PYTHON PROJECT

DIA 3

Lola-Marie MORON
Danila KILIN
Sandra FOUZARI

Estimation of obesity levels based on eating habits and physical condition

This dataset include data for the estimation of **obesity levels in individuals** from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition.

17

Attributes

2111

Records

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	Female	21.000000	1.620000	64.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	0.000000	1.000000	no	Public_Transportation	Normal_Weight
1	Female	21.000000	1.520000	56.000000	yes	no	3.0	3.0	Sometimes	yes	3.000000	yes	3.000000	0.000000	Sometimes	Public_Transportation	Normal_Weight
2	Male	23.000000	1.800000	77.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	2.000000	1.000000	Frequently	Public_Transportation	Normal_Weight
3	Male	27.000000	1.800000	87.000000	no	no	3.0	3.0	Sometimes	no	2.000000	no	2.000000	0.000000	Frequently	Walking	Overweight_Level_I
4	Male	22.000000	1.780000	89.800000	no	no	2.0	1.0	Sometimes	no	2.000000	no	0.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_II
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2106	Female	20.976842	1.710730	131.408528	yes	yes	3.0	3.0	Sometimes	no	1.728139	no	1.676269	0.906247	Sometimes	Public_Transportation	Obesity_Type_III
2107	Female	21.982942	1.748584	133.742943	yes	yes	3.0	3.0	Sometimes	no	2.005130	no	1.341390	0.599270	Sometimes	Public_Transportation	Obesity_Type_III
2108	Female	22.524036	1.752206	133.689352	yes	yes	3.0	3.0	Sometimes	no	2.054193	no	1.414209	0.646288	Sometimes	Public_Transportation	Obesity_Type_III
2109	Female	24.361936	1.739450	133.346641	yes	yes	3.0	3.0	Sometimes	no	2.852339	no	1.139107	0.586035	Sometimes	Public_Transportation	Obesity_Type_III
2110	Female	23.664709	1.738836	133.472641	yes	yes	3.0	3.0	Sometimes	no	2.863513	no	1.026452	0.714137	Sometimes	Public_Transportation	Obesity_Type_III

FAVC	Frequent consumption of high caloric food	**SCC**	Calories consumption monitoring
FCVC	Frequency of consumption of vegetables	**FAF**	Physical activity frequency
NCP	Number of main meals	**TUE**	Time using technology devices
CAEC	Consumption of food between meals	**CALC**	Consumption of alcohol
SMOKE	Is a smoker	**MTRANS**	Transportation used
CH2O	Consumption of water daily	**NObeyesdad**	Labelled data : corpulency

We have labelled data in here with the various corpulency data that we can find.

	nb_values
NObeyesdad	
Insufficient_Weight	272
Normal_Weight	287
Obesity_Type_I	351
Obesity_Type_II	297
Obesity_Type_III	324
Overweight_Level_I	290
Overweight_Level_II	290

Data exploration

- Very small dataset : 2111 records
- Two types of values (object and float)
- No NaN or missing value, there is no need for imputation

Data Exploration

- Check the writing of each values of categorical variable with np.unique, to see if there is no error.

```
np.unique(df['Gender'])
```

```
array(['Female', 'Male'], dtype=object)
```

```
catcols = ["Gender", "family_history_with_overweight", "FAVC", "CAEC", "SMOKE", "SCC", "CALC", "MTRANS", "NObeyesdad"]
```

✓ 0.2s

```
for col in catcols:
    print (col, df[col].unique())
```

✓ 0.3s

```
Gender ['Female' 'Male']
family_history_with_overweight ['yes' 'no']
FAVC ['no' 'yes']
CAEC ['Sometimes' 'Frequently' 'Always' 'no']
SMOKE ['no' 'yes']
SCC ['no' 'yes']
CALC ['no' 'Sometimes' 'Frequently' 'Always']
MTRANS ['Public_Transportation' 'Walking' 'Automobile' 'Motorbike' 'Bike']
NObeyesdad ['Normal_Weight' 'Overweight_Level_I' 'Overweight_Level_II'
 'Obesity_Type_I' 'Insufficient_Weight' 'Obesity_Type_II'
 'Obesity_Type_III']
```

Data Exploration

```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns, 'percent_missing': percent_missing})
missing_value_df
```

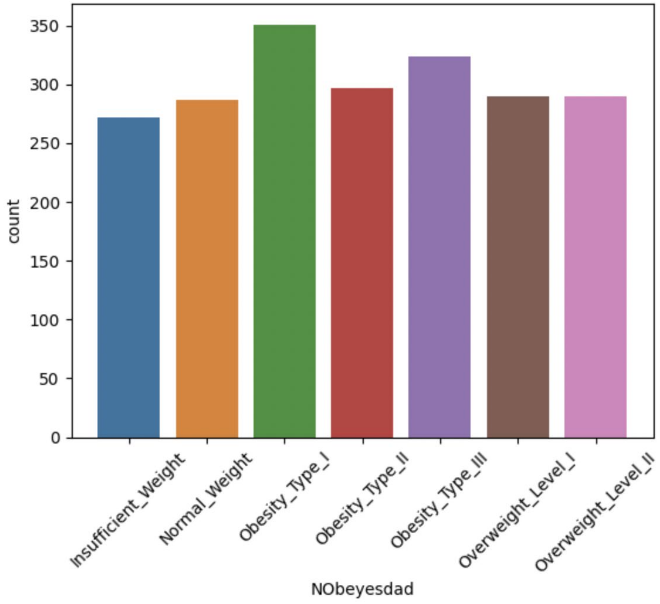
✓ 0.3s

	column_name	percent_missing
Gender	Gender	0.0
Age	Age	0.0
Height	Height	0.0
Weight	Weight	0.0
family_history_with_overweight	family_history_with_overweight	0.0
FAVC	FAVC	0.0
FCVC	FCVC	0.0
NCP	NCP	0.0
CAEC	CAEC	0.0
SMOKE	SMOKE	0.0
CH2O	CH2O	0.0
SCC	SCC	0.0
FAF	FAF	0.0
TUE	TUE	0.0
CALC	CALC	0.0
MTRANS	MTRANS	0.0
NObeyesdad	NObeyesdad	0.0

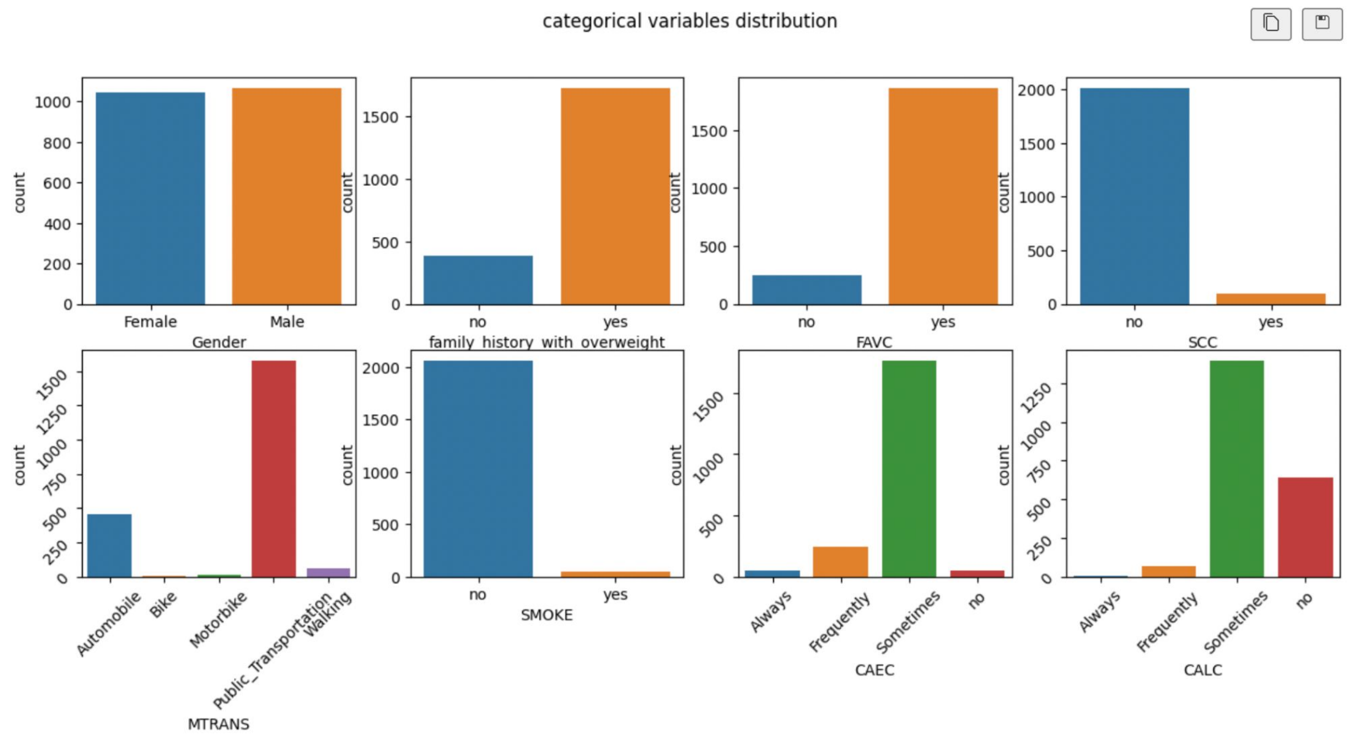
Data visualization

- We can see that only 2 female have type 2 obesity and only 1 male has level 3 obesity whereas weight categories are almost evenly distributed

Gender	NObeyesdad	number
Female	Obesity_Type_II	2
	Overweight_Level_II	103
	Normal_Weight	141
	Overweight_Level_I	145
	Obesity_Type_I	156
	Insufficient_Weight	173
	Obesity_Type_III	323
Male	Obesity_Type_III	1
	Insufficient_Weight	99
	Overweight_Level_I	145
	Normal_Weight	146
	Overweight_Level_II	187
	Obesity_Type_I	195
	Obesity_Type_II	295

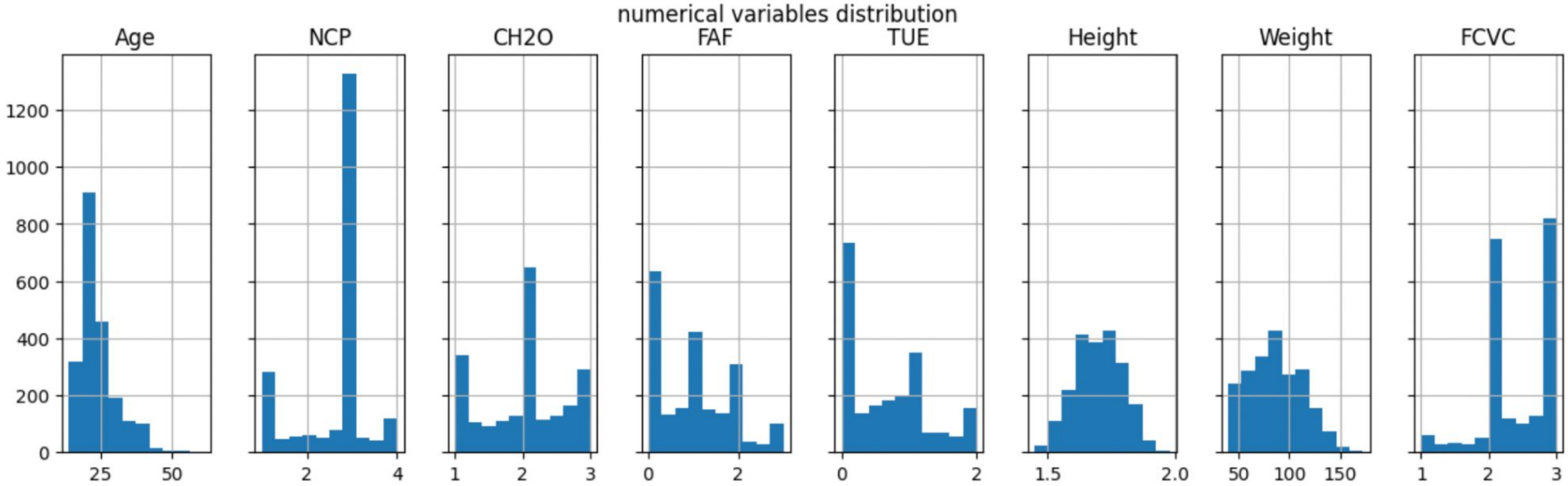


Other categorical variables distribution



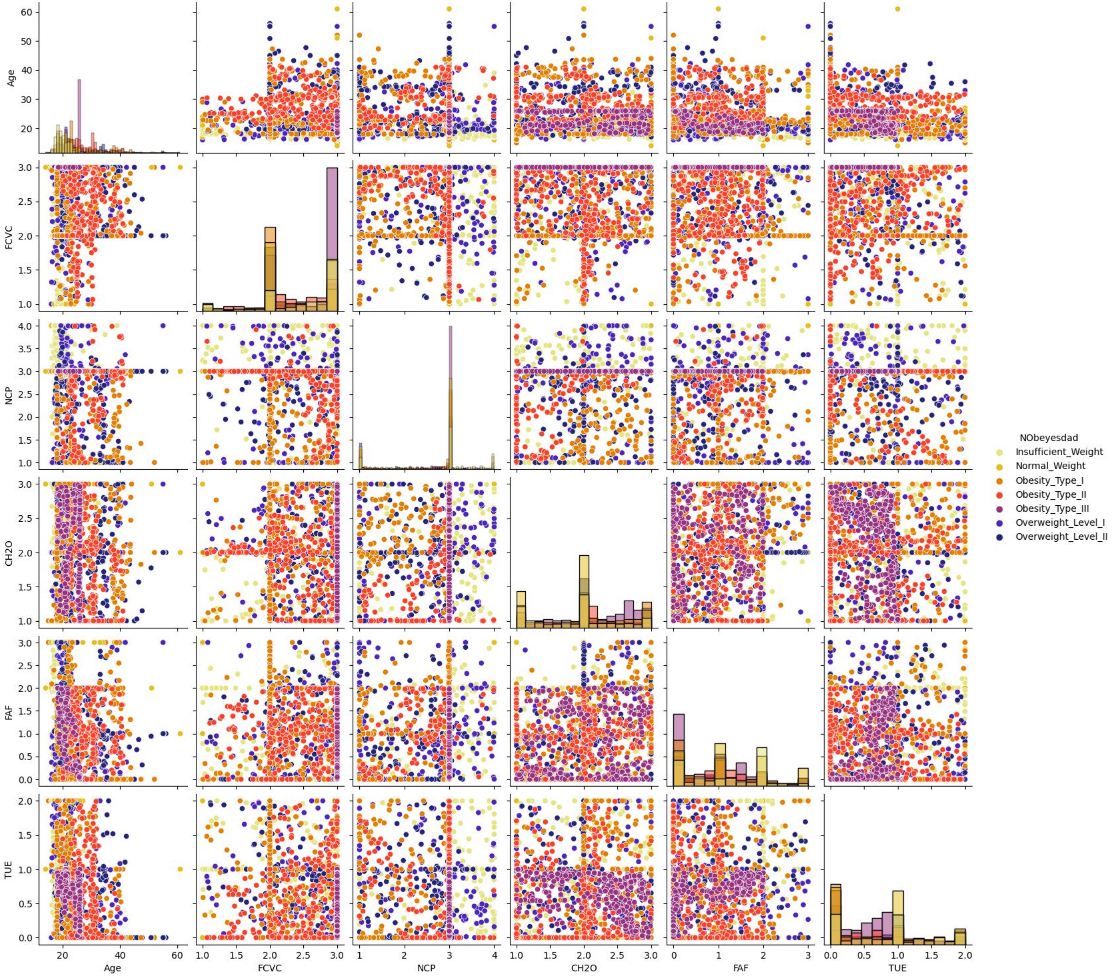
Some variables are evenly distributed while others are not. With this small size of records, it makes training a precise model harder.

Numerical variables distribution

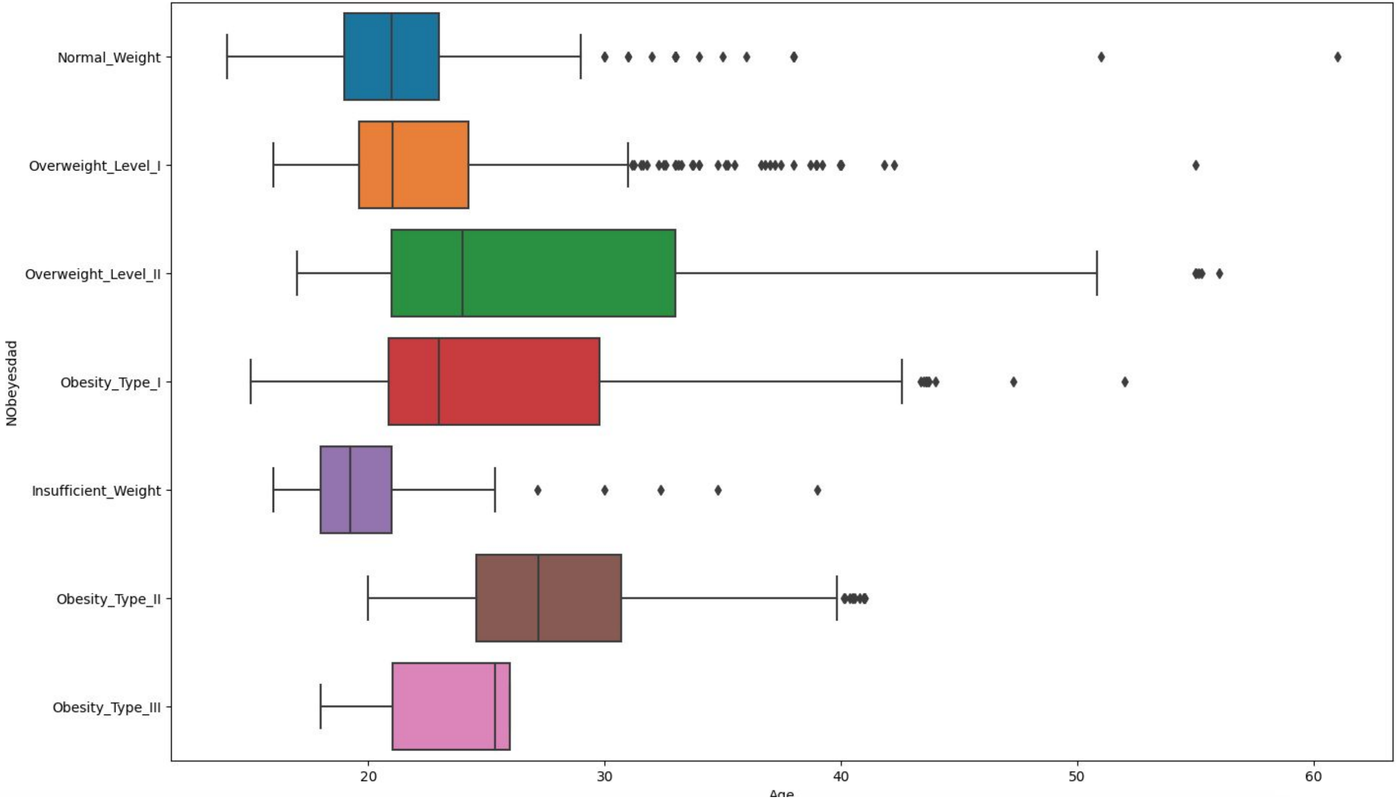


Variables are far from being evenly distributed. For example there are much more children than people above 40. Also other variables do not follow any law because they are linked to behaviour that can vary a lot from person to person.

Pairplot distribution



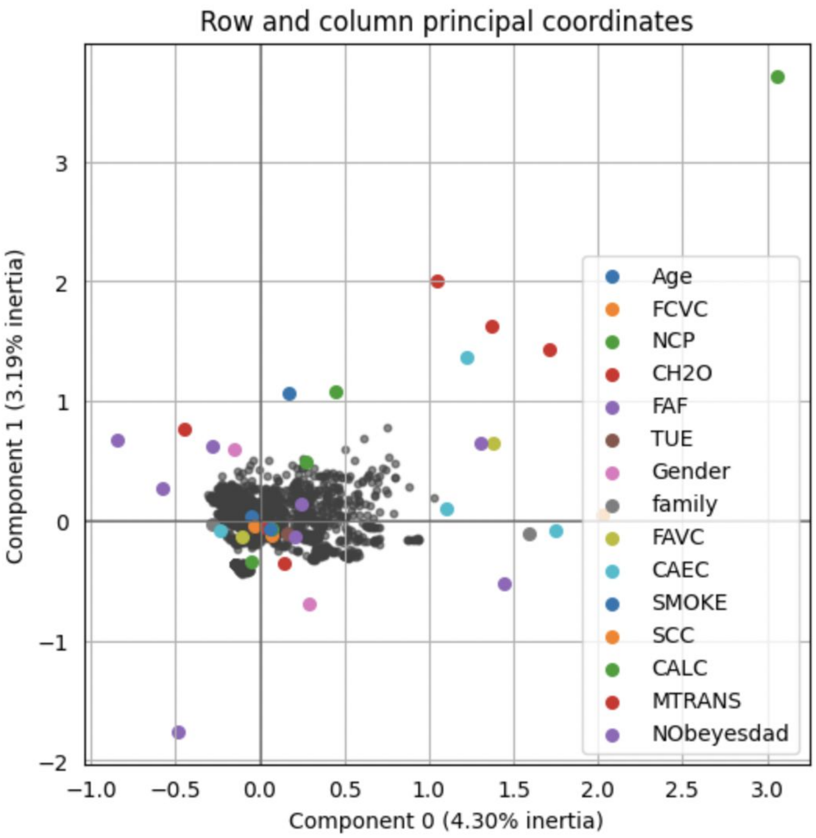
Boxplot



Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Insufficient_Weight
Obesity_Type_II
Obesity_Type_III
NObeyesdad
20
30
40
50
60
Age

MCA analysis

mca did not find components with a lot of inertia so it is a good idea to keep all variables (besides weight and height) for machine learning since they are only 14



Pre-processing

- Conversion of type

```
#do label encoding for CAEC column
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
label_encoder.fit(df['CAEC'])
df['CAEC'] = label_encoder.transform(df['CAEC'])
df
```

Why doing this ?

It is not possible to make some visualisation between numerical and categorical values.

To compare all variables with the predictor variable (visualize boxplot, correlation matrix) a conversion is useful.

Pre-processing

Hot encoding for non ordinal data

```
encoder_df = pd.DataFrame(encoder.fit_transform(df[['MTRANS']]))
```

```
encoder_df
```

	0	1	2	3	4
0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...
2106	0.0	0.0	0.0	1.0	0.0
2107	0.0	0.0	0.0	1.0	0.0
2108	0.0	0.0	0.0	1.0	0.0
2109	0.0	0.0	0.0	1.0	0.0
2110	0.0	0.0	0.0	1.0	0.0

2111 rows × 5 columns

Machine Learning

Our goal is to look for a great model allowing to predict the **Obesity levels** in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition.

Classification algorithms seem to be the most suitable for our problem because we are looking for specific data.

Machine Learning

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```
✓ 0.2s

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```
✓ 0.2s

```
(1477, 20)
(634, 20)
(1477,)
(634,)
```

We first split the dataset into training and testing data.

Scaling

We scaled the data to standardize all features.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train) # fit only on training data
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)  # apply same transformation to test data
```

fit() = generate models parameters from testing data

transform() = parameters generated from fit() method, applied to the model to obtain a scaled dataset

Tests of 2 algorithms RandomForest

To begin, we printed accuracies of the model and some classification report without touching to the parameters.

We can see that the accuracy score is already quite good.

Let’s continue our exploration.

```
#Random Forest
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy through cross-validation score : ", cross_val_score(model, X_train, y_train).mean())
print("Basic accuracy score : ", accuracy_score(y_test, y_pred))
print("Matrice de confusion \n", confusion_matrix(y_test, y_pred))
print("\nClassification report \n", classification_report(y_test, y_pred))
```

✓ 0.7s

```
Accuracy through cross-validation score :  0.9471965185524507
Basic accuracy score :  0.9384858044164038
Matrice de confusion
 [[81  5  0  0  0  0  0]
 [ 3 83  0  0  0  7  0]
 [ 0  1 97  3  0  0  1]
 [ 0  0  1 87  0  0  0]
 [ 0  0  1  0 97  0  0]
 [ 0 10  0  0  0 77  1]
 [ 0  3  0  0  0  3 73]]

Classification report
                     precision    recall  f1-score   support

Insufficient_Weight       0.96      0.94      0.95        86
      Normal_Weight       0.81      0.89      0.85        93
     Obesity_Type_I       0.98      0.95      0.97       102
    Obesity_Type_II       0.97      0.99      0.98        88
   Obesity_Type_III       1.00      0.99      0.99        98
 Overweight_Level_I       0.89      0.88      0.88        88
Overweight_Level_II       0.97      0.92      0.95        79

           accuracy                           0.94       634
          macro avg       0.94      0.94      0.94       634
       weighted avg       0.94      0.94      0.94       634
```

Tests of 2 algorithms RandomForest

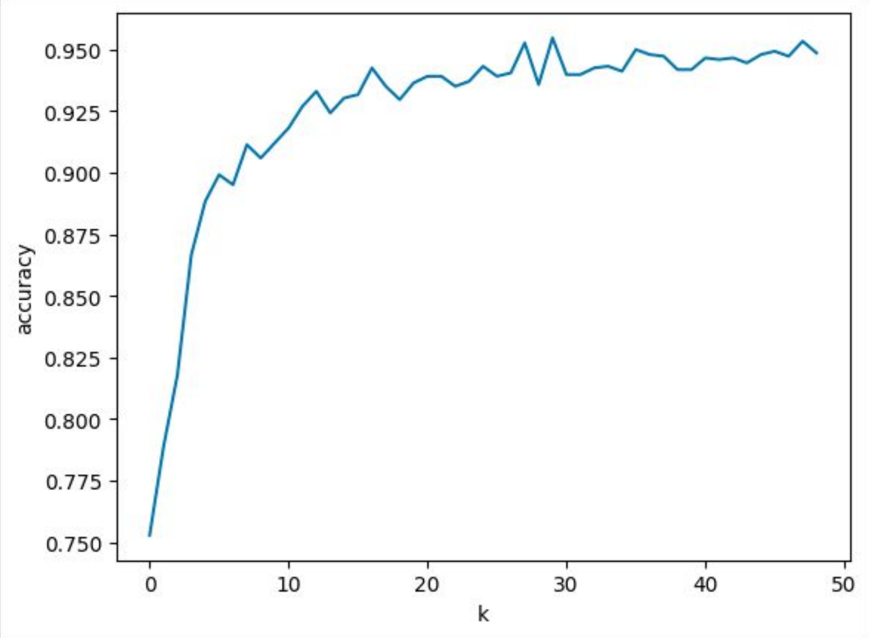
This graph shows the evolution of the accuracy with the number of k estimators (number of trees we want to build).

The goal is to find the best ratio between CPU performance (the more k in high the more time it'll take) and accuracy score.

```
val_score = []
for i in range(1, 50):
    knn = RandomForestClassifier(n_estimators=i)
    score = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy').mean()
    val_score.append(score)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.plot(val_score)
```

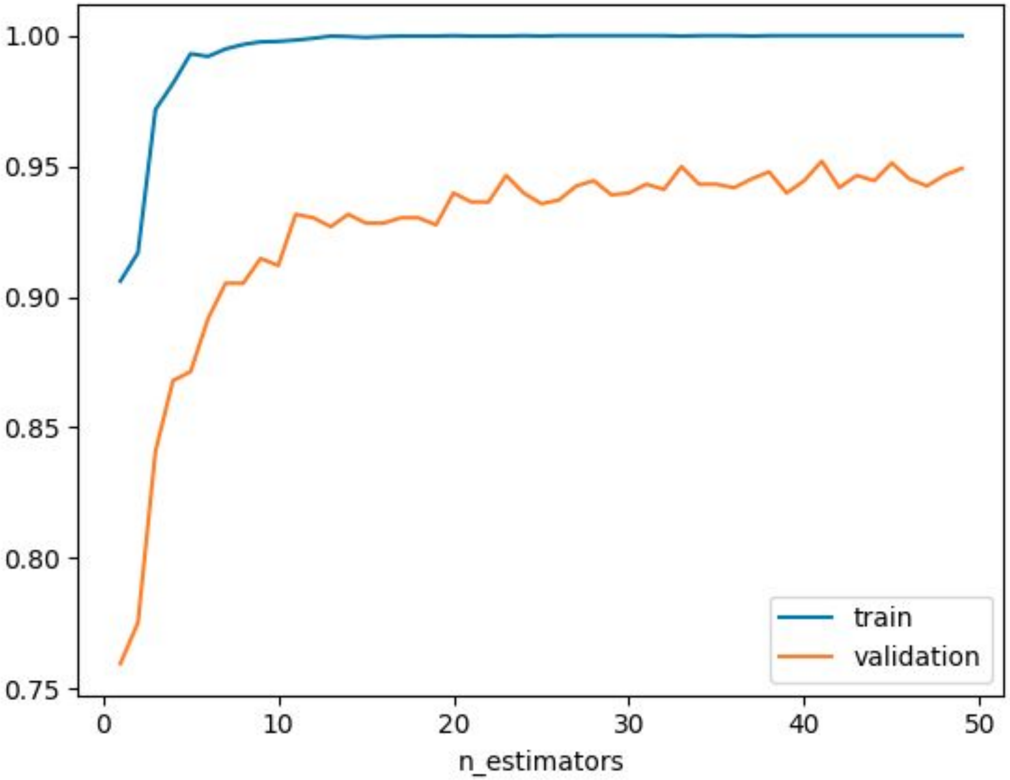
✓ 6.8s

```
[<matplotlib.lines.Line2D at 0x29acf2830>]
```



Tests of 2 algorithms RandomForest

Validation curve



We can see that we are just-right, no overfitting, and no underfitting because the validation line is stil quite high (> 0.9).

Tests of 2 algorithms KNearestNeighbors

We can already see that this algorithm has a worse accuracy than Random Forest.

```
#KNN
model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy through cross-validation score : ", cross_val_score(model, X_train, y_train).mean())
print("Basic accuracy score : ", accuracy_score(y_test, y_pred))
print("Matrice de confusion \n", confusion_matrix(y_test, y_pred))
print("\nClassification report \n", classification_report(y_test, y_pred))
```

✓ 0.3s

```
Accuracy through cross-validation score :  0.7846862116353641
Basic accuracy score :  0.805993690851735
Matrice de confusion
 [[79  4  0  0  0  2  1]
 [21 46  7  5  0  6  8]
 [ 1  2 88  7  0  2  2]
 [ 0  0  2 85  0  1  0]
 [ 0  0  0  0 98  0  0]
 [ 6 12  8  2  0 58  2]
 [ 3  4  6  4  1  4 57]]

Classification report
                     precision    recall  f1-score   support

Insufficient_Weight       0.72      0.92      0.81        86
      Normal_Weight       0.68      0.49      0.57        93
     Obesity_Type_I       0.79      0.86      0.83       102
    Obesity_Type_II       0.83      0.97      0.89        88
   Obesity_Type_III       0.99      1.00      0.99        98
 Overweight_Level_I       0.79      0.66      0.72        88
Overweight_Level_II       0.81      0.72      0.77        79

           accuracy                           0.81       634
          macro avg       0.80      0.80      0.80       634
       weighted avg       0.80      0.81      0.80       634
```

Tuning hyperparameters

We created a function that we are going to use several times to tests various hyperparameters in order to find the best ones.

```
def test_hyperparametres(model, hyperparametres):
    grid = GridSearchCV(model, hyperparametres, n_jobs=-1)
    grid.fit(X_train, y_train)
    print(f"Best score: {grid.best_score_}")
    print(f"Best hyperparametres: {grid.best_params_}")
    print(f"Best estimator: {grid.best_estimator_}")
    return grid.best_score_, grid.best_params_, grid.best_estimator_
```

Tuning hyperparameters Random Forest

Parameters we are working on, several more exists but we decided to focus on the one we understood :

- *Boostrap* : if True, samples are used when building trees. Otherwise, the whole dataset is used
- *max_depth* : maximum depth of the tree
- *max_features* : features at each split
- *min_samples_leaf* : minimum number of samples required to be at a leaf node
- *min_samples_split* : minimum number of samples required to split an internal node
- *n_estimators* : numbers of trees in the forest

Source

Tuning hyperparameters Random Forest

Some loops to find better accuracy

```
hyperparametres = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [7, 8, 9, 10],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
model = RandomForestClassifier()
test_hyperparametres(model, hyperparametres)
```

```
(0.9559894640403115,
 {'bootstrap': True,
  'max_depth': 90,
  'max_features': 10,
  'min_samples_leaf': 3,
  'min_samples_split': 10,
  'n_estimators': 200},
```

```
hyperparametres = {
    'bootstrap': [True],
    'max_depth': [60, 70, 80, 90],
    'max_features': [9, 10, 11],
    'min_samples_leaf': [2, 3, 4],
    'min_samples_split': [8, 9, 10, 11],
    'n_estimators': [100, 200, 400]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.9607352267521758,
 {'bootstrap': True,
  'max_depth': 60,
  'max_features': 10,
  'min_samples_leaf': 2,
  'min_samples_split': 8,
  'n_estimators': 400},
```

```
hyperparametres = {
    'bootstrap': [True],
    'max_depth': [40, 50, 60, 70],
    'max_features': [9, 10, 11],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [6, 7, 8],
    'n_estimators': [200, 400, 600]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.964111314704535,
 {'bootstrap': True,
  'max_depth': 50,
  'max_features': 11,
  'min_samples_leaf': 1,
  'min_samples_split': 6,
  'n_estimators': 200},
```

```
hyperparametres = {
    'bootstrap': [True],
    'max_depth': [30, 40, 50, 60],
    'max_features': [9, 10, 11],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [5, 6, 7],
    'n_estimators': [100, 200, 400]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.9681768208886853,
 {'bootstrap': True,
  'max_depth': 40,
  'max_features': 10,
  'min_samples_leaf': 1,
  'min_samples_split': 5,
  'n_estimators': 100},
```

We achieved to add +0.012 to the accuracy score

Tuning hyperparameters KNearestNeighbors

Parameters we are working on, several more exists but we decided to focus on the one we understood :

- *n_neighbors* : number of neighbors
- *weights* : weight function used. Uniform : all points in each neighborhood are weighted equally. Distance : weight points by the inverse of their distance.
- *algorithm* : Algorithm used for the computation. ball_tree : using BallTree. kd_tree = using KDTree. brute : using brute-force seach.
- *leaf_size* : leaf size passed to the BallTree or KDTree algorithms. Can have impact on the speed and memory required for the operation.

Source

Tuning hyperparameters KNearestNeighbors

Some loops to find better accuracy

```
hyperparametres = {
    'n_neighbors': [5, 10, 15, 20, 25, 30],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [20, 30, 40]
}
model = KNeighborsClassifier()
test_hyperparametres(model, hyperparametres)
```

```
(0.8645716903344022,
 {'algorithm': 'auto',
  'leaf_size': 20,
  'n_neighbors': 5,
  'weights': 'distance'},
```

```
hyperparametres = {
    'n_neighbors': [3, 5, 10, 15, 20, 25, 30],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.8740563444800733,
 {'algorithm': 'ball_tree',
  'leaf_size': 10,
  'n_neighbors': 3,
  'weights': 'distance'},
```

```
hyperparametres = {
    'n_neighbors': [2, 3, 5, 10, 15],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [5, 10, 20]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.895048098946404,
 {'algorithm': 'ball_tree',
  'leaf_size': 5,
  'n_neighbors': 2,
  'weights': 'distance'},
```

```
hyperparametres = {
    'n_neighbors': [1, 2, 3, 5],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [3, 5, 10]
}
test_hyperparametres(model, hyperparametres)
```

```
(0.895048098946404,
 {'algorithm': 'ball_tree',
  'leaf_size': 3,
  'n_neighbors': 1,
  'weights': 'uniform'},
```

We achieved to add +0.031 to the accuracy score

Find the best ML algorithm

We selected some classification models in order to test them and find the one with the best performance.

```python
def classification_models():
    models = []
    models.append(('Logisitc Rregression', linear_model.LogisticRegression()))
    models.append(('Linear Discriminant Analysis', discriminant_analysis.LinearDiscriminantAnalysis()))
    models.append(('KNN', neighbors.KNeighborsClassifier()))
    models.append(('Decision Tree', tree.DecisionTreeClassifier()))
    models.append(('Gaussian Naive Bayes', naive_bayes.GaussianNB()))
    models.append(('Support Vector Machine', svm.SVC()))
    models.append(('Random Forest', ensemble.RandomForestClassifier()))
    models.append(('Gradient Boosting', ensemble.GradientBoostingClassifier()))
    models.append(('Extra Trees', ensemble.ExtraTreesClassifier()))
    models.append(('Ada Boost', ensemble.AdaBoostClassifier()))
    models.append(('Multilayer Perceptron', neural_network.MLPClassifier()))
    models.append(('XGBoost', linear_model.SGDClassifier()))
    return models
```

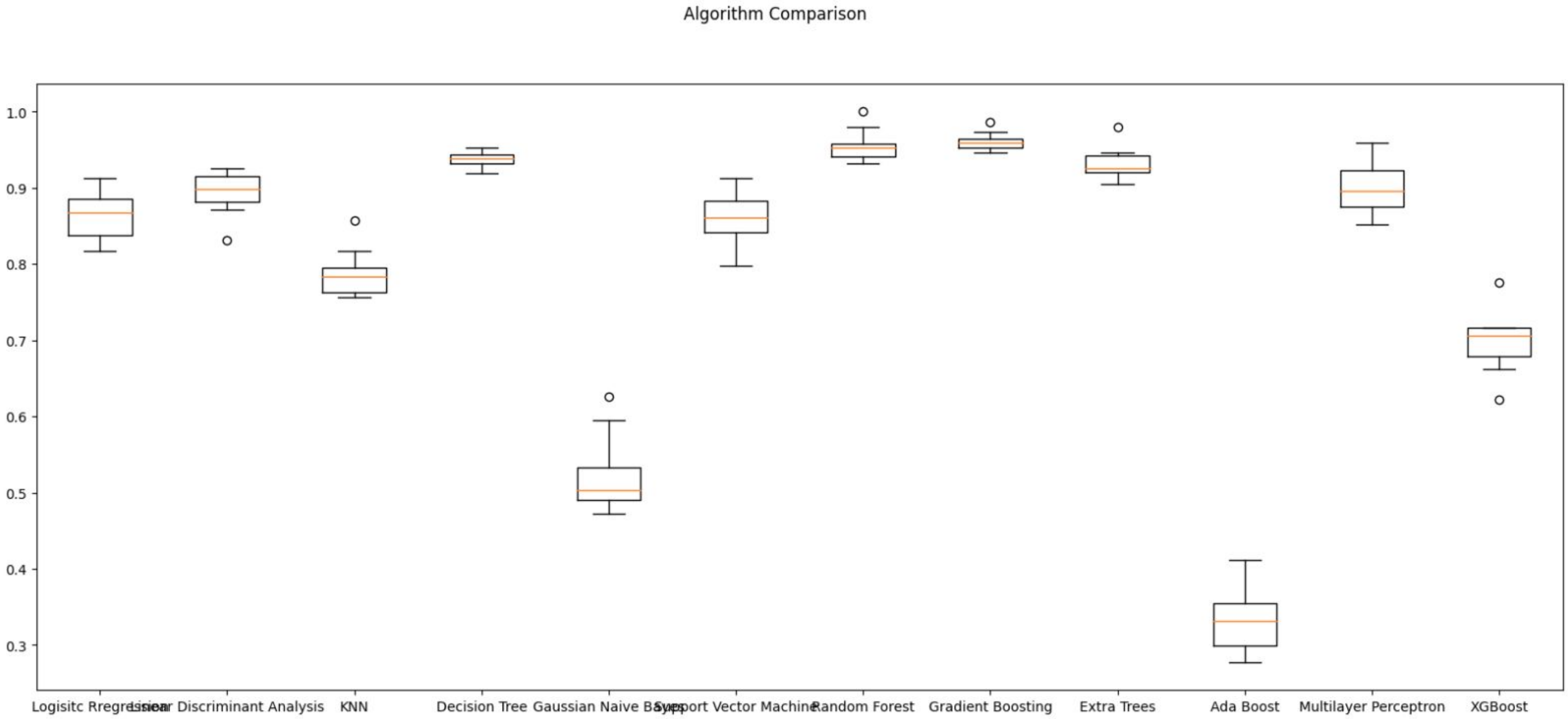
Find the best ML algorithm

We evaluated each model and stored the best performance to retrieve it later.

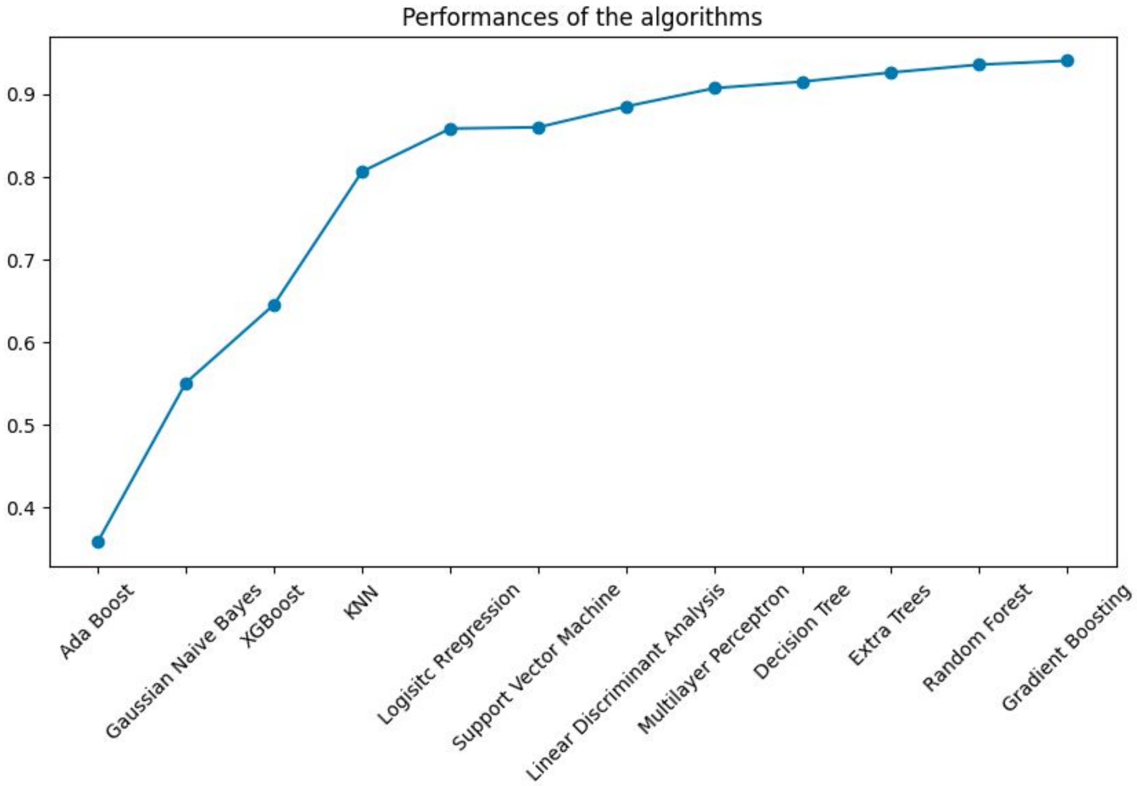
```
# evaluate each model in turn
performances_scoring = {}
performances_scoring_crosval = []
names = []
best_algorithm = 0
best_algorithm_crossval = 0
best_perf = 0
best_perf_crossval = 0
scoring = 'accuracy'
for name, model in classification_models():
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    performances_scoring_crosval.append(cv_results)
    names.append(name)
    model.fit(X_train, y_train)
    performance = model.score(X_test, y_test)
    if cv_results.mean() >  best_perf_crossval:
            best_algorithm_crossval = model
            best_perf_crossval = cv_results.mean()
    if performance >  best_perf:
            best_algorithm = model
            best_perf = performance
    if 0<performance and performance<1:
        performances_scoring[name] = [performance]

    msg_crossvalidation = "Accuracy through cross validation %s: %f" % (name, cv_results.mean())
    msg = "Accuracy scoring %s: %f" % (name, performance)
    print(msg)
    print(msg_crossvalidation)
    print ("="*30)
```

Find the best ML algorithm

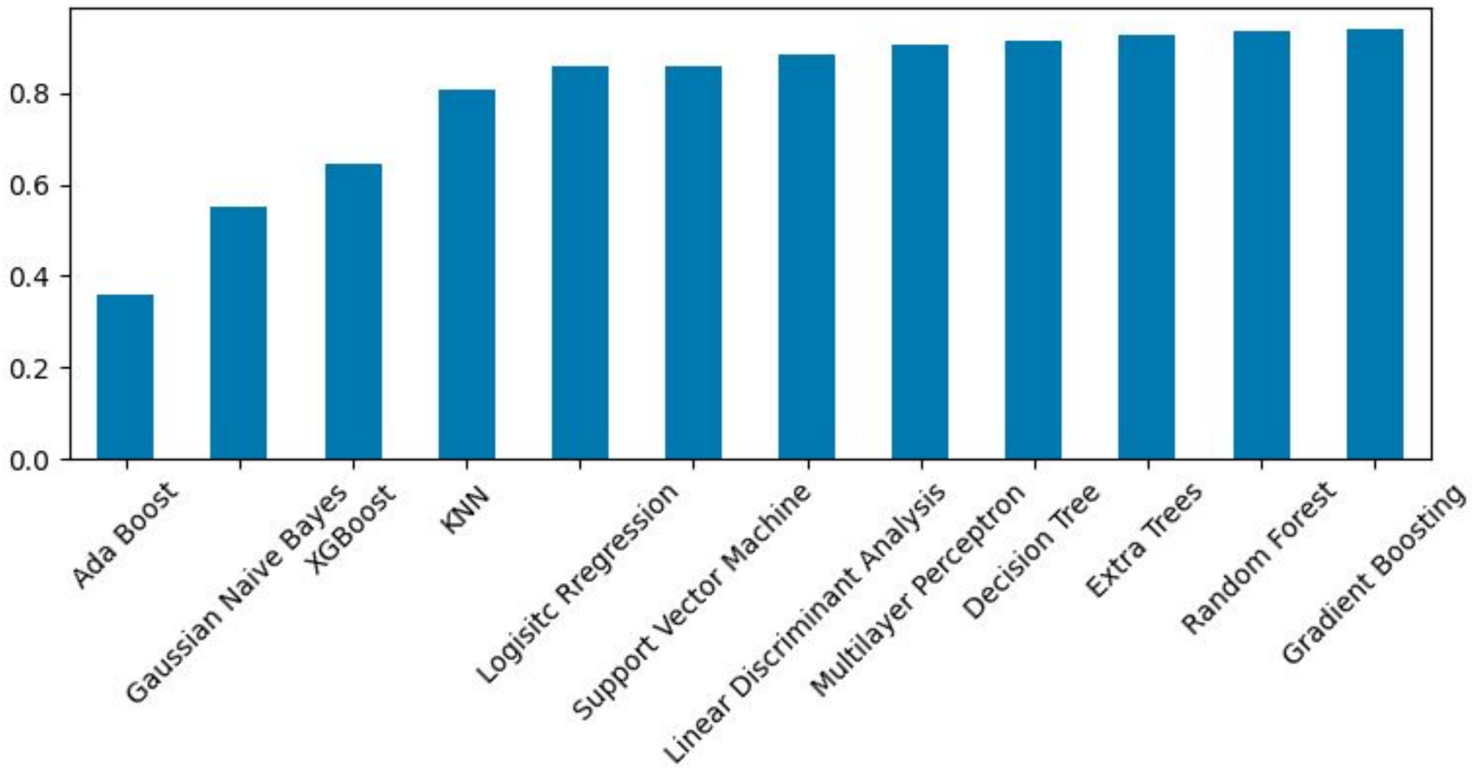
This boxplot show the different algorithms results.



Find the best ML algorithm



Find the best ML algorithm



Find the best ML algorithm

We found that the gradient boosting algorithm has the best performance.

```
best_algorithm, best_perf
✓ 0.3s

(GradientBoostingClassifier(), 0.9400630914826499)
```

Tuning hyperparameters GradientBoosting

Let's see if we can increase the accuracy of the potential best model we found.

```
hyperparametres = {
    'loss': ['log_loss','exponential'],
    'learning_rate': [10, 1, 0.1, 0.01, 0.001],
    'n_estimators': [10, 100, 1000],
    'subsample': [0.1, 0.5, 1.0],
    'criterion': ['friedman_mse', 'square_error'],
    'max_features': ['sqrt', 'log2', None],

}
model = ensemble.GradientBoostingClassifier()
test_hyperparametres(model, hyperparametres)
```

```
(0.9614109024278517,
 {'criterion': 'friedman_mse',
  'learning_rate': 0.1,
  'loss': 'log_loss',
  'max_features': None,
  'n_estimators': 1000,
  'subsample': 0.5},
```

```
hyperparametres = {
    'loss': ['log_loss'],
    'learning_rate': [1, 0.1, 0.01],
    'n_estimators': [100, 1000, 5000],
    'subsample': [0.1, 0.5, 1.0],
    'criterion': ['friedman_mse'],
    'max_features': ['sqrt', 'log2'],

}
```

```
(0.9620888685295466,
 {'criterion': 'friedman_mse',
  'learning_rate': 0.1,
  'loss': 'log_loss',
  'max_features': 'log2',
  'n_estimators': 1000,
  'subsample': 0.5},
```

We will stop there because the operation is already taking +10min.

In the end, we have an accuracy of 0.9620888685295466 with the parameters above for Gradient Boosting algorithm.

We achieved to add +0.0006 to the accuracy score.

Prediction with model

```
model = ensemble.GradientBoostingClassifier(criterion= 'friedman_mse', learning_rate= 0.1, loss= 'log_loss', max_features= 'log2', n_estimators= 1000, subsample= 0.5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy through cross-validation score : ", cross_val_score(model, X_train, y_train).mean())
print("Basic accuracy score : ", accuracy_score(y_test, y_pred))
print("Matrice de confusion \n", confusion_matrix(y_test, y_pred))
print("\nClassification report \n", classification_report(y_test, y_pred))
```

✓ 18.1s

```
Accuracy through cross-validation score :  0.9641227668346313
Basic accuracy score :  0.9463722397476341
Matrice de confusion
 [[84  2  0  0  0  0  0]
 [ 1 84  0  0  0  8  0]
 [ 0  0 96  2  0  4  0]
 [ 0  0  0 88  0  0  0]
 [ 0  0  0  1 97  0  0]
 [ 0  6  0  0  0 80  2]
 [ 0  3  2  0  0  3 71]]

Classification report
                      precision    recall  f1-score   support

Insufficient_Weight       0.99      0.98      0.98        86
      Normal_Weight       0.88      0.90      0.89        93
     Obesity_Type_I       0.98      0.94      0.96       102
    Obesity_Type_II       0.97      1.00      0.98        88
   Obesity_Type_III       1.00      0.99      0.99        98
 Overweight_Level_I       0.84      0.91      0.87        88
Overweight_Level_II       0.97      0.90      0.93        79
```

Django API

To open the django API, you have to type to command **python3 manage.py runserver** in the django directory on your terminal as shown here.

```
lola@MacBook-Pro-de-Lola Python_project % cd Django_API/
lola@MacBook-Pro-de-Lola Django_API % python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
December 06, 2022 - 22:08:50
Django version 4.1.4, using settings 'python_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
Not Found: /apple-touch-icon.png
Not Found: /favicon.ico
Not Found: /apple-touch-icon-precomposed.png
[06/Dec/2022 22:08:54] "GET /apple-touch-icon.png HTTP/1.1" 404 2250
[06/Dec/2022 22:08:54] "GET /favicon.ico HTTP/1.1" 404 2223
[06/Dec/2022 22:08:54] "GET /apple-touch-icon-precomposed.png HTTP/1.1" 404 2286
Not Found: /
[06/Dec/2022 22:08:54] "GET / HTTP/1.1" 404 2172
[06/Dec/2022 22:08:59] "GET /main/ HTTP/1.1" 200 118471
```

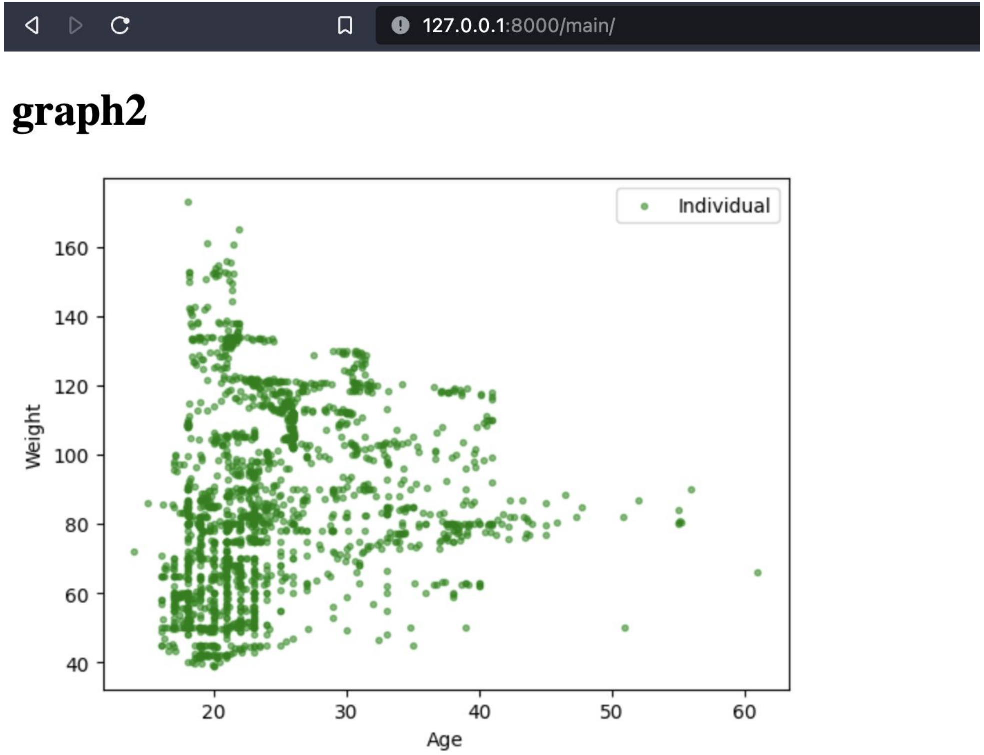
Then open your browser with the go to http://127.0.0.1:8000/main

(Don't forget the /main)

Django API

Example of utilisation :

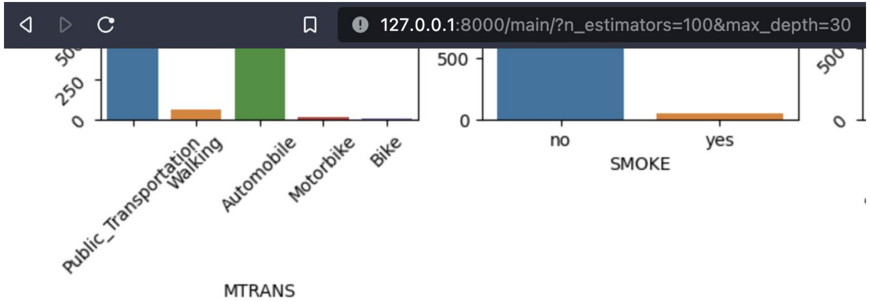
To change the feature used on the second graph from x = age to x = height do http://127.0.0.1:8000/main/?x_graph2=Height



Django API

Example of utilisation :

Change all the parameters listed in here for the Random Forest algorithm
http://127.0.0.1:8000/main/?n_estimators=100&max_depth=30



Parameters used :

bootstrap : True

max_depth : 30

max_features : 10

min_samples_leaf : 3

min_samples_split : 3

n_estimators : 100

Accuracy Score : 0.9384858044164038

Confusion Matrix :

	0	1	2	3	4	5	6
0	82	4	0	0	0	0	0
1	2	82	0	0	0	9	0
2	0	0	95	3	0	3	1
3	0	0	2	86	0	0	0
4	0	0	0	1	97	0	0
5	0	9	0	0	0	76	3
6	0	0	0	0	0	2	77

Classification Report precision recall f1-score support Insufficient_Weight 0.98 0.95 0.96 86 Normal_Weig
0.99 98 Overweight_Level_I 0.84 0.86 0.85 88 Overweight_Level_II 0.95 0.97 0.96 79 accuracy 0.94 634