Université cadi ayyad

Faculté des Sciences Semlalia
Marrakech

RAPPORT SUCCINCT DE DEVOIR

module : Big Data Analytics

réalisé par : hassan ait baha

contexte:

La prédiction du covid19 est importante pour atténuer sa propagation pour cela dans ce devoir on va utiliser un solution qui nous permet de prévoir du Covid 19 à partir d'un ensemble de paramètre vitaux en basant sur des données dans le but c'est est la classification, c'est-à-dire: étant donné un ensemble de données d'entrée avec des labels de classe (Covid ou non Covid)

Dataset :

Le dataset à utiliser est une collection des données qui contient plusieurs nombres des colonnes qui ne sont pas filtrées et non structurees et qui contient des NAN alors on doit utiliser des technique de filtrage des datasets pour les mettre structurées pour appliquer les algorithmes de machine learning.

tout d'abord l'importation des données

```
import pandas as pd
data = pd.read_csv("data1.csv")
print(data.shape)
data.head()
```

```
(950, 30)
```

	patientid	offset	sex	age	finding	RT_PCR_positive	survival	intubated	intubation_present	went_icu	...	date	location	folder	filename	doi	
0	2	0.0	M	65.0	Pneumonia/Viral/COVID-19	Y	Y	N	N	N	...	January 22, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-a-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	http
1	2	3.0	M	65.0	Pneumonia/Viral/COVID-19	Y	Y	N	N	N	...	January 25, 2020	Cho Ray Hospital, Ho Chi Minh City, Vietnam	images	auntminnie-b-2020_01_28_23_51_6665_2020_01_28_...	10.1056/nejmc2001272	http

on voit que notre dataset contient 30 colonnes alors on doit extraire seulement les colonnes qui on besoin pour faire la prédiction qui sont :

- ➔ **L'âge**
- ➔ **le sexe**
- ➔ **La saturation en oxygène SPO2**

➔ **la Température**

```
df=data[['sex','age','temperature','pO2_saturation','finding']]
df.head()
print(df)
```

```
    sex   age  temperature  pO2_saturation                  finding
0     M  65.0          NaN             NaN  Pneumonia/Viral/COVID-19
1     M  65.0          NaN             NaN  Pneumonia/Viral/COVID-19
2     M  65.0          NaN             NaN  Pneumonia/Viral/COVID-19
3     M  65.0          NaN             NaN  Pneumonia/Viral/COVID-19
4     F  52.0          NaN             NaN  Pneumonia/Viral/COVID-19
..   ..   ...          ...             ...                       ...
```

puis on calcule les nombre des NAN pour les remplacer dans ce cas j'utilise la méthode fillI , Lorsque ffill est appliqué sur l'axe des colonnes, les valeurs manquantes sont remplies par la valeur de la colonne précédente de la même ligne.

```
nbr=df['temperature'].isna().sum()
print(nbr)
nbr2=nbr=df['pO2_saturation'].isna().sum()
print(nbr2)
```

```
872
831
```

```
df['temperature'].fillna(method='ffill',inplace=True)
df['pO2_saturation'].fillna(method='ffill',inplace=True)
```

On regarde que le nombre des NAN est grand alors on doit remplacer les NAN avec des valeurs dans ce cas j'utilise la méthode fillI.

puis on supprime des NAN Pour s'assurer qu'il n'y a pas

Pour s'assurer qu'il n'y a pas au but d'assurer qu'il n'y a pas des NAN dans notre dataset

```
df.isnull().sum()
```

```
sex              0
age              0
temperature      0
pO2_saturation   0
finding          0
dtype: int64
```

puis on a l'attribut sex et finding sont de type object alors il faut les changer pour être utilisé et compatible avec les algorithmes de classification

```
df['sex'] = df['sex'].apply({'M':0, 'F':1}.get)
df["finding"] = df["finding"].apply(lambda val: 1 if val.find('COVID-19') != -1 else 0)
```

puis on sélectionne les attributs d'entrées et l'attribut cible (target)

features : sexe , age , La saturation en oxygène SPO2 , la Température

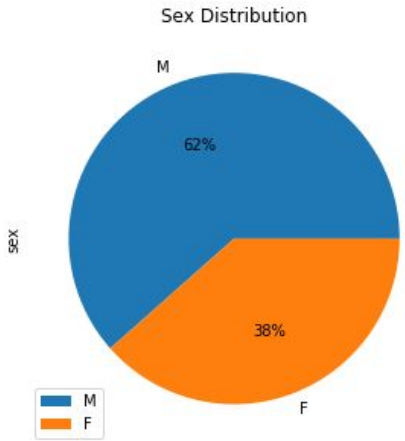
target : finding

```
X = df.iloc[:, :-1].values
y = df.iloc[:, 4].values
```

ploting:

➔ la distribution de sex

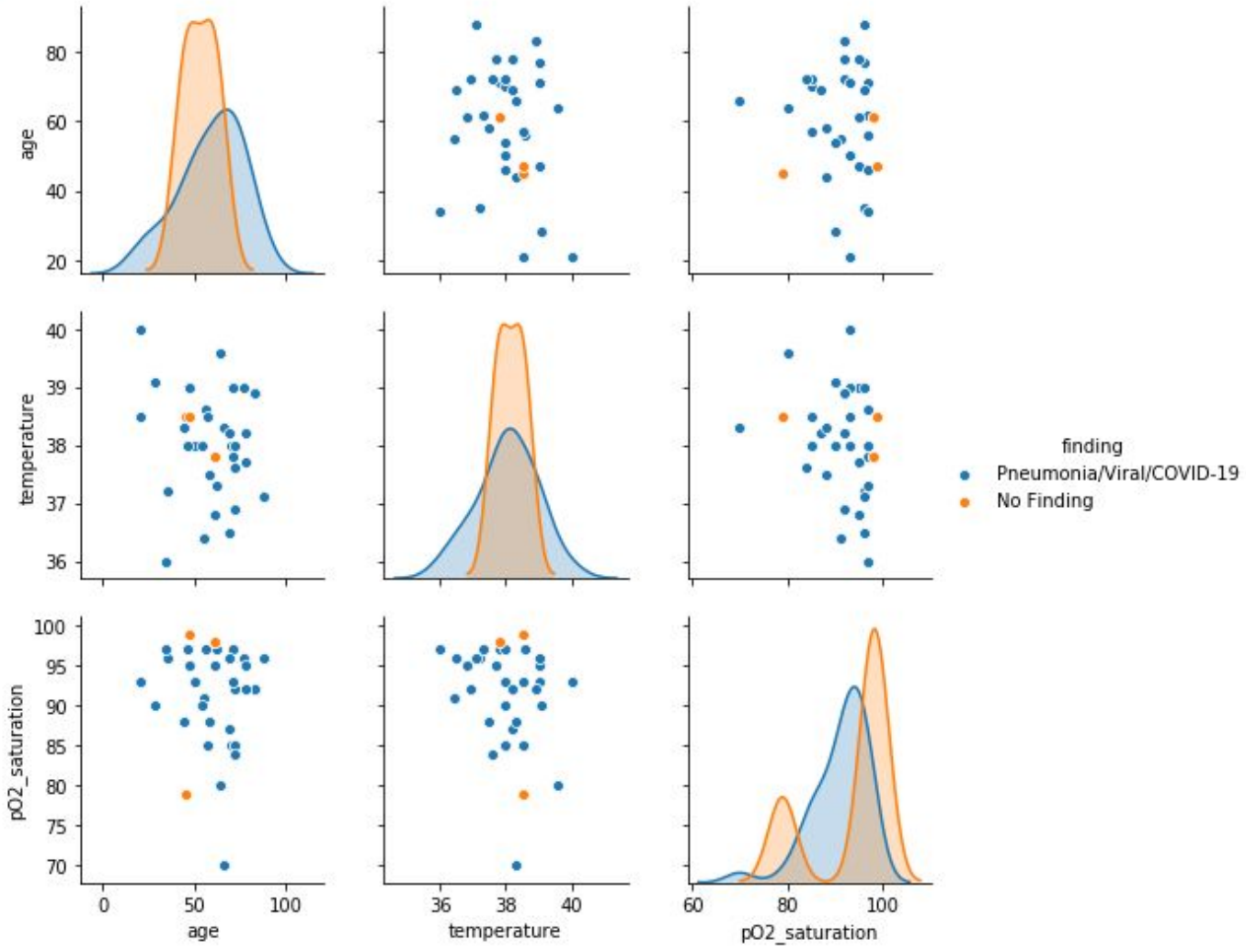
```
import matplotlib.pyplot as plt

sexd = df['sex'].value_counts().plot.pie(y='sex', legend = True, autopct='%2.0f%%', figsize = (5,5), title = 'Sex Distribution')
```



➔ la distribution de deux classe de target

```
sns.pairplot(df,hue='finding')
```

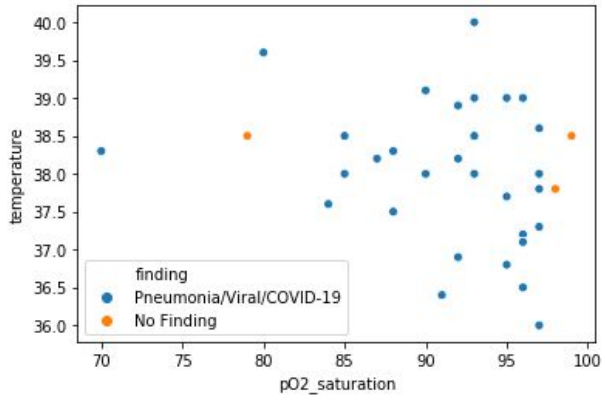
```
<seaborn.axisgrid.PairGrid at 0x14da89349c8>
```



➔ les cas de covid par température et po2_saturation

```
sns.scatterplot(data=df[["pO2_saturation", "temperature", "finding"]], x="pO2_saturation", y="temperature", hue="finding")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x14da75ac148>
```



les algorithmes de classification

➢ K-Nearest Neighbors

L'algorithme K-NN (K-nearest neighbors) est une méthode d'apprentissage supervisé. Il peut être utilisé aussi bien pour la régression que pour la classification. Son fonctionnement peut être assimilé à l'analogie suivante *"dis moi qui sont tes voisins, je te dirais qui tu es…"*.

Train Test Split :

```
from sklearn.model_selection import train_test_split
X_training, X_testing, y_training, y_testing = train_test_split(X, y, test_size=0.10)
```

Pour éviter le overfitting , nous allons diviser notre ensemble de données en fractionnements d'entraînement et de test, ce qui nous donne une meilleure idée de la performance de notre algorithme pendant la phase de test.

puis pour le training et prédiction consiste à importer la classe KNeighborsClassifier à partir de la bibliothèque sklearn.neighbors en choisissant une valeur pour le `n_neigbours`

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_training, y_training)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred = classifier.predict(X_testing)
```

puis en évaluer notre algorithme avec la matrice de confusion et leur accuracy

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_testing, y_pred))
print(classification_report(y_testing, y_pred))
```

```
[[20  6]
 [11 31]]
              precision    recall  f1-score   support

           0       0.65      0.77      0.70        26
           1       0.84      0.74      0.78        42

    accuracy                           0.75        68
   macro avg       0.74      0.75      0.74        68
weighted avg       0.76      0.75      0.75        68
```

➢ decision tree

Les arbres de décision (AD) sont une catégorie d'arbres utilisée dans l'exploration de données et en informatique décisionnelle. Ils emploient une représentation hiérarchique de la structure des données sous forme des séquences de décisions (tests) en vue de la prédiction d'un résultat ou d'une class

Train Test Split :

on utilise les mêmes paramètre de train et split pour tous les algorithmes

On entraîne l'arbre:

```
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_training,y_training)

#Predict the response for test dataset
y_pred = clf.predict(X_testing)
```

puis on regarde l'évaluation de cette algorithme

```
print("Accuracy:",metrics.accuracy_score(y_testing, y_pred))
```

présentation d'arbre:

```
from sklearn import tree
presentation = tree.export_text(clf)
print(presentation)
```

```
|--- feature_3 <= 87.65
|   |--- feature_2 <= 37.95
|   |   |--- class: 1
|   |--- feature_2 >  37.95
|   |   |--- feature_3 <= 55.50
|   |   |   |--- feature_2 <= 39.05
|   |   |   |   |--- feature_1 <= 20.50
|   |   |   |   |   |--- feature_1 <= 19.00
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_1 >  19.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_1 >  20.50
|   |   |   |   |   |--- feature_1 <= 60.50
|   |   |   |   |   |   |--- feature_1 <= 25.50
|   |   |   |   |   |   |   |--- feature_0 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_0 >  0.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_1 >  25.50
|   |   |   |   |   |   |   |--- feature_0 <= 0.50
|   |   |   |   |   |   |   |   |--- class: 0
```

puis l'évaluation

```
print(classification_report(y_testing, y_pred))
```

	precision	recall	f1-score	support
0	0.76	0.85	0.80	26
1	0.90	0.83	0.86	42
accuracy			0.84	68
macro avg	0.83	0.84	0.83	68
weighted avg	0.84	0.84	0.84	68

➢ SVM

est un algorithme d'apprentissage automatique supervisé qui peut être utilisé à des fins de classification et de régression. Les SVM sont plus généralement utilisés dans les situations de classification.

Les SVM reposent sur l'idée de trouver un hyperplan qui divise au mieux un jeu de données en deux

```
from sklearn.model_selection import train_test_split
X_training, X_testing, y_training, y_testing = train_test_split(X, y ,  test_size=0.3)
```

```
from sklearn.svm import SVC
rbf_model = SVC(kernel='rbf')
```

```
rbf_model.fit(X_training, y_training)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma will change from 'au
scaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

```
rbf_model.score(X_testing,y_testing)
```

```
0.8382352941176471
```

La fonction du noyau est de prendre des données en entrée et de les transformer dans la forme requise. Différents algorithmes SVM utilisent différents types de fonctions du noyau. Ces fonctions peuvent être de différents types dans ce cas j'utilise 'rbf'
on trouve que l'accuracy est égale 0.83

➢ Random Forests

Cet algorithme appartient à la famille des agrégations de modèles, c'est en fait un cas particulier de bagging (bootstrap aggregating) appliqué aux arbres de décision de type CART.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50)
clf.fit(X_training, y_training)
predict = clf.predict(X_testing)
```

```
print(confusion_matrix(y_testing, predict))
```

```
[[22  4]
 [ 5 37]]
```

```
print(classification_report(y_testing, predict))
```

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83        26
           1       0.90      0.88      0.89        42

    accuracy                           0.87        68
   macro avg       0.86      0.86      0.86        68
weighted avg       0.87      0.87      0.87        68
```

l'accuracy de cette algorithmes est 0.87

➢ Logistic Regression

est un modèle de classification linéaire qui est le pendant de la régression linéaire , quand ne doit prendre que deux valeurs possibles (0 ou 1). Comme le modèle est linéaire, la fonction hypothèse pourra s'écrire comme suit :

$$H(x) = a1 + a1X1 + a2X2 + \ldots\ldots+an$$

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
model = LogisticRegression(solver='liblinear', random_state=0)
```

```
model.fit(X, y)
```

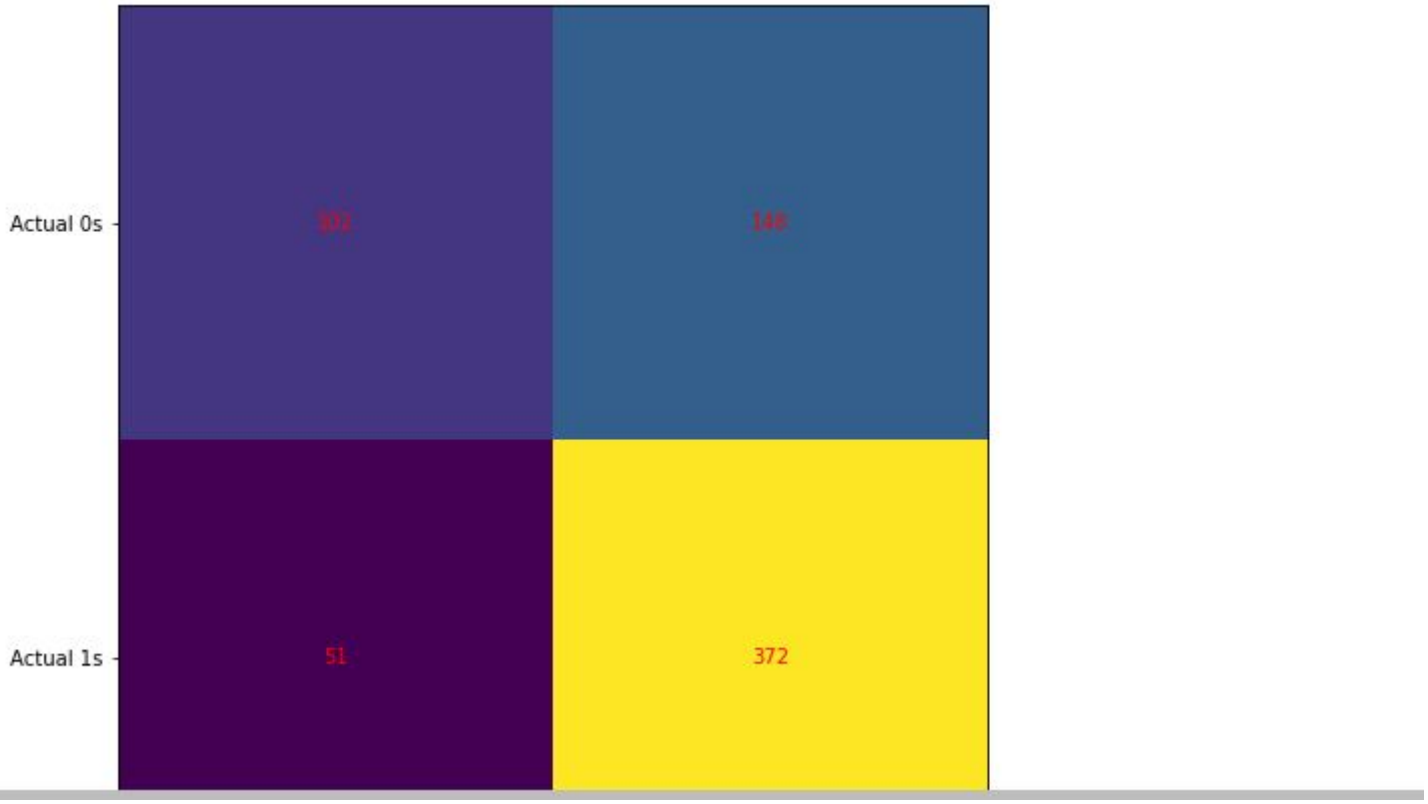
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

la matrice de confusion et l'accuracy :

```
print(classification_report(y, model.predict(X)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.41      0.51       250
           1       0.72      0.88      0.79       423

    accuracy                           0.70       673
   macro avg       0.69      0.64      0.65       673
weighted avg       0.70      0.70      0.68       673
```

```
cm = confusion_matrix(y, model.predict(X))

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



la meilleur algorithme pour ce daset c’est : RANDOM FOREST

➢ ENSEMBLE LEARNING

est le processus par lequel plusieurs modèles, tels que des classificateurs ou des experts, sont stratégiquement générés et combinés pour résoudre un problème particulier d'intelligence informatique. L'apprentissage d'ensemble est principalement utilisé pour améliorer les performances (classification, prédiction, approximation de fonction, etc.) d'un modèle

dans ce cas on va combiner tous les modèles précédents

```
from sklearn.ensemble import VotingClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import *
kfold = model_selection.KFold(n_splits=10)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
model4 = RandomForestClassifier()
estimators.append(('RandomForestClassifier', model4))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, y, cv=kfold)
print(results.mean())
```