Python for data analysis

ESTIMATION OF OBESITY LEVELS BASED ON EATING HABITS AND PHYSICAL CONDITION

Exploration du dataset

Cette partie est dédié a la compréhension du dataset et de ses paramètres.

Ce dataset comprend des données issues de 3 pays d'Amérique du sud, le Pérou, le Mexique et la Colombie. Et il a pour but d'estimer le niveau d'obésité de personnes allant de 14 à 61 ans.

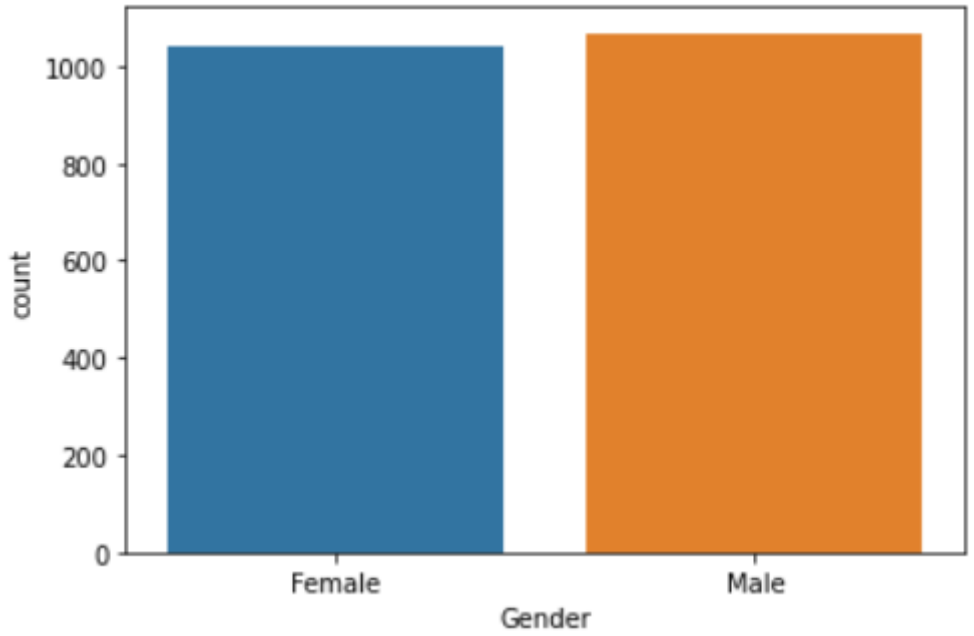
Les données ont été récolté via un sondage web où des utilisateurs anonymes ont répondu à différents questions. On ne peut être certain de la véracité de ces données, il faut donc prendre les résultats de cet exercice avec un certain recul.

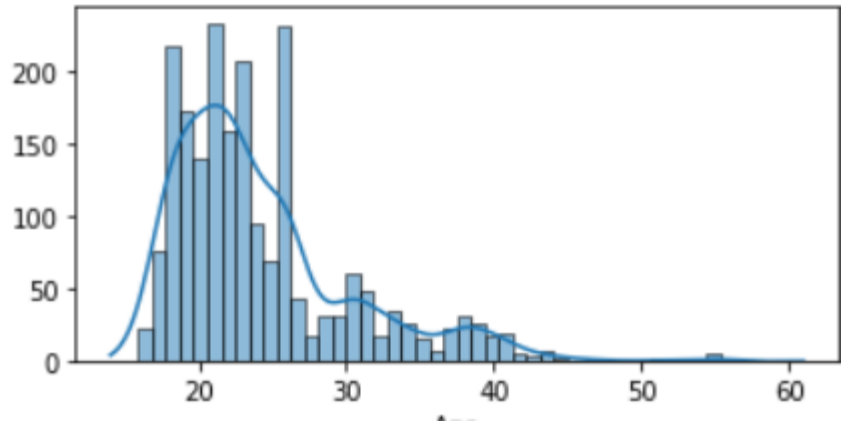
Abstract: This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition.

Data Set Characteristics:	Multivariate	**Number of Instances:**	2111	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	17	**Date Donated**	2019-08-27
Associated Tasks:	Classification, Regression, Clustering	**Missing Values?**	N/A	**Number of Web Hits:**	20749

Exploration du dataset

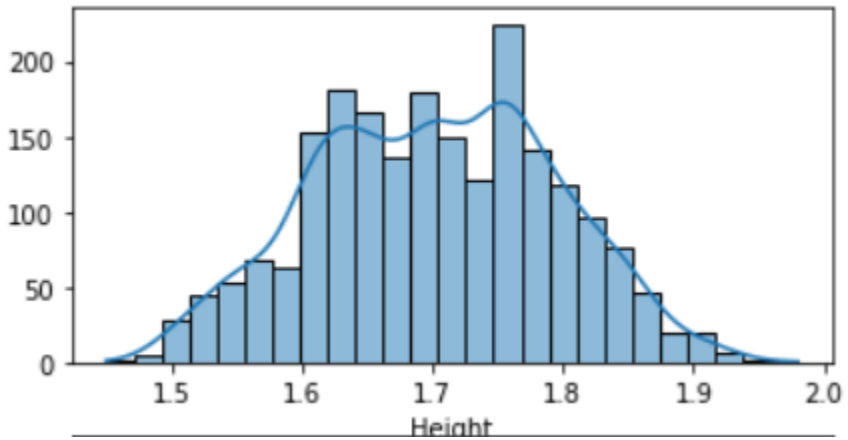
Répartition des individus selon leur sexe, puis selon leur âge

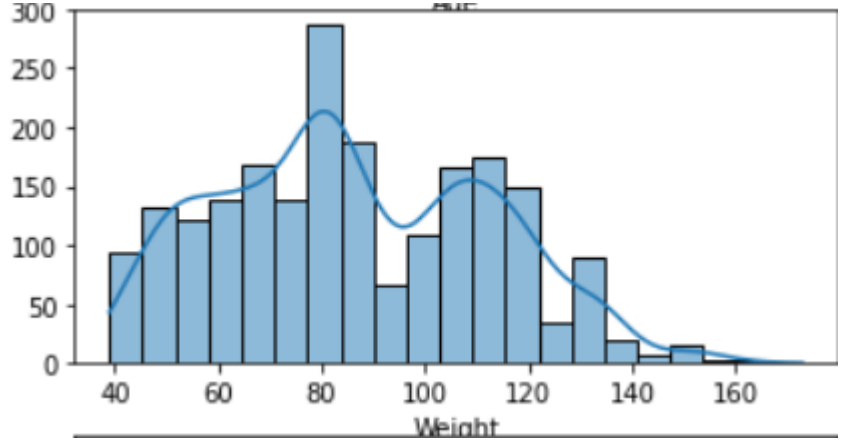




Exploration du dataset

Répartition des individus selon leur taille, puis selon leur poid

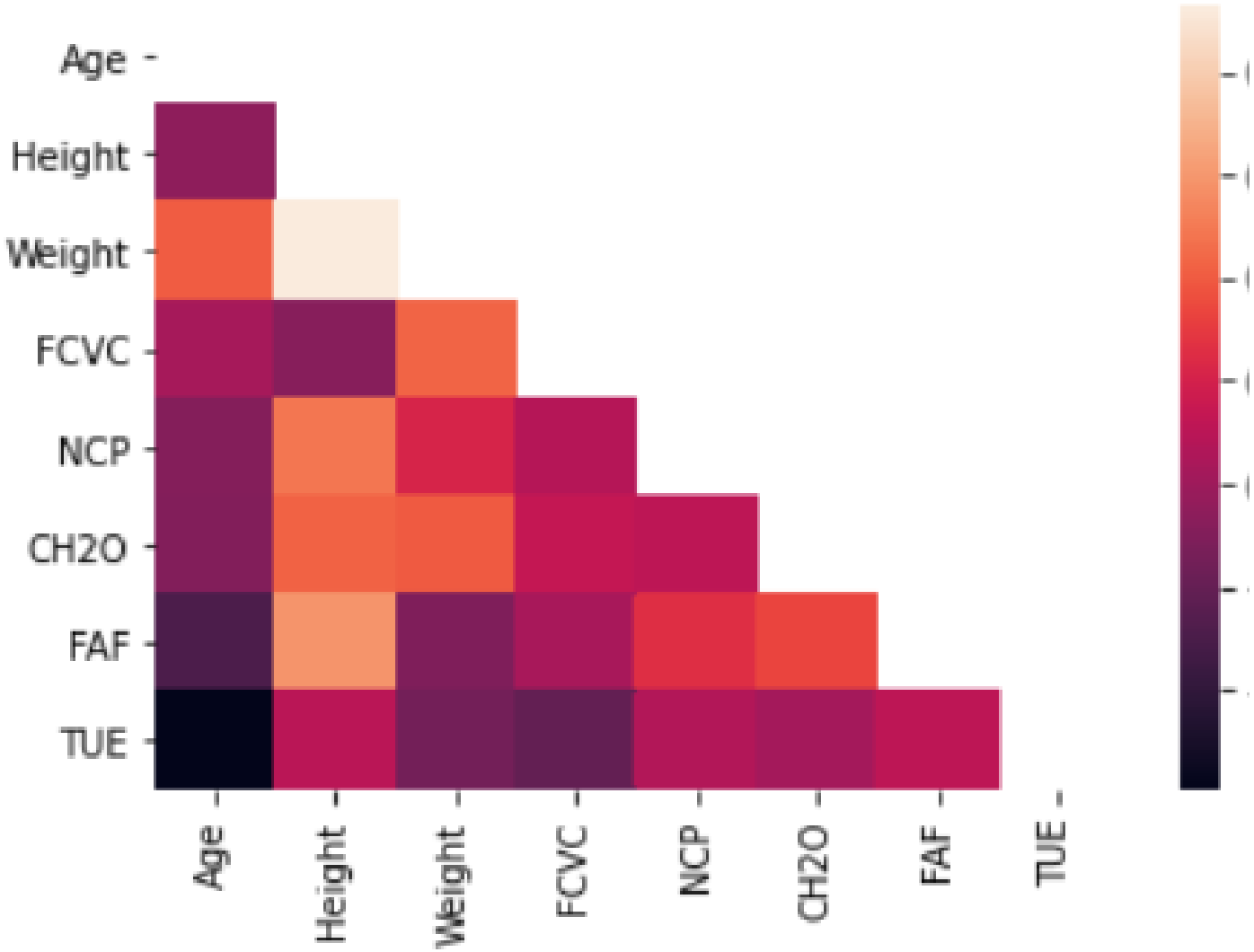




Exploration du dataset

Après avoir analyser les données, j'ai fait une matrice de corrélation des paramètres.

On voit très clairement que l'âge est très peu corrélé au moyen de transport utilisé (TUE). En revanche, le poids et très corrélé a la taille, ce qui était prévisible.



Preprocessing

Il a fallu dans un premier temps transformer toute les données non-numérique.

Par exemple : yes / no est facilement remplaçable par un 0 / 1

```
values = [df[_].unique() for _ in df if isinstance(df[_][0],str) and (_!='NObeyesdad')]
```

```
index =0
for _ in df:
  if isinstance(df[_][0],str) and (_!='NObeyesdad'):
    numbers = [__ for __ in range(0, len(values[index]))]
    df[_].replace(values[index], numbers, inplace=True)
    index+=1
df
```

Ici je remplace chaque data de type string en un nombre issue d'une suite de nombre (0, 1, 2, …) de longueur équivalente au nombre de réponses uniques possible

Preprocessing

Puis j'utilise la méthode de preprocessing présente dans la bibliotheque sklearn :

```
scaler = preprocessing.StandardScaler().fit(x)
scaler.mean_, scaler.scale_
```

```
x_scaled = scaler.transform(x)
x_scaled
```

```
array([[-1.01191369, -0.52212439, -0.87558934, ...,  0.561996
        -1.4191716 , -0.56249143],
       [-1.01191369, -0.52212439, -1.94759928, ..., -1.080624
         0.52115952, -0.56249143],
       [ 0.98822657, -0.20688898,  1.05402854, ...,  0.561996
         2.46149063, -0.56249143],
       ...,
       [-1.01191369, -0.28190933,  0.54167211, ..., -0.019018
         0.52115952, -0.56249143],
       [-1.01191369,  0.00777624,  0.40492652, ..., -0.117991
         0.52115952, -0.56249143],
       [-1.01191369, -0.10211908,  0.39834438, ...,  0.092432
         0.52115952, -0.56249143]])
```

Modèle de classifier

J’ai fais un premier essai avec une random forest

```
model = RandomForestClassifier()
model.fit(x_train, y_train)
prediction = model.predict(x_test)
```

J’ai ensuite fais varier les hyper paramètres dans un but d’amélioration de mes resultats

```
est = [i for i in range(10,200,10)]
parameters = {"n_estimators" : est,
              "criterion" : ["gini", "entropy"],
              "n_jobs"              : [ -1]}
rf = RandomForestClassifier()
grid = GridSearchCV(rf, parameters,scoring='balanced_accuracy', n_jobs=-1)
grid.fit(x_train, y_train)
print (grid.best_score_, grid.best_estimator_)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190]
0.9514602011233322 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=90, n_jobs=-1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

Modèle de classifier

J’ai fais un deuxième essai avec un Gradient Boosting

```
model = GradientBoostingClassifier()
model.fit(x_train, y_train)
gbprediction = model.predict(x_test)
```

J’ai ensuite fais varier les hyper paramètres dans un but d’amélioration de mes resultats

```
gbest = [i for i in range(10,100,10)]
gbparameters = {"n_estimators" : gbest,
                "learning_rate" : [0.01, 0.05, 0.1, 0.5]}
gb = GradientBoostingClassifier()
grid = GridSearchCV(gb, gbparameters,scoring='balanced_accuracy')
grid.fit(x_train, y_train)
print (grid.best_score_, grid.best_estimator_)
```

```
).955730427061549 GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.5, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=80,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```