Année Universitaire 2021-2022

BACHELOR EN GENIE INFORMATIQUE

Semestre S5

Rapport de projet sur l'intelligence artificielle

Diabète

Réalisé par :

- **Ziad Bougrine**

Présenté le 04/02/2022 devant le jury :

Benhlima Said de la faculté des sciences-Meknès

Oubelkacem Ali de la faculté des sciences-Meknès

Bekri Ali de la faculté des sciences-Meknès

Ba-Ichou Ayoub de la faculté des sciences-Meknès

Tout d'abord, je tiens à remercier tout particulièrement les personnes suivantes et à leur témoigner toute ma reconnaissance, pour leur dévouement et leur soutien pour la concrétisation de ce projet :

- ***M. Ali BEKRI***, responsable projet, pour m'avoir accordé toute la confiance nécessaire pour élaborer ce projet librement, et avoir mis à ma disposition tous les moyens nécessaires.

- ***M. Ba-Ichou Ayoub***, enseignant-chercheur, pour son investissement, sa contribution et toute l'aide et les conseils apportés durant ce projet.

Nous tenons à remercier également, Monsieur BENHLIMA et Monsieur Oubelkacem membre de jury

Introduction et problématique :

Dans ce problème, nous allons essayer de modéliser un modèle qui permet de prédire une maladie diabétique en se basant sur des données qui provient à l'origine d’un échantillon publié par un Institut du diabète. L'objectif de l'ensemble de données est de prédire avec un diagnostic si un patient est atteint du diabète. Les ensembles de données comprennent plusieurs variables prédictives médicales et une variable cible, « Out come ». Les variables prédictives comprennent le nombre de grossesses que le patient a eues, son IMC, son taux d'insuline, son âge, etc.

Etude de problématique

Etude des données

Grossesses: nombre de fois enceintes
Glucose: Concentration en glucose plasmatique 2 heures dans un test de tolérance au glucose par voie orale

Blood Pressure: pression artérielle diastolique (mm Hg)

Skin Thickness: Épaisseur du pli cutané des triceps (mm)

Insuline: insuline sérique de 2 heures (mu U / ml)

IMC: indice de masse corporelle (poids en kg / (taille en m) ^ 2)

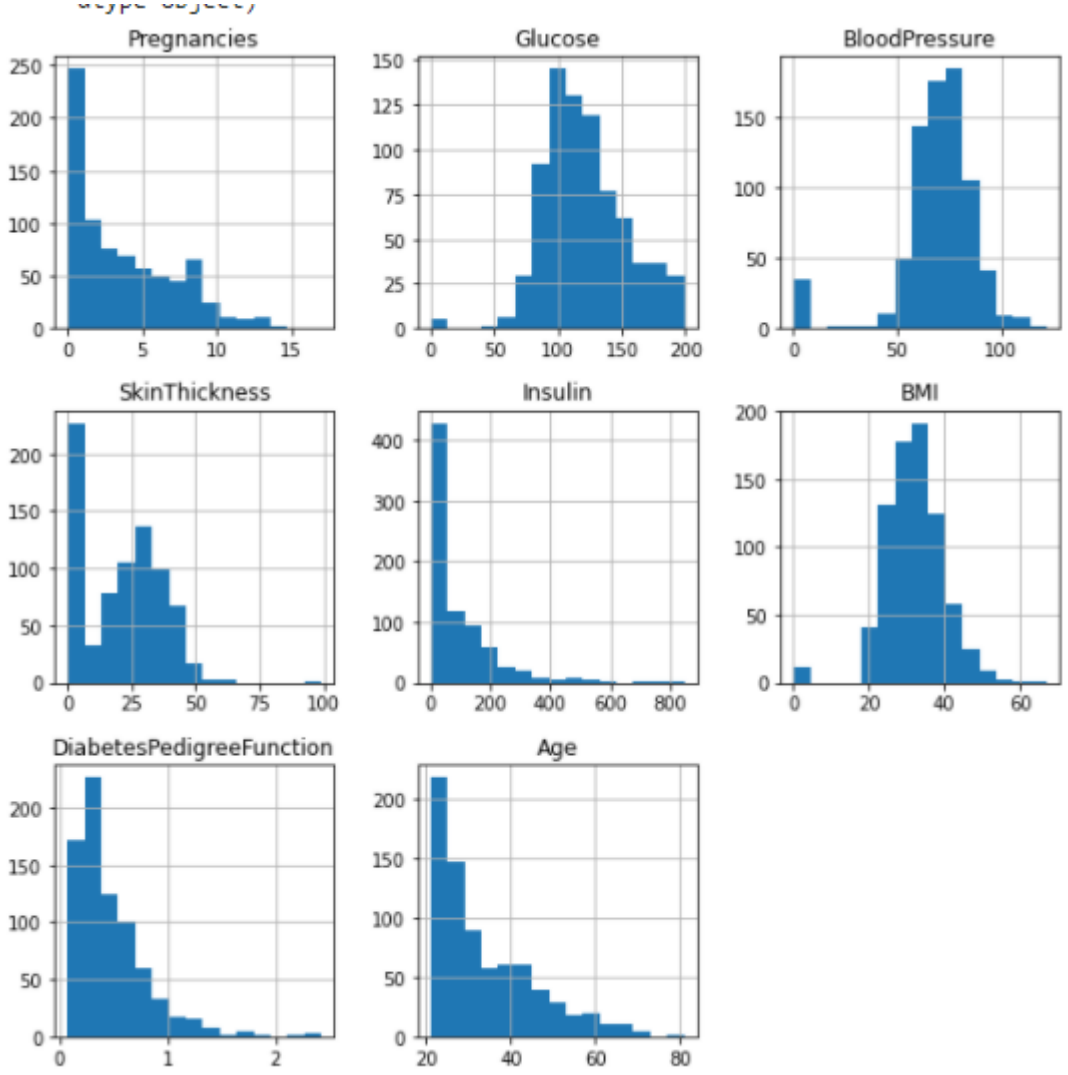
Diabetes Pedigree Function: Fonction pedigree du diabète
Age: Age (ans)
Résultat: Variable de classe (0 ou 1).

Avant de commencer, je vous présente une petite approche sur les données que nous allons étudier

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

On remarque ici un portion des données ce composant d’un ensemble des caractéristiques (features) des malades et un resultat de 0 ou 1, c’est positive ou bien négative de diabète

Pour le côté statistique, voici une distribution et concentration des données pour chaque caratéristuque des patients

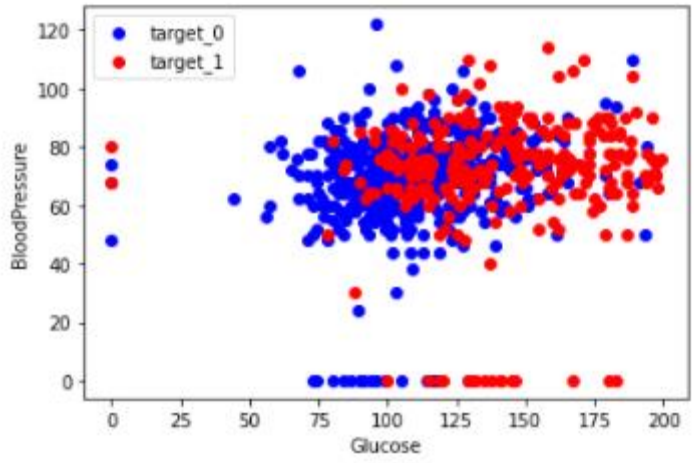


Pour nos données, la somme des données est

Malade : 268

Non malade :500

Dispertion des donées



Grâce à une recherche dans google, visualiser la dispersion des données grâce à deux caractéristiques (Glucose et et blood presure)

Finalement, pour notre étude nous allons prendre les données dans une dispersion normale pour avoir des résultats significatifs

```
X = scaler.fit_transform(X)
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)
```

Conclusion :

Nos données d’étude vont se répartir comme cela

20% Pour les données de teste

80% Pour les données d’entraînement

1 – Maching learning

Avant de commencer, l'objectif de ce projet est de réaliser une classification en se basant sur les données que nous avons visualiser

Pour notre problème, je vais comparer entre 4 modèles de machine Learning, j'utiliserai le Random Forest, régression logistique, le SVM et le KNN

Pour chaque modèle, j'utiliserai un GridSearch qui est une méthode prédéfinie par SickitLearn pour trouver les meilleurs paramètres d'un modèle, cette méthode permet de tester la précision de chaque modèle pour avoir le meilleur choix de paramètre

Pour le GridSearch nous observant que j'ai donné cv = 5 qui veut dire que le GridSearch utilisera une validation croisée pour avoir un bon modèle et tester toute portion de notre modèle

Le score que je cherche est accuracy ou exactitude

1 – Regression logistique

Pour le modèle logistique, après une recherche dans google et la bibliothèque Sickit learn, le solver, penalty et le paramètre C influence sur le modèle pour avoir un bon résultat

```
mod = LogisticRegression(max_iter=5000)
parameters = { 'solver': ['newton-cg', 'lbfgs', 'liblinear','saga'],
               'penalty':['none', 'l1', 'l2', 'elasticnet'],
               'C':[1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
               }
search = GridSearchCV(mod, parameters, scoring='accuracy', n_jobs=-1, cv=5)
result = search.fit(x_train,y_train)
```

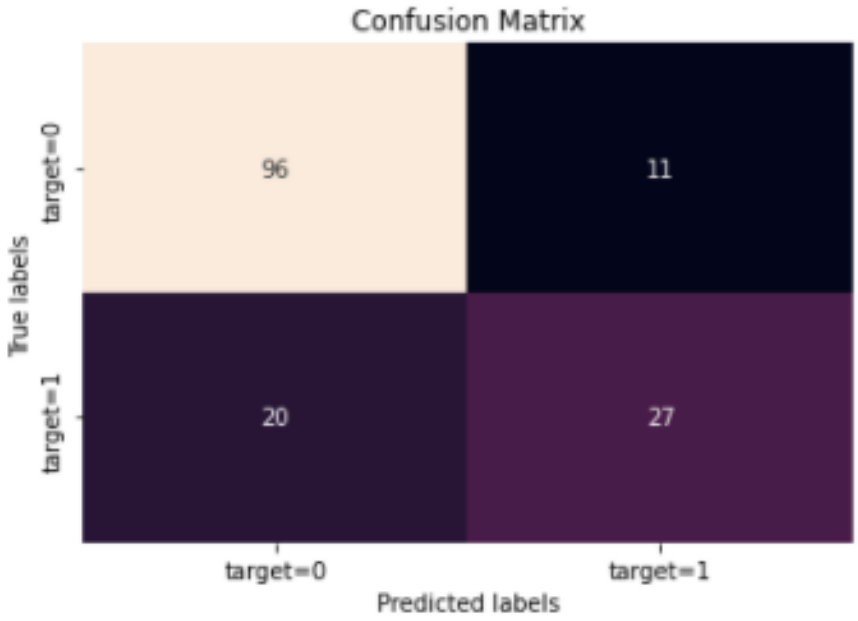
Chaque solveur essaie de trouver les poids des paramètres qui minimisent une fonction de coût.
Voici les cinq options :
newton-cg — Une méthode newton. Les méthodes de Newton utilisent une matrice hessienne exacte. Il est lent pour les grands ensembles de données, car il calcule les dérivées secondes.
lbfgs — Signifie Broyden–Fletcher–Goldfarb–Shanno à mémoire limitée. Il se rapproche des mises à jour de la matrice dérivée seconde avec des évaluations de gradient. Il ne stocke que les dernières mises à jour, ce qui économise de la mémoire. Ce n'est pas super rapide avec de grands ensembles de données.

liblinear — Bibliothèque pour la grande classification linéaire. Utilise un algorithme de descente de coordonnées. La descente de coordonnées est basée sur la minimisation d'une fonction multivariée en résolvant des problèmes d'optimisation univariés dans une boucle. En d'autres termes, il se déplace vers le minimum dans une direction à la fois. Il fonctionne plutôt bien avec une grande dimensionnalité. Il a un certain nombre d'inconvénients. Il peut rester bloqué, ne peut pas fonctionner en parallèle et ne peut résoudre la régression logistique multi-classes qu'avec un repos contre un.
saga — Extension de sag qui permet également la régularisation L1. Devrait généralement s'entraîner plus vite que l'affaissement.

Phase d'entrainement

```
lr=LogisticRegression(C=params1['C'],solver=params1['solver'],penalty=params1['penalty'],max_iter=5000)
lr.fit(x_train,y_train)
```

Matrice de confusion



Résultat

```python
#taux de succès
acc = metrics.accuracy_score(y_test,y_pred_logistic)
err=1.0-acc
#sensibilité (ou rappel)
se = metrics.recall_score(y_test,y_pred_logistic)
print("Le rappel est : {:.3f}".format(se))
print("L erreur est : {:.3f}".format(err))
print("L accuracy est : {:.3f}".format(acc))
print('Training set Accuracy :{:.3f}'.format(lr.score(x_train,y_train)))
print('Test set Accuracy :{:.3f}'.format(lr.score(x_test,y_test)))
```

```
Le rappel est : 0.574
L erreur est : 0.201
L accuracy est : 0.799
Training set Accuracy :0.775
Test set Accuracy :0.799
```

2 – SVM

Pour le SVM, j'utiliserai un classifier SVC

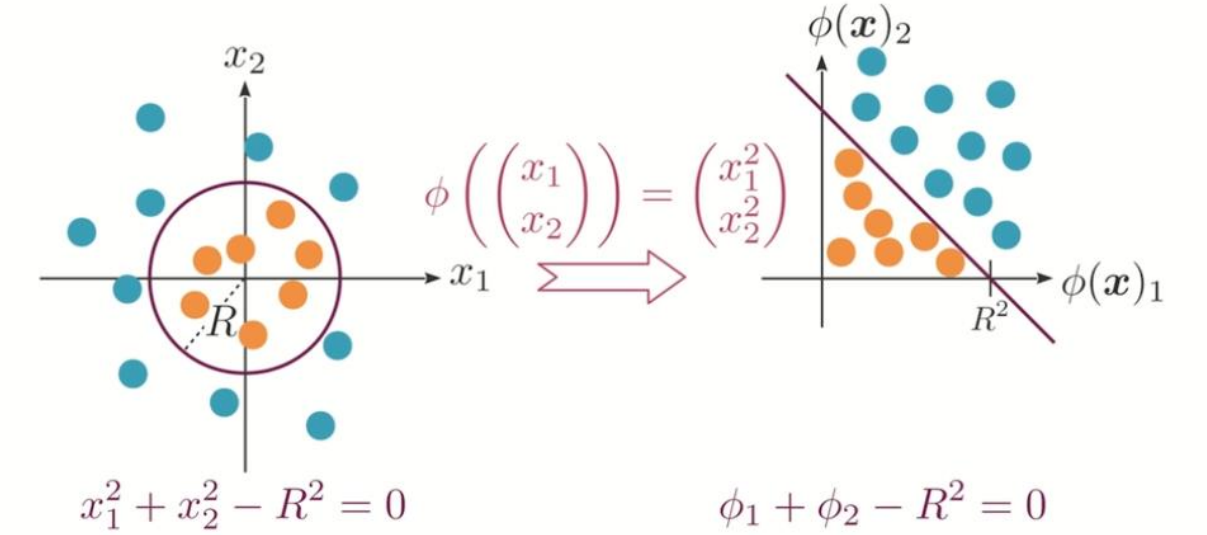
Avant d'entrainer le modèle, je commence par un GridSearch et une validation croisée pour trouver les meilleurs paramètres

```
parametres = [{'C':[0.1,1,10],'kernel':['rbf','linear']}]
grid = GridSearchCV(estimator=svm.SVC(),param_grid=parametres,scoring='accuracy',cv=5)
grid.fit(x_train,y_train)
```

J'ai cherché dans google et la documentation de Sickit learn que les paramètres noyau ou kernel et le paramètre C sont les paramètres qui améliorent le résultat et avoir un modèle puissant

Il a aussi été montré qu'en utilisant un noyau RBF, les SVMs deviennent un « approximateur universel » [1], c'est à dire qu'avec suffisamment de données, l'algorithme peut toujours trouver la meilleure frontière possible pour séparer deux classes (à condition que cette frontière existe).

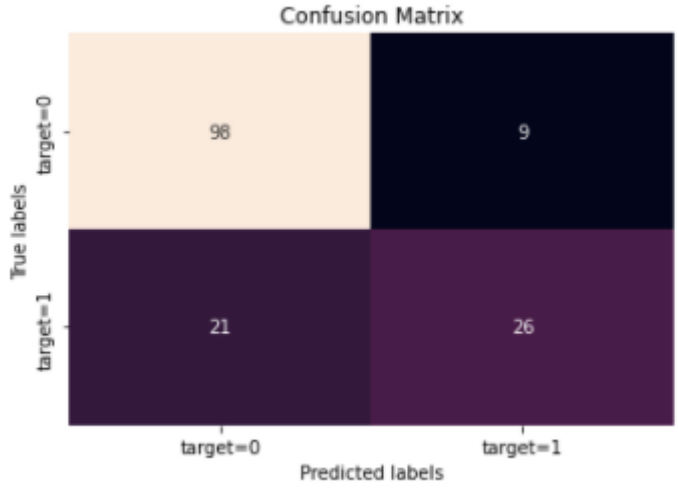
Transformation d'un modèle complexe en linéaire grâce au kernel



Phase d'entrainement

```
[26] classifier = svm.SVC(C=C,kernel=kernel)
     classifier.fit(x_train,y_train)
```

Matrice de confusion



Résultat

```
y_pred_svr = classifier.predict(x_test)
#taux de succès
acc_svr = metrics.accuracy_score(y_test,y_pred_svr)
err_svr=1.0-acc_svr
#sensibilité (ou rappel)
se_svr = metrics.recall_score(y_test,y_pred_svr)
print("Le rappel est : {:.3f}".format(se_svr))
print("L erreur est : {:.3f}".format(err_svr))
print("L accuracy est : {:.3f}".format(acc_svr))
print('Training set Accuracy :{:.3f}'.format(classifier.score(x_train,y_train)))
print('Test set Accuracy :{:.3f}'.format(classifier.score(x_test,y_test)))
```

```
Le rappel est : 0.553
L erreur est : 0.195
L accuracy est : 0.805
Training set Accuracy :0.814
Test set Accuracy :0.805
```

3 – KNN

J’ai suivi la même démarche pour le modèle KNN, je commence tous d’abord par un GridSearch pour trouver le meilleur nombre de Neighbors,

```
[29] parametres = {'n_neighbors':np.arange(2,25,1)}
     grid = GridSearchCV(estimator=KNeighborsClassifier(),param_grid=parametres,scoring='accuracy',cv=5)
     grid.fit(x_train,y_train)

     GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
                  param_grid={'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
            19, 20, 21, 22, 23, 24])},
                  scoring='accuracy')
```

Le modèle KNN permet de sélectionner le nombre de voisins que l’on spécifie, puis notre modèle sélectionne les voisins les plus proches.

Phase d'entraineme

```
[31] modelKNN= KNeighborsClassifier(n_neighbors=n_neighbors, weights='uniform')
```

Matrice de confusion

Résultat

```
[34] y_pred_knn = modelKNN.predict(x_test)
     #taux de succès
     acc_knn = metrics.accuracy_score(y_test,y_pred_knn)
     err_knn=1.0-acc_knn
     #sensibilité (ou rappel)
     se_knn = metrics.recall_score(y_test,y_pred_knn)
     print("Le rappel est : {:.3f}".format(se_knn))
     print("L erreur est : {:.3f}".format(err_knn))
     print("L accuracy est : {:.3f}".format(acc_knn))
     print('Training set Accuracy :{:.3f}'.format(modelKNN.score(x_train,y_train)))
     print('Test set Accuracy :{:.3f}'.format(modelKNN.score(x_test,y_test)))
```

```
Le rappel est : 0.447
L erreur est : 0.208
L accuracy est : 0.792
Training set Accuracy :0.765
Test set Accuracy :0.792
```

4-Random Forest

J’ai suivi la même démarche pour le modèle Random Forest, je commence tous d’abord par un GridSearch pour trouver le meilleur nombre profondeurs maximal, et le meilleur nombre d’estimateurs

```
[ ]  parametres = {'max_depth':np.arange(3,10,1),'n_estimators':np.arange(4,200,1)}
     grid = GridSearchCV(estimator=RandomForestClassifier(),param_grid=parametres,scoring='accuracy',cv=5)
     grid.fit(x_train,y_train)
```

Le nombre d’estimateurs est le nombre quand va prendre pour les entraîner sur des données et prendre le meilleur vote ou score

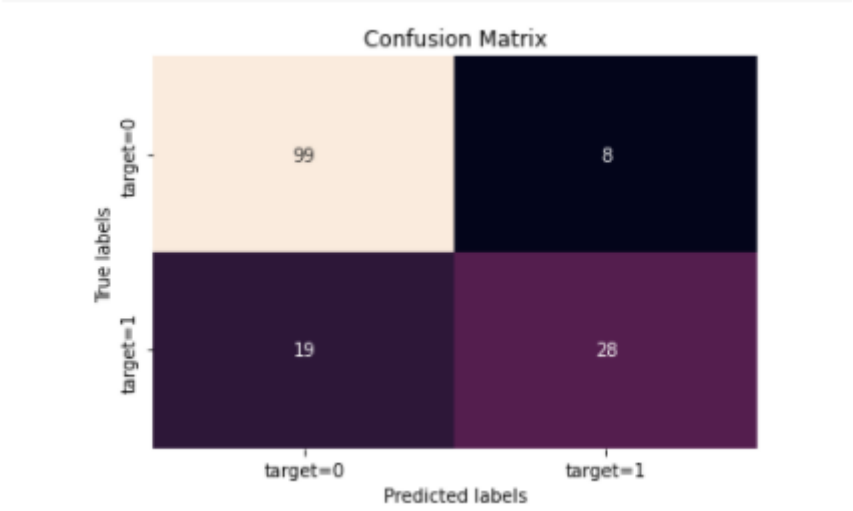
Max_depth est la profondeur maximale de ces arbres

Phase d'entrainement

```
[39] rf = RandomForestClassifier(max_depth=5,n_estimators=182, random_state=0)
     rf.fit(x_train, y_train)
```

```
RandomForestClassifier(max_depth=5, n_estimators=182, random_state=0)
```

Matrice de confusion



Résultat

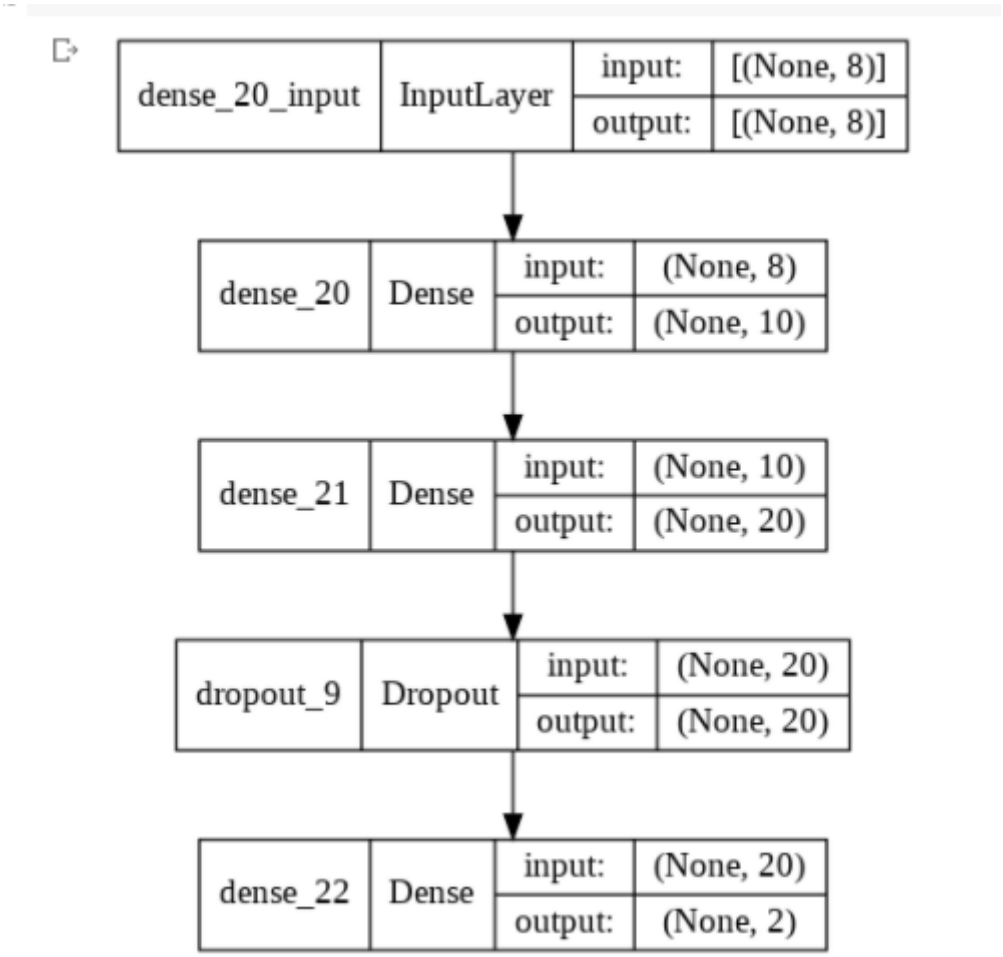
```
[40] y_pred_random = rf.predict(x_test)
     #taux de succès
     acc_random = metrics.accuracy_score(y_test,y_pred_random)
     err_random=1.0-acc_random
     #sensibilité (ou rappel)
     se_random = metrics.recall_score(y_test,y_pred_random)
     print("Le rappel est : {:.3f}".format(se_random))
     print("L erreur est : {:.3f}".format(err_random))
     print("L accuracy est : {:.3f}".format(acc_random))
     print('Training set Accuracy :{:.3f}'.format(rf.score(x_train,y_train)))
     print('Test set Accuracy :{:.3f}'.format(rf.score(x_test,y_test)))
```

```
Le rappel est : 0.596
L erreur est : 0.175
L accuracy est : 0.825
Training set Accuracy :0.850
Test set Accuracy :0.825
```

Partie Deep Learning

Introduction:

Pour la partie Deep learning, j’ai utilisé un réseau Neuron comme suit :



Mon modèle :

```
def getTheBestNeural():
    scores = []
    for i in np.arange(20,100,1):
      print("i===> {}".format(i))
      stopping_val_loss =EarlyStopping(monitor="val_loss",patience=70,mode="min")
      stopping_val_accuracy =EarlyStopping(monitor="val_accuracy",patience=70,mode="max")
      kara = Sequential()
      kara.add(Dense(i,input_dim=8, activation='relu'))
      kara.add(Dropout(0.2))
      kara.add(Dense(2, activation='softmax'))
      kara.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
      history = kara.fit(x_train, y_train1, epochs=150, batch_size=40,validation_data=(x_test,y_test1),callbacks=[stopping_val_loss,
      score = kara.evaluate(x_test,y_test1)
      score.append(i)
      scores.append(score)
    errors = []
    for element in scores:
      errors.append(element[0])
    meanError = np.mean(errors)
    max = scores[0]
    for j in range(len(scores)):
      if scores[j][0] <= meanError:
        if scores[j][1] > max[1]:
          max = scores[j]
    return max
bestones = getTheBestNeural()
```

Pour trouver le meilleur nombre de neurones de mon modèle, j'ai utilisé une boucle qui itère de 20 à 100 neurones et teste le score de chaque modèle puis je retourne le score maximal

```
[100] print(bestones)

    [0.42168688774108887, 0.8441558480262756, 40]
```

40 neurones dans la première couche m'ont donné des résultats significatifs

Code du modèle

```
kara = Sequential()
kara.add(Dense(40,input_dim=8, activation='relu'))
kara.add(Dropout(0.2))
kara.add(Dense(2, activation='softmax'))
kara.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Phase d'entrainement

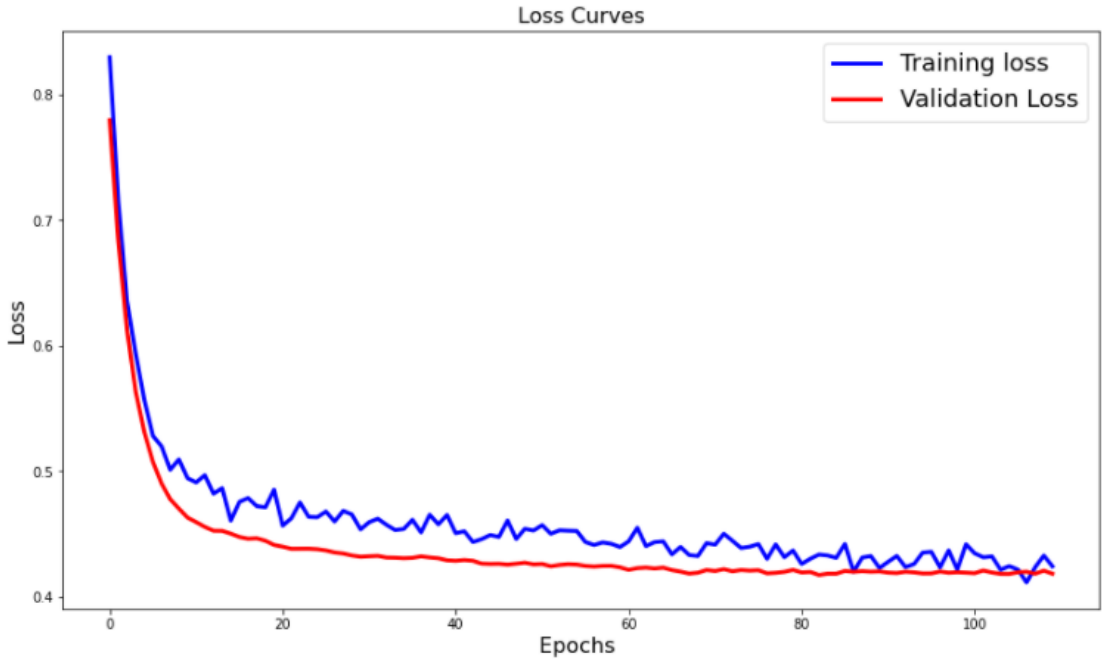
```
stopping_val_loss =EarlyStopping(monitor="val_loss",patience=70,mode="min")
stopping_val_accuracy =EarlyStopping(monitor="val_accuracy",patience=70,mode="max")
seed = 7
np.random.seed(seed)
history = kara.fit(x_train, y_train1, epochs=150, batch_size=40,validation_data=(x_test,y_test1),callbacks=[stopping_val_loss,stopping_val_accuracy],verbose=2) #
score = kara.evaluate(x_test,y_test1)
print(score)
```

La phase d'entrainement c'est un entrainement standard qu'on sait, j'ai juste ajouté des conditions pour stopper l'entrainement en cas où, on a une erreur de teste qui augmente, ou on a une validation de teste qui diminue

Je prends puis un ensemble de donnée aléatoirement de 40 pour chaque époque, j'ai 150 époques.

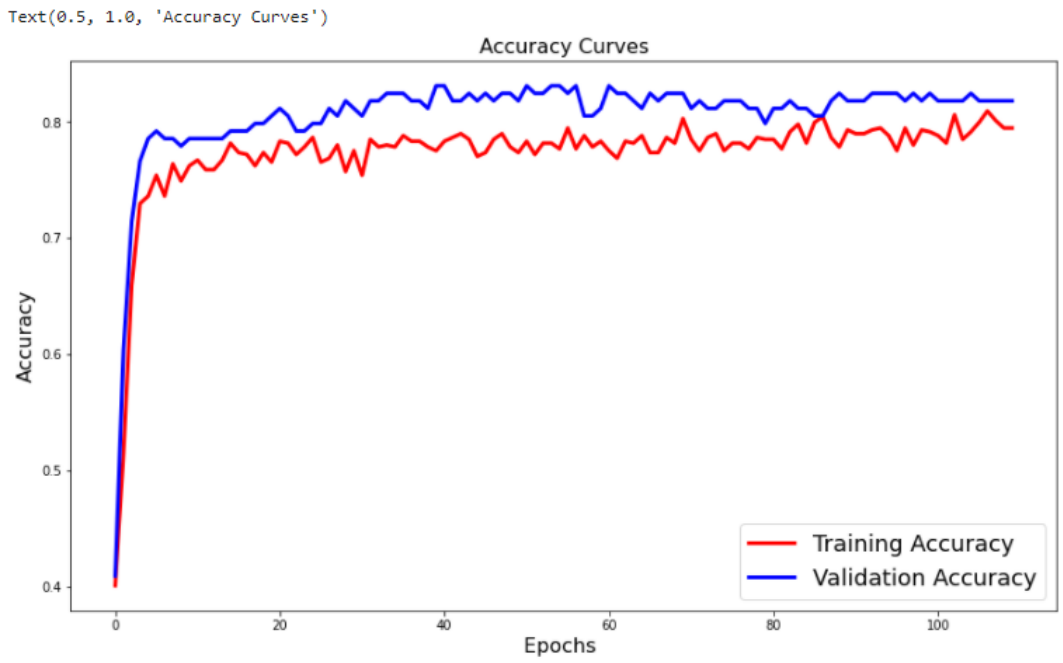
Historique d’erreur

```
Text(0.5, 1.0, 'Loss Curves')
```



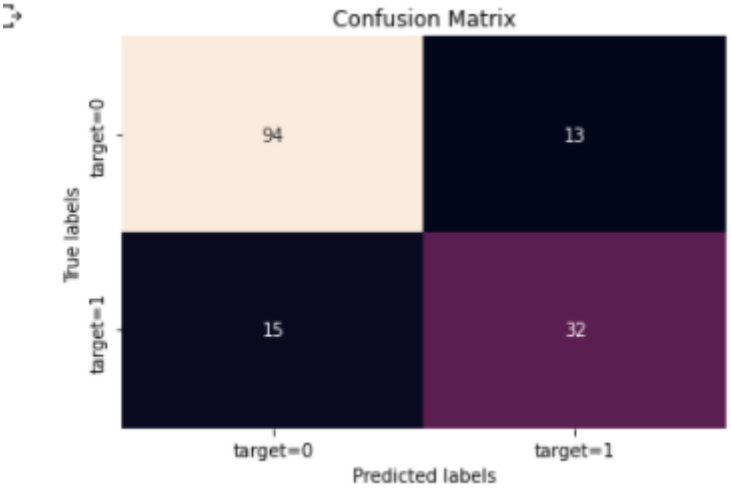
Cette historique compare une comparaison entre la diminution des erreurs de validations pour chaque époque avec les erreurs d’entrainement

Historique de validation



Cette historique compare une comparaison entre l’augmentation de la précision de validations pour chaque époque avec la précision d’entrainement

Matrice de confusion

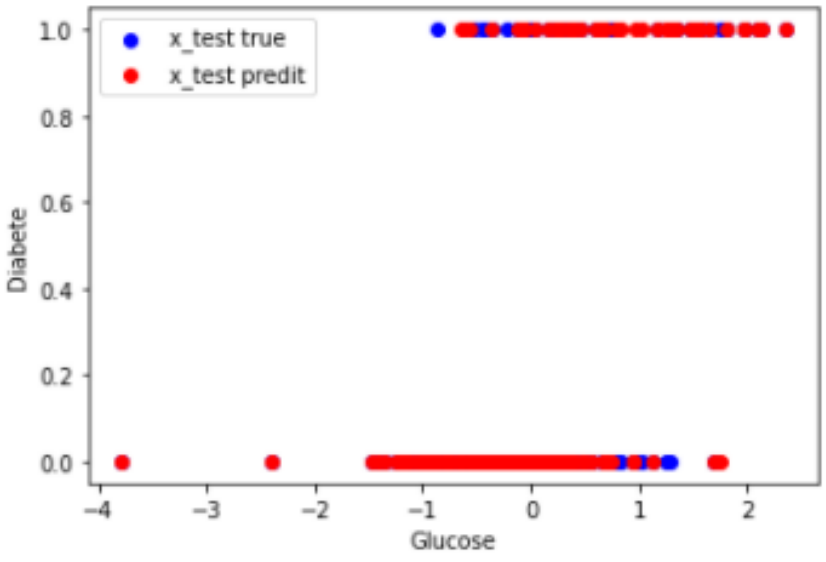


Score deep learning

```
score = kara.evaluate(x_test,y_test1)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.4182 - accuracy: 0.8182
```

Description précise de deep learning



Résultat de deep learning

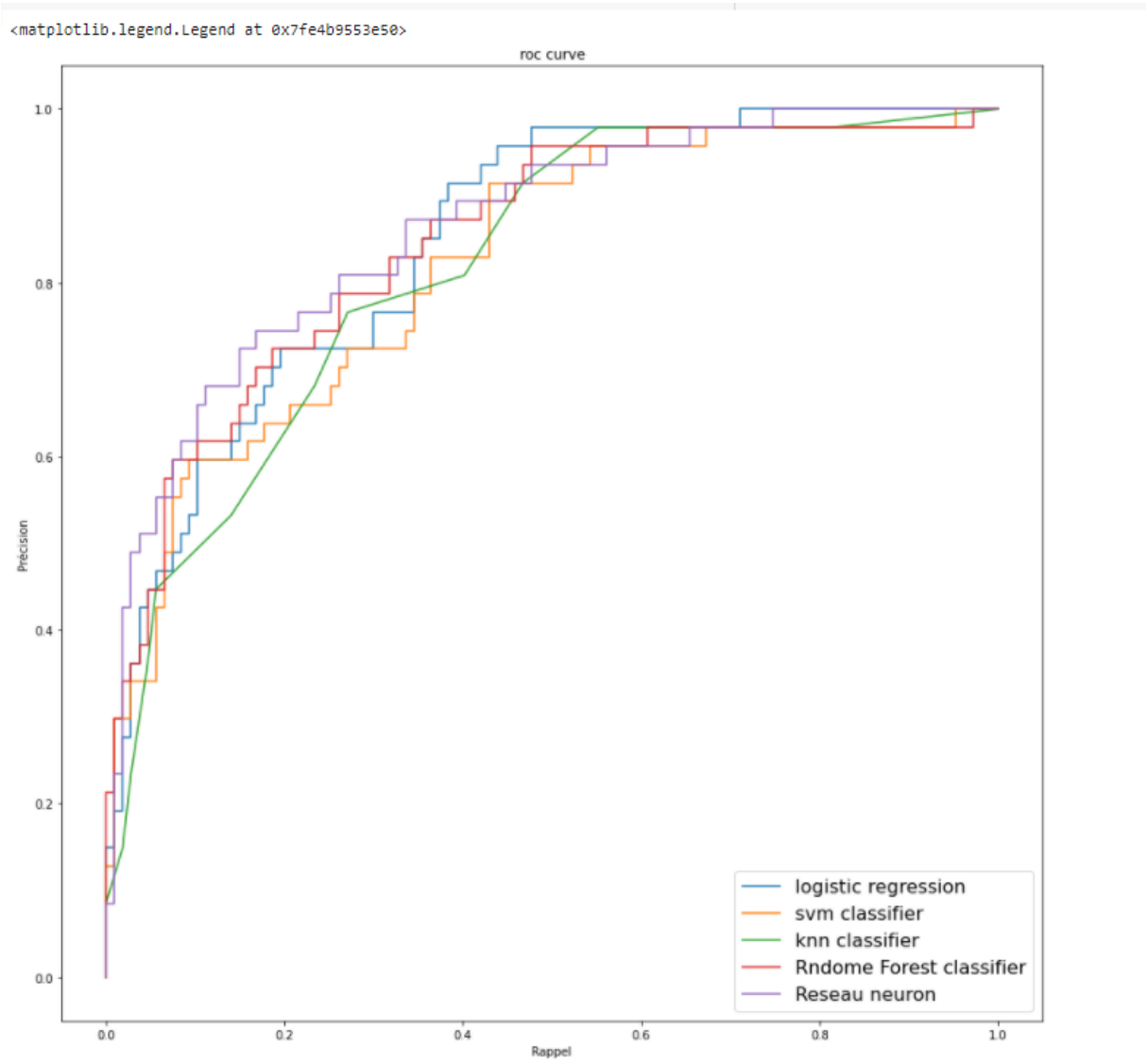
```
#taux de succès
acc_deep = metrics.accuracy_score(y_test,y_pred_deep)
err_deep=1.0-acc_deep
#sensibilité (ou rappel)
se_deep = metrics.recall_score(y_test,y_pred_deep)
print("Le rappel est : {:.3f}".format(se_deep))
print("L erreur est : {:.3f}".format(err_deep))
print("L accuracy est : {:.3f}".format(acc_deep))

scoretrain = kara.evaluate(x_train,y_train1)
scoretest = kara.evaluate(x_test,y_test1)
print('Training set Accuracy :{:.3f}'.format(scoretrain[1]))
print('Test set Accuracy :{:.3f}'.format(scoretest[1]))
```

```
Le rappel est : 0.681
L erreur est : 0.182
L accuracy est : 0.818
20/20 [==============================] - 0s 2ms/step - loss: 0.4109 - accuracy: 0.8094
5/5 [==============================] - 0s 3ms/step - loss: 0.4182 - accuracy: 0.8182
Training set Accuracy :0.809
Test set Accuracy :0.818
```

Courbe à ROC

Cette mesure permet de comparer entre les modèles de cette étude d’une manière graphique et plus significatifs



On remarque que le réseaux neurones de couleurs violets est le modèle le plus performant de notre étude

Export de données

Exporting data

```
[118] #Export machine learning
      pickle.dump(rf,open("drive/MyDrive/machineLearning.sav","wb"))
```

```
[119] model_json = kara.to_json()
      with open('drive/MyDrive/deeplearning.json', "w") as json_file:
          json_file.write(model_json)
      kara.save_weights("drive/MyDrive/deeplearning.h5")
```

```
[120] #Export machine learning
      pickle.dump(scaler,open("drive/MyDrive/myscaler","wb"))
```

Machine learning :

Je sauvegarde avec la bibliothèque Pickle mon modèle

Deep learning :

J’exporte la suite de séquences de mon modèle puis j’exporte toutes les valeurs de biais de mon modèle

Scaler :

J’exporte le scaler puisqu’il suive une loi normal de moyen et de sigma donc j’en ai besoin dans mon application pour pouvoir transformer les entrées en loi normal

Vue générale de l’application

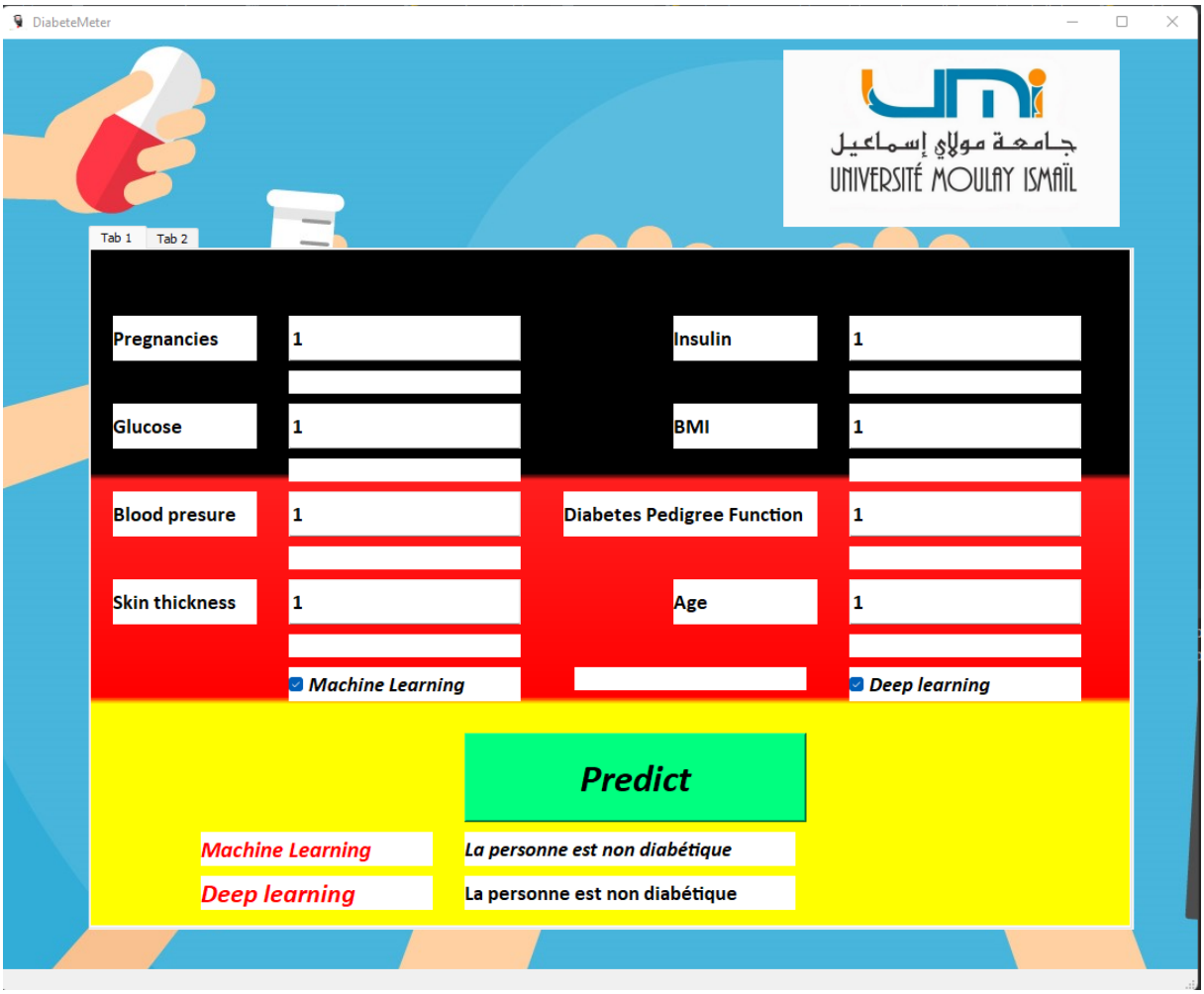
Première interface

Pour démarrer le script

Python main.py -g

Python main.py

Interface responsable pour la saisie des données manuelles et pour donner des prédictions sur la maladie



Deuxième interface

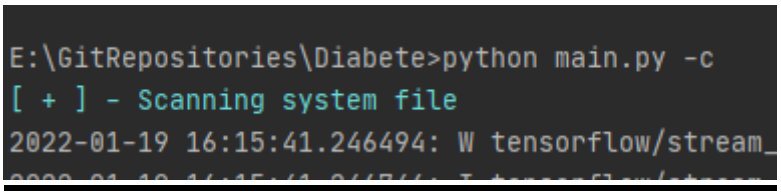
Interface responsable pour l’importation des fichiers Excel et dire des prédictions sur les maladies avec un modèle précise

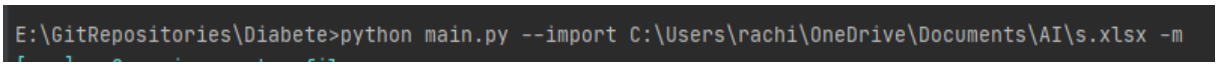
3 – Troisième interface

Interface commande pour faire des prédictions en mode commande

python main.py -c

python main.py - -import excelfile -m(machine learning) ou -d deeplearning





Pour le déploiement de l’application vous devez avoir un python de version préférable 3.9.7 configurer dans l’environnement de variable et configurer le PATH du module pip qui se trouve dans dossierpython/Scripts dans le path, l’application va se charger pour l’installation des dépendances