Mouhammed Niah

Etudiant: IMN

---

1) Exploration de nos données

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [2]:
```
df=pd.read_csv("C:\\Users\\HP\\Desktop\\Données\\diabetes.csv")
```

In [3]:
```
df.head()
```

Out[3]:

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

==========

Definition des variables

Outcome : qui veut dire resultat (resultat du diabête)\ BloodPressure: Tension artérielle\ Glucose: Glucose\ Insulin: Insuline\ Age: Age\ SkinThickness: Epaisseur du peau\ DiabetesPedigreeFunction: Pregnancies: Grossesse\ BMI:

==========

Information sur le dataset

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```
df.dtypes
```

Out[5]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

==========

Les statisstiques du dataset

In [6]:
```
df.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

==========

Le target(la variable à predire)

c'est la variable Outcome qui veut dire Resultats

==========

Division notre dataset en deux classe:

la classe des personnes atteient du diabete etla casse des personnes non attaint

In [7]:
```
df_yes=df[df['Outcome']==1]
df_non=df[df['Outcome']==0]
```

In [8]:
```
df_yes.head()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
2	8	183	64	0	0	23.3	0.672	
4	0	137	40	35	168	43.1	2.288	
6	3	78	50	32	88	31.0	0.248	
8	2	197	70	45	543	30.5	0.158	

In [9]:
```
df_non.head()
```

Out[9]:

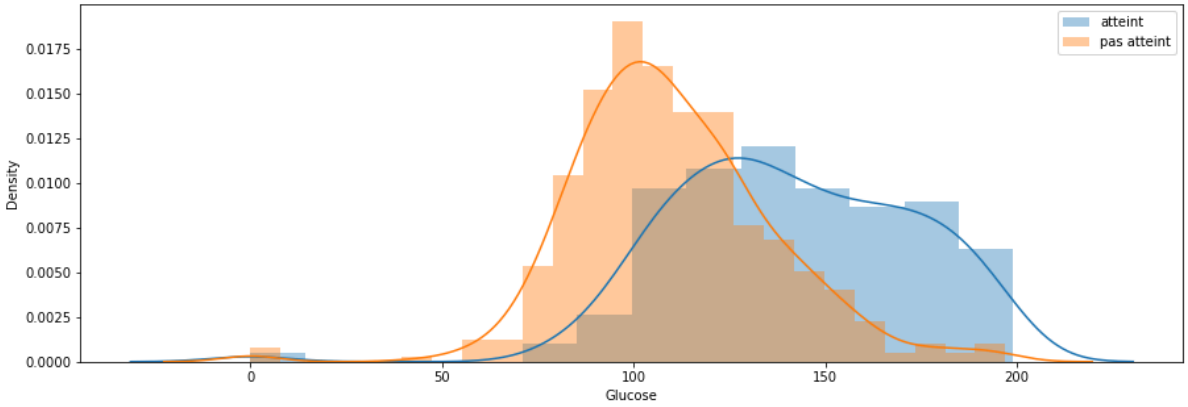
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
1	1	85	66	29	0	26.6	0.351	
3	1	89	66	23	94	28.1	0.167	
5	5	116	74	0	0	25.6	0.201	
7	10	115	0	0	0	35.3	0.134	
10	4	110	92	0	0	37.6	0.191	

==========

-Relation entre Target/Glucose

In [10]:
```
plt.figure(figsize=(15,5))
sns.distplot(df_yes['Glucose'],label='atteint')
sns.distplot(df_non['Glucose'],label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



commentaire

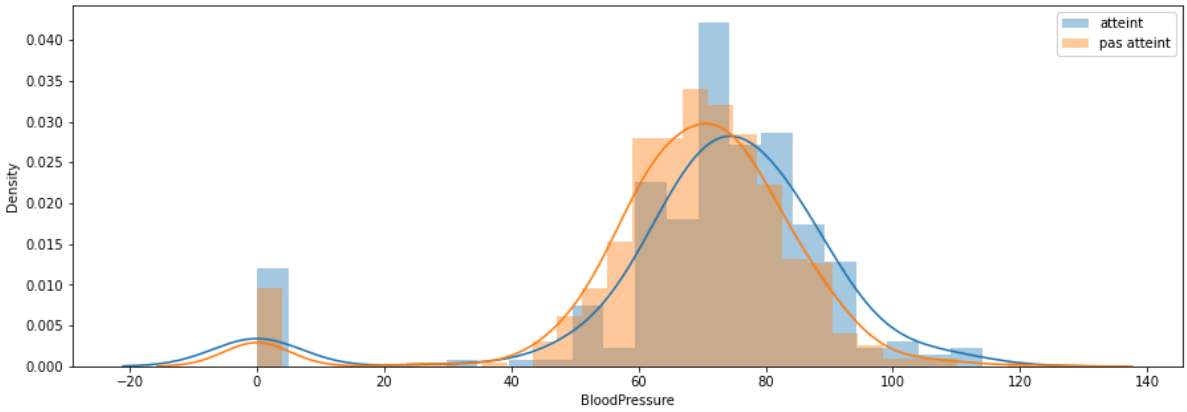
on constate une grande difference du taux de glucose entre les gens atteint et les gens non atteint du diabete, donc on peut poser une hypothése selon la quelle : Le taux de Glucose est un facteur determinant pour savoir si une personne est atteint ou non de diabete reste à verifier lors du traitement de nos données\
Ainsi nous allons consider par hypthése que la variable Glucose a un grand impact sur d

==========

-Relation entre Target/BloodPressure

In [11]:
```
plt.figure(figsize=(15,5))
sns.distplot(df_yes['BloodPressure'],label='atteint')
sns.distplot(df_non['BloodPressure'],label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



commentaire

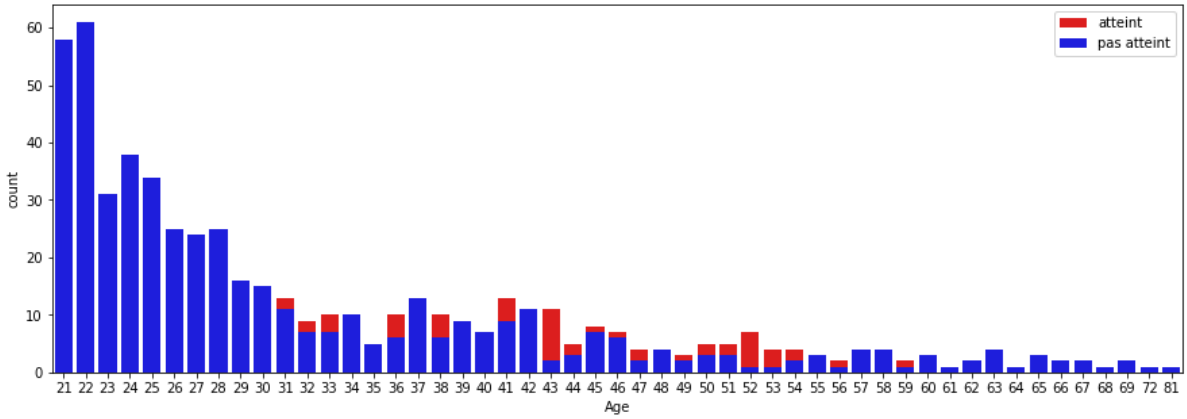
La visualisation des deux distributions pour cette variable nous permet de voir qu'il n y pas de difference entre la tension artérielle des gens atteints du diabête et ceux qui le sont pas.

==========

-Relation entre Target/ Age

In [12]:
```
plt.figure(figsize=(15,5))
sns.countplot(df_yes['Age'],color='r',label='atteint')
sns.countplot(df_non['Age'],color='b',label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```



commentaire

Avec la visualisation de l'âge des personnes on constate même si c'est pas evident que les gens qui ont moins de 31 ans atteints rarement le diabête, on note un balotage entre les gens positifs et negatives du diabete pour une age compris entre 31 ans et 40 ans,mais c'est

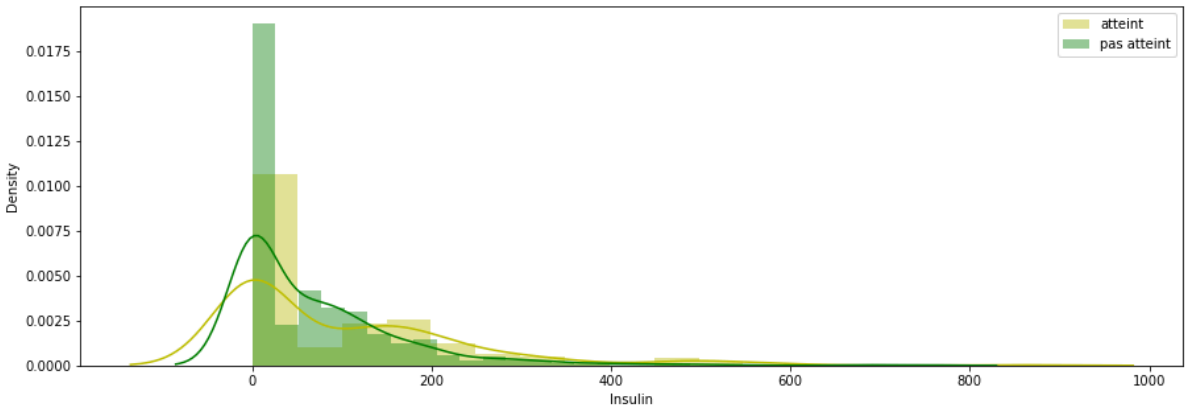
a partir des 41 ans que les gens sont caement touché par le dibête.\
Ainsi nous allons consider par hypthése que la variable Age a un impact sur diabête

==========

-Relation Target/Insulin

In [13]:
```
plt.figure(figsize=(15,5))
sns.distplot(df_yes['Insulin'],color='y',label='atteint')
sns.distplot(df_non['Insulin'],color='g',label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



commentaire

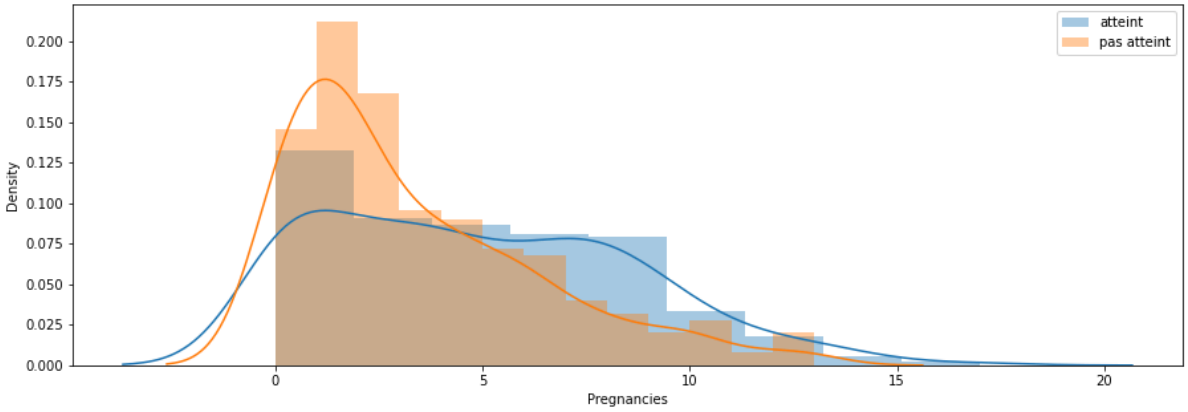
On constate que les des distribution sont quasi-identique d'aprés leurs visualisation donc on peut poser l'hypothese selon la quelle la variable Insulin n'a pas un grand impact sur la variable à predire

==========

Relation Target/Pregnancies

In [14]:
```
plt.figure(figsize=(15,5))
sns.distplot(df_yes['Pregnancies'],label='atteint')
sns.distplot(df_non['Pregnancies'],label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



commentaire

On constate que les des distribution sont quasi-identique d'aprés leurs visualisation donc on peut poser l'hypothese selon la quelle la variable Pregnancies n'a pas un grand impact sur la variable à predire

==========

- Relation entre Target/DiabetesPedigreeFunction

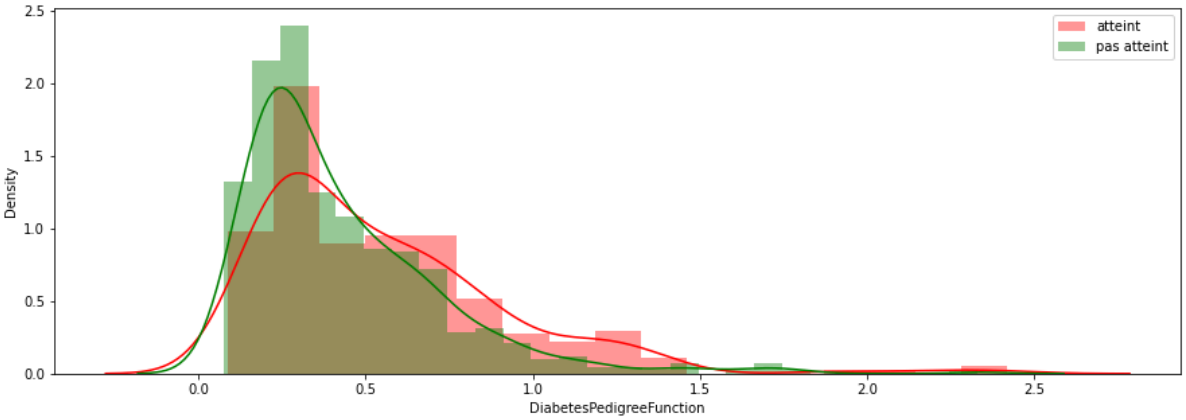
In [15]:
```
df.head(1)
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	

In [16]:
```
plt.figure(figsize=(15,5))
sns.distplot(df_yes['DiabetesPedigreeFunction'],color='r',label='atteint')
sns.distplot(df_non['DiabetesPedigreeFunction'],color='g',label='pas atteint')
plt.legend()
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. P
lease adapt your code to use either `displot` (a figure-level function with simila
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



commentaire

On constate que les des distribution sont quasi-identique d'aprés leurs visualisation donc on peut poser l'hypothese selon la quelle la variable DiabetesPedigreeFunction n'a pas un grand impact sur la variable à predire

2) Modelisation

==========

A) L'algorithme : KNeighborsClassifier (Plus proche voisin)

La classification basée sur les voisins est un type d'apprentissage basé sur les instances ou apprentissage non généralisateur : elle ne tente pas de construire un modèle interne général, mais stocke simplement les instances des données d'apprentissage. La classification est calculée à partir d'un simple vote majoritaire des voisins les plus proches de chaque point : un point d'interrogation se voit attribuer la classe de données qui a le plus de représentants parmi les voisins les plus proches du point.

A) a) Creons notre premier model de la maniere la plus basique....

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

In [18]:
```
x=df.drop('Outcome',axis=1)
y=df['Outcome']
```

In [19]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [20]:
```
model=KNeighborsClassifier()
```

In [21]:
```
model.fit(x_train,y_train)
c=model.score(x_test,y_test)
print('le modele donne \n ')
print('une efficacité de ',c*100,'% \n')
print('une erreur de',100-c*100,'%')
```

```
le modele donne

une efficacité de  75.32467532467533 %

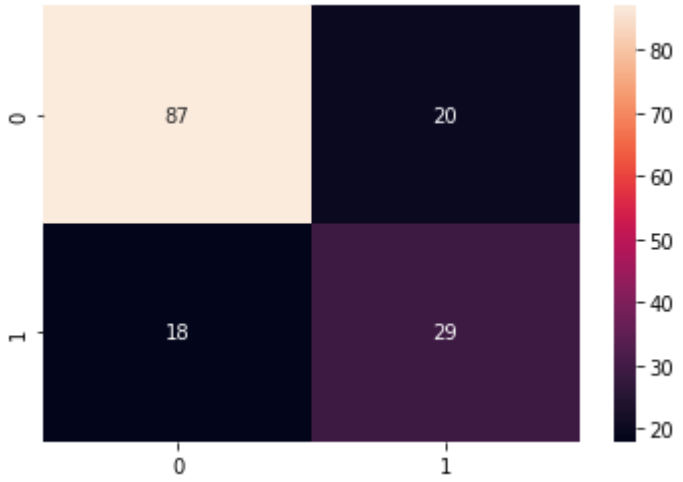
une erreur de 24.675324675324674 %
```

In [22]:
```
from sklearn.metrics import confusion_matrix
```

In [23]:
```
y_pred=model.predict(x_test)
```

In [24]:
```
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
```

Out[24]:
```
<AxesSubplot:>
```



Commentaire:

On obtient une score de 75% c'est-à-dire à 75% du temps le modele donne de bonne reponse,\ mais avec la vusialisation des resultats du modele suivant les classes on contate que le modele n'est pas tres efficace quant\ a la classement suivant la classe 1 car sur les 41 possibles le modele en a fait 18 de mauvaise classement.\ Pour ameliorer notre modele nous allons d'abord chercher le nombre de voisin le plus pertinant a fin d'avoir le meilleur\ modele possible

In [25]:
```
for k in range(3,21):
    modele=KNeighborsClassifier(k)
    modele.fit(x_train,y_train)
    score=modele.score(x_test,y_test)
    print('pour k =',k,'le modele du KNeighborsClassifier donne une score de',score
```

```
pour k = 3 le modele du KNeighborsClassifier donne une score de 72.07792207792207
%
pour k = 4 le modele du KNeighborsClassifier donne une score de 77.27272727272727
%
pour k = 5 le modele du KNeighborsClassifier donne une score de 75.32467532467533
%
pour k = 6 le modele du KNeighborsClassifier donne une score de 77.92207792207793
%
pour k = 7 le modele du KNeighborsClassifier donne une score de 75.97402597402598
%
pour k = 8 le modele du KNeighborsClassifier donne une score de 77.92207792207793
%
pour k = 9 le modele du KNeighborsClassifier donne une score de 77.27272727272727
%
pour k = 10 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
pour k = 11 le modele du KNeighborsClassifier donne une score de 76.62337662337663
%
pour k = 12 le modele du KNeighborsClassifier donne une score de 78.57142857142857
%
pour k = 13 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
pour k = 14 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
pour k = 15 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
pour k = 16 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
pour k = 17 le modele du KNeighborsClassifier donne une score de 78.57142857142857
%
pour k = 18 le modele du KNeighborsClassifier donne une score de 78.57142857142857
%
pour k = 19 le modele du KNeighborsClassifier donne une score de 79.87012987012987
%
pour k = 20 le modele du KNeighborsClassifier donne une score de 79.22077922077922
%
```

Nous allons continuer par la suite avec k=19 car le modele obtient une meilleure score(7

==========

A) b) Normalisation des données

In [26]:
```
from sklearn.preprocessing import StandardScaler
```

In [27]:
```
X=StandardScaler().fit_transform(x)
Y=y
```

In [28]:
```
X
```

Out[28]:
```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

In [29]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

In [30]:
```
model=KNeighborsClassifier(19)
```

In [31]:
```
model.fit(X_train,Y_train)
b=model.score(X_test,Y_test)
print('le modele donne \n ')
print('une efficacité de ',b*100,'% \n')
print('une erreur de',100-b*100,'%')
```

```
le modele donne

une efficacité de  80.51948051948052 %

une erreur de 19.480519480519476 %

==========================================================================
```

In [32]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

In [33]:
```
scor=[]
for k in range(1,11):
    for i in range(1,10):
        mode=make_pipeline(StandardScaler(),PolynomialFeatures(i),KNeighborsClassif
        mode.fit(x_train,y_train)
        score=mode.score(x_test,y_test)
        scor.append(mode.score(x_test,y_test))

        print('Pour k =',k,'et i=',i,'on a une score de',score*100,'%')
max(scor)
```

```
Pour k = 1 et i= 1 on a une score de 73.37662337662337 %
Pour k = 1 et i= 2 on a une score de 71.42857142857143 %
Pour k = 1 et i= 3 on a une score de 69.48051948051948 %
Pour k = 1 et i= 4 on a une score de 68.83116883116884 %
Pour k = 1 et i= 5 on a une score de 69.48051948051948 %
Pour k = 1 et i= 6 on a une score de 69.48051948051948 %
Pour k = 1 et i= 7 on a une score de 68.83116883116884 %
Pour k = 1 et i= 8 on a une score de 68.83116883116884 %
Pour k = 1 et i= 9 on a une score de 70.12987012987013 %
Pour k = 2 et i= 1 on a une score de 76.62337662337663 %
Pour k = 2 et i= 2 on a une score de 77.27272727272727 %
Pour k = 2 et i= 3 on a une score de 77.27272727272727 %
Pour k = 2 et i= 4 on a une score de 74.67532467532467 %
Pour k = 2 et i= 5 on a une score de 73.37662337662337 %
Pour k = 2 et i= 6 on a une score de 74.67532467532467 %
Pour k = 2 et i= 7 on a une score de 74.67532467532467 %
Pour k = 2 et i= 8 on a une score de 73.37662337662337 %
Pour k = 2 et i= 9 on a une score de 76.62337662337663 %
Pour k = 3 et i= 1 on a une score de 74.02597402597402 %
Pour k = 3 et i= 2 on a une score de 74.67532467532467 %
Pour k = 3 et i= 3 on a une score de 74.02597402597402 %
Pour k = 3 et i= 4 on a une score de 74.02597402597402 %
Pour k = 3 et i= 5 on a une score de 74.67532467532467 %
Pour k = 3 et i= 6 on a une score de 73.37662337662337 %
Pour k = 3 et i= 7 on a une score de 74.02597402597402 %
Pour k = 3 et i= 8 on a une score de 74.67532467532467 %
Pour k = 3 et i= 9 on a une score de 75.97402597402598 %
Pour k = 4 et i= 1 on a une score de 79.22077922077922 %
Pour k = 4 et i= 2 on a une score de 73.37662337662337 %
Pour k = 4 et i= 3 on a une score de 76.62337662337663 %
Pour k = 4 et i= 4 on a une score de 78.57142857142857 %
Pour k = 4 et i= 5 on a une score de 79.87012987012987 %
Pour k = 4 et i= 6 on a une score de 78.57142857142857 %
Pour k = 4 et i= 7 on a une score de 77.92207792207793 %
Pour k = 4 et i= 8 on a une score de 74.67532467532467 %
Pour k = 4 et i= 9 on a une score de 77.27272727272727 %
Pour k = 5 et i= 1 on a une score de 79.87012987012987 %
Pour k = 5 et i= 2 on a une score de 77.92207792207793 %
Pour k = 5 et i= 3 on a une score de 77.92207792207793 %
Pour k = 5 et i= 4 on a une score de 76.62337662337663 %
Pour k = 5 et i= 5 on a une score de 79.22077922077922 %
Pour k = 5 et i= 6 on a une score de 77.27272727272727 %
Pour k = 5 et i= 7 on a une score de 77.27272727272727 %
Pour k = 5 et i= 8 on a une score de 75.97402597402598 %
Pour k = 5 et i= 9 on a une score de 79.22077922077922 %
Pour k = 6 et i= 1 on a une score de 80.51948051948052 %
Pour k = 6 et i= 2 on a une score de 80.51948051948052 %
Pour k = 6 et i= 3 on a une score de 78.57142857142857 %
Pour k = 6 et i= 4 on a une score de 78.57142857142857 %
Pour k = 6 et i= 5 on a une score de 79.22077922077922 %
Pour k = 6 et i= 6 on a une score de 79.87012987012987 %
Pour k = 6 et i= 7 on a une score de 79.87012987012987 %
Pour k = 6 et i= 8 on a une score de 77.92207792207793 %
Pour k = 6 et i= 9 on a une score de 78.57142857142857 %
Pour k = 7 et i= 1 on a une score de 77.27272727272727 %
Pour k = 7 et i= 2 on a une score de 77.27272727272727 %
Pour k = 7 et i= 3 on a une score de 78.57142857142857 %
Pour k = 7 et i= 4 on a une score de 77.92207792207793 %
Pour k = 7 et i= 5 on a une score de 80.51948051948052 %
Pour k = 7 et i= 6 on a une score de 78.57142857142857 %
Pour k = 7 et i= 7 on a une score de 79.87012987012987 %
Pour k = 7 et i= 8 on a une score de 78.57142857142857 %
Pour k = 7 et i= 9 on a une score de 79.87012987012987 %
Pour k = 8 et i= 1 on a une score de 77.92207792207793 %
```

```
Pour k = 8 et i= 2 on a une score de 78.57142857142857 %
Pour k = 8 et i= 3 on a une score de 77.27272727272727 %
Pour k = 8 et i= 4 on a une score de 79.22077922077922 %
Pour k = 8 et i= 5 on a une score de 80.51948051948052 %
Pour k = 8 et i= 6 on a une score de 78.57142857142857 %
Pour k = 8 et i= 7 on a une score de 81.16883116883116 %
Pour k = 8 et i= 8 on a une score de 77.27272727272727 %
Pour k = 8 et i= 9 on a une score de 79.87012987012987 %
Pour k = 9 et i= 1 on a une score de 77.27272727272727 %
Pour k = 9 et i= 2 on a une score de 75.97402597402598 %
Pour k = 9 et i= 3 on a une score de 76.62337662337663 %
Pour k = 9 et i= 4 on a une score de 77.27272727272727 %
Pour k = 9 et i= 5 on a une score de 81.16883116883116 %
Pour k = 9 et i= 6 on a une score de 81.81818181818183 %
Pour k = 9 et i= 7 on a une score de 81.81818181818183 %
Pour k = 9 et i= 8 on a une score de 79.87012987012987 %
Pour k = 9 et i= 9 on a une score de 80.51948051948052 %
Pour k = 10 et i= 1 on a une score de 77.92207792207793 %
Pour k = 10 et i= 2 on a une score de 78.57142857142857 %
Pour k = 10 et i= 3 on a une score de 77.27272727272727 %
Pour k = 10 et i= 4 on a une score de 78.57142857142857 %
Pour k = 10 et i= 5 on a une score de 81.16883116883116 %
Pour k = 10 et i= 6 on a une score de 79.87012987012987 %
Pour k = 10 et i= 7 on a une score de 82.46753246753246 %
Pour k = 10 et i= 8 on a une score de 77.92207792207793 %
Pour k = 10 et i= 9 on a une score de 81.16883116883116 %
```

Out[33]:
```
0.8246753246753247
```

on obtient une meilleure score (82,5%) pour K=10,et i=7.
par la suite nous allons continuer avec k=10 (nombre de voisin) et i=7(degre du polync

In [34]:
```
modele= mode=make_pipeline(StandardScaler(),PolynomialFeatures(7),KNeighborsClassif
modele.fit(x_train,y_train)
d=score=modele.score(x_test,y_test)
print('le modele donne \n ')
print('une efficacité de ',d*100,'% \n')
print('une erreur de',100-d*100,'%')
```

```
le modele donne

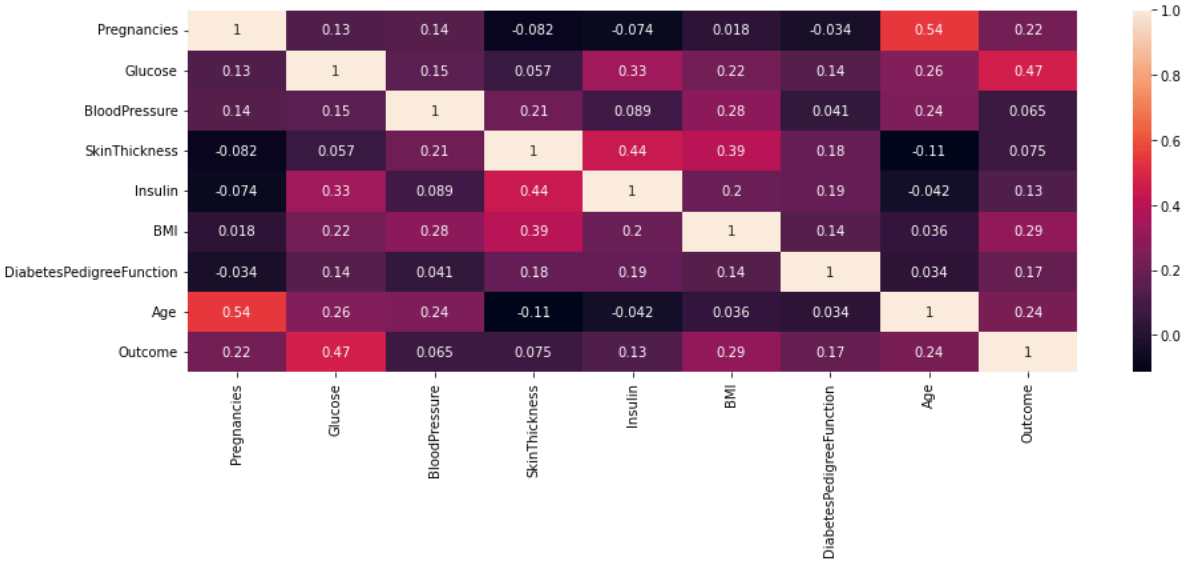
une efficacité de  82.46753246753246 %

une erreur de 17.532467532467535 %

==========================================================================
```

Corrélation

In [35]:
```
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



==========

Selection de meilleure variable

In [36]:
```
df.head(2)
```

Out[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
1	1	85	66	29	0	26.6	0.351	

In [37]:
```
from sklearn.feature_selection import SelectKBest,chi2
```

In [38]:
```
chi2(x,y)
```

Out[38]:
```
(array([ 111.51969064, 1411.88704064,   17.60537322,   53.10803984,
        2175.56527292,  127.66934333,    5.39268155,  181.30368904]),
 array([4.55261043e-026, 5.48728628e-309, 2.71819252e-005, 3.15697650e-013,
        0.00000000e+000, 1.32590849e-029, 2.02213728e-002, 2.51638830e-041]))
```

In [39]:
```
select=SelectKBest(chi2,k=4)
select.fit_transform(x,y)
select.get_support()
```

Out[39]:
```
array([False,  True, False, False,  True,  True, False,  True])
```

les 4 variables selectionnées sont: Glucose, Insulin, BMI et L

==========

In [40]:
```
df.drop(['BloodPressure','SkinThickness','DiabetesPedigreeFunction'],axis=1,inplace
```

In [41]:
```
x=df.drop('Outcome',axis=1)
```

```
y=df['Outcome']
```

In [42]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [43]:
```
scor=[]
for k in range(1,7):
    for i in range(1,10):
        mode=make_pipeline(StandardScaler(),PolynomialFeatures(i),KNeighborsClassif
        mode.fit(x_train,y_train)
        score=mode.score(x_test,y_test)
        scor.append(mode.score(x_test,y_test))

        print('Pour k =',k,'et i=',i,'on a une score de',score*100,'%')
max(scor)
```

```
Pour k = 1 et i= 1 on a une score de 73.37662337662337 %
Pour k = 1 et i= 2 on a une score de 74.02597402597402 %
Pour k = 1 et i= 3 on a une score de 75.97402597402598 %
Pour k = 1 et i= 4 on a une score de 76.62337662337663 %
Pour k = 1 et i= 5 on a une score de 75.97402597402598 %
Pour k = 1 et i= 6 on a une score de 75.32467532467533 %
Pour k = 1 et i= 7 on a une score de 74.02597402597402 %
Pour k = 1 et i= 8 on a une score de 75.32467532467533 %
Pour k = 1 et i= 9 on a une score de 74.02597402597402 %
Pour k = 2 et i= 1 on a une score de 79.22077922077922 %
Pour k = 2 et i= 2 on a une score de 77.92207792207793 %
Pour k = 2 et i= 3 on a une score de 76.62337662337663 %
Pour k = 2 et i= 4 on a une score de 74.67532467532467 %
Pour k = 2 et i= 5 on a une score de 73.37662337662337 %
Pour k = 2 et i= 6 on a une score de 74.67532467532467 %
Pour k = 2 et i= 7 on a une score de 74.67532467532467 %
Pour k = 2 et i= 8 on a une score de 74.02597402597402 %
Pour k = 2 et i= 9 on a une score de 72.07792207792207 %
Pour k = 3 et i= 1 on a une score de 77.92207792207793 %
Pour k = 3 et i= 2 on a une score de 79.22077922077922 %
Pour k = 3 et i= 3 on a une score de 77.27272727272727 %
Pour k = 3 et i= 4 on a une score de 76.62337662337663 %
Pour k = 3 et i= 5 on a une score de 77.27272727272727 %
Pour k = 3 et i= 6 on a une score de 79.22077922077922 %
Pour k = 3 et i= 7 on a une score de 78.57142857142857 %
Pour k = 3 et i= 8 on a une score de 79.22077922077922 %
Pour k = 3 et i= 9 on a une score de 79.87012987012987 %
Pour k = 4 et i= 1 on a une score de 79.22077922077922 %
Pour k = 4 et i= 2 on a une score de 75.97402597402598 %
Pour k = 4 et i= 3 on a une score de 75.32467532467533 %
Pour k = 4 et i= 4 on a une score de 75.32467532467533 %
Pour k = 4 et i= 5 on a une score de 76.62337662337663 %
Pour k = 4 et i= 6 on a une score de 77.92207792207793 %
Pour k = 4 et i= 7 on a une score de 77.92207792207793 %
Pour k = 4 et i= 8 on a une score de 77.27272727272727 %
Pour k = 4 et i= 9 on a une score de 78.57142857142857 %
Pour k = 5 et i= 1 on a une score de 80.51948051948052 %
Pour k = 5 et i= 2 on a une score de 77.92207792207793 %
Pour k = 5 et i= 3 on a une score de 77.92207792207793 %
Pour k = 5 et i= 4 on a une score de 77.27272727272727 %
Pour k = 5 et i= 5 on a une score de 78.57142857142857 %
Pour k = 5 et i= 6 on a une score de 78.57142857142857 %
Pour k = 5 et i= 7 on a une score de 78.57142857142857 %
Pour k = 5 et i= 8 on a une score de 79.22077922077922 %
Pour k = 5 et i= 9 on a une score de 79.22077922077922 %
Pour k = 6 et i= 1 on a une score de 77.92207792207793 %
Pour k = 6 et i= 2 on a une score de 76.62337662337663 %
Pour k = 6 et i= 3 on a une score de 76.62337662337663 %
Pour k = 6 et i= 4 on a une score de 77.27272727272727 %
Pour k = 6 et i= 5 on a une score de 80.51948051948052 %
Pour k = 6 et i= 6 on a une score de 81.81818181818183 %
Pour k = 6 et i= 7 on a une score de 81.16883116883116 %
Pour k = 6 et i= 8 on a une score de 83.11688311688312 %
Pour k = 6 et i= 9 on a une score de 81.81818181818183 %
```

Out[43]:
```
0.8311688311688312
```

In [44]:
```
modele=make_pipeline(StandardScaler(),PolynomialFeatures(8),KNeighborsClassifier(6)
modele.fit(x_train,y_train)
d=score=modele.score(x_test,y_test)
print('le modele donne \n ')
print('une efficacité de ',d*100,'% \n')
print('une erreur de',100-d*100,'%')
```

```
le modele donne

une efficacité de  83.11688311688312 %

une erreur de 16.883116883116884 %

============================================================
```

B) Algorithme: LogisticRegression

La régression logistique est un algorithme de classification très répandu, offrant un\ pouvoir explicatif très fort du fait de sa linéarité.

In [45]:
```
from sklearn.linear_model import LogisticRegression
```

In [46]:
```
logist=LogisticRegression()
logist.fit(x_train,y_train)
l=logist.score(x_test,y_test)
print('le modele donne \n ')
print('une efficacité de ',l*100,'% \n')
print('une erreur de',100-l*100,'%')
```

```
le modele donne

une efficacité de  80.51948051948052 %

une erreur de 19.480519480519476 %
```

In [47]:
```
    for i in range(1,10):
        mode=make_pipeline(StandardScaler(),PolynomialFeatures(i),LogisticRegressic
        mode.fit(x_train,y_train)
        score=mode.score(x_test,y_test)
        scor.append(mode.score(x_test,y_test))

        print('Pour i=',i,'on a une score de',score*100,'%')
```

```
Pour i= 1 on a une score de 80.51948051948052 %
Pour i= 2 on a une score de 79.87012987012987 %
Pour i= 3 on a une score de 76.62337662337663 %
Pour i= 4 on a une score de 78.57142857142857 %
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Pour i= 5 on a une score de 79.22077922077922 %
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Pour i= 6 on a une score de 79.22077922077922 %
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Pour i= 7 on a une score de 78.57142857142857 %
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Pour i= 8 on a une score de 79.87012987012987 %
Pour i= 9 on a une score de 75.32467532467533 %
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

==========

Conclusion :

Comme on a les meilleures scores avec l'algorithme de KNeighborsClassifier vis à vis de l'algorithme de logisticregression\ nous allons conserver la meilleure score de l'algorithme de KNeighborsClassifier qui se trouve ci-apres\ dans la suite de notre travail

In [48]:
```
modele=make_pipeline(StandardScaler(),PolynomialFeatures(8),KNeighborsClassifier(6)
modele.fit(x_train,y_train)
d=score=modele.score(x_test,y_test)
print('le modele donne \n ')
print('une efficacité de ',d*100,'% \n')
print('une erreur de',100-d*100,'%')
```

```
le modele donne

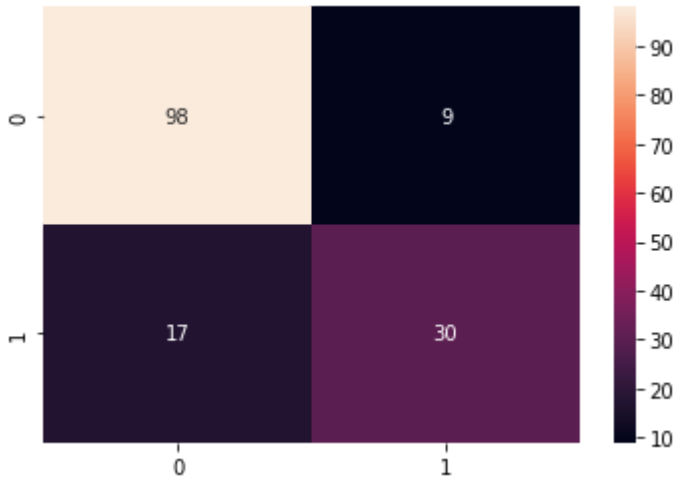
une efficacité de  83.11688311688312 %

une erreur de 16.883116883116884 %
```

In [49]:
```
y_pred=modele.predict(x_test)
```

In [50]:
```
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
```

Out[50]:
```
<AxesSubplot:>
```



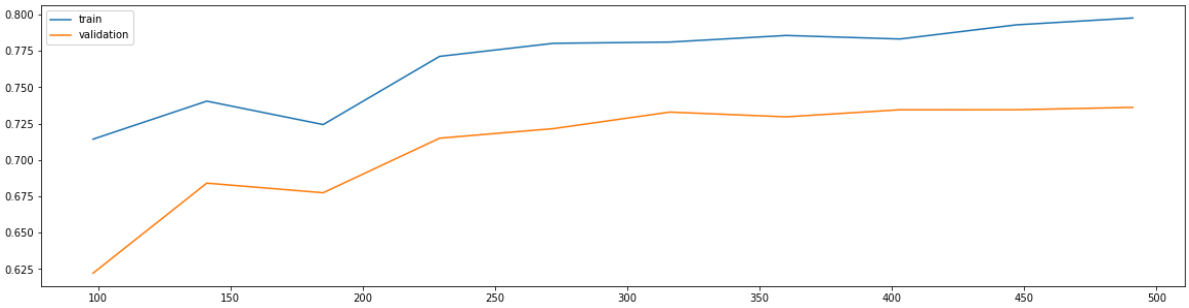
==========

Courbe d'apprentissage

In [51]:
```
from sklearn.model_selection import learning_curve
```

In [52]:
```
plt.figure(figsize=(20,5))
N,train_score,val_score=learning_curve(modele,x_train,y_train,train_sizes=np.linspa
print(N)
plt.plot(N,train_score.mean(axis=1),label='train')
plt.plot(N,val_score.mean(axis=1),label='validation')
plt.legend()
plt.show()
```

```
[ 98 141 185 229 272 316 360 403 447 491]
```



A partir de la courbe nous pouvons clairement voir qu'à mesure que la taille de l'ensembled'entraiment augmente, la courbe de score d'entraiment et la score de validation croisée converge. La précision de la validation croisée augmente a mesure que nous ajoutons plus de données d'entrainement est donc utile dans ce cas c'est-à-dire plus qu'on a des données plus nous pouvons avoir de meilleur score

In [ ]: