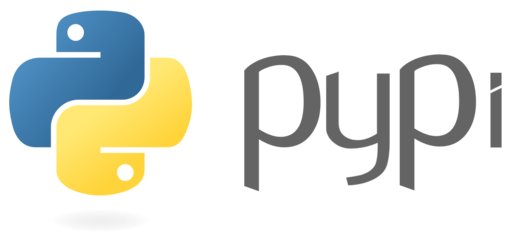
**Practical – Regularization, Ensemble and KNN – And Wine tasting!**

In this practical, you will learn about **Regularizations** and **Ensembles** which enables to get predictions from more than one model.  
You will perform **Hyper-Parameters Search** and use the models **KNN** and **LWLR**.  
You’ll practice using code from other developers from **[PyPi](https://pypi.org/)** or [**GitHub**](https://github.com/)  in your work. And lastly, use **[AutoViz](https://github.com/AutoViML/AutoViz)** to generate fast DataFrame graph reports.

**PyPi**



The Python Package Index [(PyPI)](https://pypi.org/) is a repository of software for the Python programming language.  
PyPI helps you find and install software developed and shared by the Python community.  
Package authors use PyPI to distribute their software.

**GitHub**



[GitHub](https://github.com/) is a code hosting platform for version control and collaboration.  
It lets you and others work together on projects from anywhere.  
It is also a common way to share the code you wrote with other developers.  
Sometimes, the regular packages we are using are not enough, and we want to use things that are not officially implemented yet.  
We can write them down ourselves, or search for implementations written by other developers.  
These implementations will mostly be hosted at GitHub.  
If we want to use them and their developers did not upload them to PyPi (to be downloaded easily with pip), we need to download them directly from their repositories in GitHub.  
We will use git clone to clone repositories into our machine.

**Downloads, Imports, and Definitions**

Update packages plotly and autoviz that may be too old in their Colab version.

In [1]:

!pip install --upgrade plotly

!pip install autoviz

And import our usual packages.

In [2]:

*# import numpy, matplotlib, etc.*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sns**

**import** **plotly.express** **as** **px**

**import** **matplotlib.pyplot** **as** **plt**

**import** **plotly.graph\_objects** **as** **go**

*# sklearn imports*

**from** **sklearn** **import** metrics

**from** **sklearn** **import** pipeline

**from** **sklearn** **import** linear\_model

**from** **sklearn** **import** preprocessing

**from** **sklearn** **import** neural\_network

**from** **sklearn** **import** model\_selection

**from** **sklearn.pipeline** **import** Pipeline

**from** **sklearn.model\_selection** **import** KFold

**from** **sklearn.pipeline** **import** make\_pipeline

**from** **sklearn.model\_selection** **import** LeavePOut

**from** **sklearn.linear\_model** **import** SGDRegressor

**from** **sklearn.linear\_model** **import** SGDClassifier

**from** **sklearn.metrics** **import** mean\_squared\_error

**from** **sklearn.preprocessing** **import** OneHotEncoder

**from** **sklearn.preprocessing** **import** OrdinalEncoder

**from** **sklearn.preprocessing** **import** StandardScaler

**from** **sklearn.linear\_model** **import** LinearRegression

**from** **sklearn.preprocessing** **import** PolynomialFeatures

**from** **sklearn.model\_selection** **import** train\_test\_split

**Data Exploration**

In this practical you’ll use the Vinho Verde's [White Wines](https://www.kaggle.com/danielpanizzo/wine-quality) dataset.



**Explanation**

**Source**:  
*P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.*

The study of Coerez et al. created two datasets of red and white wine samples. These datasets can be used for both classification and regression tasks.  
The inputs (features) include objective tests (e.g. PH values) and the output is based on sensory data of wine experts.  
Each expert evaluated the wine quality on a scale between 0 (very bad) and 10 (excellent). The grade for each wine was a median of at least 3 wine experts’ evaluations).

As usual, the evaluations were “blind” - the experts did not have data about grape types, wine brand, wine selling price, etc.).

The dataset here is the white wines set – of the Portuguese "Vinho Verde" wine.

The classes are ordered and not balanced (e.g. there are many more “normal” wines than “excellent” or “bad” ones).

**white wine samples** : 4898.

**Number of Attributes** : 11 + output attribute

**Attribute information**

**Input variables** (based on physicochemical tests):

1. **fixed acidity** (tartaric acid - g / dm^3) - most acids involved with wine or fixed or nonvolatile (do not evaporate readily).
2. **volatile acidity** (acetic acid - g / dm^3) - the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste.
3. **citric acid** (g / dm^3) - found in small quantities, citric acid can add 'freshness' and flavor to wines.
4. **residual sugar** (g / dm^3) - the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter, and wines with greater than 45 grams/liter are considered sweet.
5. **chlorides** (sodium chloride - g / dm^3) - the amount of salt in the wine.
6. **free sulfur dioxide** (mg / dm^3) - the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of the wine.
7. **total sulfur dioxide** (mg / dm^3) - the amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine.
8. **density** (g / cm^3) - the density of water is close to that of water depending on the percent alcohol and sugar content.
9. **pH** - describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale.
10. **sulphates** (potassium sulphate - g / dm3) - a wine additive that can contribute to sulfur dioxide gas (S02) levels, which acts as an antimicrobial and antioxidant.
11. **alcohol** (% by volume) - the percent alcohol content of the wine.

**Output variable** (based on sensory data):

1. **quality** (score between 0 and 10)



Download the dataset from Github and explore it with Pandas tools.

Download the data

In [3]:

*# download whitewines.csv file from Github*

!wget https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/master/whitewines.csv

And print its information and description.

Now use autoviz to show a report on the data.  
This report consists of useful graphs that can display the data for your observation.

In [7]:

*# import autoviz and show report on usedcars\_df*

**from** **autoviz.AutoViz\_Class** **import** AutoViz\_Class

AV = AutoViz\_Class()

dft = AV.AutoViz("", depVar='quality', dfte=whitewines\_df, verbose=1)

Note: verbose=0 or 1 generates charts and displays them in your local Jupyter notebook.

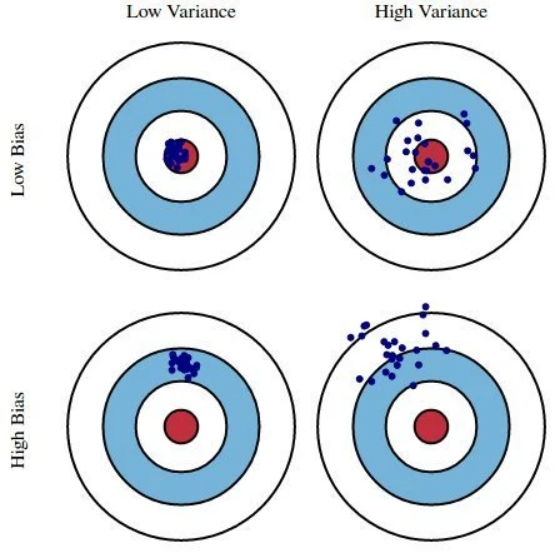
verbose=2 saves plots in your local machine under AutoViz\_Plots directory and does not display charts.

As in previous practicals, divide the data to features and target

And analyze the target classes – which are the different experts evaluation scores: how many different evaluation scores are there in the dataset? – use unique.

**Regularization**

Recall the bias and variance problems? Here’s a visual to describe them: In the figure below the target is of course the mid red circle. The blue dots are the estimations



Regularization techniques are used to reduce high-Variance. They control the variance by adding a term to the loss function thus controlling the magnitude of the model weights. This term is called a *penalty* to high weights.

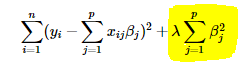
There are three regularization techniques:

1. L1 (also called Lasso).
2. L2 (also called Ridge).
3. Elastic Net (a combination of Lasso and Ridge).

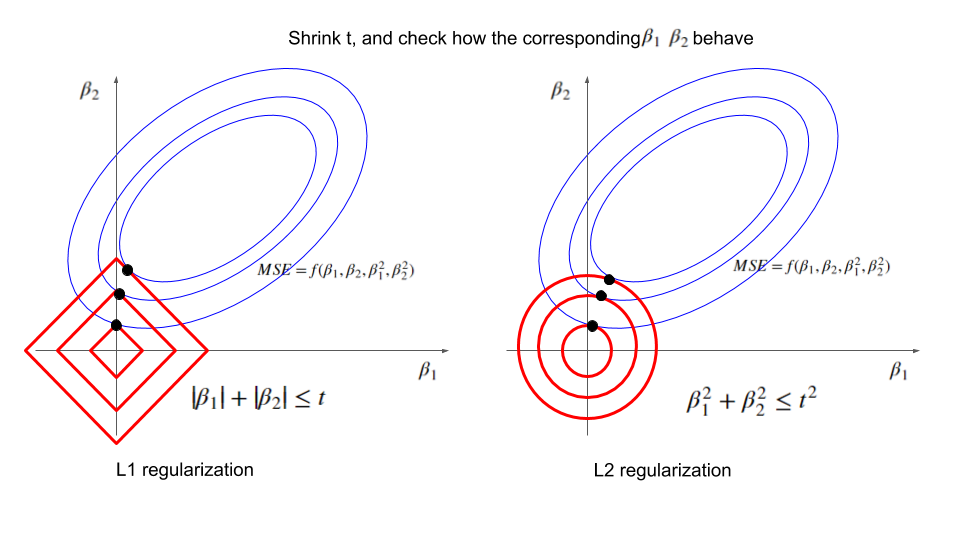
**L1 ( Lasso) regularization**

  
The L1 penalty increases or decreases in fixed-size steps.  
Unnecessary weights tend to get to zero (which makes this technique practically feature selection).

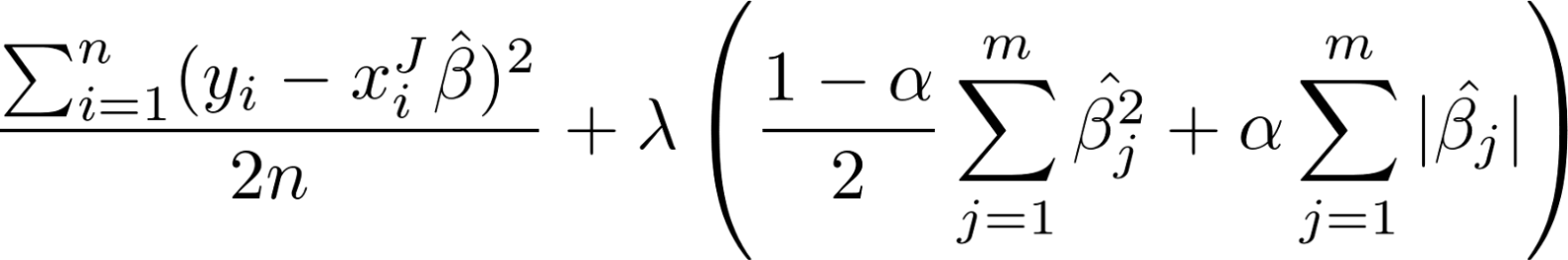
**L2 (Ridge) Regularization**

  
The L2 penalty adds big values to the loss function for **large** weights, and thus considerably reduces the weights.  
For small weights, this penalty adds small values to the loss function and thus only slightly reduces the weights.  
Unnecessary weights still stay in the model, but with insignificant values.

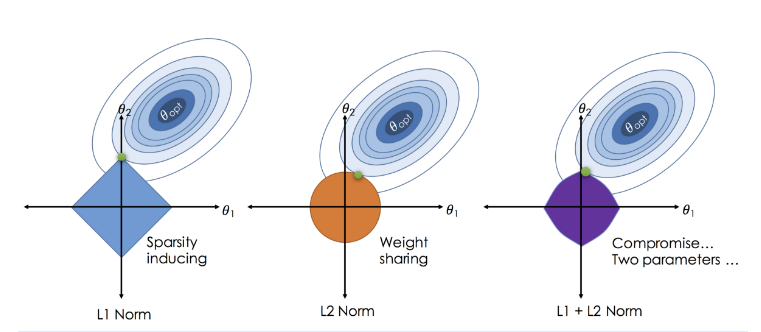
The figure below describes the difference between the two methods for 2 parameters and 2 weights: β1 and β2. The blue ellipses are the MSE loss functions for different combinations of the weights β1 and β2.  The red shapes describe the regularization penalty.



**ElasticNet**

  
This regularization is a combination of L1 (Lasso) and L2 (Ridge).

It is illustrated in the figure below.



All 3 regularizations can be used as Hyper-Parameters in Scikit-learn [SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html) and [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html).  
Or, when using the normal GD, with Scikit-learn [Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html), [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) and [ElasticNet](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html).

Start with using SGDRegressor and cross\_val\_score and print the lasso, ridge and elasticnet scores.

Now create the accuracy\_score of the regression models using Scikit-learn [make\_scorer](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html) and accuracy\_score.

Now run the classification using SGDClassifier, and check the accuracy scores for the 3 regularizations

You will see that the classifiers predicted worse than the regressors.

You can try to create [ordinal classifiers](https://towardsdatascience.com/simple-trick-to-train-an-ordinal-regression-with-any-classifier-6911183d2a3c) that will use the fact that there is an order to the labels.

**Hyper-Parameters Search**

Most of our models have lots of parameters. Each parameter can be adjusted and change the performance of the model (to better or worse) .  
The challenge is to find the best hyperparameters for our models.

Two approaches are:

1. Grid Search
2. Random Search

**Grid Search**

Checks every possible parameter and all combinations of parameters to find the best one, which gives the best score.  
This is an exhaustive approach.

**Random Search**

Chooses random combinations of parameters and check the score on each.   
This will not be as accurate as Grid Search, but take less time.

Use Scikit-learn [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html), with the hyper parameters: 'penalty': ('l2', 'l1', 'elasticnet'), 'alpha':[0.0001, 0.001, 0.01, 0.1] to perform SGDClassifier on the data and print the accuracy score and the best parameters alpha and penalty.

You should see that the best parameters on this model (obtained with Grid Search), are penalty=elasticnet and alpha=0.001.

Note that the best parameters on this model obtained with Grid Search may change with a different random\_state.

Now try Scikit-learn [RandomizedSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html).  
Use Scipy [stats.uniform](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.uniform.html) to get uniformly randomize values of alpha.  
Then use Numpy [random.seed](https://numpy.org/doc/stable/reference/random/generated/numpy.random.seed.html) to make sure that you get the same result each time you run this cell.

You should get that the best parameters on this model (obtained with Random Search) were penalty=l2 and alpha=0.417022004702574.

The Accuracy score of the Randomized Search is a little lower than the score of the Grid Search, but it may change with a different random seed.

**Ensembles**

These techniques enable the usage of a collection (ensemble) of models increase the prediction accurate and decrease the variance.

The concept is that One model may be wrong, but a lot of different models are less prone to errors.

In regression, we can take the mean of all the predictions of the models.  
In classification, we can take the mean of all the probabilities of the models or choose the class that most of the models chose for the sample.  
It is like " crowd sourcing" with a “crowd” of classifiers.

In this practical we’ll use two types of ensembles:

1. Bagging (with NFold or with Bootstrap).
2. Boosting.

**Bagging**

For this method we create several “bags of samples” from the original dataset.

A model is trained on each of the bags, and the method returns the combined score.

The bags can be created using NFold (same as KFold CV): Divide the data to N parts and use KFold cross validation (where K=N). All the N (or K) models are saved and are used as an ensemble.

The bags can be created using Bootstrap: Draw samples from the dataset (with replacement) and train the model on each group of samples. All the models and use them as an ensemble.

**Boosting**

In this method we create a model and train it on the data. Then we take the samples that the model predicted incorrectly and multiply them (thus giving them more weight in the next training).  
We repeat this procedure until we have few models, each of them is an expert on some type of samples.  
We combine all the model's predictions and return a combined score.  
There are several boosting algorithms [AdaBoost](https://www.youtube.com/watch?v=LsK-xG1cLYA&ab_channel=StatQuestwithJoshStarmer), [GradientBoost](https://www.youtube.com/watch?v=3CC4N4z3GJc&ab_channel=StatQuestwithJoshStarmer), and more.

Start with Scikit-learn [BaggingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html).  
It can be used with bootstrap=False for NFold bagging, or, with bootstrap=True for Bootstrap.  
Try both options and print the accuracy score.

Which bagging got better results?

You should get that for our model, NFold Bagging got better results but the difference is small and it may change if we use bigger n\_estimators.

Now try AdaBoosting, with AdaBoostClassifier

You should get that the bagging ensemble still performs best.

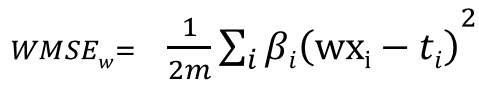
**Working with GitHub**

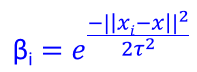
When the package we need is not included in Scikit-learn we can use use a Github package.  [packege from GitHub](https://github.com/qiaochen/CourseExercises).  
If the package is not stored in PyPi, we can not download it with pip install. In this case we can use git clone .

To practice this, as well as to practice dealing with **new** machine learning models which were not discussed in class, here is an example of a model that was not learned in class and is not included in the Scikit-learn offering.

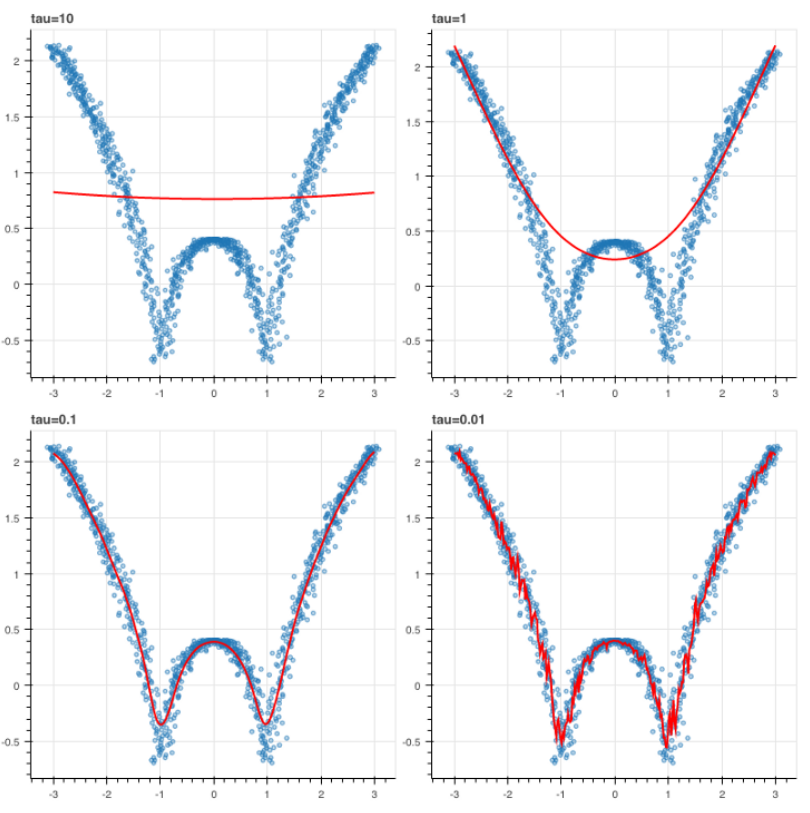
The model is *Locally Weighted Linear Regression (***LWLR)**

A short introduction to this model:

This is a special form of Linear Regression that helps in cases where the loss function (MSE) has more than one minimum.  
The MSE function in this model is updated to:  
   
The equation shows that a weight (βi) is assigned to every sample’s error (wxi-t).  
This weight emphasize the importance of the samples that are closer to the test sample.  
For every test sample, a model is trained from scratch and each training sample is given a weight that corresponds to the distance from the test sample.  
A popular weight function is the Gaussian function:

  
  
where “x” is the *mean* of all xi training samples.

For small τ values, only samples which are closer to the target are considered in the WMSE (since large distances get small weight (β)).  
For large τ values, this is the just the MSE (big distances effect is just as much as small distances).

The picture below demonstrates the effect of different τ values on the regression  
 

As Scikit-learn does not have LWLR, we need to use a [packege from GitHub](https://github.com/qiaochen/CourseExercises).  
This package is not stored in PyPi, so we can not download it with pip install, and need to use git clone .

*# clone the lwlr repo from github*

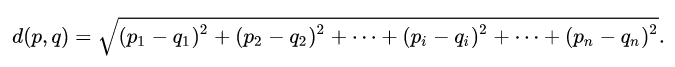
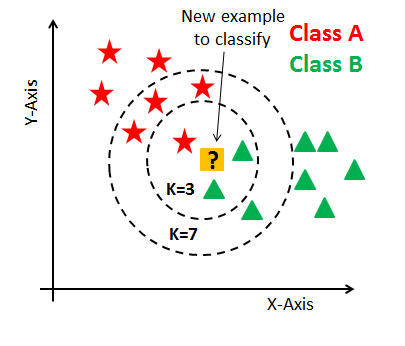
!git clone https://github.com/qiaochen/CourseExercises

Run the LWLR model with cross validation k=1 and print its R2 and accuracy cross-validation scores

Now check which is best k for this LWLR model. Run the LWLP for k values from 1 to 10.  
It may take some time, as you are building the model from scratch for every test sample, so add to the function a time to check how long it actually takes.

Print the accuracy score, the best “k” hyper parameter and the CPU time it took.

**K Nearest Neighbors (KNN)**

The concept of LWLR is taken to the extreme in *K Nearest Neighbors* (KNN) : this model predicts only based on the closest training samples to a test sample.  
  
We choose the k and the model will calculate the prediction for each test sample, based on the closest k training samples to the test sample.  
We need to determine what is the meaning of *close* or *near* - a *distance* function is needed to determine this “closeness” of each training sample to the test samples.  
There are many distance functions in Math. The simplest is the [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance):  
   
KNN can be used for both classification and regression tasks.  
For a regression task, we take the mean of each target from all the neighbors.  
For a classification task, we take the mean of the probability of all the neighbors, or we use voting, and choose the label that got the most votes from the neighbors.  
For the voting - we can either give all the neighbors in this “decision group” the same weight in the vote, or we can give the closest neighbors higher weights than the farthest.  
  
Use Scikit-learn [KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) to run KNN on the dataset and find the best number of neighbors (“K”) according to the best accuracy.

You should see that the best n\_neighbors is 16.

Now try the Scikit-learn [KNeighborsRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html).

which experiment yielded better performance – regression or classification? Can you try to give a reason to the answer?

->The KNeighborsRegressor did better than the KNeighborsClassifier on this dataset.  
It might be due to the additional information it has on the order of the classes.

**More Information**

Guide on how to upload python packeges to PyPi:  
[How to upload your python package to PyPi](https://medium.com/@joel.barmettler/how-to-upload-your-python-package-to-pypi-65edc5fe9c56)

Explanation of few EDA libraries in python:  
[4 Libraries that can perform EDA in one line of python code](https://towardsdatascience.com/4-libraries-that-can-perform-eda-in-one-line-of-python-code-b13938a06ae)

Explanation of Vinho Verde wines:  
[Portuguese Vinho Verde wine: everything you need to know](https://www.olivemagazine.com/drink/portuguese-vinho-verde-wine-guide/)

Kaggle notebook on the white wine dataset:  
[KNN for classifying wine quality](https://www.kaggle.com/raultrevino/knn-for-classify-wine-quality)

Kaggle notebook on the white wine dataset:  
[Predicting White Wine Quality](https://www.kaggle.com/indra90/predicting-white-wine-quality)

Explanation on Lowess smoothing:  
[Lowess Smoothing: Overview](https://www.statisticshowto.com/lowess-smoothing/)

Explanation on regularization:  
[REGULARIZATION: An important concept in Machine Learning](https://towardsdatascience.com/regularization-an-important-concept-in-machine-learning-5891628907ea)

Article about the geometry of Ridge and Lasso regularizations:  
[Regularization and Geometry](https://towardsdatascience.com/regularization-and-geometry-c69a2365de19)

Explanation on Ridge, Lasso, and Elastic Net regularizations:  
[An Introduction to Ridge, Lasso, and Elastic Net Regression](https://hackernoon.com/an-introduction-to-ridge-lasso-and-elastic-net-regression-cca60b4b934f)

Explanation of the differences between Ridge and Lasso regularizations:  
[Intuitive and Visual Explanation on the differences between L1 and L2 regularization](https://www.linkedin.com/pulse/intuitive-visual-explanation-differences-between-l1-l2-xiaoli-chen/)

Guide on how to use regular classifier as an ordinal classifier:  
[Simple Trick to Train an Ordinal Regression with any Classifier](https://towardsdatascience.com/simple-trick-to-train-an-ordinal-regression-with-any-classifier-6911183d2a3c)

A list of predefined scores for Sciking-learn:  
[Common cases: predefined scores](https://scikit-learn.org/stable/modules/model_evaluation.html#common-cases-predefined-values)

Explanation on LWLR and a code sample:  
[Linear Regression: How to overcome underfitting with Locally Weighted Linear Regression (LWLR)](https://itnext.io/linear-regression-how-to-overcome-underfitting-with-locally-weight-linear-regression-lwlr-e867f0cde4a4)

Example of LWLR in python:  
[Locally Weighted Linear Regression in Python](https://www.codespeedy.com/locally-weighted-linear-regression-in-python/)

Short Explanation on LWLR:  
[ML | Locally weighted Linear Regression](https://www.geeksforgeeks.org/ml-locally-weighted-linear-regression/)