*Every Machine Learning project is unique in its own ways. Although, for each such project, a predefined set of steps can be followed. There is no such strict flow to be followed, but a general template can be proposed.*

**1. Prepare the problem**

The first step in not just an ML but any project is to simply define the problem at hand. You first need to understand the situation and the problem which needs to be solved. And then devise how Machine Learning would solve that efficiently. Once you know the problem well, you then head on to solve it.

**Load libraries**

I will be sticking to Python *(for obvious reasons)*in this article. The very first step is to load or [import](https://docs.python.org/2.0/ref/import.html) the all the libraries and the packages required to get the results you want. Some very primary and almost necessary packages for Machine Learning are — [NumPy](https://numpy.org/doc/), [Pandas](https://pandas.pydata.org/docs/), [Matplotlib](https://matplotlib.org/contents.html)and [Scikit-Learn](https://scikit-learn.org/stable/).

**Load dataset**

Once the libraries are loaded, you need to get the data loaded. Pandas has a very straightforward function to perform this task — [pandas.read\_csv](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html" \t "_blank). The read.csv function is not just limited to csv files, but also can read other text based files as well. Other formats can also be read using pandas read functions like html, json, pickled files etc. One thing which needs to be kept in mind is that your data needs to be in the same working directory as your current working directory or you will need to provide the complete path prefixed with a ‘/’ within the function.

**2. Summarize Data**

Okay, so the data is loaded and ready to be actioned upon. But you first need to check how the data looks and what all does it contain. To begin with, you would want to see how many rows and columns does the data have and what all are the data types of each column (which pandas thinks they are).

A quick way to take a look type and shape of your data is — [pandas.DataFrame.info](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html). This tells you how many rows and columns your dataframe has and what data types and values do they contain.

**Descriptive statistics**

Descriptive statistics, as the name suggests, describes the data in terms of its statistics — mean,standard deviation, [quantiles](https://stats.stackexchange.com/questions/156778/percentile-vs-quantile-vs-quartile)etc. The easiest way to get a complete description is by [pandas.DataFrame.describe](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html" \t "_blank). You can easily make out if your data needs to scaled or missing values need to be added, etc. (more on this later).

**Data visualizations**

Data Visualizations are very important as they are the quickest way to know the data and the patterns — if they even exist or not. Your data may have thousands of features and even more instances. It is not possible to analyze the numeric data for all of them. And if you do that, then what point is to have such powerful visualization packages like **Matplotlib**and **Seaborn**?

Visualizations using Matplotlib, Seaborn can be used to check the [correlations](https://towardsdatascience.com/let-us-understand-the-correlation-matrix-and-covariance-matrix-d42e6b643c22)within the features and with the target, scatter plots of data, histograms and [boxplots](https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51)for checking the spread and [**skewness**](https://www.youtube.com/watch?v=XSSRrVMOqlQ)and much more. Even pandas has its own built in [visualization](https://towardsdatascience.com/a-guide-to-pandas-and-matplotlib-for-data-exploration-56fad95f951c) library — pandas.DataFrame.plot which has bar plot, scatter plot, histograms etc.

**Seaborn**is essentially a transformed matplotlib as it is built on matplotlib itself and makes the plots more beautiful and the process of plotting much quicker. [Heatmap](https://seaborn.pydata.org/generated/seaborn.heatmap.html) and [pairplot](https://seaborn.pydata.org/generated/seaborn.pairplot.html" \t "_blank) are examples of power of Seaborn to quickly plot the visualization of the whole data to check [multicollinearity](https://towardsdatascience.com/multicollinearity-why-is-it-a-problem-398b010b77ac), missing values etc.

One very efficient way to get most of the above descriptive and inferential statistics of the data is through [**Pandas Profiling**](https://pandas-profiling.github.io/pandas-profiling/docs/). Profiling generates a beautiful report of the data with all the details mentioned above to let you analyze it all in one.

**3. Prepare Data**

Once you know what your data has and looks like you will have to transform it in order to make it suitable for algorithms to process and work more efficiently in order to give more accurate and precise results. This is essentially Data Pre-Processing which is the most important and the most time consuming stage of any ML project.

**Data Cleaning**

Real life data is not arranged and presented to you nicely and in a dataframe with no abnormalities. Data usually has a lot of so called abnormalities like missing values, a lot of features with incorrect format, features on different scales etc. All this needs to be handled manually which takes a lot of time and coding skills (mostly python & pandas :D )!

Pandas has various functions to check for such abnormalities like [pandas.DataFrame.isna](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isna.html" \t "_blank) to check for values with NaNs etc. You might as well need to transform the data format in order to get rid of useless information like removing ‘Mr.’ and ‘Mrs.’ from names when a separate feature for gender is present. You might need to get it in a standard format throughout the dataframe with the function [pandas.DataFrame.replace](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html" \t "_blank) or drop irrelevant features using [pandas.DataFrame.drop](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html" \t "_blank).

**Feature Selection**

Feature selection is the process of selecting a certain number of most useful features which will be used to train the model. This is done in order to reduce the dimensionality when most of the features are not contributing enough to the overall variance. If there are 300 features in your data and 97% of variance is explained by top 120 features, then it makes no sense to pound your algorithm with so many useless features. Reducing features not only saves time but costs as well.

Some of the popular feature selection techniques are [SelectKBest](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html" \t "_blank), Feature elimation methods like [RFE](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html)(recursive feature elimination) and embedded methods like [LassoCV](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html" \t "_blank).

**Feature Engineering**

All the features might not be in their best form. Meaning — they can be transformed onto a different scale by using a set of functions. This is in order to increase the correlation with the target and hence the accuracy/score. Some of these transformations are related to [**scaling**](https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02), like StandardScaler, Normalizer, MinMaxScaler etc. Features may be even added by making linear/quadratic combinations of a few features to increase performance. Log transformations, Interactions and Box-Cox transformations are some of the other useful transformations for numerical data.

For categorical data, it becomes necessary to [encode](https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/) the categories into numbers so that the algorithm can make sense out of it. Some of the most useful encoding techniques are — LabelEncoder, OneHotEncoder and Binarizer.

**4. Evaluate Algorithms**

Once your data is ready, proceed to check the performance of the various regression/classification algorithms (based on the type of problem). You can first make a base model to set a benchmark to compare against.

**Split-out validation dataset**

Once the model is trained, it needs to be validated as well to see if it really generalized the data or it over/under fitted. The data in hand can be split up beforehand as training set and validation set. This split-out has various techniques — Train Test Split, Shuffle split etc. You can also run [Cross Validation](https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6) on the entire data set for a more robust validation. KFold Cross Validation, Leave-One-Out-CV are the most popular methods.

**Test options and evaluation metric**

The models need to evaluated based on a certain set of evaluation [**metrics**](https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234)which need to defined. For regression algorithms, some of the common metrics are — MSE and R Square.

Evaluation metrics pertaining to classification are a lot more diverse — Confusion Matrix, F1 Score, AUC/ROC curves etc. These scores are compared for each algorithm to check which ones performed better than the rest.

**Spot Check Algorithms**

Once the data is split and the evaluation metrics are defined, you need to run a set of algorithm, say, in a for-loop to check which one performed the best. It is a trial and error to discover a short list of algorithms that do well on your problem so that you can then double down on and tune them further.

A [**pipeline**](https://towardsdatascience.com/pre-process-data-with-pipeline-to-prevent-data-leakage-during-cross-validation-e3442cca7fdc) can be made and a mixture of linear and non-linear algorithms can be set to check the performances.

**Compare Algorithms**

Once you have spot run the *test harness*, you can easily see which ones performed the best for your data. The algorithms giving consistently high scores should be your target. You can then take the top ones and tune them further to improve their performance.

**5. Improve Accuracy**

After you have the best performing algorithms with you, their parameters and the Hyperparameters can be tuned to give maximum results. Multiple algorithms can be chained as well.

**Algorithm Tuning**

[Wikipedia](https://en.wikipedia.org/wiki/Hyperparameter_optimization) states that “hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm”. Hyperparameters are parameters which are not learnt and have to set before running the algorithm. Some examples of hyperparameters include penalty in logistic regression, loss in stochastic gradient descent and kernel for SVM.

These parameters can be passed in an array and the algorithms can be run recursively until the perfect hyperparameters are found. This can be achieved by methods like [Grid Search](https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e) and Random Search.

**Ensembles**

Multiple Machine Learning algorithms can be combined to make a more robust and optimal model that gives better predictions than the single algorithm. This is known as an [ensemble](https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f).

There are basically 2 types of ensembles — [Bagging (Bootstrap-Aggregating) and Boosting](https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205). Random Forest, for example, is a type of Bagging ensemble which combines multiple Decision Trees and takes the aggregate of the total output.

Boosting, on the other hand, combines a set of weak learners by learning them in an an adaptive way: each model in the ensemble is fitted giving more importance to instances in the dataset that had big errors by the previous models in the sequence. XGBoost, AdaBoost, CatBoost are some examples.

**6. Finalize Model**

**Predictions on validation dataset**

When you have got an optimum performing model with best hyperparameters and ensembles, you can validate it on the unseen test dataset.

**Create standalone model on entire training dataset**

Once validated, run the model on the entire dataset once to make sure no data points are missed while training/testing. Now, your model is at its optimal position.

**Save the model for later use**

Once you have the accurate model with you, you still need to save and load it to make it available in future when it is needed. The most common method to get this done is [Pickle](https://www.geeksforgeeks.org/understanding-python-pickling-example/).