Marion BARTIER, Alexandre LEDEUF, Alexandre VIGNAUD, Jean-Pierre TRAN

Convex Optimisation

<div align="center">Summary</div>

For the course of Convex Optimisation, the final goal is to carry out one of the presented projects. This project will allow us to show our understanding of the different concepts discovered in class and use those knowledge to solve concrete problems.

We choosed to work on the first project : the application of optimization techniques to a concrete machine learning problem. In this project we will have to solve a machine learning task on the dataset we choosed and present our approach and methods used to solve our problem.

In the following document, we will present the dataset and what we have done before starting the analysis, a description of the different models used, then a comparison between those models and finally the results and the response to our problem.

Before starting

The dataset

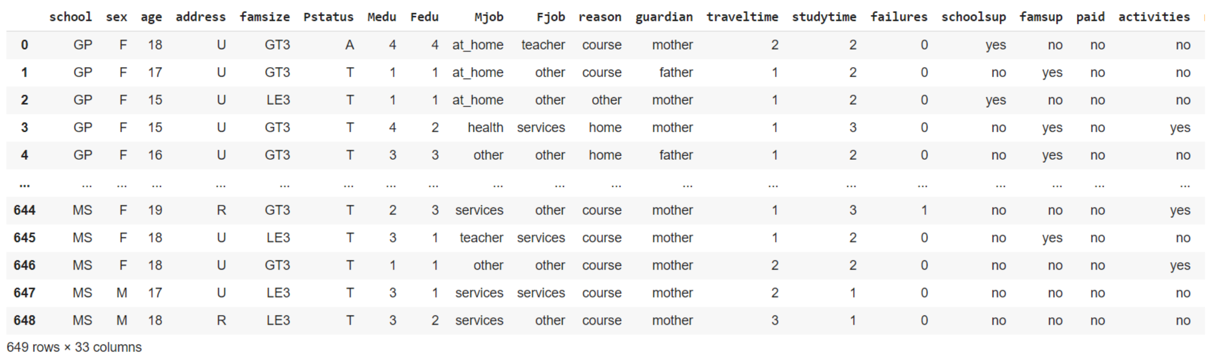
This dataset is about student performance for a secondary education in two Portuguese schools. Data are various and are related to grades, demographic, social and school features. We are going to present in detail the different features:

Variable name	Description of variable	Content
school	Student's school	GP (Gabriel Pereira) / MS (Mousinho da Silveira)
sex	Student's sex	F (female) / M (male)
age	Student's age	15 to 22
address	student's home address type	U (urban) / R (rural)
famsize	family size	LE3 (less or equal to 3) / GT3 (greater than 3)
Pstatus	parent's cohabitation status	T (living together) / A (apart)
Medu	mother's education	0 (none) / 1 (primary education (4th grade)) / 2 (5th to 9th grade) / 3 (secondary education) / 4 (higher education)
Fedu	father's education	0 (none) / 1 (primary education (4th grade)) / 2 (5th to 9th grade) / 3 (secondary education) / 4 (higher education)
Mjob	mother's job	teacher / health / (civil) services (administrative or

		police) / at_home / other
Fjob	father’s job	teacher / health / (civil) services (administrative or police) / at_home / other
reason	reason to choose this school	(close to) home / (school) reputation / course (preference) / other
guardian	student's guardian	mother / father / other
traveltime	home to school travel time	1 (<15 min) / 2 (15 to 30 min) / 3 (30 min to 1 hour) / 4 (>1 hour)
studytime	weekly study time	1 (<2 hours) / 2 (2 to 5 hours) / 3 (5 to 10 hours) / 4 (>10 hours)
failures	number of past class failures	1 to 3 if more 4
schoolsup	extra educational support	yes / no
famsup	family educational support	yes / no
paid	extra paid classes within the course subject	yes / no
activities	extra-curricular activities	yes / no
nursery	attended nursery school	yes / no
higher	wants to take higher education	yes / no
internet	Internet access at home	yes / no
romantic	with a romantic relationship	yes / no
famrel	quality of family relationships	1 (very bad) to 5 (excellent)
freetime	free time after school	1 (very low) to 5 (very high)
goout	going out with friends	1 (very low) to 5 (very high)
Dalc	workday alcohol consumption	1 (very low) to 5 (very high)
Walc	weekend alcohol consumption	1 (very low) to 5 (very high)
health	current health status	1 (very bad) to 5 (very good)
absences	number of school absences	0 to 93
G1	first period grade	0 to 20
G2	second period grade	0 to 20
G3	final grade	0 to 20

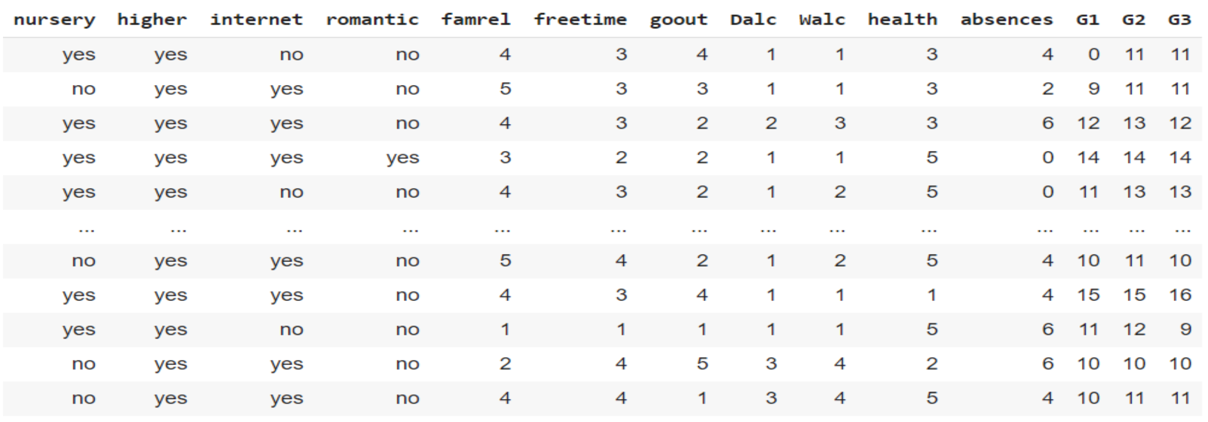
Data exploration and visualisation

We can find the dataset we will use below. It is composed of 33 Columns.



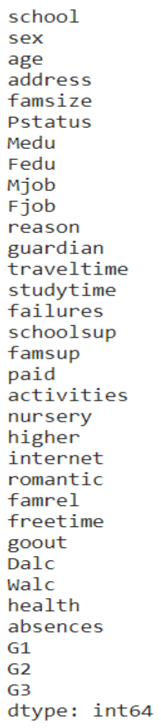
	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes	no	no	no
1	GP	F	17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no	yes	no	no
2	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother	1	2	0	yes	no	no	no
3	GP	F	15	U	GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	no	yes
4	GP	F	16	U	GT3	T	3	3	other	other	home	father	1	2	0	no	yes	no	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
644	MS	F	19	R	GT3	T	2	3	services	other	course	mother	1	3	1	no	no	no	yes
645	MS	F	18	U	LE3	T	3	1	teacher	services	course	mother	1	2	0	no	yes	no	no
646	MS	F	18	U	GT3	T	1	1	other	other	course	mother	2	2	0	no	no	no	yes
647	MS	M	17	U	LE3	T	3	1	services	services	course	mother	2	1	0	no	no	no	no
648	MS	M	18	R	LE3	T	3	2	services	other	course	mother	3	1	0	no	no	no	no

649 rows × 33 columns



nursery	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
yes	yes	no	no	4	3	4	1	1	3	4	0	11	11
no	yes	yes	no	5	3	3	1	1	3	2	9	11	11
yes	yes	yes	no	4	3	2	2	3	3	6	12	13	12
yes	yes	yes	yes	3	2	2	1	1	5	0	14	14	14
yes	yes	no	no	4	3	2	1	2	5	0	11	13	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...
no	yes	yes	no	5	4	2	1	2	5	4	10	11	10
yes	yes	yes	no	4	3	4	1	1	1	4	15	15	16
yes	yes	no	no	1	1	1	1	1	5	6	11	12	9
no	yes	yes	no	2	4	5	3	4	2	6	10	10	10
no	yes	yes	no	4	4	1	3	4	5	4	10	11	11

We need to know the quantity of missing values in our dataset.To do that, we will use a library called Missingno to see the missing values of our dataset and remove it for the data visualizations part.



```
school        0
sex           0
age           0
address       0
famsize       0
Pstatus       0
Medu          0
Fedu          0
Mjob          0
Fjob          0
reason        0
guardian      0
traveltime    0
studytime     0
failures      0
schoolsup     0
famsup        0
paid          0
activities    0
nursery       0
higher        0
internet      0
romantic      0
famrel        0
freetime      0
goout         0
Dalc          0
Walc          0
health        0
absences      0
G1            0
G2            0
G3            0
dtype: int64
```

After that we can take a look at the G3 variable, that represents the grades of the final exam. With a describe, we can display some statistic :

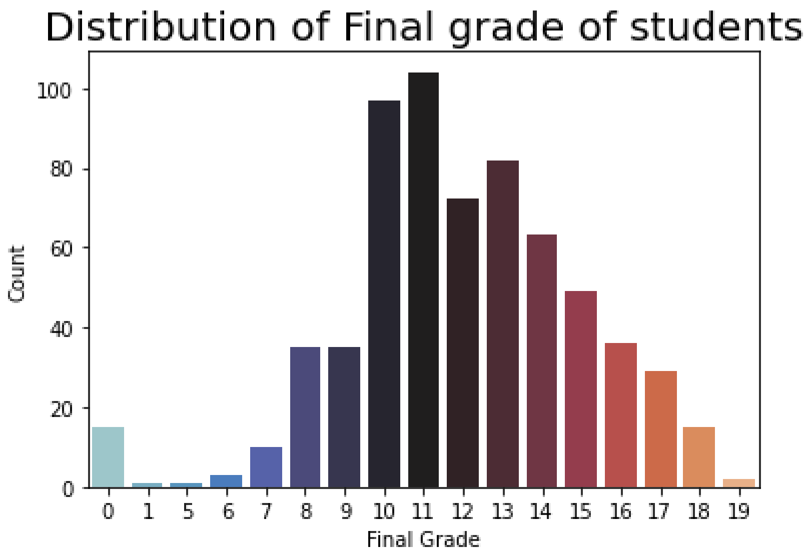
```
count    649.000000
mean      11.906009
std        3.230656
min        0.000000
25%       10.000000
50%       12.000000
75%       14.000000
max       19.000000
Name: G3, dtype: float64
```

With this command, we can observe that the grade distribution seems to be balanced. Indeed with all the values, we have a mean of 11 which is good for student scores. The first and the third quartiles

let's have a look to the final grade of student :

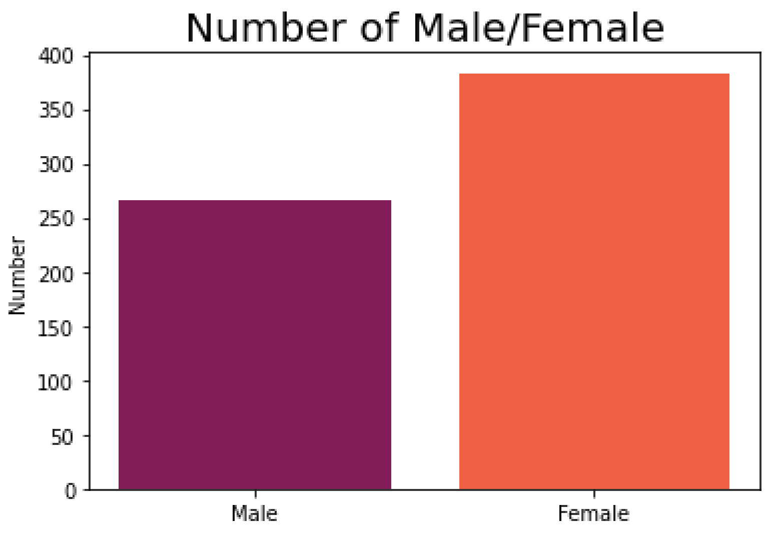
We can see that we have a lot of data and a vast majority of students with a grade between 10 and 13.

To represent it in different way, let's have a look to this :

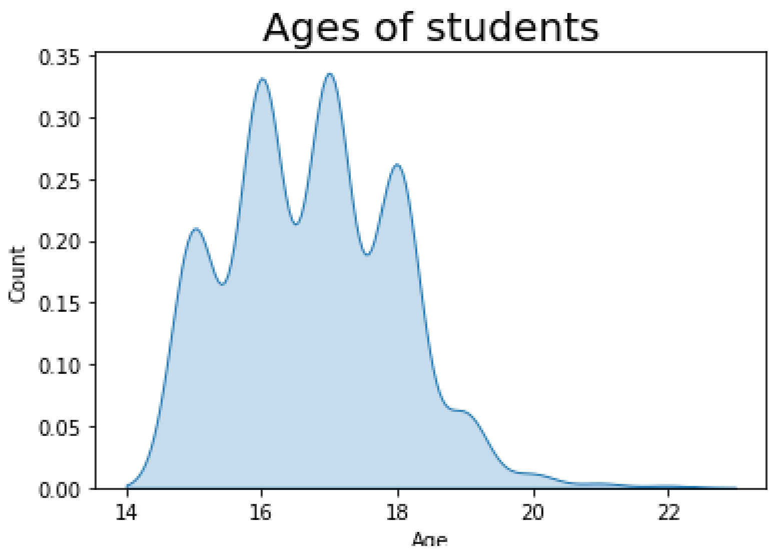


It is more relevant this time. This plot shows us that a lot 10 and 11 are the most common notes in our dataframe.
After that, we want to know if this diversity of score is related to something in particular. It can be caused by no, one or multiple variables of our dataset, so we want to verify which variable can verify this hypothesis if it exists.

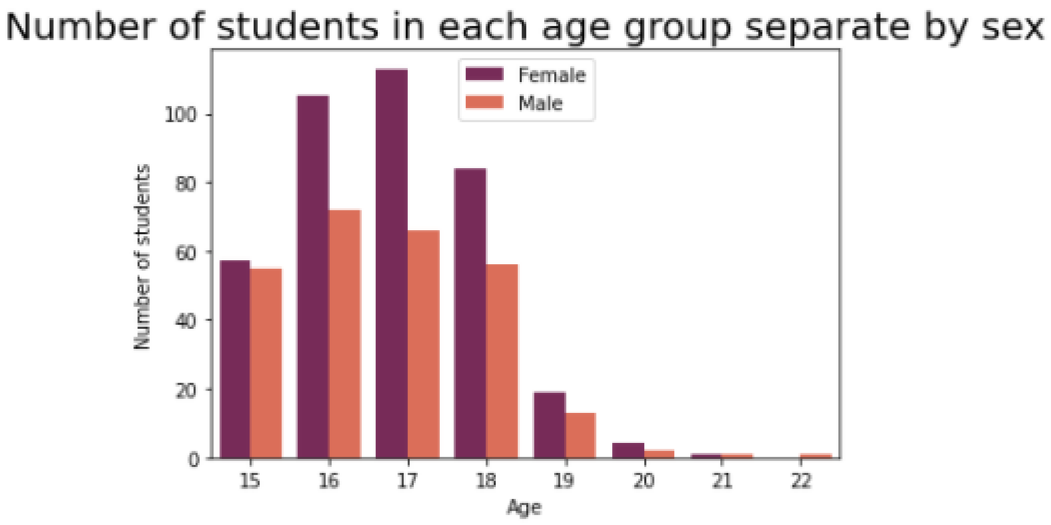
This graph shows us the quantity of male and female in this dataset. We have 266 males and 383 females.



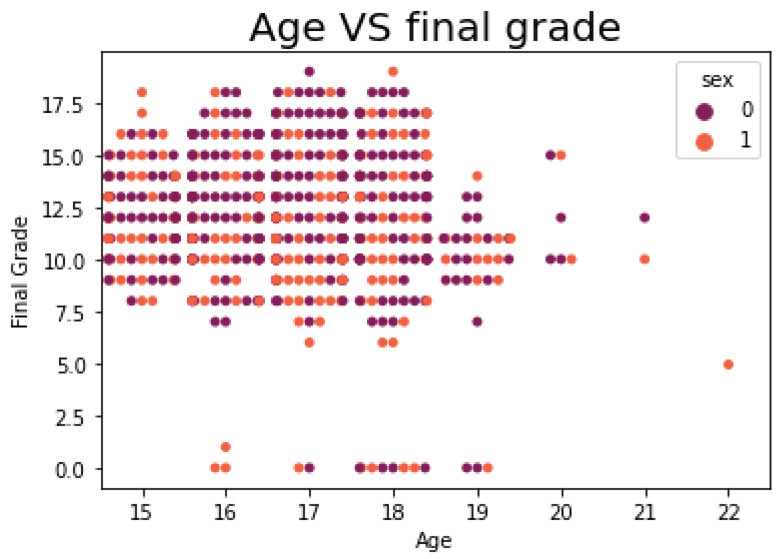
The students’ sex can possibly be a factor of the grade, but the age can also be a factor too, that is why we will verify that supposition.
On the first graph, we can see students’ age proportion.



On this one we can see the number of students group by their age and their sex.

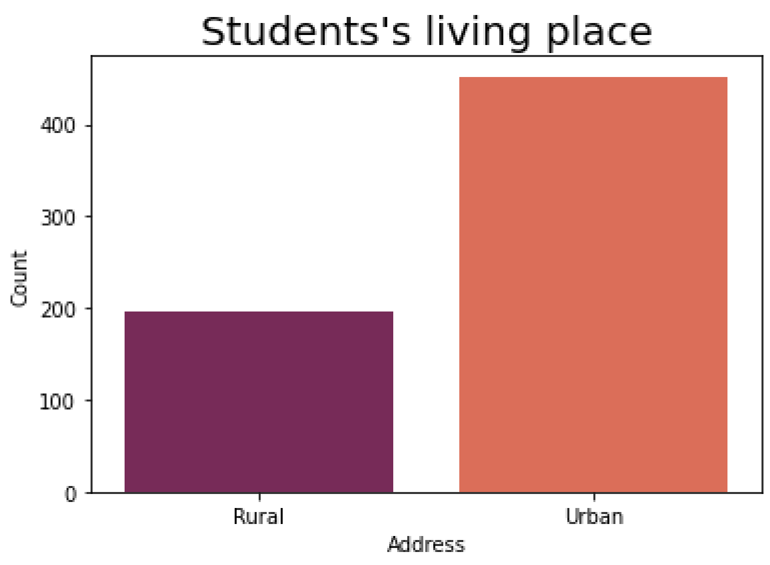


For the next plot, we decided to show the correlation between age, sex and the final grade.

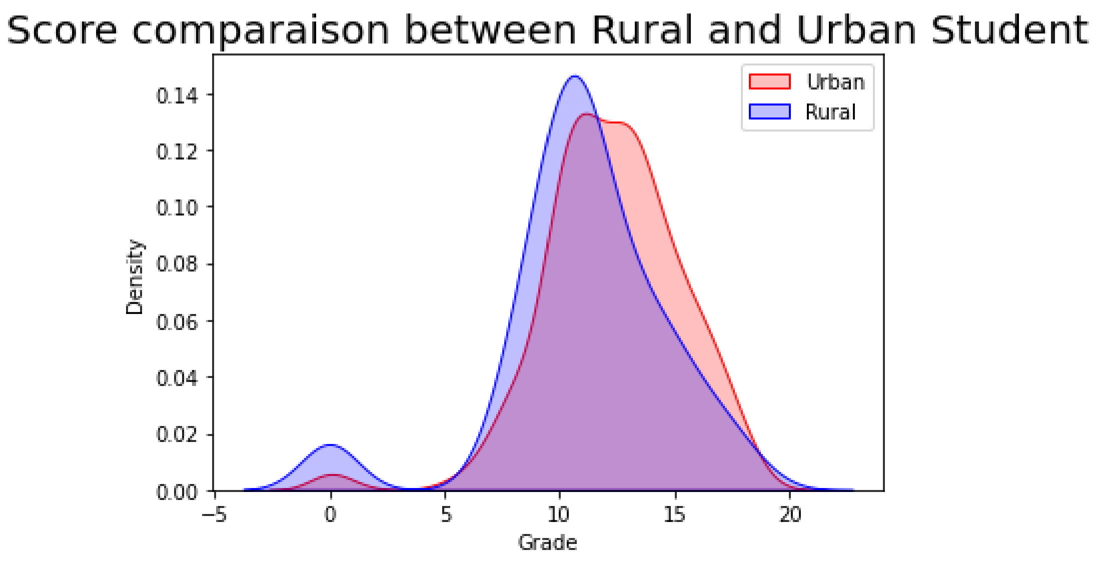


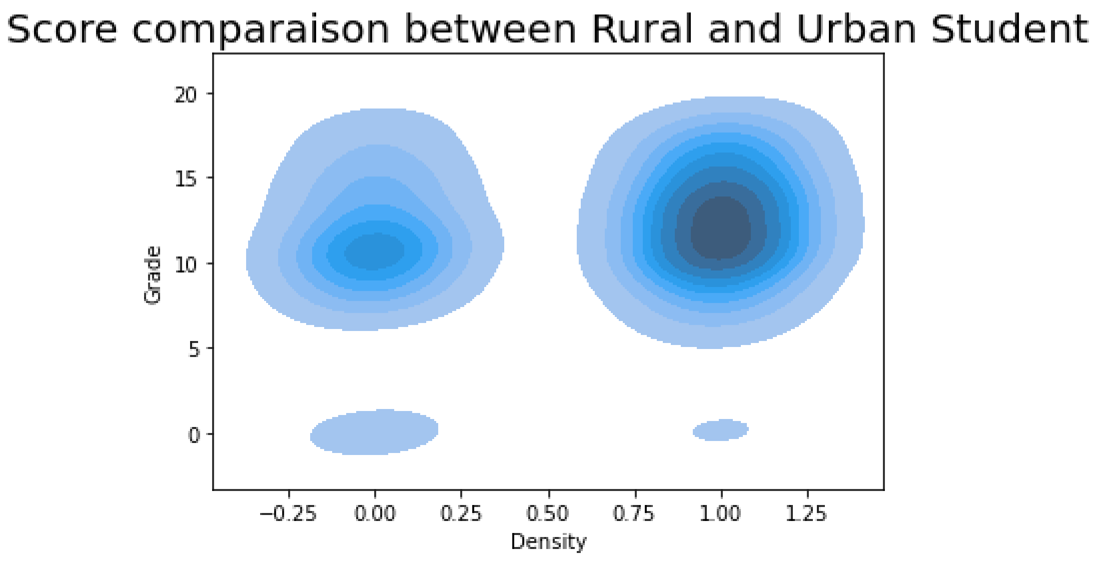
As we can see, the age and the sex does not really increase or decrease the student grade. Everyone has the same opportunity to get a good grade regardless of their age or sex.

Now we know that previous factors have no incidence on our students' grades, we can suppose that other parameters can modify their grades, like the place where students live. Our new hypothesis becomes: Do people who live in big cities have more chances to get good grades ?



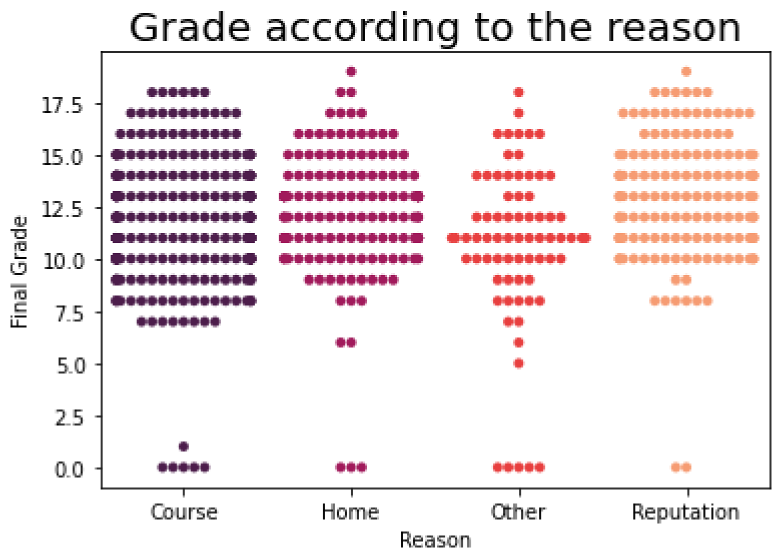
We can see that there are more students in big cities, but what about their grades? there are 2 representations :





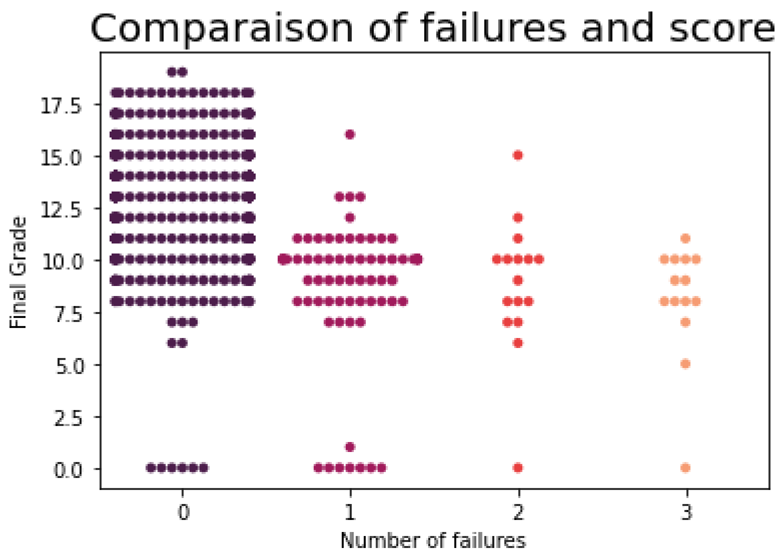
In general, we can see that it is almost the same. There are more students with good grades but it can be explained by the fact that in the dataset, the repartition is not equivalent. As the first graph showed, there are 2 times more students who live in urban areas than in rural areas.

Now, let's have a look at some reason that the student chose is school and if it modifies its grade.



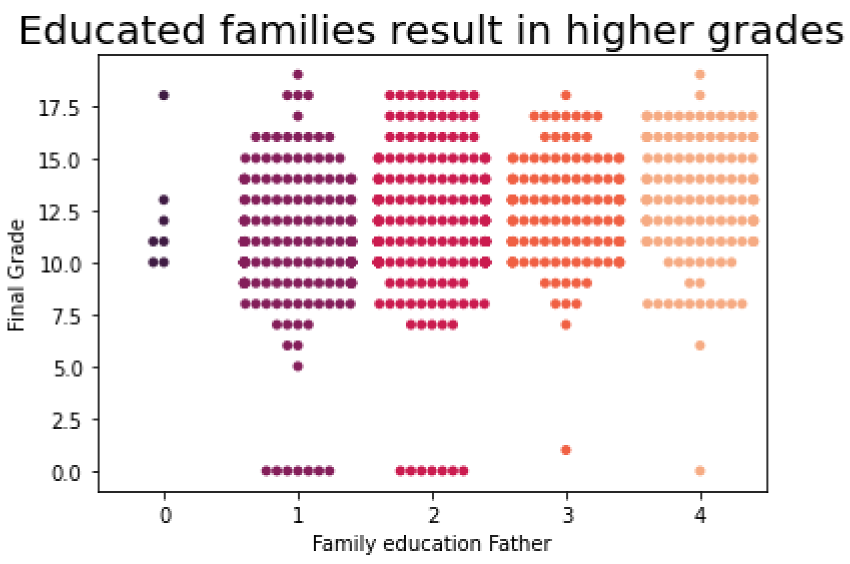
The results are quietly the same except maybe the kind of course preference which is a bit lower.

Now we take a look to the failures :



We can see that students with less failure will have better grades.

And by difference students with families that have been educated will get better scores, maybe because they are helped during their work or training.



Different models

Linear regression

Linear regression is a machine learning algorithm to predict values. The difference between linear regression and logistic regression, is that the linear regression predicts continuous values, while logistic categories discrete values. There are two types of linear regression, the simple regression and multivariable regression.

First let's check the simple regression:

$$y = \beta_0 + \beta_1 X + \varepsilon$$

Linear regression finds the best fit line through your data. y is the predicted value, x the independent variable, ß0 the intercept, ß1 is the regression coefficient and ε represents the error of the estimate. Its formula is:

$$Error = y^{(i)} - h_\theta(x^{(i)})$$

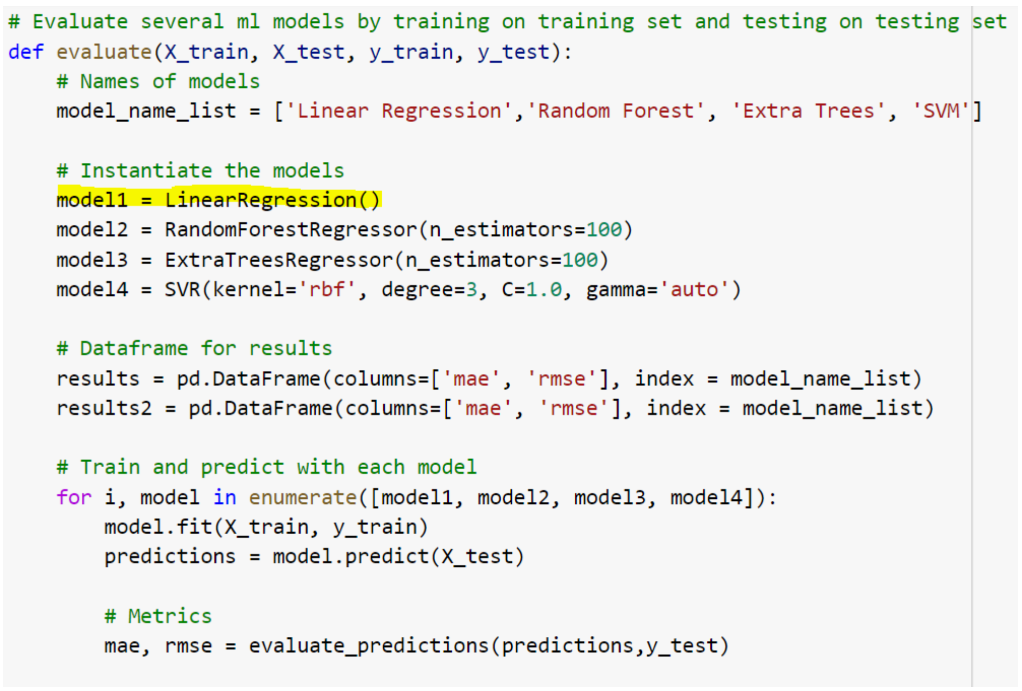
As we see in the above formula, the error is equal to the current value minus the predicted value.

Then we have the multivariable regression:

$$f(x, y, z) = w_1 x + w_2 y + w_3 z$$

We have multi-variable in the linear equation, and the w represents the coefficients, or weights.We have x, y, and z who represent the attributes.

For our implementation:



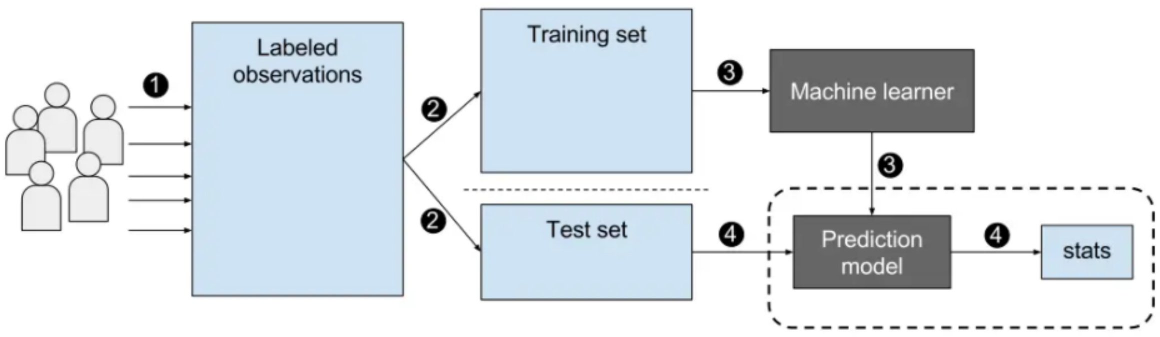
```python
# Evaluate several ml models by training on training set and testing on testing set
def evaluate(X_train, X_test, y_train, y_test):
    # Names of models
    model_name_list = ['Linear Regression','Random Forest', 'Extra Trees', 'SVM']

    # Instantiate the models
    model1 = LinearRegression()
    model2 = RandomForestRegressor(n_estimators=100)
    model3 = ExtraTreesRegressor(n_estimators=100)
    model4 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')

    # Dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)
    results2 = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        # Metrics
        mae, rmse = evaluate_predictions(predictions,y_test)
```

We have used the library sklearn and we import the LinearRegression.

SVM

Support Vector Machines are supervised learning methods used for regression and linear discriminant problems. Supervised learning uses labeled data while training an algorithm.

Supervised machine learning



The goal of SVM is to make a straight line between two classes, this straight line separates the data. All the data in one side represents a category and the other side of the line represents another category. SVM algorithm will choose the decision boundary that maximizes the distance from the nearest data points of all the classes.

We have 2 types of SVM, Simple SVM and Kernel SVM.The simple SVM are used for linear regression and classification problems, and the Kernel SVM are used for non-linear data.We will use Kernel SVM, because we can add a lot of features with it.

The most important SVR parameter is Kernel type. It can be linear, polynomial or gaussian SVR.

- Linear kernel :

$f(X) = w^\wedge T * X + b$

w means the weight vector, X is the data, b is the linear coefficient from the training data.

- Polynomial formula:

$f(X1, X2) = (a + X1^\wedge T * X2)^\wedge b$

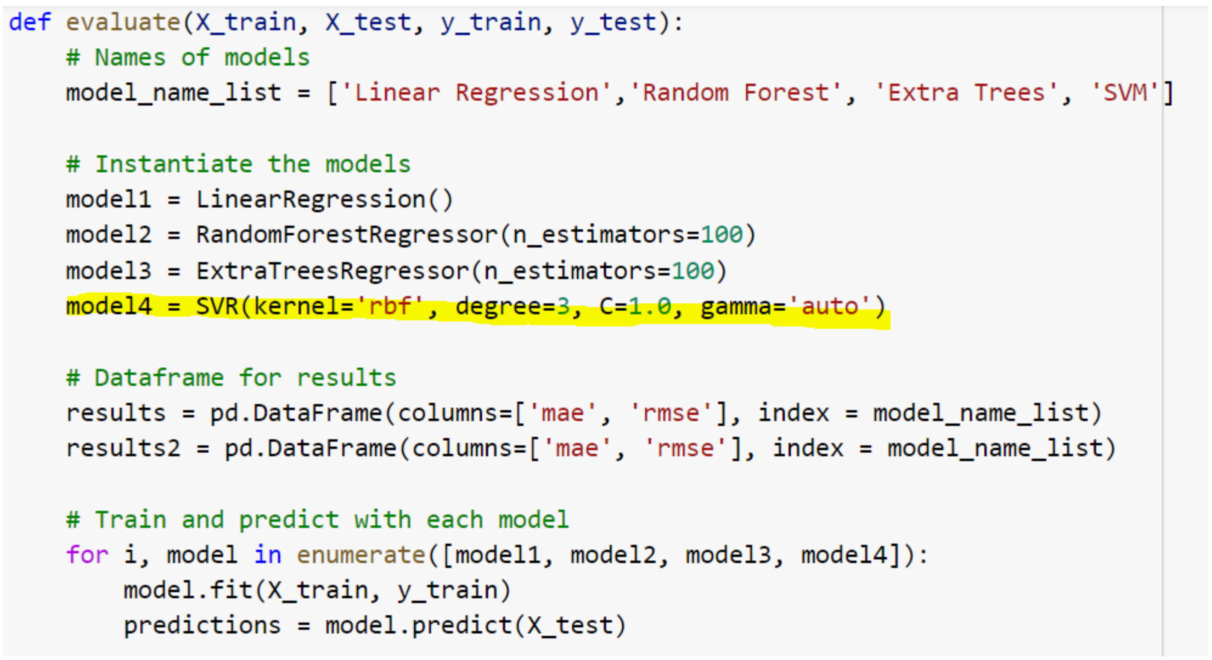
f(X1, X2) is the polynomial decision boundary that will separate the data.

- Gaussian Radial Basis Function (RBF):

$f(X1, X2) = exp(- gamma * ||X1 - X2||^\wedge 2)$

The gamma specifies how much a single training point has on the other data points around it.

We have a non-linear condition, so we will use RBF(a gaussian type) kernel. For the implementation the support vector regression (SVR) from the library sklearn.svm.



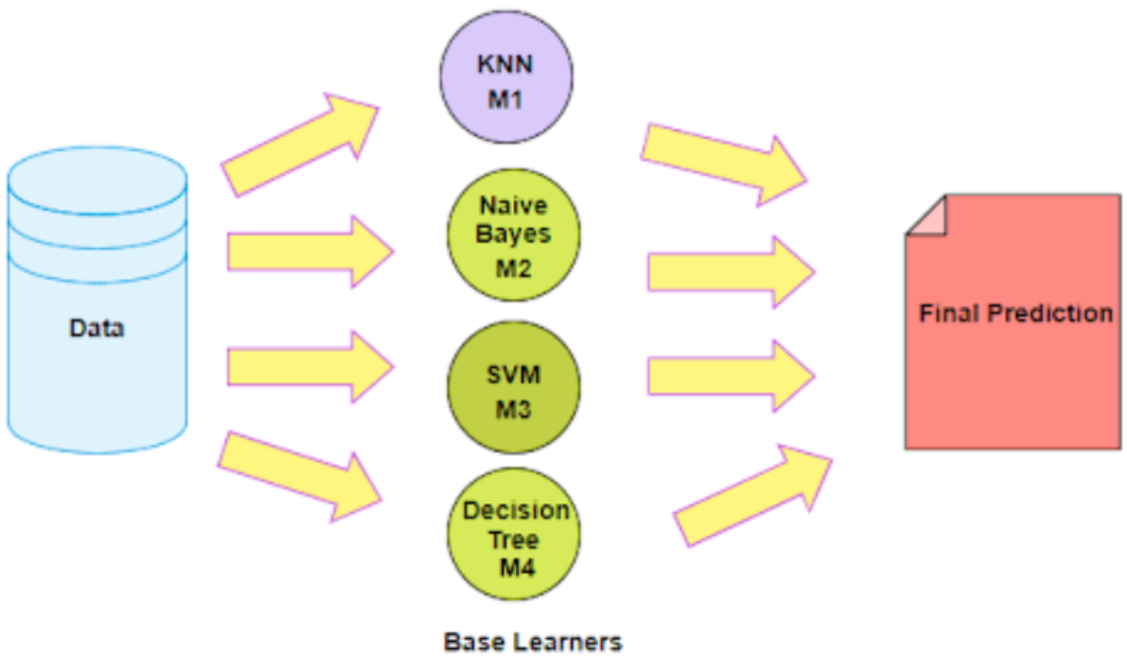
```
def evaluate(X_train, X_test, y_train, y_test):
    # Names of models
    model_name_list = ['Linear Regression','Random Forest', 'Extra Trees', 'SVM']

    # Instantiate the models
    model1 = LinearRegression()
    model2 = RandomForestRegressor(n_estimators=100)
    model3 = ExtraTreesRegressor(n_estimators=100)
    model4 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')

    # Dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)
    results2 = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
```

“c” trades off misclassification of training examples against simplicity of the decision surface. "gamma" parameter defines how far the influence of a single training example reaches.

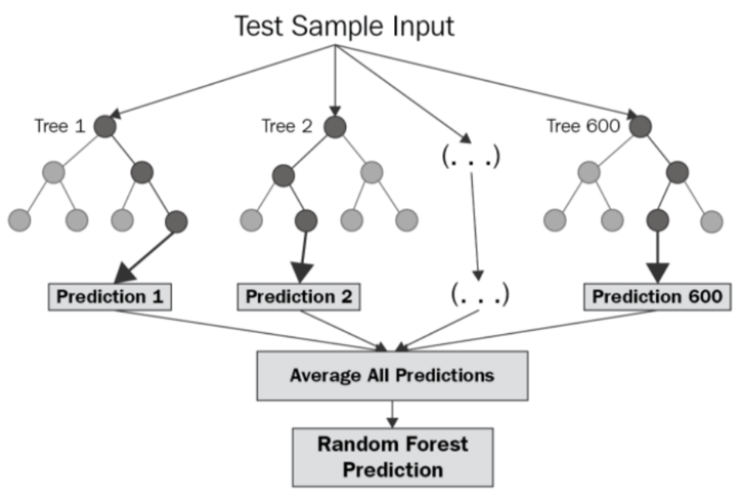
Random Forest

Random Forest is an algorithm based on an ensemble learning method for classification and regression. An Ensemble method is a technique that combines all the predictions from multiple machine learning algorithms to make the more accurate final predictions.

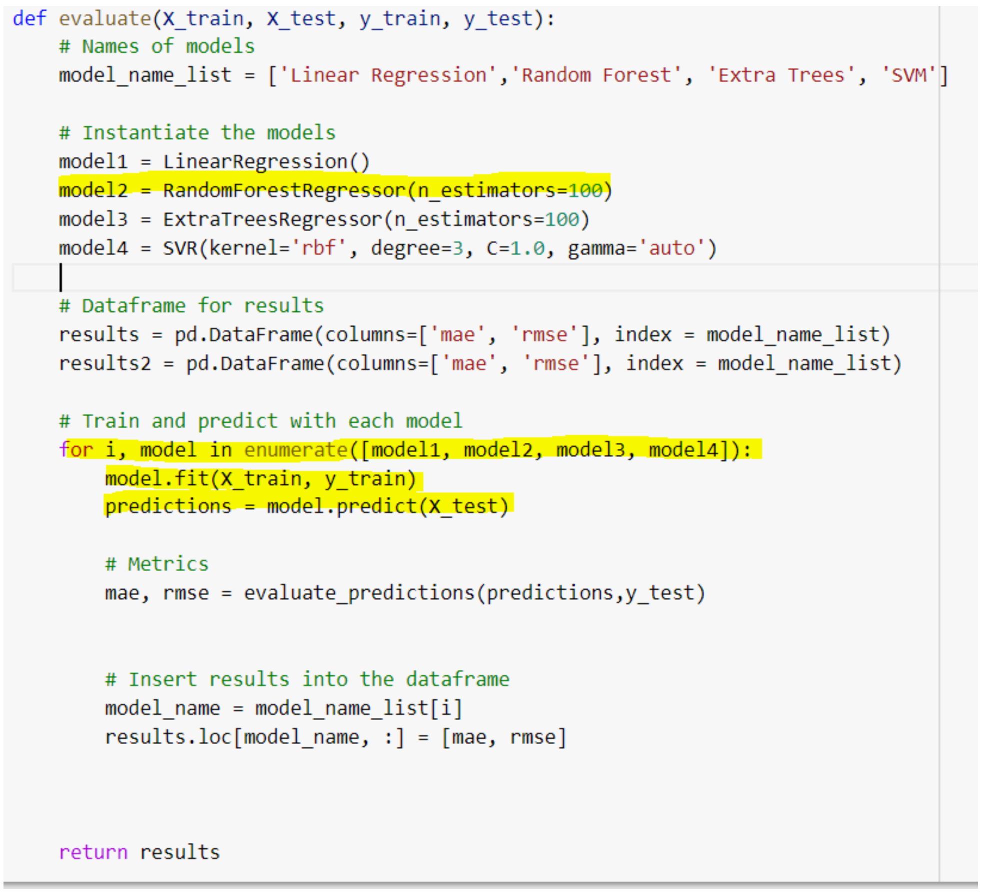
<u>Ensemble Learning Method</u>



There are two types of ensemble learning, Boosting and Bootstrap Aggregation (Bagging). Random forest uses bagging technique, Bagging makes each model run independently, who have no interaction between trees, and then aggregates many decision trees.



Then in this project, after we split the dataset into the training set and test set. We need to train the model with the Random Forest Regression model.


```
def evaluate(X_train, X_test, y_train, y_test):
    # Names of models
    model_name_list = ['Linear Regression','Random Forest', 'Extra Trees', 'SVM']

    # Instantiate the models
    model1 = LinearRegression()
    model2 = RandomForestRegressor(n_estimators=100)
    model3 = ExtraTreesRegressor(n_estimators=100)
    model4 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')

    # Dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)
    results2 = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        # Metrics
        mae, rmse = evaluate_predictions(predictions,y_test)


        # Insert results into the dataframe
        model_name = model_name_list[i]
        results.loc[model_name, :] = [mae, rmse]



    return results
```

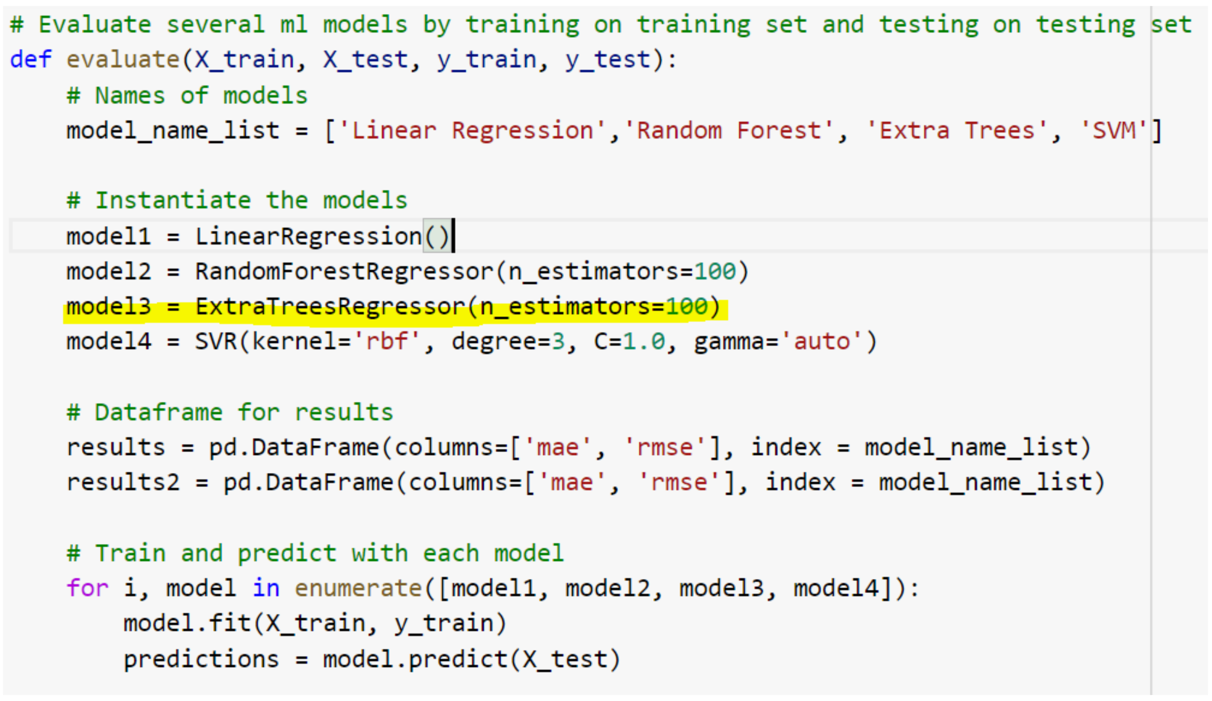
We have used the module sklearn.ensemble and imported the class RandomForestRegressor, then we chose the number of trees in the random forest, we have chosen 100 (n-estimator=100) and we fit the “model2” into the training data.

Extra Trees

The Extra Trees are very similar to Random Forest, they both have the same growing tree procedure. Random forest subsamples the input data with replacement, but Extra Trees use the entire input sample.

We also have another difference for the order to split nodes, Random Forest chooses the optimum split but Extra Trees chooses it randomly.

The Extra Trees algorithm is faster because it randomly chooses the splits point.



```python
# Evaluate several ml models by training on training set and testing on testing set
def evaluate(X_train, X_test, y_train, y_test):
    # Names of models
    model_name_list = ['Linear Regression','Random Forest', 'Extra Trees', 'SVM']

    # Instantiate the models
    model1 = LinearRegression()
    model2 = RandomForestRegressor(n_estimators=100)
    model3 = ExtraTreesRegressor(n_estimators=100)
    model4 = SVR(kernel='rbf', degree=3, C=1.0, gamma='auto')

    # Dataframe for results
    results = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)
    results2 = pd.DataFrame(columns=['mae', 'rmse'], index = model_name_list)

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3, model4]):
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
```

We have used the module sklearn.ensemble and imported the class ExtraTreesRegressor, then we chose the number of trees in the Extra Trees, we have chosen 100 (n-estimator=100) and we fit the “model3” into the training data.

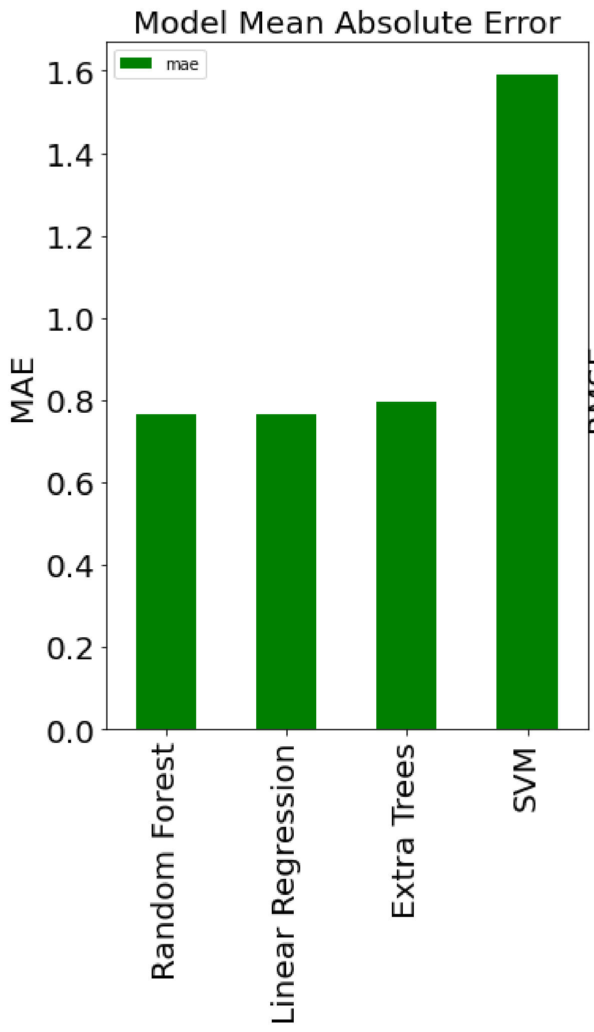
Comparison between models and results

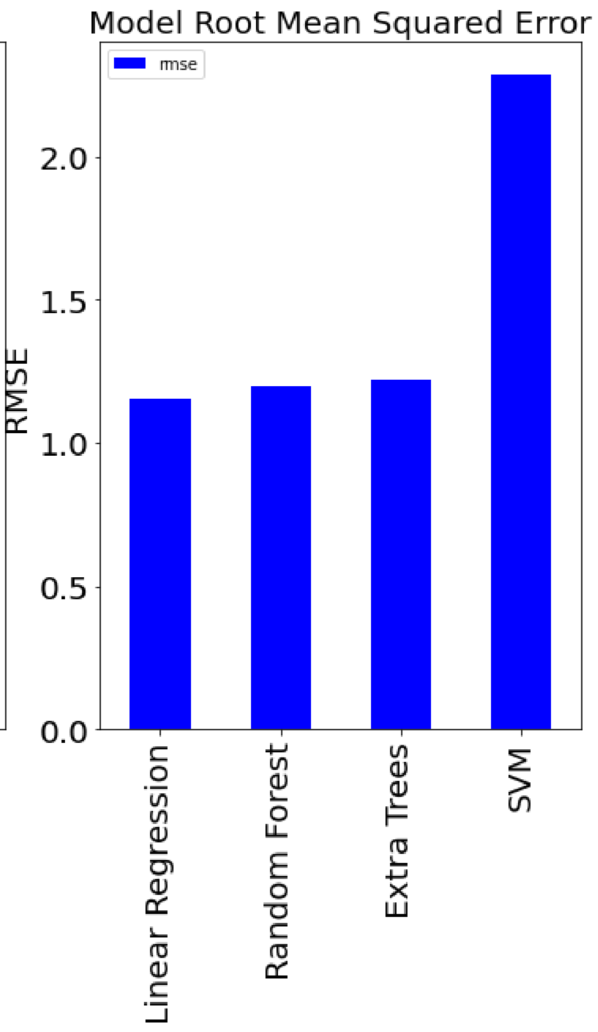
We can see that 3 models are very close, but the Linear Regression is the best model. It can be explained by the fact that each gradient's columns are optimized one by one. However, Random Forest is very close to the first one. It can be explained that each tree takes the best, different scores’ column and the more trees we have, the more the prediction can be accurate. And it’s the same for the extra trees, but this one is taken randomly.

MAE = Mean absolute error/ is a way to measure the accuracy of a given model

RMSE = Root Mean Square Error

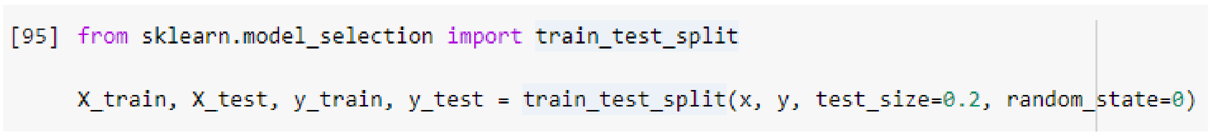
	mae	rmse
Linear Regression	0.765321	1.15352
Random Forest	0.751288	1.19721
Extra Trees	0.759816	1.17602
SVM	1.58974	2.28437





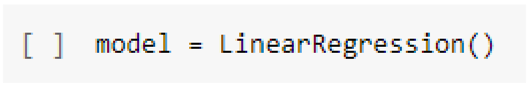
Thanks to those results, we can make the hypothesis that a Linear Regression model will have the best accuracy/score. But to be sure, we will compute this score with all the models and see more clearly the difference between predictions.

So, to get some results, we first need to separate our dataset into two distincts parts: train and test. The train part will be used to fit the model so it can learn from the data and find a way to predict values. With the test part, the model can apply what it learns and make predictions on it.

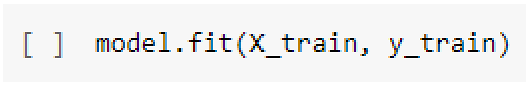


```
[95] from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

As explained, linear regression is the model where the accuracy should be the best. So we are going to present what we have done for linear regression but we also did it for the 3 others models. The results will be shown later.
So, after loading the model, we fit it on our train variables.

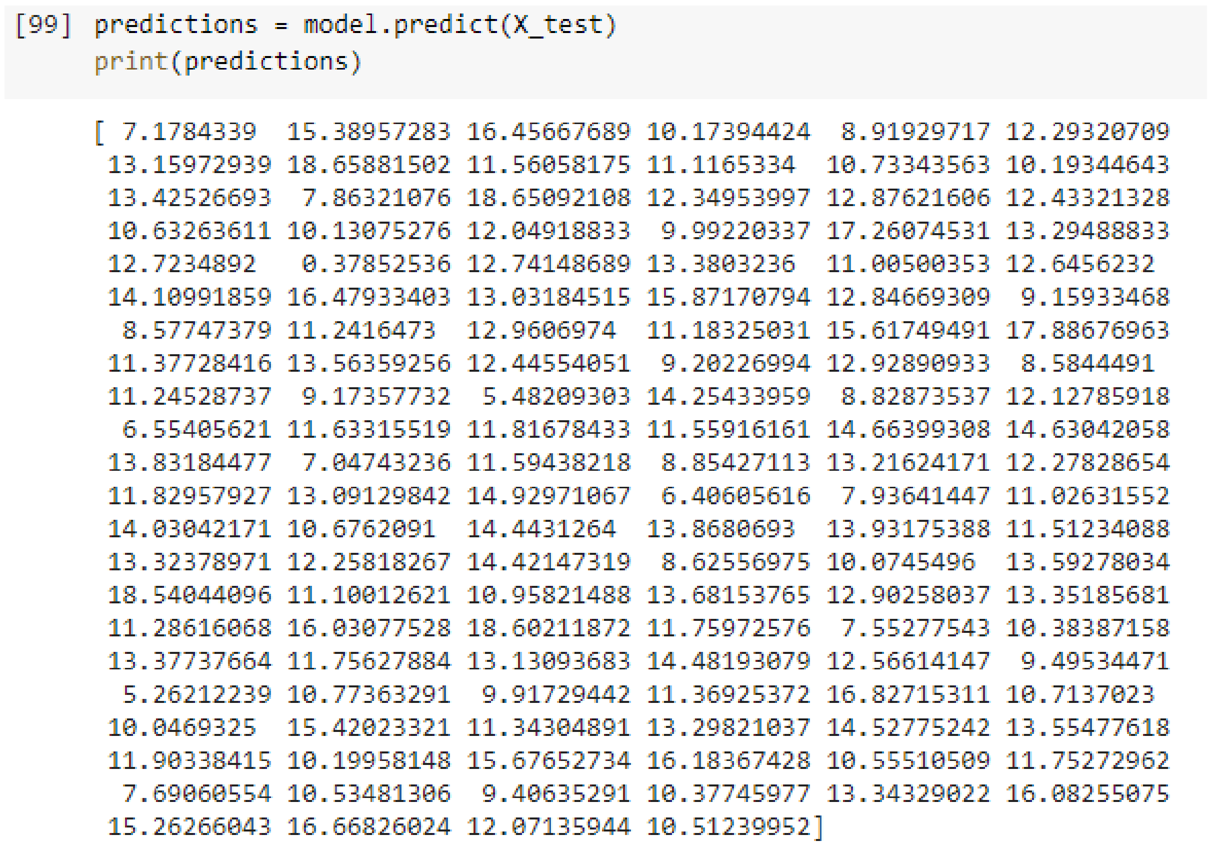


```
[ ]  model = LinearRegression()
```



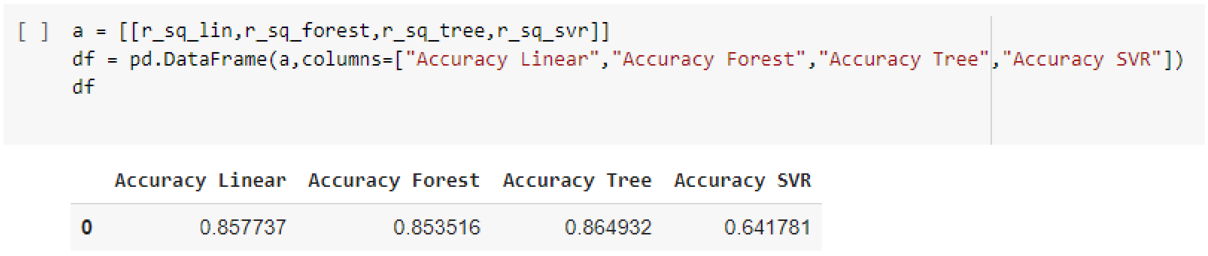
```
[ ]  model.fit(X_train, y_train)
```

After fitting the model with the train's data, we can predict new values using the test's data. We obtain the following results.



```
[99] predictions = model.predict(X_test)
     print(predictions)

     [ 7.1784339  15.38957283 16.45667689 10.17394424  8.91929717 12.29320709
      13.15972939 18.65881502 11.56058175 11.1165334  10.73343563 10.19344643
      13.42526693  7.86321076 18.65092108 12.34953997 12.87621606 12.43321328
      10.63263611 10.13075276 12.04918833  9.99220337 17.26074531 13.29488833
      12.7234892   0.37852536 12.74148689 13.3803236  11.00500353 12.6456232
      14.10991859 16.47933403 13.03184515 15.87170794 12.84669309  9.15933468
       8.57747379 11.2416473  12.9606974  11.18325031 15.61749491 17.88676963
      11.37728416 13.56359256 12.44554051  9.20226994 12.92890933  8.5844491
      11.24528737  9.17357732  5.48209303 14.25433959  8.82873537 12.12785918
       6.55405621 11.63315519 11.81678433 11.55916161 14.66399308 14.63042058
      13.83184477  7.04743236 11.59438218  8.85427113 13.21624171 12.27828654
      11.82957927 13.09129842 14.92971067  6.40605616  7.93641447 11.02631552
      14.03042171 10.6762091  14.4431264  13.8680693  13.93175388 11.51234088
      13.32378971 12.25818267 14.42147319  8.62556975 10.0745496  13.59278034
      18.54044096 11.10012621 10.95821488 13.68153765 12.90258037 13.35185681
      11.28616068 16.03077528 18.60211872 11.75972576  7.55277543 10.38387158
      13.37737664 11.75627884 13.13093683 14.48193079 12.56614147  9.49534471
       5.26212239 10.77363291  9.91729442 11.36925372 16.82715311 10.7137023
      10.0469325  15.42023321 11.34304891 13.29821037 14.52775242 13.55477618
      11.90338415 10.19958148 15.67652734 16.18367428 10.55510509 11.75272962
       7.69060554 10.53481306  9.40635291 10.37745977 13.34329022 16.08255075
      15.26266043 16.66826024 12.07135944 10.51239952]
```

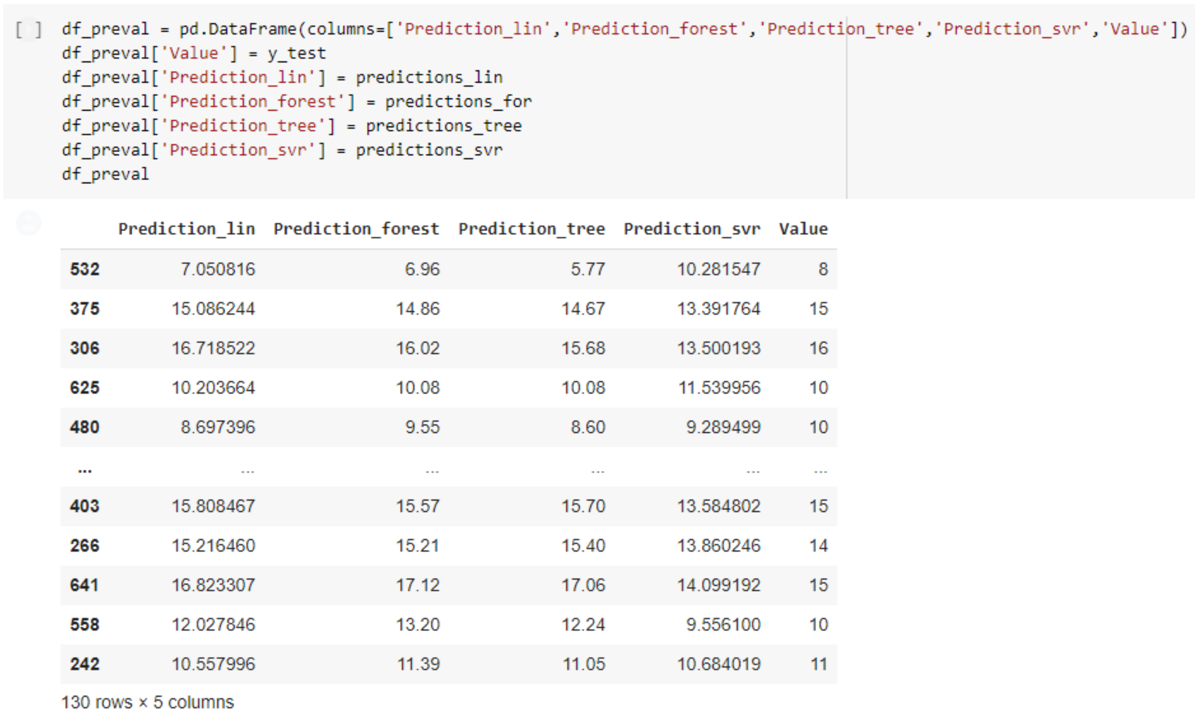
Those results alone are not very understandable by just seeing them. Despite our model being optimized (with gradient descent for linear regression for example), we can't know if the model predicts well by looking at the above values. So we chose to run another method to get the score/accuracy of the model. This score represents the precision of the prediction, it means how far the predicted values are from the original ones.



```
[ ]  a = [[r_sq_lin,r_sq_forest,r_sq_tree,r_sq_svr]]
     df = pd.DataFrame(a,columns=["Accuracy Linear","Accuracy Forest","Accuracy Tree","Accuracy SVR"])
     df
```

```
     Accuracy Linear  Accuracy Forest  Accuracy Tree  Accuracy SVR
0           0.857737         0.853516       0.864932      0.641781
```

We can observe the accuracy of the 4 models we talked, explained and tested earlier. With the analysis we made before, we can see that the linear regression model has not the best accuracy but this is the Extra Trees model with an accuracy of 0.86. We show what were the predicted values of the Linear Regression model but something interesting ould be to compare predicted values of each model between them with the actual grade of the student.



```
[ ]  df_preval = pd.DataFrame(columns=['Prediction_lin','Prediction_forest','Prediction_tree','Prediction_svr','Value'])
     df_preval['Value'] = y_test
     df_preval['Prediction_lin'] = predictions_lin
     df_preval['Prediction_forest'] = predictions_for
     df_preval['Prediction_tree'] = predictions_tree
     df_preval['Prediction_svr'] = predictions_svr
     df_preval
```

```
     Prediction_lin  Prediction_forest  Prediction_tree  Prediction_svr  Value
532        7.050816               6.96             5.77       10.281547      8
375       15.086244              14.86            14.67       13.391764     15
306       16.718522              16.02            15.68       13.500193     16
625       10.203664              10.08            10.08       11.539956     10
480        8.697396               9.55             8.60        9.289499     10
...             ...                ...              ...             ...    ...
403       15.808467              15.57            15.70       13.584802     15
266       15.216460              15.21            15.40       13.860246     14
641       16.823307              17.12            17.06       14.099192     15
558       12.027846              13.20            12.24        9.556100     10
242       10.557996              11.39            11.05       10.684019     11
130 rows × 5 columns
```

To conclude, after choosing a dataset that was interesting for us, we searched which columns could influence the one we wanted to predict, the final grade's column: G3. We searched and chose different models to compare them and find the best one for our dataset. Among different models we found that linear regression was the best choice for our machine learning problem. Thanks to it, we could predict the final grades of students quite well according to some features.

Annexes

https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/

https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f
https://www.freecodecamp.org/news/svm-machine-learning-tutorial-what-is-the-support-vector-machine-algorithm-explained-with-code-examples/
https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html
https://www.scribbr.com/statistics/simple-linear-regression/
https://archive.ics.uci.edu/ml/datasets/Student+Performance