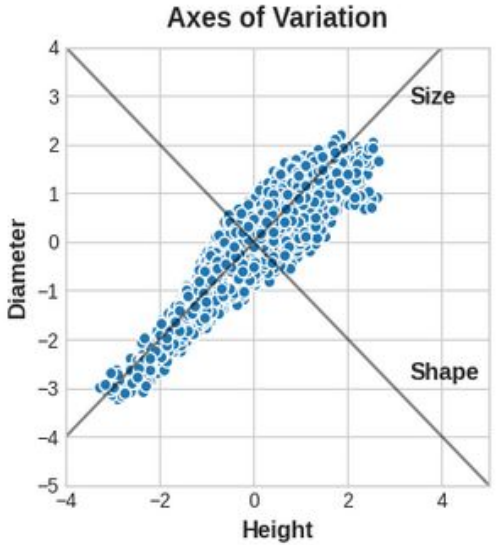
new feature

Price $y = ax_1 + bx_2 + cx_3 + c$



Feature Extraction – Principal Component Analysis (PCA)

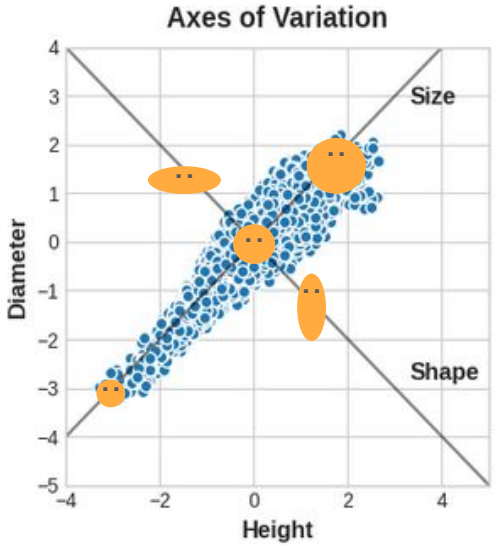
PCA is a **dimensionality reduction** technique used to transform high-dimensional data into fewer dimensions (called **principal components**) while preserving as much variability as possible.



Abalone

Feature Extraction – Principal Component Analysis (PCA)

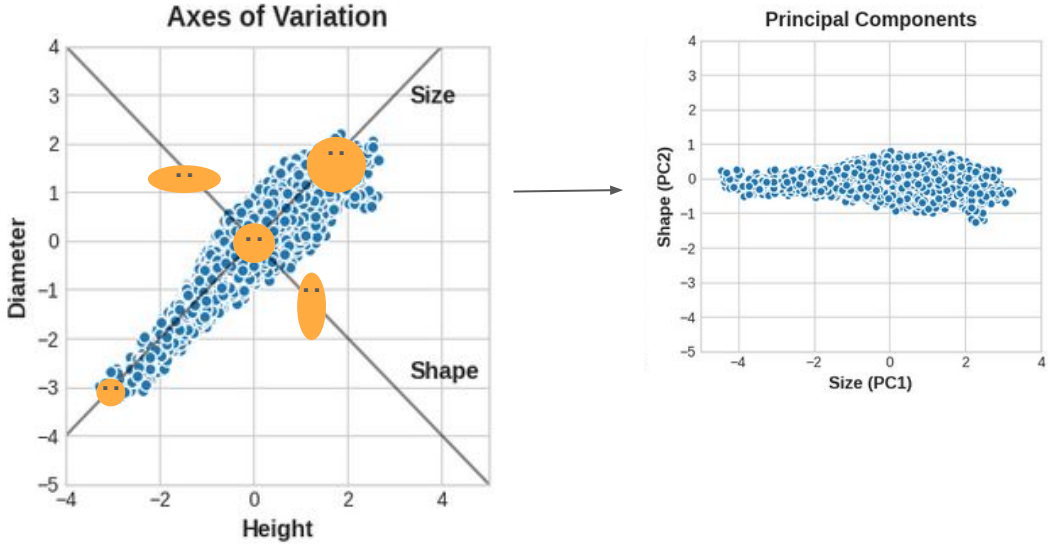
PCA is a **dimensionality reduction** technique used to transform high-dimensional data into fewer dimensions (called **principal components**) while preserving as much variability as possible.



Abalone

Feature Extraction – Principal Component Analysis (PCA)

PCA is a **dimensionality reduction** technique used to transform high-dimensional data into fewer dimensions (called **principal components**) while preserving as much variability as possible.

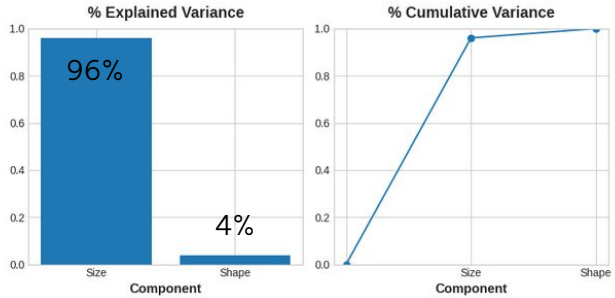


Principal components (linear combinations of the original features)

Loadings (weights)

PC1 (Size) = 0.707 * Height + 0.707 * Diameter

PC2 (Shape) = 0.707 * Height - 0.707 * Diameter



Feature Extraction – Principal Component Analysis (PCA)

PCA projects data into a **lower-dimensional space** that **maximizes variance**.

- **Unsupervised** method
- New components are **linear combinations** of original features
- First components capture most of the variance

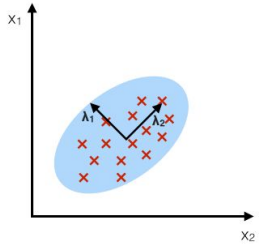
Use when:

- Features are **correlated**
- PCA for preprocessing, UMAP/t-SNE for visualization
- You don't care about preserving interpretability of original features

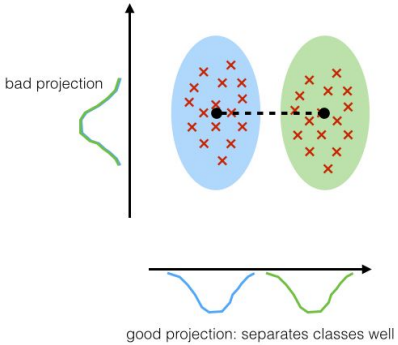
Feature Extraction – Linear Discriminant Analysis (LDA)

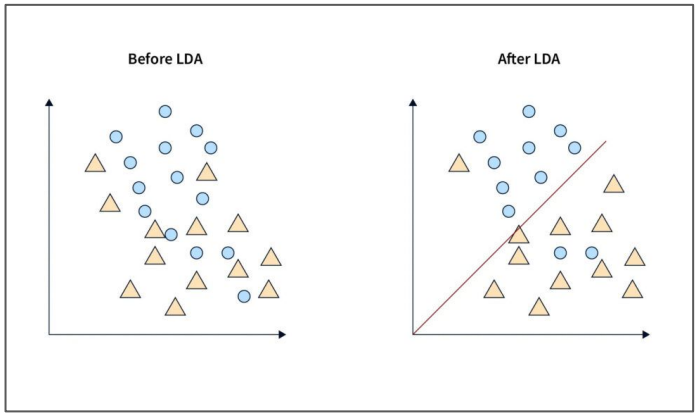
LDA is a **supervised dimensionality reduction** technique that **maximizes class separability.**
Finds a linear combination of features that separate two or more classes.

PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation





Use when:

- Your data has labels
- You want features that help classification of linearly separable classes

Feature Engineering

Final notes

- In deep learning (especially vision, NLP), raw data is often used directly.
- Neural networks learn features automatically, making manual engineering less critical.
- However, structured/tabular data still benefits heavily from thoughtful feature engineering.
- Domain knowledge & data exploration and visualization.

Thanks!