1. Introduction

Le dataset utilisé pour ce projet contient des caractéristiques extraites d'examens de cardiotocographie (CTG).

La cardiotocographie (CTG) est un test de dépistage couramment utilisé qui consiste en un bref enregistrement (environ 20 minutes) de la fréquence cardiaque fœtale et de l'activité utérine, qui est réalisé lors de l'admission de la mère en salle de travail.

Ces examens ont été classés selon 3 types : Normal, Suspect et Pathologique.
L'objectif de ce TP est de mettre en lumière les tendances de données et d'isoler les facteurs clefs permettant de réaliser des prédictions sur l'état de santé du fœtus. Il faudra que nous créions 3 classifiers afin de déterminer la valeur dans la colonne "fetal_health" à partir des autres colonnes.

2. Analyse de la base de données

a) Description de la base de données

Le dataset fournit contient 2126 lignes et 22 caractéristiques extraites d'examens de cardiotocographie (CTG).

Nous importons, dans un premier temps, les bibliothèques qui vont être nécessaires au traitement de la base de données :

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, recall_score
from sklearn.tree import DecisionTreeClassifier
```

Certaines caractéristiques ne possèdent pas de valeur pour chacune des lignes, pour chacun des foetus. Il faut donc adapter la base de données pour tenir compte du déséquilibre.
Nous allons donc commencer par importer les données, les "nettoyer", les traiter, les mettre à l'échelle et les ajuster.

Dans un premier temps, nous chargeons ensuite puis affichons le tableau afin de nous assurer qu'il n'y ait pas d'erreur ou de données manquantes.

```
# lecture du fichier de données
data = pd.read_csv("fetal_health.csv")
# première visualisation des colonnes et des datas présentes
print (f'colonnes : \n {data.columns}')
data.head()
```

Affichage :

```
colonnes :
 Index(['baseline value', 'accelerations', 'fetal_movement',
       'uterine_contractions', 'light_decelerations',
'severe_decelerations',
       'prolongued_decelerations', 'abnormal_short_term_variability',
       'mean_value_of_short_term_variability',
       'percentage_of_time_with_abnormal_long_term_variability',
       'mean_value_of_long_term_variability', 'histogram_width',
       'histogram_min', 'histogram_max', 'histogram_number_of_peaks',
       'histogram_number_of_zeroes', 'histogram_mode', 'histogram_mean',
       'histogram_median', 'histogram_variance', 'histogram_tendency',
```

```
       'fetal_health'],
      dtype='object')
```

```
# alignement des colonnes
data.describe().T
```

La fonction describe() permet d'obtenir des informations supplémentaires concernant la base de données telles que : le nombre de données, la moyenne de chacune des caractéristiques

Nous pouvons voir ci-dessous, les 22 colonnes ainsi que leur correspondance:

baseline value : fréquence cardiaque du foetus de base (nombre de battements par minute)
accelerations : nombre de mouvements foetaux par seconde
fetal_movement : nombre de mouvements foetaux par seconde
uterine_contractions : nombre de contractions utérines par seconde
light_decelerations : nombre de décélérations légères par seconde
severe_decelerations : nombre de décélérations sévères par seconde
prolongued_decelerations : nombre de décélérations prolongées par seconde
abnormal_short_term_variability : pourcentage de temps avec une variabilité à court terme anormale
mean_value_of_short_term_variability : valeur moyenne de la variabilité à court terme
percentage_of_time_with_abnormal_long_term_variability : pourcentage de temps avec une variabilité anormale à long terme
mean_value_of_long_term_variability : valeur moyenne de la variabilité à long terme
histogram_width : largeur de l'histogramme du RCF (Rythme Cardiaque du Foetus) généré à partir de l'examen
histogram_min : minimum de l'histogramme du RCF généré à partir de l'examen
histogram_max : maximum de l'histogramme du RCF généré à partir de l'examen
histogram_number_of_peaks : nombre de pics de l'histogramme du RCF généré à partir de l'examen
histogram_number_of_zeroes : nombre de zéros de l'histogramme du RCF généré à partir de l'examen
histogram_mode : mode de l'histogramme du RCF généré à partir de l'examen
histogram_mean : moyenne de l'histogramme du RCF généré à partir de l'examen
histogram_median : médiane de l'histogramme du RCF généré à partir de l'examen
histogram_variance : variance de l'histogramme du RCF généré à partir de l'examen
histogram_tendency : tendance de l'histogramme du RCF généré à partir de l'examen
fetal_health : santé du foetus évaluée par le professionnel de santé (Désignée par différents chiffres : 1=Normal, 2=Suspect et 3=Pathologique)

b) Relations entre caractéristiques

Après avoir affiché les données nous vérifions s'il y a des données manquantes ou nulles dans le tableau grâce à la commande :

```
data.info()
```

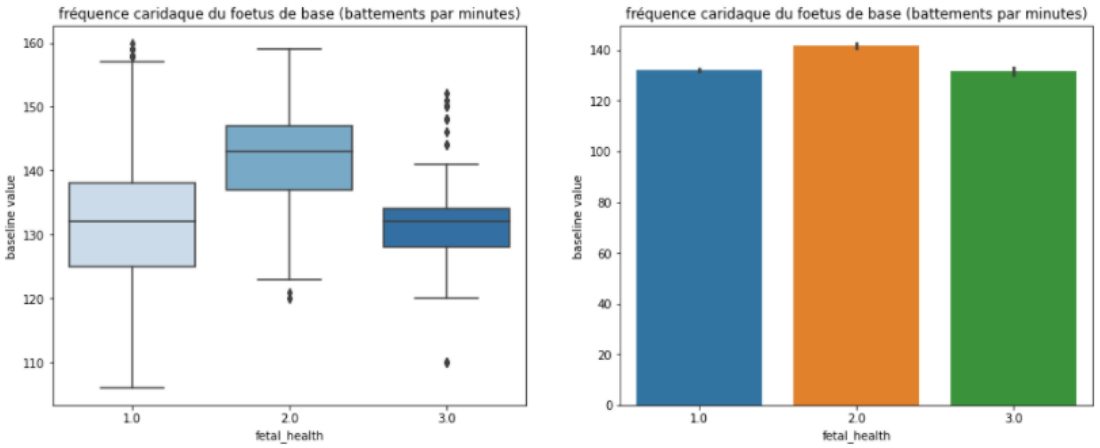
Nous observons que nous avons des données pour chacune des caractéristiques.

● Nous allons ensuite tracer des graphiques mettant en relation le “fetal_heath” et 3 des différentes caractéristiques données choisie de façon aléatoire.

Dans un premier temps nous cherchons à mettre en relation le “fetal_health” avec la fréquence cardiaque de base à l’aide du code suivant :

```
fetal = data['fetal_health']
plt.figure(figsize=(16, 6))
plt.subplot(121)
sns.boxplot(x = fetal, y = 'baseline value', data = data, palette = 'Blues')
plt.title('fréquence cardiaque du foetus de base (battements par minute)')
plt.subplot(122)
sns.barplot(x = fetal, y = 'baseline value', data = data)
plt.title('fréquence cardiaque du foetus de base (battements par minute)')
```

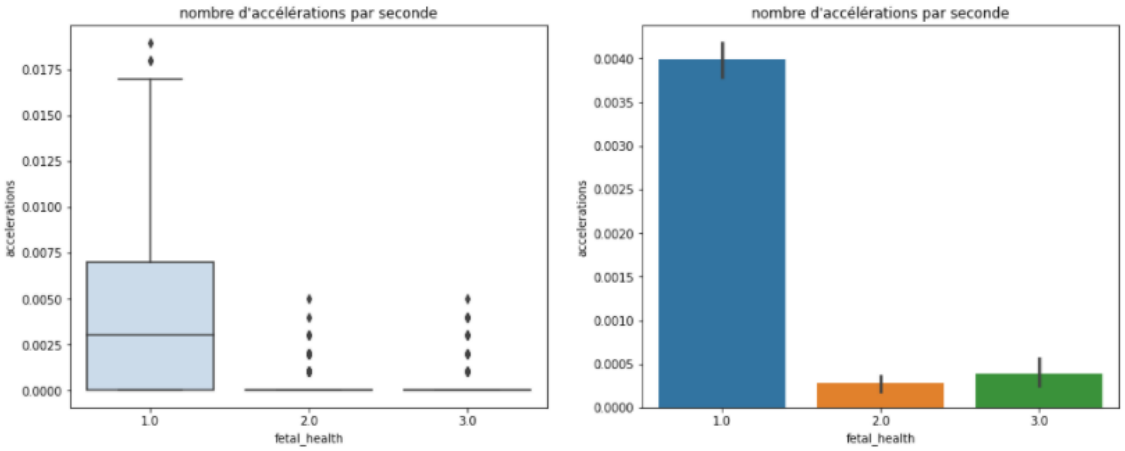
Nous obtenons ces deux graphiques :



Ensuite nous cherchons à mettre en relation le “fetal_health” avec l'accélération du coeur du foetus grâce au code suivant :

```
plt.figure(figsize=(16, 6))
plt.subplot(121)
sns.boxplot(x = fetal, y = 'accélérations', data = data, palette = 'Blues')
plt.title("nombre d'accélérations par seconde")
plt.subplot(122)
sns.barplot(x = fetal, y = 'accélérations', data = data)
plt.title("nombre d'accélérations par seconde")
```

Nous obtenons les graphiques suivants :

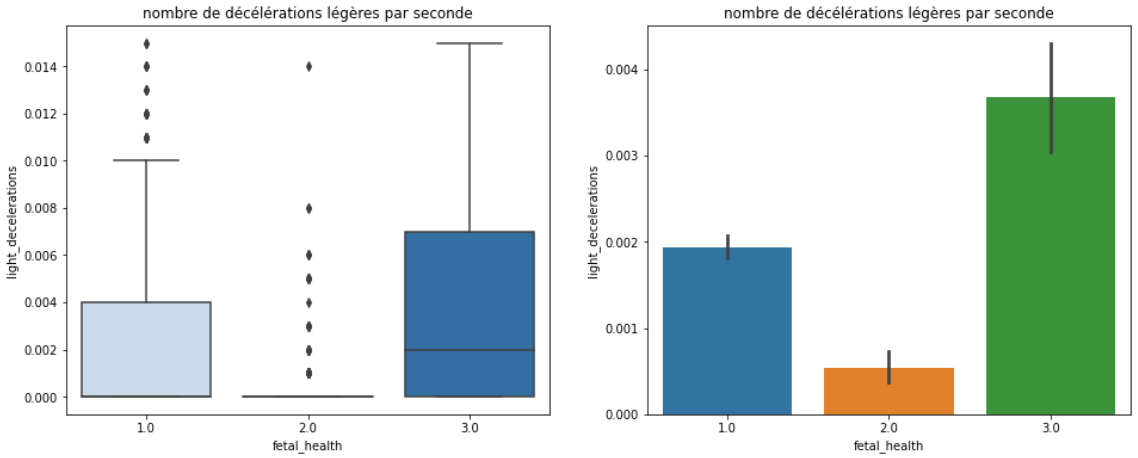


Ici, nous cherchons à mettre en relation “fetal_health” avec le nombre de décélérations légères par seconde :

```
data.info()
```

```
plt.figure(figsize=(16, 6))
plt.subplot(121)
sns.boxplot(x = fetal, y = 'light_decelerations', data = data, palette =
'Blues')
plt.title('nombre de décélérations légères par seconde')
plt.subplot(122)
sns.barplot(x = fetal, y = 'light_decelerations', data = data)
plt.title('nombre de décélérations légères par seconde')
```

Ce code nous permet d’obtenir les graphiques suivants :



Nous cherchons le nombre de chaque statut de “fetal_health” (1,2 ou 3) dans ces données pour faciliter la visualisation.
Si nous voulons considérer qu’un facteur est intéressant à étudier, il faut idéalement que 2 des histogrammes soient notablement inférieurs au troisième.
Dans le premier graphique, les 3 histogrammes sont très élevés et nous n’observons pas de différence notable entre eux. Nous pouvons donc en conclure que la caractéristique de fréquence cardiaque du fœtus n'est pas un facteur nous permettant d’analyser la santé du fœtus.

Le deuxième graphe est le plus simple à lire. En effet, il montre que la plupart des fœtus normaux ont une accélération normale. Les autres états de “fetal_health” (suspect et pathologique) ont une accélération considérablement faible. Nous en déduisons que l’accélération nous donne une information sur l’état du fœtus.

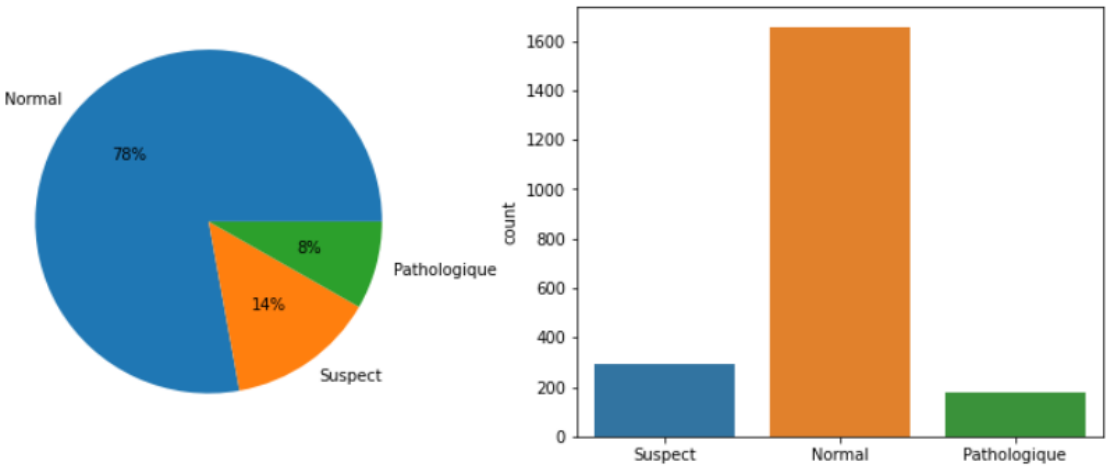
Concernant le troisième graphique, nous observons une différence notable entre les patients pathologiques et les patients suspects. La décélération légère nous permet donc de bien dissocier les fœtus suspects et pathologiques. Cependant, la différence n’est pas tranchée entre les fœtus normaux et pathologiques, il est moins évident de les analyser mais nous devons pouvoir en tirer des informations quand même.

c) Proportions de chacun des états de santé des foetus

Nous regardons ici les proportions entre les cas de foetus : normaux, suspects et pathologiques.

```
d1, d2, d3 = data['fetal_health'].value_counts()
plt.figure(figsize= (13, 5))
plt.subplot(121)
plt.pie([d1, d2, d3], labels = ['Normal', 'Suspect', 'Pathologique'],
autopct='%1.00f%%')
print(data['fetal_health'].value_counts())
Statut = {1: 'Normal',
          2: 'Suspect',
          3: 'Pathologique'}
fetal = [Statut[i] for i in data['fetal_health']]
plt.subplot(122)
sns.countplot(fetal)
```

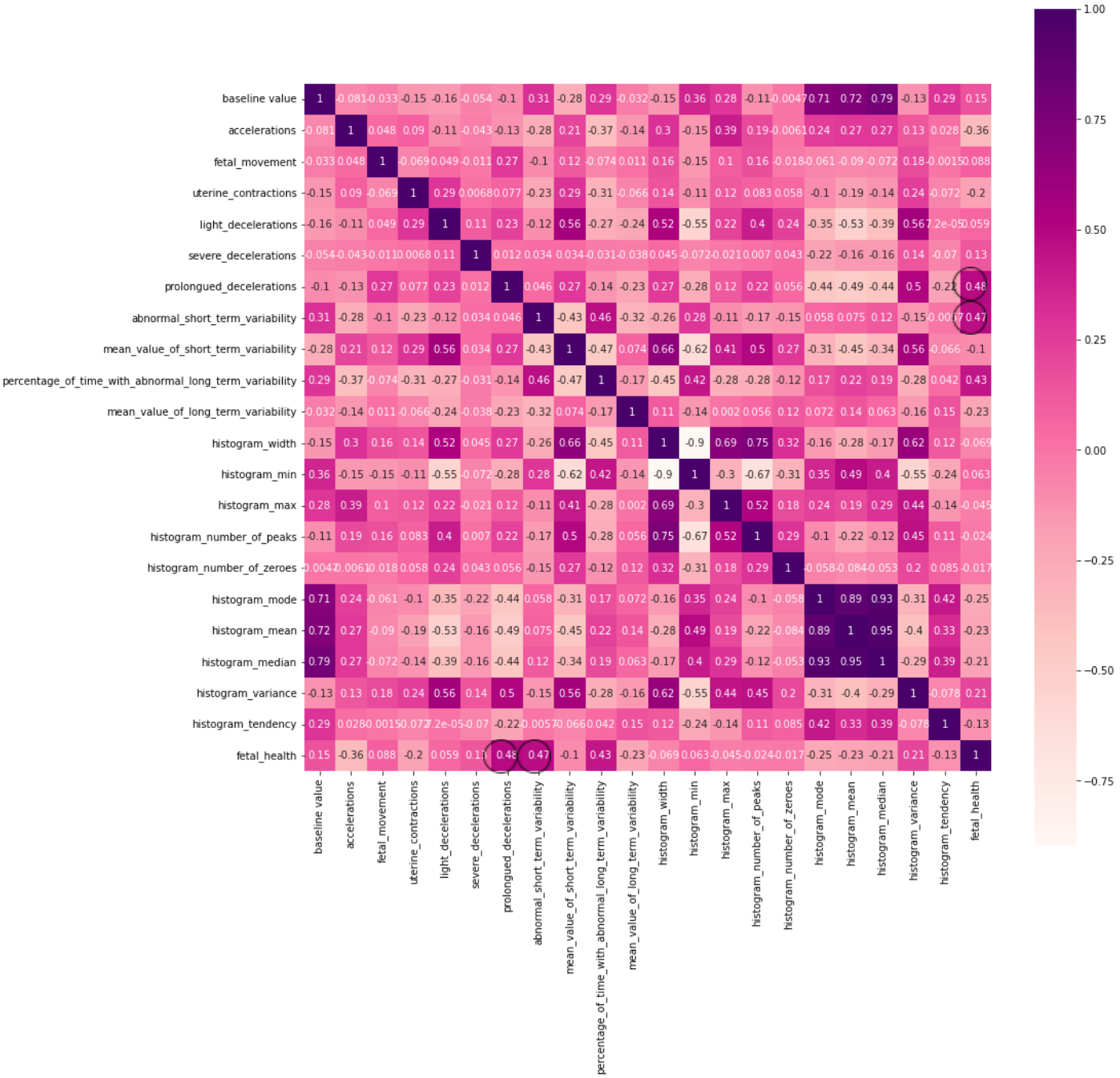
Nous obtenons les graphiques suivants :



d) Identification de corrélation entre la santé du foetus et les caractéristiques mesurés

```
# corrélation entre chaque caractéristiques
# on regarde les chances d'avoir une relation par rapport à une caractéristiq
```

```
puis on le fait pour chaque caractéristique
correlation = data.corr()
sort_correlation = correlation.sort_values(by = ['fetal_health'], ascending =
False).head(10)
plt.figure(figsize = (15, 15))
sns.heatmap(data.corr(), annot = True, cmap = 'RdPu', cbar = True, square = T
print(sort_correlation)
```

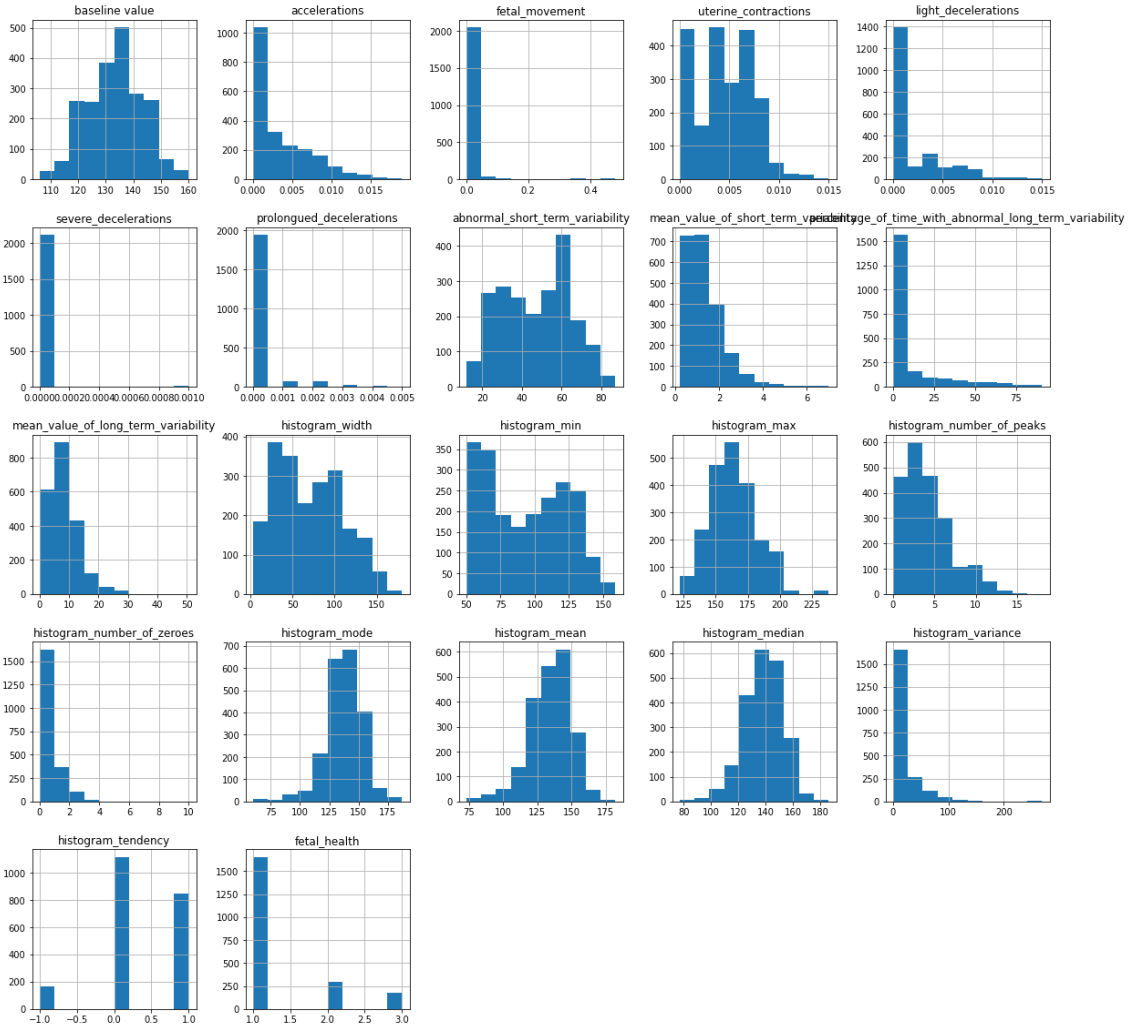


Nous cherchons désormais une corrélation entre la santé du fœtus et d'autres caractéristiques physiologiques. Cette matrice de confusion a pour avantage ici de nous aider à identifier visuellement les informations qui pourraient nous être utiles. Cependant, cela nous est davantage utile lorsque nous

disposons de moins de données. En effet, dans notre cas avec 21 inputs cela n'est pas véritablement pratique.
Néanmoins, en regardant avec précision, nous pouvons identifier une forte corrélation entre "fetal_health" et "prolongued_decelarations" avec un indice s'élevant à 0.485.
La variabilité à long terme est ici bien anormale ce qui nous apporte ainsi des informations sur l'indice de santé du fœtus.

e) Distribution des données

Pour chaque facteur, nous réalisons un histogramme afin de visualiser la distribution des données.



Ces histogrammes nous permettent de visualiser l'asymétrie de la distribution des variables. Le coefficient d'asymétrie correspond à une mesure de l'asymétrie de la distribution d'une variable aléatoire réelle. Un coefficient nul indique une distribution symétrique, un coefficient négatif indique une distribution décalée vers la droite et un coefficient positif indique une distribution décalée vers la gauche.
Parmi ces histogrammes, nous pouvons par exemple nous intéresser à "accélérations". En effet, tout comme pour "foetal_movement", son maximum est situé à gauche cela est représentatif d'un coefficient d'asymétrie positif.
Une distribution est dite symétrique si les valeurs se répartissent de manière uniforme autour de la moyenne, le mod et la médiane. "Baseline value" à ici une distribution symétrique.
Lorsque l'on étudie un histogramme, il faut être prudent et vérifier le nombre de valeurs étudiées. Plus le nombre de valeurs est grand et plus l'étude sera fiable. Ici, si nous regardons "severe_decelerations"

et que nous cherchons à connaître le nombre de valeurs, nous nous apercevons qu'il n'y en a que sept grâce à la commande suivante : data.severe_decelerations.value_counts(). Le résultat est alors peu fiable.
Selon des recherches internet, il est normal d'avoir beaucoup de variabilité pour l'histogramme "abnormal_short_term_variability" cependant, le résultat parait étonnant vu que nous savons que la majorité des foetus sont normaux.

f) Normalisation des données

Nous avons dû réécrire les colonnes dans une liste de strings, afin de pouvoir faire un dataframe après.

```
colonnes = ['baseline value', 'accelerations', 'fetal_movement',
'uterine_contractions', 'light_deceleration', 'severe_deceleration',
'prolongued_decelerations', 'abnormal_short_term_variability',
'mean_value_of_short_term_varibility',
'percentage_of_time_with_abnormal_long_term_variability',
'mean_value_of_lonh_term_variability', 'histogram_width', 'histogram_min',
'histogram_max', 'histogram_number_of_peaks', 'histogram_number_of_zeroes',
'histogram_mode', 'histogram_mean', 'histogram_median', 'histogram_variance',
'histogram_tendency']
standard_scaler = StandardScaler()
x = standard_scaler.fit_transform(data.drop(['fetal_health'], axis = 1))
data_nor = pd.DataFrame(x, columns = colonnes)
data_nor.head()
```

3. Explication de l'algorithme du classifier

On va pouvoir ensuite diviser les données entre celles de *training* et celles de *test* pour entraîner le modèle et le tester avec les données de test.

On divise les valeurs d'entraînement et les valeurs à tester en sous-ensembles grâce à la fonction : *train_test_split*

La fonction prend un ensemble de données en entrée et permet d'avoir en sortie l'ensemble de données divisé en deux sous-ensembles.
On divise notre ensemble de données en colonnes d'entrée (X) et de sortie (Y), on appelle ensuite la fonction *train_test_split* en passant les deux tableaux.

➢ test_size (compris entre 0 et 1) : représente la proportion de l'ensemble de données à inclure dans la division de test.
➢ random_state : contrôle le brassage appliqué aux données avant d'appliquer le fractionnement.

Etant donné que nous n'avons pas un nombre équilibré de données sur chacune des étiquettes, il nous fallait une solution afin de diviser l'ensemble des données en ensemble d'apprentissage et de test qui préserve les mêmes proportions d'exemples dans chaque étiquettes. Cela est appelé un fractionnement stratifié train-test.
Pour le réaliser, nous donnons à l'argument "stratify" l'ensemble des données d'origine. Grâce à cela, les sous-ensembles ont la même proportion d'exemples dans chaque classe que celle présente dans la base de données d'origine.

a) Classifier 1

Nous utilisons dans un premier temps la régression linéaire pour faire une prédiction de l'état du fœtus.

L'objectif est de trouver une fonction de prédiction décrivant la relation entre les données de test et d'entraînement. C'est à dire qu'à partir des valeurs connues (les valeurs d'entraînement) nous arrivons à donner une prédiction des valeurs de test. Nous utilisons ensuite les données de test pour mesurer la précision de la prédiction du modèle.

Pour se faire, on a d'abord créé une instance grâce à la fonction LogisticRegression().
Ensuite, nous utilisons la *logistic_regression.fit(X_train, Y_train)* afin d'entraîner le modèle, c'est à dire que le modèle identifie les relations entre les données d'entraînement d'entrée (X_train) et de sortie (Y_train) afin de comprendre le comportement des données.

Nous utilisons ensuite les informations apprises par le modèle pendant l'étape d'avant dans le but de les utiliser pour prédire les données de test avec la commande : *logistic_regression_mod.predict(X_test)*

Enfin, nous mesurons les performances du modèle = $\frac{Nombre\ de\ prédictions\ correctes}{Nombre\ de\ données\ \ total}$ grâce à : *logistic_regression_mod.score(X_test, Y_test)*

```
target = data['fetal_health']
X_train, X_test, Y_train, Y_test = train_test_split(data_nor, target,
test_size = 0.4, random_state = 42, stratify = target)
logistic_regression = LogisticRegression()
logistic_regression_mod = logistic_regression.fit(X_train, Y_train)
logistic_regression_mod.predict(X_test)
logistic_regression_mod.score(X_test, Y_test)
```

b) Classifier 2

Nous allons à présent continuer avec une classification vectorielle.
" sklearn.SVC " est très utilisé dans la classification et la prédiction. C'est ce que l'on peut appeler une "machine à vecteurs de support séparateurs à vaste marge". Il s'agit d'un ensemble de techniques d'apprentissage supervisé, destiné à résoudre des problèmes de discrimination et de régression.

Exemple :
Prenons l'exemple où nous concevons un graphique dans lequel nous plaçons des points avec des valeurs propres et que nous souhaitons répartir dans des zones différentes grâce à leur similarité.
C'est par l'intermédiaire d'un entraînement que nous parviendrons à répartir nos points dans des zones différentes.
Nous allons ainsi pouvoir créer un graphe avec un autant de dimensions qu'il y a de caractéristiques à étudier. Dans notre cas, nous aurons 21 caractéristiques qui nous permettront de trouver θS. Nous aurons donc 21 dimensions dans notre graphe.
Durant le test, nous avons ainsi pu regarder dans quelle région se sont retrouvés nos points (nos données).

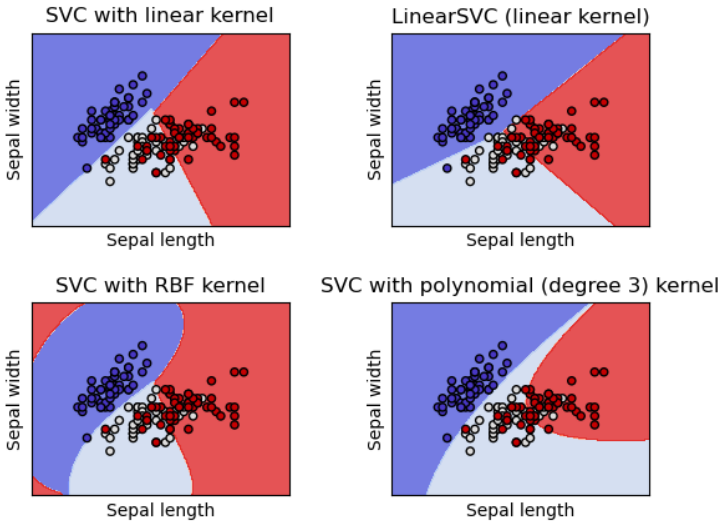


Illustration de la création de zones de similarité en fonction des particularités de chaque point..

c) Classifier 3

Pour ce dernier classifier, nous commençons par utiliser DecisionTreeClassifier afin de réaliser une classification multi-classe à l'aide d'un arbre de décision.
Nous choisissons également quelques paramètres tels que la profondeur maximale de l'arbre, le type de critère de fractionnement, le poids de la classe ou encore le caractère aléatoire de l'estimateur.
Pour le type de critère de fractionnement, nous avons choisi “gini”. Cela permet de mesurer la fréquence à laquelle tout élément de l'ensemble de données sera mal étiqueté lorsqu'il est étiqueté de manière aléatoire.
Pour le poids de la classe, nous avons mis le paramètre “balanced”. Il permet de répliquer la plus petite classe jusqu'à ce que son nombre d'échantillons soit égal à celui de la plus grande. Cela est utile car nous n’avons pas la même quantité d’informations pour chaque donnée.

Une fois l’arbre de décision créé, nous entraînons l’arbre de décision puis prédisons la réponse pour l’ensemble des données de test.

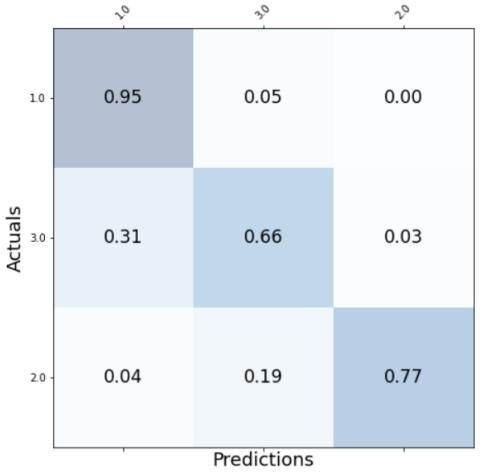
```
#On crée l'arbre de décision
tree = DecisionTreeClassifier(criterion = 'gini', max_depth = 5, class_weight
= 'balanced', random_state = 42)
#On entraîne l'arbre de décision
tree.fit(X_train, Y_train)
#Prédit la réponse pour l'ensemble de données de test
predict = tree.predict(X_test)
print(classification_report(Y_test, predict))
print('Résultat classification : ', recall_score(Y_test, predict, average =
'micro'))
```

4. Résultats pour chaque classifier

a) Classifier 1

Concernant le classifier 1, nous obtenons **89,77%** de réussite.
Au travers de l'analyse de la matrice de confusion nous pouvons voir que le classifier 1 permet :

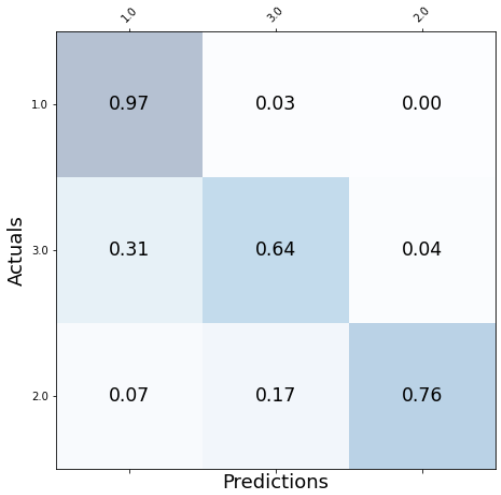
- D'identifier avec un taux de prédiction correct de 95% concernant les fœtus normaux (5% de faux positifs confondus avec des fœtus pathologiques).
- D'identifier avec un taux de prédiction correct de 66% concernant les fœtus pathologiques (34% de faux positifs : 31% confondus avec des fœtus normaux et 3% avec des fœtus suspects).
- D'identifier avec un taux de prédiction correct de 77% concernant les fœtus suspects (23% de faux positifs : 4% confondus avec des fœtus normaux et 19% avec des fœtus pathologiques).



b) Classifier 2

Concernant le classifier 2, nous obtenons **90,7%** de réussite.
Au travers de l'analyse de la matrice de confusion nous pouvons voir que le classifier 2 permet :

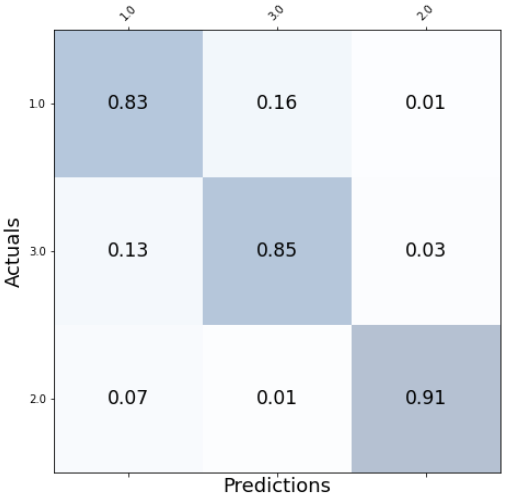
- D'identifier avec un taux de prédiction correct de 97% concernant les fœtus normaux (3% confondus des fœtus pathologiques).
- D'identifier avec un taux de prédiction correct de 64% concernant les fœtus pathologiques (35% de faux positifs : 31% confondus avec des fœtus normaux et 4% avec des fœtus suspects).
- D'identifier avec un taux de prédiction correct de 76% concernant les fœtus suspects (24% de faux positifs : confusion de 7% des cas avec des fœtus normaux et de 17% des cas avec des fœtus pathologiques).



c) Classifier 3

Concernant le classifier 3, nous obtenons **84,25%** de réussite.
Au travers de l'analyse de la matrice de confusion nous pouvons voir que le classifier 3 permet :

- D'identifier avec un taux de prédiction correct de 83% concernant les fœtus normaux (17% de faux positifs : confusion de 16% des cas avec des fœtus pathologiques et 1% des cas avec des fœtus suspects).
- D'identifier avec un taux de prédiction correct de 85% concernant les fœtus pathologiques (16% de faux positifs : confusion de 13% des cas avec des fœtus normaux et de 3% des cas avec des fœtus suspects).
- D'identifier avec un taux de prédiction correct de 91% concernant les fœtus suspects (8% de faux positifs : confusion de 7% des cas avec des fœtus normaux et de 1% des cas avec des fœtus pathologiques).



5. Conclusion

Pour conclure, nous avons pu voir que les taux de réussite dépendent des méthodes de traitement utilisées. Certaines sont davantages adaptées à pour un grand nombre de données, quand d'autres sont plus sensibles à l'aspect temporel. Nous observons que, globalement, les foetus normaux ont le meilleur taux de "true positive". Le classifier n°3 est le classifier le plus équilibré, en effet, le classifier utilisant un arbre de décision permet d'identifier plusieurs classes de façon assez précise. La classifier n°2 nous permet d'obtenir le meilleur taux de réussite, or, son taux de valeurs "true positive" est mal réparti. En effet, il possède un taux de réussite élevé, particulièrement sur les cas normaux qui représentent 78% des cas. C'est pourquoi nous pouvons dire que la troisième méthode (classifier n°3) est la plus fiable concernant l'ensemble des cas.

Durant ce TP, le principal problème que nous avons rencontré est la non-uniformité des données. En effet, chacune des informations ne possédait pas le même nombre de données.

Aujourd'hui dans le domaine de la santé, les capteurs sont de plus en plus présents et les données qu'ils génèrent peuvent être colossales. Cependant, cette masse précieuse d'informations doit-être triée et analysée. Ce TP, nous a permis de prendre toute la mesure de la nécessité de comprendre et de mettre en pratique notre capacité à gérer cette grande quantité d'informations, avec les notions importantes propres à la santé.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6317385/

revue scientifique prouvant nos résultats

Références

- https://medium.com/analytics-vidhya/classification-fetal-health-bd578beae25