Présentation finale LU1SXARE-ARESD :

25/04/2024

-Zane Rayane.
-Ben Chabane Amel.
-Ait Ouahioune Sara.
-Tafat Melia.

Plan de la présentation :

PROBLÈME POSÉ

ANALYSE ET ÉTUDE

CONCLUSION

Problème posé :

Dans cet atelier de recherche, on s'est intéressé à l'émission du CO_2, générée par la production d'électricité qu'elle soit carbonée, décarbonée ou normale qui sont nos trois modalités. Tout cela pour prédire la qualité des émissions.

Présentation des données :

Entrée [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Entrée [5]:
```
data=pd.read_csv("benchmark.csv")
```

Entrée [6]:
```
data
```

Out[6]:

	DateTime	MixProdElec
0	2019-01-01 00:00:00	Normal
1	2019-01-01 00:30:00	Normal
2	2019-01-01 01:00:00	Normal
3	2019-01-01 01:30:00	Normal
4	2019-01-01 02:00:00	Normal
...	...	...
35035	2021-12-31 21:30:00	Normal
35036	2021-12-31 22:00:00	Normal
35037	2021-12-31 22:30:00	Normal
35038	2021-12-31 23:00:00	Normal
35039	2021-12-31 23:30:00	Normal

35040 rows × 2 columns

Le tableau Benchmark

Le tableau Train

Entrée [8]:
```
data2=pd.read_csv("train.csv")
data2
```

Out[8]:

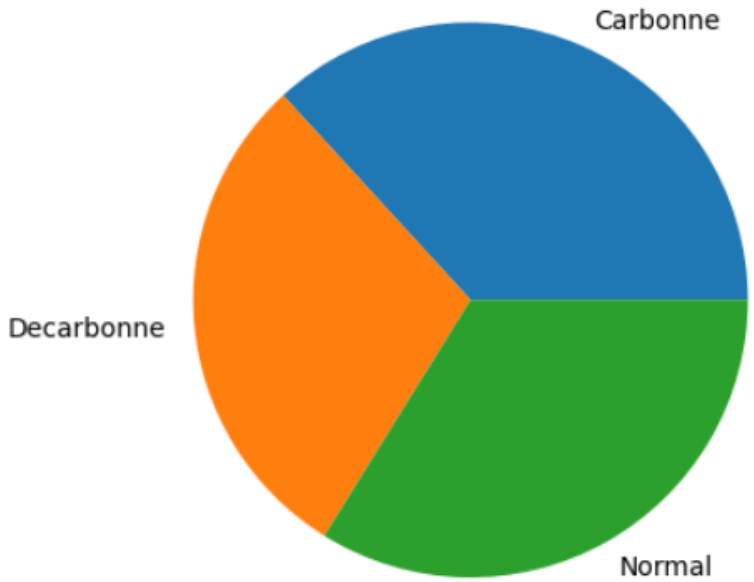
	DateTime	MixProdElec	EmissionCO2	PositionDansAnnee	Annee	Mois	DemiHeure	Jour	JourFerie	JourFerieType	VacancesZoneA	VacancesZon
0	2016-01-01 00:00:00	Normal	932.5575	0.000000	2016	janvier	0	vendredi	True	1janvier	True	Tr
1	2016-01-01 00:30:00	Normal	912.7485	0.000057	2016	janvier	1	vendredi	True	1janvier	True	Tr
2	2016-01-01 01:00:00	Normal	906.8480	0.000114	2016	janvier	2	vendredi	True	1janvier	True	Tr
3	2016-01-01 01:30:00	Normal	903.3600	0.000171	2016	janvier	3	vendredi	True	1janvier	True	Tr
4	2016-01-01 02:00:00	Normal	892.4800	0.000228	2016	janvier	4	vendredi	True	1janvier	True	Tr
...	...	...	...	...	...	...	...	...	...	...	...	
95980	2023-06-22 14:30:00	Decarbonne	652.9410	0.472915	2023	juin	29	jeudi	False	nonFerie	False	Fal
95981	2023-06-22 15:00:00	Decarbonne	646.9470	0.472972	2023	juin	30	jeudi	False	nonFerie	False	Fal
95982	2023-06-22 15:30:00	Decarbonne	661.9340	0.473029	2023	juin	31	jeudi	False	nonFerie	False	Fal
95983	2023-06-22 16:00:00	Decarbonne	701.6100	0.473086	2023	juin	32	jeudi	False	nonFerie	False	Fal
95984	2023-06-22 16:30:00	Decarbonne	707.0325	0.473143	2023	juin	33	jeudi	False	nonFerie	False	Fal

Deux types de données :

Qualitatives :

```
Entrée [22]: plt.pie(n1,labels=x)

Out[22]: ([<matplotlib.patches.Wedge at 0x1f9a4ab4f50>,
           <matplotlib.patches.Wedge at 0x1f9a4ab5ed0>,
           <matplotlib.patches.Wedge at 0x1f9a4ab7110>],
          [Text(0.4428833576697579, 1.0069033377137855, 'Carbonne'),
           Text(-1.0952110606581347, -0.10253161762151049, 'Decarbonne'),
           Text(0.5348092544596585, -0.9612382958165494, 'Normal')])
```

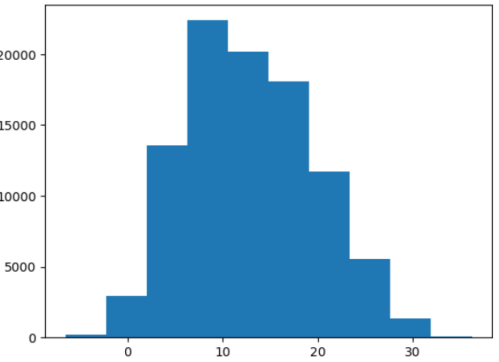


- Jour,
- Jour férié,
- Mois,
- Jour férié type,
- Vacances Zone A,B,C,
- MixProdElec.

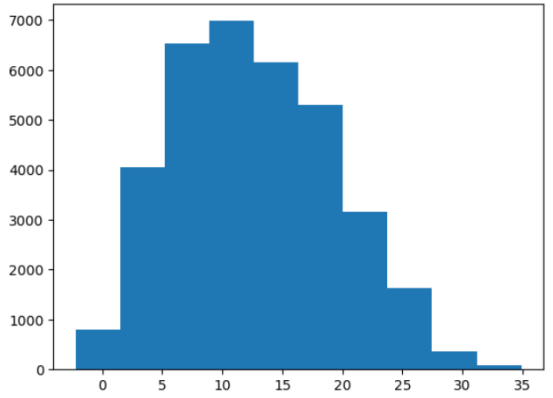
Quantitatives :

```
Entrée [10]: plt.hist(Tmp16,bins=10)

Out[10]: (array([  192.,  2920., 13543., 22387., 20154., 18057., 11736.,  5565.,
         1328.,   103.]),
array([-6.54615385, -2.26923077,  2.00769231,  6.28461538, 10.56153846,
       14.83846154, 19.11538462, 23.39230769, 27.66923077, 31.94615385,
       36.22307692]),
<BarContainer object of 10 artists>)
```

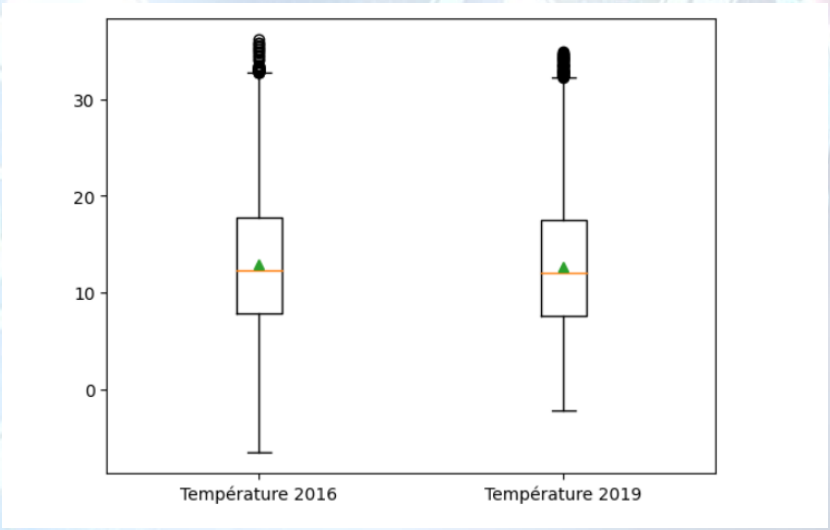



```
Out[13]: (array([ 801., 4051., 6536., 6980., 6158., 5294., 3156., 1624.,  358.,
         82.]),
array([-2.21081081,  1.49977027,  5.21035135,  8.92093243, 12.63151351,
       16.34209459, 20.05267568, 23.76325676, 27.47383784, 31.18441892,
       34.895    ]),
<BarContainer object of 10 artists>)
```



TEMPÉRATURES 2016

TEMPÉRATURES 2019



- Date et heure,
- Demi-heure,
- Emission CO_2,
- Année,
- Température,
- Humidité,
- Précipitation.

Plan de l'étude :

- Classifieur Naïf Bayes avec une variable.
- Classifieur Naïf Bayes avec deux variables.
- Classifieur Naïf Bayes avec plusieurs variables.
- Classification avec la méthode des K Plus Proches Voisins (KPPV ou KNN en anglais).
- Validation Croisée.

Let's Go!!

1) Classifieur Naïf Bayes avec une variable :

Entrée [9]:
```
MixProdElec=data2['MixProdElec'].to_numpy()
```

Entrée [10]:
```
MixProdElec
```

Out[10]:
```
array(['Normal', 'Normal', 'Normal', ..., 'Decarbonne', 'Decarbonne',
       'Decarbonne'], dtype=object)
```

Entrée [11]:
```
Jourférie=data2['JourFerie'].to_numpy()
```

Entrée [28]:
```
tab2=pd.crosstab(MixProdElec,Jourférie)
tab2
```

Out[28]:

col_0	False	True
row_0		
Carbonne	35008	324
Decarbonne	26249	1924
Normal	31800	680

Entrée [18]:
```
n_carbonne=np.sum(tab2.iloc[0])
n_carbonne
```

Out[18]:
```
35332
```

Entrée [19]:
```
n_decarbonne=np.sum(tab2.iloc[1])
n_decarbonne
```

Out[19]:
```
28173
```

Entrée [20]:
```
n_normal=np.sum(tab2.iloc[2])
n_normal
```

Out[20]:
```
32480
```

Entrée [21]:
```
n1=np.array([n_carbonne,n_decarbonne,n_normal])
n1
```

Out[21]:
```
array([35332, 28173, 32480], dtype=int64)
```

Entrée [23]:
```
py_decarbonne=n_decarbonne/n
py_decarbonne
```

Out[23]: 0.2935146116580716

Entrée [24]:
```
py_carbonne=n_carbonne/n
py_carbonne
```

Out[24]: 0.36809918216387977

Entrée [25]:
```
py_normal=n_normal/n
py_normal
```

Out[25]: 0.33838620617804865

Entrée [26]:
```
py_normal+py_carbonne+py_decarbonne
```

Out[26]: 1.0

Entrée [72]:
```
PXTrueYDecarbonne=len(data2[(data2['JourFerie']== True) & (data2['MixProdElec']=='Decarbonne')])/len(data2[data2['MixProdElec']==
PXTrueYDecarbonne
```

Out[72]: 0.06829233663436624

Entrée [73]:
```
PXFalseYDecarbonne=len(data2[(data2['JourFerie']== False) & (data2['MixProdElec']=='Decarbonne')])/len(data2[data2['MixProdElec'
PXFalseYDecarbonne
```

Out[73]: 0.9317076633656337

Entrée [74]:
```
PXTrueYNormal=len(data2[(data2['JourFerie']== True) & (data2['MixProdElec']=='Normal')])/len(data2[data2['MixProdElec']=='Normal
PXTrueYNormal
```

Out[74]: 0.020935960591133004

Entrée [76]:
```
PXFalseYNormal=len(data2[(data2['JourFerie']== False) & (data2['MixProdElec']=='Normal')])/len(data2[data2['MixProdElec']=='Norma
PXFalseYNormal
```

Out[76]: 0.979064039408867

Entrée [77]:
```
PXTrueYCarbonne=len(data2[(data2['JourFerie']== True) & (data2['MixProdElec']=='Carbonne')])/len(data2[data2['MixProdElec']=='Car
PXTrueYCarbonne
```

Out[77]: 0.009170157364428846

Entrée [79]:
```
PXFalseYCarbonne=len(data2[(data2['JourFerie']== False) & (data2['MixProdElec']=='Carbonne')])/len(data2[data2['MixProdElec']=='(
PXFalseYCarbonne
```

Out[79]: 0.9908298426355712

Entrée [ ]:
```
#P(Y=c|xj)=(p(xj|Y=c)*p(Y=c))/p(xj)
#Pour trouver le classsificateur de Bayes, on compare les P(Y=c|xj) de tous les Y=c|xj, d'après la formule on peut négliger le
#p(xj) car il suffit de comparer les numérateurs (ils sont égaux pour toutes les modalités considérées), de ce fait le
#classificateur prend la modalité de Y la plus probable suivant les xj.
```

Entrée [88]:
```
py_carbonne*PXTrueYCarbonne,py_decarbonne*PXTrueYDecarbonne,py_normal*PXTrueYNormal
```

Out[88]: (0.0033755274261603376, 0.0200447986664583, 0.007084440277126634)

Entrée [90]:
```
max(py_carbonne*PXTrueYCarbonne,py_decarbonne*PXTrueYDecarbonne,py_normal*PXTrueYNormal)
#Le classificateur de Bayes pour xj=True est: Decarbonne car il a la probabilité la plus élevée(le max correspond à la valeur
#de Decarbonne)
```

Out[90]: 0.0200447986664583

Entrée [91]:
```
py_carbonne*PXFalseYCarbonne,py_decarbonne*PXFalseYDecarbonne,py_normal*PXFalseYNormal
```

Out[91]: (0.36472365473771945, 0.27346981299161327, 0.33130176590092203)

Entrée [92]:
```
max(py_carbonne*PXFalseYCarbonne,py_decarbonne*PXFalseYDecarbonne,py_normal*PXFalseYNormal)
#Le classificateur de Bayes pour xj=False est: Carbonne car il a la probabilité la plus élevée(le max correspond à la valeur
#de Carbonne)
```

Out[92]: 0.36472365473771945

Score obtenu : 0.32258

Entrée [ ]:
```
#Conclusion:
#On constate qu'on peut déduire le classificateur de Bayes directement de la loi jointe pxy(X=x,Y=y) en prenant la valeur
#maximale pour chaque xj.
```

2) Classifieur Naïf Bayes avec 2 variables :

Entrée [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Entrée [2]:
```
data2=pd.read_csv("train.csv")
data2
data=pd.read_csv("benchmark.csv")
data3=pd.read_csv("prev.csv")
```

Entrée [3]:
```
data
```

Out[3]:

	DateTime	MixProdElec
0	2019-01-01 00:00:00	Normal
1	2019-01-01 00:30:00	Normal
2	2019-01-01 01:00:00	Normal
3	2019-01-01 01:30:00	Normal
4	2019-01-01 02:00:00	Normal
...	...	...
35035	2021-12-31 21:30:00	Normal
35036	2021-12-31 22:00:00	Normal
35037	2021-12-31 22:30:00	Normal
35038	2021-12-31 23:00:00	Normal
35039	2021-12-31 23:30:00	Normal

35040 rows × 2 columns

Entrée [4]:
```
data3
```

Entrée [4]:
```
data3
```

Out[4]:

	DateTime	PositionDansAnnee	Annee	Mois	DemiHeure	Jour	JourFerie	JourFerieType	VacancesZoneA	VacancesZoneB	VacancesZoneC	Temp
0	2019-01-01 00:00:00	0.000000	2019	janvier	0	mardi	True	1janvier	True	True	True	5
1	2019-01-01 00:30:00	0.000057	2019	janvier	1	mardi	True	1janvier	True	True	True	5
2	2019-01-01 01:00:00	0.000114	2019	janvier	2	mardi	True	1janvier	True	True	True	5
3	2019-01-01 01:30:00	0.000171	2019	janvier	3	mardi	True	1janvier	True	True	True	5
4	2019-01-01 02:00:00	0.000228	2019	janvier	4	mardi	True	1janvier	True	True	True	5
...	...	...	...	...	...	...	...	...	...	...	...	
35035	2021-12-31 21:30:00	0.999772	2021	décembre	43	vendredi	False	nonFerie	True	True	True	8
35036	2021-12-31 22:00:00	0.999829	2021	décembre	44	vendredi	False	nonFerie	True	True	True	8
35037	2021-12-31 22:30:00	0.999886	2021	décembre	45	vendredi	False	nonFerie	True	True	True	8
35038	2021-12-31 23:00:00	0.999943	2021	décembre	46	vendredi	False	nonFerie	True	True	True	8
35039	2021-12-31 23:30:00	1.000000	2021	décembre	47	vendredi	False	nonFerie	True	True	True	8

35040 rows × 16 columns

Entrée [5]:
```
loi_j=pd.crosstab(data2["MixProdElec"],data2["JourFerie"]) # création d'un tableau de contigence
n=len(data2["MixProdElec"])
a=np.array(loi_j/n) #loi jointe (MixProdElec,JourFerie)
a
```

Out[5]:
```
array([[0.36472365, 0.00337553],
       [0.27346981, 0.0200448 ],
       [0.33130177, 0.00708444]])
```

Entrée [6]:
```
loi_j=pd.crosstab(data2["MixProdElec"],data2["JourFerie"])
loi_j/n # loi jointe(MixProdElec,JourFerie)
```

Out[6]:

JourFerie	False	True
MixProdElec		
Carbonne	0.364724	0.003376
Decarbonne	0.273470	0.020045
Normal	0.331302	0.007084

Entrée [7]:
```
loi_j2=pd.crosstab(data2["MixProdElec"],data2["VacancesZoneC"])
b=np.array(loi_j2/n) # loi jointe(MixProdElec,VacancesZoneC)
b
```

Out[7]:
```
array([[0.27812679, 0.08997239],
       [0.18692504, 0.10658957],
       [0.21189769, 0.12648851]])
```

Entrée [20]:
```
loi_j2=pd.crosstab(data2["MixProdElec"],data2["VacancesZoneC"])
loi_j2/n # loi jointe(MixProdElec,VacancesZoneC)
```

Out[20]:

VacancesZoneC	False	True
MixProdElec		
Carbonne	0.278127	0.089972
Decarbonne	0.186925	0.106590
Normal	0.211898	0.126489

Entrée [9]:
```
p_carbonne=np.sum(loi_j.iloc[0])/n # p(carbonne)
p_carbonne
```

Out[9]:
```
0.36809918216387977
```

Entrée [10]:
```
p_decarbonne=np.sum(loi_j.iloc[1])/n # p(decarbonne)
p_decarbonne
```

Out[10]:
```
0.2935146116580716
```

Entrée [11]:
```
p_normal=np.sum(loi_j.iloc[2])/n # p(normal)
p_normal
```

Out[11]:
```
0.33838620617804865
```

Nos probabilités

Test d'indépendance :

Entrée [12]:
```
L1=[p_carbonne,p_decarbonne,p_normal] # création d'une liste des trois probabilités
L1
```

Out[12]:
```
[0.36809918216387977, 0.2935146116580716, 0.33838620617804865]
```

Entrée [32]:
```
pxy=np.array(pd.crosstab(data2["JourFerie"],data2["VacancesZoneC"]))/n # loi jointe (JourFerie,VacancesZoneC)
pxy

px=np.sum(pxy,axis=1)   # p(JourFerie)
px

py=np.sum(pxy,axis=0)    # p(VacancesZoneC)
py

def produit(x,y):        # p(JourFerie)*p(VacancesZoneC)
    xy=np.random.rand(len(x),len(y))
    for i in range(len(x)):
        for j in range(len(y)):
            xy[i,j]=x[i]*y[j]
    return xy
pxpy=produit(px,py)
pxpy

def indep(x,y):     # Test d'indépendance
    lignes,colonnes=(np.matrix(x)).shape
    for i in range(lignes):
        for j in range(colonnes):
              if abs(x[i,j]-y[i,j])>10**-2:
                    return False
    return True
indep(pxpy,pxy)
```

Out[32]:
```
True
```

Entrée [ ]:
```
# cela confirme que les deux variables JourFerie et VacancesZoneC sont indépendantes.
```

On applique la formule de Bayes :

Entrée [13]:
```
L=[]  # estimation de la probabilité par la formule de Bayes à deux variables
for i in range(3):
    L.append([])
    for j in range(2):
        for k in range(2):
            L[i].append((a[i][j]*b[i][k])/L1[i])
L
```

Out[13]:
```
[[0.2755763242068991,
  0.08914733053082036,
  0.0025504664374724436,
  0.0008250609886878941],
 [0.17415949258848987,
  0.09931032040312339,
  0.012765547782401406,
  0.007279250884056893],
 [0.2074614106114179,
  0.12384035528950414,
  0.004436281736344785,
  0.0026481585407818493]]
```

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Entrée [ ]:
```
# On conclue que (JourFerie==True et VacancesZonec==True) et (JourFerie==True et VacancesZoneC==False) correspond
# à la valeur Decarbonne,(JourFerie==False et VacancesZonec==True) correspond à Normal,(JourFerie==False et VacancesZonec==False)
# correspond à carbonne.
```

Entrée [14]:
```
jrfrie=np.array(data3['JourFerie'])
jrfrie
```

Out[14]:
```
array([ True,  True,  True, ..., False, False, False])
```

Entrée [15]:
```
vac=np.array(data3['VacancesZoneC'])
vac
```

Out[15]:
```
array([ True,  True,  True, ...,  True,  True,  True])
```

Entrée [22]:
```
pred=[]  # Prédiction de la variable MixProdElec
for i in range(len(jrfrie)):
    if (jrfrie[i]==True and vac[i]==True) or (jrfrie[i]==True and vac[i]==False) :
        pred.append('Decarbonne')
    elif (jrfrie[i]==False and vac[i]==True):
        pred.append('Normal')
    else :
        pred.append('Carbonne')
pred
```

Out[22]:
```
['Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
```

Entrée [23]:
```
data3['MixProdElec']=pred
data3
```

Out[23]:

	DateTime	PositionDansAnnee	Annee	Mois	DemiHeure	Jour	JourFerie	JourFerieType	VacancesZoneA	VacancesZoneB	VacancesZoneC	Temp
0	2019-01-01 00:00:00	0.000000	2019	janvier	0	mardi	True	1janvier	True	True	True	5
1	2019-01-01 00:30:00	0.000057	2019	janvier	1	mardi	True	1janvier	True	True	True	5

Entrée [24]:
```
soumission=data
soumission['MixProdElec']=pred
soumission
np.unique(soumission['MixProdElec'])
```

Out[24]:
```
array(['Carbonne', 'Decarbonne', 'Normal'], dtype=object)
```

Entrée [26]:
```
soumission.to_csv('predictions3.csv',index=False)
```

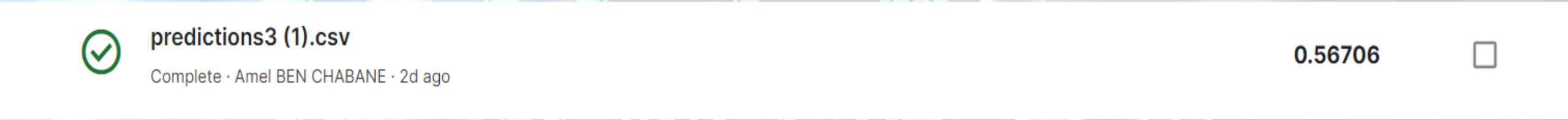
Score obtenu : 0.37480

On a battu Benchmark !!

Et c’est là qu’on s’est dit pourquoi ne pas essayer avec 3 variables ! On a essayé plusieurs combinaisons…

predictions3.csv Complete · Amel BEN CHABANE · 2d ago	0.44073	☐
predictions3.csv Complete · Melia TAFAT · 2d ago	0.37252	☐
predictions3.csv Complete · Melia TAFAT · 2d ago	0.50998	☐
predictions10.csv Complete · Rayane ZANE · 25d ago	0.43378	☐
predictions9.csv Complete · Rayane ZANE · 25d ago	0.44387	☐
predictions8.csv Complete · Rayane ZANE · 1mo ago	0.43302	☐
predictions7.csv Complete · Rayane ZANE · 1mo ago	0.43302	☐

Et la meilleure a donné ce score :



Qui est notre meilleur score, on n'a pas pu faire mieux (on vous spoile la fin c'est pas cool on sait ☹)

3) Classifieur Naïf Bayes avec plusieurs variables (trois) :

Entrée [7]:
```
np.unique(data3["DemiHeure"].iloc[:2544])
```

Out[7]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47],
      dtype=int64)
```

Entrée [30]:
```
loi_j=pd.crosstab(data2["MixProdElec"],data2["Jour"])
n=len(data2["MixProdElec"])
a=np.array(loi_j/n)      # Loi jointe(MixProdElec,Jour)
a
```

Out[30]:
```
array([[0.02821274, 0.06390582, 0.05265406, 0.06248893, 0.06373913,
        0.03673491, 0.0603636 ],
       [0.07595979, 0.02869198, 0.04030838, 0.02767099, 0.02673334,
        0.06277022, 0.0313799 ],
       [0.03884982, 0.05027869, 0.05005991, 0.05236235, 0.05254988,
        0.0435068 , 0.05077877]])
```

data -> Benchmark
data2 -> Train
data3 -> Prev

Entrée [31]:
```
loi_j/n
```

Out[31]:

Jour	dimanche	jeudi	lundi	mardi	mercredi	samedi	vendredi
MixProdElec							
Carbonne	0.028213	0.063906	0.052654	0.062489	0.063739	0.036735	0.060364
Decarbonne	0.075960	0.028692	0.040308	0.027671	0.026733	0.062770	0.031380
Normal	0.038850	0.050279	0.050060	0.052362	0.052550	0.043507	0.050779

Entrée [ ]:
```
# Prédiction de MixProdElec à travers la variable Jour:
# A l'oeil nu on a conclu que :
# Carbonne correspond aux jours de:Lundi, Mardi,Mercredi, Jeudi,Vendredi.
#Decarbonne correspond aux jours de: Samedi, Dimanche
#Normal ne correspond à aucune journée
```

Entrée [32]:
```
loi_j2=pd.crosstab(data2["MixProdElec"],data2["Mois"])
b=np.array(loi_j2/n)   # Loi jointe(MixProdElec,Mois)
b
```

Out[32]:
```
array([[0.00742824, 0.01370006, 0.05074751, 0.05420639, 0.05624837,
        0.01459603, 0.00338595, 0.00232328, 0.04758035, 0.0550086 ,
        0.04007918, 0.02279523],
       [0.0321196 , 0.04602803, 0.01139761, 0.00705319, 0.01022035,
        0.02384748, 0.04813252, 0.06489556, 0.01559619, 0.00477158,
        0.01174142, 0.0177111 ],
       [0.03796427, 0.03028598, 0.01536698, 0.02375371, 0.02654581,
        0.0390686 , 0.0343387 , 0.0257957 , 0.029838  , 0.01523155,
        0.02569151, 0.03450539]])
```

Entrée [34]:
```
loi_j2/n
```

Out[34]:

Mois	août	avril	décembre	février	janvier	juillet	juin	mai	mars	novembre	octobre	septembre
MixProdElec												
Carbonne	0.007428	0.013700	0.050748	0.054206	0.056248	0.014596	0.003386	0.002323	0.047580	0.055009	0.040079	0.022795
Decarbonne	0.032120	0.046028	0.011398	0.007053	0.010220	0.023847	0.048133	0.064896	0.015596	0.004772	0.011741	0.017711
Normal	0.037964	0.030286	0.015367	0.023754	0.026546	0.039069	0.034339	0.025796	0.029838	0.015232	0.025692	0.034505

Entrée [41]:
```
# Prédiction de MixProdElec à travers la variable Mois:
# A l'oeil nu on a conclu que :
# Carbonne correspond aux mois de:Décembre,Février,Janvier,Mars,Novembre,Octobre
#Decarbonne correspond aux mois de:Avril,Juin,Mai
#Normal correspond aux mois de : Août ,Juillet,Septembre
```

Entrée [36]:
```
loi_j3=pd.crosstab(data2["MixProdElec"],data2["DemiHeure"])
loi3=loi_j3/n    # Loi jointe(MixProdElec,DemiHeure)
loi3
```

Out[36]:

DemiHeure	0	1	2	3	4	5	6	7	8	9	...	38	39	40	41	
MixProdElec																
Carbonne	0.007907	0.006949	0.005793	0.005772	0.005574	0.005553	0.005084	0.004845	0.004667	0.004740	...	0.009293	0.009387	0.009262	0.009012	0.
Decarbonne	0.005282	0.006355	0.007532	0.007501	0.007730	0.007730	0.008387	0.008480	0.008699	0.008678	...	0.004928	0.004813	0.004813	0.005011	0.
Normal	0.007637	0.007532	0.007512	0.007564	0.007532	0.007553	0.007366	0.007512	0.007470	0.007418	...	0.006605	0.006626	0.006751	0.006803	0.

3 rows × 48 columns

Entrée [46]:
```
pr=[]   #Prédiction de MixProdElec à travers la variable DemiHeure :
for i in range (48):
    mod=0 # la modalité qui correspond à la DemiHeure i
    maxx=loi3[i][0] # la probabilité de la modalité majoritaire
    for j in range(3):
        if(loi3[i][j]>maxx):
            maxx=loi3[i][j]
            mod=j
    if mod==0: # ajouter la modalité correspondante à la prédiction
        pr.append('Carbonne')
    elif mod==1:
        pr.append('Decarbonne')
    else:
        pr.append('Normal')
pr
```

Out[46]:
```
['Carbonne',
 'Normal',
 'Decarbonne',
 'Normal',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
```

Entrée [22]:
```
jr=np.array(data3['Jour'])
jr
```

Out[22]:
```
array(['mardi', 'mardi', 'mardi', ..., 'vendredi', 'vendredi', 'vendredi'],
      dtype=object)
```

Entrée [23]:
```
mois=np.array(data3['Mois'])
mois
```

Out[23]:
```
array(['janvier', 'janvier', 'janvier', ..., 'décembre', 'décembre',
       'décembre'], dtype=object)
```

Entrée [24]:
```
DH=np.array(data3['DemiHeure'])
DH
```

Out[24]:
```
array([ 0,  1,  2, ..., 45, 46, 47], dtype=int64)
```

Entrée [43]:
```
# Liste représentant les demiheures qui correspondent à la modalité Carbonne
DHC=[i for i in range(14,48)]
DHC.append(0)
```

Entrée [47]:
```
pred=[] # Prédiction de MixProdElec à travers les trois variables :
for i in range(len(jr)):
    if ((mois[i]=='décembre' or mois[i]=='février' or mois[i]=='janvier' or mois[i]=='mars' or mois[i]=='novembre' or mois[i]=='
        pred.append('Carbonne')
    elif ((mois[i]=='août'  or mois[i]=='juillet' or mois[i]=='septembre')& (DH[i]==1 or DH[i]==13 or DH[i]==3)):
        pred.append('Normal')
    else :
        pred.append('Decarbonne')
pred
```

Out[47]:
```
['Carbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
 'Decarbonne',
```

Entrée [48]:
```
data3['MixProdElec']=pred # insertion de la colonne MixProdElec au tableau prev
data3
```

Out[48]:

	DateTime	PositionDansAnnee	Annee	Mois	DemiHeure	Jour	JourFerie	JourFerieType	VacancesZoneA	VacancesZoneB	VacancesZoneC	Temp
0	2019-01-01 00:00:00	0.000000	2019	janvier	0	mardi	True	1janvier	True	True	True	5
1	2019-01-01 00:30:00	0.000057	2019	janvier	1	mardi	True	1janvier	True	True	True	5
2	2019-01-01 01:00:00	0.000114	2019	janvier	2	mardi	True	1janvier	True	True	True	5
3	2019-01-01 01:30:00	0.000171	2019	janvier	3	mardi	True	1janvier	True	True	True	5
4	2019-01-01 02:00:00	0.000228	2019	janvier	4	mardi	True	1janvier	True	True	True	5
...	...	...	...	...	...	...	...	...	...	...	...	
35035	2021-12-31 21:30:00	0.999772	2021	décembre	43	vendredi	False	nonFerie	True	True	True	8
35036	2021-12-31 22:00:00	0.999829	2021	décembre	44	vendredi	False	nonFerie	True	True	True	8
35037	2021-12-31 22:30:00	0.999886	2021	décembre	45	vendredi	False	nonFerie	True	True	True	8
35038	2021-12-31 23:00:00	0.999943	2021	décembre	46	vendredi	False	nonFerie	True	True	True	8
35039	2021-12-31 23:30:00	1.000000	2021	décembre	47	vendredi	False	nonFerie	True	True	True	8

35040 rows × 17 columns

```
Entrée [49]: soumission=data
             soumission['MixProdElec']=pred
             soumission
             np.unique(soumission['MixProdElec'])

Out[49]: array(['Carbonne', 'Decarbonne', 'Normal'], dtype=object)

Entrée [50]: soumission.to_csv('predictions3.csv',index=False)
```

Score obtenu : 0.56706

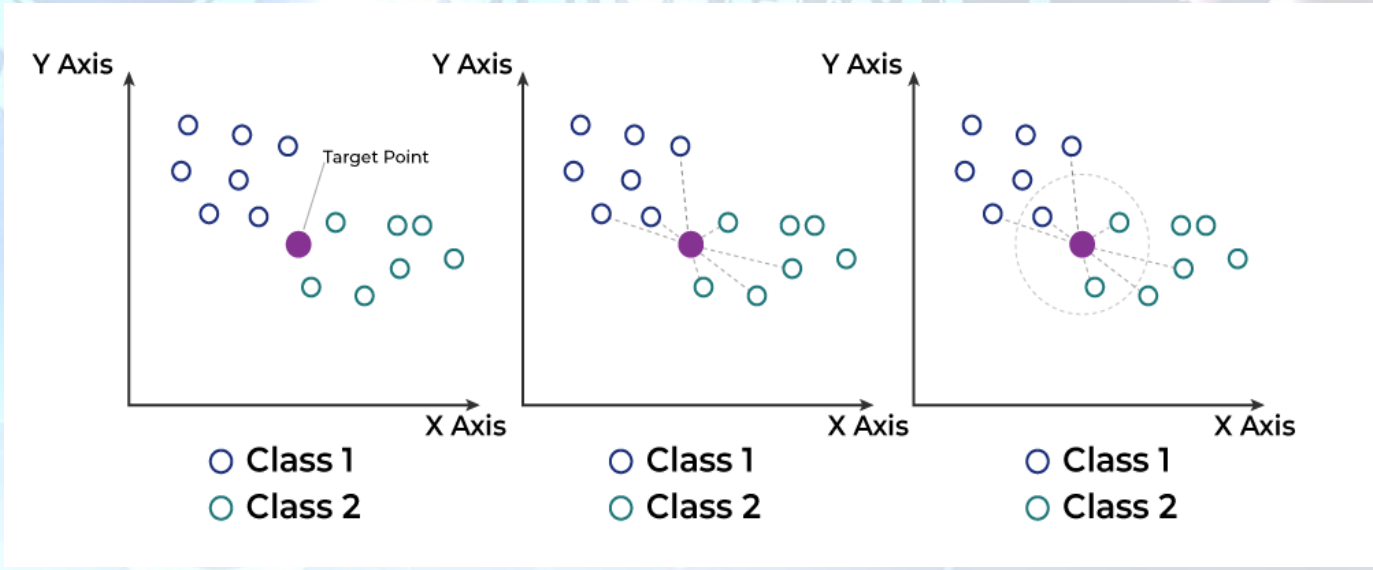
Conclusion :

La méthode du classifieur de Bayes est efficace pour la prédiction, surtout avec 3 variables. Elle permet d'obtenir des résultats fiables avec des probabilités d'erreurs minimes.

4) Classification avec la méthode des K Plus Proches Voisins :

Principe :

- On sélectionne une valeur de K(nombre de voisins),
- On calcule la distance,
- On trouve les points les plus proches,
- On vote pour la classification(c'est la classe majoritaire qui l'emporte).



Notre code :

Entrée [94]:
```
def classe_majoritaire(a): # Fonction pour déterminer la classe majoritaire
    valeurs, nb_fois = np.unique(a, return_counts=True) # Calcul des valeurs uniques et de leur fréquence
    index_max_fois = np.argmax(nb_fois)  # Obtient l'index de la fréquence la plus élevée
    return valeurs[index_max_fois]
    # Retourne la valeur correspondant à l'index de la fréquence la plus élevée, qui représente la classe majoritaire
classe_majoritaire(base_classe) # teste de la fonction
```

Out[94]:
```
'Carbonne'
```

Entrée [95]:
```
def classifie_kppv(k,mdesc, mclass, x): # Fonction pour classifier avec la méthode des k plus proches voisins (kNN)
    dist = np.zeros((len(mdesc)))
    for i in range(len(mdesc)):
        dist[i] = (x[0] - mdesc[i][0])**2 + (x[1] - mdesc[i][1])**2 + (x[2] - mdesc[i][2])**2 + (x[3] - mdesc[i][3])**2
        # Calcul des distances euclidiennes
    tridist = np.argsort(dist)
     # Tri des distances
    classe = [mclass[tridist[i]] for i in range(k)]
    return classe_majoritaire(classe) # Retourne la classe majoritaire parmi les k plus proches voisins
```

Entrée [96]:
```
classifie_kppv(1,base_data,base_classe,np.array(data2.iloc[5962][['DemiHeure','Temperature','Nebulosity','Humidity']]))
# teste de la fonction
```

Out[96]:
```
'Normal'
```

Entrée [*]:
```
pred=[]
for i in range(len(data2)):
    pred.append(classifie_kppv(1,base_data,base_classe,np.array(data2.iloc[i][['DemiHeure','Temperature','Nebulosity','Humidity']
    # prédiction de la variable MixProdElec sur les données prev avec l'algorithme des KNN
pred
```

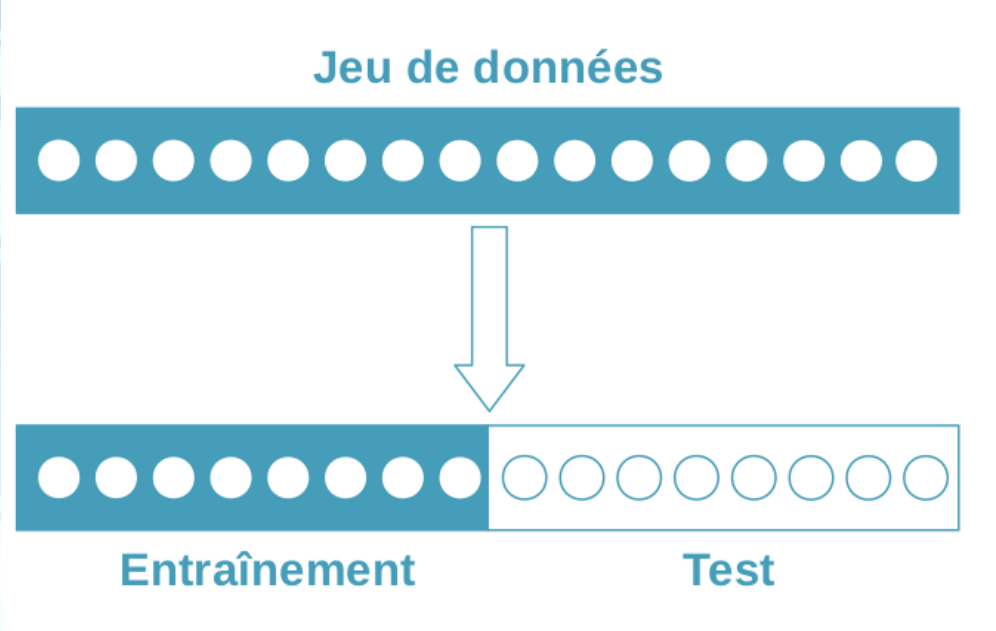
Conclusion :

La méthode des KNN n'est pas la méthode la plus apte à fournir des résultats optimaux et ceci est dû au problème de la dimensionnalité : l'algorithme a du mal à classer correctement lorsque la dimension est trop élevée…

Score obtenu : 0.44073

5) Validation croisée :

La validation croisée est une méthode qui permet d'évaluer les performances des modèles d'apprentissage(choisir le K optimal).
On divise notre jeu de données en deux : Données d'apprentissage et données de test.



Notre code :

Entrée [10]:
```
def taux_reussite(k):
    pred_curr=[]
    taux=[]
    for j in range(0,1000,200): # une boucle avec pas=200 pour parcourir les données d'entrainement
        moyenne=0
        for i in range(j,j+200): # une boucle pour faire la prédiction des 200 données avec l'algorithme des KNN
            pred_curr.append(classifie_kppv(k,base_data_curr,base_classe,np.array(data.iloc[i][['DemiHeure','Temperature','
            if pred_curr[i]==entrainement[i]: # si cette prédiction est la méme que la donnée on incrémente la moyenne
                moyenne=moyenne+1
        taux.append(moyenne/200) # on ajoute le taux de réussite des 200 données au tableau taux
    return taux
k_1=taux_reussite(1)
k_3=taux_reussite(3)
k_5=taux_reussite(5)
k_7=taux_reussite(7)
k_9=taux_reussite(9)
```

Entrée [11]:
```
k_1,np.mean(k_1)
```
Out[11]: ([1.0, 1.0, 1.0, 1.0, 1.0], 1.0)

Score obtenu : 0,44948

Entrée [12]:
```
k_3,np.mean(k_3)
```
Out[12]: ([0.675, 0.875, 0.8, 0.99, 0.985], 0.865)

Score obtenu : 0,43871

Entrée [13]:
```
k_5,np.mean(k_5)
```
Out[13]: ([0.715, 0.89, 0.79, 0.98, 0.995], 0.874)

Entrée [14]:
```
k_7,np.mean(k_7)
```
Out[14]: ([0.66, 0.85, 0.725, 0.97, 1.0], 0.841)

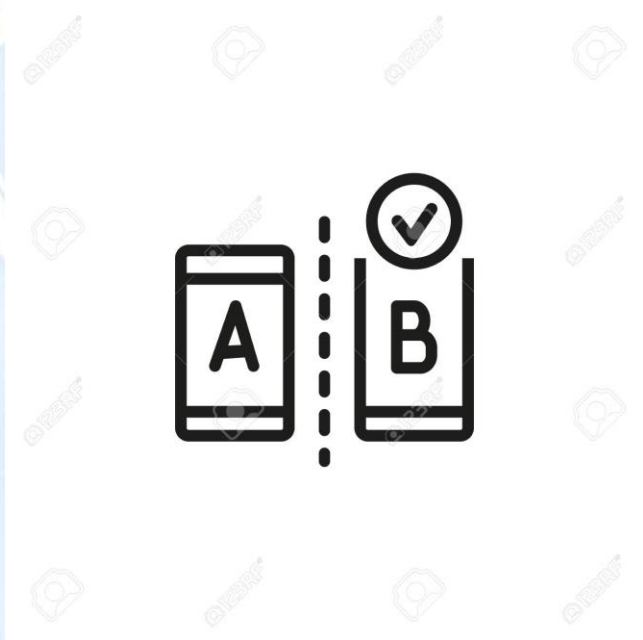
Score obtenu : 0,39783

Entrée [15]:
```
k_9,np.mean(k_9)            # on remarque que le k optimal est 1.
```
Out[15]: ([0.625, 0.835, 0.705, 0.965, 0.995], 0.825)

Entrée [ ]:

Conclusion :

La validation croisée fournit des résultats meilleurs que ceux de KNN, elle permet d'avoir plus de précision mais malgré ça on obtient un score inférieur à celui obtenu avec le classifieur de Bayes…



Conclusion de l'étude :

Pour conclure notre étude,

D'après nos résultats (la meilleure prédiction), la qualité « bas carbone » du mix de production d'électricité est la plupart du temps Carbonée.

Il est donc nécessaire de trouver des solutions environnementales et qui font appel à l'énergie renouvelable.

Merci pour votre attention !