

Methods on using: Machine learning

Contents

Mandatory libraries to know:	2
General definition:	2
What is supervised learning?	2
What is unsupervised learning?	3
The difference between supervised and unsupervised?	4
General method of Machine learning: THEORY & DEFINITIONS	5
Introduction	5
Step 1: Gather the data – Target leakage:	5
Target leakage:	5
Train-Test Contamination	6
Step 2: Prepare de data:	7
Missing values:	8
Categorical values:	8
General method of Machine learning: PRACTICE GUIDE.....	9
1) Data analysis	9
Scaling and Normalization of Data - NEED TO RESEARCH BETTER.....	10
2) Import Data, split your Data & Define your features (Train/Valid)	11
3) Address missing values, encoding categorical Datas WITH PIPELINES :	12
Pipelines & Optimization (Need more research) :	12
4) Evaluate and optimize your model with your validation DataSet :	14

Mandatory libraries to know:

Machine learning with Python involve working and manipulating Datasets, and displaying your results, you need:

- 1) **Pandas** = Library that you will use to manipulate Datasets -> [Pandas](#)
- 2) **NumPy** = For Array and mathematical (Use NumPy like you would use Matlab) -> [NumPy](#)
- 3) **Matplotlib** = For displaying your results with graphs and charts [Matplotlib](#)
- 4) **Scikit-learn** = For calling algorithm such ad Random Forest [scikit-learn](#)

Off course you will need a good understanding of Python specially List and Dictionaries comprehensive forms.

General definition:

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

What's required to create good machine learning systems?

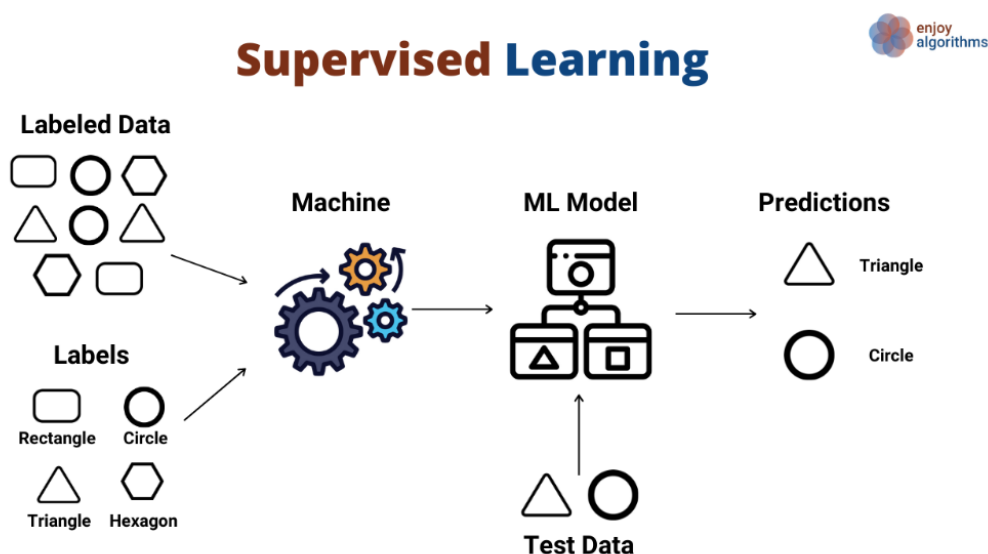
- Data preparation capabilities.
- Algorithms – basic and advanced.
- Automation and iterative processes.
- Scalability.
- Ensemble modeling.

Did you know?

- In machine learning, a target is called a label.
- In statistics, a target is called a dependent variable.
- A variable in statistics is called a feature in machine learning.
- A transformation in statistics is called feature creation in machine learning.

What is supervised learning?

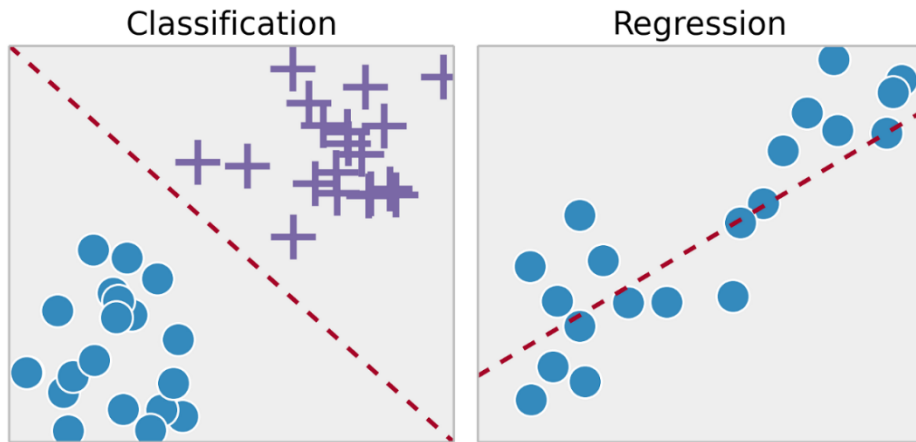
Supervised learning is a machine learning approach that's defined by its **use of labeled datasets**. **These datasets are designed to train or "supervise" algorithms** into classifying data or predicting outcomes accurately. Using labeled inputs and outputs, the model can measure its accuracy and learn over time.



Supervised learning can be separated into two types of problems when data mining: classification and regression:

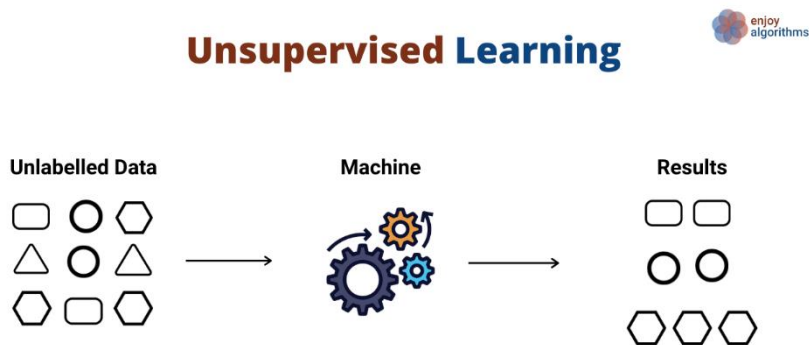
Classification problems use an algorithm to accurately **assign test data into specific categories**, such as separating apples from oranges, or to classify spam in a separate folder from your inbox. Linear classifiers, support vector machines, decision trees and random forest are all common types of classification algorithms.

Regression is another type of supervised learning method that uses an algorithm to **understand the relationship between dependent and independent variables**. Regression models are helpful for predicting numerical values based on different data points, such as sales revenue projections for a given business. Some popular regression algorithms are linear regression, logistic regression and polynomial regression.



What is unsupervised learning?

Unsupervised learning uses machine learning algorithms to **analyze and cluster unlabeled data sets**. These algorithms **discover hidden patterns in data without the need for human intervention** (hence, they are “unsupervised”).



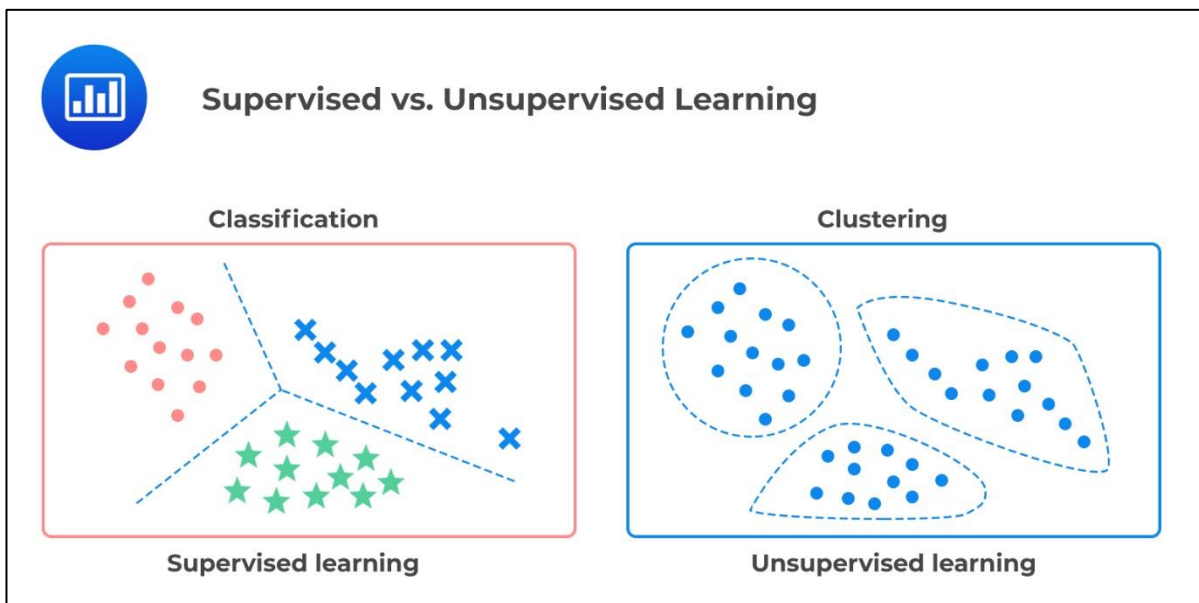
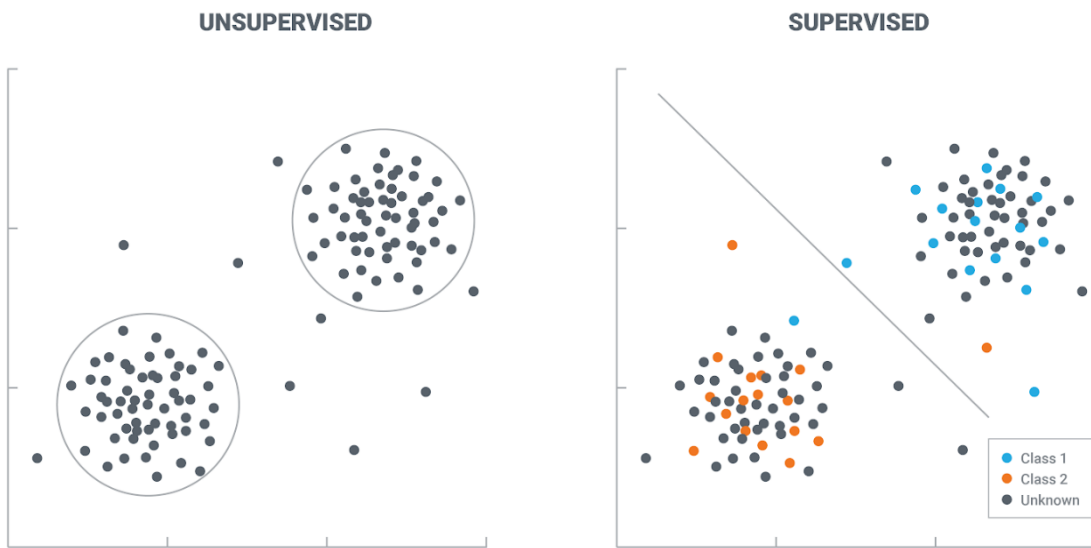
Unsupervised learning models are used for three main tasks: clustering, association and dimensionality reduction:

Clustering is a data mining technique for **grouping unlabeled data based on their similarities or differences**. For example, K-means clustering algorithms assign similar data points into groups, where the K value represents the size of the grouping and granularity. This technique is helpful for market segmentation, image compression, etc.

Association is another type of unsupervised learning method that uses different rules to **find relationships between variables in a given dataset**. These methods are frequently used for market basket analysis and recommendation engines, along the lines of “Customers Who Bought This Item Also Bought” recommendations.

Dimensionality reduction is a learning technique used when the number of features (or dimensions) in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the data integrity. Often, this technique is used in the preprocessing data stage, such as when autoencoders remove noise from visual data to improve picture quality.

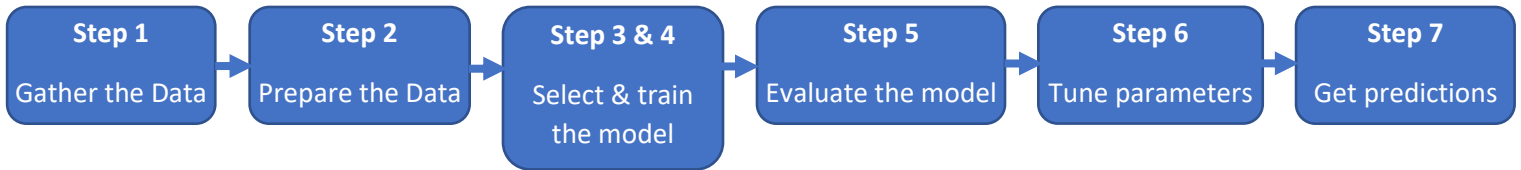
The difference between supervised and unsupervised?



General method of Machine learning: THEORY & DEFINITIONS

Introduction

When applying machine learning to real-world data, there are a lot of steps involved in the process -- starting with collecting the data and ending with generating predictions.



It all begins with Step 1: Gather the data. In industry, there are important considerations you need to take into account when building a dataset, such as target leakage.

Step 1: Gather the data – Target leakage:

Data leakage happens when your training data contains information about the target, but similar data will not be available when the model is used for prediction.

This leads to high performance on the training set, but the model will perform poorly in production.

In other words, leakage causes a model to look accurate until you start making decisions with the model, and then the model becomes very inaccurate.

- There are two main types of leakage: **target leakage** and **train-test contamination**.

Target leakage:

Target leakage occurs when your predictors include data that will not be available at the time you make predictions. It is important to think about target leakage in terms of the timing or chronological order that data becomes available, not merely whether a feature helps make good predictions.

An example will be helpful: Imagine you want to predict who will get sick with pneumonia. The top few rows of your raw data look like this:

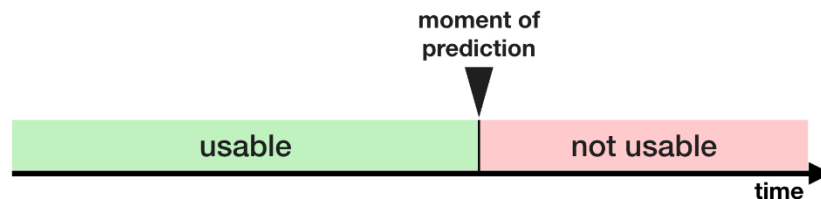
Got_pneumonia	Age	Weight	Male	Took_antibiotic	...
False	65	100	False	False	...
False	72	130	True	False	...
True	58	100	False	True	...

People take antibiotic medicines after getting pneumonia in order to recover. The raw data shows a strong relationship between those columns, but **took_antibiotic_medicine** is frequently changed after the value for **got_pneumonia** is determined. This is target leakage.

The model would see that anyone who has a value of `False` for `took_antibiotic_medicine` didn't have pneumonia. Since validation data comes from the same source as training data, the pattern will repeat itself in validation, and the model will have great validation (or cross-validation) scores.

But the model will be very inaccurate when subsequently deployed in the real world, because even patients who will get pneumonia won't have received antibiotics yet when we need to make predictions about their future health.

To prevent this type of data leakage, any variable updated (or created) after the target value is realized should be excluded



Train-Test Contamination

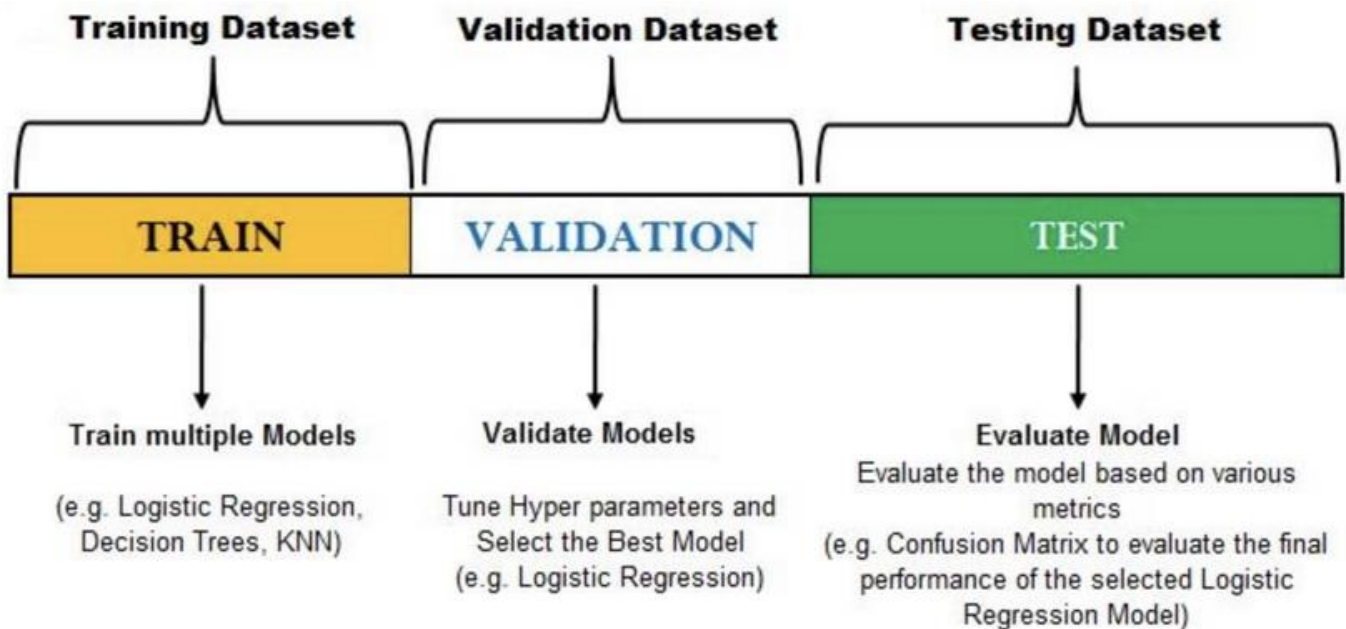
A different type of leak occurs when you aren't careful to distinguish training data from validation data.

Recall that validation is meant to be a measure of how the model does on data that it hasn't considered before. You can corrupt this process in subtle ways if the validation data affects the preprocessing behavior. This is sometimes called train-test contamination.

For example, imagine you run preprocessing (like fitting an imputer for missing values) before calling `train_test_split()`.

The end result ?

Your model may get good validation scores, giving you great confidence in it, but perform poorly when you deploy it to make decisions.



Step 2: Prepare de data:

Preparing the data is a difficult step in building your machine learning project.

Each dataset is different and highly specific to the project, building a model for predicting the house value among real estate will be different than evaluating if a patient has lung cancer.

Data preprocessing, is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

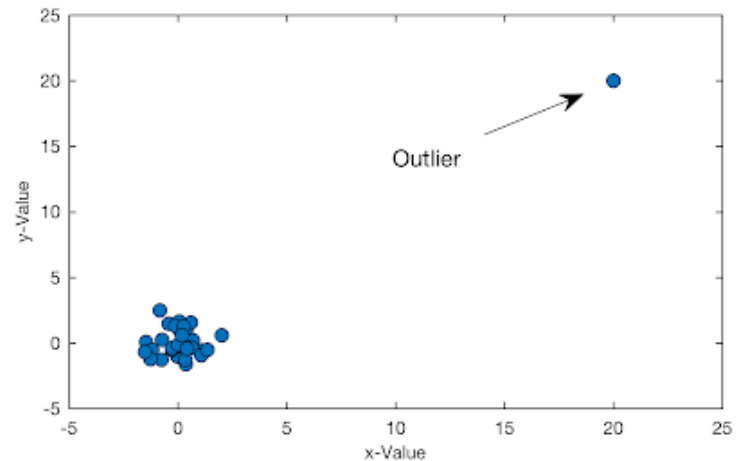
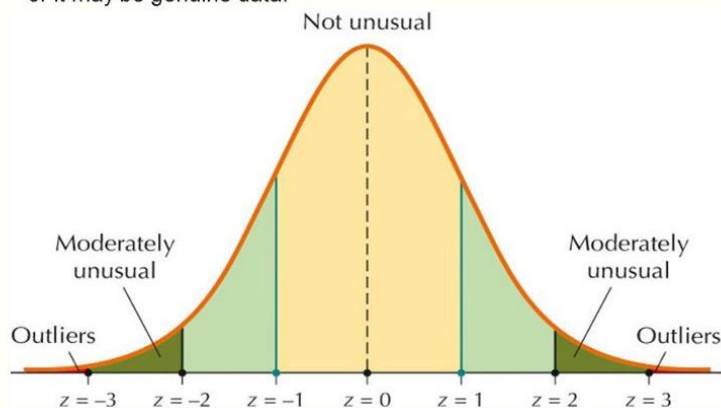
The data preparation process can be complicated by issues such as:

Missing or incomplete records: It is difficult to get every data point for every record in a dataset. Missing data sometimes appears as empty cells, values (e.g., NULL or N/A, NaN), or a particular character, such as a question mark.

AGE	SEXE	WEIGHT
65	Male	NaN
58	Female	123
32	NaN	145

Outliers or anomalies: unexpected values often surface in a distribution of values, especially when working with data from unknown sources which lack poor data validation controls.

An **outlier** is an extremely large or extremely small data value relative to the rest of the data set. It may represent a data entry error, or it may be genuine data.




Inconsistent values and non-standardized categorical variables. When combining data from multiple sources, we can end up with variations in variables like company names or states. For instance, a state in one system could be "Texas," while in another it could be "TX." Finding all variations and correctly standardizing will improve the model accuracy.

AGE	STATE	AGE	STATE
65	Texas	65	Tx
58	TX	58	Tx
32	Tx	32	Tx

Missing values:

Imputation fills in the missing values with some number. For instance, we can fill in the mean value along each column.

Age	Weight
20.0	175.0
32.0	123.0
NaN	135.0



Age	Weight
20.0	175.0
32.0	123.0
26	135.0

The imputed value won't be exactly right since it's calculated, but it usually leads to more accurate models than you would get from dropping the column entirely.

We use SimpleImputer to achieve this: [SimpleImputer](#)

```
from sklearn.impute import SimpleImputer

# Imputation
my_imputer = SimpleImputer()
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns
```

Categorical values:


- 1) Categorical data is a collection of information that is divided into groups, it can only take a limited number of values.
- 2) Categorical data is a type of data that can be stored into groups or categories with the aid of names or labels.

You will get an error if you try to plug these variables into most machine learning models in Python without preprocessing them first.

We will need to use methods to assign each unique value to a different integer among.

Here one method called `ordinal_encoder` is being used.

Student Id	Degree
1	Bachelor
2	PhD
3	Master
4	Bachelor
5	No degree



Student Id	Degree
1	1
2	3
3	2
4	1
5	0

General method of Machine learning: PRACTICE GUIDE

1) Data analysis

Data analysis is where you are going to use descriptive statistics, visualization, correlation assessment and structure description to have a better understanding of your dataset, typically :

First look a the DataSet

- **DataFrame.pd.head()** shows you first rows of the DataFrame
- **DataFrame.pd.shape()** representation of total rows and columns present

Descriptive statistics

- **DataFrame.pd.describe().transpose()** return a descriptive table of the data set

	count	mean	std	min	25%	50%	75%	max
Sepal Length (in cm)	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
Sepal Width in (cm)	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
Petal length (in cm)	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9

Correlation between features in a DataSet

- **DataFrame.pd.corr()** calculates the correlation between features pairwise excluding null values. Very useful to delete some features.

	Hours spent studying	Exam score	IQ score	Hours spent sleeping
Hours spent studying	1.00	0.82	0.48	-0.22
Exam score	0.82	1.00	0.33	-0.04
IQ score	0.08	0.33	1.00	0.06
Hours spent sleeping	-0.22	-0.04	0.06	1.00

Checking about Data Types and more infos

- **DataFrame.pd.info()** returns the data type of each column present in the dataset. Also, it tells you about null and not null values present

#	Column	Non-Null Count	Dtype
0	Pregnancies	768 non-null	int64
1	Glucose	768 non-null	int64
2	BloodPressure	768 non-null	int64
3	SkinThickness	768 non-null	int64
4	Insulin	768 non-null	int64
5	BMI	768 non-null	float64

Checking about missing values

- **DataFrame.isnull()**. returns the boolean values that are true and false.
- **DataFrame.isnull().sum()** Total number of missing values in each column.

Scaling & Normalization of Data

- Scaling and Normalizing are very important step that I will develop on their own.

Scaling and Normalization of Data - NEED TO RESEARCH BETTER

Links: [Feature scaling GFG](#), [Perform feature scaling](#)

Feature Scaling or Standardization: It is a step of Data Pre-Processing which is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

Why and where to apply feature scaling :

Real world dataset contains features that highly vary in magnitudes, units, and range :

- Normalization should be performed when the scale of a feature is irrelevant or misleading.
- It should not be applied when the scale is meaningful.

The algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Here feature scaling helps to weigh all the features equally.

Formally, if a feature in the dataset is big in scale compared to others, then in algorithms where Euclidean distance is measured this big scaled feature becomes dominating and needs to be normalized.

Examples of Algorithms where Feature Scaling matters:

1. **K-Means** uses the Euclidean distance measure here feature scaling matters.
2. **K-Nearest-Neighbours** also require feature scaling.
3. **Principal Component Analysis (PCA):** Tries to get the feature with maximum variance, here to feature scaling is required.
4. **Gradient Descent (XGBoost):** Calculation speed increase as Theta calculation becomes faster after feature scaling.

Scale : X_{scaled} has now unit variance and zero mean

- $X_{scaled} = \text{preprocessing.scale}(X_{train})$

Scaling with the presence of outliers

- If the data has outliers in it then scaling that sort of data **using mean and variance is not a good approach**. You can use `robust_scale` & `Robust_Scaler` as drop-in substitution.

Normalization of DataSets

- $X_{normalized} = \text{preprocessing.normalize}()$

2) Import Data, split your Data & Define your features (Train/Valid)

This is the step where you are going to import the data, split your training set and define your predicting features.

In this step, it is crucial to have a good split between train and validation, and to **avoid train test contamination**.

(Inplace parameter)

Read the data

- `X = pd.read_csv('./input/train.csv', index_col='Id')` read a CSV file and store it in X

Remove rows with missing target, separate target from predictors

- `X.dropna(axis=0, subset=['target'], inplace=True)` Axis = 0 Lines, inplace = True means that we overwrite the existing DataFrame
- `y = X.target` Here we select our targets to predict
- `X.drop(['target'], axis=1, inplace=True)` We drop the targets from our DataFrame

Select features and Separate data into training and validation sets

- `cols_to_use=['Rooms','Distance','Landsize']` Features selection if necessary
- `X=data[cols_to_use]` Redefine our DataFrame with only the above features
- Here we use `train_test_split` in order to create Train and Validation DataFrames :
`X_train,X_valid,y_train,y_valid=train_test_split(X,y)` Important step !!!

Split your X train into : Low cardinality categorical data and Numerical Data

- We split our DataFrame into Categorical and Numerical to preprocess them separately. Remember that Categorical data need to be encoded to numerical before being used !!
- `low_cardinality_cols = [c for c in X_train_full.columns if X_train_full[c].nunique() < 10 and X_train_full[c].dtype == "object"]`
- `numeric_cols = [c for c in X_train_full.columns if X_train_full[c].dtype in ['int64', 'float64']]`

Reform your X_train, X_valid, X_test with your selected features above

- `my_cols = categorical_cols + numerical_cols`
- `X_train = X_train_full[my_cols].copy()`
- `X_valid = X_valid_full[my_cols].copy()`
- `X_test = X_test_full[my_cols].copy()`

3) Address missing values, encoding categorical Datas WITH PIPELINES :

Pipelines & Optimization (Need more research) :

Typically, when running machine learning algorithms, it involves a **sequence of tasks including, pre-processing, feature extraction, model fitting, and validation stages** :

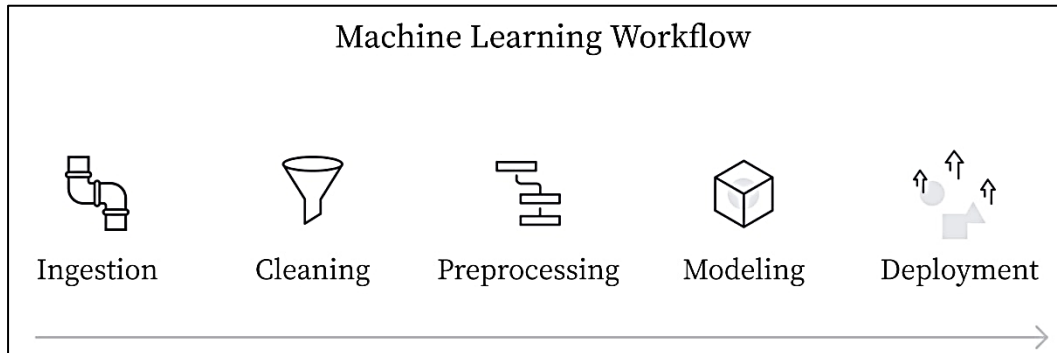


Figure 1: Workflow

What is a ML pipeline ?

A machine learning pipeline is used to **help automate ML workflows.**

They operate by enabling a sequence of data to be transformed and correlated together in a model than can be tested and evaluated to achieve an outcome.

ML pipelines consist of several steps to train a model. They are iterative as every step is repeated to continuously improve accuracy of the model and achieve successful algorithm.

Another type of ML pipeline is the art of splitting up your machine learning workflows into independent, reusable, modular parts that can then be pipelined together to create models. This type of ML pipeline makes building models more efficient and simplified, cutting out redundant work.

Resources : [ML Pipelines Mastery](#) , [ML Pipelines Valohai](#), [Kaggle pipelines](#)

Define your preprocessing strategy with pipeline :

- **Preprocessing for numerical data** : You can use different imputing strategy for missing values such as SimpleImputer from Sklearn.
- **Preprocessing for categorical data** : For categorical data you have to fill in missing values and then transform those categorical values in numerical values.

Here we can see the pipeline structure:

Pipeline (Steps = [('Name', Object)])

With the name of your model, and the type of preprocessor you want to use.

```
# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy='constant')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Use ColumnTransformer to selectively apply data preprocessing :

- ColumnTransformer is a class in the scikit-learn Python machine learning library that allows you to **selectively apply data preparation transforms**.
- It allows you to apply a specific transform or sequence of transforms to just the numerical columns, and a separate sequence of transforms to just the categorical columns.

ColumnTransformer structure:

ColumnTransformer (transformers = [(Name, Object, Columns)])

```
# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(|
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

Define your predictive model :

- Here we want to define the predictive model we are going to use in our algorithm there is so many model like RandomForest, Gradient_descent (XG_Boost)
- Then we want to bundle the preprocessing and the model in a pipeline :

CLF pipeline type :

Here our CLF pipeline can be fit directly to a data set because :

- Preprocessor has ColumnTransfo
- Model need to be fit to DataSet

```
# Define model
model = RandomForestRegressor(n_estimators=100, random_state=0)

# Bundle preprocessing and modeling code in a pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
    ('model', model)
])
```

Fit the pipeline :

- It's time to apply your executable pipeline (CLF here) to your training set :

```
# Preprocessing of training data, fit model
clf.fit(X_train, y_train)
```

4) Evaluate and optimize your model with your validation DataSet :