**AI / ML NOTES**

**Machine Learning:**

Machine Learning is the process of training a model (algorithm) on data so it can make predictions or decisions without being manually coded for every scenario. Without being explicitly programmed.

**Narrow AI** is only good at do the one task at a time.

**Machine Learning** is a subset of AI.

**Deep learning** is subset of Machine Learning.

**Data Science** means the analysing the data.

Relational DB

SQL

Spreadsheets

Big Data

NoSQL(MongoDB)

Big Data

NoSQL(MongoDB)

A diagram of a data flow

AI-generated content may be incorrect.

**Types of Machine Learning:**

**1. Supervised Learning:**

A type of machine learning where the model is trained on **labelled data** (input-output pairs).

**Goal:**  
Learn a mapping from inputs to known outputs.

**Examples:**

* Email spam detection (Spam or Not Spam)
* House price prediction (based on size, location, etc.)

**Algorithms:**

* Linear Regression
* Classification
* Decision Trees
* Support Vector Machines (SVM)
* Neural Networks

**2. Unsupervised Learning:**

A type of machine learning where the model is trained on **unlabelled data** and finds hidden patterns or groupings.

**Goal:**  
Explore structure in data without explicit labels.

**Examples:**

* Customer segmentation
* Anomaly detection
* Market basket analysis

**Algorithms:**

* K-Means Clustering
* Hierarchical Clustering
* Principal Component Analysis (PCA)
* Autoencoders

**3. Reinforcement Learning:**

A learning method where an **agent learns to make decisions** by interacting with an environment and receiving **rewards or penalties.**

**Goal:**  
Maximize cumulative reward over time.

**Examples:**

* Game playing (e.g., Chess, Go, Atari games)
* Robotics (e.g., walking, grasping)
* Self-driving cars

**Key Concepts:**

* Agent, Environment
* Actions, States
* Rewards, Policy
* Q-Learning, Deep Q-Networks (DQN)

**4. Transfer Learning:**

T**ransfer learning** is a machine learning technique where a **pre-trained model** (trained on a large dataset) is **reused or fine-tuned** for a **new but related task**.

**MACHINE LEARNIING:**

DATA

MACHINE LEARNING ALGORITHM

PATTERNS

**DATA ANALYSIS :**

The process of **inspecting, cleaning, transforming**, and modelling data to **extract useful insights**.

**Goal:**  
Understand the **past and present** using data.

**Tasks Involved:**

* Data cleaning
* Exploratory Data Analysis (EDA)
* Data visualization
* Descriptive statistics
* Reporting findings

**Tools/Technologies:**

* Excel
* SQL
* Python (Pandas, Matplotlib, Seaborn)
* R
* Power BI / Tableau

**DATA SCIENCE:**

A broader field that includes **data analysis**, but also involves **machine learning, predictive modelling, and data engineering**.

**Goal:**  
Use data to **predict future outcomes** and build intelligent systems.

**Tasks Involved:**

* All tasks of data analysis
* Building machine learning models
* Feature engineering
* Model deployment
* Big data handling

**Tools/Technologies:**

* Python (Scikit-learn, TensorFlow, PyTorch)
* R
* SQL + NoSQL
* Hadoop / Spark
* Jupyter Notebooks
* Git & Docker

**MACHINE LEARNIING / DATA SCIENCE FRAMEWORK:**

**6 FRAMEWORK:**

1. **PROBLEM DEFINITION :** What problem are we trying to solve?

**Supervised, Unsupervised, Classification, Regression**

1. **DATA :** What kind of Data We have?

**Structured, Unstructured**

1. **EAALUATION:** What defines success for us ?

**HOUSE DATA** 🡪 **MACHIINE LEARNING MODEL** 🡪 **HOUSE PRICE** 🡪 **PREDICTED PRICE & ACTUAL PRICE**

1. **FEARTURES:** What do we already know about the data ?

**Like** for heart disease body weight, sex is features and many more.

1. **MODELLING:** Based on our problem and data , what model should we use ?

**PROBLEM** 🡪 MODEL1

1. **EXPERIMENTATION:** How could we improve/what can we try next ?

**TYPES OF DATA:**

**Structured Data :**

Data that is **organized in rows and columns**, usually stored in **relational databases** (like SQL).

**Key Features:**

* Predefined schema (fixed format)
* Easy to search, filter, and analyze
* Machine-readable and queryable

**Examples:**

* Excel sheets
* SQL tables (e.g., custome r\_id, name, age)
* Sensor data (e.g., temperature, pressure)
* Financial records (e.g., sales, expenses)

**Tools to Work With:**

* SQL
* Excel
* Pandas (Python)
* Business Intelligence tools (Power BI, Tableau)

**Unstructured Data :**

Data that **doesn’t have a predefined format** or structure. It's harder to organize and analyze without preprocessing.

**Key Features:**

* No fixed schema
* Rich in information but difficult to process directly
* Often text-heavy or multimedia

**Examples:**

* Images and Videos
* Audio files
* Emails
* PDFs and scanned documents
* Social media posts
* Chat conversations

**Tools to Work With:**

* NLP libraries (like NLTK, spaCy for text)
* Computer Vision (OpenCV, TensorFlow for images/videos)
* Hadoop/Spark (for large-scale unstructured data)

**TYPES OF EVALUATION:**

**In Data Science / Machine Learning**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| 1. Hold-out Evaluation | Data is split into training and testing sets. | 80% training, 20% testing |
| 2. Cross-Validation | Data is split into multiple folds to test generalizability. | k-Fold Cross Validation |
| 3. Metrics-based Evaluation | Uses accuracy, precision, recall, F1-score, RMSE, etc. | Classification or regression metrics |

**📊 1. Classification Metrics**

**Used when the model predicts categories/labels (e.g., spam vs. not spam).**

| **Metric** | **Description** |
| --- | --- |
| Accuracy | % of correct predictions: (TP + TN) / Total |
| Precision | TP / (TP + FP) – How many predicted positives are correct |
| Recall (Sensitivity) | TP / (TP + FN) – How many actual positives were found |
| F1 Score | Harmonic mean of precision and recall |
| AUC-ROC | Measures classification performance across thresholds |
| Confusion Matrix | Shows TP, FP, TN, FN values in a table |

**📈 2. Regression Metrics**

**Used when the model predicts continuous values (e.g., price, temperature).**

| **Metric** | **Description** |
| --- | --- |
| MAE (Mean Absolute Error) | Average of absolute prediction errors |
| MSE (Mean Squared Error) | Average of squared errors (penalizes large errors) |
| RMSE (Root MSE) | Square root of MSE |
| R² (R-squared Score) | Proportion of variance explained by the model |

**🎯 3. Recommendation System Metrics**

**Used when the system recommends items to users (e.g., Netflix, Amazon).**

| **Metric** | **Description** |
| --- | --- |
| Precision @K | Proportion of recommended items in top-k that are relevant |
| Recall @K | Proportion of relevant items that appear in top-k |
| NDCG (Normalized Discounted Cumulative Gain) | Penalizes relevant items appearing lower in the list |
| MAP (Mean Average Precision) | Average precision across all users |
| Hit Rate | Whether at least one relevant item was recommended |
| Coverage | Proportion of items/users for which the system makes recommendations |
| MRR (Mean Reciprocal Rank) | Reciprocal of the rank of the first correct item |

**FEATURES OF DATA:**

| **Feature Type** | **Description** | **Example** |
| --- | --- | --- |
| **Numerical** | Continuous or discrete numeric values | Age, Salary, Height |
| **Categorical** | Non-numeric categories | Gender, Colour, Location |
| **Ordinal** | Categories with a natural order | Low, Medium, High |
| **Binary** | Only two values | Yes/No, 0/1, True/False |
| **Textual** | Text data (often pre-processed into vectors) | Reviews, Tweets |
| **Time-based** | Date or time-related features | Timestamp, Day, Hour |
| **Image/Audio** | Pixel/audio features (often extracted automatically) | Photos, Voice recordings |

**MODELLING:**

**3 Parts to Modelling:**

* Choosing and training a model
* Tuning a model
* Model comparison

**Splitting Data: (the training, Validation and test sets)**

**DATA**

**Split**

**Test**

**Validation**

**TRAINING**

**Train your model on this (70-80%)**

**Tune Your Model on this (10-15%)**

**Test and compare on this (10-15%)**

**Generalization:**

The ability for a machine learning model to perform well on data it hasn’t seen before.

**Picking the Model:** Choosing the model according to data.

**TUNNING THE MODEL:**

Model tuning (also called **hyperparameter optimization)** is the process of adjusting the settings of a model to **improve its performance** on unseen data.

Tuning can take place on training or validation data sets.

**MODEL COMPARISON:**

**Model comparison** is the process of evaluating and selecting the **best-performing model** among multiple options based on defined metrics and validation strategies.

**Overfitting:**

The model learns the **training data too well**, including **noise and irrelevant patterns**, and performs poorly on **new/unseen data**.

**Characteristics:**

* High accuracy on training data
* Low accuracy on test/validation data
* Model is **too complex**

**Causes:**

* Too many parameters/features
* Training too long
* Very little data
* Complex models (deep trees, high-degree polynomials)

**How to Reduce Overfitting:**

* Use **simpler models**
* **Regularization** (L1/L2)
* **Cross-validation**
* **Dropout** (in neural networks)
* **More training data**
* **Prune** decision trees

**Underfitting:**

The model is **too simple** to capture the underlying pattern of the data, leading to **poor performance** on both training and test data.

**Characteristics:**

* Low accuracy on training and test data
* Model is **not learning enough**

**Causes:**

* Oversimplified model
* Too few features
* Not enough training
* High bias

**How to Reduce Underfitting:**

* Use a **more complex model**
* Train for **more epochs**
* Reduce **regularization**
* Add more **relevant features**

A graph of a function

AI-generated content may be incorrect.

A graph of a graph of underfitting

AI-generated content may be incorrect.

**Training Data**

**Data mismatch**

**EXPERIMENTATION:**

Experimentation is a key phase in building machine learning models where different techniques, parameters, and approaches are tested to identify the best-performing model. It's all about **trial, testing, and tuning.**

**TOOLS WE USE:**

**Anaconda** : all we get python library like juypter, pandas, matplotliab etc.

**Tools we used for data Analysis:**

* Pandas, matplotlib, NumPy

**Tools we used for Machine learning:**

* TensorFlow, PyTorch, XGBoost, CatBoost

**CONDA** is known as package manager.

**PANDAS**

**PANDSAS:**

**Pandas** is a powerful Python library used for **data manipulation, analysis, and cleaning**. It provides two core data structures:

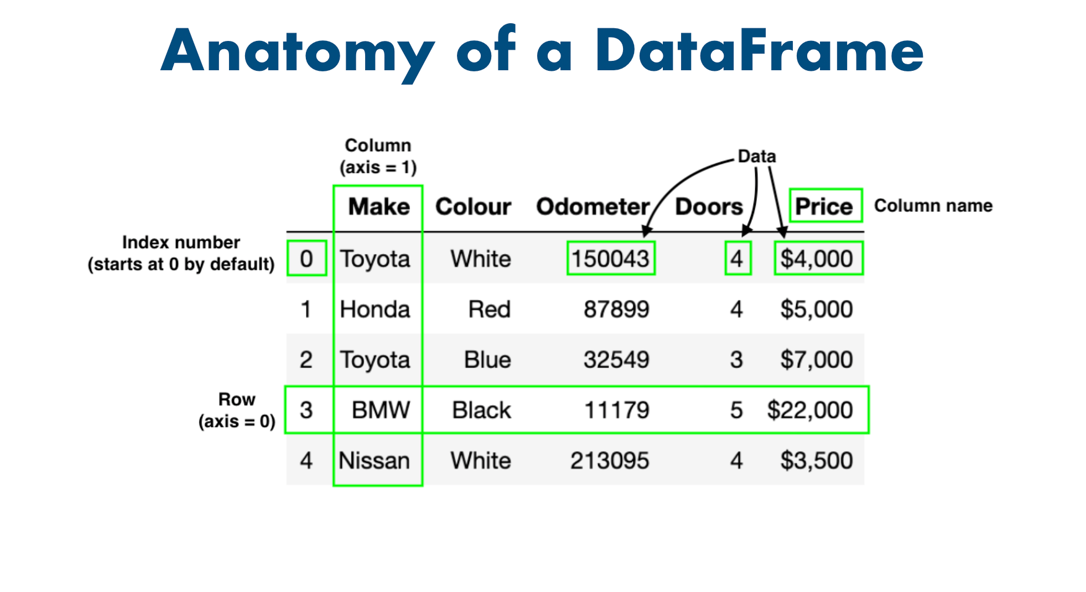
* Series – 1D labeled array
* DataFrame – 2D labeled table (like Excel/SQL table)

**Pandas:**

* Is simple to use
* Integrated with many other data science & ML python tools.
* Helps you to get your data ready for machine learning.

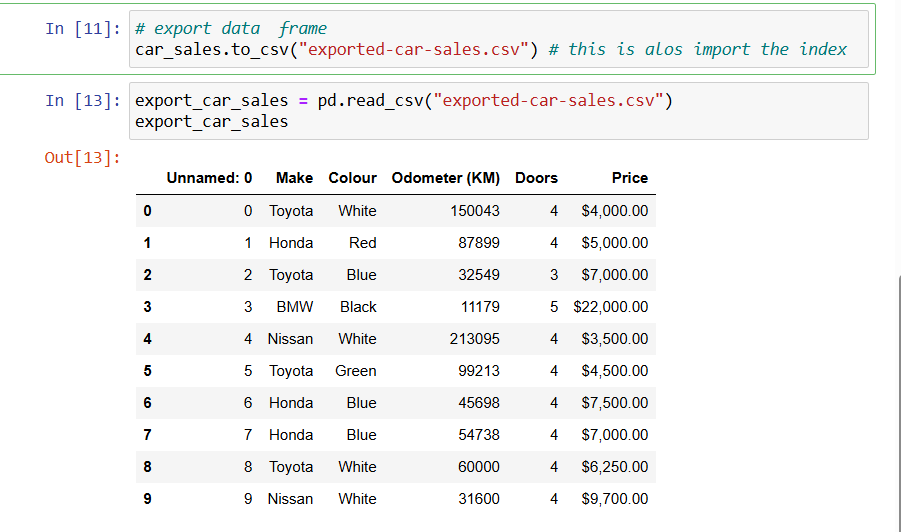
Pandas Documentation:

<https://pandas.pydata.org/pandas-docs/stable/>



axis = 0 {**row**}

axis =1 {**column**}



**Data from URLs**

In some of the lectures, you'll notice .csv files being imported from file using something like:

**heart\_disease = pd.read\_csv("data/heart-disease.csv")**

This is helpful if you have the data downloaded to your computer or in the same directory as your notebook.

But if you don't, another great feature of pandas is being able to import .csv files directly from a URL.

For example, for the heart-disease.csv file, using the read\_csv() function you can directly import it using the URL [from the course GitHub repo](https://github.com/mrdbourke/zero-to-mastery-ml):

**heart\_disease = pd.read\_csv("https://raw.githubusercontent.com/mrdbourke/zero-to-mastery-ml/master/data/heart-disease.csv")**

Note: If you're using a link from GitHub, make sure it's in the "raw" format, by clicking the raw button.

**A screenshot of a computer

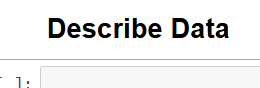
AI-generated content may be incorrect.**

Intro to pandas notebook on GitHub: <https://github.com/mrdbourke/zero-to-mastery-ml/blob/master/section-2-data-science-and-ml-tools/introduction-to-pandas.ipynb>

Intro to pandas notebook on course book website: <https://dev.mrdbourke.com/zero-to-mastery-ml/introduction-to-pandas/>

**## content**

**Esc key + M give the Heading**



**NumPy**

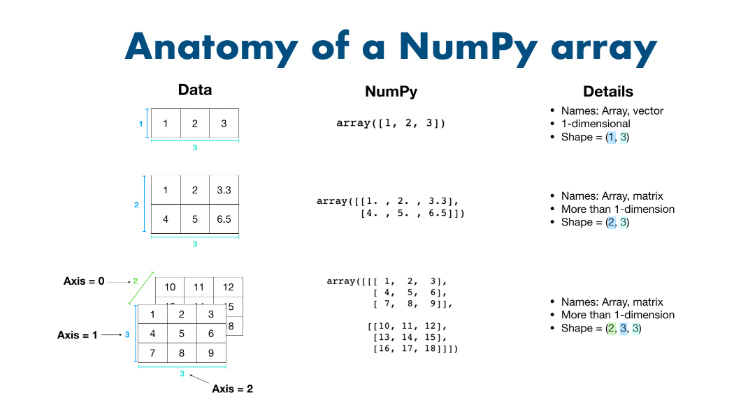
**NumPy (**Numeric Python):

**NumPy** (short for **Numerical Python**) is a powerful open-source Python library used primarily for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a wide collection of high-level mathematical functions to operate on these arrays.

| **Feature** | **Description** |
| --- | --- |
| **n-dimensional array** | The core data structure is the ndarray, a fast, space-efficient container for data. |
| **Vectorized operations** | Performs element-wise operations without explicit loops (faster and cleaner). |
| **Broadcasting** | Allows arithmetic operations between arrays of different shapes. |
| **Mathematical functions** | Supports a wide range of operations: sum, mean, std, sin, log, etc. |
| **Linear algebra support** | Includes matrix multiplication, inverse, eigenvalues, and more. |
| **Random number generation** | Comes with a random module for generating pseudo-random numbers. |
| **Integration with other libraries** | Forms the foundation for libraries like Pandas, SciPy, Scikit-learn, TensorFlow, etc. |

**NumPy:**

* It is fast.
* Behind the scenes optimizations written in C.
* Vectorization via broadcasting (avoid loops).
* Backbone of other Python scientific packages.



**NumPy intro:**

[**https://github.com/mrdbourke/zero-to-mastery-ml/blob/master/section-2-data-science-and-ml-tools/introduction-to-numpy-video.ipynb**](https://github.com/mrdbourke/zero-to-mastery-ml/blob/master/section-2-data-science-and-ml-tools/introduction-to-numpy-video.ipynb)

[**https://github.com/mrdbourke/zero-to-mastery-ml/blob/master/section-2-data-science-and-ml-tools/introduction-to-numpy.ipynb**](https://github.com/mrdbourke/zero-to-mastery-ml/blob/master/section-2-data-science-and-ml-tools/introduction-to-numpy.ipynb)

**NumPy Documentation link**

[**https://numpy.org/doc/**](https://numpy.org/doc/)

**np.ones():**

**Signature:** np**.**ones**(**shape**,** dtype**=None,** order**='C',** **\*,** like**=None)**

**Docstring:**

Return a new array of given shape and type, filled with ones.

Parameters

----------

shape : int or sequence of ints

Shape of the new array, e.g., ``(2, 3)`` or ``2``.

dtype : data-type, optional

The desired data-type for the array, e.g., `numpy.int8`. Default is

`numpy.float64`.

order : {'C', 'F'}, optional, default: C

Whether to store multi-dimensional data in row-major

(C-style) or column-major (Fortran-style) order in

memory.

like : array\_like

Reference object to allow the creation of arrays which are not

NumPy arrays. If an array-like passed in as ``like`` supports

the ``\_\_array\_function\_\_`` protocol, the result will be defined

by it. In this case, it ensures the creation of an array object

compatible with that passed in via this argument.

.. versionadded:: 1.20.0

Returns

-------

out : ndarray

Array of ones with the given shape, dtype, and order.

See Also

--------

ones\_like : Return an array of ones with shape and type of input.

empty : Return a new uninitialized array.

zeros : Return a new array setting values to zero.

full : Return a new array of given shape filled with value.

Examples

--------

>>> np.ones(5)

array([1., 1., 1., 1., 1.])

>>> np.ones((5,), dtype=int)

array([1, 1, 1, 1, 1])

>>> np.ones((2, 1))

array([[1.],

[1.]])

>>> s = (2,2)

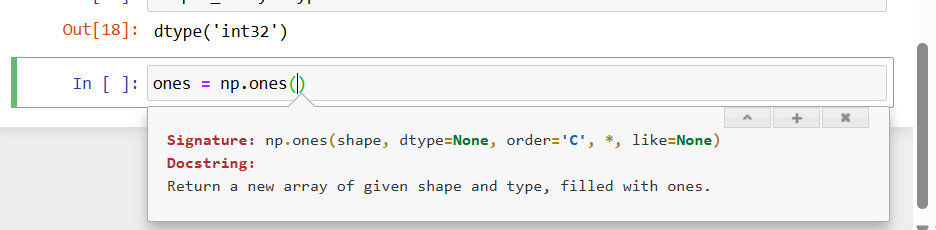
>>> np.ones(s)

array([[1., 1.],

[1., 1.]])

**File:** c:\users\himan\anaconda3\lib\site-packages\numpy\core\numeric.py

**Type:** function

****

**np.zeroes():**

zeros(shape, dtype=float, order='C', \*, like=None)

Return a new array of given shape and type, filled with zeros.

Parameters

----------

shape : int or tuple of ints

Shape of the new array, e.g., ``(2, 3)`` or ``2``.

dtype : data-type, optional

The desired data-type for the array, e.g., `numpy.int8`. Default is

`numpy.float64`.

order : {'C', 'F'}, optional, default: 'C'

Whether to store multi-dimensional data in row-major

(C-style) or column-major (Fortran-style) order in

memory.

like : array\_like

Reference object to allow the creation of arrays which are not

NumPy arrays. If an array-like passed in as ``like`` supports

the ``\_\_array\_function\_\_`` protocol, the result will be defined

by it. In this case, it ensures the creation of an array object

compatible with that passed in via this argument.

.. versionadded:: 1.20.0

Returns

-------

out : ndarray

Array of zeros with the given shape, dtype, and order.

See Also

--------

zeros\_like : Return an array of zeros with shape and type of input.

empty : Return a new uninitialized array.

ones : Return a new array setting values to one.

full : Return a new array of given shape filled with value.

Examples

--------

>>> np.zeros(5)

array([ 0., 0., 0., 0., 0.])

>>> np.zeros((5,), dtype=int)

array([0, 0, 0, 0, 0])

>>> np.zeros((2, 1))

array([[ 0.],

[ 0.]])

>>> s = (2,2)

>>> np.zeros(s)

array([[ 0., 0.],

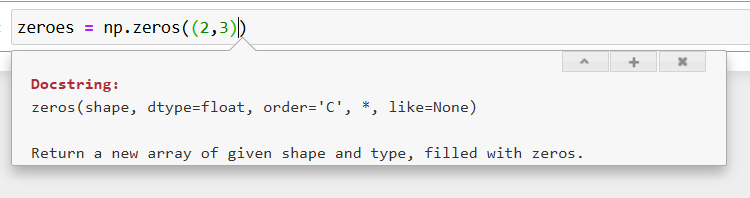
[ 0., 0.]])

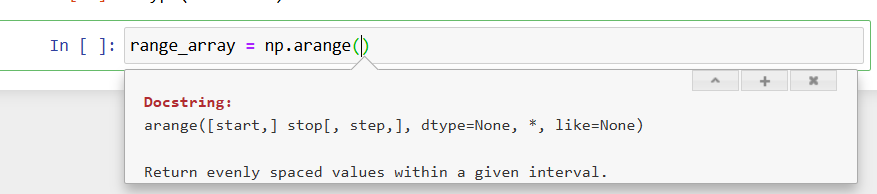
>>> np.zeros((2,), dtype=[('x', 'i4'), ('y', 'i4')]) # custom dtype

array([(0, 0), (0, 0)],

dtype=[('x', '<i4'), ('y', '<i4')])

**Type:** builtin\_function\_or\_method

****

****

**np.arnage()**

**Docstring:**

arange([start,] stop[, step,], dtype=None, \*, like=None)

Return evenly spaced values within a given interval.

Values are generated within the half-open interval ``[start, stop)``

(in other words, the interval including `start` but excluding `stop`).

For integer arguments the function is equivalent to the Python built-in

`range` function, but returns an ndarray rather than a list.

When using a non-integer step, such as 0.1, the results will often not

be consistent. It is better to use `numpy.linspace` for these cases.

Parameters

----------

start : integer or real, optional

Start of interval. The interval includes this value. The default

start value is 0.

stop : integer or real

End of interval. The interval does not include this value, except

in some cases where `step` is not an integer and floating point

round-off affects the length of `out`.

step : integer or real, optional

Spacing between values. For any output `out`, this is the distance

between two adjacent values, ``out[i+1] - out[i]``. The default

step size is 1. If `step` is specified as a position argument,

`start` must also be given.

dtype : dtype

The type of the output array. If `dtype` is not given, infer the data

type from the other input arguments.

like : array\_like

Reference object to allow the creation of arrays which are not

NumPy arrays. If an array-like passed in as ``like`` supports

the ``\_\_array\_function\_\_`` protocol, the result will be defined

by it. In this case, it ensures the creation of an array object

compatible with that passed in via this argument.

.. versionadded:: 1.20.0

Returns

-------

arange : ndarray

Array of evenly spaced values.

For floating point arguments, the length of the result is

``ceil((stop - start)/step)``. Because of floating point overflow,

this rule may result in the last element of `out` being greater

than `stop`.

See Also

--------

numpy.linspace : Evenly spaced numbers with careful handling of endpoints.

numpy.ogrid: Arrays of evenly spaced numbers in N-dimensions.

numpy.mgrid: Grid-shaped arrays of evenly spaced numbers in N-dimensions.

Examples

--------

>>> np.arange(3)

array([0, 1, 2])

>>> np.arange(3.0)

array([ 0., 1., 2.])

>>> np.arange(3,7)

array([3, 4, 5, 6])

>>> np.arange(3,7,2)

array([3, 5])

**Type:** builtin\_function\_or\_method

**np.random.randint():**

**Docstring:**

randint(low, high=None, size=None, dtype=int)

Return random integers from `low` (inclusive) to `high` (exclusive).

Return random integers from the "discrete uniform" distribution of

the specified dtype in the "half-open" interval [`low`, `high`). If

`high` is None (the default), then results are from [0, `low`).

.. note::

New code should use the ``integers`` method of a ``default\_rng()``

instance instead; please see the :ref:`random-quick-start`.

Parameters

----------

low : int or array-like of ints

Lowest (signed) integers to be drawn from the distribution (unless

``high=None``, in which case this parameter is one above the

\*highest\* such integer).

high : int or array-like of ints, optional

If provided, one above the largest (signed) integer to be drawn

from the distribution (see above for behavior if ``high=None``).

If array-like, must contain integer values

size : int or tuple of ints, optional

Output shape. If the given shape is, e.g., ``(m, n, k)``, then

``m \* n \* k`` samples are drawn. Default is None, in which case a

single value is returned.

dtype : dtype, optional

Desired dtype of the result. Byteorder must be native.

The default value is int.

.. versionadded:: 1.11.0

Returns

-------

out : int or ndarray of ints

`size`-shaped array of random integers from the appropriate

distribution, or a single such random int if `size` not provided.

See Also

--------

random\_integers : similar to `randint`, only for the closed

interval [`low`, `high`], and 1 is the lowest value if `high` is

omitted.

Generator.integers: which should be used for new code.

Examples

--------

>>> np.random.randint(2, size=10)

array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0]) # random

>>> np.random.randint(1, size=10)

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

>>> np.random.randint(5, size=(2, 4))

array([[4, 0, 2, 1], # random

[3, 2, 2, 0]])

Generate a 1 x 3 array with 3 different upper bounds

>>> np.random.randint(1, [3, 5, 10])

array([2, 2, 9]) # random

Generate a 1 by 3 array with 3 different lower bounds

>>> np.random.randint([1, 5, 7], 10)

array([9, 8, 7]) # random

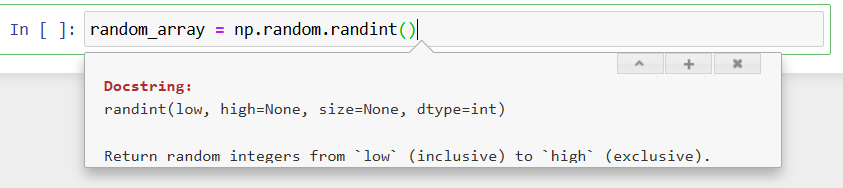
Generate a 2 by 4 array using broadcasting with dtype of uint8

>>> np.random.randint([1, 3, 5, 7], [[10], [20]], dtype=np.uint8)

array([[ 8, 6, 9, 7], # random

[ 1, 16, 9, 12]], dtype=uint8)

**Type:** builtin\_function\_or\_method



**np.random.random():**

**A screenshot of a computer

AI-generated content may be incorrect.**

**np.random.rand():**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Np.unique()**

**Signature:**

np**.**unique**(**

ar**,**

return\_index**=False,**

return\_inverse**=False,**

return\_counts**=False,**

axis**=None,**

**)**

**Docstring:**

Find the unique elements of an array.

Returns the sorted unique elements of an array. There are three optional

outputs in addition to the unique elements:

\* the indices of the input array that give the unique values

\* the indices of the unique array that reconstruct the input array

\* the number of times each unique value comes up in the input array

Parameters

----------

ar : array\_like

Input array. Unless `axis` is specified, this will be flattened if it

is not already 1-D.

return\_index : bool, optional

If True, also return the indices of `ar` (along the specified axis,

if provided, or in the flattened array) that result in the unique array.

return\_inverse : bool, optional

If True, also return the indices of the unique array (for the specified

axis, if provided) that can be used to reconstruct `ar`.

return\_counts : bool, optional

If True, also return the number of times each unique item appears

in `ar`.

.. versionadded:: 1.9.0

axis : int or None, optional

The axis to operate on. If None, `ar` will be flattened. If an integer,

the subarrays indexed by the given axis will be flattened and treated

as the elements of a 1-D array with the dimension of the given axis,

see the notes for more details. Object arrays or structured arrays

that contain objects are not supported if the `axis` kwarg is used. The

default is None.

.. versionadded:: 1.13.0

Returns

-------

unique : ndarray

The sorted unique values.

unique\_indices : ndarray, optional

The indices of the first occurrences of the unique values in the

original array. Only provided if `return\_index` is True.

unique\_inverse : ndarray, optional

The indices to reconstruct the original array from the

unique array. Only provided if `return\_inverse` is True.

unique\_counts : ndarray, optional

The number of times each of the unique values comes up in the

original array. Only provided if `return\_counts` is True.

.. versionadded:: 1.9.0

See Also

--------

numpy.lib.arraysetops : Module with a number of other functions for

performing set operations on arrays.

repeat : Repeat elements of an array.

Notes

-----

When an axis is specified the subarrays indexed by the axis are sorted.

This is done by making the specified axis the first dimension of the array

(move the axis to the first dimension to keep the order of the other axes)

and then flattening the subarrays in C order. The flattened subarrays are

then viewed as a structured type with each element given a label, with the

effect that we end up with a 1-D array of structured types that can be

treated in the same way as any other 1-D array. The result is that the

flattened subarrays are sorted in lexicographic order starting with the

first element.

.. versionchanged: NumPy 1.21

If nan values are in the input array, a single nan is put

to the end of the sorted unique values.

Also for complex arrays all NaN values are considered equivalent

(no matter whether the NaN is in the real or imaginary part).

As the representant for the returned array the smallest one in the

lexicographical order is chosen - see np.sort for how the lexicographical

order is defined for complex arrays.

Examples

--------

>>> np.unique([1, 1, 2, 2, 3, 3])

array([1, 2, 3])

>>> a = np.array([[1, 1], [2, 3]])

>>> np.unique(a)

array([1, 2, 3])

Return the unique rows of a 2D array

>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])

>>> np.unique(a, axis=0)

array([[1, 0, 0], [2, 3, 4]])

Return the indices of the original array that give the unique values:

>>> a = np.array(['a', 'b', 'b', 'c', 'a'])

>>> u, indices = np.unique(a, return\_index=True)

>>> u

array(['a', 'b', 'c'], dtype='<U1')

>>> indices

array([0, 1, 3])

>>> a[indices]

array(['a', 'b', 'c'], dtype='<U1')

Reconstruct the input array from the unique values and inverse:

>>> a = np.array([1, 2, 6, 4, 2, 3, 2])

>>> u, indices = np.unique(a, return\_inverse=True)

>>> u

array([1, 2, 3, 4, 6])

>>> indices

array([0, 1, 4, 3, 1, 2, 1])

>>> u[indices]

array([1, 2, 6, 4, 2, 3, 2])

Reconstruct the input values from the unique values and counts:

>>> a = np.array([1, 2, 6, 4, 2, 3, 2])

>>> values, counts = np.unique(a, return\_counts=True)

>>> values

array([1, 2, 3, 4, 6])

>>> counts

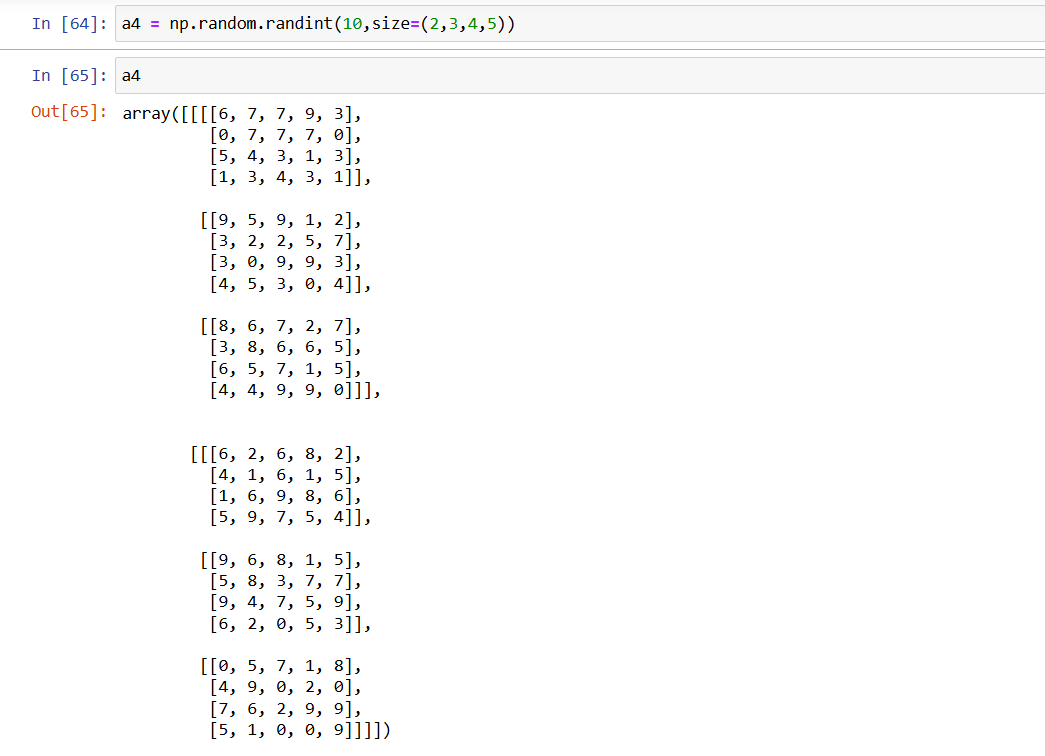
array([1, 3, 1, 1, 1])

>>> np.repeat(values, counts)

array([1, 2, 2, 2, 3, 4, 6]) # original order not preserved

**File:** c:\users\himan\anaconda3\lib\site-packages\numpy\lib\arraysetops.py

**Type:** function

  
SIZE=(2,3,4,5)

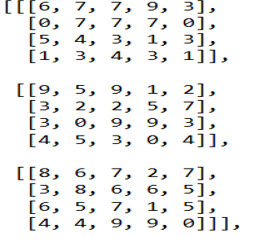
**5:** **** the element in the metix or array.

**4:**

**A number with black lines

AI-generated content may be incorrect.** the all these element in the metix or array

**3:**

****

**2:**

**A white background with black numbers

AI-generated content may be incorrect.**

**Standard Deviation and Variance:**

*Deviation means how far from the normal*

**Standard Deviation**

The Standard Deviation is a measure of how spread out numbers are.

Its symbol is **σ** (the greek letter sigma)

The formula is easy: it is the**square root** of the **Variance.** So now you ask, "What is the Variance?"

**Variance**

The Variance is defined as:

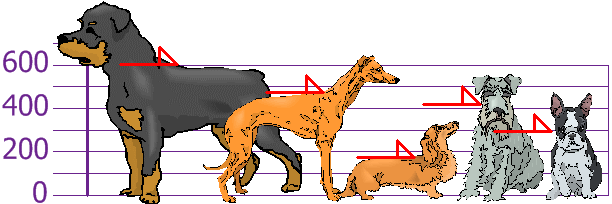
The average of the **squared** differences from the Mean.

To calculate the variance follow these steps:

* Calculate the [Mean](https://www.mathsisfun.com/mean.html) (the simple average of the numbers)
* Then for each number: subtract the Mean and square the result (the *squared difference*).
* Then calculate the average of those squared differences. ([Why Square?](https://www.mathsisfun.com/data/standard-deviation.html#WhySquare))

**Example**

You and your friends have just measured the heights of your dogs (in millimeters):



The heights (at the shoulders) are: 600 mm, 470 mm, 170 mm, 430 mm and 300 mm.

Find out the Mean, the Variance, and the Standard Deviation.

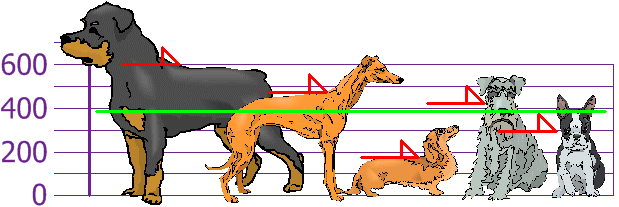
Your first step is to find the Mean:

Answer:

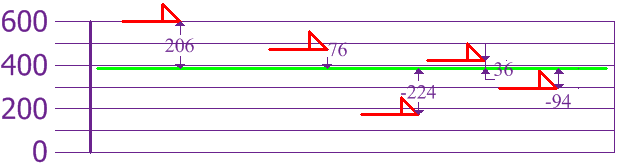
so the mean (average) height is 394 mm. Let's plot this on the chart:

A close-up of numbers

AI-generated content may be incorrect.



Now we calculate each dog's difference from the Mean:



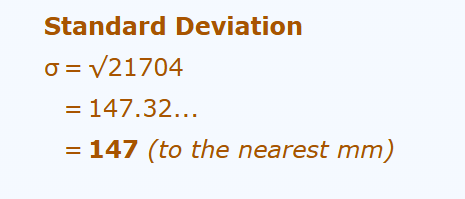
To calculate the Variance, take each difference, square it, and then average the result:

A math equations on a white background

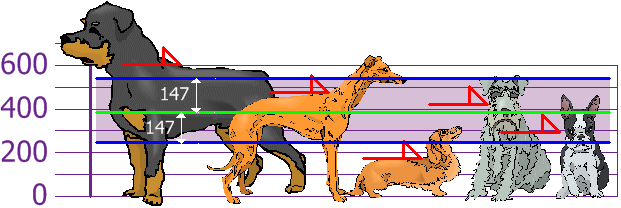
AI-generated content may be incorrect.

So the Variance is **21,704**

And the Standard Deviation is just the square root of Variance, so:



And the good thing about the Standard Deviation is that it is useful. Now we can show which heights are within one Standard Deviation (147 mm) of the Mean:



So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.

Rottweilers **are** tall dogs. And Dachshunds **are** a bit short, right?

**Using**

We can expect about 68% of values to be within plus-or-minus 1 standard deviation.

Read [Standard Normal Distribution](https://www.mathsisfun.com/data/standard-normal-distribution.html) to learn more.

Also try the [Standard Deviation Calculator](https://www.mathsisfun.com/data/standard-deviation-calculator.html).

**But ... there is a small change with Sample Data**

Our example has been for a **Population** (the 5 dogs are the only dogs we are interested in).

But if the data is a **Sample** (a selection taken from a bigger Population), then the calculation changes!

When we have "N" data values that are:

* **The Population**: divide by **N** when calculating Variance (like we did)
* **A Sample**: divide by **N-1** when calculating Variance

All other calculations stay the same, including how we calculated the mean.

Example: if our 5 dogs are just a **sample** of a bigger population of dogs, we divide by **4 instead of 5** like this:

Sample Variance = 108,520 / **4** = **27,130**

Sample Standard Deviation = √27,130 = **165** (to the nearest mm)

Think of it as a "correction" when our data is only a sample.

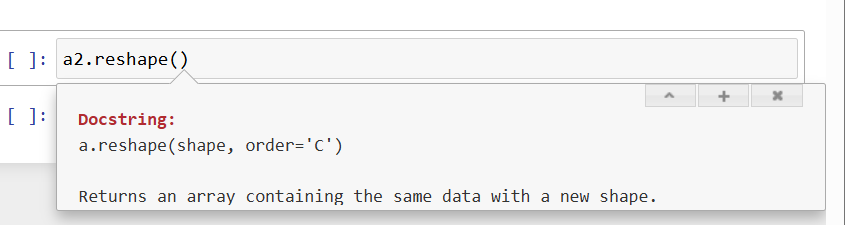
**Formulas**

Here are the two formulas, explained at [Standard Deviation Formulas](https://www.mathsisfun.com/data/standard-deviation-formulas.html) if you want to know more:

|  |  |  |
| --- | --- | --- |
| The "**Population** Standard Deviation": |  |  |
| The "**Sample** Standard Deviation**":** |  |  |

Looks complicated, but the important change is to  
divide by **N-1** (instead of **N**) when calculating a Sample Standard Deviation.

**%timeit** is used to calculate the how much time the function it take the time to run.



The multiplication possible if one of the **shape** is one and or equal shape

**MATRIX MULTIPLICATION:**

<https://www.mathsisfun.com/algebra/matrix-multiplying.html>

**How to Multiply Matrices**

A Matrix is an array of numbers:

A number and text on a white background

AI-generated content may be incorrect.  
 Matrix  
(This one has 2 Rows and 3 Columns)

To multiply a matrix by a single number, multiply it by every element of the matrix:

A diagram of numbers and symbols

AI-generated content may be incorrect.

These are the calculations:

|  |  |
| --- | --- |
| 2×4=8 | 2×0=0 |
| 2×1=2 | 2×-9=-18 |

We call the number (**2** in this case) a **scalar**: a single number used to **scale** (↕) the values in the matrix. And this is called "scalar multiplication".

**Multiplying a Matrix by Another Matrix**

But to multiply a matrix **by another matrix** we need to do the [dot product](https://www.mathsisfun.com/algebra/vectors-dot-product.html) of rows and columns ... what does that mean? Let us see with an example:

To work out the answer for the **1st row** and **1st column**:

A diagram of a number equation

AI-generated content may be incorrect.

The **dot product** is where we **multiply matching members**, then sum up:

(1, 2, 3) • (7, 9, 11)= 1×7 + 2×9 + 3×11

= 58

We match the 1st members (1 and 7), multiply them, likewise for the 2nd members (2 and 9) and the 3rd members (3 and 11), and finally sum them up.

Want to see another example? Here it is for the **1st row** and **2nd column**:

A math equation with numbers and lines

AI-generated content may be incorrect.

(1, 2, 3) • (8, 10, 12)= 1×8 + 2×10 + 3×12

= 64

We can do the same thing for the **2nd row** and **1st column**:

(4, 5, 6) • (7, 9, 11)= 4×7 + 5×9 + 6×11

= 139

And for the **2nd row** and **2nd column**:

(4, 5, 6) • (8, 10, 12)= 4×8 + 5×10 + 6×12

= 154

And we get:

A math equation with numbers and words

AI-generated content may be incorrect.

DONE!

**Why Do It This Way?**

This may seem an odd and complicated way of multiplying, but it is necessary!

I can give you a real-life example to illustrate why we multiply matrices in this way.

Example: The local shop sells 3 types of pies.

* Apple pies cost **$3** each
* Cherry pies cost **$4** each
* Blueberry pies cost **$2** each

And this is how many they sold in 4 days:

A calendar with numbers and letters

AI-generated content may be incorrect.

Now think about this ... the **value of sales** for Monday is calculated this way:

Apple pie value + Cherry pie value + Blueberry pie value

$3×13 + $4×8 + $2×6 *=* $83

So it is, in fact, the "dot product" of prices and how many were sold:

($3, $4, $2) • (13, 8, 6) = $3×13 + $4×8 + $2×6  
    = $83

We **match** the price to how many sold, **multiply** each, then **sum** the result.

**In other words:**

* The sales for Monday were: Apple pies: **$3×13=$39**, Cherry pies: **$4×8=$32**, and Blueberry pies: **$2×6=$12**. Together that is $39 + $32 + $12 *=* **$83**
* And for Tuesday: **$3×9 +** **$4×7 + $2×4 =** **$63**
* And for Wednesday: **$3×7 +** **$4×4 + $2×0 =** **$37**
* And for Thursday: **$3×15 +** **$4×6 + $2×3 =** **$75**

So it is important to match each price to each quantity.

Now you know why we use the "dot product".

And here is the full result in Matrix form:

A calculator with numbers and a coin

AI-generated content may be incorrect.

They sold **$83** worth of pies on Monday, **$63** on Tuesday, and so on.

Try those values in the [Matrix Calculator](https://www.mathsisfun.com/algebra/matrix-calculator.html) to see if they work.

In fact, have a play with it, see how it all works!

**Rows and Columns**

To show how many rows and columns a matrix has we often write **rows×columns**.

Example: This matrix is **2×3** (2 rows by 3 columns):

A number on a blue background

AI-generated content may be incorrect.

When we do multiplication:

* The number of **columns of the 1st matrix** must equal the number of **rows of the 2nd matrix**
* And the result will have the same number of **rows as the 1st matrix**, and the same number of **columns as the 2nd matrix**

Example from before:

A math problem with numbers and arrows

AI-generated content may be incorrect.

***In General:***

To multiply an  **m×n**  matrix by an **n×p** matrix, the **n**s must be the same,  
and the result is an **m×p** matrix.

So ... multiplying a **1×3** by a **3×1** gets a **1×1** result:

A number and equation on a white background

AI-generated content may be incorrect.

But multiplying a **3×1** by a **1×3** gets a **3×3** result:

A number of numbers on a white background

AI-generated content may be incorrect.

**Identity Matrix**

The "Identity Matrix" is the matrix equivalent of the number "1":

A number and line with numbers

AI-generated content may be incorrect.  
A 3×3 Identity Matrix

* It is "square" (has same number of rows as columns)
* It can be large or small (2×2, 100×100, ... whatever)
* It has **1**s on the main diagonal and **0**s everywhere else
* Its symbol is the capital letter **I**

It is a **special matrix**, because when we multiply by it, the original is unchanged:

A number of letters and numbers

AI-generated content may be incorrect.

**Order of Multiplication**

In arithmetic we are used to:

A number and equal sign

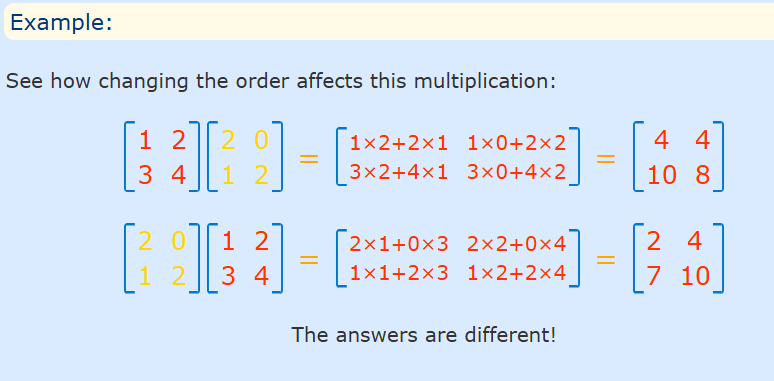
AI-generated content may be incorrect.  
(The [Commutative Law](https://www.mathsisfun.com/associative-commutative-distributive.html) of Multiplication)

But this is **not** generally true for matrices (matrix multiplication is **not commutative**):

AB ≠ BA

When we change the order of multiplication, the answer is (usually) **different**.

=



**Matplotlib:**

**Matplotlib:**

**Matplotlib** is a comprehensive library in Python used for creating **static**, **animated**, and **interactive visualizations**. It is one of the most widely used libraries for data visualization in Python, especially for 2D plotting.

| **Feature** | **Description** |
| --- | --- |
| **2D plotting** | Line plots, bar charts, scatter plots, histograms, pie charts, etc. |
| **Customizable** | Titles, labels, colours, markers, and annotations can be extensively customized. |
| **Subplots and figures** | Supports multiple plots in one figure and complex layouts. |
| **Interactive backends** | Can be used with GUIs like Tkinter, PyQt, or web apps like Jupyter. |
| **Integration** | Works well with NumPy, Pandas, and other scientific libraries. |

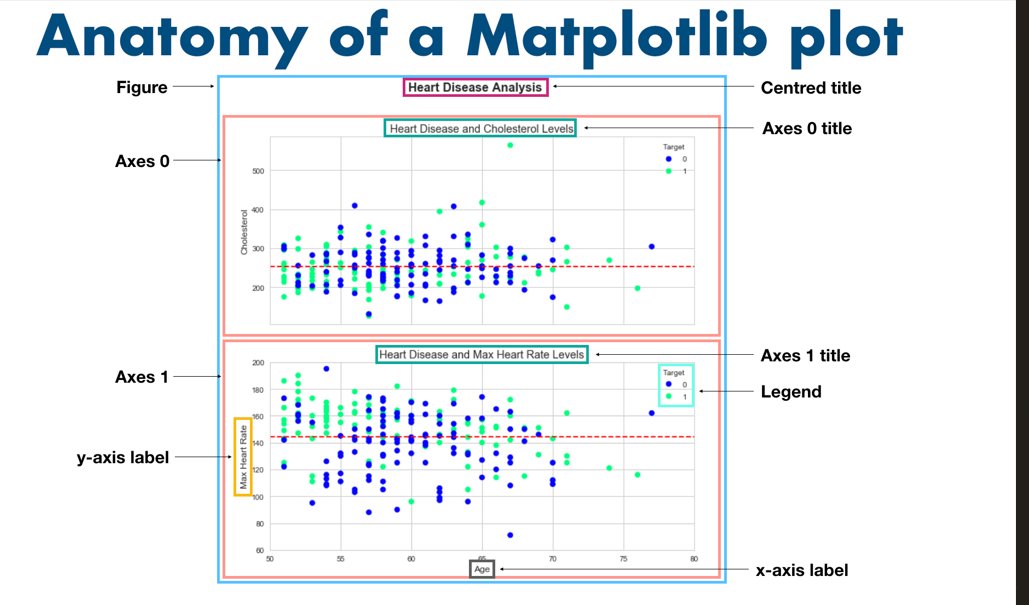
| **Function** | **Description** |
| --- | --- |
| **plt.plot()** | Line plot |
| **plt.scatter()** | Scatter plot |
| **plt.bar(), plt.barh()** | Bar and horizontal bar charts |
| **plt.hist()** | Histogram |
| **plt.pie()** | Pie chart |
| **plt.xlabel(), plt.ylabel()** | Axis labels |
| **plt.title()** | Plot title |
| **plt.legend()** | Display legend |
| **plt.grid()** | Show grid |
| **plt.subplot()** | Create multiple subplots |
| **plt.savefig()** | Save figure as image |
| **plt.show()** | Render/display the plot |

**Matplotlib:**

* Built on NumPy arrays (and Python)
* Integrates directly with pandas.
* Can create basic and advanced plots.
* Simple to use

**Documentation:**

**https://matplotlib.org/3.1.1/contents.html**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**SciKit-Learn(sklearn)**

**Scikit-learn:**

**Scikit-learn** (or sklearn) is a powerful, easy-to-use **machine learning library in Python**. It is built on top of NumPy, SciPy, and matplotlib, and is used for **data mining, data analysis**, and **machine learning** tasks such as classification, regression, clustering, dimensionality reduction, and model selection.

| **Feature** | **Description** |
| --- | --- |
| **Algorithms** | Built-in support for classification, regression, clustering, etc. |
| **Preprocessing** | Tools for data normalization, encoding, imputation, and scaling |
| **Model selection** | Cross-validation, grid search, and evaluation metrics |
| **Pipelines** | Combine preprocessing + modeling into reusable workflows |
| **Easy API** | Consistent fit–predict–evaluate interface across models |
| **Integration** | Works seamlessly with Pandas, NumPy, and matplotlib |

| **Module** | **Purpose** |
| --- | --- |
| sklearn.linear\_model | Linear models like LinearRegression, LogisticRegression |
| sklearn.ensemble | Ensemble methods like RandomForest, GradientBoosting |
| sklearn.tree | Decision trees |
| sklearn.cluster | Clustering (e.g., KMeans, DBSCAN) |
| sklearn.svm | Support Vector Machines |
| sklearn.model\_selection | Train/test split, cross-validation, grid search |
| sklearn.preprocessing | Scaling, encoding, imputation, etc. |
| sklearn.metrics | Accuracy, precision, recall, MSE, etc. |

**Scikit-learn:**

* Built on NumPy and matplotlib (and python).
* Has many in-built machine learning models.
* Methods to evaluate your machine learning models.
* Very well-designed API.

**Documentation:**

[**https://scikit-learn.org/stable/user\_guide.html**](https://scikit-learn.org/stable/user_guide.html)

**A Scikit-Learn Workflow**

**Evaluate the Model**

**Fit the Model to the data and make a prediction**

**GET DATA A READY**

**Pick a Model**

**(to suit your problem)**

**Save and reload your train model**

**Improve through experimentation**

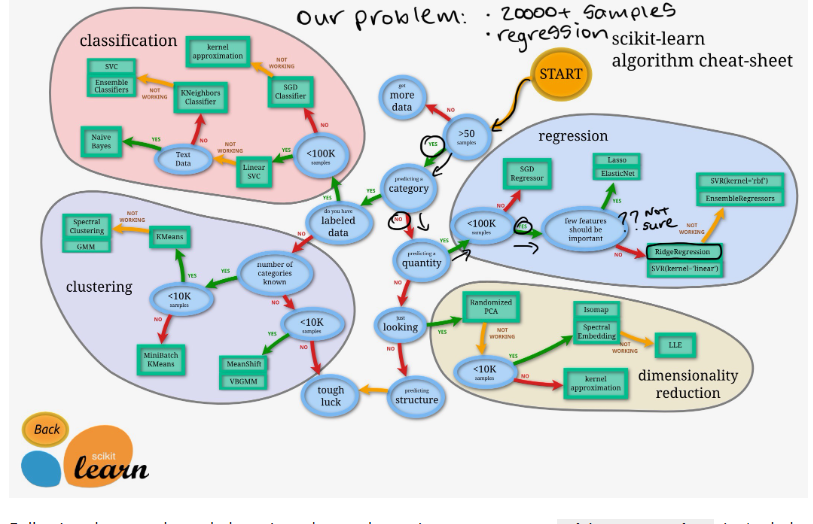
**One Hot Encoding:**

**One-Hot Encoding** is a method to convert **categorical data** into a format that can be provided to **ML algorithms**, which typically require numeric input.

| **Colour** |
| --- |
| Red |
| Blue |
| Green |

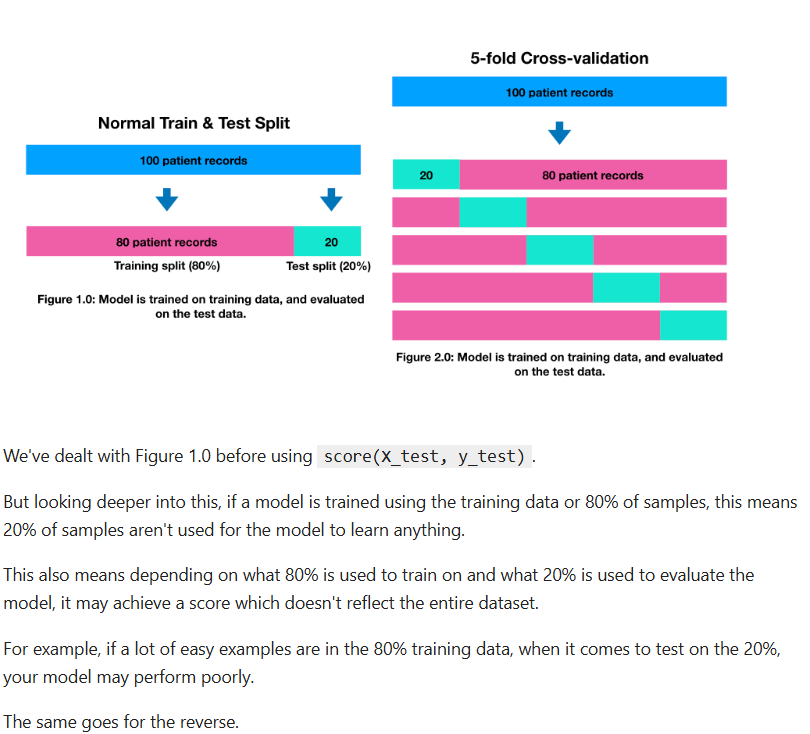
**One-hot encoding will turn this into:**

| **Colour\_Blue** | **Colour\_Green** | **Colour\_Red** |
| --- | --- | --- |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |



<https://scikit-learn.org/stable/machine_learning_map.html>

**CROOS – VALIDATION:**

****