UFR Sciences et Techniques

Détection du Diabète de type II par l'intermédiaire de l'apprentissage automatique

Rapport du projet personnel de recherche

Nessia Fennech

2023-2024

Tutrice : Gloria Faccanoni
Laboratoire : Imath

Table des matières

1. Introduction

De nos jours, le machine learning est employé dans de nombreux domaines pour effectuer des analyses prédictives, notamment pour créer des modèles de détection précoce et de prévention des maladies. Aussi appelé apprentissage automatique, il représente un modèle d'apprentissage qui permet à des applications de prédire des résultats de plus en plus précis. Dans le monde, plus de 500 millions de personnes vivent avec le diabète dont 96 % souffrent d'un diabète de type II. Ce nombre devrait plus que doubler et atteindre 1,3 milliard de cas dans les 30 prochaines années. Cette augmentation met en évidence l'importance d'améliorer le dépistage pour mieux gérer et prévenir le diabète (Voire par exemple [2]).

Le diabète de type II est une maladie qui est caractérisé par un taux de glucose ou de sucre trop élevé dans le sang. Elle survient lorsque le pancréas ne produit pas suffisamment d'insuline ou lorsque l'organisme ne parvient pas à utiliser efficacement l'insuline produite. Les complications de cette maladie sont sérieuses pouvant affecter significativement les vaisseaux sanguins, les reins ainsi que les nerfs.

Le Machine Learning offre une approche prometteuse pour aborder ce problème. Dans notre étude, nous explorerons un exemple concret d'application du Machine Learning, en utilisant un classificateur basé sur un arbre de décision pour tenter de résoudre le problème du dépistage du diabète. Un classificateur d'arbre de décision est un modèle qui utilise une série de choix simples pour diviser les données en groupes basés sur leurs caractéristiques. Cette approche nous permettra d'analyser et de choisir de manière efficace les données les plus pertinentes, offrant ainsi une prise en charge précoce du diabète.

Nous utiliserons des données provenant de de l'étude Pima Indians Diabetes faite par l'Université de Californie, School of Information and Computer Science. (Sujet issu de [1]) Nous nous intéresserons à plusieurs critères sélectionnés parmi une base de données plus grande. En particulier, tous les patients, ici, sont des femmes âgées d'au moins 21 ans. De plus, les patients sont également d'origine indienne Pima, population la plus touchée dans le monde par le diabète de type II. L'ensemble de données comprennent plusieurs caractères, notamment une variable cible spécifique, « M ». Cette variable de classe indiquera si l'individu est sain ou malade. Elle vaut 1 si l'individu est malade et 0 si ce n'est pas le cas.

Dans notre jeu de données, chaque ligne représente un patient et les colonnes correspondent aux variables prédictives et à la variable cible. Les variables sont les suivantes :

- Grossesses : nombre de fois enceinte
- Glucose : concentration en glucose plasmatique 2 heures dans un test de tolérance au glucose par voie orale
- Pression : pression artérielle diastolique (mm Hg)
- Insuline : insuline sérique de 2 heures (mu U / ml)
- IMC : indice de masse corporelle (poids en kg) / (taille en m^2)
- K : coefficient lié à la présence du diabète parmi ses parents et proches
- Age : âge en année (ans)
- M : variable de classe (0 ou 1) qui indique si l'individu est sain (0) ou malade (1).

Le nombre total de patient étudié est de 500. Pour tenter de résoudre le problème du diabète, nous allons construire un modèle qui va faire des prédictions. Pour cela, nous devons trouver un moyen d'évaluer la qualité de ces prédictions. Étant donné que les prédictions, par définition, ne concernent que des données inédites, nous ne pouvons pas dépendre des données utilisées pour créer le modèle. Nous devons, tout d'abord, diviser le jeu de données en deux parties non croisées : les données d'entraînement qui seront utilisées pour construire le modèle et les données de test qui serviront à évaluer les prédictions du modèle. Nous utiliserons, ensuite, l'ensemble d'entraînement pour construire notre classificateur. Enfin, nous évaluerons la performance de notre modèle sur l'ensemble de test.

Nous allons nous servir des bibliothèques Pandas, Pylab et Seaborn pour analyser nos données. Dans notre cas, il existe déjà deux fichiers séparés (avec des données déjà bien préparées). La table T sera utilisée pour l'entraînement et la table P pour la phase de test. Les données d'entraînement contiennent les données de 400 individus, tandis que celles de test contiennent les données de 100 patients. Nous analyserons nos données et étudierons une méthode de machine learning permettant de réaliser des analyses prédictives.

2. Exploration des données

Notre premier objectif est de comprendre les données et trouver les paramètres qui influent le plus sur la variable cible ("M"). La table T contient les données d'entraînement de 400 individus qui vont permettre à notre modèle de reconnaître une personne diabétique.

Nous devons, tout d'abord, savoir quelle est la proportion de malade dans notre jeu de données. Dans notre jeu de données, il y a autant de personnes malades que de personnes saines. Notre jeu de données est donc équitablement réparti pour construire notre modèle. Notre dataset ne possède pas de valeurs manquantes. Ensuite, nous pouvons observer le résumé statistique de de nos données pour en avoir un premier aperçu.

	grossesses	glucose	pression	insuline	imc	K	age	M
count	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	3.820000	107.422500	72.15250	155.992500	31.853995	0.427129	30.780000	0.500000
std	3.526854	31.304535	16.42021	140.449859	9.642392	0.425077	12.612863	0.500626
min	0.000000	45.000000	27.00000	14.000000	18.217352	0.078656	21.000000	0.000000
25%	1.000000	85.000000	60.00000	45.000000	21.561065	0.137473	22.000000	0.000000
50%	3.000000	105.000000	74.00000	127.500000	31.839371	0.198598	24.000000	0.500000
75%	6.000000	125.000000	84.00000	218.000000	39.107978	0.666726	38.250000	1.000000
max	14.000000	190.000000	116.00000	758.000000	55.421694	2.278046	71.000000	1.000000

FIGURE 1 – Statistiques des données d'entraînement

Par exemple, nous pouvons constater que la moyenne de grossesses dans notre échantillon est de 3.82, ce qui est assez élevé par rapport à la moyenne nationale française de 1.8, selon l'INSEE. De plus, le nombre maximal de grossesses observé dans notre série est de 14, ce qui est également un chiffre très élevé.

Ces observations suggèrent que notre échantillon pourrait présenter une prévalence de grossesses plus élevée que la moyenne nationale. Il serait intéressant d'explorer comment cette variable est corrélée au diabète. En particulier, nous devrions examiner si les femmes ayant un nombre élevé de grossesses sont plus susceptibles de développer le diabète par rapport à celles ayant moins de grossesses.

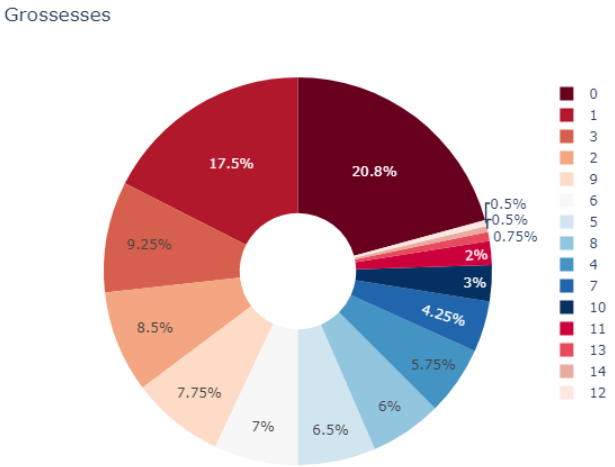


FIGURE 2 – Répartition du Diabète en fonction du nombre de grossesses

Ce diagramme montre que si on exclu les femmes sans enfants, plus le nombre de grossesses augmente, plus le nombre de femmes diabètiques augmente. Cependant, ce n'est pas suffisant

pour conclure sur l'influence de la grossesse sur le diabète. Nous devons effectuer des analyses statistiques pour valider notre observation. Nous pouvons explorer d'autres paramètres comme l'IMC. D'après E-santé l'IMC d'une femme se situe en moyenne entre 18.5 et 25. Nous pouvons chercher à voir combien de personnes ont un IMC supérieur à la moyenne en considérant seulement les individus qui ont un IMC supérieur à 25 dans notre jeu de données. Dans notre jeu de données, 272 individus sur 400 ont un IMC supérieur à 25. Ici, il y a donc plus de la moitié de la population étudiée qui a un IMC supérieur à la moyenne. Ce qui peut nous faire penser que l'IMC est un paramètre indicateur du diabète.

De plus, nous pouvons chercher à séparer la moyenne de chaque variable en deux catégories : ceux malades (attribués à la valeur 1) et ceux sains (attribués à la valeur 0).

M	grossesses	glucose	pression	insuline	imc	K	age
0	2.0	105.110	70.890	114.075	29.342952	0.342397	27.005
1	5.0	109.735	73.415	197.910	34.365038	0.511860	34.555

FIGURE 3 – Répartition des variables en fonction du statut de maladie

À première vue, nous aurions pu penser que l'insuline est le paramètre influençant principalement le diabète. Cependant, en observant les données, nous constatons que les personnes malades présentent des valeurs beaucoup plus élevées que les personnes saines pour chaque paramètre. Nous pouvons, notamment remarquer que, les personnes ont en moyenne plus que deux fois plus d'enfants que celles non-malades.

2.1. Boîte à moustaches

Nous pouvons, ici, utiliser un box plot, aussi appelé boîte à moustaches pour analyser la distribution des données. Il nous permettra d'avoir un résumé graphique pour identifier l'asymétrie de l'ensemble des données.

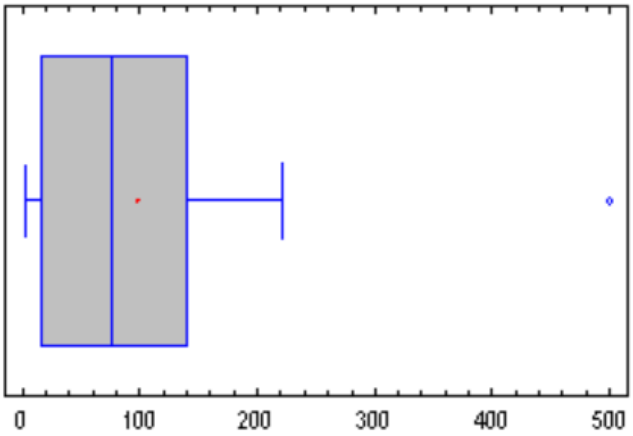


FIGURE 4 – Boîte à moustaches

Une boîte à moustache est composée d'une boîte qui correspond à l'écart interquartile. Ses cotés gauches et droits représentent les quartiles des données. Elle est coupée en deux par une ligne qui représente la médiane. Le point rouge qui est situé à droite de la médiane représente la moyenne. De plus, des moustaches (des barres) se situent à gauche et à droite de la boîte (Extrait de [3]).

Ce graphique est surtout intéressant lorsqu'on essaye de comparer des variables sur des échelles similaires. Nous pouvons, ici, utiliser un box plot pour analyser la distribution des données. On obtient le graphique suivant :

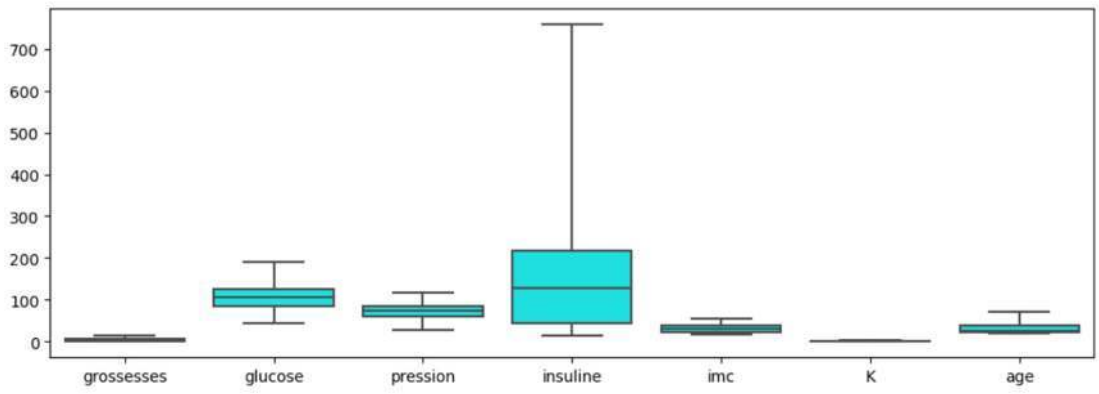


FIGURE 5 – Boîte à Moustache T

Nous pouvons voir que les variables n'ont pas le même ordre de grandeur, notamment l'insuline et le glucose. Ce qui empêche une comparaison directe.

2.2. Normalisation : centrage et réduction des données

Pour pouvoir comparer les différents paramètres (qui n'ont pas les mêmes unités, ni la même étendue), nous allons standardiser les caractéristiques en les centrant autour de 0 et avec un écart-type de 1 en les normalisant. Pour cela, on applique la formule :

$$T_n = \frac{T[\text{parametres}] - m}{\sigma}$$

où m représente la moyenne et σ l'écart-type. On obtient une nouvelle table Tn avec les données normalisées. Pour voir si les valeurs sont normalisées, nous pouvons alors vérifier si la moyenne est nulle et si la variance égale à 1. Affichons notre boxplot avec les nouvelles variables normalisées :

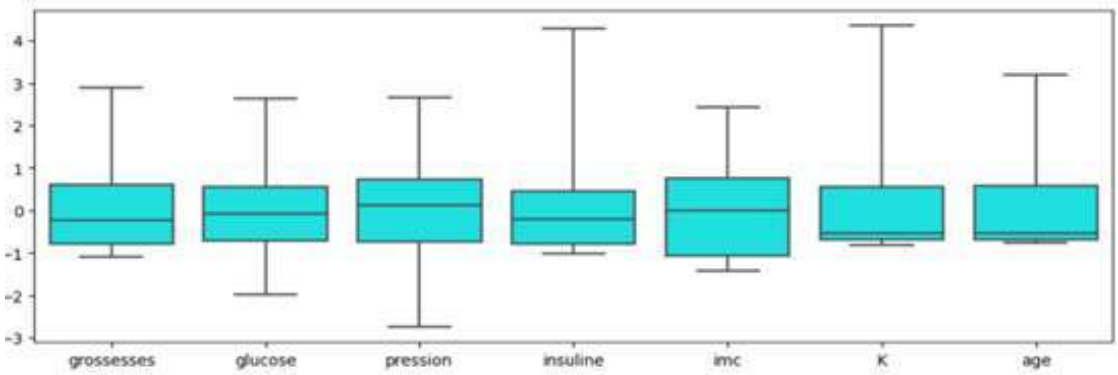


FIGURE 6 – Boîte à Moustache Tn

Nous remarquons que maintenant que les valeurs sont normalisées, les dispersions sont comparables. On normalise également les paramètres des patients de la table de test avec la même moyenne et le même écart-type que ceux de la table précédente et on crée la table Pn avec les données normalisées. On obtient la boîte à moustache suivante :

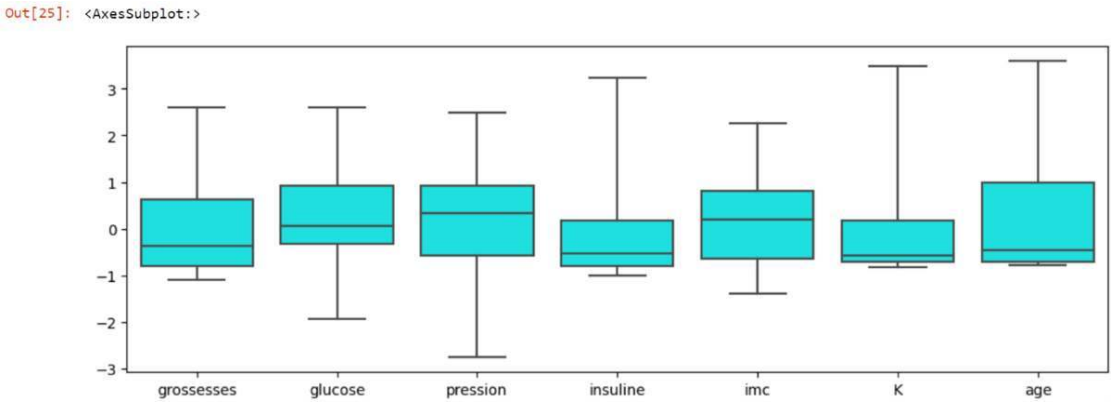


FIGURE 7 – Boîte à Moustache pour les données de test

3. Choix des paramètres

Pour construire notre modèle, nous devons identifier les variables les plus prépondérantes afin de déterminer les caractéristiques qui ont le plus d'impact sur les personnes atteintes d'un diabète. Pour cela, nous devons chercher visuellement les attributs qui séparent nos données entre les malades et le non-malades.

3.1. Matrice de Corrélation

Nous pouvons notamment utiliser une matrice de corrélation pour voir quels paramètres sont les plus liés à la variable 'M'. Une matrice de corrélation permet d'évaluer la relation entre plusieurs variables. Nous allons visualiser la matrice de corrélation de notre jeu de données avec une "heatmap" (ou carte de chaleur) qui est un outil permettant de remplacer les nombres par des couleurs et décrire l'intensité des données :

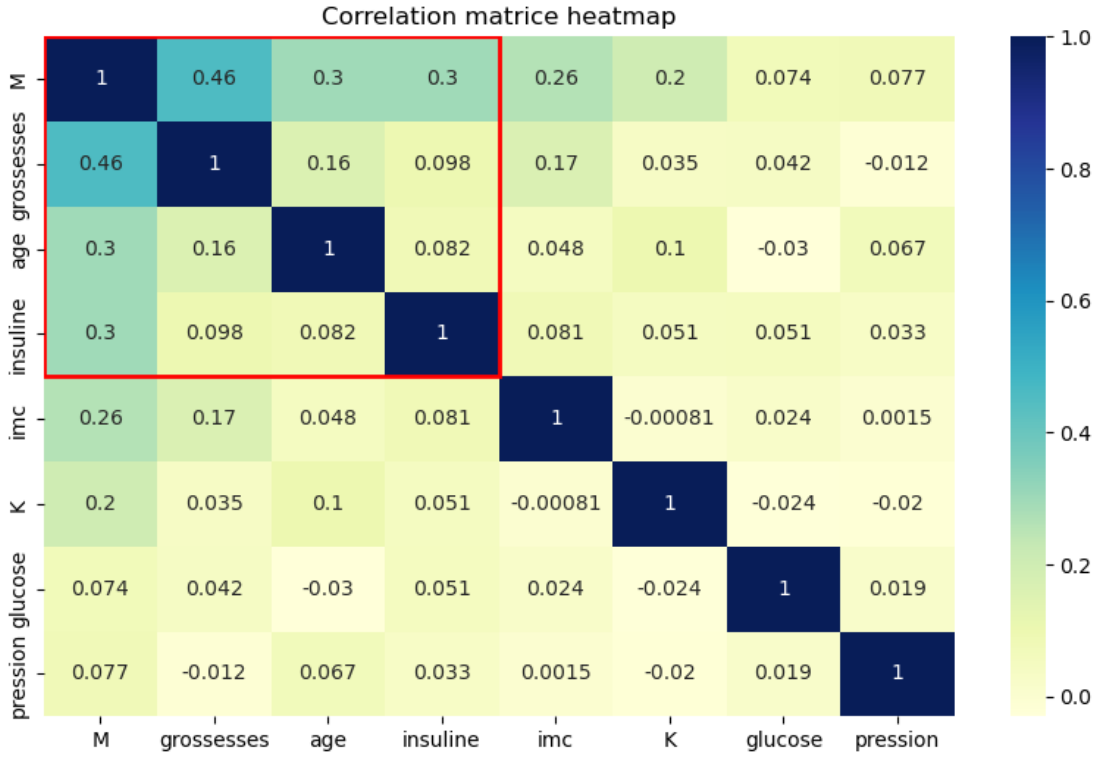


FIGURE 8 – Matrice de Corrélation

Avec un facteur de corrélation de 0,46, la Grossesse a la corrélation la plus élevée avec la variable "M", c'est-à-dire que la Grossesse est le facteur de risque le plus important pour détecter le diabète d'après la heatmap. De plus, l'insuline et l'âge peuvent également être considérés comme des facteurs significatifs pour prédire le diabète. Les autres paramètres ont des coefficients de corrélation beaucoup plus faibles. Par ailleurs, nous pouvons voir que plus la case a une teinte de bleu foncée, plus le paramètre correspondant est en forte corrélation avec la variable "M". Les paramètres les plus corrélés avec la variable "M" semblent ainsi être la Grossesse, l'insuline et

l'âge. Nous pouvons chercher à utiliser d'autres méthodes pour déterminer quelles variables ont le plus de corrélation avec la variable "M" pour choisir les paramètres les plus influents.

3.2. Visualisation 3D

Nous avons, aussi, la possibilité de déterminer les facteurs qui permettent de distinguer les malades des non-malades par une représentation 3D des individus en fonction de trois paramètres. Nous pouvons notamment nous servir d'un diagramme de dispersion 3D qui permet de produire des graphiques en trois dimensions. Nous choisissons alors la représentation qui est la plus favorable pour distinguer le nuage des malades avec celui des non malades. On représente les individus malades en rouge et les personnes saines en vert. En testant plusieurs paramètres pour les valeurs x,y,z, nous pouvons remarquer que les valeurs les plus prépondérantes semblent toujours être : l'insuline, l'imc et les grossesses. Nous associons alors à la fonction "Par" ces 3 paramètres que nous utiliserons pour nos prédictions. Le diagramme correspondant aux 3 variables choisies est le suivant :

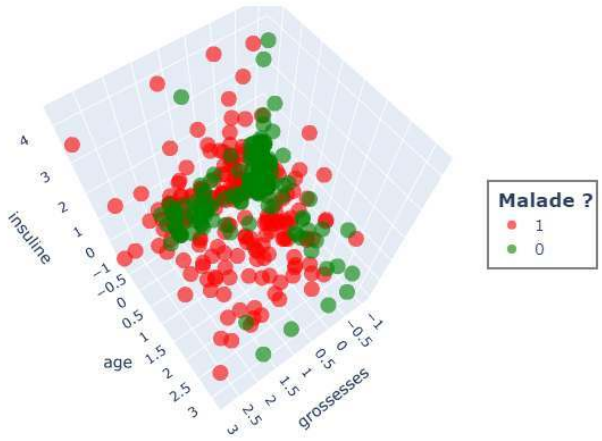


FIGURE 9 – Diagramme de dispersion

4. Méthode des k plus proches voisins

Le Machine learning ou apprentissage automatique se base sur un concept simple : l'utilisation des données pour que la machine apprenne et construise des modèles qui pourrons être appliqués sur des nouvelles données. Nous allons utiliser la méthode des k plus proches voisins pour construire notre modèle. Il est basé sur l'algorithme des k plus proches voisins et a pour objectif d'identifier les voisins les plus proches d'un point afin de pouvoir attribuer un statut à ce point. Dans notre cas, le point correspond à un patient pour lequel on veut savoir s'il est atteint du diabète ou pas. Comme discuté dans l'article [4], cette méthode repose sur la mesure des distances entre les points de données pour effectuer des prédictions.

4.1. Introduction à la méthode des k plus proches voisins

La méthode des k plus proches voisins est une technique de base en Machine learning qui nécessite seulement une valeur k et une mesure de distance. Nous allons voir comment elle fonctionne. Cette méthode est globalement composée de 4 étapes.

— **1ère étape : Sélection du nombre k de plus proches voisins.** Nous devons choisir un nombre k assez grand pour que notre modèle soit fiable. Pour qu'il soit le plus fiable possible, il faut que l'on prenne un paramètre k qui minimise le taux d'erreur de l'ensemble du test.

— **2ème étape : Calcul de la distance entre les données de la table Tn et l'entrée de la table Pn.** Elle consiste à calculer la distance entre toutes les données déjà classifiées de la table Tn et la nouvelle donnée entrée Pn. Il est possible d'utiliser plusieurs mesures de distances différentes. La distance que nous allons utiliser est la distance euclidienne entre deux points (x_1, y_1) et (x_2, y_2) qui représente le chemin en ligne droite le plus court entre deux points. Elle est définie comme :

$$distance_{euclidienne} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Nous prendrons ensuite une autre distance, la distance de Manhattan. Nous comparerons cette distance avec la distance euclidienne. Le but est alors d'essayer d'avoir une prédiction plus fiable qu'avec la distance euclidienne. La distance de Manhattan calcule la somme des différences absolues de coordonnées entre deux points. Mathématiquement, la distance de Manhattan entre deux points (x_1, y_1) et (x_2, y_2) est définie comme :

$$distance_{\text{Manhattan}} = |x_1 - x_2| + |y_1 - y_2|$$

Ça peut être utile d'utiliser la méthode de Manhattan quand les variables d'entrée ne sont pas de type similaire c'est-à-dire pour des données qui n'ont pas été mises dans la même échelle. Ce qui est le cas ici. (Voir [6])

— **3ème étape : Extraction des k plus proches voisins.** Notre modèle extrait ensuite les k données déjà classifiées qui ont la distance d la plus petite avec la nouvelle donnée entrée.

— **4ème étape : Test de la méthode** On doit alors tester la méthode avec nos patients. L'idée centrale est alors de prendre un nouvel individu dans la table 'Pn' et d'observer ses k plus proches voisins dans la table 'Tn'. On a alors deux possibilités. Si la majorité de ses voisins est malade, on prédira qu'il est malade, et dans le cas contraire, qu'il est sain.

4.2. Application de la méthode des k plus proches voisins

Appliquons la méthode des k plus proches voisins comme énoncé précédemment.

Nous devons tout d'abord choisir le nombre de k plus proches voisins Nous choisissons $k = 11$ pour obtenir un résultat assez fiable. On pourrait aussi tester d'autres nombres pour le paramètre k comme $k = 7$ et voir lequel donne la meilleure prédiction. Nous allons extraire les trois paramètres que nous avons déterminé dans le chapitre précédent pour réaliser une première prédiction : la grossesse, l'insuline et l'âge. Nous utilisons seulement ces variables pour nos prédictions. Nous allons, tout d'abord, le faire pour l'individu 0 de la population totale 'Tn' et l'individu 0 du groupe des patients 'Pn'. On associe à chaque individu dans 'Tn' sa distance au patient 0. Puis, on ordonne la table 'Tn' selon la distance au patient $p = 0$. On considère les 11 individus les plus proches, c'est-à-dire les 11 personnes qui ont la distance la plus petite avec le patient 0 :

	grossesses	glucose	pression	insuline	imc	K	age	M	distance
378	1.185192	1.072608	-0.740094	0.142453	-1.325189	0.182555	-0.616831	0	0.081397
59	1.185192	-0.332939	0.356116	-0.035546	1.653417	0.263949	-0.775399	1	0.181715
162	1.185192	-0.173218	-0.374691	-0.078266	1.582386	0.728298	-0.696115	1	0.206647
181	1.185192	0.178169	-0.374691	0.413012	-1.318690	-0.668360	-0.696115	1	0.284644
249	1.468731	1.519828	-0.800995	0.206533	-0.331803	0.484469	-0.616831	0	0.303461
204	1.468731	-1.099601	-0.740094	0.242133	0.753170	-0.487350	-0.775399	0	0.314077
102	0.901654	-0.205162	0.051613	0.021413	-0.029408	0.204744	-0.775399	1	0.314368
340	0.901654	1.647605	0.538818	0.021413	-1.069393	-0.595446	-0.616831	0	0.314807
131	1.468731	0.465667	0.417017	-0.014187	0.657378	0.975584	-0.696115	1	0.316036
58	1.185192	0.753166	-0.070188	0.455732	0.361753	2.131227	-0.696115	1	0.327363
80	1.468731	1.998992	0.173414	0.291973	-0.485054	-0.532658	-0.775399	1	0.335362

FIGURE 10 – Tableau de Prédiction du patient 0

On peut voir que le patient $p = 0$ a des paramètres semblables à 7 individus malades parmi les 11 les plus proches. Comme $7 > 11/2$, cela veut dire que le patient 0 a des caractéristiques semblables avec plus de la moitié des 11 plus proches voisins considérés. On en déduit que ce patient a plus de paramètres similaires avec les individus malades que les sains et on prédit que ce patient est diabétique. Nous devons alors généraliser cette méthode pour tous les patients. Pour cela, on va alors créer une fonction de prédiction qui associe à chaque patient p dans 'Pn' les 11 individus 'Tn' les plus proches. Si parmi ces 11 individus plus de 5 patients sont malades, on suppose que le patient p est malade. Si on vérifie avec notre fonction de prédiction le pronostic du patient $p = 0$, notre fonction nous indique que celui-ci est bien prédit malade en retournant le nombre 1. On ajoute, pour finir, dans la table 'Pn' une colonne 'T' qui indique la prédiction à partir de notre fonction de prédiction. Ici, on a utilisé la distance euclidienne pour appliquer la méthode des k plus proches voisins. Nous pouvons aussi chercher à utiliser une autre distance comme la distance de Manhattan pour essayer d'avoir une prédiction plus fiable. On refait le même procédé et on remplace dans notre fonction de prédiction la distance euclidienne par la distance Manhattan. On ajoute dans la table 'Pn', une colonne 'T2' qui désigne la nouvelle prédiction obtenue avec la distance de Manhattan. On obtient le tableau suivant :

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0
...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0

FIGURE 11 – Tableau de Prédiction

Nous pouvons aussi tester notre fonction de prédiction avec la distance euclidienne et celle de Manhattan et un k plus petit comme $k = 7$. On applique alors le même procédé qu'avant et on rajoute les deux nouvelles prédictions à la table 'Pn'. Il vient alors la nécessité de pouvoir valider notre modèle de prédiction.

5. Evaluation de notre modèle

Après avoir tester la méthode des k plus proches voisins, nous devons vérifier si notre modèle est performant.

5.1. Croisement des variables

Nous allons faire le croisement des variables afin de pouvoir comparer les données réelles pour la variable 'M' et à celle prédite par notre modèle. Nous pouvons nous représenter les valeurs prédictives à l'aide d'une matrice de confusion. Elle possède une première colonne qui distingue les cas des personnes malades et non malades et deux colonnes pour les prédictions.

T	0	1	All
M			
0	44	6	50
1	9	41	50
All	53	47	100

FIGURE 12 – Matrice de Confusion

Nous constatons que 44 personnes non-malades ont été correctement prédites comme négatives et 41 personnes malades ont été correctement prédites comme positives. Cela signifie que 44 personnes sur 50 non-malades ont été correctement identifiées, ainsi que 41 patients sur 50 malades. Ces résultats indiquent une assez bonne performance du modèle.

Cependant, il est important de noter que 6 personnes non-malades ont été incorrectement prédites comme malades. Bien que les traitements pour le diabète, tels que les modifications alimentaires et les médicaments comme l'insuline, soient généralement sûrs lorsqu'ils sont administrés correctement, ils peuvent néanmoins entraîner des effets secondaires et ne devraient pas être utilisés inutilement.

En revanche, il est plus préoccupant de constater que 9 personnes malades ont été incorrectement prédites comme négatives. Cette erreur pourrait être dangereuse dans le contexte de la détection du diabète. En effet, des complications peuvent survenir si les malades ne sont pas traités, avec des symptômes tels que la perte de poids, une vision trouble, et d'autres complications graves. Il serait ainsi souhaitable de réduire le nombre de non-malades incorrectement prédits comme malades et le nombre de malades incorrectement prédits comme négatifs pour éviter ces risques et garantir que les patients reçoivent les soins appropriés.

5.2. Mesure de fiabilité des prédictions

Dans cette partie, nous allons évaluer la fiabilité de notre modèle. Nous allons considérer les événements suivants :

- M : la personne est malade,
- T : son test est positif.

Nous noterons respectivement $\bar{M}$ l'événement "la personne n'est pas malade" et $\bar{T}$ l'événement "le test est négatif".

Pour mesurer la pertinence du classement, nous avons besoin des élements suivants régroupés dans une *Matrice de confusion* :

— vrai positif : MT (malade avec un test positif), i.e. $M = 1$ et $T = 1$
— vrai négatif : $\bar{M}\bar{T}$ (sain avec un test négatif) i.e. $M = 0$ et $T = 0$
— faux positif : $\bar{M}T$ (sain avec un test positif) i.e. $M = 0$ et $T = 1$
— faux négatif : $M\bar{T}$ (malade avec un test négatif) i.e. $M = 1$ et $T = 0$

5.3. Les Métriques

5.3.1. Introduction aux métriques

Nous allons également utiliser des métriques qui sont des variables quantitatives qui caractérisent des valeurs numériques et qui sont couramment utilisées pour évaluer la performance d'un modèle en apprentissage automatique. Voici les métriques que nous allons utiliser, accompagnées de leur formule et leur signification :

— Accuracy :

$$\frac{MT + \bar{M}\bar{T}}{MT + \bar{M}\bar{T} + \bar{M}T + M\bar{T}}$$

— Précision :

$$\frac{MT}{MT + \bar{M}T}$$

— Sensibilité :

$$\frac{MT}{MT + M\bar{T}}$$

— f1-score :

$$2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

— Spécificité :

$$\frac{\bar{M}\bar{T}}{\bar{M}\bar{T} + \bar{M}T}$$

Ces formules sont extraites de [5]. L'Accuracy nous permet de connaître la proportion de bonnes prédictions par rapport à toutes les prédictions. L'opération est simplement : $\frac{\text{Nombre de bonnes prédictions}}{\text{Nombre total de prédictions}}$ La précision correspond au nombre de malade correctement prédit sur le nombre total de personnes prédits malades. Il permet de mesurer le coût des faux positifs, c'est-à-dire d'estimer le nombre d'individus détectés par erreur. La sensibilité (aussi appelée Recall) correspond à la probabilité d'être positif si on est malade. Elle mesure la proportion des prédictions positives correctement identifiées (qui étaient réellement positives). Ce calcul permet d'estimer combien de vrais positifs nous avons réussi à déterminer et ceux qu'on n'est pas parvenu à détecter. De plus, on souhaite avoir un Recall (Rappel) = 1 où le Recall représente $\frac{\text{le nombre de positifs bien prédit (Vrai Positif)}}{l'ensembledespositifs(VraiPositif+FauxNégatif)}$ et permet de savoir le pourcentage de positifs bien prédit par notre modèle. Le f1-score combine la précision et le rappel. Il est plus intéressant que la précision car le nombre de vrais négatifs n'est ici pas pris en compte. Dans le cas de données déséquilibrées, nous avons une majorité de vrais négatifs qui faussent complètement notre perception de la performance de l'algorithme. La spécificité représente la probabilité d'avoir un test négatif sachant qu'on n'est pas malade. La spécificité mesure, à l'inverse de la sensibilité, la proportion d'éléments négatifs correctement identifiés.

Nous pouvons chercher à les calculer pour estimer la précision de notre modèle.

5.3.2. Calcul des métriques

Maintenant que nous avons les valeurs de MT, $\bar{M}\bar{T}$, $\bar{M}T$ et $M\bar{T}$, nous pouvons calculer les métriques énoncés précédemment pour pouvoir déterminer la fiabilité de notre modèle de prédictions. Après avoir fait le calcul de l'accuracy, la précision, la sensibilité et la spécificité, nous allons analyser les résultats.

Tout d'abord, nous avons calculer l'accuracy de notre modèle en utilisant la formule mentionnée précédemment. Le résultat obtenu était de 0.85, ce qui indique que 85 % des prédictions étaient correctes.

Ensuite, nous calculons la précision. Nous avons obtenu un résultat de 0.8723404255319149 , ce qui signifie que 8723404255319149 % des cas positifs prédits étaient réellement positifs. Nous avons mesuré la sensibilité de notre modèle, aussi connue sous le nom de Recall, en utilisant la formule spécifiée précédemment. Nous avons obtenu un résultat de 0.82, ce qui montre que 82 % des vrais positifs ont été correctement identifiés parmi tous les vrais positifs. La sensibilité est supérieure à 0.5. On peut considérer que le test est performant s'il est utilisé pour des individus

testés positifs. Le f1-score a été calculé en combinant la précision et le rappel, comme indiqué dans l'introduction. Nous avons obtenu un f1-score de 0.8453608247422681. Notre f1-score est plutôt élevé. Ce qui montre une bonne performance globale du modèle. Enfin, nous avons évalué la spécificité de notre modèle en utilisant la formule mentionnée précédemment. Nous avons obtenu un résultat de 0.88, montrant que 88 % des vrais négatifs ont été correctement identifiés parmi tous les vrais négatifs. La spécificité est proche de 1. Le test semble ainsi performant lorsqu'il est utilisé sur des personnes négatifs. Notre modèle présente une performance satisfaisante dans la prédiction du diabète avec des scores élevés dans toutes les métriques. Cependant, pour valider notre modèle, nous devons nous assurer que la prédiction soit bonne quelque soit la proportion de malade. Supposons maintenant que la **prévalence du diabète** (c'est-à-dire la probabilité dêtre malade) $\mathbb{P}(M)$ dans la population est égale à 0.1 :

$$\mathbb{P}(M) = 0.1, \qquad \mathbb{P}(\bar{M}) = 0.9$$

On utilise la méthode de prédiction obtenue par Machine learning décrite plus haut. Si une personne voit son test positif, nous allons chercher quelle est la probabilité qu'elle soit malade. Cette probabilité est appelée la **valeur prédictive du test**, i.e. la valeur de $\mathbb{P}_T(M)$. D'après ce qui précède, la sensibilité est égale à 0.82 dans notre cas. On a alors :

$$\mathbb{P}_M(T) = 0.1 \times 0.82, \quad \mathbb{P}_M(\bar{T}) = 0.1 \times 0.18, \quad \mathbb{P}_{\bar{M}}(T) = 0.9 \times 0.12, \quad \mathbb{P}_{\bar{M}}(\bar{T}) = 0.9 \times 0.88.$$

On a :

$$\mathbb{P}(T) = \mathbb{P}_M(T) + \mathbb{P}_{\bar{M}}(T) = 0.1 \times 0.82 + 0.9 \times 0.12 = 0.19$$

.

La valeur prédictive du test est :

$$\mathbb{P}_T(M) = \frac{\mathbb{P}(T \cap M)}{\mathbb{P}(T)} = \frac{\mathbb{P}_M(T)}{\mathbb{P}(T)} = \frac{0.1 \times 0.82}{0.19} = 0.43$$

.

5.4. Valeur prédictive en fonction de la prévalence

Quelle que soit la prévalence $p = \mathbb{P}(M)$ du diabète, la valeur prédictive du test est :

$$\mathbb{P}_T(M) = \frac{\mathbb{P}(T \cap M)}{\mathbb{P}(T)} = \frac{\mathbb{P}_M(T)}{\mathbb{P}_M(T) + \mathbb{P}_{\bar{M}}(T)} = \frac{p \times 0.82}{p \times 0.82 + (1-p) \times 0.12} = \frac{0.82p}{0.12 + 0.70p}.$$

On affiche le graphe de :

$$p \mapsto \mathbb{P}_T(M)$$

qui correspond à la valeur prédictive du test en fonction de la prévalence de la maladie.

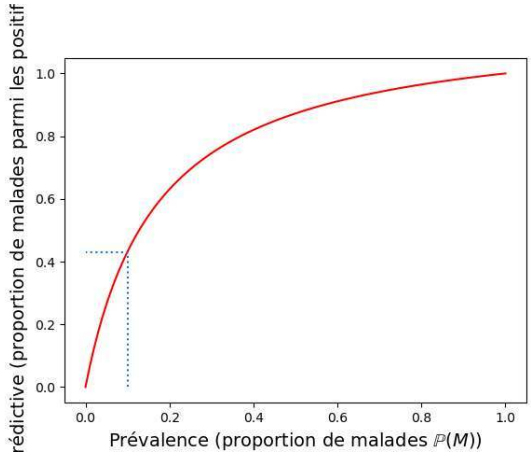


FIGURE 13 – Valeur prédictive du test en fonction de la prévalence de la maladie

Nous pouvons voir que lorque la prévalence est élevée la valeur prédictive est assez élevée. Cependant, on remarque que lorsque la prévalence est faible, la valeur prédictive est plutôt faible. Cela signifie que quand la proportion de malades est basse, la qualité de notre prédiction est faible. Lorsque la maladie est rare, il serait mieux que la valeur prédictive soit plus importante afin de réduire les erreurs de pronostic et de minimiser le risque de manquer des cas de maladie. Essayons de voir si avec la distance Manhattan, la valeur prédictive est plus élevée pour une prévalence faible. On refait le même procédé avec la distance Manhattan. On crée une fonction de prédiction que utilise la méthode des k plus proches voisins mais cette fois-ci en prennant en compte la distance de Manhattan. Puis, on affiche notre matrice de confusion. On calcule la sensibilité, la spécificité, la précision et le f1-score pour la distance Manhattan. En les rangeant avec les valeurs pour la distante euclidienne on trouve :

	spécifité	sensibilité	précision	f1_score
Manhattan	0.84	0.82	0.836735	0.828283
Euclidienne	0.88	0.82	0.872340	0.845361

FIGURE 14 – Métriques en fonction de la distance euclidienne ou de la distance de Manhattan

On observe que la sensibilité est la même pour les deux distances. La spécificité est plus basse avec la distance de Manhattan qu'avec la distance euclidienne (spécificitéManhattan=0.84 $<$ spécificitéeuclidienne=0.88). Cela signifie que moins de vrais négatifs ont été correctement identifiés parmi tous les vrais négatifs avec la distance de Manhattan. Le f1-score et la précision sont légèrement plus faibles aussi pour cette distance. De plus, si on calcule l'accuracy de la distance de Manhattan il est aussi légèrement plus faible.

Quelle que soit la prévalence $p = \mathbb{P}(M)$ du diabète, la valeur prédictive du test avec la distance de Manhattan est :

$$\mathbb{P}_T(M) = \frac{\mathbb{P}(T \cap M)}{\mathbb{P}(T)} = \frac{\mathbb{P}_M(T)}{\mathbb{P}_M(T) + \mathbb{P}_{\bar{M}}(T)} = \frac{p \times 0.82}{p \times 0.82 + (1-p) \times 0.16} = \frac{0.82p}{0.16 + 0.66p}.$$

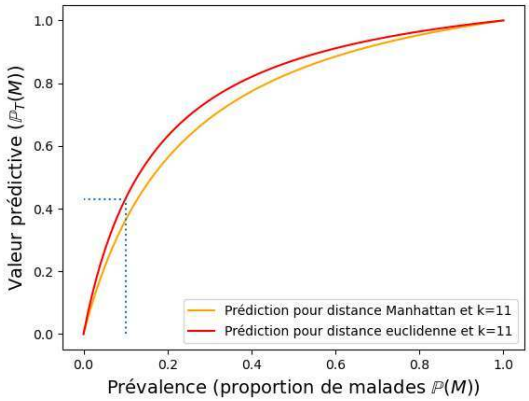


FIGURE 15 – Valeur prédictive du test en fonction de la prévalence de la maladie

Nous pouvons voir que pour une prévalence faible, la valeur de prédiction est la même pour les deux distances. Pour une valeur plus élevée, la valeur de la prédiction pour la distance Manhattan est plus faible que celle pour la distance euclidienne. Utiliser la distance euclidienne semble être le plus adéquat pour notre modèle si on choisit $k = 11$.

Nous pouvons aussi changer le nombre de k plus proches voisins et prendre $k = 7$. On refait le même processus qu'avant en remplaçant 11 par 7 dans notre fonction de prédiction et en utilisant la distance euclidienne.

On obtient les valeurs suivantes :

	spécifité	sensibilité	précision	f1_score
k = 11	0.88	0.82	0.87234	0.845361
k = 7	0.96	0.64	0.94000	0.760000

FIGURE 16 – Métriques en fonction de la distance euclidienne pour k=7 et k=11

Nous pouvons remarquer que pour $k = 7$, la sensibité est plutôt faible par rapport à la sensibilité pour $k = 11$. Cela signifie que le taux des vrais positifs a été moins bien identifié parmi tous les vrais positifs pour $k = 7$ que pour $k = 11$. Cependant, la spécifité et la précision sont plus élevés dans le cas où on prend $k = 7$. Cela signifie que pour $k = 7$, le test est plus performant sur des personnes négatifs (car la spécificité se rapproche plus de un) mais que la probabilité d'être positif si on est malade est plus faible que pour $k = 11$. Si nous choisissons $k = 7$, nous obtenons donc un meilleur résultat permettant de diminuer le coût des vrais négatifs et ainsi diminuer le nombre de patients diagnostiqués négatifs alors qu'ils sont malades. Si on affiche la courbe de la valeur prédictive du test en fonction de la prévalence de la maladie pour $k = 7$, nous pouvons observer que la valeur prédictive est beaucoup plus élevée quand la prévalence est faible et légèrement plus grande quand la prévalence est grande. Ce résultat montre qu'avec $k = 7$, la probabilité d'être malade augmente si l'on est testé positif, améliorant ainsi la performance du modèle.

6. Conclusion

Tandis que de plus en plus de méthodes émergent en Machine learning, il est essentiel de tester la fiabilité de ces modèles afin de les appliquer dans une variété de domaines tels que le dépistage du diabète de type II. La construction d'un modèle efficace nécessite une analyse approfondie des données et la compréhension des variables qui influencent le plus la variable cible, en utilisant par exemple des matrices de corrélation ou des diagrammes de dispersion. Nous avons sélectionné trois paramètres principaux : la grossesse, l'insuline et l'âge. Ces paramètres sont souvent influencés par des facteurs de risque majeurs du diabète de type II, comme la sédentarité et l'alimentation. En prenant en compte ces paramètres, nous avons pu améliorer la précision de nos prédictions.

Parmi les nombreuses méthodes disponibles, celle des k plus proches voisins se distingue par sa précision et sa facilité d'application. Dans notre étude, nous avons utilisé la méthode des k plus proches voisins avec k = 11 pour créer un modèle de prédiction basé sur le calcul de la distance euclidienne. Pour évaluer la performance de notre modèle, nous avons calculé divers métriques telles que l'accuracy et la précision. Nous avons également expérimenté avec la distance de Manhattan pour améliorer la fiabilité de nos prédictions. Cependant, les résultats obtenus étaient moins bons que ceux avec la distance euclidienne. Cette observation souligne que l'efficacité de la méthode des k plus proches voisins dépend fortement du choix de la métrique de distance utilisée.

Notre analyse a montré que la méthode des k plus proches voisins est particulièrement efficace lorsque la prévalence de la maladie est élevée, mais présente des limites lorsque la proportion de malades est faible. En ajustant le nombre de voisins k, nous avons pu améliorer la précision de nos prédictions. Par exemple, nous avons constaté que la distance euclidienne avec k = 7 offrait une meilleure performance prédictive comparée à k = 11 pour cette même distance.

Ces résultats démontrent l'importance de choisir judicieusement le paramètre k pour optimiser les performances du modèle des k plus proches voisins. D'autres distances, comme celle de Chebychev, pourraient encore modifier les résultats.

Ces résultats cumulés avec d'autres prédictions sont illustrés dans le graphique suivant, qui montre la performance prédictive en fonction de la proportion de malades :

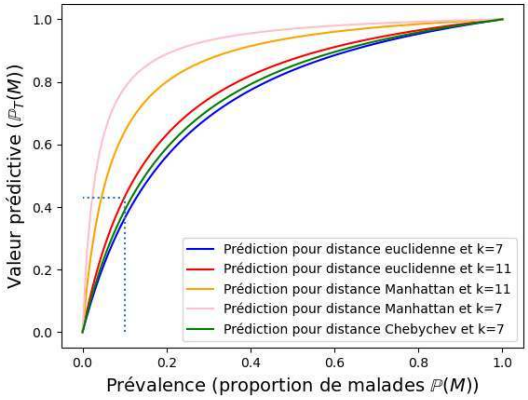


FIGURE 17 – Diagramme de dispersion

Par exemple, le graphique montre clairement que la distance de Manhattan avec k = 7 offrait une meilleure performance que k = 11, tandis que la distance euclidienne était plus efficace pour

k = 11. D'autres distances, comme celle de Chebychev, pourraient encore modifier les résultats. C'est donc la distance de Manhattan avec k = 7 qui nous a donné le modèle de prédiction le plus efficace.

Pour aller plus loin, il serait intéressant d'améliorer notre modèle en testant d'autres valeurs de k, en explorant différentes mesures de distance, ou en considérant davantage de paramètres. De plus, nous pourrions développer d'autres modèles de prédiction en utilisant des méthodes alternatives telles que les forêts d'arbres décisionnels ou les réseaux de neurones, afin de comparer leur efficacité et fiabilité dans le dépistage du diabète de type II.

Pour finir, je souhaite remercier Mme Faccanoni du laboratoire Imath pour son engagement, son aide, le temps qu'elle m'a consacré et ses précieux conseils tout au long de la réalisation de mon projet de recherche sur le Diabète de type II.

Références

[1] Frédéric Bro et Chantal Remyl. *Problème 35 de "Python et Pandas et les 36 problèmes de Data Science"*. 1re éd. Ellipses, déc. 2021. 542 p.

[2] « Charge mondiale, régionale et nationale du diabète de 1990 à 2021, avec projections de prévalence jusqu'en 2050 ». In : *THE LANCET* (2023). URL : https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(23)01301-6/fulltext.

[3] Emmanuel Jakobowicz. « Le box-plot ou la fameuse boîte à moustache ». In : *Stat4decision* (2020). URL : https://www.stat4decision.com/fr/le-box-plot-ou-la-fameuse-boite-a-moustache.

[4] « L'algorithme des k-plus proches voisins (kNN) en Python ». In : (2024). URL : https://fr.python-3.com/?p=232.

[5] Beranger Natanelic. « ML : Précision F1-Score, Courbe ROC, que choisir ? » In : *Medium* (2020). URL : https://beranger.medium.com/ml-accuracy-pr.

[6] Rajesh. « Qu'est-ce que Manhattan Distance dans le Machine Learning ? Programmable bibliographies and citations ». In : *Datapro* (2024). URL : https://datapro.blog/author/rajesh/.

A. Annexe

Nous présentons ici les codes utilisés pour réaliser ce rapport, accompagnés de courtes descriptions de leur fonction.

In [2]:
```
import pandas as pa
from seaborn import boxplot
import pylab as pl
import seaborn as sns
import pylab as pl
import plotly.express as px
import numpy as np
```

Détection du diabète

Nous allons voir un exemple de mise en oeuvre d'un classificateur d'arbre de décision pour le problème du diabète. L'objectif de l'ensemble de données est de prédire avec un diagnostique si un patient est atteint du diabète. Plusieurs critères ont été sélectionnés parmi une base de données plus grande. En particulier, tous les patients ici sont des femmes âgées d'au moins 21 ans. Les patients sont également d'origine indienne Pima, population la plus touchée dans le monde par le diabète de type 2.

Les données proviennents de l'étude Pima Indians Diabetes faite par l'Université de Californie, School of Information and Computer Science.

Les ensembles de données comprennent plusieurs paramètres et une variable cible, « M ». Chaque ligne représente un patient et les colonnes sont les variables prédictives et la variable cible:

- Grossesses: nombre de fois enceintes
- Glucose: concentration en glucose plasmatique 2 heures dans un test de tolérance au glucose par voie orale
- Pression: pression artérielle diastolique (mm Hg)
- Insuline: insuline sérique de 2 heures (mu U / ml)
- Imc: indice de masse corporelle (poids en kg / (taille en m)^2)
- K: coefficient lié à la présence du diabète parmi ses parents et proches
- Age: age (ans)
- M: variable de classe (0 ou 1) qui indique si l'individu est sain (0) ou malade (1).

Nous construisons un modèle qui va faire des prédictions, nous devons donc trouver un moyen d'évaluer la qualité de ces prédictions. Étant donné que les prédictions par définition ne concernent que des données inédites, nous ne pouvons pas dépendre des données utilisées pour créer le modèle. Nous devons d'abord diviser le jeu de données en deux parties non croisées: les données de formation qui seront utilisées pour

construire le modèle et les données de test pour évaluer les prédictions du modèle. Nous utiliserons l'ensemble d'entrainement pour construire notre modèle pour les arbres de decision. Puis on évaluera son score sur l'ensemble de test.

Dans notre cas il existe déjà deux fichiers séparés (avec des données déjà bien préparées). Si on a un seul fichier, la division de l'ensemble de données en un ensemble d'entrainement et un ensemble de test peut être réalisée par le module sklearn comme ci-dessous:

Données d'entrainement

Dans la table T nous importons les données permettant d'apprendre à reconnaître une personne diabétique.

In [3]:
```
T =pa.read_csv('diabete_population (1).csv')
```

In [4]:
```
paramètres=[c for c in T.columns]         #énumère les paramètres
print('les paramètres étudiés sont:', paramètres)
print('(Nombre d individus, Nombre paramètres)=', T[paramètres].shape) # 400 individus classés selon 7 paramètres
```

```
les paramètres étudiés sont: ['grossesses', 'glucose', 'pression', 'insuline', 'imc', 'K', 'age', 'M']
(Nombre d individus, Nombre paramètres)= (400, 8)
```

Résumé statistique de chaque colonne de notre tableau de données:

In [5]:
```
T.describe()
```

Out[5]:

	grossesses	glucose	pression	insuline	imc	K	age	M
count	400.000000	400.000000	400.00000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	3.820000	107.422500	72.15250	155.992500	31.853995	0.427129	30.780000	0.500000
std	3.526854	31.304535	16.42021	140.449859	9.642392	0.425077	12.612863	0.500626
min	0.000000	45.000000	27.00000	14.000000	18.217352	0.078656	21.000000	0.000000
25%	1.000000	85.000000	60.00000	45.000000	21.561065	0.137473	22.000000	0.000000
50%	3.000000	105.000000	74.00000	127.500000	31.839371	0.198598	24.000000	0.500000
75%	6.000000	125.000000	84.00000	218.000000	39.107978	0.666726	38.250000	1.000000
max	14.000000	190.000000	116.00000	758.000000	55.421694	2.278046	71.000000	1.000000

Nous pouvons détecter et compter les valeurs manquantes de notre jeu de données.

In [6]:
```
T.isnull().sum()
```

Out[6]:
```
grossesses    0
glucose       0
pression      0
insuline      0
imc           0
K             0
age           0
M             0
dtype: int64
```

On remarque qu'il n'y a pas de valeurs manquantes dans notre jeu de données.

On peut également choisir de se focaliser sur des données statistiques spécifiques

In [7]:
```
Moy=T[paramètres].mean()
Moy
```

Out[7]:
```
grossesses      3.820000
glucose       107.422500
pression       72.152500
insuline      155.992500
imc            31.853995
K               0.427129
age            30.780000
M               0.500000
dtype: float64
```

On a ici les moyennes de chaque paramètre.

On peut afficher les premières lignes de notre tableau:

In [8]:
```
T =pa.read_csv('diabete_population (1).csv') #Test
T.head()
```

Out[8]:

	grossesses	glucose	pression	insuline	imc	K	age	M
0	2	104	75	28	39.020106	0.145273	35	1
1	10	121	64	517	30.672607	0.154211	43	1
2	2	153	59	87	26.295668	1.029407	22	1
3	9	145	47	226	46.077807	0.086713	37	1
4	3	102	76	315	28.335339	0.109751	21	1

Données de test

Dans la table P, nous importons les données pour mesurer la fiabilité de prédiction d'une personne diabétique via notre modèle.

In [9]:
```
P = pa.read_csv('diabete_patients.csv') #Vérification de la fiabilité du test de prédiction
P.head()
```

Out[9]:

	grossesses	glucose	pression	insuline	imc	K	age	M
0	8	107	68	174	25.048805	0.125076	22	1
1	0	72	84	253	29.623036	0.837026	44	1
2	2	173	73	611	31.119992	1.099852	60	1
3	3	106	74	204	27.036600	0.100967	21	1
4	9	136	62	155	38.134675	0.500380	22	1

1.Exploration des données

Objectif:

Nous cherchons ici à comprendre les données et trouver les paramètres qui influe le plus sur la variable cible ("M")

In [10]:
```
print("Combien de malades et combien de sains?")
T['M'].value_counts()
```

```
Combien de malades et combien de sains?
1    200
0    200
Name: M, dtype: int64
```

Out[10]:

Parmi T, il y a 200 malades et 200 non malades. Il y a donc autant de malades que de personnes saines.

Nous pouvons également chercher à appliquer un masque pour filter les valeurs qui nous semble le plus intéressante. Nous pouvons par exemple, créer un masque qui filtre les IMC supérieurs à 25 de notre jeu de données.

In [11]:
```
mask = T.imc>25
extraction = T[mask]
print(f"{extraction.shape[0]} sur {T.shape[0]} individus ont un IMC>25")
```

```
272 sur 400 individus ont un IMC>25
```

D'après E-santé l'IMC d'une femme se situe en moyenne entre 18.5 et 25. Ici, il y a donc plus de la moitié de la population étudiée qui a un IMC supérieur à la moyenne. Ce qui peut nous faire penser que l'IMC est un paramètre indicateur du diabète.

Nous pouvons aussi chercher à voir si des paramètres comme le nombre de grossesses non pas un lien avec le diabète.

In [12]:
```
print("Combien de grossesses en moyenne?")
T.grossesses.mean()
```

```
Combien de grossesses en moyenne?
3.82
```

Out[12]:

Le nombre moyen de grossesses semble élevé,on pourrait ici essayer de voir à l'aide d'un diagramme à quel point le nombre de grossesses influencerai la prédominance au diabète.

In [13]:
```
labels = T['grossesses'].value_counts().index
values = T['grossesses'].value_counts().values
px.pie(T,labels=labels, values=values,title='Grossesses',names=labels,color_discrete_sequence=px.colors.sequential.RdBu,hole=0.3)
```

```
C:\Users\33767\anaconda3\lib\site-packages\plotly\express\_core.py:137: FutureWarning: Support for multi-dimensional indexing
(e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  return args["labels"][column]
```

Ce diagramme montre que si on exclu les femmes sans enfants, plus le nombre de grossesses augmente, plus le nombre de femmes diabètiques augmente.

Calcul de la Moyenne et de l'écart-type

On exclue le paramètre M de nos paramètres:

In [14]:
```
paramètres=[c for c in T.columns if c!="M"]          #énumère les paramètres
paramètres
```

Out[14]:
```
['grossesses', 'glucose', 'pression', 'insuline', 'imc', 'K', 'age']
```

In [15]:
```
Moy=T[paramètres].mean()        #calcul de la moyenne pour chaque paramètre
Moy
```

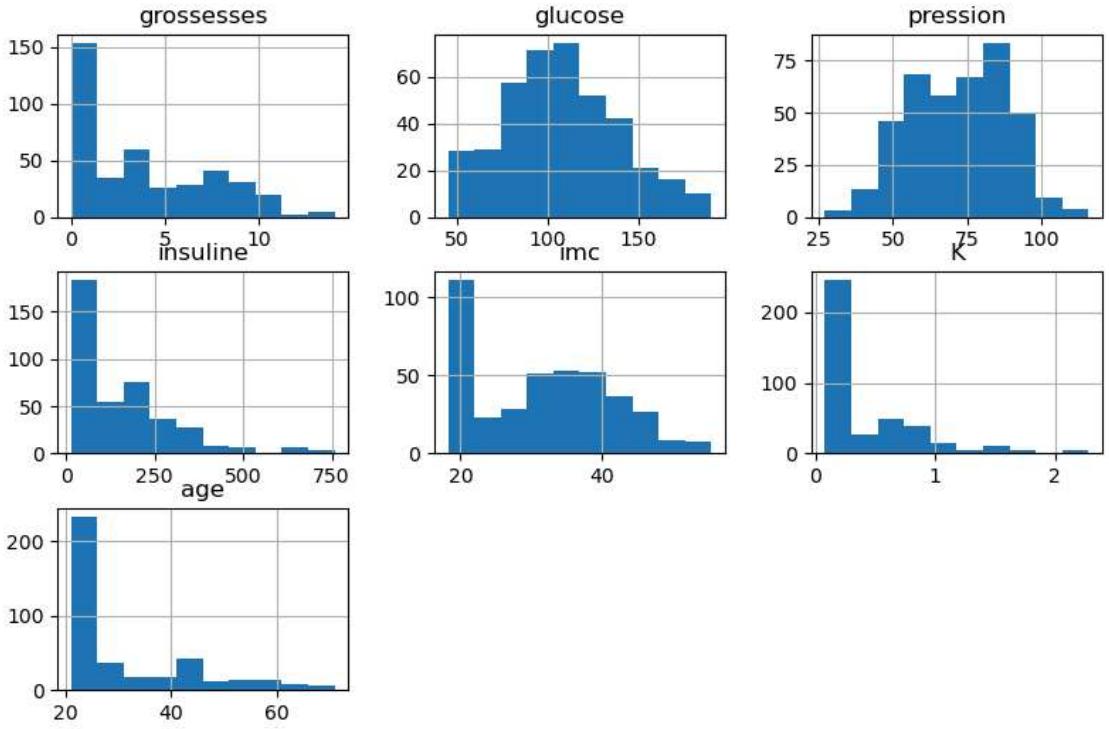
Out[15]:
```
grossesses      3.820000
glucose       107.422500
pression       72.152500
insuline      155.992500
imc            31.853995
K               0.427129
age            30.780000
dtype: float64
```

In [16]:
```
ect = T[paramètres].std()
ect
```

Out[16]:
```
grossesses      3.526854
glucose        31.304535
pression       16.420210
insuline      140.449859
imc             9.642392
K               0.425077
age            12.612863
dtype: float64
```

Nous pouvons aussi visualiser la répartion des individus en fonction des paramètres.

In [17]:
```
T[paramètres].hist(figsize = (9.5,6))
pl.show()
```



In [18]:
```
T.groupby(['M']).mean()
```

Out[18]:

M	grossesses	glucose	pression	insuline	imc	K	age
0	2.205	105.110	70.890	114.075	29.342952	0.342397	27.005
1	5.435	109.735	73.415	197.910	34.365038	0.511860	34.555

On arrondit la moyenne du nombre de grossesses pour avoir des valeurs plus cohérentes avec la réalité (le nombre de grossesses ne peut pas être une fraction dans la réalité

In [19]:
```
T.groupby('M')['grossesses'].mean().round()
Groupby=T.groupby(['M']).mean()
Groupby.iloc[0,0]=2.0
Groupby.iloc[1,0]=5.0
Groupby
```

Out[19]:

M	grossesses	glucose	pression	insuline	imc	K	age
0	2.0	105.110	70.890	114.075	29.342952	0.342397	27.005
1	5.0	109.735	73.415	197.910	34.365038	0.511860	34.555

On peut voir que les personnes malades ont des valeurs beaucoups plus élevées que les non malades pour chaque paramètre. En particulier, le taux d'insuline semble beaucoup plus élevé chez les malades.

Calculons l'écart-type

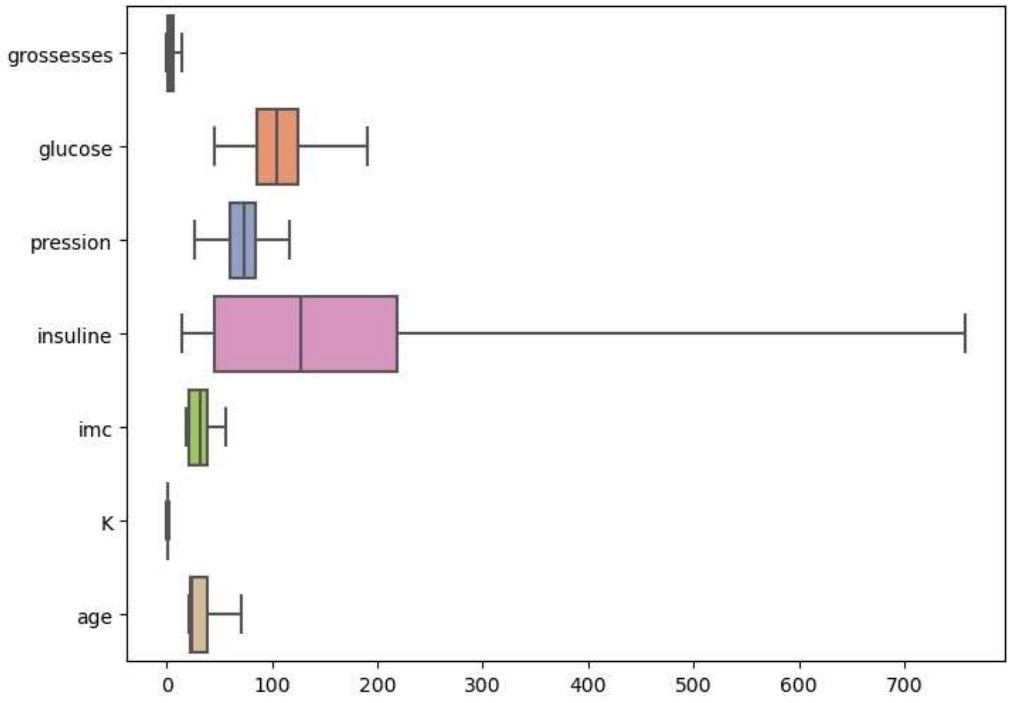
In [20]:
```
Sig=T[paramètres].std(ddof=0)
Sig
```

Out[20]:
```
grossesses      3.522442
glucose        31.265380
pression       16.399672
insuline      140.274187
imc             9.630331
K               0.424546
age            12.597087
dtype: float64
```

Nous pouvons, ici, utiliser Un box plot pour analyser la distribution des données. Il nous permettra d'avoir un résumé graphique pour identifier l'asymétrie de l ensemble de données.

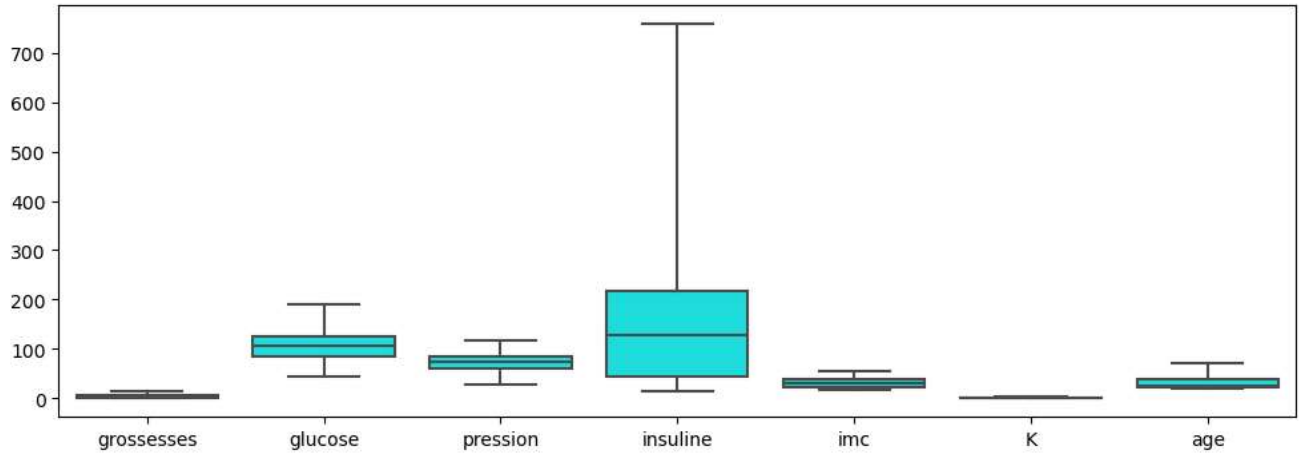
Boîte à moustaches:

In [21]:
```
pl.figure(figsize=(8,6))
boxplot(data=T[paramètres], color='lightblue', whis=[0,100],orient='h',palette='Set2');
```



Une boîte à moustache est composée d'une boîte correspondant à la plage dans laquelle se situent 50 % des données. Ses cotés gauche et droit représentent les quartiles des données. Elle est coupée en deux par une ligne qui représente la médiane. De plus, des barres appelées "moustaches" représentent les lignes qui se situent à gauche et à droite de la boîte. Elles indiquent les valeurs minimales et maximales, et définissant les bornes des moustaches.

In [22]:
```
pl.figure(figsize = (12,4))
boxplot(data =T[paramètres],
color='cyan',
whis = [0, 100])
pl.savefig("boxplotzoom.png");
#Dispersion entre taux d'insuline et de glucose
```



On remarque que dans notre cas l'insuline et le glucose n'ont pas le même ordre de grandeur. Nous devons alors normaliser les données.

La dose d'insuline d'un individu est de 200 mUI/L, sa valeur normalisée est donc

In [23]:
```
(200-Moy['insuline'])/ect['insuline']
```

Out[23]:
```
0.3133324615663732
```

Normalisation : centrage et réduction des données

Pour pouvoir comparer les différents paramètres (qui n'ont pas les mêmes unités), nous allons standardider les caractéristiques autour du centre et de 0 et avec un écart-type de 1 en les normalisant.

On normalise alors chaque colonne de `T` (sauf la colonne `M`). Il s'agit de faire en sorte que la moyenne soit nulle et que la variance soit égale à 1.

In [24]:
```
Tn = (T[paramètres]-Moy)/ect
Tn
```

Out[24]:

	grossesses	glucose	pression	insuline	imc	K	age
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399
...	...	...	...	...	...	...	...
395	0.051037	-1.003768	-0.922796	-0.790264	-1.210104	0.315929	-0.696115
396	-0.799580	-1.163490	-1.105497	-0.932664	-1.078406	-0.583855	-0.696115
397	-0.799580	0.401779	0.538818	0.235013	-1.210369	-0.670262	-0.696115
398	-0.799580	-0.939880	-0.374691	-0.120986	-1.090458	-0.682149	-0.458262
399	-0.799580	0.912887	0.599718	-0.982504	-0.039462	-0.635427	0.017443

400 rows × 7 columns

Pour voir si les valeurs sont normalisées nous pouvons véridier si la moyenne est nulle et si la variance égale à 1:

In [25]:
```
Tn.std()
```

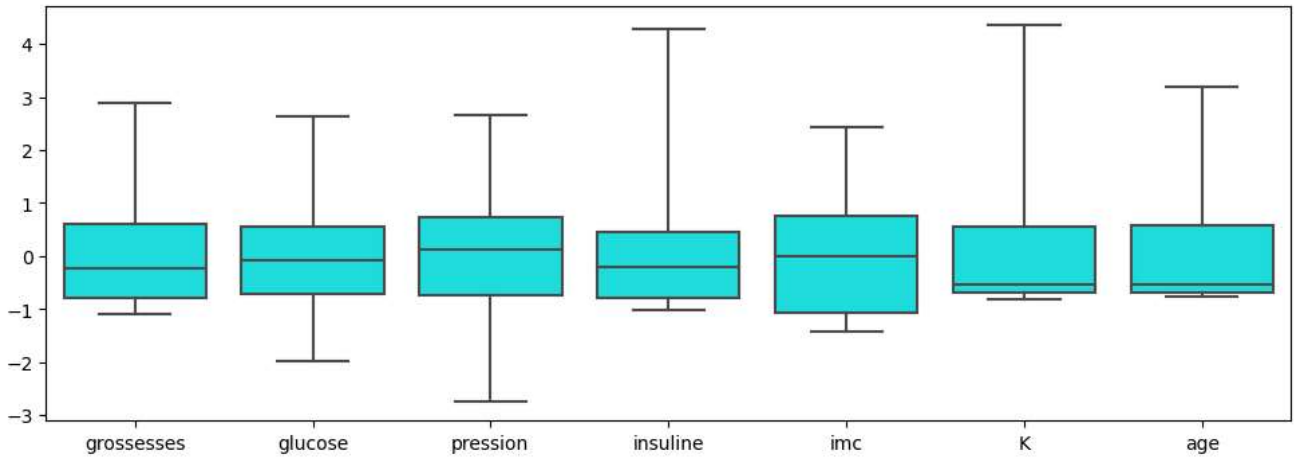
Out[25]:
```
grossesses    1.0
glucose       1.0
pression      1.0
insuline      1.0
imc           1.0
K             1.0
age           1.0
dtype: float64
```

In [26]:
```
Tn.describe()
```

Out[26]:

	grossesses	glucose	pression	insuline	imc	K	age
count	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02	4.000000e+02
mean	1.942890e-17	2.831069e-17	-1.917910e-16	-1.082467e-17	1.868072e-15	1.074141e-16	-3.824198e-17
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.083118e+00	-1.994040e+00	-2.749813e+00	-1.010984e+00	-1.414239e+00	-8.197870e-01	-7.753989e-01
25%	-7.995795e-01	-7.162700e-01	-7.400941e-01	-7.902642e-01	-1.067467e+00	-6.814188e-01	-6.961148e-01
50%	-2.325018e-01	-7.738495e-02	1.125138e-01	-2.028660e-01	-1.516604e-03	-5.376222e-01	-5.375465e-01
75%	6.181147e-01	5.615001e-01	7.215194e-01	4.414921e-01	7.523012e-01	5.636558e-01	5.922525e-01
max	2.886425e+00	2.637877e+00	2.670337e+00	4.286281e+00	2.444176e+00	4.354307e+00	3.188808e+00

In [27]:
```
pl. figure(figsize = (12,4))
boxplot(data =Tn[paramètres],
color='cyan',
whis = [0, 100])
pl.savefig("boxplotzoomN.png")
```



On remarque que maintenant que les valeurs sont normalisées, les dispertions sont comparables

In [28]:
```
Tn['M'] = T['M'] #Ajout de M à Tn
Tn.head()
```

Out[28]:

	grossesses	glucose	pression	insuline	imc	K	age	M
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579	1
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852	1
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115	1
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147	1
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399	1

On normalise également les paramètres des patients (avec la même moyenne et le même écart-type que ceux de la table présédente)

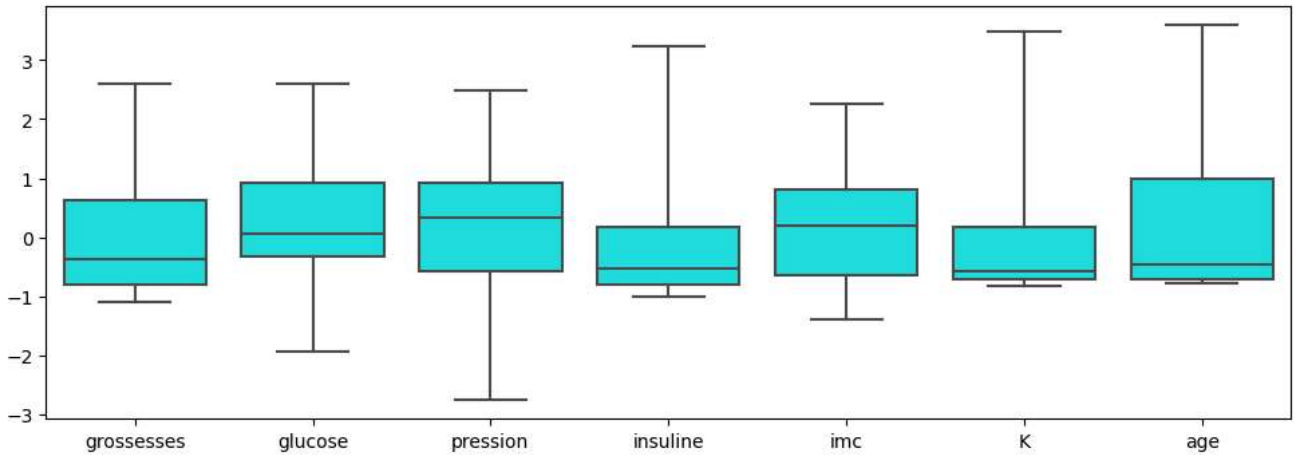
In [29]:
```
Pn=(P[paramètres]-Moy)/Sig
Pn.head()     #Affiche valeur normalisée de chaque paramètre
```

Out[29]:

	grossesses	glucose	pression	insuline	imc	K	age
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987

In [30]:
```
pl. figure(figsize = (12,4))
boxplot(data =Pn[paramètres],
color='cyan',
whis = [0, 100])
#Dispersion entre taux d'insuline et de glucose
```

Out[30]:
```
<AxesSubplot:>
```

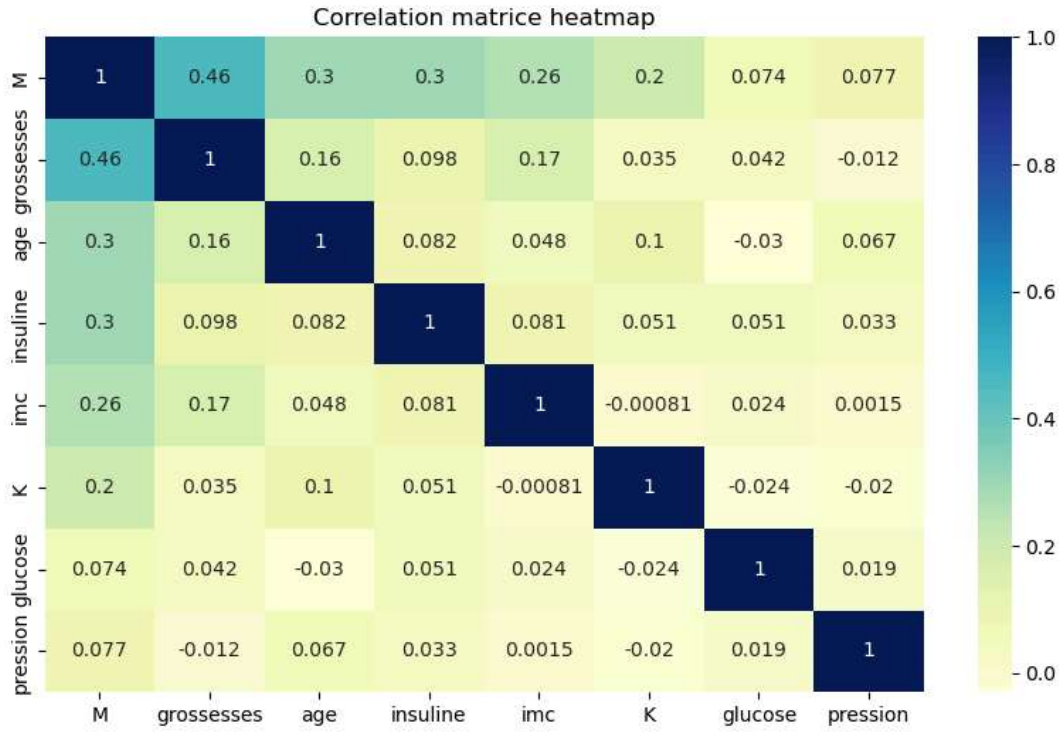


2.Estimation des paramètres qui permettent de distinguer les malades des non-malades

Nous pouvons notamment utiliser une matrice de corrélation pour voir quels paramètres sont les plus liées à la variable 'M'

In [31]:
```
corr = T.corr()
row_order = corr.abs().sum(axis=1).sort_values(ascending=False).index
col_order = corr.abs().sum(axis=0).sort_values(ascending=False).index

corr = corr.loc[row_order, col_order]
pl.figure(figsize=(10,6))
sns.heatmap(corr, cmap="YlGnBu", annot=True)
pl.title("Correlation matrice heatmap")
pl.show()
pl.savefig("matrice.png")
```

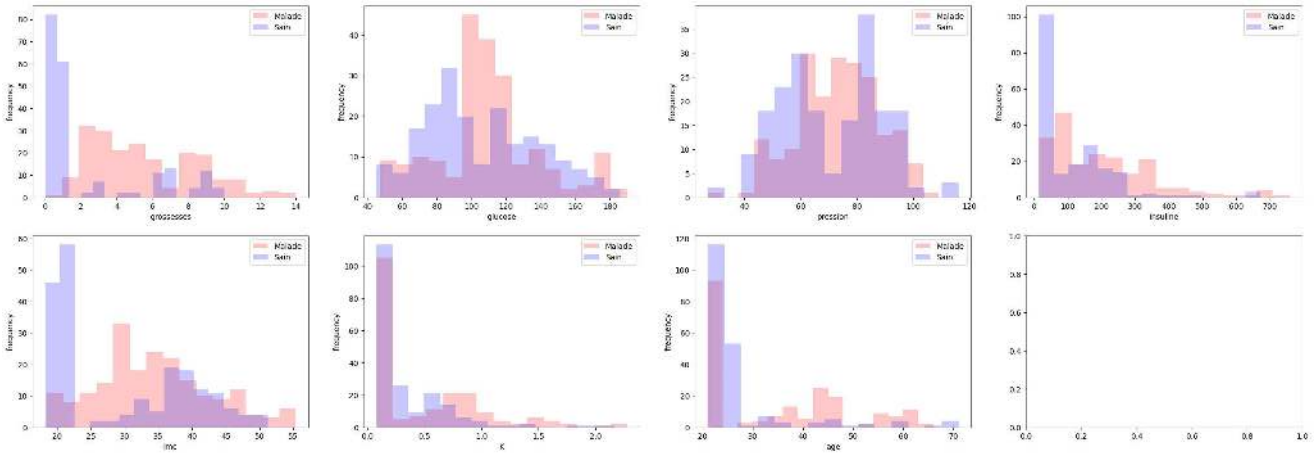



```
<Figure size 640x480 with 0 Axes>
```

Avec un facteur de corrélation de 0,46, le Grossesse a la corrélation la plus élevée avec le résultat, c'est-à-dire que la Grossesse est le paramètre le plus élevé pour prédire le diabète. De plus, l'insuline et l'âge peuvent également être considéré comme un facteur significatif pour prédire le diabète.

Visualisations 2d

In [32]:
```
fig, axs = pl.subplots(nrows=2,ncols=4,figsize=(30,10))
for c,ax in zip(paramètres,axs.flatten()):
    colonna = T.eval(c)
    #
    ax.hist(colonna[T.M==1],bins=15,label='Malade',color='r',alpha=0.2)
    ax.hist(colonna[T.M==0],bins=15,label='Sain',color='b',alpha=0.2)
    #
    # histplot(colonna[T.M==1],bins=15,label='Malade',color='r',alpha=0.2, ax=ax)
    # histplot(colonna[T.M==0],bins=15,label='Sain',color='b',alpha=0.2, ax=ax)
    #
    # distplot(colonna[T.M==1],bins=15,label='Malade',color='r', ax=ax)
    # distplot(colonna[T.M==0],bins=15,label='Sain',color='b', ax=ax)
    #
    ax.legend()
    ax.set_xlabel(c)
    ax.set_ylabel('frequency')
```



In [33]:
```
sns.pairplot(T, hue = 'M')   #Crée une matrice de diagramme de dispersion avec en bleu les malades et vert les non malades + les
```

Out[33]:
```
<seaborn.axisgrid.PairGrid at 0x204d5bd9a90>
```

Ce graphique associe les constructions de tracés à partir de deux figures de base, l'histogramme et le nuage de points. L'histogramme sur la diagonale nous permet de voir la distribution d'une seule variable tandis que les nuages de points sur les triangles supérieur et inférieur montrent la relation (ou l'absence de relation) entre deux variables.

Visualisation 3D

In [34]:
```
import plotly.express as px
fig=px.scatter_3d(
    data_frame=Tn,
    x='grossesses',
    y='age',
    z='insuline',
    opacity = 0.6,
    color =Tn['M'].astype(str),
    color_discrete_map = {'1' : 'red',
                          '0' : 'green'}

)

fig.update_layout(
width= 700,
legend={'x':0.9,
        'y':0.5,
        'title' : '<b> Malade ?</b>',
        'bordercolor':'gray', 'font': {'family': 'Verdana',
        'size': 14},
        'borderwidth' : 2,})
fig.show()
pl.savefig("diagrammedispersion.png")
```

```
<Figure size 640x480 with 0 Axes>
```

En testant plusieurs paramètres pour les valeurs x,y,z, nous pouvons remarquer que les valeurs les plus prépondérantes semblent toujours être : l'insuline,l'imc et les grossesses. Nous associons alors à la fonction "Par" ces 3 paramètres.

Choix des paramètres

In [35]:
```
Par = ['grossesses','insuline','age']
```

3. Prédictions

Machine learning : pour un patient p on doit prédire s'il est malade.

Le *machine learning* ou apprentissage automatique se base sur un principe simple: l'utilisation des données pour que la machine apprenne et construire des modèles qui pourrons être appliqués sur des nouvelles données.

Idéé: on prend un nouvel individu dans la table `Pn` et on observe ses 11 plus proches voisins dans la table `Tn`.
Si la majorité de ses voisins est malade on prédira qu'il est malade, dans le cas contraire qu'il est sain.

Prédictions du patient 0

On extrait les paramètres de l'individu 0 (de la population totale):

In [36]:
```
L1 = Tn[Par].loc[0]
L1
```

Out[36]:
```
grossesses   -0.516041
insuline     -0.911304
age           0.334579
Name: 0, dtype: float64
```

On extrait les paramètres de l'individu 0 (du patient):

In [37]:
```
L2 = Pn[Par].loc[0]
L2
```

Out[37]:
```
grossesses    1.186677
insuline      0.128374
age          -0.696987
Name: 0, dtype: float64
```

On calcule la distance entre les deux points

In [38]:
```
from math import dist
dist(L1,L2)
```

Out[38]: 2.2459526024552336

Prédiction pour le patient p et l'individu i

In [39]:
```
distance = lambda i,p : dist( Tn[Par].loc[i] , Pn[Par].loc[p] )
```

In [40]:
```
Tn['distance']= [distance (i, 0) for i in range (400) ]
Tn. head ()
```

Out[40]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579	1	2.245953
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852	1	3.009691
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115	1	1.811946
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147	1	1.277862
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399	1	1.740041

In [41]:
```
selected_columns = Tn.iloc[:, [7,8]]
selected_columns.head()
```

Out[41]:

	M	distance
0	1	2.245953
1	1	3.009691
2	1	1.811946
3	1	1.277862
4	1	1.740041

On choisit de prendre les 11 individus les plus proches:

In [42]:
```
Tn.sort_values('distance').iloc[:, [7,8]].head(11) # par rapport au patient 0, 7 personnes avec les valeurs les plus proches sont
```

Out[42]:

	M	distance
378	0	0.081397
59	1	0.181715
162	1	0.206647
181	1	0.284644
249	0	0.303461
204	0	0.314077
102	1	0.314368
340	0	0.314807
131	1	0.316036
58	1	0.327363
80	1	0.335362

In [43]:
```
Tn.sort_values('distance').head(11).M.sum()
```

Out[43]: 7

le patient p=0 a des paramètres similaires à 7 individus malades parmi les 11 les plus proches.

Comme 7>11/2, on prédit que ce patient est diabétique.

Fonction de prédiction

On associe à chaque patient p dans `Pn` les 11 individus `Tn` les plus proches. Si parmi ces 11 individus il y en a plus de 5 malades, on suppose que le patient p est malade.

In [44]:
```
def prediction(p):                                          #fonction de prediction
    Tn['d']= [distance (i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(11).M.sum()
    if nb_malades >= 6:
        return 1
```

```
    else:
        return 0
```

In [45]:
```
prediction(0)
```

Out[45]:
```
1
```

On ajoute dans la table `Pn` une colonne `T` qui indique la prédiction.

In [46]:
```
Pn['M'] = P['M']
Pn.head()
```

Out[46]:

	grossesses	glucose	pression	insuline	imc	K	age	M
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1

In [47]:
```
Pn['T'] = [prediction(p) for p in range(100)]
Pn
```

Out[47]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1
...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1

100 rows × 9 columns

Prédiction pour k = 7 et la distance euclidienne

Regardons maintenant en prenant les 7 plus proches voisins.

In [48]:
```
Tn.sort_values('distance').head(7) # par rapport au patient 0, 4 personnes avec les valeurs les plus proches sont malades
```

Out[48]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d
378	1.185192	1.072608	-0.740094	0.142453	-1.325189	0.182555	-0.616831	0	0.081397	0.650768
59	1.185192	-0.332939	0.356116	-0.035546	1.653417	0.263949	-0.775399	1	0.181715	0.604930
162	1.185192	-0.173218	-0.374691	-0.078266	1.582386	0.728298	-0.696115	1	0.206647	0.580383
181	1.185192	0.178169	-0.374691	0.413012	-1.318690	-0.668360	-0.696115	1	0.284644	0.822410
249	1.468731	1.519828	-0.800995	0.206533	-0.331803	0.484469	-0.616831	0	0.303461	0.932860
204	1.468731	-1.099601	-0.740094	0.242133	0.753170	-0.487350	-0.775399	0	0.314077	0.961137
102	0.901654	-0.205162	0.051613	0.021413	-0.029408	0.204744	-0.775399	1	0.314368	0.380379

In [49]:
```
Tn.sort_values('distance').head(7).M.sum()
```

Out[49]:
```
4
```

In [50]:
```
def prediction2(p):                                    #fonction de prediction
    Tn['d']= [distance (i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(7).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [51]:
```
prediction2(0)
```

Out[51]:
```
0
```

In [52]:
```
Pn['T2'] = [prediction2(p) for p in range(100)]
Pn
```

Out[52]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0
...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0

100 rows × 10 columns

Prédiction pour k = 11 et la distance Manhattan

In [53]:
```
from scipy.spatial.distance import cityblock
distance2 = lambda i,p : cityblock(Tn[Par].loc[i], Pn[Par].loc[p])
```

Une autre façon de calculer la distance manhattan est d'utiliser la fonction suivante:

In [54]:
```
manhattandistance= lambda i,p : sum(abs(Tn.loc[i, Par] - Pn.loc[p, Par]))
```

In [55]:
```
def prediction3(p):                                    #fonction de prediction
    Tn['d']= [distance2 (i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(11).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [56]:
```
prediction3(0)
```

Out[56]:
```
1
```

In [57]:
```
Pn['T3'] = [prediction3(p) for p in range(100)]
Pn
```

Out[57]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0	1

100 rows × 11 columns

In [58]:
```
Tn['distance2']= [manhattandistance (i, 0) for i in range (400)]
Tn. head ()
```

Out[58]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d	distance2
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579	1	2.245953	2.820247	3.773960
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852	1	3.009691	5.468371	4.673424
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115	1	1.811946	1.526497	2.323188
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147	1	1.277862	2.637214	1.842267
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399	1	1.740041	2.319485	2.501347

Prédiction pour k = 7 et la distance Manhattan

In [59]:
```
def prediction4(p):                                    #fonction de prediction
    Tn['d']= [distance2(i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(7).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [60]:
```
Pn['T4'] = [prediction4(p) for p in range(100)]
Pn
```

Out[60]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3	T4
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1	0
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1	0
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0	1	0

100 rows × 12 columns

Prédiction pour k = 11 et la distance Chebychev

In [61]:
```
Chebyshevdistance= lambda i,p : max((abs(Tn.loc[i, Par] - Pn.loc[p, Par])))
```

In [62]:
```
def prediction5(p):                                    #fonction de prediction
    Tn['d']= [Chebyshevdistance(i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(11).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [63]:
```
prediction5(0)
```

Out[63]:
```
1
```

In [64]:
```
Pn['T5'] = [prediction5(p) for p in range(100)]
Pn
```

Out[64]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3	T4	T5
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1	0	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1	0	1
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0	0	0	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0	0	0	1
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0	1	0	0

100 rows × 13 columns

4. Mesures de fiabilité des prédictions

Croisement des variables

On considère les événements suivants:

- M: la personne est malade,
- T: son test est positif.

On notera $\bar M$ et $\bar T$ respectivement les évenements la personne n'est pas malade et le test est négatif.

Pour mesurer la pertinence du classement, nous avons besoin des élements suivants régroupé dans la **Matrice de confusion**:

- vrai positif: MT (malade avec un test positif), i.e. $M=1$ et $T=1$
- vrai négatif: $\bar M \bar T$ (sain avec un test négatif) i.e. $M=0$ et $T=0$
- faux positif: $\bar M T$ (sain avec un test positif) i.e. $M=0$ et $T=1$
- faux négatif: $M \bar T$ (malade avec un test négatif) i.e. $M=1$ et $T=0$

Les prédictions pour les patients sont stockes dans `Pnorm` colonne `T` et il faut la comparer à la colonne `M`.

On construit la matrice de confusion en croisant les variables `M` et `T` :

Annexe

Cf. https://beranger.medium.com/ml-accuracy-pr%C3%A9cision-f1-score-courbe-roc-que-choisir-5d4940b854d7

Distance Euclidienne ($k=11$)

Croisement des variables pour méthode k=11 plus proches voisins / Euclidienne

In [65]:
```
#Prédiction variables M et T
E = pa.crosstab(Pn['M'], Pn['T'],margins=True)
E
```

Out[65]:

M	T 0	1	All
0	44	6	50
1	9	41	50
All	53	47	100

In [66]:
```
M_T = E.loc[1,1]        # tp
M_barT = E.loc[1,0]     # fn
barM_T = E.loc[0,1]     # fp
barM_barT = E.loc[0,0]  # tn
```

Métriques:

- Les métriques telles que l'accuracy, la précision, la sensibilité ou la spécificité sont souvent utilisés pour évaluer la précision des modèles de prédictions. Nous pouvons chercher à les calculer pour estimer la précision de notre modèle. Les formules des principaux métriques sont les suivantes :
- Accuracy: $\dfrac{MT+\bar M \bar T}{MT+\bar M \bar T + \bar M T + M \bar T}$
- Précision: $\dfrac{MT}{MT+\bar M T}$
- Sensibilité: $\dfrac{MT}{MT+M \bar T}$
- f1-score: $2\dfrac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$
- Spécificité: $\dfrac{\bar M\bar T}{\bar M\bar T+\bar M T}$

Les formules de la spécificité et de la sensibilité parlent d'elles-mêmes : les deux sont complémentaires. Regarder l'une sans l'autre n'apporte rien.

On calcule l'**accuracy** du test.
La précision permet de connaître la proportion de bonnes prédictions par rapport à toutes les prédictions.
L'opération est simplement : Nombre de bonnes prédictions / Nombre total de prédictions.

In [67]:
```
accuracy = (M_T + barM_barT) / ( M_T + barM_barT + barM_T + M_barT )
accuracy
```

Out[67]:
```
0.85
```

In [68]:
```
precision = M_T / ( M_T + barM_T )
precision
```

Out[68]:
```
0.8723404255319149
```

On calcule la **sensibilité** du test.
La sensibilité mesure la proportion des prédictions positives correctement identifiées (qui étaient réellement positives). Ce calcul permet d'estimer combien de vrais positifs nous avons réussi à déterminerr et ceux pas détecté. On souhaite avoir un Recall(Rappel) = 1.

In [69]:
```
sensibilité = M_T / ( M_T + M_barT)
sensibilité
```

Out[69]:
```
0.82
```

On calcule le **f1-score**.
Le F1-Score combine subtilement la précision et le rappel. Il est intéressant et plus intéressant que la précision car le nombre de vrais négatifs n'est pas pris en compte. Et dans les situations d'imbalanced class nous avons une majorité de vrais négatifs qui faussent complètement notre perception de la performance de l'algorithme. Un grand nombre de vrais négatifs laissera le F1-Score de marbre.

In [70]:
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[70]:
```
0.8453608247422681
```

On calcule la **spécificité** du test, c'est-à-dire la probabilité d'avoir un test négatif sachant qu'on n'est pas malade.
La spécificité mesure, à l'inverse de la sensibilité, la proportion d'éléments négatifs correctement identifiés.

In [71]:
```
specificité =  barM_barT / ( barM_barT + barM_T)
specificité
```

Out[71]:
```
0.88
```

Valeur prédictive du test pour une prévalence donnée

On suppose que la **prévalence du diabète** (i.e. la probabilité dêtre malade) $\mathbb{P}(M)$ dans la population est égale à 0.1: $$ \mathbb{P}(M)=0.1, \qquad \mathbb{P}(\bar M)=0.9 $$

On utilise la méthode de prédiction obtenue par machine learning décrite plus haut. Si une personne voit son test positif; quelle est la probabilité qu'elle soit malade? On appelle cette probabilité la **valeur prédictive du test**, i.e. la valeur de $\mathbb{P}_{T}(M)$.

D'après ce qui précède, il est convenu que la sensibilité est 0.82. On a alors $$ \mathbb{P}_{M}(T)=0.1\times0.82, \quad \mathbb{P}_{M}(\bar T)=0.1\times0.18, \quad \mathbb{P}_{\bar M}(T)=0.9\times0.12, \quad \mathbb{P}_{\bar M}(\bar T)=0.9\times0.88. $$

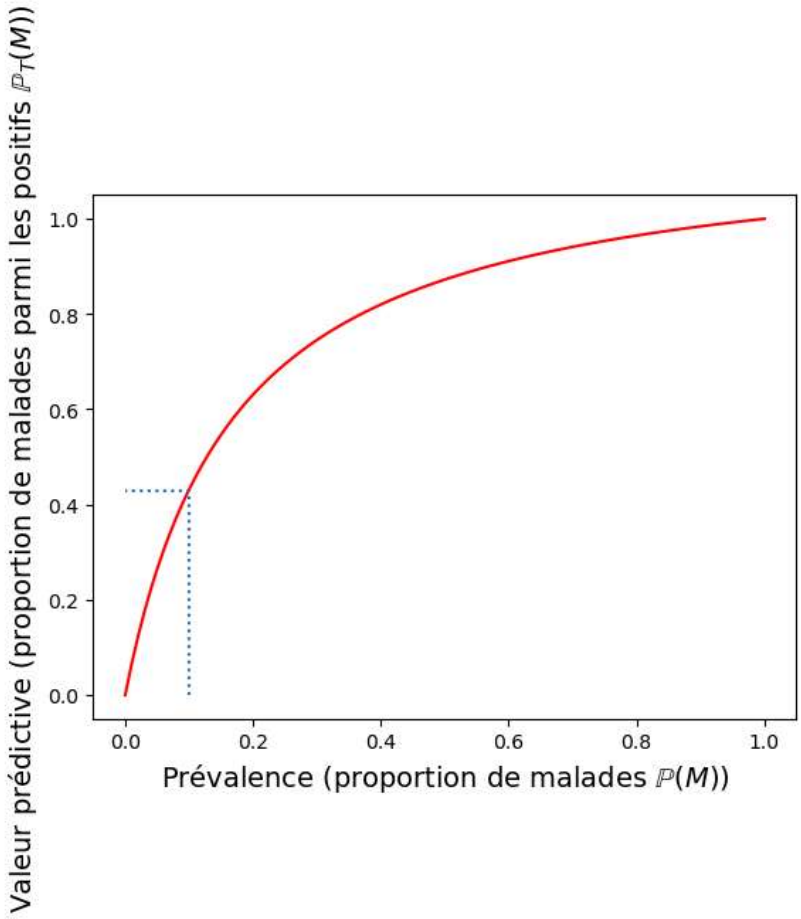
On a $\mathbb{P}(T)=\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)=0.1\times0.82+0.9\times0.12=0.19$.

La valeur prédictive du test est $\mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}(T)}=\frac{0.1\times0.82}{0.19}=0.43$.

Valeur prédictive en fonction de la prévalence

Quelle que soit la prévalence $p=\mathbb{P}(M)$ du diabète, la valeur prédictive du test est $$ \mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)}=\frac{p\times0.82}{p\times0.82+(1-p)\times0.12}=\frac{0.82p}{0.12+0.70p}. $$ On affiche le graphe de $p\mapsto \mathbb{P}_{T}(M)$ qui à la prévalence de la maladie fait correspondre la valeur prédictive du test.

In [72]:
```
PM = [k/100 for k in range(100+1)]
L = [0.82*p/(0.12+0.7*p) for p in PM]
pl.plot(PM,L,color='red')
pl.plot([0.1,0.1,0],[0,0.43,0.43],':')
pl.xlabel('Prévalence (proportion de malades $\mathbb{P}(M)$)',fontsize=14)
pl.ylabel('Valeur prédictive (proportion de malades parmi les positifs $\mathbb{P}_{T}(M)$)',fontsize=14);
pl.savefig("Courbe_de_prevalencedist.png")
```



Nous pouvons voir que lorque la prévalence est élevée la valeur prédictive est assez élevée. Cependant, on remaque que lorsque la prévalence est faible, la valeur prédictive est plutôt faible. Cela signifie que quand la proportion de malades est basse, la proportion de malades parmi les positifs est faible.

Lorsque la maladie est rare (c'est-à-dire lorsque la prévalence est faible), il serait souhaitable que sa valeur prédictive soit plus importante.

Courbe ROC : vrais positifs en fonction de faux positifs

La courbe ROC qui permet de mesurer la performance d'un classificateur. $\mathbb{P}_{T}(M) \mapsto \mathbb{P}_{T}(\bar M)$

Quelle que soit la prévalence $p=\mathbb{P}(M)$ du diabète, on a $$ \mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)}=\frac{p\times0.82}{p\times0.82+(1-p)\times0.12}=\frac{0.82p}{0.12+0.70p}, $$ $$ \mathbb{P}_{T}(\bar M)=\frac{\mathbb{P}(T\cap\bar M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{\bar M}(T)}{\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)}=\frac{(1-p)\times0.12}{p\times0.82+(1-p)\times0.12}=\frac{0.12(1-p)}{0.12+0.70p}. $$

Ses probabilités permettent d'obtenir la courbe ROC.

AUC

Nous pouvons alors calculer l'aire sous la courbe ROC (AUC ROC) qui permet d'évaluer la performance globale d'un modèle. L'AUC se situe entre 50% (pour un modèle non-informatif) et 100% (pour un modèle parfait).

In [73]:
```
# Aire = [ (L[0]+L[idx]+2*sum(L[1:idx]))*(PM[1]-PM[0])/2 for idx in range(len(L)) ]
# pl.plot(PM,Aire)
# pl.plot([PM[0],PM[-1]],[0.7,0.7])
# pl.grid();
AireTOT = (L[0]+L[-1]+2*sum(L[1:-1]))*(PM[1]-PM[0])/2
AireTOT
```

Out[73]:
```
0.7854415197276429
```

L'AUC ROC de notre modèle se situe bien au-dessus de celui d'un modèle non-informatif et en-dessous de celui d'un modèle parfait. Avec 78% d'AUC ROC, il s'agit d'un modèle assez performant.

Distance Chebychev ($k=11$)

In [74]:
```
E5= pa.crosstab(Pn['M'], Pn['T5'],margins=True)
E5
```

Out[74]:

M	T5 0	1	All
0	43	7	50
1	10	40	50
All	53	47	100

In [75]:
```
M5_T = E5.loc[1,1]        # tp
M5_barT = E5.loc[1,0]     # fn
barM5_T = E5.loc[0,1]     # fp
barM5_barT = E5.loc[0,0]  # tn
```

In [76]:
```
precision = M5_T / ( M5_T + barM5_T )
precision
```

Out[76]: 0.851063829787234

In [77]:
```
sensibilité = M5_T / ( M5_T + M5_barT)
sensibilité
```

Out[77]: 0.8

In [78]:
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[78]: 0.8247422680412372

In [79]:
```
specificité =  barM5_barT / ( barM5_barT + barM5_T)
specificité
```

Out[79]: 0.86

Distance Manhattan ($k=7$)

In [80]:
```
E4 = pa.crosstab(Pn['M'], Pn['T4'],margins=True)
E4
```

Out[80]:

M	T4 0	1	All
0	49	1	50
1	18	32	50
All	67	33	100

In [81]:
```
M4_T = E4.loc[1,1]        # tp
M4_barT = E4.loc[1,0]     # fn
barM4_T = E4.loc[0,1]     # fp
barM4_barT = E4.loc[0,0]  # tn
```

In [82]:
```
precision = M4_T / ( M4_T + barM4_T )
precision
```

Out[82]: 0.9696969696969697

In [83]:
```
sensibilité = M4_T / ( M4_T + M4_barT)
sensibilité
```

Out[83]: 0.64

In [84]:
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[84]: 0.7710843373493975

In [85]:
```
specificité =  barM4_barT / ( barM4_barT + barM4_T)
specificité
```

Out[85]: 0.98

Pour k=7 voisins la sensibilité pour la distance euclidienne et distance Manhattan est la même

Distance Euclidienne ($k=7$)

Méthode k=7 plus proches voisins / Distance Euclidienne

In [86]:
```
#Prédiction variables M et T
E2 = pa.crosstab(Pn['M'], Pn['T2'],margins=True)
E2
```

Out[86]:

T2	0	1	All
M			
0	48	2	50
1	18	32	50
All	66	34	100

In [87]:
```
M2_T = E2.loc[1,1]        # tp
M2_barT = E2.loc[1,0]     # fn
barM2_T = E2.loc[0,1]     # fp
barM2_barT = E2.loc[0,0]  # tn
```

In [88]:
```
(E2.loc[0,0]+ E2.loc[1,1]) / 100  #Spécificité du test
```

Out[88]:
```
0.8
```

In [89]:
```
precision = M2_T / ( M2_T + barM2_T )
precision
```

Out[89]:
```
0.9411764705882353
```

In [90]:
```
sensibilité = M2_T / ( M2_T + M2_barT)
sensibilité
```

Out[90]:
```
0.64
```

In [91]:
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[91]:
```
0.7619047619047621
```

In [92]:
```
specificité =  barM2_barT / ( barM2_barT + barM2_T)
specificité
```

Out[92]:
```
0.96
```

Valeur prédictive en fonction de la prévalence

Quelle que soit la prévalence $p=\mathbb{P}(M)$ du diabète, la valeur prédictive du test est $$ \mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)}=\frac{p\times0.64}{p\times0.64+(1-p)\times0.04}=\frac{0.64p}{0.04+0.6p}. $$ On affiche le graphe de $p\mapsto \mathbb{P}_{T}(M)$ qui à la prévalence de la maladie fait correspondre la valeur prédictive du test.

Distance Manhattan ($k=11$)

Croisement des variables pour méthode k=11 plus proches voisins / Manhattan

In [93]:
```
E1 = pa.crosstab(Pn['M'], Pn['T3'],margins=True)
E1
```

Out[93]:

T3	0	1	All
M			
0	42	8	50
1	9	41	50
All	51	49	100

In [94]:
```
M1_T = E1.loc[1,1]        # tp
M1_barT = E1.loc[1,0]     # fn
barM1_T = E1.loc[0,1]     # fp
barM1_barT = E1.loc[0,0]  # tn
```

In [95]:
```
precision = M1_T / ( M1_T + barM1_T )
precision
```

Out[95]:
```
0.8367346938775511
```

In [96]:
```
sensibilité = M1_T / ( M1_T + M1_barT)
sensibilité
```

Out[96]:
```
0.82
```

In [97]:
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[97]:
```
0.8282828282828283
```

In [98]:
```
specificité =  barM1_barT / ( barM1_barT + barM1_T)
specificité
```

Out[98]:
```
0.84
```

In [99]:
```
accuracy = (M1_T + barM1_barT) / ( M1_T + barM1_barT + barM1_T + M1_barT )
accuracy
```

Out[99]:
```
0.83
```

Valeur prédictive en fonction de la prévalence

Quelle que soit la prévalence $p=\mathbb{P}(M)$ du diabète, la valeur prédictive du test est $$ \mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)}=\frac{p\times0.82}{p\times0.82+(1-p)\times0.16}=\frac{0.82p}{0.16+0.66p}. $$ On affiche le graphe de $p\mapsto \mathbb{P}_{T}(M)$ qui à la prévalence de la maladie fait correspondre la valeur prédictive du test.

In [100…
```
donnees = {'spécifité': [0.84,0.88], 'sensibilité': [0.82,0.82], 'précision': [0.8367346938775511,0.8723404255319149], 'f1_score'

tableau = pa.DataFrame(donnees)
tableau.index = ['Manhattan', 'Euclidienne']
print(tableau)
```

```
             spécifité  sensibilité  précision  f1_score
Manhattan         0.84         0.82   0.836735  0.828283
Euclidienne       0.88         0.82   0.872340  0.845361
```

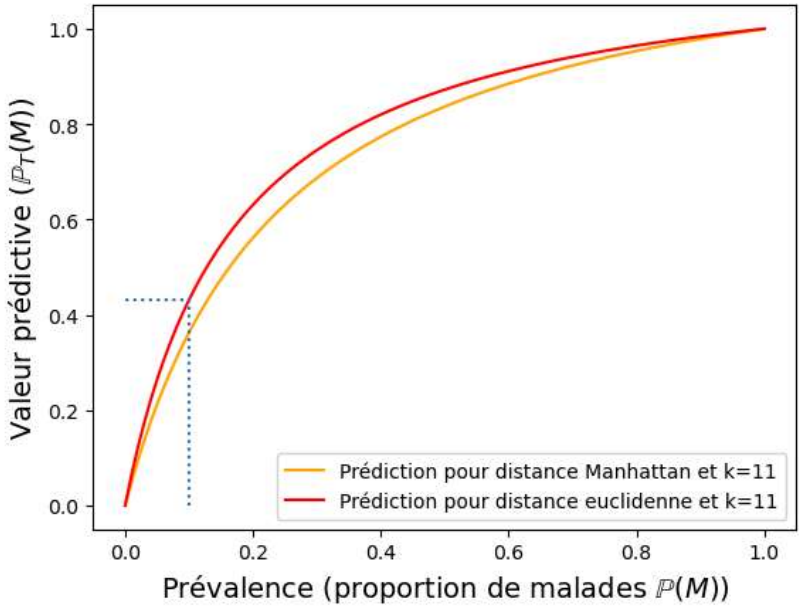
In [101…
```
donnees = {'spécifité': [0.88,0.96], 'sensibilité': [0.82,0.64], 'précision': [0.8723404255319149,0.94], 'f1_score': [0.845360824

tableau = pa.DataFrame(donnees)
tableau.index = ['k = 11', 'k = 7']
print(tableau)
pl.savefig("Tableau.png")
```

```
        spécifité  sensibilité  précision  f1_score
k = 11       0.88         0.82    0.87234  0.845361
k = 7        0.96         0.64    0.94000  0.760000
<Figure size 640x480 with 0 Axes>
```

On affiche la valeur prédictive en fonction de la prévalence avec $k=11$ pour la distance Manhattan et Euclidienne

In [102…
```
PM = [k/100 for k in range(100+1)]
L1 = [0.82*p/(0.12+0.7*p) for p in PM]
L2= [0.82*p/(0.16+0.66*p) for p in PM]
pl.plot(PM,L2,color='orange',label='Prédiction pour distance Manhattan et k=11')
pl.plot(PM,L1,color='red',label='Prédiction pour distance euclidenne et k=11')
pl.plot([0.1,0.1,0],[0,0.43,0.43],':')
pl.xlabel('Prévalence (proportion de malades $\mathbb{P}(M)$)',fontsize=14)
pl.ylabel('Valeur prédictive ($\mathbb{P}_{T}(M)$)',fontsize=14);
pl.legend()
pl.savefig("Courbe_de_prevalencek=11.png")
```

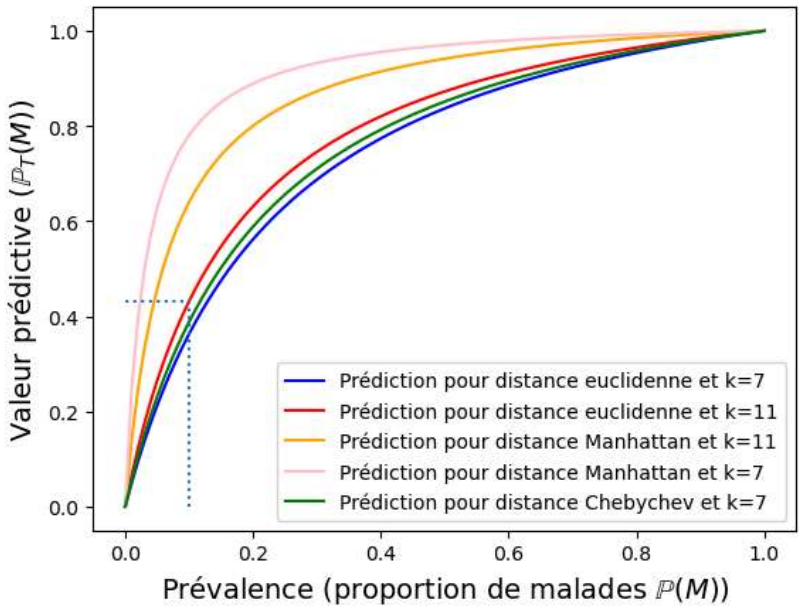


Nous pouvons voir que pour une prévalence faible, la valeur de prédiction est la même pour les deux distances. Pour une valeur plus élevée, la valeur de la prédiction pour la distance Manhattan est plus faible que celle pour la distance euclidienne. Utiliser la distance euclidienne semble être le plus adéquat pour notre modèle si on choisit $k = 11$.

In [103…

```
PM = [k/100 for k in range(100+1)]
L1 = [0.82*p/(0.12+0.7*p) for p in PM]
L2= [0.82*p/(0.16+0.66*p) for p in PM]
L3= [0.64*p/(0.04+0.6*p) for p in PM]
L4=[0.64*p/(0.02+0.62*p)for p in PM]
L5=[0.8*p/(0.14+0.66*p)for p in PM]
pl.plot(PM,L2,color='blue',label='Prédiction pour distance euclidenne et k=7')
pl.plot(PM,L1,color='red',label='Prédiction pour distance euclidenne et k=11')
pl.plot(PM,L3,color='orange',label='Prédiction pour distance Manhattan et k=11')
pl.plot(PM,L4,color='pink',label='Prédiction pour distance Manhattan et k=7')
```

```
pl.plot(PM,L5,color='green',label='Prédiction pour distance Chebychev et k=7')
pl.plot([0.1,0.1,0],[0,0.43,0.43],':')
pl.xlabel('Prévalence (proportion de malades $\mathbb{P}(M)$)',fontsize=14,)
pl.ylabel('Valeur prédictive ($\mathbb{P}_{T}(M)$)',fontsize=14);
pl.legend()
pl.savefig("Courbefinal_de_prevalence.png")
```



On constate que la distance de Manhattan avec k = 7 offre la meilleure performance prédictive.

Choix de paramètres différents

On étudie les correlations pour mieux choisir les paramètres significatifs.

Nous pouvons nous intéresser individuellement à chaque paramètre.

Intéressons nous aux grossesses.

Grossesses

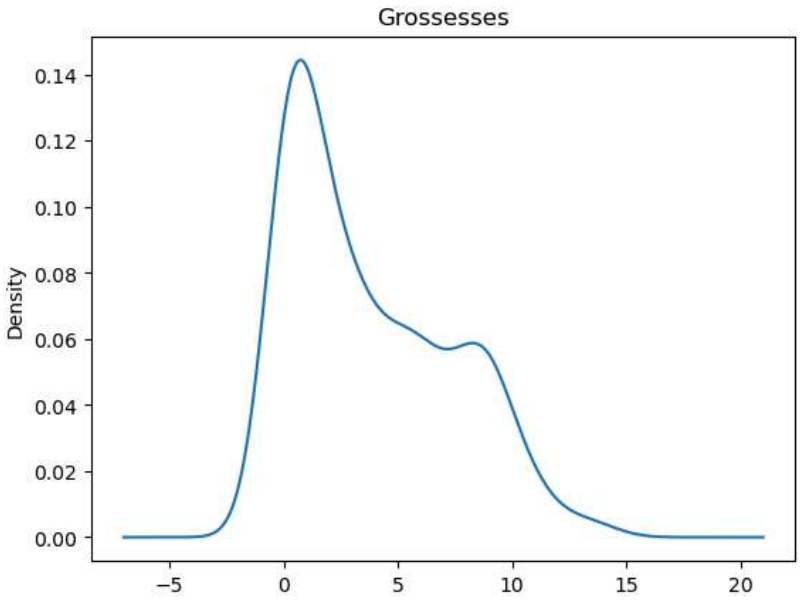
On peut chercher à voir qu'elle est le plus grand et le plus petit nombre de grossesses

In [104…
```
T.grossesses.agg(['max','min', 'mean', 'std'])
```

Out[104]:
```
max     14.000000
min      0.000000
mean     3.820000
std      3.526854
Name: grossesses, dtype: float64
```

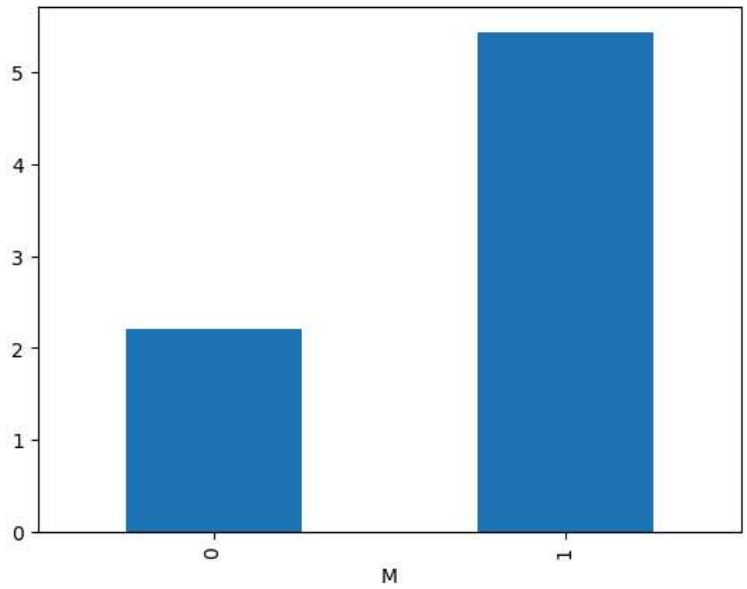
On peut regarder également la répartition des grossesses.

In [105…
```
T.grossesses.plot.kde( legend=False, title='Grossesses');
```



On peut égalemnt créer un graphique à barres pour visualiser le nombre de grossesses en fonction du diabète:

In [106…
```
T.groupby('M').grossesses.mean().plot(kind='bar');
```



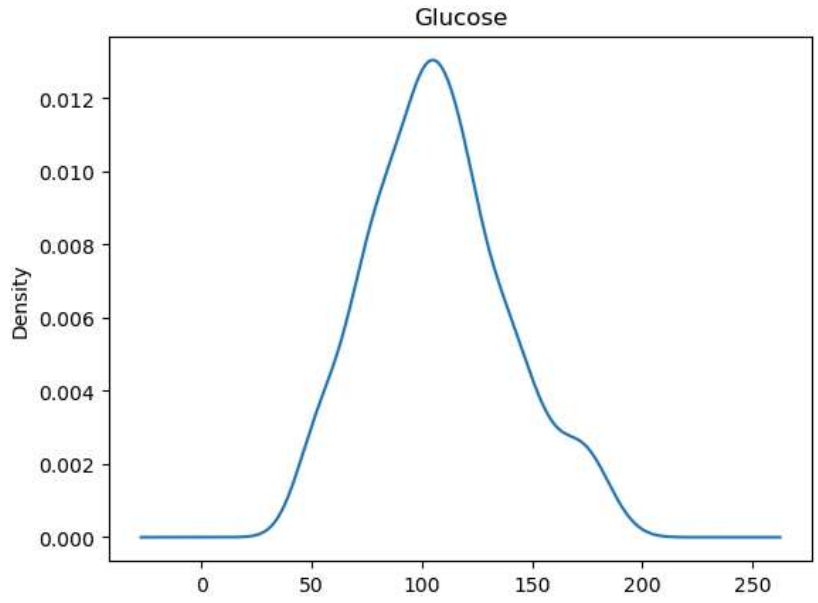
On peut faire pareil avec la variable "Glucose"

Glucose

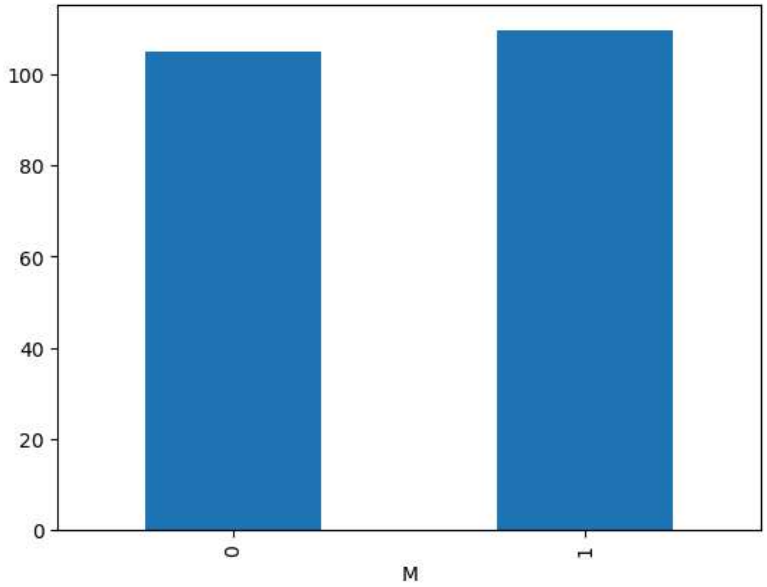
In [107…
```
T.glucose.agg(['max','min', 'mean', 'std'])
```

Out[107]:
```
max     190.000000
min      45.000000
mean    107.422500
std      31.304535
Name: glucose, dtype: float64
```

In [108…
```
T.glucose.plot.kde( legend=False, title='Glucose');
```



In [109…
```
T.groupby('M').glucose.mean().plot(kind='bar');
```



Choix d'un seul paramètre : les grossesses

In [110…
```
Par2 = ['grossesses']
```

On extrait les paramètres de l'individu 0 (de la population totale):

In [111…
```
L1 = Tn[Par2].loc[0]
L1
```

Out[111]:
```
grossesses   -0.516041
Name: 0, dtype: float64
```

On extrait les paramètres de l'individu 0 (du patient):

In [112…
```
L2 = Pn[Par2].loc[0]
L2
```

Out[112]:
```
grossesses    1.186677
Name: 0, dtype: float64
```

In [113…
```
distance2 = lambda i,p : dist( Tn[Par2].loc[i] , Pn[Par2].loc[p] )
```

In [114…
```
Tn['distance2']= [distance2 (i, 0) for i in range (400) ]
Tn. head ()
```

Out[114]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d	distance2
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579	1	2.245953	1.134929	1.702717
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852	1	3.009691	2.748535	0.565593
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115	1	1.811946	1.134929	1.702717
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147	1	1.277862	1.110750	0.282055
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399	1	1.740041	1.310299	1.419178

In [115…
```
Tn.sort_values('distance2').head(11)
```

Out[115]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d	distance2
117	1.185192	-0.684326	1.635028	1.424049	-0.201843	0.681394	1.048136	1	2.173530	1.665739	0.001484
162	1.185192	-0.173218	-0.374691	-0.078266	1.582386	0.728298	-0.696115	1	0.206647	0.566304	0.001484
378	1.185192	1.072608	-0.740094	0.142453	-1.325189	0.182555	-0.616831	0	0.081397	0.566304	0.001484
156	1.185192	-0.109329	0.599718	2.128927	0.897002	-0.655265	-0.775399	1	2.002090	2.307096	0.001484
181	1.185192	0.178169	-0.374691	0.413012	-1.318690	-0.668360	-0.696115	1	0.284644	0.591181	0.001484
94	1.185192	-1.291267	0.599718	-0.590905	0.337352	0.899225	2.078830	1	2.867494	2.696433	0.001484
24	1.185192	2.126769	0.599718	1.210450	-1.084534	-0.534648	-0.141126	1	1.216500	1.388619	0.001484
188	1.185192	0.274002	0.356116	-0.626505	1.948693	-0.607003	-0.696115	1	0.754880	0.566304	0.001484
147	1.185192	0.912887	-0.618293	0.982610	-0.359527	-0.638287	-0.696115	1	0.854239	1.160779	0.001484
127	1.185192	-1.674598	0.417017	-0.505465	-0.465336	-0.690816	-0.696115	1	0.633841	0.566304	0.001484
88	1.185192	-0.013496	-1.592702	0.334692	0.163701	-0.572345	0.968852	1	1.678567	1.586455	0.001484

On choisit de prendre les 11 individus les plus proches:

In [116…
```
Tn.sort_values('distance2').head(11).M.sum()
```

Out[116]:
```
10
```

le patient p=0 a des paramètres similaires à 10 individus malades parmi les 11 les plus proches.

Comme 10>11/2, on prédit que ce patient est diabétique.

In [117…
```
def prediction2(p):                                    #fonction de prediction
    Tn['d']= [distance2 (i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(11).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [118…
```
prediction2(0)
```

Out[118]:
```
1
```

In [119…
```
Pn['M'] = P['M']
Pn['G'] = [prediction2(p) for p in range(100)]
Pn
```

Out[119]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3	T4	T5	G
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1	0	1	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1	0	1	0
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0	0	0	0	1
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0	0	0	1	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0	0	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0	0	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0	1	0	0	1

100 rows × 14 columns

In [120…
```
#prédiction variables M et T
E2 = pa.crosstab(Pn['M'], Pn['G'],margins=True)
E2
```

Out[120]:

G	0	1	All
M			
0	42	8	50
1	7	43	50
All	49	51	100

In [121…
```
(E2.loc[0,0]+ E2.loc[1,1]) / 100  #Spécificité du test
```

Out[121]:
```
0.85
```

In [122…
```
E2.loc[0,0] / 50
```

Out[122]:
```
0.84
```

In [123…
```
M_T = E2.loc[1,1]        # tp
M_barT = E2.loc[1,0]     # fn
barM_T = E2.loc[0,1]     # fp
barM_barT = E2.loc[0,0]  # tn
```

In [124…
```
precision = M_T / ( M_T + barM_T )
precision
```

Out[124]:
```
0.8431372549019608
```

In [125…
```
sensibilité = M_T / ( M_T + M_barT)
sensibilité
```

Out[125]:
```
0.86
```

In [126…
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[126]:
```
0.8514851485148515
```

In [127…
```
specificité =  barM_barT / ( barM_barT + barM_T)
specificité
```

Out[127]:
```
0.84
```

Valeur prédictive du test pour une prévalence donnée

On suppose que la **prévalence du diabète** (i.e. la probabilité dêtre malade) $\mathbb{P}(M)$ dans la population est égale à 0.1: $$ \mathbb{P}(M)=0.1, \qquad \mathbb{P}(\bar M)=0.9 $$

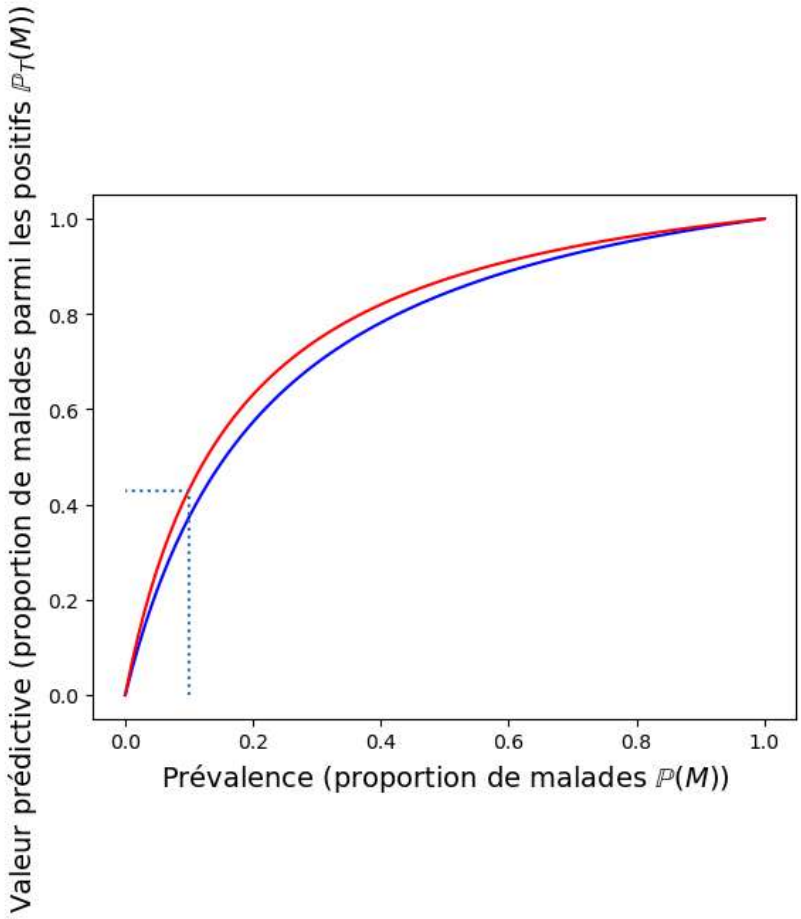
On utilise la méthode de prédiction obtenue par machine learning décrite plus haut. Si une personne voit son test positif; quelle est la probabilité qu'elle soit malade? On appelle cette probabilité la **valeur prédictive du test**, i.e. la valeur de $\mathbb{P}_{T}(M)$.

D'après ce qui précède, il est convenu que la sensibilité est 0.86. On a alors $$ \mathbb{P}_{M}(T)=0.1\times0.86, \quad \mathbb{P}_{M}(\bar T)=0.1\times0.14, \quad \mathbb{P}_{\bar M}(T)=0.9\times0.16, \quad \mathbb{P}_{\bar M}(\bar T)=0.9\times0.84. $$

On a $\mathbb{P}(T)=\mathbb{P}_{M}(T)+\mathbb{P}_{\bar M}(T)=0.1\times0.86+0.9\times0.16=0.226$.

La valeur prédictive du test est $\mathbb{P}_{T}(M)=\frac{\mathbb{P}(T\cap M)}{\mathbb{P}(T)}=\frac{\mathbb{P}_{M}(T)}{\mathbb{P}(T)}=\frac{0.1\times0.86}{0.19}=0.45$.

In [128…
```
PM = [k/100 for k in range(100+1)]
LG = [0.86*p/(0.16+0.7*p) for p in PM]
L = [0.82*p/(0.12+0.7*p) for p in PM]
pl.plot(PM,LG,color='blue')
pl.plot(PM,L,color='red')
pl.plot([0.1,0.1,0],[0,0.43,0.43],':')
pl.xlabel('Prévalence (proportion de malades $\mathbb{P}(M)$)',fontsize=14)
pl.ylabel('Valeur prédictive (proportion de malades parmi les positifs $\mathbb{P}_{T}(M)$)',fontsize=14);
```



La Valeur prédictive en fonction de la prévalence semble plus élevée pour les 3 paramètres choisis précédemment que pour le seul paramètre 'grossesses'.

Choix d'autres paramètres : les grossesses, l'insuline et l'IMC

In [129…
```
Par3 = ['grossesses','insuline','imc']
```

In [130…
```
L1 = Tn[Par3].loc[0]
L1
```

Out[130]:
```
grossesses   -0.516041
insuline     -0.911304
imc           0.743188
Name: 0, dtype: float64
```

In [131…
```
L2 = Pn[Par3].loc[0]
L2
```

Out[131]:
```
grossesses    1.186677
insuline      0.128374
imc          -0.706641
Name: 0, dtype: float64
```

In [132…
```
distance3 = lambda i,p : dist( Tn[Par3].loc[i] , Pn[Par3].loc[p] )
```

In [133…
```
Tn['distance3']= [distance3 (i, 0) for i in range (400) ]
Tn. head ()
```

Out[133]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d	distance2	distance3
0	-0.516041	-0.109329	0.173414	-0.911304	0.743188	-0.663069	0.334579	1	2.245953	1.134929	1.702717	2.466208
1	1.752270	0.433723	-0.496492	2.570366	-0.122520	-0.642042	0.968852	1	3.009691	1.133381	0.565593	2.573795
2	-0.516041	1.455939	-0.800995	-0.491225	-0.576447	1.416869	-0.696115	1	1.811946	1.134929	1.702717	1.816618
3	1.468731	1.200385	-1.531801	0.498452	1.475133	-0.800832	0.493147	1	1.277862	0.849842	0.282055	2.230841
4	-0.232502	-0.173218	0.234315	1.132130	-0.364915	-0.746634	-0.775399	1	1.740041	0.851391	1.419178	1.771545

In [134…
```
Tn.sort_values('distance3').head(11)
```

Out[134]:

	grossesses	glucose	pression	insuline	imc	K	age	M	distance	d	distance2	distance3
122	0.901654	0.465667	1.391426	-0.042666	-0.524592	-0.533592	1.048136	1	1.776498	0.282765	0.285023	0.378992
80	1.468731	1.998992	0.173414	0.291973	-0.485054	-0.532658	-0.775399	1	0.335362	0.849842	0.282055	0.394234
51	1.468731	-0.141273	-1.349100	0.469972	-0.703703	-0.604655	-0.616831	1	0.450188	0.849842	0.282055	0.443004
168	1.468731	0.210113	-0.557392	0.149573	-0.352730	-0.819787	0.413863	1	1.146295	0.849842	0.282055	0.453053
294	0.901654	0.401779	-1.714503	0.092613	-1.070478	-0.013104	-0.378978	0	0.428540	0.282765	0.285023	0.463567
340	0.901654	1.647605	0.538818	0.021413	-1.069393	-0.595446	-0.616831	0	0.314807	0.282765	0.285023	0.473569
249	1.468731	1.519828	-0.800995	0.206533	-0.331803	0.484469	-0.616831	0	0.303461	0.849842	0.282055	0.475571
361	0.901654	0.912887	0.721519	0.249253	-1.142783	-0.783014	-0.378978	0	0.443823	0.282765	0.285023	0.534855
378	1.185192	1.072608	-0.740094	0.142453	-1.325189	0.182555	-0.616831	0	0.081397	0.566304	0.001484	0.618709
180	0.618115	-0.300995	-0.131088	0.149573	-0.454408	1.061130	-0.775399	1	0.574335	0.000774	0.568562	0.622362
177	1.468731	-1.930152	-0.435591	0.519812	-0.285073	-0.642275	-0.061842	1	0.797614	0.849842	0.282055	0.640701

In [135…
```
Tn.sort_values('distance3').head(11).M.sum()
```

Out[135]: 6

le patient p=0 a des paramètres similaires à 10 individus malades parmi les 11 les plus proches.

Comme 6<11/2, on prédit que ce patient est diabétique.?

In [136…
```
def prediction1(p):                                    #fonction de prediction
    Tn['d']= [distance3 (i , p) for i in range (400)]
    nb_malades = Tn.sort_values('d').head(11).M.sum()
    if nb_malades >= 6:
        return 1
    else:
        return 0
```

In [137…
```
prediction1(0)
```

Out[137]: 1

In [138…
```
Pn['M'] = P['M']
Pn.head()
```

Out[138]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3	T4	T5	G
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1	0	1	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1	0	1	0
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1	0	0	1

In [139…
```
Pn['M'] = P['M']
Pn['A'] = [prediction1(p) for p in range(100)]
Pn
```

Out[139]:

	grossesses	glucose	pression	insuline	imc	K	age	M	T	T2	T3	T4	T5	G	A
0	1.186677	-0.013513	-0.253206	0.128374	-0.706641	-0.711474	-0.696987	1	1	0	1	0	1	1	1
1	-1.084475	-1.132962	0.722423	0.691556	-0.231660	0.965497	1.049449	1	1	0	1	0	1	0	1
2	-0.516687	2.097448	0.051678	3.243701	-0.076218	1.584572	2.319584	1	1	1	1	1	1	1	1
3	-0.232793	-0.045498	0.112655	0.342240	-0.500231	-0.768260	-0.776370	1	1	1	1	1	1	1	1
4	1.470571	0.914030	-0.619067	-0.007075	0.652177	0.172542	-0.696987	1	1	0	1	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.618889	-0.557246	0.905353	-0.791254	-1.068121	-0.750247	-0.617603	0	0	0	0	0	0	1	0
96	-0.800581	2.161416	0.783400	-0.919574	-1.379340	0.172170	0.334998	0	0	0	0	0	1	0	0
97	-0.800581	-1.005025	0.478516	0.206791	1.427533	-0.391351	-0.776370	0	0	0	0	0	0	0	0
98	-1.084475	1.969511	0.051678	0.591752	-1.084774	-0.606006	-0.379453	0	0	0	0	0	0	0	0
99	0.618889	-0.877088	-0.862975	-0.178169	0.624063	1.284638	-0.617603	0	1	0	1	0	0	1	1

100 rows × 15 columns

In [140…
```
#prédiction variables M et T
E3 = pa.crosstab(Pn['M'], Pn['A'],margins=True)
E3
```

Out[140]:

A	0	1	All
M			
0	43	7	50
1	7	43	50
All	50	50	100

In [141…
```
(E3.loc[0,0]+ E3.loc[1,1]) / 100  #Spécificité du test
```

Out[141]:
```
0.86
```

In [142…
```
E3.loc[0,0] / 50
```

Out[142]:
```
0.86
```

In [143…
```
M3_T = E3.loc[1,1]        # tp
M3_barT = E3.loc[1,0]     # fn
barM3_T = E3.loc[0,1]     # fp
barM3_barT = E3.loc[0,0]  # tn
```

In [144…
```
precision = M3_T / ( M3_T + barM3_T )
precision
```

Out[144]:
```
0.86
```

In [145…
```
sensibilité = M3_T / ( M3_T + M3_barT)
sensibilité
```

Out[145]:
```
0.86
```

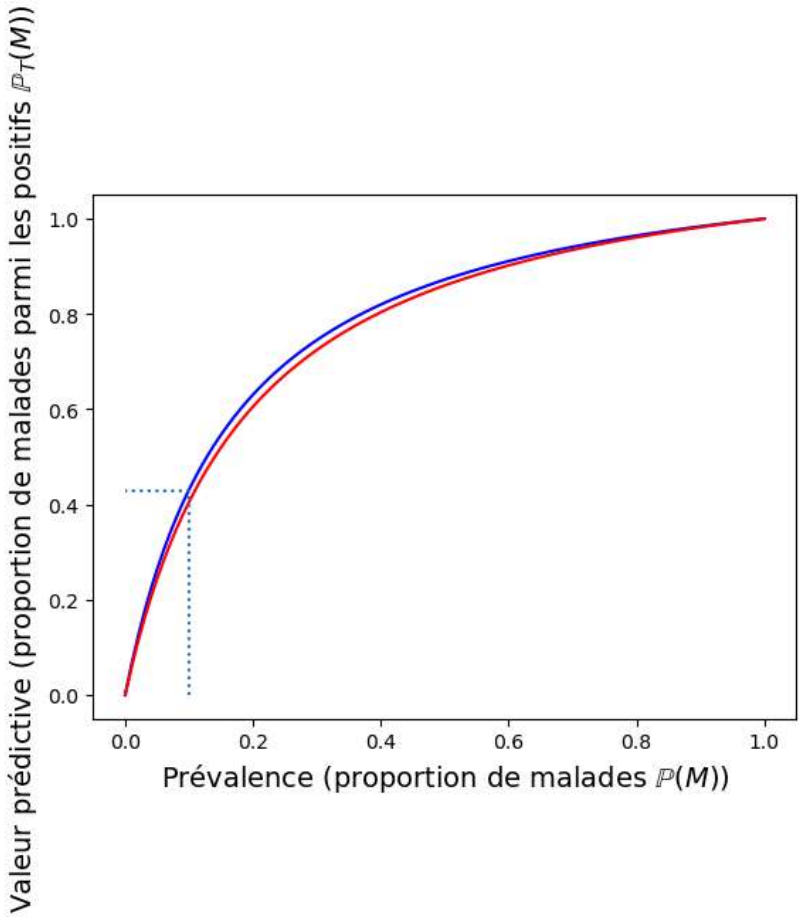
In [146…
```
f1_score = 2*(precision*sensibilité)/(precision+sensibilité)
f1_score
```

Out[146]:
```
0.8599999999999999
```

In [147…
```
specificité =  barM3_barT / ( barM3_barT + barM3_T)
specificité
```

Out[147]:
```
0.86
```

In [148…
```
PM = [k/100 for k in range(100+1)]
LA = [0.86*p/(0.14+0.72*p) for p in PM]
L = [0.82*p/(0.12+0.7*p) for p in PM]
pl.plot(PM,L,color='blue')
pl.plot(PM,LA,color='red')
pl.plot([0.1,0.1,0],[0,0.43,0.43],':')
pl.xlabel('Prévalence (proportion de malades $\mathbb{P}(M)$)',fontsize=14)
pl.ylabel('Valeur prédictive (proportion de malades parmi les positifs $\mathbb{P}_{T}(M)$)',fontsize=14);
```



La Valeur prédictive semble moins bonne avec les 3 nouveaux paramètres considérés. Le modèle paraît plus performant pour les varibles grossesses, insuline et âge.

Conclusion

Notre analyse a montré que la méthode des k plus proches voisins est particulièrement efficace lorsque la prévalence de la maladie est élevée, mais présente des limites lorsque la proportion de malades est faible. En ajustant le nombre de voisins k, nous avons pu améliorer la précision de nos prédictions. Par exemple, nous avons constaté que la distance de Manhattan avec k = 7 offrait une meilleure performance prédictive comparée à k = 11 pour cette même distance. Toutefois, la distance euclidienne s'est avérée plus efficace pour k = 11.

Pour aller plus loin, il serait intéressant d'améliorer notre modèle en testant d'autres valeurs de k, en explorant différentes mesures de distance, ou en considérant davantage de paramètres.